SmartP: Revisiting the Parking Experience

BY

DAVIDE LEONARDUZZI
Laurea, Politecnico di Milano, Milano, Italy, 2010

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2013

Chicago, Illinois

Defense Committee:

Robert V. Kenyon, Chair and Advisor
Tom Moher, University of Illinois at Chicago
Marco D. Santambrogio, Politecnico di Milano

To my parents, who are the best anyone could hope for and are always there for me, being supportive and constructive. I can barely express my gratitude for everything they have done for me.

And to Prof. Marco D. Santambrogio, who is more than just a Professor to me: he is a mentor, and a friend.

Davide Leonarduzzi

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

DEFINITIONS AND ACRONYMS

Definitions and acronyms Here is a list of words and acronyms used in this thesis, whose definitions are useful to understand the document itself.

API application programming interface, it describes the communications between the components of the system

DBMS database management system

Fare inspector the person responsible of ensuring that cars are legally parked

Final user the driver who pays for the parking fare

GPS global position system

GUI graphical user interface

GWT Google Web Toolkit

JDO Java Data Objects

JSON JavaScript Object Notation

Management anyone who works in the back office of a parking company or in the back office of the parking fine procedure

Mock-up a simplified preview of the elements constituting a Graphical User Interface

MVP Minimum Viable Product, a product which has all of the features required for it to be used but which lacks any extra features

noSQL a generic name to describe all databases which are not SQL compatible

OS operating system

PCI Payment Card Industry Data Security Standard

Prepaid parking session a parking session where the user pays in advance for the whole amount of time s/he expects to be staying

DEFINITIONS AND ACRONYMS (Continued)

Software Requirements Specifications according to IEEE standard 830-1998 this document describes the requirements of the system formally, in order to make sure that the final result meets the high level objectives which were set

Start-stop parking session a parking session where the driver does not have to define its length at arrival time but simply stops it when leaving

UML Unified Modelling Language

Use case description of how a user would interact with the system in order to access a specific feature

SUMMARY

The initial objective of smartP, which is going to be presented in this work, is to offer a complete solution to allow drivers to pay for on-street parking using their mobile phones. In order to reach this goal a series of steps have been undertaken, and they are going to be described in this document.

The use of parking fares has a positive impact on reducing traffic and pollution in the city: setting the right price for parking so as to keeping the filling rate of parking spaces at around 80% would have significant benefits in terms of traffic and pollution, but also in terms of car turnover and therefore potential customers for local businesses.

For this reason in most cities there are already different solutions in place to collect the parking fares and check that there are no infringements, but all of these solutions have their own peculiar disadvantages, which are the initial motivations for this work. The worst aspect of existing solutions is that they require more effort on the driver's side than it would be required, because the payment procedures are not as straightforward as they could be.

The objective was to build a solution which could take advantage of the technologies available in smartphones to allow drivers to pay the parking fare as easily as possible, while increasing the overall quality of the service and respecting any requirements the parking companies might have.

The proposed solution is composed of a series of components: a final user application, an inspector application, a management portal, a smartP back-end and a parking company back-end. The reason for decoupling the two back-ends will be later explained in detail, but this solves one of the most annoying problems for current users: it allows for different companies to have their own back-end while users only have one application and one account. Moreover the decoupling allows other companies to offer fare payment, by interacting with the company back-end, thus competing with smartP at the final user level.

This work is an enabling technology for a series of future works, global position system (GPS) such as the addition of new payment methods, the guidance of drivers to free parking spots and even parking reservation. Moreover benefits for the parking company and city council will be introduced in the future thanks to this proposed solution as an enabling factor: additional information and insight into the typical parking usage will be provided, and a dynamic and more elaborate pricing policy will be possile, in order to reach the goal of an 80% filling rate of parking spaces.

A general overview is in Chapter 1. First of all the reason for the payment of a fare to park cars is explained. Moreover the main disadvantages of current payment approaches are presented, and they mainly are about the system not being as easy to use as possible.

Chapter 2 is about the existing solutions, both the ones which are actually in use in at least one city and those which have only been developed at a theoretical level, for future implementation. Each alternative is presented in terms of how it works, its pros and cons. To begin with, scratch cards are the first option, which is peculiar because of the total lack of any hardware investment.

Then hardware based solutions are presented, and they are divided into two parts: park-meters and custom hardware to be given to the driver. Different types of park-meters exist, but they usually involve high initial investments and high maintenance costs, even though they are usually more practical than scratch-cards for drivers. Custom hardware solutions are usually devices which are to be placed under the windscreen of the car and to be used as proof of payment for parking; this solution is particularly convenient for drivers who frequently pay for parking, because each of them needs to pay for the device first, and then it is usually required to fill the account with a fixed minimum amount of money before it can be used.

Finally the last type of payment method is based on mobile phones and it further divides into two categories. SMS texts can be used to communicate the intention of paying the parking fare; this solution is cumbersome and still requires the user to initially fill the account, usually through a computer. The second alternative is for smartphones which support applications. The use of an application to pay for parking offers drivers many advantages, and the biggest problem with this approach is the number of adoption barriers, such as the need of a smartphone, a connection to the Internet and usually a credit card.

Chapter 3 presents the Software Requirements to build a system which would make the adoption barriers of an application based payment solution lower than the existing ones, which is why the proposed solution only directly compares to phone based solutions. Here the stakeholders are described, and more importantly their needs and their interactions with the system. Starting from this, a series of required features is presented, so that the proposed solution can be built in such a way which would respect them. Some of the broader requirements are the need for scalability and for a fully managed solution for smartP, which would not require system architects to manage.

In Chapter 4 the proposed solution is presented. Each of the components constituting the framework of the proposed solution is described individually, but essentially the goal is to build a solution which satisfies all the requirements. For the back-ends Google App Engine was chosen, using Java and the Java EE framework. This decision was based on the requirements and the flexibility offered by a solution which could be easily ported from the cloud to other servers. In terms of final user application a series of solutions are presented to reduce the user input required (e.g. the plate number can be read by taking a picture instead of typing).

Finally conclusions are presented in Chapter 6. Essentially in the first part a comparison between the requirements and the proposed solution is presented, showing how the second one matches the first one and how adoption barriers are reduced copared to other existing phone based solutions.

In this chapter future works are presented as well. After some general future works (such as offering the final user application on as many platforms as possible) some specific ones are considered. The first one is about payment methods: they are an adoption barrier and therefore there should be as many alternatives as possible, so that if one of the methods is not an option for a specific user, s/he will opt for a different one.

The second one is about helping drivers identify available parking spaces, and here different possible approaches are considered, even if it is out of the scope of this document to determine which one would be the best one. The third is closely connected to the second one, because it is about allowing final users to book a parking space before they arrive. The obvious connection is that in order for either of these to work there has to be a way to determine where the available spots are and, since the investment required to build the two systems independently would be higher than building one which supports both features, they can be considered together; but, again, this is a future work and not part of the proposed solution.

Finally the focus is put on the city council or park owner because, even if the current solution already offers benefits to the parking company or city council, a series of additional features can be added to provide them with additional information and insight into the typical parking usage, in fact from the beginning additional data is collected constantly about people parking around the city and paying for the fare with the smartP final user application. This information can be used to infer statistics about the status of parking spaces and over time the accuracy will increase, and the addition of other features to the final user application will provide even more information to be used in the management portal. Additionally all of this information could be used in the future to determine a dynamic and more elaborate pricing policy, in order to reach the goal of an 80% filling rate of parking spaces.

CHAPTER 1

CONTEXT DESCRIPTION

In most cities the number of people travelling by car is rapidly increasing, which causes both serious traffic congestion in the streets and fewer available parking spaces. Both problems are constantly being dealt with, different solutions have been suggested and the optimal approach has not been definitely found. Moreover, different success stories prove different approaches successful in different cities, where people's mentality and habits are in alignment with the chosen option.

In this work the focus will be on the parking spaces, even though a study presented in Section 2.4 shows how people looking for a parking space can actually make a significant percentage of traffic and therefore while accounting for the parking problem a positive side effect is the traffic reduction. Since the effects of actions towards the solution of either of these problems are reciprocal, the two problems can and should be considered together by an administration, but the aim of this research is only focused on parking.

1.1 Problems involving parking spaces

Nowadays, more and more drivers[1] want to park their cars in cities and especially in city centers, and this is not a recent trend. Different solutions have been implemented over the years to address this new need, such as building new parking lots or multi-storey car parks, and charging for on-street parking. The first example addresses the problem by increasing the available parking spaces and is generally the most convenient option for drivers. On the other hand, this has the implication of allowing more people to park and therefore to drive to the place they are going to, when they could use public transportation. This has a negative impact on car traffic and city pollution but it is sometimes required due to the increase in the number of people who own cars.

The second example solves the problem by forcing drivers to park their cars in the city for as little as possible, because they would not pay more than they need to. Moreover this approach does not require any additional area for cars to park, because it solves the problem with the existing car spaces. On the other hand this solution has political implications, both because people do not want to pay for something they used to have for free, and also because a price increase always has a stronger impact on utilization than the price change significance.

In most cities both solutions have to be applied, due to the fact that drastic solutions are required and either of them without the other would not be enough for drivers to be able to accomplish their tasks with the minimum discomfort possible. One of the mechanisms to increase the number of available off-street parking spaces is through parking-ratio requirements, which consists in regulations stating how many car spaces should be built when new buildings are constructed, according to a series of considerations which include usage of the building. In this chapter on-street parking will be considered, and therefore only a price increase can have a significant impact because there are not many changes which can take place in order to increase the number of on-street parking spaces and in fact usually changes involve fewer on-street parking spaces (often compensated by more off-street parking lots).

So the main subject of this chapter is paid on-street parking, with its inconveniences both for drivers on the one side and for managers and administration on the other side. Even though the reasoning behind the decision to set a fare for the use of the parking spaces is to reduce traffic congestion, and more specifically make sure that people spend less time looking for a parking space, parking revenues now have a significant impact on the overall revenue of some cities[1]. For this reason the city is not only a stakeholder who wants car traffic reduced, but it is often in the best interest of the city to maximize profits on their paring assets.

[1]As an example, income for parking in Milan in 2009 was 7,348,500.06 and the total income of the city was 3,569,908,936.93[2]. Income for parking in Edinburgh in Fiscal Year 2011-12 was 15,288,801.63[3] and the total income of the city was 993.1 million[4].

In Section 1.2 and Section 1.3 the main inconveniences for both drivers and the management (park owners, management company etc.) related to on-street parking will be presented.

1.2 Inconveniences for park managers

The first inconvenience for a city council, where the decision is made to ask drivers to pay a fare for on-street parking, is to choose a system, architecture or solution to accomplish a series of steps. As a first thing, all drivers should have an easy way of paying for the parking session, in the sense that the adoption barrier should be minimal for all the population. For this reason the driver cannot be forced to make any initial investment, e.g. in custom hardware. Moreover the driver should not be forced to use any form of payment other than cash, even though alternatives could be offered and also appreciated, because a public service should be accessible for the entire population. Furthermore illegal parking should be detectable and it should be possible for the city council (potentially by delegation) to fine any car parked for free in a parking space which requires a fare.

Under these constraints a series of viable solutions will be introduced in Chapter 2, some of which having general coverage of the population, others being more specific in terms of potential users but solving some of the inconveniences present in the most common approaches.

These disadvantages vary based on the specific solution being adopted, but generally involve a substantial initial investment in order for the system to start. These costs might include infrastructure investments in the case of park-meter based solutions, while set-up costs are present with other solutions as well, for example scratch-card solutions still require signs to be placed near parking spaces to inform drivers about the rules for parking.

Another problem for the city council is that they either manage everything internally or they have a private company manage a part or the entirety of the process instead (in alignment with the local law, some steps might not be performed by private companies in some cities and/or Countries, such as the detection of cars being parked illegally). Whatever they manage externally usually contributes to lower earnings for the city council, while whatever they manage internally accounts for more effort and responsibility on the council side. Some of the tasks and costs of normal maintenance are money collection (which consists in bringing all money being paid locally to the council), ticket inspection, fine generation and delivery, maintenance of any custom hardware, printing of tickets and/or scratch cards. Each of these steps require people devoted to the tasks and with different expertises. For example printing scratch cards might require a tender in certain cities for the assignment to an external company, and therefore will involve the legal department as well as the management. For the ticket inspection and fine generation local Police might have to be involved, or other people might be able to execute the same tasks, based on the local law, but would still have to be instructed on the laws and procedures for parking and related fines.

This non-exhaustive list excludes all maintenance costs a city council would undertake if parking spaces were free as well.

It is true that on-street pay parking is an opportunity for the city to increase income, but most cities now depend on it as a significant contributor (as mentioned earlier) and therefore it is important for them to minimize the costs of park management, or in other words to maximize profits on parking fares while not increasing the cost of the fare for final users, or at least while keeping it as low as possible. This is a particularly important point, in fact increasing the price is an option to be adopted rarely because it is not well accepted by citizen, who are also the people who will vote for the next city council. For this reason price increases have to be justified and are rare, and therefore the focus is in the reduction of management (operational) costs.

It is important to note that all problems presented until now matter to the city in terms of money, in the sense that they either cause lower income or more costs, but either way there are no problems for the city council which cannot be fixed if the economic loss can be withstood. This point is important because it shows what the city council is looking for as a stakeholder, assuming that they operate in the best interest of their citizens.

1.3 Inconveniences for users

Here we consider inconveniences for drivers who use the services introduced by city councils or park managements in order to pay for the fare required to park their car on an on-street parking space. To begin with, it is important to point out that the user is often forced to accept whatever is expected of him/her, because there are not always valid alternatives both in terms of pricing and in terms of locations and park owners. Basically the driver will try and park in the free spaces, which are usually the first to fill up, unless s/he values his/her time more than the money required to park the car close to where s/he wants to park.

For this reason the driver is in a condition of inferiority compared to the parking space owner. There is a part of people who decide not to pay the fare, and this has to do with a series of factors, not limited to user willingness to risk, the effort by the city council to validate tickets, cultural peculiarities. The problem of the driver who receives a fine because s/he does not pay the parking fare is not considered to be an inconvenience, because s/he is willingly assuming the risks for it. On the other hand a better procedure for the fine payment would make the experience better for those drivers as well, and potentially decrease the percentage of unpaid fines.

Rare cases where people were fined while paying at the park-meter happen, but are not statistically significant and usually can be solved afterwards in court to the driver's favour. On the other hand the more common scenario is that the driver does not pay because s/he cannot do that. In fact, based on the solution adopted by the park owner, situations arise when the driver cannot pay. As an example when using scratch cards the driver has to find a reseller open at the time and day of need, unless s/he bought the cards in advance. For park meters the drivers might not have coins with them and the park-meter might not accept any form of payment other than coins, in which case they would have to find someone who would change their bills for coins. In these situations the driver might risk a fine but the reason why s/he is not paying is not because s/he does not want to, but it is because the barriers are too high for the risk of getting a fine or being late.

After addressing fines, which are particular situations and should involve a very limited number of people compared to the total population using the parking spaces, the inconveniences for drivers who use the service properly and pay regularly will be introduced. These inconveniences will be better described based on the adopted solution in Chapter 2, but a general overview will be presented here.

To begin with, the same problems already outlined for the causes of people receiving fines unfairly is also an inconvenience for people who do pay for parking: it is as inconvenient for them as it is for anyone else, with the only difference that they plan in advance to be prepared for the situation. The situation is unbalanced for the reasons explained earlier, but the result is that drivers have to comply with the rules even though they could be made simpler and more convenient for them while preserving the profits for park owners.

In addition to these issues another one is present where these traditional solutions are adopted: drivers have to know in advance how long they will be staying, or they have to go back to their car to feed the park-meter or renew the parking session in any permitted way. This turns out to be another occasion for fine generation, because drivers might not be able to leave whatever they are doing and go back to pay again, if their activity is taking longer than they originally planned. This, again, forces them to either risk a fine and not pay or risk paying more than they are going to use (in order to make sure that they do not risk ending up in the situation just described) or they should be able to go back to their cars.

Some of these issues are solved by some of the presented solutions. It is worth noting that there is a tendency to offer custom alternative solutions which solve most of these problems for final users, even though these solutions are usually built for drivers who park often, because of entry barriers such as initial investment or prepaid approach. What, on the other hand, is often missing is interoperability between different cities, and custom solutions which work in a city rarely work in others. Drivers who softer use pay parking spaces would also want a solution which works in all cities they travel to, but this custom solution approach does not allow them to accomplish this. Anyway these solutions are much more practical for people who do use it compared to traditional solutions.

In Chapter 2 a list of existing solutions will be presented in more detail, providing a description of advantages and flaws of each solution.

1.4 Main motivations for this work

The first observation is that requiring drivers to pay for parking in cities can have two positive effects on the city itself: higher availability of empty car spots, and at the same time a new revenue stream for the city council. This second point becomes extremely relevant for some cities, therefore it is of the utmost importance that profits are maximized without having to increase the fare value (which would be against the drivers' benefit).

That said, the main motivations for this work are to be found on the driver's side: the currently offered solutions are not as convenient for them as they could be. Certain solutions require drivers to have coins with them (e.g. the park-meter ones), others force drivers to look for ticket resellers (e.g. scratch cards). Either way the driver has to put an unnecessary effort in order to have the right to park his/her car.

Another example of disadvantages of some of the current solutions is the need to plan the stay beforehand, in more ways. One of them is about having to decide the length of the stay at arrival time, and to pay in advance. This has a series of effects, such as people having to leave in situations where they would have stayed longer, and people staying where they are even after their parking session expire, thus risking a fine.

Finally a solution which would allow the city council to reduce operational costs would have a positive impact in all those cities where parking fare revenue makes up for a significant part of the revenue of the entire city.

CHAPTER 2

STATE OF THE ART

Different solutions have been thought for on-street parking, some of which are currently in use in various cities around the World. This chapter is an overview of those solutions, ordered based on the type of mechanism used for the payment, which is the only focus of this chapter. It is important to point out that while some of the solutions presented here are actually implemented in many cities, all of them are implemented in at least one city. Off-street parking, such as multi-storey parking, is not considered because the use of an application would not significantly improve the experience for drivers, if it did at all.

2.1 Paper based solutions

Scratch cards are the simplest solution to be implemented, because they heavily rely on people and working structures and minimize the role of the system itself. These cards are printed (usually by the park owner or City council) and distributed to a network of businesses authorized to sell them, such as newsagents. Any person who wants to park in an area where parking space has to be paid using scratch cards will have to find a business where these cards are sold and buy them, or buy them in advance. Either way, when the person is ready to park the car s/he has to scratch it according to instructions in order to show all relevant information (e.g. date and time of arrival). Assuming that the user has to stay longer than the time allowed by the card s/he scratched, s/he has to scratch a new one setting the arrival time equal to the expiration time of the previous card, and repeat this process until the entire parking session is covered. All cards have to be displayed inside the car, under the windscreen. An example is in Jersey[5], where parking can be paid using scratch-cards.

This first rudimentary solution is good because it requires very little intervention for it to work, and in fact, apart from signs to inform drivers that they need a card to park, there is no need to set up any custom hardware to manage the service. It is also very good in terms of barriers for people to start using this service: any owner of a car can use this service as there are no requirements on his/her side.

On the other hand this solution is also the one with the highest amount of disadvantages, again, for its simplicity. First of all cards have to be printed, then they have to be distributed to all businesses with a permission to sell them; finally the owner of the business will get a percentage of the sale of the card. These three actions are both expensive as opposed to solutions with park-meters (e.g. the cost of printing 20000 cards could be around 1300GBP[6], which of course would be lower in cities where a large amount of cards could be printed at the same time). It also has another huge disadvantage for park owners, and this is very little knowledge of and control over what is going on. In fact whatever the agreement between park owner and individual businesses, the only data coming back to the owner is how many tickets are requested and how much money they generate, but at a large grain because businesses only have to report this information when they are out of scratch cards or in specific times of the year. Moreover the park owner will not get any insights as to where the scratch cards are used, assuming that they own parking spaces in multiple areas, let alone having a live report on the status of their parking spaces. Finally it takes longer to check scratch-cards than to check park-meter tickets, and there is a risk of fraud[7].

For drivers the disadvantages are various as well. To begin with, this method requires them to buy the scratch cards before starting the parking session. This could be a reasonable task for people who often pay for parking in a particular city and buy an amount of cards which allows them to park several times before having to buy new ones. People who use the service for the first time or seldom will find it very inconvenient and potentially difficult to buy cards from an authorized business, both because of the waste of time and the difficulty of finding an open business close-by[7]. Whether a driver already had a card or just bought one, from then on s/he faces the same disadvantages. First of all scratching the card takes time, especially if more than one is required to cover the entire time of the parking session. Moreover the user has to know in advance how long s/he will be staying and scratch the right number of cards accordingly. Finally the user can only choose a multiple of the value of the cheapest card s/he has, because all it says is that it is valid for the specified time, starting when the user decides.

In conclusion this is a good solution for park owners, who can reduce their involvement in the process to the minimum, at the cost of lower margins as opposed to other available alternatives. On the other hand this is a very inconvenient solution for drivers, being this the solution which creates most problems to them.

2.2 Hardware based solutions

Hardware based solutions require investments in custom hardware but are usually much more convenient than the paper based solutions. Very different options are available, and the first distinction is between public park-meters and park-meters inside the driver's cars.

Park-meter

In terms of public park meters an evolution took place over the years, but it is worth presenting all stages because each of them is still in use in at least one city.

The first park-meter was fully mechanical[8] and it was associated with a single car space. This later evolved to a newer version of it, which is still associated with just one parking space, but it is not fully mechanical and this solution is still widely adopted. Since the evolution faces some of the same advantages and disadvantages of the original version, pros and cons will be presented together for both alternatives.

First of all the advantage is that again there are very few requirements for any driver to be able to use this service, which makes this a good solution to make sure everyone can park. Moreover, the driver will find the park-meter right next to the parking space, thus making it extremely easy to find and convenient location-wise. The disadvantages, on the other hand, are various. First of all, drivers have to pay using coins, which they might not have and might need to find a way to change banknotes on their own. Secondly, not all coins are always supported, usually only some values are accepted. Finally, for users it is very inconvenient if they have to go back to their car every time the parking session expires in order to renew it, in the case that the park-meter does not allow the user to insert an amount of money according to the length of the parking session, but it requires the user to insert one or more (but a fixed number) coins of a given value, thus forcing the session to last exactly a specific amount of time (this is, of course, under the assumption that it is legal to feed the park-meter). Even if this is not the case, the user still has to know in advance how long s/he is going to stay in the parking space.

For car park owners there are advantages and disadvantages as well. The potential advantage is that the park owner could have an idea of how many and which paring spaces are free at any given time, assuming park-meters communicate in a network. On the other hand this approach requires a lot of management effort, because money has to be collected periodically from each and every park-meter, and the number equals the total number of parking spaces. The initial cost for a system like this to work is significant, because the park owner has to pay for the park-meters and the installation, which is time consuming, again, because of the high number of park-meters compared to the number of car spaces. Finally, another disadvantage of this solution is vandalism: having all of these park-meters around the city means being subject to many kind of damages, which require the park owner to pay for unplanned maintenance. Finally, this approach limits the way parking spaces are used: since there is a park-meter for each space, it has to be clearly delimited in all sides. Other solutions would allow to only surround the parking area, so that drivers can park as close to each other as they can, further optimizing space usage and therefore profits for the park owner.

A newer version of the park-meter is a more complex machine, which can be used to pay for an entire area of parking spaces. Each of these park-meters is more expensive than the older version, but at the same time fewer of them are required, so it might make economical sense.When a park-meter is installed which covers an entire area, two main approaches are available; the first one entails each parking space being numbered, so that drivers park their car and pay using the park-meter for the parking space where they parked, by typing the appropriate parking space number. The other solution entails printing a ticket which the driver should display inside the car, under the windscreen. The numbered approach is for example available in certain areas of Los Angeles[9], while the unnumbered solution is available for example in New York City[10].

