Journal of Computer and Communications, 2015, 3, 42-64
Published Online June 2015 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2015.36006

An Initial Approach for a NFC M-Ticketing Urban Transport System

Cosmina Ivan, Roxana Balag

Department of Computer Science, Faculty of Automation and Computer Science,
Technical University of ClujNapoca, Cluj-Napoca, Romania
Email: cosmina.ivan@cs.utcluj.ro, roxana.balag@gmail.com

Received 19 April 2015; accepted 25 May 2015; published 28 May 2015

Abstract

Public transport plays an important role in the daily lives of hundreds of millions of people. The limited national and local finances starve the public transport networks of funding, although greater usage would imply important environmental and financial benefits. The recent advances in mobile technologies and services could be harnessed to increase the attractiveness and efficiency of various public transport systems. Increasingly flexible electronic ticketing, smart cards and real-time information feeds via smartphones are making transportation networks faster, cheaper and more efficient for both the passengers and operators. In this paper, we present our approach in designing and implementing an M-ticketing system for urban transport based on NFC technology. We make initial design decisions based on a study of two similar systems Oyster and U-Go. We make use of Microsoft technologies for the prototype implementation and we propose a test suite to validate the system. We conclude on presenting the experience gained from this project and propose a set of further developments in order for the prototype to become more realistic.

Keywords

M-Ticketing, NFC (Near Field Communication), Urban Transport

1. Introduction

Transport operators around the world have tried using mobile technologies and services to enhance the passenger experience. In some parts of Asia, such as Japan and South Korea, mobile handsets are already widely used to purchase and validate tickets using Near Field Communication technology (NFC). Also similar systems are being deployed in parts of Europe and the Americas, which shows the scale and growth of pilots and deploy-

How to cite this paper: Ivan, C. and Balag, R. (2015) An Initial Approach for a NFC M-Ticketing Urban Transport System. *Journal of Computer and Communications*, **3**, 42-64. http://dx.doi.org/10.4236/jcc.2015.36006

ments around the world. The adoption of mobile technologies and services by the transport sector is being driven by two major trends. Firstly, consumers increasingly rely on smartphones to manage their daily lives. The use of SMS and QR codes is now common in the transport industry. In developed markets, in particular, consumers have become accustomed to using social network, dedicated apps and mobile web sites to quickly access information and make purchases. As a result, both commuters and occasional passengers now expect to be able to user their smartphones to interact also with the public transportation system. Many transport operators are under pressure from transport authorities to cut costs, while at the same time making their services easier to use and more appealing for passengers, encouraging the switch away from the private car. At the same time, transport operators are typically required to keep ticket prices down to a level where public transport is affordable for the vast majority of the population.

There are different kinds of values that an e-ticketing system based on mobile technologies can create for consumers, transport operators, mobile network operators and public transport authorities. However, the value created by mobile NFC, in combination with other technologies, could go beyond localized benefits as traveling can become smarter. These potential benefits spread across four different categories:

- Time savings-due to the fact that tickets can be bought on the go, without the need to wait at a queue.
- Information availability-this benefit is relevant for all implied parties, not only for the customers who are more likely to use public transport when the relevant information is at hand, but also for the transport and mobile operators, who are in a better position to offer more valuable services for customers if they can be based on the customers' habits and journeys.
- Financial savings-this represents an essential aspect for transport operators and could arise for example from fraud reduction or the cheaper distribution of tickets.
- Financial gains-new and innovative services can create new streams of revenue.

This paper is organized as follows: in Section 2 a bibliographic study is performed on the domain of e-ticketing, NFC technology and two e-ticketing systems. Section 3 contains the functional requirements for the system, an overview of the technological decisions, the proposed architecture and specific implementation aspects. In Section 4, we present two most important tests scenarios, and in the last section we present concluding remarks and propose guidelines for future development.