The first solution has the great potential advantage of having a clear idea of the car spaces being used at any given time. On the other hand it requires additional effort on the driver's part, who has to check the space number and type it in the park-meter before proceeding with the payment. The second solution is easier for the driver (because the payment process involves fewer tasks and is therefore quicker) and most importantly it allows drivers to make the best use of space possible, but at the same time it gives park owners worse insight into what parking spaces are available at any given time. If car parks are numbered the driver can potentially extend the parking session from any park-meter in the city by either remembering the parking space number or by reading a code printed on the initial receipt[11]. Apart from these differences, park-meters for a group of parking spaces have the same advantages and disadvantages, so they can be considered together.

To begin with, for drivers it is difficult at times to locate the park-meter closest to the place they parked their car in. Moreover, based on the payment options being offered, the payment itself might be inconvenient. This is actually a potential advantage of this new solution as opposed to the park-meter for an individual car space: given the requirement of fewer park-meters, a higher investment can be made, offering a more sophisticated payment procedure, potentially including various payment methods. In particular, coins of different values are accepted most of the times, while other optional payment methods might include credit cards, prepaid parking cards and banknotes. As far as cash payments are concerned, even the option for change is potentially available, even though this is not always the case. Credit cards are a very practical tool for people to pay but a small percentage of people do have a credit card (even though this substantially varies on the country) plus certain people do not trust automatic machines with their credit cards. Prepaid parking cards, on the other hand, are a solution as good as the credit card, while lowering transaction costs for park owners, but this is a custom solution which requires drivers to pay in advance for more than they would pay in a single parking session, in the form of prepaid parking credit[7]. This custom solution works in a city or in all parking spaces owned by the same company, but it is usually incompatible with solutions implemented in other cities, which is a problem people really care about.

In[12] they introduce a mechanism to accept many payment methods at park-meters for multiple parking spaces, and more specifically: VISA Wave and MasterCard PayPass (both contact-less credit cards), and wireless phone billing; they also introduce a main park-meter for an area. The architecture has one main park-meter for each area, capable of communicating both with the surrounding park-meters and the back-end parking management system. The advantage of this approach is that park-meters can be individual (for each parking space) and cheaper than the main one, while offering the same functionality to the final user (and saving energy, thus requiring fewer battery changes).

Apart from the differences pointed out previously, all of the disadvantages of the park-meter for a single car space can be found in park-meters for multiple car spaces, such as vandalism, cash collection, initial investment.

Knowing what car spaces are being used has been presented as a potential advantage without detailing the idea. Basically this allows the company to know more about its customers/users, which allows better insights and more control over the status of the company. As an example, this allows for a better price tuning, based on space usage statistics[13].

A solution has been presented[14] to notify the user about the expiration of the parking session through his/her phone after s/he paid using a park-meter. This solution interacts with the phone only to remind the user to either leave the parking space or, if legal, to pay for a new parking session. Either way the user has to proceed by physically coming back to the car before the expiration of the session.

Smart ePark[15] offers a mechanism to automatically detect cars which are being parked without a valid payment session, which allows for automatic remote fine generation by use of cameras and sensors.

Custom hardware in the user's car

Another solution for users to pay for on-street parking is by adopting custom hardware to be put inside each user's car. This solution is obviously only viable for people who frequently pay for parking and, since these solutions are custom-made for a specific park owner, the ideal user is a driver who parks often in the same city. Because of all of these restrictions, this solution can only be offered as an alternative option to more traditional solutions, like the ones presented before this one. In particular this mechanism works best in conjunction with scratch cards or park-meters which print a ticket, because just like all these other solutions, it can be displayed under the wind shield and therefore it can be easily checked by inspectors. Examples of this approach are EasyPark[16] and the Comet by Ganis Systems[17].

This hardware requires a relatively small investment, but at the same time the investment can only be repaid by the user who uses it, and therefore it is hardly profitable. For this reason a person who is interested in this service is often required to pay for the hardware which s/he then puts in his/her car.

The mechanism works as follows: the device installed in the car can deduct money from prepaid cards, according to the settings the driver picks (e.g. the area where the car is being parked, and therefore the price per hour). The advantages are various: the driver does not have to look for a way to pay, because s/he already has it in his/her car. Moreover, this mechanism allows for the payment of the sole duration of the session, by starting it when parking the car and stopping it when leaving the car space. This idea is introduced by[18] where they also introduce a way to check parking fare payments automatically. The disadvantages are present as well: first of all, this device does not have any remote connection and can only be refilled by using prepaid cards, so the user does have to buy them and insert them in the device to use the credit. Both disadvantages are involved with this: users have to pay in advance for parking an amount which is much larger than the cost of a single parking session and they still have to look for a place where they can actually purchase a prepaid card, because it is necessary for the system to work that a physical card is used (as it is the only communication mechanism for the device).

Another disadvantage for final users might be to forget to stop paying when they leave: the device does not notify the user, it just keeps charging the user as if s/he were still parked, until s/he stops the parking session. Some devices account for periods of time where parking is free (e.g. nights and weekends) but they start charging the user again when these periods are over.

Assuming that the adoption of this solution is not widespread enough to justify attempts at hacking it and assuming that maintenance costs are reasonable for the park owner, this is a very good solution for park owners who collect most of the money at select locations and do not need to maintain devices around the city. In other words this solution puts most of the investment costs in the final users; actually the company will still have to buy a large number of devices before reselling them so the initial investment has to be made anyway, but it is limited and then it only takes a limited amount of time for park owners to get all money back, as long as they make a good estimate on how many devices they will sell over any period of time. There is also a cost for prepaid cards, but it is low compared to the value any prepaid card is worth in terms of money to pay for parking.

A more evolved solution which would still be part of this category is presented in [19], where they present a fully automated mechanism to charge the user for parking and check whether drivers are paying when they should. Basically the device installed in the driver's car is capable of connecting to a remote server (via an ad-hoc network) and communicating a precise car location using a GPS. This patent also includes the validation part, for inspectors to check which cars do not have a valid parking session.

A similar solution, again involving custom hardware on the car and an automatic mechanism for parking payment is presented in , and this also includes a solution based on cameras to read the car plate of cars being parked, thus allowing a more accurate remote generation of automatic fines, when appropriate[20]. This solution provides a stack of layers which have different responsibilities for the overall system to work. This allows payments either through the user's credit card or through e-park credits.

2.3 Mobile phone based solutions

There are a series of solutions which make use of phones in different ways in order to offer a payment method for parking. There are no known real-world applications where these phone-based solutions make a significant part of the profits for a specific parking vendor, and this is clearly due to the presence of adoption barriers. Particular focus will be put into these barriers, trying to point out if they are necessary or just present because they are more convenient for those who built the system or for the car park owner.

Different solutions will be presented in the same chapter, but they all have the same feature of making use of phones to work, which in itself presents advantages and disadvantages. For this reason these properties are presented here, and then the following subsections will focus on the differences between the general approach and the specific solutions.

To begin with, the use of phones is a good idea in terms of investment, because it relies on drivers paying for the hardware (apart from any server-side hardware). Moreover, this scales really well, because careless of how many parking spaces are supported by the service and how many people use it, the software investment is almost constant (apart from minor customization and set-up costs), so from an investment perspective this solution is a very good one for both the company and the drivers (as they do not invest in a parking solution, they already invested in a phone for other reasons, so they just make a better use of their investment).

On the other hand, it is noteworthy that it might be necessary for parking space owners to invest in hardware to validate parking: these technologies do not offer any physical receipt to be displayed under the wind shield, therefore if car spaces are not numbered the car plate (or some sort of code on a sticker attached to the car windscreen) is the only code which an inspector can rely on in order to validate the session. This technology is not required for traditional parking solutions and therefore it requires an additional investment. If, on the other hand, parking spaces are numbered and the park-meter stores data about which parking spaces are free and which ones are in a valid parking session, then the inspector already has hardware dedicated to the validation task and therefore all that is required is to integrate the data from the new system of payment through phones directly into the original hardware. The same stands for park-meters for single car parks, as long as these devices have any communication channel to the company (otherwise there would be no way of notifying the park-meter about the payment through a different channel or payment method).

The key potential advantage is of course the opportunity to offer an easier to use and more practical service, because people who have phones carry them around most of the time when out of their homes. Then, of course, based on the adopted mechanism, this potential advantage can be used.

First of all a general overview of the various alternatives will be presented. It is important to point out that most observations about commercially used solutions are either about user experience or the high level mechanism and procedures to pay for parking, and the reason is that information about how these software solutions work is not publicly available. When, on the other hand, academic research is presented, additional information is usually available and therefore it will be presented.

It is also important to point out that the market is very fragmented, so an effort will be made to present the most different solutions being adopted, but the discussion will focus only on some of the many existing ones.

2.3.1 SMS texts or phone call

The first solution to take advantage of phones chronologically makes use of SMS technology. This first approach has the advantage that hardware requirements are the minimal to offer a payment service through a phone, since SMS technology is very widely adopted and supported in almost all mobile phones. For this reason adoption barriers on the user's hardware are very low.

The procedure to use this service is very complicated and cumbersome. To begin with, the driver has to create an account, which usually takes place online but it is not required for the system to work. This in itself increases the adoption barrier limiting it to people who have a computer with internet connection only, thus making the bar high enough for people who do not park often enough to pay for parking using this service. Moreover, this approach requires drivers to prepay for parking, and usually there is a fixed amount of money as a minimum for refilling the account. This again is another reduction in the total number of users who can ultimately use the service if the account refill has to take place online, through a credit card transaction. Especially in certain countries the percentage of people who have a credit card is significantly lower than the number of people who own a phone.

A good solution to avoid these issues would be to allow for charging directly on the phone bill (or as phone traffic if prepaid): this would not put the limitations of owning a credit card nor of having to pay beforehand for a fixed amount of money independently of how many times the parking service will be used by the driver. The reason why this approach has not yet been successful is that phone companies ask a transaction fee which does not make the parking business interested, as it is not the only option for them to charge users and it would significantly decrease their earnings.

Once the user has a registered account with enough money, the service can be used from any phone which supports the SMS technology: the driver has to send a text to a phone number of the parking company, respecting a particular syntax which may vary based on the data required to park. Assuming that the company associates the driver's phone number to his/her account, for each account there is only one car plate, and parking spaces of this company are not numbered, two values are required: the area where the driver is parking and the duration of the paring session. As far as the first one is considered, the parking spaces owner could write an area code on the signs describing the parking rules. This area code could even correspond to the amount of money being charged per hour. For the second value an interesting approach is possible. In this case the user has already paid for future parking sessions, so the user can either say for how long s/he will be parking or just that s/he just parked, and then stop the session by sending a new text. This second option is actually one of the biggest advantages of the text based approach, because drivers do not have to know how long they are going to park for. Even the method expecting the session duration is better than traditional approaches with scratch-cards or park-meters, because it can be renewed before expiration without having the user to go back to the car. This also allows for the company to easily notify the user about the session expiring soon, as introduced for park-meter based sessions in[14].

The text solution is a first attempt at using the phone as a new method to pay for parking, but it has a series of disadvantages which make it a very rarely used solution by drivers even in cities where it is offered. In addition to the flaws presented beforehand, it is not practical to use this mechanism because of the syntax of the text messages, which has to be respected in order for the system to understand the request. This either means that the user has to follow the instructions on how to prepare the text each time, or s/he has to memorize the rules. Moreover texts are expensive for both park owners and most drivers, because the phone company charges them at normal rate based on the user's tariff or company agreements. This is not something to be taken lightly, because for a parking session of 30 minutes sending a text would cost the user up to 50% of the total parking costs, just in text costs. Moreover the company would have to pay to confirm the parking session with a text, and potentially even send a second text as a reminder for the parking session expiring, thus reducing earnings for the park owner. Costs become even higher for users who decide to start a parking session without setting up a stop time in the first text: this means that both the user and company will have to pay a total of two texts each, just for one parking session. Another disadvantage in terms of user experience is that it is not easy for people to write text on phones, especially if they have to write codes (as opposed to words, which might be quicker to type for some users by enabling T9 dictionary). Finally this system is very limiting both in terms of options (for example having multiple cars associated with an account or paying for another person's car) and in terms of features: this service can only be used to pay and it would not make sense to extend it any other way.

The alternative to the text mechanism is by making phone calls. The two approaches are very similar in terms of both advantages and disadvantages. The phone call mechanism is based on an automated answering machine which asks for and collects the same data which the user would otherwise send via text. An example of this approach is presented in [21]. The differences in terms of advantages and disadvantages are mainly the fact that for the user there is no need to remember the syntax to write the SMS text because the answering machine asks for the required data explicitly. On the other hand this procedure might be more expensive for the final user and more importantly take more time than writing a single SMS text and send it to the company phone number.

2.3.2 Smartphone application

This is by far the best approach in terms of practicality for final users, because it takes advantage of the hardware of the phone to make the user experience as intuitive and simple as possible. There are many different solutions taking advantage of smartphones with different benefits and disadvantages, so first of all a general overview will be presented, with additional details about differentiations.

The first thing to point out is that application based solutions have very high adoption bars in terms of initial requirements, in other words few people are potential users. It is important to note that this is going to change, because all of the requirements are expected to grow in adoption by the general public over the next years. In order to run the application a compatible smartphone must be used. Most solutions offer applications for Android and iOS smartphones, while others support other platforms as well. Supporting a wide variety of platforms is important for both supporting the highest percentage of smartphone users as possible and for publicly owned parking companies which cannot favour some platforms over the others. Of course the reason why some companies only support Android and iOS is that this covers the highest percentage of smartphone users (as opposed to any other combination of two platforms out of the existing ones) and maintaining the application on multiple platforms might be considerably more expensive than only maintaining two.

The second element which is required is that the user needs to be connected to the internet to use the service. Of course technically any smartphone can accomplish this (assuming that it can operate in the country it is being used, but it is a good assumption to think so), therefore the problem is not technical but economic: if the user has a data plan, then s/he will likely be able to use the service for free, assuming that the data plan is usually much bigger in terms of data transfer allowance than the very small data requirements of the application. The alternative would be to pay for the KB of transmitted data. Of course the second approach is as inconvenient in terms of expenses for the final user as the SMS text solution and therefore most users without a data plan should not be considered as part of the potential user base. This reduces the number of people who would use the service compared to the the total number of smartphone owners who park cars on street.

Finally the last adoption barrier is the payment method. Different solutions are available in various cities, each having advantages and disadvantages. All companies which offer the application as an alternative to sending SMS texts (that is, they offer a better graphical interface to access the exact same service) make their users face the same problems related to paying via SMS, so users have to own an internet connected computer, use it to create the account initially and pay to refill the account. Even if this limitation can be avoided by allowing the user to refill the account directly using the smartphone application, a payment method has to be provided and the choice determines costs and practicality of the overall service for any user. The most common approach is to charge the driver's credit card, even though it is not the only option. Again, the use of a credit card as the sole payment mechanism is an adoption barrier, as in many countries credit card owners are less than the owners of smartphones, but on the other hand this solution is much more practical than any other option for those who do have a credit card. Other possible payment mechanisms are prepaid cards, which have the advantage of not requiring the use of a credit card but have disadvantages similar to scratch cards, and therefore they are not practical.

Different applications tackle the problem of the payment differently, but the general idea is the same. To begin with, unnumbered parking spaces will be considered. Most applications require an account, then they require a car plate to associate it with, but potentially multiple car plates can be supported by a single account. Moreover they require information about the location where the user is parking (usually this coincides with an area, described on signs around the area where they are parking). Finally they require payment information, which is related to the previously presented point. An example of an application to pay for unnumbered parking spaces is[22].

The main criticism to the vast majority of the existing applications is that they simply offer a better user interface for a service which would otherwise work over SMS texts, while in fact the hardware potential could be used to offer drivers a better user experience and not just a better user interface which does not require the user to remember the syntax of SMS texts (which, of course, is an advantage itself). The other aspect is that some applications require the driver to tell the system the length of the parking session in advance. With the use of an application this can only be a political choice, because the technology to allow for the session to be started and stopped manually is available. Not allowing sessions of dynamic length also means that companies can ask a premium to renew the parking session remotely before it expires. For unnumbered parking spaces gamification can be used to ask users to communicate the presence of free spaces, but it was tested at a University of Queensland[23] and, even if accuracy was high, the population is not representative of average population in a regular city center, both in terms of smartphone adoption and in terms of

The other alternative is to number the parking spaces, which means that the user does not need to communicate the car plate number, but just the number of the place where s/he is parking. For example this approach is used in Salt Lake City with QP QuickPay[24]. The disadvantage of this approach is that the user is required to type more than the car plate, in the sense that every time s/he parks s/he has to type the parking space number. It is also important that car spaces are numbered, thus spaces are not used as efficiently as with unnumbered spaces, for parallel parking, but it does not make a difference for perpendicular nor angle parking. On the other hand there are several advantages: first of all communicating the park space instead of the car plate allows the company to know what car spaces are being used and which ones are empty at any given time. This is useful for the park owner to gain a better understanding of how the various parking spaces are being used and to have better control over the current status of the traffic in the city. Most importantly, this information could potentially be used to help people find available car spaces, and even to book spaces in advance.

A solution to deliver data about available car spaces can be more easily achieved in off-street parking. For example a proposed solution[25] is to install sensors (magnetometers) at every entrance and exit of an off-street parking lot, then send information about the space count to a cloud service which then informs users through an application (built using PhoneGap to be installable on the highest amount of devices while limiting development time).

2.4 Pricing policies

The proposed solution will allow a deeper level of detail on the choice of the pricing models for on street parking, just like all solutions which make use of electronics and not paper as a proof of purchase (e.g. texts, smartphone applications, potentially park-meters for numbered parking spaces). This might seem irrelevant, as in most cities the current approach is to decide a hourly price (optionally differing by area) and ask drivers to pay for it during the day, while leaving car spaces free during the night and in the weekends. The reason why this is the most common approach is that it used to be the only one and few cities changed but, since the solution being presented with this thesis allows for a more dynamic pricing approach, it is worth analysing some data about parking space pricing and its effects.

There is a strong link between the price and the percentage of free parking spaces: if the price is high enough less people will park in those locations, thus decreasing the percentage of used spaces. Moreover people will find a parking space more easily, and in particular the time spent now in business districts to find a free parking space averages from 3.5 to 14 minutes, and 8-74% of the traffic is generated by people looking for a free parking space[26].

Several studies show that a 80-90% occupancy rate is optimal, and likely to leave a free parking space on each block. A study on Seattle[26] has been conducted to determine whether to increase or decrease pricing based on occupancy rate in the past. There are advantages for everyone: people decide whether they want to park or not, based on the location and price. This means that they will not stay longer than they should thus freeing parking spaces earlier. Businesses in the area are advantaged as well, because the percentage of occupied parking spaces is high (as usual in busy areas, potentially increased in less busy areas) but turnover drastically increases. This is important for businesses because after a user has purchased a product (or service) the parking space s/he occupies will not generate any more profits for this business until it is freed and occupied by another car.

In conclusion offering pricing variability is important for the city and goes to the benefit of all drivers, businesses and citizens (less pollution due to parking space search), and different tariffs by day, time and location will have this positive impact. In order to determine the tariffs

TABLE I: COMPARISON OF PROS AND CONS OF THE PRESENTED SOLUTIONS

	Initial investment	Operational costs	User initial investment	Ease of setup	Ease of use	Adoption barriers
Paper based	+	- -	++	++	- -	++
Park-meter	- -	-	++	++	-	++
Custom Hardware	-	+	- -	- -	++	- -
SMS text	-	++	+	- -	+	-
Smartphone application	-	++	+	- -	++	-

data has to be collected about occupancy rate over time at any location. It is also important to note that parking demand is more sensitive to rate increases than to rate decreases.

2.5 Comparison of the existing solutions

Table I sums up pros and cons[1] of each solution presented in this chapter. These are not objective measurements but way of comparing the different solutions on some key factors.

Paper based solutions are the cheapest to start with, and the driver does not need to make any sort of initial investments, therefore adoption barriers are very low. On the other hand they involve high operational costs, and the ease of use is very low for drivers. Park-meters are similar for final users, but a little bit easier to use, while being worse in terms of initial investment for the parking company. This approach has also a slightly lower operational cost than paper based solutions. Custom hardware has great ease of use for drivers and very low

[1]For each parameter being presented the plus symbol is used to describe its positive connotation and the minus represents a negative one. For example a plus symbol for an initial investment means that the investment is lower than the other solutions.

operational costs, but at the expense of a significant initial investment both for the company and the driver. Adoption barriers are especially high for this solution, because of the need to own the hardware. SMS text and smartphone application based solutions are similar and have some significant initial investment by the parking company and very low operational costs, no initial investments for the drivers (assuming that they already own a phone or compatible smartphone, respectively) and very high ease of use (of course it is better in the application version). Ease of initial setup is currently bad because of the data which is now collected before a driver can start using these solutions. Adoption barriers tend to be high mainly for the need to own a phone and a payment method supported by the service, such as a credit card.

CHAPTER 3

SOFTWARE REQUIREMENTS

In this chapter a series of requirements for the system to work effectively will be presented. This description will be as detailed as possible on the initial features, which are those features considered to be part of the Minimum Viable Product (MVP). Additional features will later be presented in Chapter 6, where future developments will be considered and the direction will be set for new features to be added in order to offer a more comprehensive service for both drivers and car park owners. In this MVP solution all requirements for the system to work initially are met, for all stakeholders, thus making this service usable. Chapter 4 will then describe how these requirements are taken into account in order to build the system. In this chapter the essential elements of the Software Requirements Specifications document are presented. In fact this document was prepared as an early phase of the project, and an extract from it is presented here.

3.1 Product perspective

This software is mainly divided into three parts, which together offer the overall service. Considering the fact that the idea is to be able to re-use as much as possible and make the system as flexible as possible as to reach all requirements of different towns and cities, modularity is a key focus, in fact different modules offer similar functionalities in different environments and with different constraints. The three parts are:

- User application: it allows drivers to pay for parking. In the future it will also offer additional features, such as the option to find free parking spaces
- Inspector application: it allows the inspector to validate the cars of users who are paying for the parking with the application; this is also be offered as an API, to allow access to this data from preexisting systems

- Management console: this is where the management can access all information and setup features for all pertaining parking spaces. In the beginning this portal only allows management to set up expenses by area and specific payment rules, such as maximum parking time (which is the equivalent of enforcing a rule against feeding park-meters), and to verify parking sessions in the past, for fine verification purposes.

3.2 Product functions

In the system we can identify various functions, for each of the parts. They will now be analyzed individually. A more detailed list of the features for each part can be found in chapter 3.

The user application will have to be offered in various platforms. The first platform to be supported is currently Android. Other platforms will be considered in the future. Initially the application will only support credit card payment (through a third party online service) and in the prepaid mechanism through the online account. A list of features that the application supports from the beginning is presented here.

- Ability to setup a new parking request in both prepaid and postpaid methods, both knowing the time beforehand and in a start/stop manner
  - in order to set up a valid parking session the application has to collect the plate of the car being parked (or the location, based on the agreements with the parking owner), the location where the parking space is (this is both to determine the fare and to allow for later validation) and information on whether the transaction will be paid beforehand or in the start/stop manner.
  - If the transaction happens beforehand, the application should direct the user to the payment page, also providing the length of the stay

  - Otherwise the length of the stay will be provided by the server, the application will have to only refer to the appropriate transaction when sending the user to the payment page
  - If the user is paying prepaid but the available money is less than the required, then either s/he refills using the credit card or the session cannot last more than the available amount.