2. Bibliographic Research

As stated by the authors in [1], paper-based tickets are still a reality in public transportation, although for many years the public transport market entities tried to replace paper tickets by electronic media or e-tickets. Despite its many benefits for the transport operators, e-ticketing has not yet been able to live up to the users expectations. According to [2], public transport operators in many countries have implemented or are about to introduce e-ticketing systems, and according to [3] hundreds of millions of people are familiar with payment and ticketing cards that use contactless techniques because of the expansion of public transport payment systems such as those in Hong Kong, Tokyo and London since the late 1990s. It is stated in [3] [4] that Near Field Communication is a relatively new technology that, among other things, allows mobile phones to emulate smart cards such as the travel cards used in public transportation. Despite its promising services and optimistic predictions, NFC technology and mobile ticketing services based on it, have yet to take off. However, NFC holds much potential for mobile services and one of the most promising early applications of this technology is considered to be *mobile ticketing*. Studies presented in [4] suggest that there is general consumer interest in the services mobile commerce provides, for example purchases on web sites, routine bank services, and electronic receipts and tickets. In the context of mobile payments, both technology trust and trust in transaction partners (mobile network operators, payment services providers) are likely to play a role in trust formation.

Each of these aspects will be presented in more detail along with the benefits of building a NFC-based mobile e-ticketing system.

2.1. E-Ticketing

A ticket can be defined as a contract between a user and a service provider. If the user can prove his/her ownership of the ticket, then he/she is granted the right to use the service under some terms and conditions [4]. Usually, the ticket needs to be validated in order for the user to be able to use the service. Basically, ticketing represents a

company's pricing policy, with the consideration of operational, commercial and social objectives. The ticketing system is the translation of fares into concrete means of payment for the passenger and fare collection for the operator [5]. The main steps in using a service with such a pricing policy are: issue, ticket payment and validation. In the public transportation market, several types of tickets can be identified: single journey tickets, single-operator tickets, multi-journey tickets, weekly or monthly passes, group tickets and special event tickets. Tickets can be delivered as paper tickets or electronically (e-tickets), but paper-based tickets usually imply relatively high maintenance costs.

To facilitate the use of public transport, cities aim at making the ticketing system as easy and as attractive as possible. Electronic ticketing is a form of electronic commerce for different kinds of tickets, having as main characteristic the fact that the tickets are sold and stored in an electronic device, such as smart cards or mobile phones. Like paper tickets, electronic tickets should include some basic information for their practical use. Some of the information electronic tickets could contain are presented in [4]: Unique identifier for every ticket, Issuer—the entity responsible for issuing the e-ticket, and can be different from the service provider, service provider—the entity who delivers services to the user, User—information about the e-ticket owner, Service—description of the service contract, Terms and conditions of use, Type of ticket, Transferability, Number of uses, Destination, Attributes—other information about the e-ticket, Date of issue, Issuer's digital signature—if issuer has a public key cryptosystem, being able to digitally sign the e-ticket and Device identification.

Based on how complex the e-ticketing system is, in [5], five different business models for public transport e-ticketing systems have been identified:

- Prepaid value model—it is currently the most common form of automated ticketing. The ticket is issued by the transport operator and based on the value stored on a card (value which can represent money, number of rides, time-interval).
- Enhanced payment card—it is represented by a contactless credit or debit card or a payment application on an NFC phone which is used to pay the fares. The user presents the card to a reader and the payment transaction is processed based on the public transport operator's fees.
- Postpaid model—it is based on smart cards or NFC-enabled phones. The fares are billed afterwards, according to the usage which was recorded (user's location and identification are required).
- Combined/enhanced collaborative models—in this case, a smart card or phone incorporates multiple applications (transit and payment).
- Embedded secure element/(U)SIM—in this model, an intermediary, usually a trusted service manager, or a mobile network operator, or a handset manufacturer determines the business rules. This can be the case for all business models mentioned above.

According to [5], there are 26 European countries that operate smart card ticketing systems, some of them nationwide, but most countries have at least an e-ticketing system in their capital. A next step forward is to combine existing schemas to create national solutions to finally come to cross-border interoperability. In Asia, for example, where the largest e-ticketing schemes exist, there is no true interoperability between neighboring territories or networks. In order to take advantage of the many technologies that exist for e-tourism, also interoperability of different products is necessary. Apart from its core application in the transport sector, e-ticketing could be extended to other areas, such as tourism and leisure activities. That way, users of e-ticket could make use of a service package that promises greater flexibility and convenience.

According to the Urban ITS Expert Group [2], one of the major advantages of smart ticketing should certainly be to propose complementary services, other than transport payment services. They emphasize that smart ticketing does not necessarily mean to have one ticket for one journey, but one wallet with several tickets (which can easily be bought) and in the future possibly one wallet for several services. Integrating other services, such as the possibility to enter a museum with that ticket, represents the most advanced level of integration. Until today there is no real interoperability for touristic e-services, resulting in high deployment costs and a lack in flexibility. A customer should be able to download an application onto their preferred (and compatible) media (smart card or mobile phone), which can be recognized in all participating countries. When traveling abroad, users would be able to use that media and buy a transit pass for the duration of the stay. Interoperability in e-ticketing for public transport implies removing the obstacles for the customer to switching transport modes. All ticketing needs should be in one place and on their local transport smart ticketing media. It has already been shown that it is technically possible to adapt applications and download them onto multi-application cards/mediums.

2.2. Near Field Communication

In 2004, NXP Semiconductors, Sony and Nokia founded the NFC Forum in order to bring existing standards and efforts of the RFID and smart card technology together and to create a novel and innovative capability for short-range communication.NFC is a short-range wireless communication technology that is based on approved and mature standards in the field of RFID and smart cards. RFID, which has already been introduced in the 1970s, realizes automatic identification and data transfer via electromagnetic radio signals, typically by means of an active reader that is connected to a source of energy and a passive electronic tag that is a transponder receiving its power from the reader by magnetic induction. The RFID tag normally contains an antenna for receiving and transmitting the radio signal and an integrated circuit for processing and storing information and for modulating and demodulating the signal [6]. A small microchip is surrounded by a copper antenna. The RFID tag can be placed almost everywhere and is normally hidden behind existing material, like the packaging of a product, thus being invisible to the user.

NFC is a wireless proximity communication technology that enables data transfers between two devices which are close to each other, typically to a distance of less than 10 centimeters [6]. This simple means of establishing a connection is a major advantage of NFC over other wireless communication technologies, such as Bluetooth and Wi-Fi. However, compared to connection speeds of Bluetooth and Wi-Fi, NFC provides slower data transmission rate of up to 424 kilobits per seconds (kbps). In addition, the communication range of NFC is shorter than that of other communication technologies. However, this is not considered a drawback, but an inherent characteristic and a technical advantage of this technology. To be more specific, the close communication range enables intuitive transfers of data by tapping one device against a desired peer device and ignoring other devices that are outside this communication range. This not only helps to prevent signal interference between devices, but also secures users and applications, since users must be close enough to an NFC device to be able to nearly "touch" it, or in other words, in most cases, they intentionally wish to use the application.

For a long time, only a small handful of NFC mobile phones were available, mainly manufactured by Nokia, until Samsung and Google attracted a large audience when releasing the NFC supporting Nexus S phone in 2010. With the current rush on smartphones mentioned above and further successful NFC field tests in the past years, it is expected that in near future most of the top class smartphones will be equipped with NFC support [7].

On most mobile devices, such as mobile phones, there is no way to store secure data directly. For most NFC applications, *i.e.* payment and authentication solutions, secure storage systems are essential. For sensitive data, the storage needs to be resistant to manipulation and it must be able to execute cryptographic functions and to execute security-relevant software. Smartcards usually implement these requirements [8]. To implement such secure elements, there are different possibilities, each with its own advantages and disadvantages:

NFC tags. In June 2006, the NFC Forum introduced standardized technology architecture, initial specifications and tag formats for NFC-compliant devices [9]. These include Data Exchange Format (NDEF), and three initial Record Type Definition (RTD) specifications for smart poster, text and Internet resource reading applications. In addition, the NFC Forum announced the initial set of four tag formats that all NFC Forum-compliant devices must support. These are based on ISO 14443 Types A and B (the international standards for contactless smartcards) and FeliCa (conformant with the ISO 18092, passive communication mode, standard). Already more than one billion tags of this kind have been deployed globally, although for non-NFC applications like mass transit and access control.