- Ability to display pricing policy based on location
  - the application will have to make it as easy for the user as possible to detect the current location
  - it is then the responsibility of the user to check its correctness
  - it is responsibility of the server to communicate pricing to the user

- Ability to store user preferences, including car plates of the user and payment info. This feature is useful so that the user needs less input in order to accomplish what s/he wants
  - the list of car plates should be stored locally, so that the user can pick the car currently used out of a list; simple rules should be put in place to choose a plate automatically when multiple plates are available.
  - payment information won't be stored by the service, but will be stored by a third party; what the application will do is associate the payment information to an account for future payments, so that the PIN code of the account can be used for future purchases

- Ability to set a location or time alarm to disable paying for parking
  - in the event of a start/stop transaction, a notification should be displayed by the application if the user gets back to his/her car

  - in the event of a prepaid session, a notification should be displayed by the application if the parking session has expired

- Ability for users to refill their account through a web portal, for those people who don't trust their phones with credit card information
- Ability to delete account or personal information

The second application is for ticket inspectors. Initially this application will be offered for Android devices, later optionally supporting Windows Phone as well. A list of features that the application should support initially is presented here.

- ability to read the plate using the camera
- ability to retrieve payment information for that car in a specific interval of time (for example current if valid, for the last hour otherwise)
- the server side is built in order to offer APIs to integrate this service with other existing systems

Support for fine generation will be discussed later in this document, as it requires a tight relationship with the city council or park owner to determine the regulation involved in the management of a parking fine.

Finally the management console will be a web-based portal for the management team to do any back-office work they might need parking data for. A list of features that the console should support initially is presented here.

- Ability to set rules for paying such as: allowing start/stop, maximum time, pricing rules, on/off times of the day/week/holidays
  - as part of the rules a local holiday should be set as such in the system
- Ability to apply rules to all parking spaces of competence or to a specific area

- Ability to check past parking records for all customers, by car plate, user ID or other parameters.
- Ability to decide which powers to give to accounts of their companies

Fine configuration will be later introduced, after thorough discussion with city councils and park owners.

3.3 User Profiles

Considering the whole system, which is the combination of the user application, inspector application and management portal, we can identify three kinds of users: final user, inspector, manager. In the bullet-point list below there is a brief description of the three kinds of users (plus a fourth type, the administrator, which is not part of the MVP but will later become important).

- Final user: this is the driver who will use the application to pay for parking space. In the future this user will get access to all features for final users, such as the ability to find free parking space, when these features will be offered.
- Inspector: this is the person whose responsibility is to validate parking. Initially the feature offered to this user through the application will be only to check the status of a parked car, or in other words to verify whether the owner of the car is paying for it to be where it is. The procedure might change based on the solution adopted, but initially the search will be based on the plate.
- Manager: this is generally anyone who is responsible for the management of the car parks. This could be an employee of a private company or of the public institution responsible for car parks. Features offered to this user will be to setup payment rules and access data about payments made by final users.
- Administrator: there will be a need for a type of user who can create new accounts for managers, help final users with the system and managers with the portal. In the

beginning this feature won't be offered as it is not core for the initial phase, as the same result can be accomplished by letting the administrator have a management account for each county/company managing their car parks with this service.

3.4 Constraints

This system will be subject to many constraints based on what the car park owner will require. Some of the requirements can be expected beforehand, others are well known and others might rise during adaptation for each owner. In this section all constrains which are known beforehand will be presented.

- Parking should be payable by credit card. In order to achieve this, the final user application should support credit card payments, while respecting all security standards. In the beginning PCI compliance is not respected by the application itself, but the payment procedure is straightforward for final users by use of third party solutions.
- It should not be possible to create or edit parking transactions in the past, because this is the only way to make sure that final users have paid appropriately (if they actually have, of course).
- All edits to parking rules should be stored indefinitely (the time might be decided in the future, but data will be stored for several years anyways). This includes time when a new rule is added and an existing one is deleted.
- The software should be easily portable in different languages, initially English and Italian.
- All data about current people being parked should be easily exposed through an API, so that inspectors can actually check parking through the already existing solution for digital payment (where available).
- It should be possible for the park owner or city council to verify that the transactions taking place correspond to the total amount of money they collect, in other words the

park owner should be able to verify that the company offering the service is paying the park owner what it deserves based on transactions.

3.5 Assumptions and dependencies

In this project a few assumptions are made and dependencies are present to actually make it possible to build the system and offer the service. The main ones will be presented in this section.

In order to accept credit card payments without having to obtain sensitive information, the assumption is that a third service will be used so that the data of the user is stored for future purchases. Different options and services will be considered, but this should be as easy as possible in terms of user interaction and at the same time the final user applications will have to be developed in a way that allows for easy integration with any of these services. If this service does not allow for credit card validation, a first transaction should be performed before allowing postpaid parking.

Another point to be considered is that the administration of the infrastructure is out of the scope of this project, as it will be deployed on the cloud. The server-side software should be thought in a way to allow for quick porting from one cloud to another without having to change anything apart from the most superficial modules. As a general rule, the expected DBMS is a non - relational one, and since it is not known beforehand, a subset of the available features would be a safe assumption.

Finally the service should be able to interact with existing systems by the various car park companies. Of course this is not something that can be clearly formalized in terms of specifications because it is not known as of now, so the requirement is that there should be an API that allows for third party software to get access to all required information by the various users through the existing infrastructure. This also means that the parking rules should be flexible enough to support all rules of the companies which will use the service.

3.6 Use cases

In this chapter of the SRS document the use case model will be used to describe the system and to present the main features of smartP. Figure 1 shows a UML Use Case that represents the three actors of the system (final user, inspector, manager) and the interactions allowed by the system.

As shown in the picture, the types of actors are the final user, the inspector, and the manager. The textual use cases will cover all of the main features offered by the system and they will provide realistic scenarios. In this document UML 2 will be adopted. For more detailed information about actors see Section 3.7 of this document. The following paragraphs present some system-level textual use cases. UML is adopted as a standard to define them. Their detailed functionality is later presented in this document, but here the main elements are presented.

- A final user may: create or delete his/her account, log in, manage the account, add credit card information, register a car with the service, start a new parking session, stop a parking session, refill the account from a computer, setup an alarm for parking expiration.
- An inspector may: log in, log out, create a new account, validate parking through either a photo of the car plate or typing the car plate, check current car status and view its history, and (optionally) generate fines.
- A manager may: create or edit rules, both on a local and global (to all pertaining car parks) basis, manage calendar and holidays, log in and log out, create an account, check parking record, setup and change a parking area. As part of the rules a manager may setup or change pricing.

More detailed specifications are available in Section 3.7.

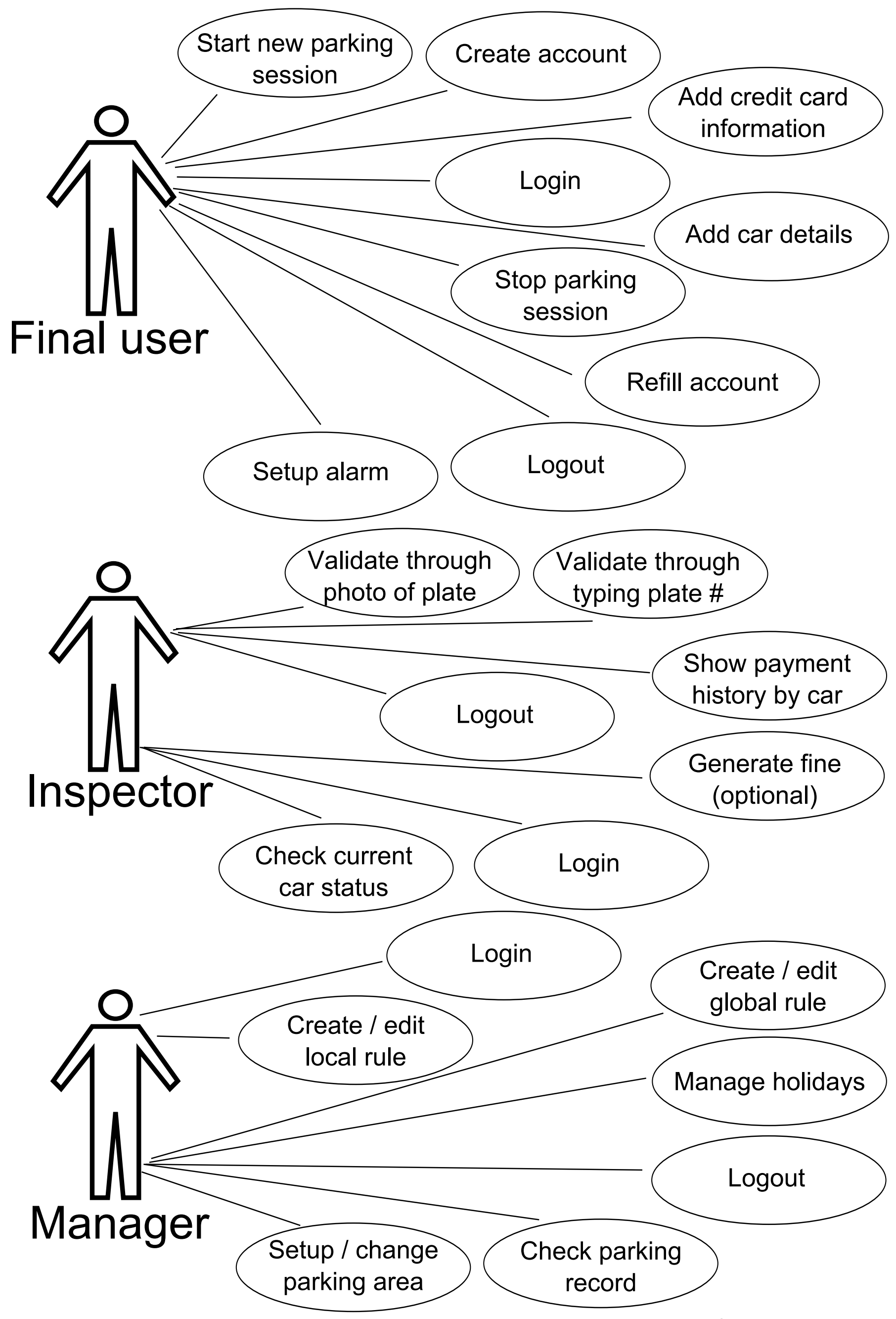


Figure 1: UML use case diagram

Use case: final user starts a parking session

type:	primary
precondition:	user has been activated for this feature by either validating payment method in the past or having an account balance greater than 0

normal flow:

1. the user enters the process for starting a parking session
2. a quick preview is presented to the user where s/he can specify which car s/he is currently using, which payment method s/he prefers for the specific transaction, the location where the car is being parked and the price in that location per time unit. If the car park owner does not allow to pay at the end based on the length of the stay, a length for the parking session will also be asked. All parameters are prefilled with the most likely values
3. the user changes any values which are not correct or filled in
4. the user starts the session
5. the user types the four digit PIN password
6. the system confirms that the session has started both through the application and by sending an email confirmation with a signed PDF document
7. the system asks the user whether to setup any alarms for the expiration of the parking session
8. the user sets up any alarms s/he wants

postcondition:	the parking session is registered in the system, with all of its data; the alarms are setup locally in the user application

extensions:

- if the user has to change the default answers to the various questions to start the session, different scenarios would be required, where the user either picks the preference from a list or has to add the required data to the system

- if the parking space owner allows for the start/stop paying mechanism, the payment process will have to take place when the user returns to her/his car through the application, just before leaving the car park.

Use case: final user creates an account

type:	primary
precondition:	the user respects all requirements to use the service

normal flow:

1. the user types in their email address and picks a four digit numerical PIN password
2. the system sends an email to the user, and asks him/her to activate his/her account by checking his/her email address.
3. the user verifies the account
4. the system notifies the application

postcondition: the user is added to the system as soon as s/he creates the account

extensions:

- if the user does not validate his/her email address the application will be in a state that says that the account needs to be activated
- only when the user actually verifies his/her email address is the account activated by the system - a disabled account cannot perform any operation other than the operations which can be performed by users who are not logged in
- final user login is exactly the same as account creation, but with the difference that after the user has typed the email address and PIN password, s/he is logged in skipping the email validation steps (that is, if credentials are correct)

Use case: final user adds credit card information

type: primary

precondition: the user is logged in, a feature has been requested which needs to charge money, and the user has chosen to add a new credit card as a payment method for this specific transaction (and potentially other future ones)

normal flow:

1. the user is presented a web page of the bank inside the application, where the amount to be charged is specified by the application through the system
2. the user fills in all required information and submits the form
3. the bank gives a confirmation on whether the transaction was successful or not
4. If it was successful the user will receive a confirmation inside the application and an email with a receipt of the transaction

postcondition: the transaction is recorded in the system, whether successful or not

extensions:

- if the transaction fails the user is notified, the system provides as much information as available about the cause for the failure

Use case: final user adds car details

type: primary

precondition: the user is logged in, the user is requested to add a new car (either because the application needs to have at least one to offer a feature, or because the user wants to add another one to the list

normal flow:

1. the user is presented a compulsory field (car plate) or to take a picture of the car plate
2. the user provides the required information
3. the user is then led to the application functionality, and step 4 happens only after the user has paid a first time for this car

4. the user is asked to type in (optionally) a name for the car, and to take a picture of the car
5. the user provides optional information if appropriate

postcondition: the new car is added to the user account, thus available for use in future car parking transactions

extensions:

- if the user adds a car which is already present in his/her account the operation is disregarded
- if the user does not provide optional information in its entirety only the available information is stored and will be presented when appropriate. It will be possible for the user to add optional information later

Use case: final user stops a parking session

type: primary

precondition: the user is logged in, the user has started a parking session with the start/stop manner, and has not stopped it yet

normal flow:

1. the user manually stops the parking session
2. the system notifies the user that the parking session has expired
3. the system presents the user a page of the bank, where s/he only has to confirm that the total is correct to proceed with the payment
4. the user confirms
5. the system notifies the user about the status, it offers to try again in case of failure and, if successful, sends an email with a signed PDF as proof of the details of the session, as a receipt

postcondition: all information about the transaction and about the parking session is added to the database, whether successful or not.

extensions:

- if the user cannot complete the payment transaction s/he can ask to receive an email with the link to proceed with the payment from his/her PC or tablet
- if the transaction fails the user is given the choice to change his/her payment method
- if the user has not paid within a certain amount of time (to be decided) s/he will be solicited to pay both through the application and email and finally be charged a fine

Use case: final user refills his/her account

type: primary

precondition: the user is logged into the website

normal flow:

1. the user accesses the feature for refilling the account
2. the user is asked to pick an amount within certain ranges (minimum amount, maximum amount)
3. the user is sent to the bank web page, where s/he will give all information required for the transaction to take place
4. if possible, the user will be registered when s/he comes back from the bank page
5. the user is presented with a confirmation of the transaction and new account balance
6. a receipt will be sent via email in case of a successful transaction, with a signed PDF as a receipt

postcondition: all information about the transaction is added to the database, whether successful or not; moreover information about user payment method is stored in the database if available

extensions:

- if the user cannot complete the payment transaction s/he can ask to receive an email with the link to proceed with the payment from his/her PC or tablet
- if the transaction fails the user is given the choice to change his/her payment method
- if the user has not paid within a certain amount of time (to be decided) s/he will be solicited to pay both through the application and email and finally be charged a fine

Use case: final user sets up an alarm

type: primary

precondition: the user is logged in and has just started a parking session

normal flow:

1. if the parking session is of start/stop type, the user is suggested to start a geographic alarm otherwise a timing alarm
2. if the first type is used and the location is not available to the application the user is asked to allow access to this information, if it is of the second type the user is asked how long before the expiration time s/he wants to be reminded about it
3. the system notifies the user when the condition is met

postcondition: all information about the alarm is stored remotely on the database and locally for the alarm to work

extensions:

- when a user logs in the application checks whether there are alarms about current sessions related to the current user, and if there are any sessions; if there are alarms they are setup and, if the location is required but not available, the user is notified
- in some car parks it might become possible to renew (or extend) the duration of the parking session remotely, in this case the feature is offered automatically when the alarm is active

Use case: inspector validates the current status of a car

type: primary

precondition: the inspector is logged in

normal flow:

1. the inspector chooses the feature to check a car either by camera photo or by typing
2. if the inspector chose to type a field for the car plate is presented, otherwise the camera application is launched to allow the inspector to take a picture of the car
3. if the inspector took the photo, the car is translated into text and written in the same field that the inspector would have used if s/he typed it originally

postcondition: the system then provides the inspector with current transaction of the car (if valid) and all transactions in the past for a given amount of time and up to a certain number (it could be the 3 latest transactions in the last 24 hours

extensions:

- if the inspector uses a custom preexisting system smartP has to account for that, offering compatible APIs
- if there is a current transaction for the car a check is made on whether the payment is right according to location, using the location of the inspector

Use case: inspector generates a parking fine (optional, will be implemented at a later time when possible)

type: primary

precondition: the inspector is logged in

normal flow:

1. the inspector chooses the feature to generate a fine either through the feature to get details about a car or through the main menu

2. the application asks for all additional information required by law to generate a fine, including a signature if required
3. the system checks if the fine is consistent with the available data
4. the data is stored in the database
5. if the car plate is present in the system as one of the users the fine is sent directly via email
6. otherwise (or if the user does not pay within a certain amount of time) the fine is sent the same way as it traditionally would

postcondition: the fine is saved and added to a queue according to company policy; it might be sent to the final user either through email or by post

extensions:

- if the user pays the fine in time through email s/he receives a receipt and no fine will be sent by post
- all data is stored and cannot be changed afterward

Use case: manager creates a local rule

type:	primary
precondition:	the manager is logged in
normal flow:	

1. the manager selects the feature to create a new rule
2. the system suggests a drop down menu for the type of rule
3. based on the choice of the user the parameters of the rule are presented
4. the user fills all required fields in
5. the user picks the area where the rule applies
6. the system checks for conflicting rules

7. the rule is added to a change list
8. when the manager is happy with the changes and the new changes are congruent with what was present, the manager can apply all changes (a time of validity can be specified)

postcondition: all changes are applied and a history of changes is kept in the database, together with the account number of the manager who applied them

extensions:

- if any rule is not valid the conflicting rule will be presented
- if an existing rule is changed it is as if the old one is deleted and a new one is created

Use case: manager manages holidays

type: primary

precondition: the manager is logged in

normal flow:

1. the manager selects the feature to manage holidays
2. a calendar is presented where it is possible to view all days in two colors: either non holidays and holidays
3. the manager can manually click on any day which will change status based on what it currently is
4. a list of all holidays is available, focusing on the changes being applied
5. the manager approves changes

postcondition: all changes are applied and a history of changes is kept in the database, together with the account number of the manager who applied them

extensions:

- this does not include details on whether during holidays there is a different pricing scheme, as this is setup through a rule
- this does not include Sundays, as they are known by the system and can be selected in rules as Sundays

Use case: manager checks parking record

type: primary

precondition: the manager is logged in

normal flow:

1. the manager selects the feature to check parking records
2. a form is displayed where the manager can type the car plate number, and optionally specify a time interval
3. the system will provide a list of all pertaining transactions, with all of the available data on those transactions and the parking spaces of competence

postcondition: the requested data is presented to the manager

extensions:

- if no time period is presented, all results by car plate and park owner are presented
- (low priority) historic data of acquired companies which used smartP should be accessible

Use case: manager sets a new parking area up

type: primary

precondition: the manager is logged in

normal flow:

1. the manager selects the feature to manage parking areas
2. the manager selects the feature to add a new area
3. the manager selects all parts of the area (either graphically or in textual form, this is yet to be determined)
4. the new area is described by the system (again either graphically or in textual form) and points out conflicting areas
5. the manager approves all changes to areas (a time of validity can be specified)

postcondition: all borders of the changed areas are updated, together with time of change and manager ID

extensions:

- if a new or changed area overlaps an existing one the conflict is presented to the user and, until it is fixed, changes cannot be applied
- an automatic fix of removing all points pertaining to the newly created area from the old overlapping area automatically will be provided

3.7 System requirements

This section contains both functional and non-functional software requirements which are either supported by the current version of the system or are important to be introduced in the future, in which case the current system takes into account those future developments in order to maintain backward compatibility as much as possible.

Functionality

In this section the functionality requirements for all the components of the smartP service will be covered. These will be grouped based on the user who can get access to them. They might involve one or more components, as they might involve both server side and client side. To begin with, all functions which involve the system but not a specific user will be presented.

The system will have to support three types of accounts: final user, inspector and manager. Based on the type of the account, the user will gain access to different functionalities. Each account should be uniquely identified by a user-name and verified with a password. They should be also be able to create their own accounts, whether active or to be activated. The administration of accounts of other people should be only allowed to administrators.Users should be able to log out from any device they might be logged in, from the device itself. This feature is also to be used in order to log in with a different account from the same device. Users should also be able to delete their account.

A series of requirements involve the different types of users. To begin with, the ones pertaining to final users (which almost always are drivers) are listed. The final user should be able to start a new parking session. The application will have to communicate to the server according to the specific requirements for a specific location in terms of data to be collected for the parking session to be valid. If the session is postpaid, it should be possible for the user to stop the session when s/he is leaving, leading him/her to the payment page if required by the credit card processor. The final user should be able to add credit card information (as mentioned earlier, it is the initial objective to offer this through a third party provider). Initially it will be possible to store only one credit card per account at a time , but different payment methods will be added in the future. The final user should be able to add details about a car s/he wants to use the service with. The car plate number is compulsory while the other data is not (other data initially includes car name, but could potentially be extended to other relevant information about the car). For people who do not trust their phone with credit card details, it should be possible to refill their account on the web using a credit card. Finally, to improve the user experience an alarm feature should be offered to notify users before their parking session expires.

Requirements for inspectors involve the inspector application exclusively, as it is the only interaction they have with the system. They should be able to validate the status of a parked car both by taking a picture of its plate and by typing its number. Either method should work and it should be the inspector's decision which one to use, unless s/he is generating a fine, in which case s/he has to manually type the car plate number for absolute certainty of the status (this of course only becomes relevant when the fine generation feature is activated for a specific parking company or city).

Managers should be able to create new rules and edit existing ones. Rules should have a local or global validity, where global means in all parking spaces of responsibility of the manager. Rules are a form to describe the parking conditions, and therefore it is the management responsibility to set them up initially and keep them up to date. For example the manager should be able to express which days of each year correspond to holidays. These should include local holidays (if any) and therefore should be applicable to local areas as well as to all areas. Moreover the manager should be able to access a history by car plate of all parking transactions, optionally filtered by initial or final time, or specifying a time interval. History should include all available information about the parking transaction. The manager should also be able to check whether a car had the right to park freely in a specific area, time and day, according to the rules at the time (it is important that this is according to the rules at the time, because the system should not use the rules which are valid at the time of the query by the management, even if the rules have changed in the mean time. The manager should be able to create a new parking area or edit an existing one. No area should overlap an existing one. When this happens either the overlapping area ends in the old preexisting area or in the newly created area, based on the manager's decision. Rules should be validated against existing ones before being applied. If there is an already existing update plan, the manager has either the choice of deleting it or starting the new changes from there.

Logical Controller

The pictures below shows a series of finite state automata which model the abstract controller of the overall system, using the UML2 conventions. They show the logical interaction between the states based on different conditions. In Figure 2 the automaton for the inspector application is presented. The area surrounded by dots shows the part which will not necessarily be developed initially, as it is not a critical requirement. In Figure 3 the one pertaining to managers is presented and in Figure 4 the one pertaining to the final users application. The account refill feature is not presented here as it is not offered through the application but solely through the website.