The NFC Forum chose the initial tag formats to cater for the broadest possible range of applications and device capabilities [3]:

- Tag 1 Type-based on the ISO14443A standard. They are read and re-write capable. Memory availability is 96 bytes and is expandable up to 2 kbyte. The communication speed of this NFC tag is 106 kbit/s.
- Tag 2 Type-based on ISO14443A. They are read and re-write capable. The basic memory size is 48 bytes and can be expanded to 2 kbyte. The communication speed is 106 kbit/s.
- Tag 3 Type-based on the Sony FeliCa system. Memory availability is variable, theoretical memory limit is 1 MByte per service.
- Tag 4 Type-defined to be compatible with ISO14443A and B standards. These tags are preconfigured at manufacture and they can be read, re-writable, or read-only. They have a memory capacity up to 32 kbytes. The communication speed is in the range of 106 kbit/s and 424 kbit/s.

It is worth noting that Type 1 and 2 tags and Type 3 and 4 tags are very different groups, with very different

memory capacities, and there is very little overlap in the types of applications they are likely to be used for.

Operation modes. There are three operating modes: reader/writer, peer-to-peer, and card emulation [10]. The reader/writer mode enables one NFC mobile to exchange data with one NFC tag. The peer-to-peer mode enables two NFC enabled mobiles to exchange data with each other. In card emulation mode, a mobile phone can be used as a smart card to interact with an NFC reader. Each operating mode has a different technical infrastructure as well as advantages for the users.

Reader/writer mode. The NFC device behaves as a reader for NFC tags, such as the contactless smart cards and RFID tags. It detects a tag immediately in close proximity by using the collision avoidance mechanism. An application on an NFC device can read data from and write data to the detected tag using the read/write mode operations. The reader/writer mode is about the communication of an NFC enabled mobile phone with an NFC tag for the purpose of either reading or writing data from or to those tags. It internally defines two different modes: reader mode and writer mode. In reader mode, the initiator reads data from an NFC tag which already consists of the requested data. In addition to the requirement that the NFC tag already consists of the requested data, it also consists of the program which performs returning the requested data to the initiator. In writer mode, the mobile phone acts as the initiator and writes data to the tag. If the tag already consists of any data prior to the writing process, it will be overwritten.

Peer-to-peer mode enables two NFC enabled mobile devices to exchange information such as a contact record, a text message, or any other kind of data. This mode has two standardized options; NFCIP-1 and LLCP. NFCIP-1 takes advantage of the initiator–target paradigm in which the initiator and the target devices are defined prior to starting the communication. However, the devices are identical in LLCP communication. After the initial handshake, the decision is made by the application that is running in the application layer. On account of the embedded power to mobile phones, both devices are in active mode during the communication in peer-to-peer mode. Data are sent over a bidirectional half duplex channel, meaning that when one device is transmitting, the other one has to listen and should start to transmit data after the first one finishes [6].

Card emulation mode. Card emulation mode provides the opportunity for an NFC enabled mobile device to function as a contactless smart card. Mobile devices can even store multiple contactless smart card applications in the smart card. The leading examples of emulated contactless smart cards are credit card, debit card, loyalty card, transport cards, identity or access cards. Card emulation mode only removes the need for carrying the cards. People carry mobile phones with them most of the time so coupling mobile phones with the human body fits with their use. One can expect that in the near future people will carry NFC enabled mobile phones not just to gain mobility but also to perform daily functions as well. All credit cards, keys, tickets, and so on will be possibly embedded into mobile phones. Hence, there will be more opportunities to integrate daily objects into NFC enabled mobile phones in the future [6].

Data formats. In all modes of operation an NFC Data Exchange Format (NDEF) message is used for the transfer of data, no matter whether the communication takes places between two NFC devices or between one device and a passive NFC tag [6]. To transfer NFC information from one device or tag to another device or tag, a standard encapsulated format is used. The NFC Data Exchange Format (NDEF) describes how information should be sent and organized in the exchange. The standard contains only information on how to organize the information transferred and does not define what is transferred or how the transmission is done. Two NFC devices close to each other will start sending NDEF messages over the NFC Forum Logical Link Control Protocol which is a part of ISO18092, but when a device is close to a tag it will start communicating NDEF messages over the specific tag protocol. Specifically, an NDEF message contains one or more NDEF records. These records can be chained together to support a larger payload. Each NDEF record uses three parameters: payload length, payload type, and an optional payload identifier to describe its payload. The first NDEF record in an NDEF message has the MB (Message Begin) flag set, while the last NDEF record is marked with an ME (Message End) flag. This means that an NDEF message with only one NDEF record has both MB and ME flags set.