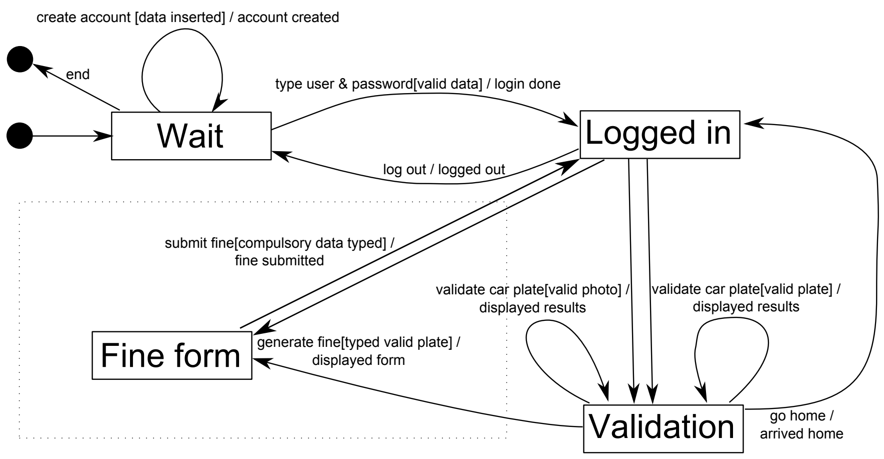

create account [data inserted] / account created
end
type user & password[valid data] / login done
Wait
Logged in
log out / logged out
submit fine[compulsory data typed] / fine submitted
validate car plate[valid photo] / displayed results
validate car plate[valid plate] / displayed results
Fine form
generate fine[typed valid plate] / displayed form
Validation
go home / arrived home

Figure 2: Finite state automaton of the inspector application

Usability

A series of graphical user interface considerations are presented here, even if Chapter 4 will present a mockup of the various elements of the system and additional detail on the choices made. The user application should minimize user input, especially it should avoid typing as much as possible. Moreover the inspector application should make it possible for inspectors to check a car in less than 5 seconds, using the camera mode, and the same time (excluding typing) in the version where the inspector types the plate number. Both of these requirements are essential for the success of the system.

The system should be built to support multiple languages. Initially it will be available in Italian and English, with additional languages added at later times. All notifications should provide a clear description in the language of choice of the user out of the available ones of the service.

Reliability

The system will be available 24/7. Requirements and conditions on availability will be discussed with each company using the service. The service might be put off line in certain

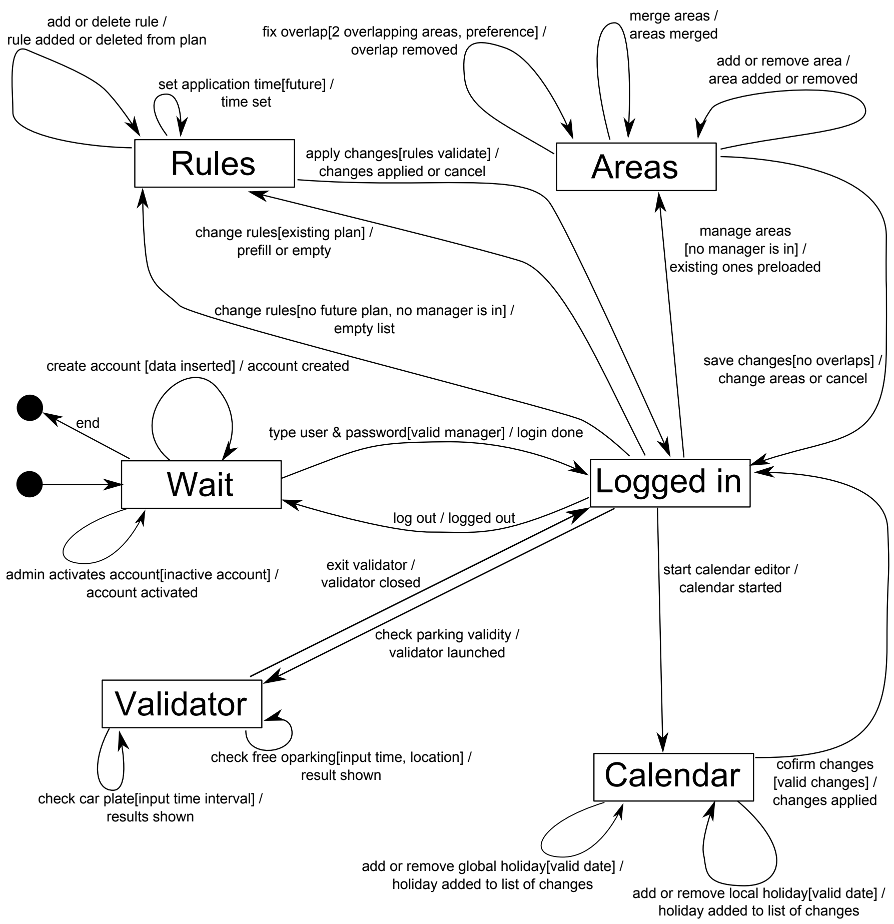

add or delete rule /
rule added or deleted from plan
set application time[future] /
time set
fix overlap[2 overlapping areas, preference] /
overlap removed
merge areas /
areas merged
add or remove area /
area added or removed
Rules
apply changes[rules validate] /
changes applied or cancel
Areas
change rules[existing plan] /
prefill or empty
manage areas
[no manager is in] /
existing ones preloaded
change rules[no future plan, no manager is in] /
empty list
save changes[no overlaps] /
change areas or cancel
create account [data inserted] / account created
end
type user & password[valid manager] / login done
Wait
Logged in
log out / logged out
admin activates account[inactive account] /
account activated
exit validator /
validator closed
start calendar editor /
calendar started
check parking validity /
validator launched
Validator
check free oparking[input time, location] /
result shown
Calendar
cofirm changes
[valid changes] /
changes applied
check car plate[input time interval] /
results shown
add or remove global holiday[valid date] /
holiday added to list of changes
add or remove local holiday[valid date] /
holiday added to list of changes

Figure 3: Finite state automaton of the manager functionality

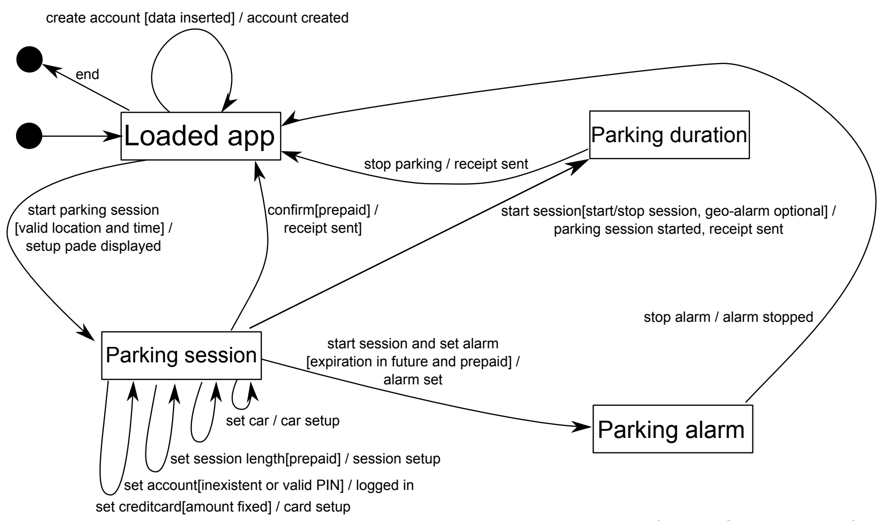

create account [data inserted] / account created
end
Loaded app
Parking duration
stop parking / receipt sent
start parking session [valid location and time] / setup pade displayed
confirm[prepaid] / receipt sent]
start session[start/stop session, geo-alarm optional] / parking session started, receipt sent
stop alarm / alarm stopped
Parking session
start session and set alarm [expiration in future and prepaid] / alarm set
Parking alarm
set car / car setup
set session length[prepaid] / session setup
set account[inexistent or valid PIN] / logged in
set creditcard[amount fixed] / card setup

Figure 4: Finite state automaton of the final user functionality

conditions, for example for maintenance, but this should not be a problem for any user who was using the service at the time. In other words all parking sessions should expire before the system can be turned off and a mechanism should be put in place to manage the situation where this is not possible (for example by terminating all postpaid parking sessions after the downtime period without charging for that time interval).

Error rate should be lower than 1 showstopper error per hour of continuous use. User will never see a technical error message. In case of an error, the user will see a brief explanation of it. If an unhandled error occurs, the user will get a generic error alert and there will always be instructions or suggestions on how to proceed.

Security

Security is a very important part of this project, as money is involved and all stakeholders have potentially conflicting interests. For this reason a series of security requirements are introduced. Based on the account type (level), different functions should be available to the user. In terms of authorization, the system has to allow only people who have an appropriate account to access their corresponding features. User access rights are defined in Section 3.2.

There are also a series of authentication requirements for different types of users. The administrator has to confirm the action by retyping his/her password, before applying any management change to the system. Moreover the manager is logged out every 20 minutes and s/he has to log in again by using his/her password in order to create a new valid session. The ticket inspector will also have to type his/her password for a specific feature, which is the fine generation and submission, when this option is available. The session for an inspector lasts one hour, after which s/he has to log in again by typing the password. Finally, final users should be logged in on the device before they can start or stop a parking session.

Auditing is also very important for a series of reasons. To begin with, this should reduce chances of hacking the system in order to park for free or at least find out when and how this happens. Moreover this allows for retrieval and differential reconstruction of the status of the system at any given time, which is already important for some features (e.g. the verification of past parking sessions) but might be necessary for future features as well.

For the final user's money transactions the following data is collected:

- time and date
- payment mechanism
- amount
- cause for transaction
- user name associated with transaction

For the final user's parking transactions the following data is collected:

- start time and date
- stop time and date (recorded separately if in the start/stop manner and not transaction)

- location
- car plate
- user account
- money transaction associated (if start/stop manner only present with stop record)

For the manager changing rules the following data is collected:

- differential rules as compared to the previous status
- date set for the application of the rules
- manager account of the responsible for the changes

For the manager enabling accounts the following data is collected:

- date and time of activation
- account being activated
- type of activation (inspector and/or manager)
- manager account of the responsible for the changes

For the manager changing calendar holidays the following data is collected:

- list of holidays added
- list of holidays removed
- areas of applicability for each holiday being added
- manager account of the responsible for the changes

For the manager changing areas the following data will be collected:

- list of all areas being created, with all included land
- list of all areas being changed, with all included land

- list of areas being removed
- manager account of the responsible for the changes

The data to be collected for the inspector filing a fine is susceptible to change, as it greatly varies on jurisdiction and will be discussed on a case by case manner. Generally the following data will be collected:

- location where the car was parked
- photo of the car
- car plate
- inspector account of the responsible for the changes
- initial value of the fine
- date and time the fine was generated
- any additional information, based on requirements of each company

Performance

All of the data suggested here is a maximum limit, over which the service would result seriously inconvenient. On the other hand the goal is to perform, on average, much better than the worst cases presented here. Launching any of the developed applications should always take less than 30 seconds. The loading of the website should be under 15 seconds. As far as memory consumption is considered on the server side there are no limitations (other than the ones imposed by the cloud service provider), for the applications the maximum limit should be 50MB, and all data should be freed when the application is in the background unless it is required for the application to function properly.

Response time should be lower than 5 seconds for mobile applications and lower than 2 seconds for the management console at least in 90% of the situation, assuming a working internet connection is available.

Scalability and deployment

The system is designed with scalability in mind, initially designed for Google App Engine in a way that it makes it easily portable (that is, with minor changes) to other similar cloud offerings.

Installation by final users will be accomplished through official application stores for the specific operating system, while on the business side installation will be performed by the smartP staff. Installation by managers is not required as the cloud service is offered as an option. On the other hand managers will have to setup the service with the specific rules for their managed parking spaces before final users can start using the service in those spaces. Installation by inspectors will not be required, as handsets will be provided with a preinstalled version of the application for inspectors.

Upgrade for managers will not require any action at all, as this will entirely happen automatically, on the server side. On the other hand upgrade for final users and inspectors will be suggested or forced (based on requirements) remotely, and final users will be guided to upgrade to the latest version through the store, while inspectors might use a different upgrade methodology (it will be determined case by case).

Maintainability

Since this software is freeware for final users, no warranty is offered to them, apart from the fact that the signed PDFs they receive from the service have legal value. Specific warranties might be offered to park owners, such as availability, but this will be considered case by case.

Security fixes will be forcing updates in the users' devices and will be released as frequently as required. Other bug fixes will be aggregated to security fixes whenever a new release will be required.

In case of a change in the developer team, the new developer should be able to understand the source code easily by reading the comments in the code and all related documentation, such as the Software Requirements Specification document.

Operating system

The initial choice for the final user application will be Android 4.0, later porting it to previous versions of Android up to version 2.2. The iPhone version will also be released in the future, with a support from iOS 3 onwards. Other application development might be considered in the future, such as Windows Phone and Blackberry. An internet connection should be available when actively using the application.

The inspector application will be initially developed for Android 4.0 and then retro-compatibility will be added up to version 2.2. A Windows Phone version might be built in the future to offer choice and support more hardware. An internet connection should be available when actively using the application.

All areas of the website will be developed using GWT, so that most browsers will be supported. Requirements will be to have a working internet connection, a browser which supports JavaScript and cookies.

Design constraints

Java EE will be used as a development language for the server side code, as it will be run on Google App Engine initially. Java will also be used for all Android development. Other devices will be later considered when their development will be planned.

Purchased components

There will not be any need to purchase components initially, and this will be avoided as much as possible to keep costs down. Devices for inspectors might have to be purchased if the company/city council does not have a current solution. If these devices have to be purchased they have to be connected to the internet, have a camera, use an Operating System supported by the inspector application and a battery life of a working day, under heavy use conditions.

Interfaces

Hardware requirements for smartphones are satisfied if the appropriate operating system is installed. Of course location based features work best with a GPS. Moreover, inspectors need devices which can operate long enough to last for a day before being recharged. On the server side, requests are that it scales automatically, that it abstracts a simple database and that it offers MemCache. Initially Google App Engine will be used but the system will be designed to require minor changes to move it from one hosting provider to another.

User interfaces for the system are the application user interfaces and website. All interactions between users and the system are described in Section 3.6.

The various software components of the overall system will be able to communicate with each other. Moreover an additional set of APIs will be offered to allow for existing systems on the management side to access information about recorded transactions.

A network connection is required from all devices by all users when actively using the service. During a parking transaction the connection might not be required, but it is then required to stop the transaction when the user is leaving, if it was of the start/stop type.

3.8 Guidelines for an application-based parking payment solution

The suggested framework should have a series of modules which would allow for all of re requirements of this Chapter to be met. More specifically it should take into account the three main stakeholders: drivers, ticket inspectors and the parking management.

To begin with, drivers should be able to park their cars and pay for it using their smartphone, and the procedure should be as easy as possible, in terms of GUI and in terms of required steps and data input by the user. Moreover the application should be compatible with the main smartphone operating systems in order to reach the highest audience possible, and it should also allow drivers to pay for parking in as many locations as possible. The ability to offer post-paid parking is a bonus for drivers, so the system should allow the management to enable this feature.

Secondly, ticket inspectors should be able to constantly verify if a car they see parked is paying the parking fare through the smartP application or is not paying at all. For this reason these people have their own accounts and smartphones with a custom application which lets them check the parking status of any car, based on either car plate or parking spot number (if parking is numbered).

Moreover the management should be able to verify that fines are valid. Two relevant features for park owners are also introduced. First of all they should be able to verify internally that they get paid the right amount by smartP whenever a driver pays for parking. Moreover they should be able to allow as many competitors of smartP as they see appropriate: all of them should have a common API to access information and to communicate sales of parking spots. Finally park owners should be able to set up rules for smartP to know how much it should charge users. This characteristic has to be as elastic as possible, to fit different business models and underlying systems, therefore the rules should have as few limitations as possible.

Finally a few considerations are made about the need for a solution which scales and does not require system architects to operate; at the same time the solution would have to provide a certain level of generality to be ported from a cloud environment to a different one, or to privately owned servers.

CHAPTER 4

THE PROPOSED SOLUTION: SMARTP

The goal of this chapter is to present the proposed solution, which is a description of the decisions made to build the system and the reasoning behind these decisions. When this is beneficial, a reference to the requirements will be made, but not all requirements will be referenced here: some of them are very specific and they do not involve the architecture of the system or its modules. For example the ability for a user to delete his/her account is of course a requirement, but this is just a feature to be implemented and therefore there is no point in discussing it at an architectural level, and this goes for other requirements as well.

This chapter is divided into a series of sections: in Section 4.1 the whole system is presented, with different alternatives of deployment and the essential interactions between components. In the following sections each module constituting the overall system is described in more detail. Different versions of each module might be available (and will be later explained for the specific cases) and all of them should be interchangeable.

4.1 Overall system architecture

In this first section the overall system structure is introduced. First of all, Figure 5 gives an overview of the system. As the picture shows, there are various parts or components. The arrows represent communications taking place between components, which are described in specific APIs, and are not constantly happening but only when the requesting component sends the request. All cross-component communications happen through JSON, in order to support compatibility between alternative versions for each module and to optimize the data transfer (e.g. compared to XML, which would have the same compatibility level, JSON allows for a more synthetic syntax of the API requests). Here is an example of a JSON response of the smartP back-end to the User application, when the request for a full update is made:

```
{
    new-key:chgdeqdhnhs3qtvb4c968dmer7,
    status:update-all,
    cars-list:
    [
        {"index":"ABC123","name":"BMW"},
        {"index":"456CDE","name":"Ford Ka"},
        {"index":"789GHI","name":"Chevy Volt"}
    ]
}
```

In this example there are three cars in store for the requiring user. For each access to the login APIs a new key is generated. This is to be used by the device for the next communication.

The user application is the application to be installed on the driver's smartphone. Initially this is supported by Android phones, versions 2.2 and up, but other versions will later be introduced to support other platforms. The key element is that these applications are interchangeable and use the same APIs to communicate to the rest of the system.

The user portal is a module in its early stage and it currently offers only one functionality which is an initial requirement: to allow for users to refill their accounts online. Given the fact that this feature is not complex, this module will not be presented in a separate section, but in the long term many other features will be offered through this channel. On the user's side all that's required is an Internet connected device: the service will be served as HTML pages and JavaScript client side code which will be executed inside the user's web browser. In order to support the largest amount of web browsers possible, Google Web Toolkit will be adopted as a client-side framework. This is also positive because it is built as Java code, with all the advantages which come from this in terms of code testing and debugging. Communication between the GWT module and server services will be, again, through JSON responses.

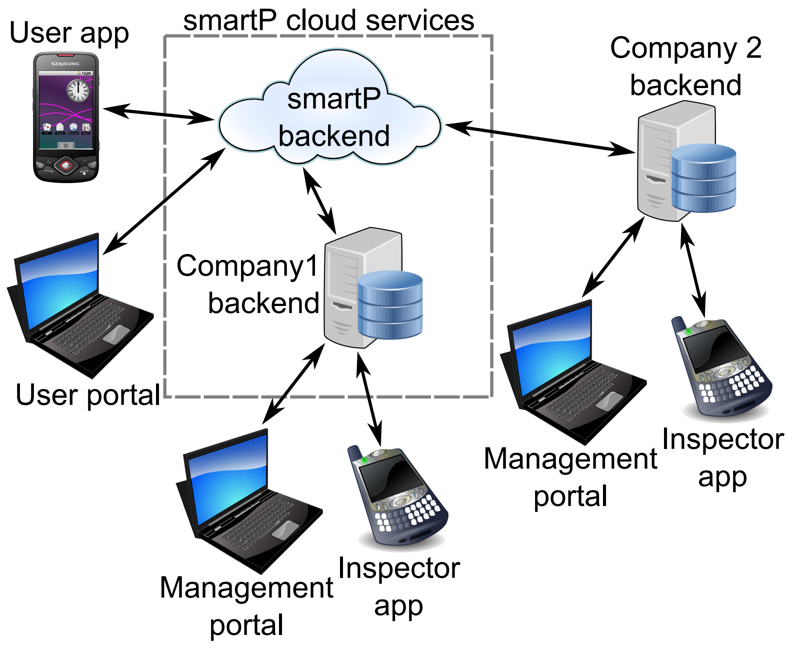


Figure 5: Overall system architecture of the proposed solution

The smartP back-end is the main server, where all information about users and their payments is kept. The decision was to use Google App Engine and therefore Java EE as a programming language and framework. The goal of this component is to make sure the company back-end is informed about any parking session and to make sure that the user pays according to the fair amount for the service used.

The company back-end has the same role the smartP back-end has, but for the Inspector portal and the management portal: the goal is to make sure that these modules can send and retrieve the information they need. The biggest difference with the smartP back-end is that this module is at a company level: different companies will have different deployments of the module, which is the reason for the decoupling of the APIs into two back-ends (the smartP and the company ones). In the figure two company back-ends are presented. The difference between them will be later explained in Section 4.5, together with the explanation for the reasoning for the decoupling.

The inspector application is initially an application to be installed on Android devices and, just like the user application, different platforms might be supported in the future. The reasoning behind the choice of supported platforms is here important: the ordering will be based on the variety of devices supporting the operating system, in other words Android was chosen as the first platform because it is the OS with the widest environment of devices. Again, the API will be built in a way which allows for new applications supporting other platforms to be compatible with the relevant modules.

The management application is similar to the user portal in terms of requirements on the client side: a web browser and an internet connection. The client-side framework is again GWT, and the server side is in the company back-end. The management portal offers the same features which are required and were presented as requirements, more details on how this works will later be presented.

Finally the company back-end is based on the Google App Engine, so it is developed in Java and with the EE framework. This one communicates with the inspector application about the status of parked cars and, in the future, it will communicate about fines. It also communicates with the management portal, to give information to the management about past parking sessions (or lacks thereof) and it also allows the management portal to edit existing parking rules. Finally this module communicates with the smartP back-end to give information about parking rules in a specific location at a specific time, and also to receive information about parking sessions taking place or ending at the time.

A simulation of the system would be for example: the manager sets the rules for the parking company. The driver wants to start a parking session, the user application requests information about the rules to the smartP back-end which forwards the request to the back-end of the appropriate company. This gives a reply which is then routed to the user application.Then the application shows the information to the user, who is given the chance to start a parking session according to these rules. When the user decides to start the parking session the economic transaction happens (either at the beginning or end of the parking session) this information is only shared with the smartP back-end, user application and payment gateway (more information about the payment gateway will be presented in Section 4.3). On the other hand information about the parking session is passed to the company back-end, so that it can answer to requests about parking sessions both from inspector applications and from the management portal. If the ticket inspector checks the parking session with the application, it will request information to the company back-end, which already knows the answer about all parking session questions in the present and in the past.

4.2 User application

Initially the decision was to develop the user application on Android (because of re-usability of some modules with the inspector application) based on the following criteria: to begin with, the Android platform is now the most adopted smartphone platform, moreover it is the platform with the highest variety of devices. These two things together make it especially convenient to start with Android, later extending both applications to other platforms but not necessarily in parallel.

The application is built in a way which aims at minimizing manual user input and, in order to accomplish this, a series of inputs is used, as shown in Figure 6. In particular the camera is used to detect the car plate without the user having to type anything. In the future the camera will also be used to read credit card information, and for this feature only an initial version of a mock-up will be presented, because the idea is definitely important but it cannot be built without having the application itself manage the credit card data collection and transmission, which means that the smartP service would then have to process this information directly, without the option of a payment gateway as an intermediary. More information on why this is something which will not be implemented initially is presented in Section 4.3.

The GPS represents location services, ton only GPS geolocation. In fact in Android different sensors can provide this functionality, for example the use of WiFi to determine the approximate location or assist the GPS in more quickly determining actual location. All that matters is that if, by using sensors, the location is clear enough to unequivocally determine the parking rules or set up a parking session, then no user input is required for the location detection.

Finally there will always be a component of manual user input. The goal is to minimize the number of clicks to achieve user's objectives and, more importantly, minimize the amount of input through keypads or hardware keyboards. The current version of the Android application will be presented both in a description and with screen-shots. An additional feature to reduce the number of clicks will be a Wizard path for the first time the application is launched, which is not part of the first version but, given the importance of reducing user input, it will be presented with a mock-up.

The last input for the application is the internet connection, from which it gathers information by requesting features to the smartP back-end. It is essential for the first version of the application to be connected to the Internet to accomplish most features successfully, even if in the future an option to use texts instead of Internet requests might be offered (especially to account for those situation when the network signal is not strong enough for an Internet connection but it is strong enough to send or receive texts.

In terms of GUI a strong attempt has been made to respect the rules of the Operating System, and more specifically to create an experience which is consistent with the underlying Operating System and version. For example different layouts in Android have been created for different versions of the OS as exemplified in Figure 7. This shows how, from version 2 to version 3 of Android, they inverted the button for the positive and negative action. Only supporting the older format (on the left of the picture, depicting version 2.2) of the menus with the confirmation on the left side of the menu and the negative response on the right of the menu, would feel unnatural to users of Android from version 3, who on the other hand are

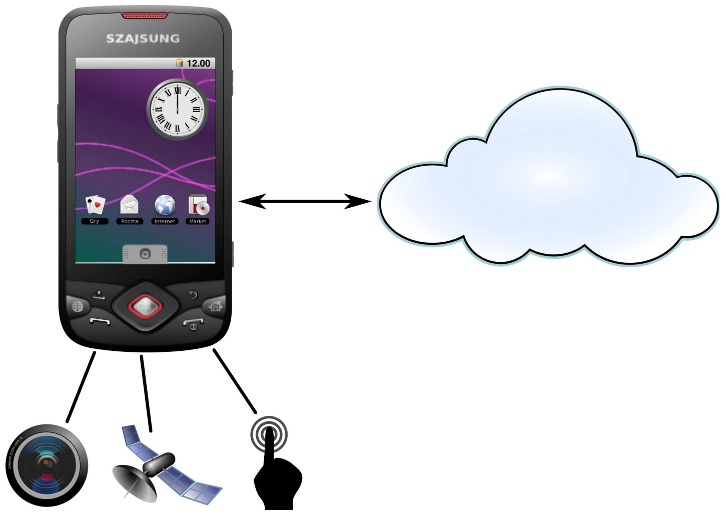


Figure 6: Overview of the user application and its interactions with the outside World

used to having the positive button on the right (as shown on the right picture, depicting the corresponding Activity under version 4.2).

There is a limited set of activities in the application, which are the minimum required to offer all features. Figure 8 presents all current Activities, in order to allow for a better understanding of the description of the application. A code snippet is provided to show how a different graphic style is provided for Android versions 11 and greater (this is part of value-v11/styles.xml):

```
<resources>
    <style name="AppBaseTheme" parent="@android:style/Theme.Holo">
</resources>
```

All activities are described here in order from the top left to the bottom right. The application also has a payment section, but it is not depicted in any of the screen-shots for two main reasons: technically it is accomplished by using a third party which decides the graphics, and also the payments procedure will be described in Section 4.3.

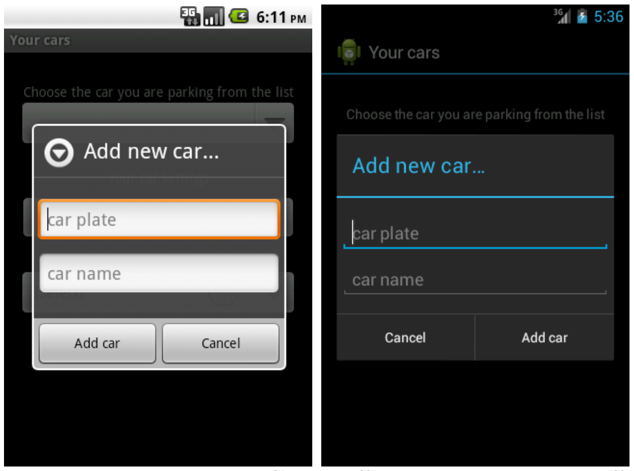


Figure 7: Example of the user application GUI differences between different Android versions

The login screen is essential, simply asking for an email address and a password. This form can be used by both people logging in and people creating a new account. The user input required for this step could be reduced by supporting Google Accounts already registered on the device. The reason why both fields are important for smartP to collect is for authentication and authorization.

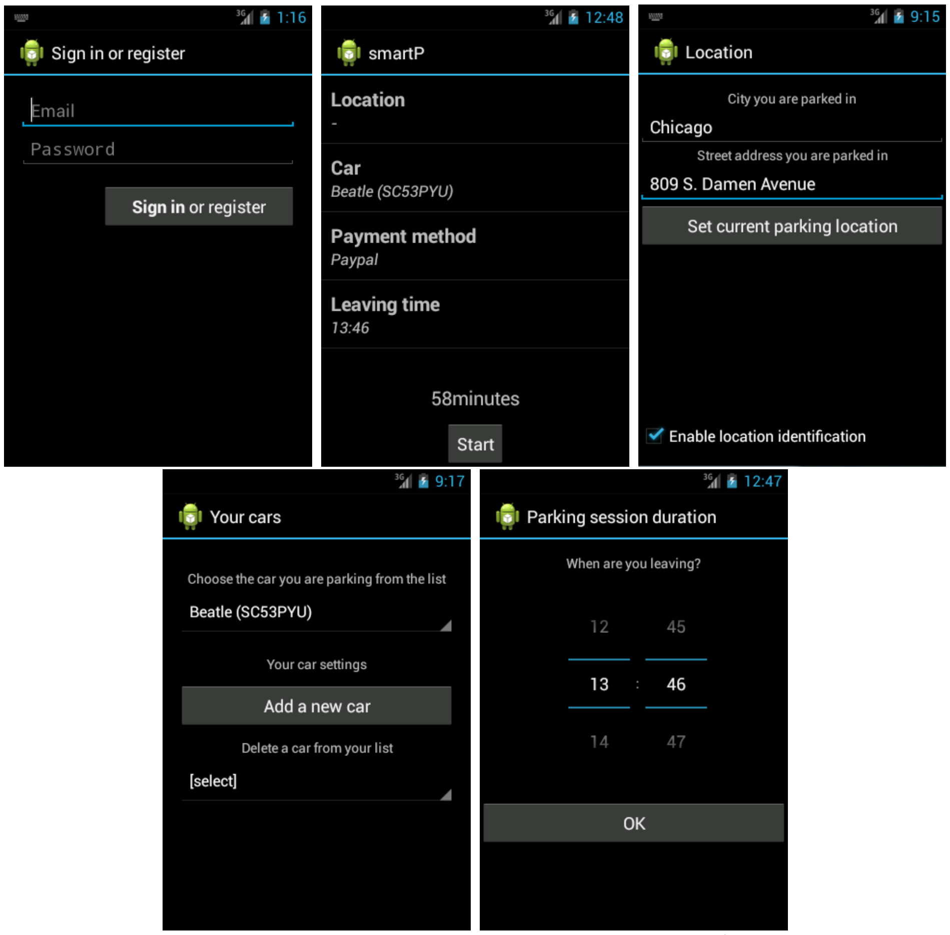
The home screen essentially presents all settings to start a parking session, whether they are known or not, allowing the user to launch different activities by clicking on the sections to set-up data which either is missing or incorrect. In most uses it is expected that all fields will be prefilled correctly and therefore the user will simply have to start the parking session with a single tap on the appropriate button of the home screen. The text above the start button shows how much time there is before the parking session expires. Both this last text and the last section (Leaving time) disappear where the parking owner allows for start-stop parking sessions, in which case the user just starts the session by setting up the other information.

The location screen has two fields: first of all a city and then a street. When a database is ready, while the user types the beginning a list of compatible values appear, for making this process faster. To make it even faster, location services can be used (if enabled) to automatically fill in these values. This is the reason for the check-box at the bottom of the screen, which allows for users to go to the settings of their device where they can activate location services. Most of the time users will not have to set up this screen and the relative information, because the location services will provide the required information which will be displayed on the home screen.

The car management screen allows for users to set up or change the car they are parking. This is divided into three parts: a drop-down menu to select a car from the list of the cars already added to the account by the driver in the past, a button which allows the user to add a new car (by showing a pop-up window where the user can type the car plate and optionally choose a car name), and finally a drop-down menu which allows the user to delete a car which is already associated to the account in the system. Again this feature will rarely be used, because the assumption is that any driver will likely use the same car every time s/he parks and therefore remembering the car which was last chosen is enough to show the correct one to the user on the home screen and avoid asking him/her to pick a different one (unless of course s/he wants to).

Finally the parking session duration screen allows users to set up the duration of their parking session. This uses the rules set by the parking owner to determine what is the minimum leaving time (assuming that the company decides that there should be one, e.g. each parking session should last at least 30 minutes). As a default value the screen will suggest a session of one hour from the moment it is opened, the user can then pick the right time for them. Of course before a session is started from the home screen the minimum duration is checked to make sure that the session is valid before remotely asking for the parking session to start.

Another important feature which is not present in the first version of the application (because it is currently at the mock-up stage) is a Wizard which guides the user to the first use of the application by requesting the minimum required data for the first transaction to take place.



Location
-

Car
Beatle (SC53PYU)

Payment method
Paypal

Leaving time
13:46

58minutes

City you are parked in
Chicago

Street address you are parked in
809 S. Damen Avenue

✓ Enable location identification

Choose the car you are parking from the list
Beatle (SC53PYU)

Your car settings

Delete a car from your list
[select]

When are you leaving?
12 45
13 : 46
14 47

Figure 8: Screen-shots of the user application on Android 4.2

Since it is a Wizard procedure, each picture in Figure 9 represents one screen, and they are presented sequentially to the user as soon as the information required previously is available.

As soon as the application is launched, the location services are started to determine the pricing rules, so that the waiting time will be lower (or even absent) based on how long it takes for the user to finish the Wizard procedure. The first picture shows the launch screen, where the user is given the chance to either use the Wizard or skip to the home screen previously presented. If the Wizard is chosen, the second step of the Wizard is shown, where the user is invited to log in using one of the accounts available in the system. Compared to the other login this is much quicker, because the user only has to pick an account, as opposed to typing the email address and a password. At the same time this gives all information required for both the authentication and authorization. It is also important to point out that this is possible on Android based devices, while on others this might not always be an option. If it is not, a simple login screen will be presented, just like the one which is offered in the current version of the application. Also, if the account chosen is already in the system then the Wizard procedure exits immediately and the home screen is presented, because all other information should already be available on the server and therefore downloadable by the client application.

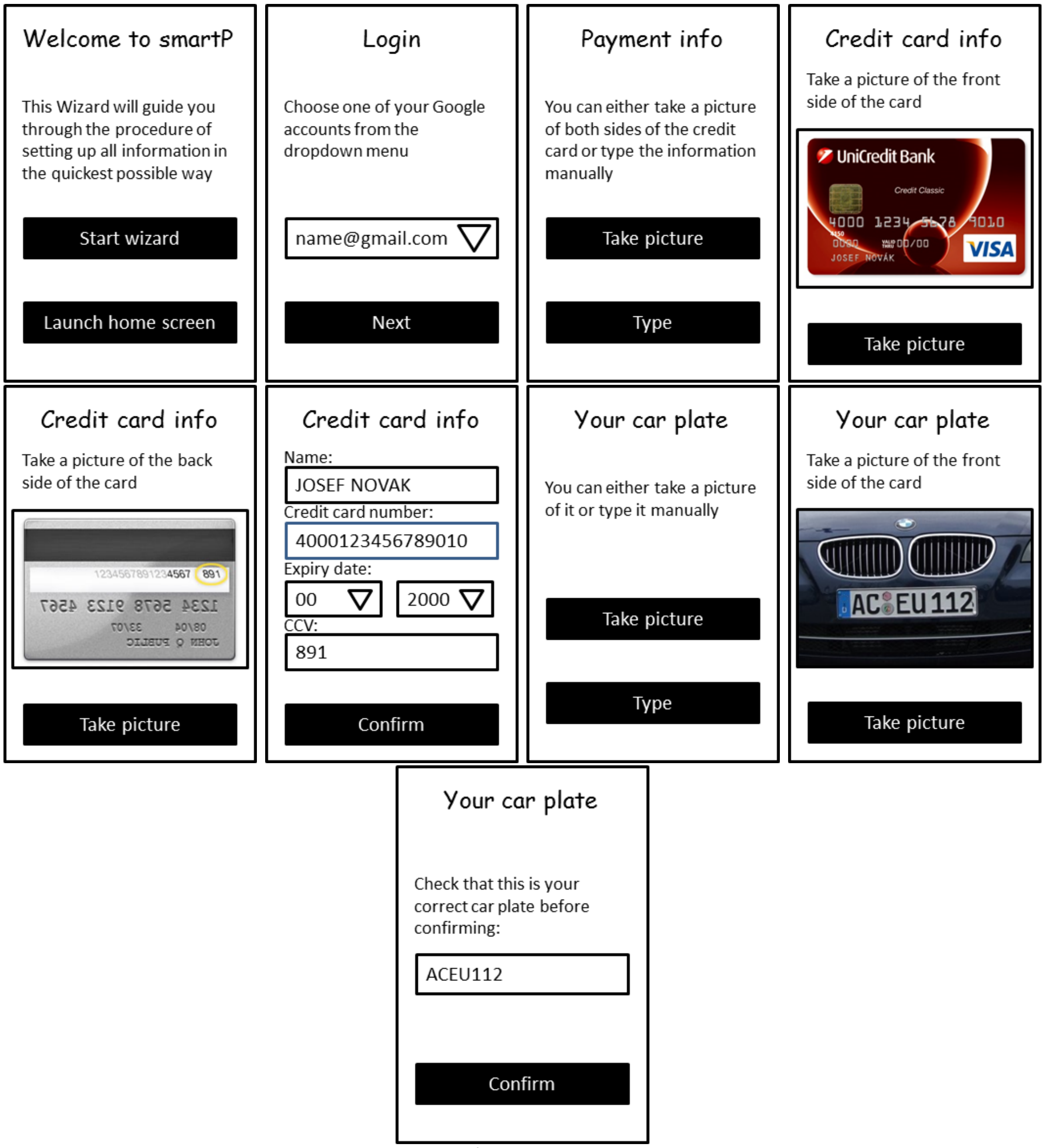
After providing login information the user is asked to give credit card information by either typing them or taking pictures of the credit card. If the user decides to manually type this information, the Wizard skips steps 4 and 5, and it displays an empty form. Otherwise in step 4 the user takes a picture at the first side of the credit card, while in step 5 s/he takes a picture at the back side of it. Finally the user is still led to step 6, where all fields are prefilled based on the information gathered form the pictures.

As soon as the user confirms the information about the credit card, the next step about the car plate is presented. In order to have the car plate number two ways are offered: either by typing or by taking a picture. Again, as for the credit card information, if the user decides to type the number the final step of the Wizard is presented, with an empty field. Otherwise the user is asked to take a picture of the car plate and then led to the same page, where the field is prefilled with the value read from the car plate. As soon as the user confirms the home screen appears, and all information should be ready to start a parking session (assuming that in the mean time the location sensors were able to determine the pricing and rules for the area where the user is parked). This Wizard is offered any time the application is not logged into any account.

4.3 smartP back-end

The smartP back-end is the server side application which connects the user (through any application) with the local parking companies. The responsibility for this component is entirely on smartP, and therefore a series of requirements are in place to make sure that the uptime, response time and all safety objectives are met. Modularity is also important, because new components might interact with this module and, for this reason, all communication between the smartP back-end and the other modules happens through a defined set of APIs. An example of how important this is pertains to the user application: as of now the Android version is available but, as soon as possible in terms of development time, other applications for other devices using different operating systems will be developed. This new addition should not require any changes on the smartP back-end, which offers the same API for all devices.

A series of decisions was taken on how to build this module in order to best satisfy the initial restrictions. Considering the limited development resources and the strong focus on the core business of managing parking spaces both for companies/cities and for final users, the choice was to opt for a cloud solution, and more specifically a Platform As A Service one. The motivations are several: with such a solution the scalability issues, uptime and availability, response time, backup and system architecture should not be addressed explicitly by us, because the system takes care of all of these aspects. On the other hand the pricing is not the best (as compared to other solutions where some of these aspects are not taken care of) and the code is in part more complex, because of some limitations imposed by the platform. The advantage is that, even if



Welcome to smartP

This Wizard will guide you through the procedure of setting up all information in the quickest possible way

Login

Choose one of your Google accounts from the dropdown menu

name@gmail.com

Payment info

You can either take a picture of both sides of the credit card or type the information manually

Credit card info

Take a picture of the front side of the card

Credit card info

Take a picture of the back side of the card

Credit card info

Name: JOSEF NOVAK

Credit card number: 4000123456789010

Expiry date: 00 2000

CCV: 891

Your car plate

You can either take a picture of it or type it manually

Your car plate

Take a picture of the front side of the card

Your car plate

Check that this is your correct car plate before confirming:

ACEU112

Figure 9: Mock-up of the initial Wizard procedure

more effort is required to write the code for these cloud solutions, this additional effort allows to manage all of those constraints and requirements without having to deal with them directly.

Another goal was to decide for a solution which could be easily moved from one platform to another with minimum to no change to the code. For all of these reasons the final decision was to develop the smartP back-end to work on the Google App Engine, which is the Google Platform As A Service offering. All that is required is to provide the code (in either Java, Python or Go languages) and all static files required for the application to run. The platform takes care of the deployment, scalability, positioning on the servers, content distribution, database set-up and everything which would normally have to take place if the service was not managed.

Because of the limitations of App Engine and the ones due to the initial requirements (such as compatibility with other cloud offerings) the decision was to go for Java as a programming language and therefore Java EE as a server side framework. The framework was a forced choice, in fact the use of the Spring framework was considered but, due to the fact that it was only partly supported by the App Engine the decision was to choose Java EE, which instead is the recommended choice. Moreover another strong limitation was the database. An SQL based solution has just been introduced for App Engine, but the limitations in terms of size of the database go against the scalability requirements. For this reason the decision was to use the DataStore, which is the no-SQL database of App Engine. This one has a lot of limitations, plus it is proprietary. In order to be able to move to other solutions JDO is used to access the DataStore: this abstracts the DataStore low level APIs with a standard set of APIs supported by other databases as well. Moreover there is an open source project[27] which aims at offering the Google App Engine platform on private clouds. This would be another option to move the smartP back-end to a privately managed cloud. Finally the use of JDO is important because it makes the code general and not constrained to the platform in any way. In order to support the largest amount of databases possible, the DataStore is used only as a key-value store. More advanced features, such as the join clause in a SQL database, are implemented at an application level. For example each user can have one or more cars. The link between each user and the cars associated to his/her account is only at an application level, where for example methods exist to retrieve all cars of a user or add a car to an account. The following code snippet shows a method to retrieve a list with all cars of a user, and as such it is a method of the UserProfile.

```
public ArrayList<Car> getCars() {
    ArrayList<Car> res = new ArrayList<Car>();
    PersistenceManager pm = PMF.get().getPersistenceManager();
    try {
        for(String plate: cars) {
            res.add(pm.getObjectById (Car.class, ↙
                ↳ Car.generateIndex(email, plate)));
        }
    } finally {
        pm.close();
    }
    return res;
}
```

Figure 10 presents an overview of the smartP back-end and its interactions with the other modules. The cloud itself is made of different components, but they are not particularly important as they are on the cloud and for this reason the physical details are not strongly linked with either the code or the decisions made. In particular the DataStore only offers eventual consistency, which is a serious issue but this problem is solved by the addition of a cache layer based on MemCache. More specifically all read and write requests are saved in the MemCache both for faster retrieval and to make sure read operations happen on the most recent version of the data. The cloud is an abstraction also of the servers, because there is no fixed number of deployed servers for the application (in the sense that it can be specified but it is not com-

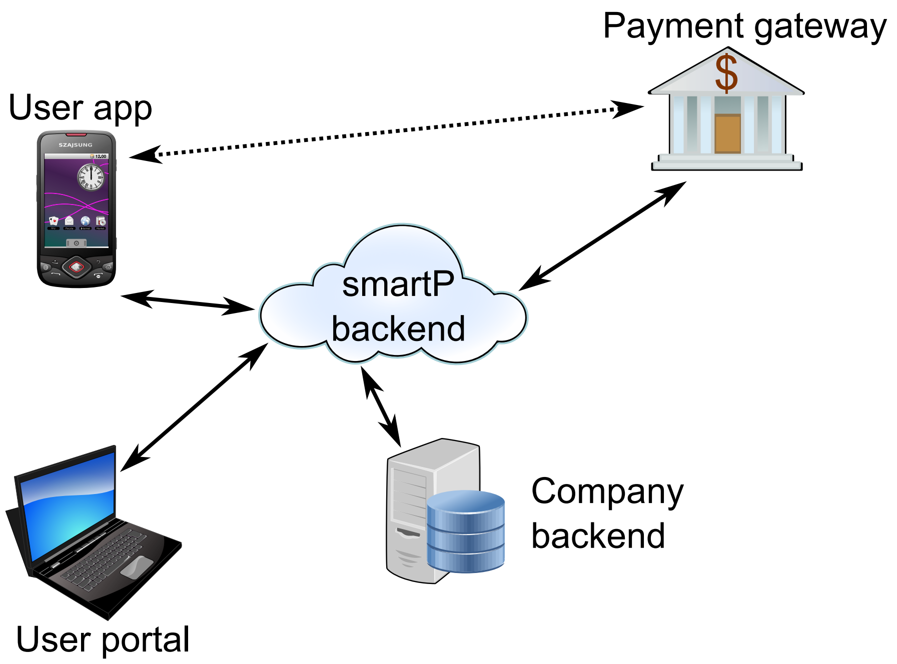


Figure 10: Overview of the smartP back-end and the interactions with the other modules, plus payment connections

pulsory) and some services (like sending emails) are performed asynchronously, and potentially on separate physical machines.

The connections between the user application and the smartP back-end has been partially described in Section 4.2, and they happen through HTTPS requests in the JSON syntax. Not being able to trust the client side user application, which is not under control and could be potentially substituted by other clients, all relevant data is stored on the smartP back-end. This is important because the user application has a local copy of data (e.g. the list of cars associated to the user who is logged into the account) but this information might not be up to date with other devices the user might have. For this reason each time the application connects to the smartP back-end it receives the updated data about the user's account. A more serious consideration is that user authentication has to happen on the back-end before allowing access to any service: in fact client side validation, again, would not be reliable. For this reason API requests are supposed to be self-sufficient, so that each request has all the information required to get a response and, if this is not true (because some information is wrong or missing) an error is returned to the client side. As of now there is an instantaneous sync of data between the user application and the remote server, while in the future some sync operations will be performed asynchronously, because in the rare event of a failure no real damage would happen. For example in the future there would be an asynchronous request to add a new car to an account. In reality, these calls are already asynchronous, but the interface is put in a waiting state so that the user waits for the response. In the future the user will be able to perform other tasks, thus making the application apparently faster to respond. The configuration of the services offered by the smartP back-end is described in a configuration file. For each servlet, code similar to this is present:

```
<servlet>
    <servlet-name>loginServlet</servlet-name>
    <servlet-class> com.smartp.server.server.LoginServiceImpl
        </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/api/login</url-pattern>
</servlet-mapping>
```

The first part is used to communicate that the class is in fact a servlet which should be used to rely to requests, while the second one bings the servlet to the URL syntax where this API service should be found

The communication with the user portal is secondary, as the user portal is only one and only stored on the server, therefore it can be changed without having to update anything other than the deployed version of the smartP back-end. Again, as of now the only available feature is the account refill, and for this reason there is no real point in commenting more about this module.

Connections with the company back-end are done through JSON over HTTPS requests. This is important to allow for integration with different systems. More specifically in Section 4.5 our proposed solution is presented, but the idea is that other solutions should be integrated into the smartP back-end so that they become supported and people can use the service with the companies already owning alternative solutions. The integration is not the main focus right now, because some of the existing solutions do not offer APIs to integrate with them. On the other hand, the separation between smartP back-end and company back-end also has this advantage: alternative solutions would require minimal code changes to be integrated and connected to the smartP back-end. Additional information about the communication between the smartP back-end and the company back-end is presented in Section 4.5.