Security aspects. Most NFC scenarios are required to deal with sensitive data like credit card numbers, bank account details, account balances, personalized tickets or other personal data. For data storage and wireless data transfer security is therefore an essential issue. NFC thereby provides several mechanisms for security and immunity.

First of all, there is obviously a certain physical security due to the touch-to connect principle. As a matter of fact, the NFC technology only provides data transfer between two devices or between a device and a tagged object when the distance between the two items falls below a certain range. Data skimming, that is capturing and

intercepting transferred data by a distant attacker, is hence not possible that easily [6] [11]. Misuse is however possible in the form of relay attacks. Such attack is basically accomplished via an attacker serving as a man-in-the-middle who is forwarding transmitted data between a reader, e.g. a reading device for NFC payments, and a target transponder, e.g. a NFC device serving as a credit card, that is actually out of the reading range. A possible countermeasure for such relay attack could for example be built upon a quite short timeout threshold that avoids transactions if the response time is too slow. The concept of Google Wallet is however also secured against such attacks as thereby a PIN has to be entered in order to activate the phone's NFC broadcast hardware and to activate the Secure Element that is storing all the sensitive data. The Wallet PIN also prevents unauthorized usage of the payment card in case the NFC phone is stolen.

For the wireless channel communication itself the NFC specifications do not ensure any secure encrypting mechanisms. On higher layers however, NFC applications can of course use industry-standard cryptographic protocols like SSL/TLS based methods to encrypt the data that is to be transferred over the air and that is stored in the Secure Element. Other possible NFC vulnerabilities involve the manipulation of NFC tags [12]. Existing passive NFC tags, e.g. on a smart poster, could be replaced by spoofed tags such that a modified NDEF message is read by a NFC reading device. Possible attacks could for example replace the URI record with a malicious URL, e.g. in order to make the user load a phishing website that steals the users credentials. Furthermore, it is even possible to create a malformed NDEF message that causes the some applications to crash. This form of manipulation could be used by attackers to debase the relationship between a user and the pretended service provider. The Signature Record Type that was previously addressed in this paper and that signs a whole NDEF message can however serve as a countermeasure against URL spoofing and similar attacks. By manipulating the signed tag payload a signed NDEF message will lose its integrity and will be recognized as not trusted.

In general, one can summarize as in [13] that NFC is not more insecure than other related technologies. It offers options for encrypting data on the application layer the same way as WiFi or Bluetooth, but additionally provides safety through the requirement of very close physical proximity. Compared to traditional payment methods including magnetic strip cards that can easily be skimmed or cloned, it is quite difficult to tamper NFC hardware. Beside the physiological concern of transferring money or sensitive data without wired connection over the air, the NFC technology can thoroughly be considered as secure enough for mobile ticketing and payment-at least, if the application developers make use of the provided security mechanisms.

2.3. E-Ticketing Systems for Urban Transport

As already outlined, different types of e-ticketing exist. The most prominent schemes have their origin in public transport; and of those the most extensive schemes are domiciled in Asia. However, not only Asian systems enjoy the status of being exemplary. The Oyster card scheme in London and the OV-chipkaart in the Netherlands as well are often referred to as being excellent, though they can also not easily be transferred to other contexts [8]. In the following, two such e-ticketing systems will be presented in more detail, a complex one and a similar application but with and Android implementation, followed by a comparison with our system.