The last important topic to be discussed in the smartP back-end section is of course about payments. To begin with, the decision was to make the user application as easy to use as possible, and some sort of payment gateway has to be involved at some level. Two main solutions remain the only options and are presented here. Of course only one is finally chosen, but the idea is that switching to the other one should be possible without significantly changing the overall system. Both proposed solutions involve credit cards as a payment method.

The first option would be to collect the credit card data directly from the user application in the phone (or the user portal) to collect them on the smartP back-end and finally store them for future use. This would be the ideal solution, because it would allow the creation of the Wizard procedure in its entirety, including the use of the camera to detect the credit card information. After the data is collected, when the user application requests any payments, it is the smartP back-end responsibility to connect to the gateway and communicate all information pertaining to the credit card transaction. This is usually called a server-to-server transaction, because it happens between the two servers without involving a client (in this case the user application), and due to current rules it requires the companies to have the system certified to be PCI compliant, according to the Security Standards Council. This requirement rules out this option, which would be the best for final users, both because of time required to achieve this and in terms of costs, and responsibility. Costs are usually prohibitive for most businesses, moreover the security is to be taken care of, and only recently has there been some documentation about compliance on the cloud[28], which is where the smartP back-end is being deployed. Finally, this solution puts the responsibility of keeping the data stored safely on smartP as opposed to only giving the responsibility to the payment gateway.

For these reasons the adopted solution had to be different, and more specifically a solution which would use the payment gateway as the sole responsible of the management of credit card information. These solutions usually work by offering a virtual POS, made of a web service where the user is asked to fill in the required fields for a web-based credit card transaction to take place. Since one of the requirements was that the user would not have to type this information every time a purchase took place, the payment gateway has to offer a service where smartP can ask for an authorization to charge in the future directly, so that after the initial procedure where the user has to fill in the data, later requests can be done directly by the smartP back-end to the payment gateway, and this does not require any level of PCI compliance. Examples of companies offering a service which meets these requirements are X-Pay by CartaSi and Paypal; the initial decision was to use Paypal, because of the better awareness about the brand by users. The procedure works by presenting the authorization page (from the payment gateway server) directly to the user inside the application. As soon as authorization is confirmed future payments can happen automatically through the application. In Figure 10 the dotted line is exactly the initial transaction, while the following ones happen through the connection between the smartP back-end and the payment gateway.

There is another solution which would solve all of the issues but is currently being studied and discussed with a potential payment gateway, so it could not be implemented in a limited amount of time. The idea is that, if the gateway allows for prefilling the form of the first payment authorization, the user application would offer the user to take pictures of the credit card and load the authorization step with the collected data, so that basically this would still be on the gateway server, but it would not require the user to type everything (unless of course s/he decides to. This solution requires an agreement with the payment gateway and a certain level of security of the user application, which would then have to temporarily keep credit card information in memory. Ultimately this would be the best compromise, which is why it is still being considered.

4.4 Inspector application

As of now the inspector application has only one feature: to allow for ticket inspectors to check whether a parked car is legal (and therefore there is a corresponding parking session in the company back-end) or, if not, whether it expired or it was never started. For this reason as of now the application only offers login functionality (which is to make sure that the user is a valid inspector) and check functionality, both by taking a picture of the car and by typing the value.

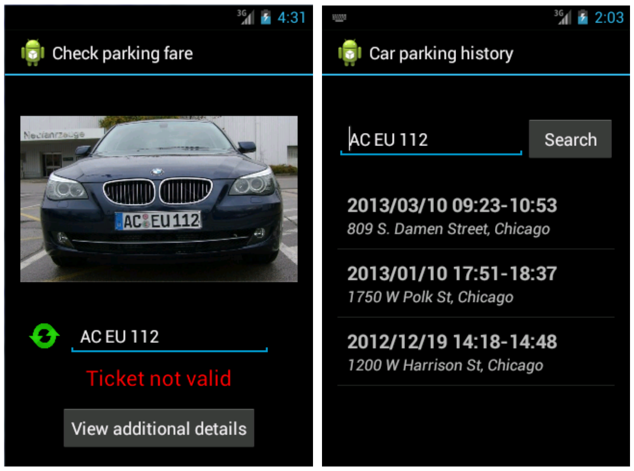
The number plate recognition is accomplished by the use of a third party library, because there is no point in re-implementing it from scratch. Due to limited hardware resources the search for a plate is only performed on a static picture and not on all frames filmed by the camera. As always the recognition is not perfect, and the accuracy has not been formally assessed, but it is ultimately the inspector's responsibility to verify that the plate has been correctly identified. Since for the inspector application the correctness of the plate at the first time is not as important as for the user application (in the sense that the user cannot change it later, while the inspector can try and identify a different plate number if the automatically detected one fails), the procedure does not involve two stages where in the first one the picture is taken and in the second one the plate number is presented for the inspector to confirm or edit it. Instead, as soon as the plate number has been identified using the camera, the application starts looking for a valid session. The inspector will be presented the details and of course a chance to edit the plate number and start a new search.

The interaction of this application with the external World is limited to a few sensors and a connection to the company back-end, which is the exclusive connection to the Internet. In terms of sensors, the aforementioned camera is used to detect whether the user is paying for a given car, through the car plate. The location sensors are used to determine the location where the car is being parked, to determine whether the parking session is valid also in terms of location for which the user is paying. As far as the Internet connections are concerned, the company back-end is the only reliable source of this information on the company side and for this reason it is the only one providing information to the final user.

The distribution of this application will be through an executable file (APK file for Android devices), because it should not be available through a store such as the Google Play to the general public. This means that future updates do not happen through the store but manually, which is important for the parking companies which want to constantly control and make sure that they can check the honest behavior of the smartP back-end.

The initial version of the inspector application only offers the essential features, as shown in Figure 11. The use is as simple as possible. The login screen is not reported here, because it is the same as the one for the final user application (in terms of GUI, because on the other hand all connections happen with the company back-end, and therefore a valid account on either of the applications has nothing to do with the other application).

The first screen-shot (on the left) represents the main page, where the inspector can select a car plate. The camera can be used to read the plate number, and every time the green


4:31
Check parking fare
AC EU 112
Ticket not valid
View additional details

2:03
Car parking history
AC EU 112
Search
2013/03/10 09:23-10:53
809 S. Damen Street, Chicago
2013/01/10 17:51-18:37
1750 W Polk St, Chicago
2012/12/19 14:18-14:48
1200 W Harrison St, Chicago

Figure 11: Screen-shots of the inspector application

refresh button is pressed the current frame is analyzed. The text field is then prefilled, while a remote search is performed on the shown value. In the example above the ticket is not valid, which means that the user is not paying for parking through smartP (nor any other competitor interacting through the parking company back-end). If the inspector realizes that the application misread the plate number, s/he can manually edit it in the provided field and the result is refreshed with the new value.

The second screen-shot (on the right) is opened when the inspector taps on *View additional details* of the first one: This provides the latest parking sessions for a specific car plate, so that the inspector can actually make sure that a fine should be generated. In this example it is 4:31 P.M., but the latest valid parking session for this car was until 10:53 A.M. and therefore a fine should be generated (assuming of course the there is no right for the user to park for free, e.g. because the parking spot is a free one or because it is a holiday).

A similar mechanism can be used for numbered parking spaces: the number of the parking space can be used to determine if the fare is being paid, and a history of the payments for the specific spot the inspector is interested in.

The following code snippet provides an example of the use of XML to define the GUI of Android activities. This is the layout for the activity presenting the seach results by car plate. As shown in the code, this only represents the structure of the activity, while strings are defined in a separate file (which supports multiple languages) and the real contents are filled by the corresponding Java code for the same activity.

```
<RelativeLayout
    xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:paddingBottom = "@dimen/activity_vertical_margin"
    android:paddingLeft = "@dimen/activity_horizontal_margin"
    android:paddingRight = "@dimen/activity_horizontal_margin"
    android:paddingTop = "@dimen/activity_vertical_margin"
    tools:context = ".PlateDetailsActivity" >

    <LinearLayout
        android:id = "@+id/mainAreaPlate"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_alignParentTop = "true"
        android:layout_centerHorizontal = "true"
        android:layout_marginTop = "26dp" >
        <EditText
            android:id = "@+id/plateFieldDetails"
            android:layout_width = "195dp"
            android:layout_height = "wrap_content"
            android:inputType = "text" />
        <Button
            android:id = "@+id/button1"
            android:layout_width = "94dp"
            android:layout_height = "wrap_content"
            android:layout_marginRight = "43dp"
            android:text = "@string/search" />
    </LinearLayout>
    <ListView
        android:id = "@+id/listView7"
        android:layout_width = "match_parent"
        android:layout_height = "0dp"
        android:layout_centerHorizontal = "true"
        android:layout_marginTop = "20dp"
        android:layout_below = "@+id/mainAreaPlate"
        android:layout_alignParentBottom = "true" >
    </ListView>
</RelativeLayout>
```

4.5 Company back-end

The company back-end is a piece of software which allows the management web portal to be used to set up parking rules and to retrieve information about paid parking sessions. It also creates a bridge between the smartP back-end, which knows about the new parking sessions being created (or start-stop parking sessions being stopped), and the inspector application, which needs to retrieve this type of information. There are no other interactions, between the company back-end and the Internet.

Before describing the choices made with this module, some information will be presented on the background of the reason for separating this module from the smartP back-end: it would be easier to build just one back-end which has all the data without unuseful data duplicates and unnecessary Internet communications (plus some sort of APIs which allow for communications between the twos).

The first reasoning behind this decision is based on the requirement for companies to be able to make sure that any person who pays for parking is counted and that the amount of money which they get for payment by smartP equals to the amount corresponding to the actual transactions registered on the system. The reason why this system works is that, if a ticket inspector validates a car whose owner paid for parking but the smartP back-end did not communicate this information back to the company back-end, the result is a generation of a fine, but the driver can complain because s/he has the proof of the transaction (in the form of the confirmation PDF) and the scam would be discovered. On the other hand, by summing up the revenues generated by all transactions received by the company back-end, the company management can verify that they get paid the corresponding amount of money. This is not a perfect solution, but it is a solution which allows the company to check that everything is fine and they get paid fairly.

The second reasoning behind the decision to separate the two back-ends is to potentially offer competitors to be able to allow for parking with a specific company or city. This is important because the parking spaces are only owned or managed by a single company (either public or private), but there is no theoretical restriction on the number of official resellers for the parking service. Of course few companies will be interested in this aspect, because one of the biggest risks is of course the proliferation of fake applications which do not have a legal contract with the company but ask for money in exchange for supposed parking fares. The solution to this would be an endorsement by the parking company, but again this is a minor problem because it is unlikely that many companies will want to compete at the transaction selling level. On the other hand it is positive for a public company to be able to support competition if this does not come at an extra cost to the population.

Finally it is worth explaining why the solution of only offering one back-end at the company level is not a recommendable solution. Before describing this, most of the App-based solutions presented in the State of the Art are using this approach, which is one of the reasons why the adoption of these new solutions by drivers has not been significant in most locations yet. The fact is that drivers want and expect an interoperable application, which allows them to pay for parking in all the cities they visit. Having custom back-ends for each parking company would mean that the user would have to create an account before parking at each different company, and if the prepaid option is the only one available, they would have to refill their accounts, which are different company by company. A study by MIP (School of Management of the Politecnico di Milano) showed that more than 50% of the users of an application to pay for parking would want it to support at least one more city, where they currently cannot pay using this application[?].

Under these assumptions a proposed solution has been formulated. The company back-end is similar to the smartP back-end, in terms of programming language (Java), framework (Java EE), cloud solution (Google App Engine), and DataStore abstraction layer (JDO). The decision is based mainly on the fact that this approach would allow for many easy deployments and alternative offerings, and more specifically: the parking company can decide to trust smartP entirely, in which case they get a fully managed deployment, which means that they only have access to their back-end through the management portal and the inspector application. This first possible deployment does not allow the company to verify whether they are getting paid as much as they deserve, but the decoupling between the two back-ends still allows for competitors at the parking transaction level.

The second approach would be that the parking company deploys the back-end on Google App Engine, thus being the only one controlling it. This is enough for them to make sure that the only communication happening outside of the company devices is between their back-end and the smartP back-end, which means that the only requests being made comply with the standards and therefore assure the ability of the parking company to check on their earnings. This is a very convenient solution, because it grants the company the chance of controlling transactions while not requiring their internal management of the infrastructure. For this solution any updates to the back-end have to be deployed by the parking company.

The third approach is a deployment on the parking company servers. This solution offers even more control over the data compared to the second solution, and all the same advantages, plus: the company would know where the data is kept, which is a prerogative for privacy reasons in some countries, but is not guaranteed by the self-service version of Google App Engine, where the deployment is entirely up to Google in terms of location of the physical servers.On the other hand the disadvantages of this solution have to do with the additional costs of managing the infrastructure, which is owned by the parking company. Costs might be lower if some economies of scale are present, for example if the company already has an infrastructure to manage. The disadvantage of having to manually perform deployments of updates of the company back-end is still the company responsibility, as for the second approach.

None of these approaches is the best in all situations, so the best one for the specific company will be determined on a case by case manner. The key aspect is that the code of this module has been designed in a way that it can be used in all scenarios, as explained earlier.

In terms of offered functionality, the interactions with the inspector application have been described in Section 4.4, while the interaction with the management portal will be described in Section 4.6. The interactions between the two back-ends entail the following exchange of information: given a location the company back-end should be able to provide a set of parking rules for that location to the smartP back-end. The smartP back-end is able to request a parking session (and terminate it, if postpaid is allowed) and receive a confirmation on the correctness of the information of the session and the compliance with the local rules. Since the company can potentially support different payment companies as competitors of smartP, the company back-end keeps track of the company requesting a new parking session, whether it is smartP (through the smartP back-end) or some other company, as long as they all use the appropriate APIs.

4.6 Management web portal

The management web portal is an essential component of the system which allows the managers of the parking company to access the system (for all back-office tasks they might need to perform). First of all, this component can be accessed through a website, accessible from most web browsers which support JavaScript. Due to differences between browsers there would have been a lot off effort required to build a solution which would support most web browsers, and for this reason the Google Web Toolkit was chosen. There are other advantages to using GWT, because the client-side code is written in Java and then compiled in JavaScript by the Google Web Toolkit. For example there is an abstraction over the differences between the web browsers, which are taken care of by the toolkit: different versions of the portal are generated so that each browser represents it and makes it behave the way that it was meant to. Moreover, the fact that the code is Java allows for the same debugging a Java application offers, while writing in JavaScript directly would mean that different tools would have to be used for debugging. Here is an example of a Java class which is turned to JavaScript by the Google Web Toolkit, and more specifically this is the feature to check the status of rules in the past, depicted in Figure 14.

```
package com.smartp.companyBackend.client;

import com.google.gwt.i18n.client.DateTimeFormat;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.datepicker.client.DateBox;

public class PastStatusView extends VerticalPanel {
    public PastStatusView() {
        this.setWidth("100%");
        VerticalPanel verticalPanel = new VerticalPanel();
        verticalPanel.setStyleName ("manyElems");
        add(verticalPanel);
        HorizontalPanel horizontalPanel = new HorizontalPanel();
        verticalPanel.add(horizontalPanel);
        horizontalPanel.setWidth("100%");
        Label lblAddress = new Label("Address");
        horizontalPanel.add(lblAddress);
        TextBox textBox = new TextBox();
        horizontalPanel.add(textBox);
        Label lblCity = new Label("City");
        horizontalPanel.add(lblCity);
        TextBox textBox1 = new TextBox();
        horizontalPanel.add(textBox1);

        Label lblDate = new Label("Date");
        horizontalPanel.add(lblDate);
        DateBox dateBox = new DateBox();
        horizontalPanel.add(dateBox);
        dateBox.setFormat(new DateBox.DefaultFormat
            (DateTimeFormat.getFormat("yyyy/MM/dd")));
        Label lblTime = new Label("Time");
        horizontalPanel.add(lblTime);
        TextBox textBox2 = new TextBox();
        horizontalPanel.add(textBox2);
        horizontalPanel.setStylePrimaryName(
            "gwt-TabLayoutPanelContent");
        Button btnNewButton = new Button("Search");
        verticalPanel.add(btnNewButton);
        HorizontalPanel panRes = new HorizontalPanel();
        panRes.setStylePrimaryName("ruleRes");
        Label lblPrice = new Label("Cost per hour:");
        panRes.add(lblPrice);
        Label lblRes = new Label();
        Calculate cal = new Calculate(lblRes);
        btnNewButton.addClickHandler(cal);
        panRes.add(lblRes);
        this.add(panRes);
    }
}
```

In terms of deployment this web portal is in the company back-end, where it is served from. This means that companies which opted for the fully managed version do not need to take care of anything for their management portal to be upgraded to the newest version, while the others have their portal upgraded by upgrading their back-end deployment. On the client side there is no need to install any specific software, which is another good reason to opt for this solution, because the update of the web portal does not require any action on the devices used to access the portal. Of course this portal only communicates to the management back-end, both because it is the only point which has all the required information and because it is the only location where information can be assumed to be trustworthy by the parking company.

In terms of security the client side code cannot be trusted, therefore each request is treated as potentially dangerous by the back-end. The reason why client side scripting is being used in the management portal is to offer a better experience to the user (manager): for example the risk of losing unsaved changes can be mitigated with features like auto-save, loading times of various pages can be reduced with content preloading, and changes can be applied without having to refresh the web page.

The features initially offered by the first version of the portal are the essential ones for a company to make use of the service: this is in compliance with the previously defined Minimum Viable Product. To begin with, login functionality is essential, because of the powers which follow from having access to the portal. For this reason a user name and a password are required to access the service. A session is kept alive for up to thirty minutes, after which the manager will have to perform the login procedure again.

Moreover one of the two available essential features is the ability to check the session history by car plate, either in its entirety or in a specified period of time. All results pertaining to the search are presented in a list. This feature also allows to check if a driver had the right to park for free at a specific location and at a specific time and day, according to the rules which applied at the time. Of course this cannot rule out all cases of having the right to park for free, but only the ones contemplated by the rules set in the system.

Finally the last feature which is offered through the management portal is of course the ability for the management to set rules which apply to their parking spaces. Different approaches were possible, and the chosen solution was to offer the easiest one to understand by the people who would have to set these rules. The chosen solution is for each rule to have three pieces of information:

1. What: this part of the rule describes the actual reason to specify the rule, e.g. to allow or forbid parking sessions of the start/stop type, or to set the hourly parking price, or to set the minimum session duration.
2. Where: this allows to select one or more areas. If there is a conflicting rule in one or more of the selected areas a message is presented and the manager is offered to either substitute the old rule with the new one or to reduce the zone of applicability of this rule by deselecting the areas with preexisting conflicting rules.
3. When: this field describes when this rule takes effect, e.g. only during the week ends or only from 8 A.M. to 6 P.M. This can of course be a specific date, or any range of time, once only or recurring.

After a new set of rules has been specified, a date for their initial application has to be specified: after this date and time the old rules are changed according to the new ones for drivers using the service. Of course all rules are kept in the system for future reference, but also to be able to offer the feature which allows managers to check what the rules were at a given time and date, and at a given location.

Figure 12 presents the login screen, where the management staff is supposed to use an account which is allowed to access these features. As of now all features in the management console are accessible to all management accounts, even though in the future it might make sense to create different management account types for different company functions, and only give access to the pertinent features.

Figure 13 shows an example search result using the feature which allows the management to check past payments of any driver who used smartP. As shown in the screen-shot, the start and end date are not compulsory: if present they will potentially filter out part of the results,

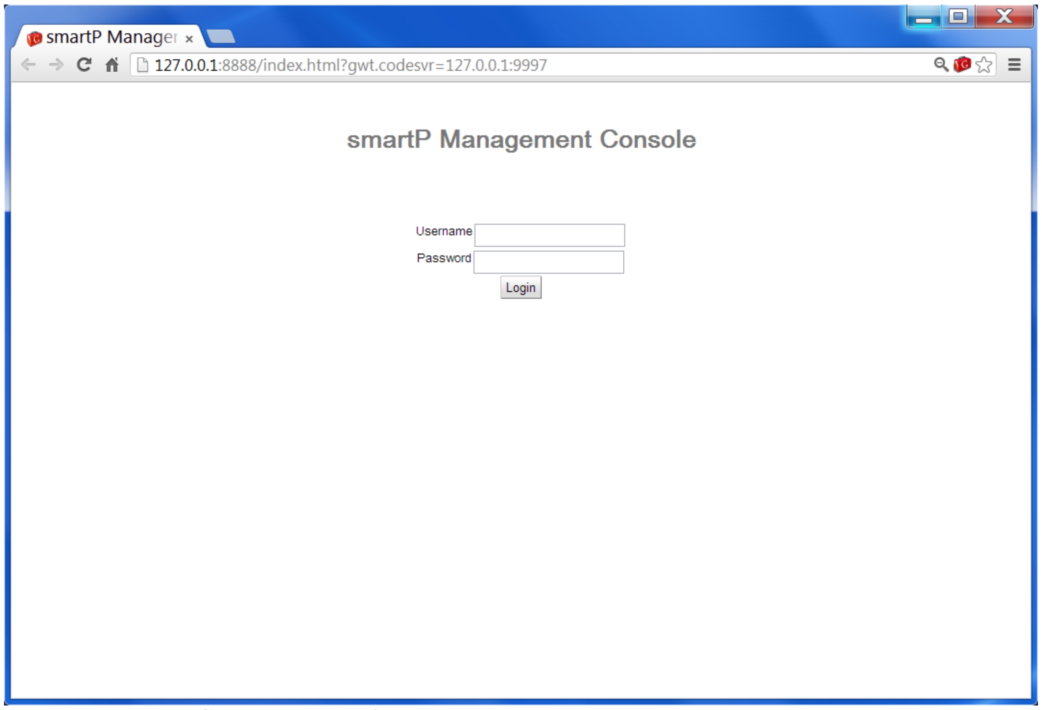

smartP Manager
127.0.0.1:8888/index.html?gwt.codesvr=127.0.0.1:9997
smartP Management Console
Username
Password
Login

Figure 12: Screen-shot of the login screen in the management web portal

but not being compulsory the management can perform searches over the entire time. The plate number field is on the other hand compulsory: this narrows down the search results significantly and it is not limiting to the management, because there are no use cases when they would want to search single past payments for more than one driver at the same time, and the only parameter which can uniquely identify the driver (or the car) is the car plate. For companies which use numbered parking spaces the compulsory field is the parking number and not the car plate. The results show arrival date and time, leaving date and time, location where the car was parked, value of the paid fare.

Figure 14 is useful to determine if a driver paid the right amount of money for a parking fare according to the rules set by the management; this feature can also be used to determine whether the driver had the right to park for free, according to these rules. This feature requires

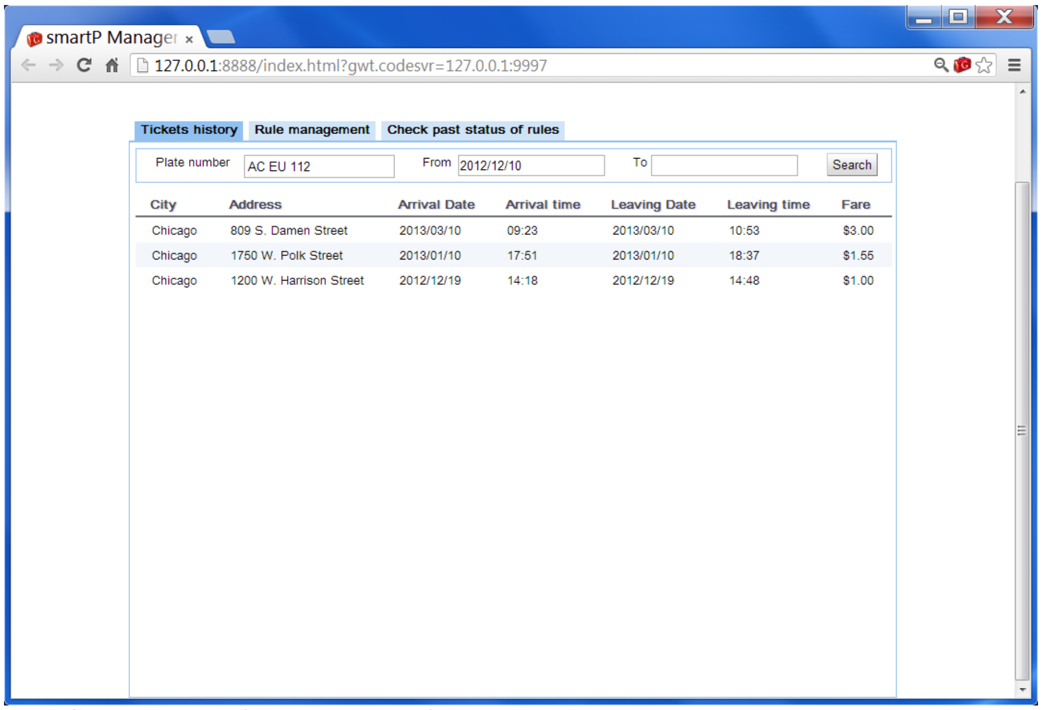
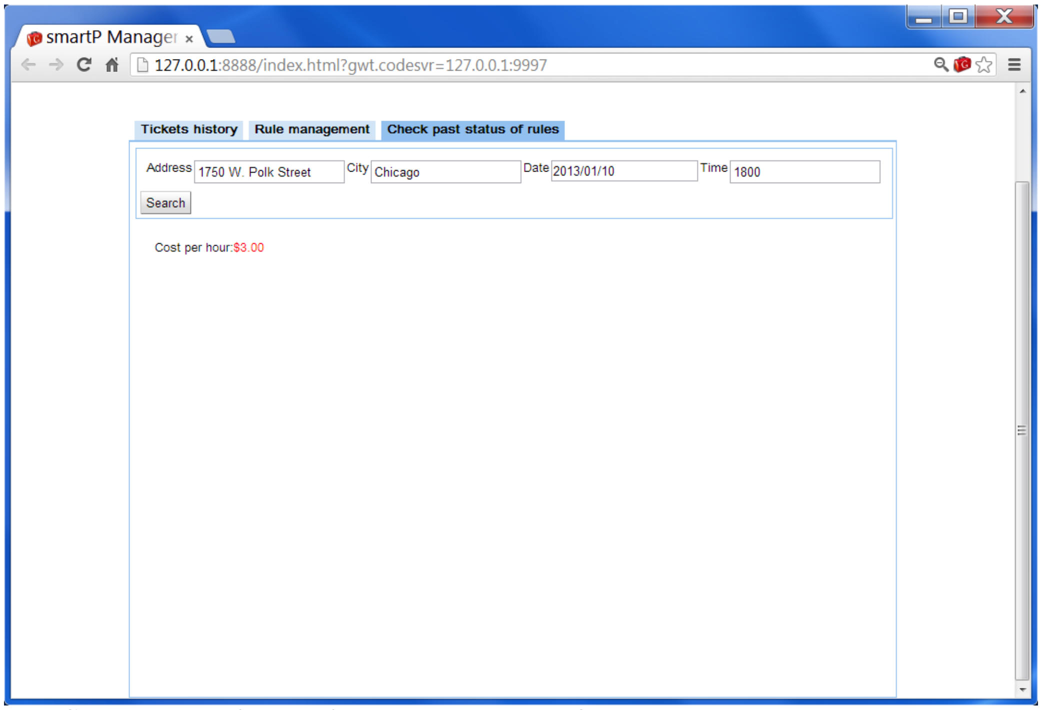

smartP Manager
127.0.0.1:8888/index.html?gwt.codesvr=127.0.0.1:9997
Tickets history
Rule management
Check past status of rules
Search

Plate number: AC EU 112 — From: 2012/12/10 — To:

City	Address	Arrival Date	Arrival time	Leaving Date	Leaving time	Fare
Chicago	809 S. Damen Street	2013/03/10	09:23	2013/03/10	10:53	$3.00
Chicago	1750 W. Polk Street	2013/01/10	17:51	2013/01/10	18:37	$1.55
Chicago	1200 W. Harrison Street	2012/12/19	14:18	2012/12/19	14:48	$1.00

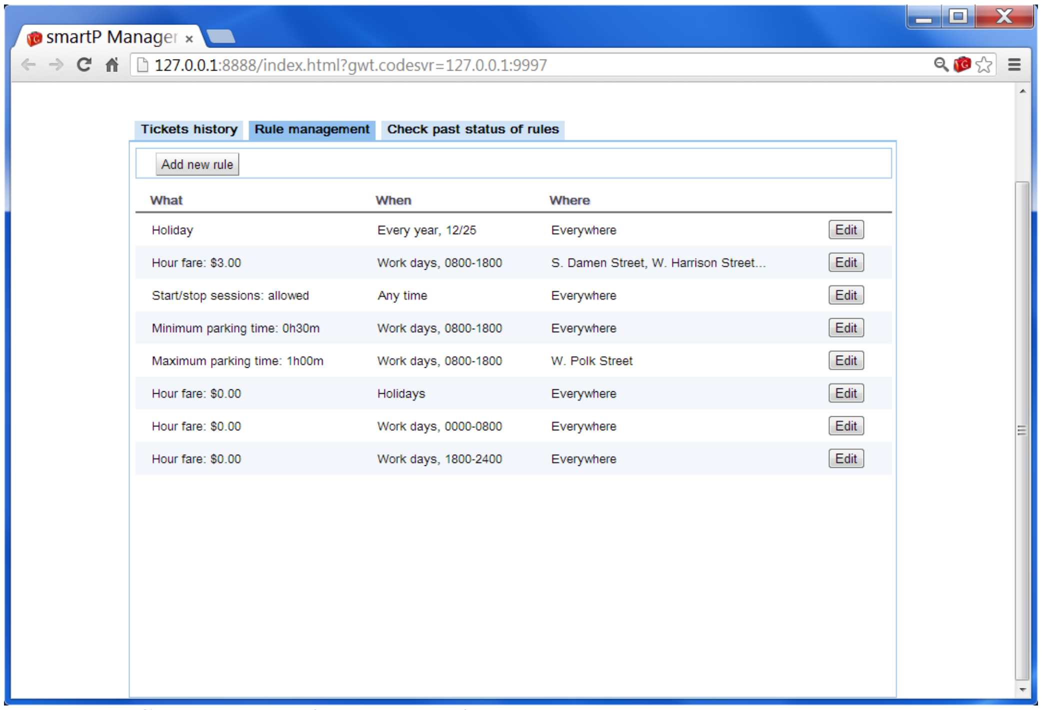
Figure 13: Screen-shot of the history of purchased tickets by a user, in the management web portal


smartP Manager
127.0.0.1:8888/index.html?gwt.codesvr=127.0.0.1:9997
Tickets history
Rule management
Check past status of rules
Search

Address 1750 W. Polk Street City Chicago Date 2013/01/10 Time 1800

Cost per hour:$3.00

Figure 14: Screen-shot of cost of parking at a specific time and location in the past, presented in the management web portal

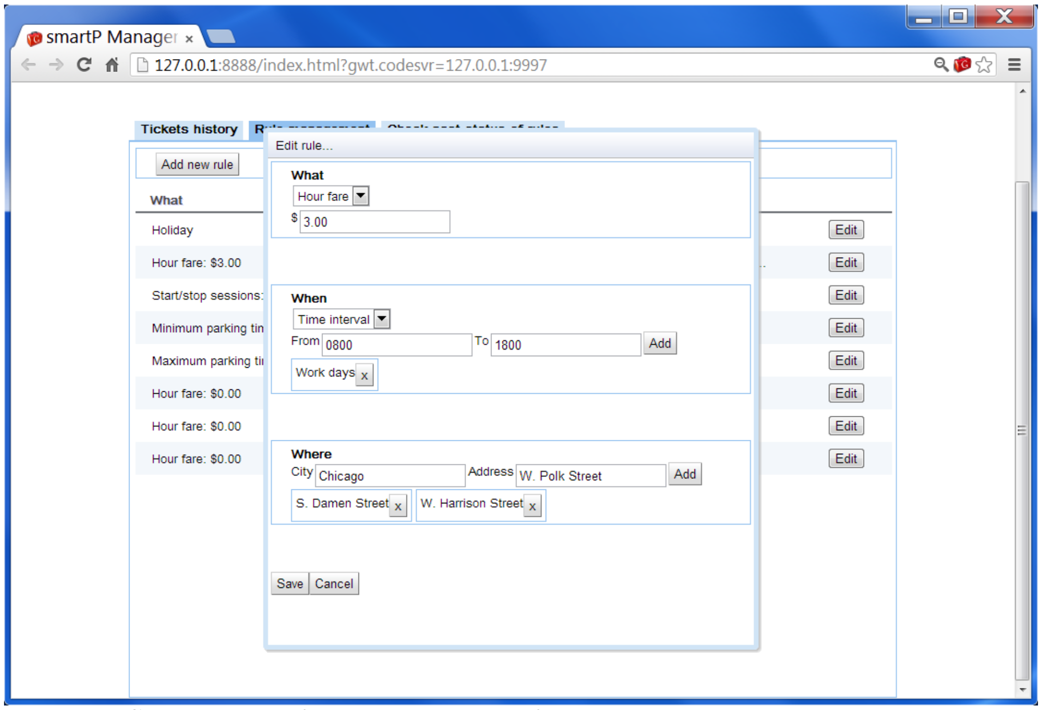

smartP Manager
127.0.0.1:8888/index.html?gwt.codesvr=127.0.0.1:9997
Tickets history
Rule management
Check past status of rules
Add new rule

What	When	Where	
Holiday	Every year, 12/25	Everywhere	Edit
Hour fare: $3.00	Work days, 0800-1800	S. Damen Street, W. Harrison Street...	Edit
Start/stop sessions: allowed	Any time	Everywhere	Edit
Minimum parking time: 0h30m	Work days, 0800-1800	Everywhere	Edit
Maximum parking time: 1h00m	Work days, 0800-1800	W. Polk Street	Edit
Hour fare: $0.00	Holidays	Everywhere	Edit
Hour fare: $0.00	Work days, 0000-0800	Everywhere	Edit
Hour fare: $0.00	Work days, 1800-2400	Everywhere	Edit

Figure 15: Screen-shot of the view of current rules, in the management web portal

a time and date, and a location. Based on this information and the rules present in the company back-end, the system calculates the fare amount and reports it to the management (in red if not free, in green if free). One explanatory situation is to make sure that a driver has to get fined: assuming that the application claimed (according to the company back-end) that the user had the right to park for free, while in fact the rules were wrong at the time and the driver should have paid a certain amount for his/her stay, this feature will notify the management that the driver respected the rules set by the management through the management console and thus should not be fined.

Figure 15 allows the management to have a general view of all currently enforced parking rules. The structure is the same as described earlier, and a tabular view provides information about the location, time and actual details of the rule. The “What” column provides



What
Holiday
Hour fare: $3.00
Start/stop sessions:
Minimum parking tin
Maximum parking ti
Hour fare: $0.00
Hour fare: $0.00
Hour fare: $0.00

What
Hour fare
$ 3.00

When
Time interval
From 0800 To 1800
Work days x

Where
City Chicago Address W. Polk Street
S. Damen Street x
W. Harrison Street x

Figure 16: Screen-shot of the setup screen for a rule, in the management web portal

information about the type of rule and any parameters which are required to specify a rule of that particular type: each rule type might have a custom number of additional details. The "When" column gives information about the time constraints of the rule it refers to. There are different types of time constraints, but additional information about this will be provided while describing the rule editor. The location, which is presented in the "Where" column, can be one or more areas, up to everywhere which means in all locations pertaining to the company.

Finally figure 16 allows the management to add new rules or edit existing ones. The "What" section provides a drop-down list of existing rule types. Based on the chosen rule, the rest of this section changes to provide enough input fields for all additional required or optional information (e.g. in the screen-shot a hour fare is being set; this only requires one field to provide the price; this could change in the future, if a company were to manage areas in multiple currencies: at that point an additional field for the currency would appear). The “When” section starts with a drop-down menu offering the management to choose the type of time constraint to add to the rule. Examples could be day of every week, work days, holidays, time interval, from date to date etc. In this example a time interval is added, and therefore two additional fields are presented: start time and end time. In the lower part of this box there is a list of all time constraints for the rule being edited. The management can delete any of these by clicking on the “x” button. The “Where” section is similar to the “When” section, in the sense that the management can pick one or more locations and they appear as a list in the lower part of this section. Again, areas can also be deleted from there. As of now the user can type the name of the city and the address in order to set a new location to add.

CHAPTER 5

PRELIMINARY RESULTS

The project is at a prototypal stage, and therefore some of the data which could be collected in a real World scenario are not available yet. Since, on the other hand, the system has already been built, a series of experimental validations can already be performed, and some data can be presented which sums up the obtained technical results of the proposed solution. In fact initial results are already promising, in terms of the performance of the system and its adherence to the set requirements. In Section 5.1 procedures will be presented for the various stakeholders to set-up their components of the system. In section 5.2 some details about the system performance will be introduced, and these results will also be compared to other websites to assess them and show what the achievements actually mean.

5.1 Set-up procedures

For final users all that is required to start using smartP is to download the smartP final user application. Even if it is not available in any application store as of now, it will be part of the available applications in any store compatible with the platform it was developed for (initially all Android-based ones). Right after the driver installs the application s/he has to create his/her own account by providing an email address and a password (or log in, if the user had previously registered to the service with a different device). Either way this will suffice to start using smartP, because any additional information will be requested when required.

The management will have to deploy the company back-end as a first step. If they decide to opt for the fully managed option, they can entirely skip this step. If, on the other hand, they do want to manage their own back-end, the procedure can still be straightforward. The first option, which is to deploy on the Google App Engine, only requires them to upload their back-end (potentially after creating a Google account and providing payment information) through

Eclipse or the command line tool. The second option is to deploy on their own servers, and as of now this is only supported through AppScale. This procedure requires a significantly higher amount of time but, if it turns out to be the preferred one for companies, it will be improved in the future. Currently this requires AppScale to be installed and configured, and therefore a database has to be set up on the system beforehand. After the company back-end is deployed the management will have to access the management console, log in for the first time and generate all the rules necessary to correctly manage all of their car parks. Once these steps are accomplished the set-up on the management side is complete.

Finally inspectors will find the application already installed on their devices, and all they will have to do is log in with their account before they can start using the application. After they log in their set-up procedure is therefore complete. On the other hand the deployment of the application can be performed by the management, or inspectors can be provided with devices which have it preinstalled, directly by smartP.

5.2 System performance

To begin with, the smartP back-end performance is assessed. Since this component is deployed on the Google App Engine, the results are mainly due to the App Engine architecture more than the code itself. On the code itself speed was considered when making use of storage, and more specifically the DataStore: all entities are independent to maximize parallelism and concurrency. The results provided here assume that there is only one live instance of the back-end, but the positive aspect is that the App Engine scales automatically and allocates more instances as it sees fit (e.g. in the event of a burst in user requests). After a certain amount of time, instances which are not being used are dismissed, all the way to having zero instances in the cloud. If this is the situation, when the first request arrives a new instance has to be loaded before a response can be sent, and this takes around 11 s [1], but this can be avoided both by

[1]The exact average of the last 10 times this occurred is 11.126s

paying for an instance to always be live and by making sure that the expiration time does not pass without a new request taking place.

On the other hand, times for required on average for a response are much shorter. As an example the log-in requires 270ms while the car management requires 239 ms [1]. These smartP back-end times actually affect the final user application, because it is the one which makes these requests to the back-end. A final remark[2] for the smartP back-end is shown in Table II: the first column represents the location where the Ping test was sent from, the second one shows the IP address which the smartP URL resolved to, the third column shows the minimum Round trip time (RTT) and the third column shows the maximum RTT; finally the fourth column represents the response time. All times are expressed in milliseconds. It is clear how the use of the Google App Engine is beneficial in terms of performance: users from different locations are routed through different paths to different servers, in order to minimize response time from any location.

The final user app requires around 800ms to load initially (this means if it was not loaded previously), after which time the log-in procedure can start. This procedure can take a variable amount of time, based on when was the last time the user accessed the app. Either way, all future operations which take any time significant to a person are related to the management console, because the bottleneck is the web service and Internet connection.

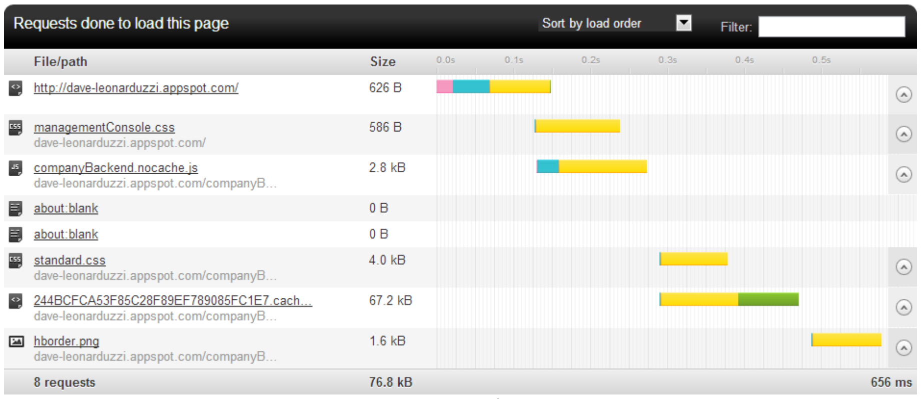
Finally the smartP back-end is considered. All the results provided here are based on the company back-end being deployed on Google App Engine, but different deployments might offer different performances and therefore results. the focus, on the other hand, is put on the code here, and more specifically on the use of the GWT. Because of how GWT works, the application

[1]Both measurements are an average of the response times of the last 10 requests. These measurements account for the fact that the device was connected to the Internet through a mobile Internet connection, even if no information is available on whether 3G was available at the time of the requests or not.

[2]The data which lead to this remark is obtained through the site24x7 service at https://www.site24x7.com/ping-test.html

TABLE II: PING FROM VARIOUS GEOGRAPHIC LOCATIONS TO THE SMARTP BACK-END

Location	IP	MinRTT	MaxRTT	Resp.
California	74.125.28.141	24.011	29.224	25
Seattle	74.125.129.141	21.859	27.912	23
Dallas	173.194.64.141	7.636	7.698	7
Texas	173.194.64.141	7.377	7.422	7
Chicago	173.194.77.141	26.64	26.808	26
New Jersey	173.194.73.141	18.218	18.807	18
Virginia	173.194.74.141	9.434	10.061	9
Los Angeles	74.125.129.141	30.287	30.519	30
New York	173.194.75.141	16.927	17.515	17
Denver	173.194.77.141	28.454	28.513	28
Washington	173.194.77.141	16.342	16.997	16
Montreal, Canada	173.194.68.141	24.045	24.158	24
Phoenix	74.125.141.141	39.454	39.645	39
Atlanta	173.194.77.141	26.685	29.305	28
Toronto, Canada	74.125.142.141	25.514	26.124	25
Sao Paulo, Brazil	74.125.140.141	123.901	124.420	124
London	74.125.136.141	11.448	11.480	11
Cologne, Germany	173.194.65.141	11.412	12.126	11
Sweden	173.194.71.141	9.812	9.842	9
Rio de Janeiro, Brazil	74.125.137.141	127.529	127.826	127
France	173.194.66.141	6.063	6.115	6
Spain	173.194.66.141	38.822	39.136	38
Italy	173.194.67.141	21.316	21.594	21
Munich, Germany	173.194.65.141	11.379	11.508	11
Belgium	74.125.132.141	16.656	16.791	16
Denmark	173.194.66.141	25.894	41.247	35
Nagano, Japan	74.125.31.141	37.109	37.364	37
Singapore	74.125.135.141	8.524	10.371	9
Hong Kong	74.125.128.141	2.905	3.296	3



Requests done to load this page

Sort by load order

Filter:

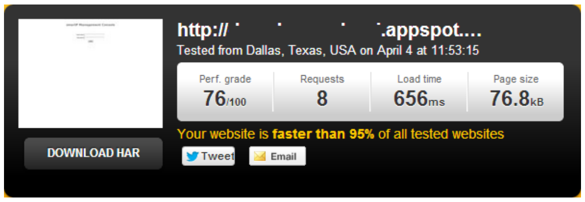
File/path	Size
http://dave-leonarduzzi.appspot.com/	626 B
managementConsole.css dave-leonarduzzi.appspot.com/	586 B
companyBackend.nocache.js dave-leonarduzzi.appspot.com/companyB...	2.8 kB
about:blank	0 B
about:blank	0 B
standard.css dave-leonarduzzi.appspot.com/companyB...	4.0 kB
244BCFCA53F85C28F89EF789085FC1E7.cach... dave-leonarduzzi.appspot.com/companyB...	67.2 kB
hborder.png dave-leonarduzzi.appspot.com/companyB...	1.6 kB
8 requests	76.8 kB

656 ms

Figure 17: Loading time line of the management console

logic is moved to the client side in the web browser when first accessing the service. This does not mean that the client is considered as reliable by the company back-end (attacks would be possible exploiting this choice), but this means that any activity which does not require to retrieve information from the back-end or change the status of this information, can happen entirely on the client side without any network requests. The first access to the website is described in Figure 17: this provides a time line[1] of the files being requested. From the list of files some are of particular interest: the file which ends in .nocache.js is not cached, as the home page. This file requests the current version of the cahceable JavaScript, which has a name starting with "244BCF" in the example provided. The use of these two separate files means that the second one can have a very long caching time (365 days), while the website can change by simply modifying the .nocache.js file. Future accesses will not require any time to access the cached contents, and therefore this time is significantly reduced (the biggest file is cached, accounting for 67.2kB out of 76.8kB totally to be loaded).

[1]This time line was generated as part of the results of Pingdom Tools: http://tools.pingdom.com/



http://[redacted].appspot....

Tested from Dallas, Texas, USA on April 4 at 11:53:15

Perf. grade	Requests	Load time	Page size
76/100	8	656ms	76.8kB

Your website is **faster than 95%** of all tested websites

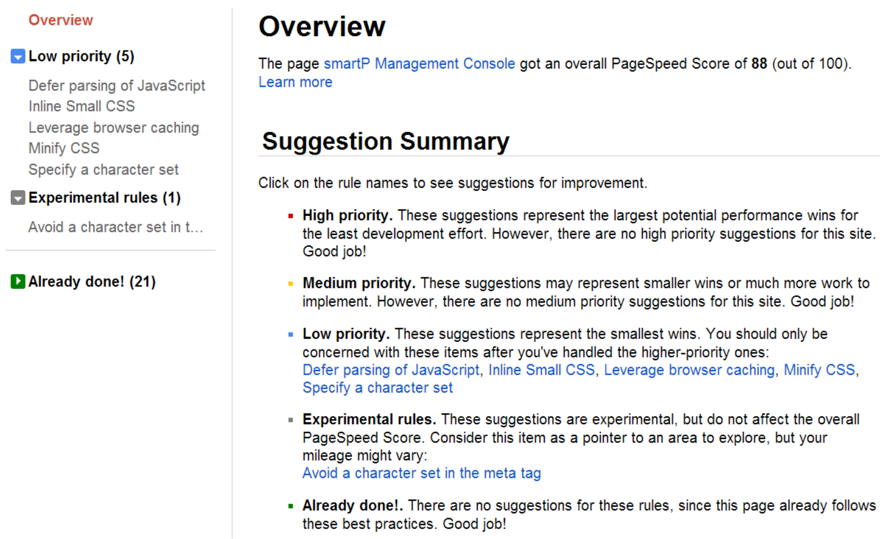
Figure 18: Results of the Pingdom Tools on the company back-end

The overall performance is very high, as shown in Figure 18: the company back-end, and more specifically the management console, appears to be faster than 95% of the websites tested by Pingdom. Of course this result is more than what would be needed for the management console: indeed it is not supposed to be used often, because it is only relevant to the management. But since the management console is used by the decision makers, it is important that the quality is as high as possible. Talking about quality, all of the benefits presented here are due to the fact that GWT was chosen as a framework, in fact all caching preferences are configured automatically for files generated by the GWT.