Oyster Card. Public transport in London is mainly organized and managed by "Transport for London" (TfL). TfL was created in 2000 as an integrated body responsible for all transport issues in London, for public transport, road based transport, cycling and walking, for managing the congestion charge and regulating the taxis. The Oyster card, as stated in [14] [15] was a success story from the beginning on: more than 3000 applicants registered for the card on the first weekend alone. Smart card ticketing is available in a number of UK cities and regions with London's Oyster card being by far the most successful card scheme. The government vision is to extend e-ticketing across the country and to potentially use mobile phones or contactless bank cards instead of smart cards. For using Oyster cards, passengers need to touch their smart card on a reader at ticket gates at the start and end of their journey. It is applicable for everyone, whether living or just visiting London. The Oyster card can store pre-purchased tickets (including weekly or monthly passes) and single fares (that are cheaper than single cash fares). An online auto top-up option that is linked to ones bank account ensures that credit never drops below a specified amount. In 2007/2008 TfL and O_2 ran a six month trial with Oyster products stored on *an NFC enabled phone* which proved to be a success from the customers point of view. More than seven million cards are used regularly in London, each week 57 million journeys are made using Oyster and more than 80% of all bus and tube payments are with Oyster.

U'Go Mobile Application. In France, each city has its own ticketing system. CTS, the transport operator in

Strasbourg, launched ticketing on NFC smart phones running Android in June 2013 in partnership with mobile operators Orange, Bouygues Telecom, SFR and NRJ Mobile. CTS has negotiated a transaction fee with each individual mobile network operator, which it pays for each purchase. Using the U'Go mobile application, passengers can buy either monthly or daily tickets for bus and tram journeys. NFC tags are located in tram stations and attached to ticketing machines on buses in the Strasbourg region [14] [15]. Customers can tap their phones against the tags to validate their prepaid ticket before travelling. The transport applet on the passenger's SIM card is then updated via a mobile data connection, as soon as it's possible. The system also allows ticket inspectors to check mobile tickets for up to 24 hours after a passenger's battery goes flat or while their phone is turned off. CTS isn't using photos for ID, the inspectors typically just check that the ticket has been validated. If they have doubts, they can check the name encrypted on the SIM card and ask the passenger to show some matching ID. The proposed system is a *mobile application running on NFC-enabled devices*. The tickets of each user are stored on the smartphone device and in the service provider's database, such that when the user replaces his smartphone he can receive access to his/her tickets, independently on the smartphone or the mobile network operator.

The tickets by themselves do not store any information about the user, the client application being responsible for granting access to the tickets based on the user's identity. To purchase tickets, the user needs to access the mobile application and select the corresponding option (to buy more tickets). The tickets are, therefore, prepaid, and the user is constantly deciding if he/she wants to get more. No bill is issued in his name and no automatic „top-up" is performed. The user also has the possibility to purchase monthly passes on one or several bus lines, by purchasing a subscription during a freely-chosen period. In order to validate the ticket, the user needs to tap the tag located in the common transportation vehicles with his/her NFC-enabled device. When performing this action, a ticket is subtracted from the user's remaining tickets and the details of this validation are displayed. To check that a ticket has been validated, the user only needs to show to the inspector the details of the last ticket validation. **Table 1** summarizes the capabilities of described systems and compares them with our prototype.

3. System Design

3.1. System Aim and Requirements

For the public transportation companies, which need to provide accessibility and ease of user for their clients while minimizing their own expenses, this project is a solution that supports online payment and electronic format of tickets and subscriptions, which increase the user's interest towards public transportation due to its ease of use and convenience. Unlike manual release of paper tickets and subscription, this project has the advantage of releasing tickets and subscriptions in an electronic format, easy to acquire, without the need to go to a facility.

The project's *main objective* is to offer a way to *release travel tickets and subscriptions in an electronic format*, directly on the customer's smartphone device and to *validate those tickets* in an easy and intuitive way.

Table 1. Related work on e-ticketing systems for urban transport/to be moved upper.

	Oyster	U'Go	Proposed system
Scope of application	Public transportation	Public transportation	Public transportation
System technology	Contactless smart card using MIFARE technology	NFC smartphones running Android	NFC smartphones running **Windows Phone 8**
Payment system	Stored value smart card, possibility to link card to bank account for "auto top up"	Payments charged to the user's mobile phone bill or paid via their bank card	Prepaid tickets, bought through **PayPal,** using the mobile application
Ticket validation	Need to touch smart card on a reader at ticket gates at the start and end of a journey	Need to tap phone against the tag located in tram stations or in buses to validate prepaid ticket before travelling	Need to tap phone against the tag located in trams, trolleybuses or buses to validate prepaid ticket before travelling
Services related information	-	None	**Bus lines information, routes, stations**

This ensures a "*green*" *alternative* to the paper tickets, a way of minimizing operational costs of public transport operators and a minimal effort from the passenger's side.