Finally, as proof of the quality of the company back-end and management console in terms of optimizations to speed up loading time, the Google PageSpeed was used to assess the service. Overall results are presented in Figure 19: the score is 88 out of 100, and there are no high priority nor medium priority suggestions to implement in order to improve performance. 100 represents the maximum theoretical loading speed, while applying all possible optimizations.

In order to grasp the result obtained here, the same test was performed on other websites, and the results were always worse. It is important to note a series of aspects which influence the results: first of all, all of the websites being tested[1] are supposed to invest significantly in these kinds of optimizations, and therefore they can be used as benchmarks of high quality. At

[1]For W3Schools URL http://www.w3schools.com/ was used, while for CBS http://www.cbs.com/ and for Oracle http://www.oracle.com/index.html and for Wikipedia http://www.wikipedia.org/ These tests were performed on the 4th of April, 2013.



Overview

The page smartP Management Console got an overall PageSpeed Score of **88** (out of 100).
Learn more

Suggestion Summary

Click on the rule names to see suggestions for improvement.

- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. However, there are no high priority suggestions for this site. Good job!
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. However, there are no medium priority suggestions for this site. Good job!
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones: Defer parsing of JavaScript, Inline Small CSS, Leverage browser caching, Minify CSS, Specify a character set
- **Experimental rules.** These suggestions are experimental, but do not affect the overall PageSpeed Score. Consider this item as a pointer to an area to explore, but your mileage might vary: Avoid a character set in the meta tag
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!

Figure 19: Results of the PageSpeed test on the company back-end

the same time, the management console has significantly lower content and therefore is easier in general. Finally, because of how the scoring system is defined, web pages with less content do not give unfair advantages compared to the ones which have rich content: all that matters is how optimized the content is. Table III reports these results: the first row describes the score, on a value from 0 to 100, while the other rows indicate the absolute number of possible optimizations to speed up page loading and are divided based on importance, from high to low. Experimental are the less urgent suggestions to take care of, and "Done" represents those suggestions the website already complies with..

All of the results provided here are only technical, while it will become important to consider human results as well in the future. That said, based on the results obtained, the original requirements in terms of performance, simplicity and scalability are met by the proposed solution which was validated here.

TABLE III: PAGESPEED COMPARISON BETWEEN DIFFERENT WEBSITES

	ManagementC.	W3Schools	CBS	Oracle	Wikipedia
Score	88	87	82	64	64
High	0	1	2	3	0
Medium	0	1	1	2	3
Low	5	10	10	8	2
Experim.	1	1	0	0	0
Done	21	14	14	14	22

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

The proposed solution meets the Minimum Viable Product requirements initially set, which allow the people who can access the service to take advantage of the new technologies and have a better service in terms of user experience, while effectively satisfying all the needs of the parking companies. This solution is effective, in the sense that it has all the necessary components to start offering the service, and it has everything which might be required to obtain an agreement with parking companies.

Moreover the proposed solution offers the base to start building additional services for all stakeholders, not only drivers but also parking companies. And the key element is that anything that is part of the future works, presented in Section 6.2, can be built with minor involvement of the parking companies. More specifically, anything which involves the user application and smartP back-end does not even require any sort of collaboration on the parking company side. On the other hand, anything which requires the company web portal or the inspector application (or the company back-end) will usually require a new deployment of the back-end for each parking company. This should not be an issue, because usually the big step is in starting to offer the service, and not in upgrading it later, but it has to be kept in mind that any user application changes which require additional information from the company back-end will require to go through this step.

The choice of decoupling the smartP back-end and the company back-end has a really significant benefit over the currently available digital solutions: many companies can potentially participate in the competition of acquiring customers and offer them to pay for parking through their application. The long term potential is way more than just this: any additional information which is collected or calculated at a company level can potentially be released to use by a series of services through the same APIs which can take advantage of this new data to come up with innovative services for citizens. Again, decoupling the two back-ends allows competition on both levels: more companies can compete on pricing at the parking level, where they can use the smartP application to let users know about the existence of their parking spots and they can attract users if the price is competitive. At the same time the decoupling reaches the other advantage of allowing multiple companies to offer the service of selling parking space, thus competing with smartP and the user application. This might be important for some cities in terms of having non-exclusive agreements.

Of course the goal was not to build a service which could entirely substitute the traditional approaches, and in fact the proposed solution cannot do that: there are some adoption barriers in place which make it usable only by some of the drivers. On the other hand, the real goal was to offer an alternative to the current digital solutions (whether application based or custom hardware based) which would be good enough for people who can use it to switch to the new solution in higher percentages as compared to those digital offerings, where the adoption rate is still very low. Even though the proposed solution is not yet commercially offered, and therefore there is no data on adoption compared to previous solutions, the proposed one reduces all adoption barriers about bad user experience, such as the required user input which is reduced compared to previous solutions. Moreover our solution does not require any interactions of the drivers with a computer, and finally it solves the issue of being local by offering a layer which hides the single company back-ends. Again, this achieves a very important result, which is being able to offer a single application supported by many cities where it can be used by drivers to pay for parking. This is something that more than half of the users of other applications to pay for parking ask exactly for this, which is why it is of particular importance.

The cloud choice of using Google App Engine offers a good compromise between not taking care of the server infrastructure and being able to scale, while having a performant deployment of the back-end with all required features to make it work. As mentioned earlier, thanks to the existence of AppScale, and thanks to the use of JDO as a connector to the DataStore, the code is generic enough for the system to be ported on custom servers with minimal changes (if any), which allows for example the deployment of the company back-end on their servers if they so choose.

The modularity allows the management to describe in detail the policies their parking spaces have, which is another key aspect: this means that the system is general enough to offer the same features of most of the ones presented in the State of the Art (related to payments) both based on custom hardware and on applications. This will make adoption on the parking company side easier, as it would be much more difficult to require a change in the policies to start offering smartP, compared to just adding support for a new payment mechanism which does not require any changes to the strategic decisions of the company. At the same time the support for different policies means that the company can keep using the service even if they make a significant change in their payment rules (e.g. if they decide to switch from unnumbered car spaces to numbered ones).

6.2 Future works

As mentioned earlier, the proposed solution is the base for a series of possible future extensions, which would offer a better and more complete service for all stakeholders. Before starting to add additional functionality a series of works will take place. To begin with, user applications for other platforms will be developed, starting with iPhone and then other devices, based on popularity. Moreover a Windows Phone version of the inspector application might be required, based on agreements with parking companies. Moreover a discussion with companies will allow smartP to start offering a digital fine system which complies with the law and specific requirements of the company. Furthermore a method to interact with existing validation mechanisms will be built, based on requirements, based on specific requests from the parking owner; in fact the modular structure allows for an easy interaction, but as of now only interaction with our proposed company back-end is supported. Finally a smartP back-end management portal should be added, so that accounts and companies can be managed from there.

Other aspects to be considered will be determined by studying the use of the final user application. In particular feedback from final users will be used to improve the application itself, plus the user interface will be additionally optimized to reduce the effort for the final user to use it. Also feedback of the inspectors and management will be taken into account in order to improve the service for the future.

Once these general goals are reached, the service will be of better quality than the current one. After this step, a series of projects can start, each of which tackling a specific issue, and for this reason they will be presented independently. The list of these extensions is partial, because other potentially interesting extensions are possible, but these are the ones we expect to be the most beneficial to smartP users.

New forms of payment

In terms of payment, the Minimum Viable Product only allows to pay by credit card, which is of course an adoption barrier for all of those people who do not have a credit card, and those who do have it but would not trust giving credit card details over their phone. For this reason it is important to start offering additional payment methods to lower the adoption barrier. Of course the best solution would be to simply charge the user directly on the phone bill but, given the current fees for such a service, it is unlikely that this will ever become a viable solution.

One of the possible alternatives is offering prepaid cards which could be bought at businesses such as news agents. This would be a good option because for those people who do not have a credit card the service now becomes accessible. The costs could be high but, by increasing the value of a refill card, these costs can be small in terms of percentage.

Another alternative is to use accounts of other payment methods. For example in Italy there is a hardware device (called TelePass) which is used by people to pay for highway fares without having to stop at the toll booths or waiting in queues for their turn at the toll booths.

These devices already have an associated account which is then used to charge any expenses. An agreement with these companies would allow to charge smartP session fees on the same existing accounts users already have.

Finally park-meters, if advanced enough, could be used to refill accounts as well. These park-meters would become self service machines for smartP users, who do not use park-meters otherwise. This option is a very good one in terms of transaction fees, because the same costs which the parking company already pays in order to manage the money collected through park-meters would be used to collect this additional money and, for as long as the park-meters are required, it is the solution which allows to keep the highest percentage of the money being paid by drivers. Of course when having a smartphone can be expected of people, and the adoption barriers to using smartP are low enough for any driver to use it, then park-meters will not be maintained any more, thus making this option expensive. The reason why it is worth considering this option is that in the foreseeable future it is unlikely that adoption barriers to using smartP become low enough to stop offering the park-meter alternative.

Guide to available parking spaces

Being able to drive drivers to available parking spaces would be beneficial to everyone: the driver in the first place, because it would have a significant impact to know about the availability in various areas beforehand. Different pieces of information could be provided to the driver: for example the closest parking space, but also the availability of another parking space which is further away and has a lower price: basically the driver becomes now fully aware of the status of parking spaces and can make an informed choice, without having to go around looking for parking spaces.

For drivers this means that easier choices can be made, and at a lower cost compared to not having this level of information. Moreover this also means less time spent looking for parking spaces, because the driver can go straight to the final parking destination as soon as s/he is close to the destination and starts a search for available parking spaces.

For the city there is a significant advantage in terms of traffic, because this would reduce the part of it due to people looking for free parking spaces. This accounts for more than 30% of the traffic in cities, and it is important to note in this context that trip time is not linearly correlated to the traffic congestion, in the sense that the worse the traffic condition, the longer the time, but having twice the amount of cars does not mean just taking twice the same time for the trip. And of course reducing traffic has a positive effect on all citizen, because of the reduced pollution.

Based on the fact that both drivers and the city would significantly benefit from such a service, different business models could be proposed, where either one of the stakeholders (or even both of them) would support the costs of the project.

In terms of technology different strategies can be used to be able to achieve this goal, and they vary in terms of required investment, obtainable accuracy, ease of use and distance from the solutions currently in use. For example, where parking spaces are already numbered and drivers pay for a specific parking space it is sufficient to just collect the data and present it to final users through the application. On the other hand, where parking spaces are not numbered one option would be to change to numbered parking spaces. This solution would be especially good for the parking reservation, which is hard to accomplish without numbered spaces (unless a significant investment is made).

The cheapest solution for unnumbered parking spaces would be to use the people who pay with smartP to calculate some statistics on available parking spaces and give suggestions on the likelihood of finding a parking space in a particular area, based on statistical data and current information. Of course this approach is also the least accurate, because it relies on people using the application to behave in a statistically significant way. The first issue with this is that, if the number of drivers using the application is low, the information is not enough to determine the status of the parking spaces. Moreover a scarce presence of drivers using the application in a certain area might mean either that there are a lot of available spaces or that all spaces are already taken by people who do not use the application, and for this reason the information about the status of parking gathered through the application is partial and subject to interpretation. Of course with the increase in adoption of smartP this solution will become more and more reliable.

Moreover another solution is to put hardware sensors to check if cars are parked in parking spaces or not. This solution is definitely the most expensive, requiring a sensor for each parking space, but it is also the most accurate, while being compatible with any kind of preexisting solution: it does not interact with the payment solutions at all, being entirely independent from it. After the initial investment the only problem with this solution is to make sure that enough people use it, because the only way to gain benefits from this is if people use it to reach the closest parking space available, otherwise there is no significant benefit in terms of traffic congestion reduction.

Finally the last approach, found in the State of the Art and potentially worth exploring is the use of cameras around the city to detect available car spaces. This idea is less expensive than having sensors at each space, and at the same time it might require even lower investments if the cameras are already in place for other types of monitoring, so the only investment is to collect and analyze the data. On the other hand the accuracy of this approach is lower, because of the difficulty to accurately determine whether cars are parked by analyzing the video in an automated way, and also the coverage is probably only limited to a number of car spaces, because it is not an easy task to collect video data for an entire city and analyze it constantly. Similar to this approach an alternative would be to install cameras on vehicles which monitor the streets and roads they visit and report back. This has similar issues, mainly the low accuracy, but it offers higher coverage for the same investment.

Parking reservation

Being able to reserve a parking spot might be yet another reason for people to use the smartP application. This is a feature which would be beneficial for drivers who are willing to pay a premium for the benefit of having a parking spot reserved for when they arrive, and it would also be beneficial for smartP to have a more elaborate business model, because it would be an alternative revenue stream.

Apart from the obvious advantage for the user and smartP, the advantage is also for the parking owner, because they could start getting additional revenue from a service which they would not offer otherwise. On the other hand, this service requires a series of enabling conditions which might make it difficult to offer. Before describing these conditions it is important to note that some of them are in common with some of the options to monitor available parking spaces, and therefore if this service is worth offering the decision on the best approach to guide people to a free parking space would be impacted by the requirements of offering the parking reservation.

Being able to build both services at the same time would have scale economies, because in fact the investment to offer them both would be lower than the investment required to offer the two services separately.

First of all, the place has to be free for someone to reserve it, or the system can speculate that the place will be free at the time of arrival of the person who booked it. For example if a place is occupied by a car and the parking session lasts another 45 minutes, a different driver books a place in an hour from now, the system can assume that the place will be free at the time. This is not to say that this example is a safe assumption but, based on the behavior of the drivers over time, safer assumptions could be made.

The second thing is that there should be a way for a driver who is not using the service to know whether a parking place is available or not, even if it is currently free of other parked cars: in the earlier example, other drivers could be arriving during those 15 minutes between the first car goes away and the second one arrives. At the same time there should be a way for the driver who booked the car space to know which one it is, without any margin of error: this feature is different from the one which leads people to empty spaces in the sense that this service gives the right to find the space, while the previous one only gives a hint, because there is no promise as the service is offered for free.

Finally there should be a way to punish people who park on empty spaces without permissions severely: this is the only way to make sure that other drivers will actually care if they do find a free space. This might actually be a serious problem, in the sense that satisfaction level in general might decrease if drivers who do not have a smartphone (or an internet connection) keep finding empty spaces where they have no right to park. Again, with the increase in adoption this problem will decrease.

Different approaches to offer this service are possible. First of all switching to numbered car spaces might be an option. The advantage is of course that investment costs are kept to a minimum, and that it has all of the hardware elements to offer both services of finding a free parking space and of finding a parking space to book: all data is collected either via park-meters or via the Internet and applications, which allows the system to take a decision by relying on information which is very accurate (even if not perfect, because drivers might for example leave their place before than the session expiration time, in which case the system does not know about it and people who book their space will not be assigned to one which was freed before the time. Coupling this solution with a more sophisticated one to determine available parking spaces (e.g. the one using cameras) would offer even more precise information, solving the specific problem presented in the example.

A second possible solution would be to install a park-meter for each parking space. In terms of investment this is the most expensive solution, but at the same time it would reduce the chances of people parking in booked spaces, because the park-meter could give this information and the driver would not even park in a booked space. Moreover this approach would make it possible to know, through specific sensors, if there is a car in the corresponding parking space. This way ticket inspectors could be informed about all anomalies, and their role would be to check these anomalies and generate fines most of the time, while still doing occasional random checks to make sure that there are no workarounds to avoid paying the parking session and getting caught. Of course these are only suggestions based on the state of the art, as it is not the objective of this work to determine a favorite method to find free parking spaces nor to reserve them.

When this feature is considered, some thought should be put on determining the right pricing as well: as usual the best goal would be to encourage positive behavior. One way to accomplish this would be to determine a price for the reservation, and then increase it for bad behaviors such as arriving late or not showing up at all. Of course the price increase would be based on the gravity of the action and it could comprise a component of loss of profit for the unnecessary empty space and a component of damage to the other drivers, who could have used the empty parking space. Every time the driver books a new parking space the price for the service could be updated based on his/her past behavior. Positive behavior (such as showing up on time or canceling a reservation as early as possible) could be corresponded with a price decrease just as bad behavior would be punished.

Better knowledge of the city mobility by the city council

The proposed solution is already collecting a variety of useful data, which of course is more significant if more people pay using smartP than the other existing solutions. With the addition of other features, even more data will be collected, and all of this additional information can be used to improve the service to drivers and the city in general.

To begin with, the information collected could be used to show the filling rates of parking spaces by area. This information is particularly valuable, because it allows to determine the right price for the area to achieve a 80-85% filling rate. It is already possible to assess it, by observing the cars around the city, but it has to be performed manually by people who go around the city. In order to have a pricing based on the hour of the day (or even day of the week) a lot of data has to be collected, and this is not cheap to accomplish unless the data is collected automatically. As soon as the data is available almost instantly, the problem is to just make a decision on the pricing policy and apply it. Of course the benefits of having a more elaborate pricing policy than the traditional one is the constant availability of some parking spaces, a reduction in traffic and the highest number of people being able to reach businesses in the area with the quickest turnover possible.

A mechanism like the smartP application is an enabling factor for a more sophisticated pricing policy which accounts for traffic: the company back-end can already be used to set up different hourly prices based on location and time, as opposed to traditional solutions like scratch-cards, where implementing such a complex mechanism would be possible but particularly cumbersome for final users.

Of course when implementing this feature a lot of studies will have to be done, especially on acceptance by people. Similar tests have been performed, but they are not conclusive for all cities and the risk is that people become aware of different pricing policies and change their behavior to the point that they would rather avoid going to an area where parking is charged a variable price than risk paying too much more than what they are used for. Moreover past results show that a price decrease does not have significant effects in terms of parking fill rate, while price increases usually have a greater impact. All of these considerations do not make the idea bad per se, but it is of the utmost importance to account for them when considering the adoption of such a project.

CITED LITERATURE

1. The World Bank: *Passenger cars (per 1,000 people).* http://data.worldbank.org/indicator/IS.VEH.PCAR.P3, Retrieved May 22, 2013. [Citation at page 1]

2. Comune di Milano: *Bilancio di previsione 2011.* http://allegati.comune.milano.it/redazioneprogrammazionebilancioetributi/Bilancio2011/Bilancio%20di%20previsione%202011.pdf, Retrieved May 22, 2013. [Citation at page 2]

3. Council of Edinburgh: *Parking Facts and Figures.* http://www.edinburgh.gov.uk/info/1268/pay_and_display_parking/1126/parking_facts_and_figures/2, Retrieved May 22, 2013. [Citation at page 2]

4. Council of Edinburgh: *Council information, performance and statistics.* http://www.edinburgh.gov.uk/info/695/council_information_performance_and_statistics/873/key_facts_and_figures/2, Retrieved May 22, 2013. [Citation at page 2]

5. Citizens Advice Bureau: *Parking, Scratch Cards, Season Tickets.* http://www.cab.org.je/index.php?option=com_content&task=view&id=54&Itemid=59, Retrieved May 22, 2013. [Citation at page 8]

6. Proactive (South) Ltd.: *Print Scratch Cards.com.* http://printscratchcards.com/super-scratch-cards.php, Retrieved May 22, 2013. [Citation at page 9]

7. Pickett, M.: *Cashless payment systems in parking.* http://www.britishparking.co.uk/write/Documents/Library/IHT%20Cashless%20Payment%20Systems%20in%20Parking.pdf, Retrieved May 22, 2013. [Citations at pages 9 and 13]

8. Babson, R.W.: *Check controlled apparatus for measurement and payment of intangible values.* Patent, 1929. [Citation at page 10]

9. LADOT: *Cashless payment systems in parking.* http://ladot.lacity.org/pdf/PDF1.pdf, Retrieved May 22, 2013. [Citation at page 12]

10. The City of New York: *NYC DOT - Parking at a Muni-Meter.* http://www.nyc.gov/html/dot/html/motorist/meterpark.shtml, Retrieved May 22, 2013. [Citation at page 12]

11. Comune di Treviso: *iPARK.* http://www.actt.it/uploaded_files/info_parcheggiA3_07.pdf, Retrieved May 22, 2013. [Citation at page 12]

12. Wong, Y.: *On-street parking meter system supporting multiple payment tools and signal transmission method thereof.* Patent, 2012. [Citation at page 13]

13. Melanson, D.: *San Francisco rolls out new smart parking meters with 'demand-responsive pricing'.* http://www.engadget.com/2010/08/07/san-francisco-rolls-out-new-smart-parking-meters-with-demand-re/, Retrieved May 22, 2013. [Citation at page 13]

14. Silberberg, M.E.: *Parking system for sending messages.* Patent, 2007. [Citations at pages 13 and 19]

15. eSmart21: *Automated parking enforcement system.* http://www.esmart21.com.au/images/brochures/smart%20epark%20brochure_v6%20for%20website.pdf, Retrieved May 22, 2013. [Citation at page 14]

16. OTI - On Track Innovation LTD.: *EasyPark.* http://www.otiglobal.com/EasyPark, Retrieved May 22, 2013. [Citation at page 14]

17. Ganis Systems: *Comet - Personal Parking Meter.* http://www.ganisparking.com/, Retrieved May 22, 2013. [Citation at page 14]

18. Tomer, N.: *Parkulator photo parking.* Patent, 2001. [Citation at page 14]

19. Odinak, G.: *Vehicle parking validation system and method.* Patent, 2005. [Citation at page 15]

20. Shanker, S.: *An intelligent architecture for metropolitan parking control and toll collection "IMPACT"*. Masters thesis, Wayne State University, 2005. [Citation at page 16]

21. Wang, M.: *Parking toll system*. Patent, 2004. [Citation at page 20]

22. EasyPark: *Pay Parking by App - EasyPark World*. http://easypark.net/motorists-interfaces/smartphone-app/, Retrieved May 22, 2013. [Citation at page 22]

23. Rismanchian, F.: *UQParking: A Smartphone Application to help locate parking places at The University of Queensland*. Master's thesis, The University of Queensland, 2012. [Citation at page 23]

24. Parking Salt Lake City: *QP Quick Pay Smartphone App*. http://www.parkingslc.com/maps-and-apps/quick-pay-smart-phone-app, Retrieved May 22, 2013. [Citation at page 23]

25. Bernspång, F.: *Smart Parking using Magnetometers and Mobile Applications*. Masters thesis, Luleå University of Technology, 2011. [Citation at page 23]

26. Ottoson, D.B., Lin, H., Chen, C.: *Price elasticity of on-street parking demand - a case study from Seattle*. 4th Transportation Research Board Conference on Innovations in Travel Modeling (ITM), 2011. [Citation at page 24]

27. Chohan, N., Bunch, C., Pang, S., Krintz, C., Mostafa, N., Soman, S., Wolski, R.: *AppScale: Scalable and Open AppEngine Application Development and Deployment*. First International Conference, CloudComp - Munich, Germany, 2009. [Citation at page 72]

28. Cloud Special Interest Group, PCI Security Standards Council: *Information Supplement: PCI DSS Cloud Computing Guidelines*. https://www.pcisecuritystandards.org/pdfs/PCI_DSS_v2_Cloud_Guidelines.pdf, Retrieved May 22, 2013. [Citation at page 77]

VITA

Davide Leonarduzzi

EDUCATION

- Maturità Scientifica (equivalent to High School Diploma) at Liceo Scientifico Giovanni Marinelli (final grade: 92/100), 2007.
- Laurea (equivalent to Bachelor of Science) in Ingegneria Informatica at Politecnico di Milano (final grade: 100/110), 2010.
- Laurea Magistrale (equivalent to Master of Science) in Ingegneria Informatica at Politecnico di Milano (final grade: 109/110), 2013.
- Attending Master of Science in Computer Science at University of Illinois at Chicago (final GPA: 4.00/4.00).