Among the *secondary objectives*, we can mention the following:

- *Buying tickets and subscriptions anywhere and anytime*, without the need to travel to a dedicated facility to make the purchase—this objective ensures that the payment industry's needs will be met (an alternate method of payment must be used such that the passenger is able to purchase tickets directly from his/her mobile device-online payment).
- *Reducing susceptibility of losing and damaging the aforementioned paper tickets and subscriptions*—the passengers keep their interest in the system and the risk of prejudice is minimized, while ensuring ease of usage.
- *Keeping a history of the passenger's former travels*—the public transport operators can customize and shape their services according to the passengers' needs and travel, providing an opportunity of financial gains and profit maximization.
- Keeping transit services related information at the reach of the passenger whenever he/she needs it—this objective could help maximize the transport operators' profits, by offering up-to-date information about the services they offer, thus increasing the attractiveness and efficiency of the system.
- Get people acquainted with Near Field Communication and increase their trust in this technology—the suppliers get to sell their services, while the public transportation operators increase their attractiveness towards the clients. The passengers are presented with an intuitive way of validating tickets by using their smartphones, in an already known manner.

The identified stakeholders are:

- *Public* transport authorities—this stakeholder is represented by authorities on local, regional and nationwide level, which are interested in an environment friendly transportation system. It is the project sponsor, responsible for finding ways to ensure a "green" transportation system.
- *Public* transport operators—the public transport organizations and their staff are interested in having a tool which simplifies their work, while minimizing costs. This is the main stakeholder, responsible for attracting customers while reducing operational costs and maximizing profit.
- *Payment* industry—banks, credit cards companies and mobile phone operators are interested in integrating their services in transportation systems. They are responsible for delivering solutions for payment to their clients.
- *Suppliers*—necessary suppliers of hardware, software and all kinds of consultants (market, financial) are interested in selling their services. This particular stakeholder is responsible for the development and maintenance of the system (mainly hardware—NFC tags and devices).
- *Passengers*—transit services consumer, interested in buying tickets and subscriptions with a minimal effort. This stakeholder is responsible for using the system.

Functional requirements specify what the system or a component must be able to perform. They specify specific behaviors or functions. The functional requirements of the system are:

- *Registration*—the system shall be able to provide a way for a new user to use the system's services, by creating a personal, individual account from his smartphone device.
- *Authentication*—the system shall be able to provide the user a way to authenticate into the application, such that only the right user has access to the system and user's own private data.
- *Ticket* validation—the system shall provide a way for the user to validate a ticket located on his device.
- *Validations* tracking—the system shall provide a way for the user to see past ticket validations with their respective detailed information, so that the user can provide proof of his/her validation when requested to do so.
- *Subscriptions* tracking—the system shall provide the user a list of all his/her subscriptions, with detailed information about the validity of the subscription (the time interval in which the subscription can be used) and the bus lines supported by the subscription (on which buses can the user travel using a particular subscription).
- *Tickets* status tracking—the system shall provide the user information regarding the status of his/her tickets (number of tickets bought over time, number of remaining/usable tickets) and a brief overview of the last ticket validation.
- Ticket purchasing—the system shall provide a way for the user to buy more tickets, regardless of time and location, without the overhead of traveling to a kiosk.

- Bus lines consulting—the system shall provide the user a list of all the company's bus lines with their afferent *information*, such that the user can make an informed decision about which bus lines to travel on and which lines to subscribe to, such that his/her needs are met.
- Subscription purchasing—the system shall provide the user a way to subscribe to the organization's services, for a month's period, on the bus lines of his/her choice, starting from a chosen date.
- *Logging* out—the system shall provide the user a way to log out, such that at the next usage, his/her credentials will be required.
- Tag writing—the tag writing system shall provide the user a way to configure a tag (acting as ticket validator) with the information of the bus on which the tag will be located.

While features (functional requirements) are the simplest to identify, they are not what determines how successful a solution is. There are a multitude of possibilities that provide the necessary features but what makes one solution a better fit than another are the nonfunctional requirements or qualitied of the system.

3.2. Technology Perspective

In this section, we present the technology perspective for the project, emphasizing arguments in terms of advantages and benefits that lead to choosing that specific technology among other.

Windows Phone 8 *OS* [16]. Windows Phone 8 is the second generation of Microsoft's new mobile client operating system. Windows Phone was a complete and breaking departure from Microsoft's previous platform. At the heart of this evolution is Microsoft's cross-device convergence strategy. With this release, the phone OS is getting one step closer to the Windows 8 operating system for tablets and PCs. In other words, Windows Phone 8 now has a shared core with Windows 8, which includes the well-proven NT kernel that's optimized for multi-core chips, that very same kernel that can run on desktop machines and servers with up to 64 cores. A key consequence of the shared core is that we now have the same.NET engine on the phone and on the desktop, as well as a shared set of native APIs for things such as media, networking, and storage. This means we can build apps that share an increasing amount of code between the phone and the PC. Another consequence of the shared core is that Windows Phone now supports native code and Direct3D.

Another characteristic that has made Windows Phone unique in the market since its launch in 2010 is its distinctive user experience as compared to other mobile platforms. Windows Phone positions itself as the most personal phone in the marketplace, and it does so by putting the user rather than the apps at the center of the experience. At the UI level, this is reflected in the tiles and hubs approach, which represents a major paradigm shift from competing platforms such as iOS and Android. Tiles are really Windows Phone signature feature. The traditional static application icons that are found in other platforms are replaced with live squares that animate in the start screen and can display constantly updated content. The other key component of the Windows Phone user experience is the Microsoft design language, which is a typography-based set of design principles that define the look and feel of the phone. One of these key principles is a focus on application content versus UI frames and graphics, and the same design philosophy is also used for the Windows 8 user experience.

WCF Services. Today, more than ever before, companies demand widespread technology freedom, and interoperability throughout their connected systems. We can accomplish this through what are called services. A service is simply a unit of functionality that you can expose to the outside world via messaging. The way we accomplish interoperability is by implementing that messaging using standard protocols, and message formats. For example, one of the most common transport choices is HTTP, because of its widespread ubiquity throughout the web today. Pretty much every platform has built-in support for this popular protocol. And one of the most common message format choices is XML, either a custom XML dialect of your choosing, or a standard XML dialect that has well defined semantic meaning. The bottom line is, as long as your services communicate using standard protocols and message formats, you will be able to achieve your interoperability requirements. And thanks to this focus on messaging, you will also achieve a more loosely coupled architecture.

There are two primary philosophies that one can adopt when designing service-oriented solutions. The first design philosophy centers on the ideas of SOAP, and the various WS-* specifications. Another design philosophy centers on the idea of REST, or Representational State Transfer. Each of these philosophies comes with its own pros and cons, so it's important to understand the tradeoffs between them. The SOAP approach is probably most commonly used within the context of the enterprise, where you have a more controlled environment to take advantage of the various WS-* protocols, and REST, on the other hand, is probably more commonly used in public-facing web service scenarios, where you might have the requirement for a high degree of scalability.

SOAP and WS-* based services build on a great deal of work to implement a completely new protocol stack for services. One of the primary design goals for this new protocol stack was that of transport neutrality, meaning that any additional features, or capabilities that we want to implement for our services, should be possible to implement in a transport neutral way. And we'll accomplish that in this protocol stack by using an XML-based messaging layer. SOAP is a particular XML vocabulary for packaging up messages that we need to transmit to our services. RESTful services are fundamentally different than SOAP-based services, because they don't attempt to achieve transport neutrality. In fact, RESTful services typically embrace HTTP as the only transport used throughout the system. With REST, you model services as resources, and you give them unique identifiers in the form of URIs., and then you interact with those resources through a standard uniform interface, or service contract. In this case, it would be the methods defined by the HTTP protocol; specifically GET, POST, PUT, DELETE, and HEAD.

By standardizing on a uniform interface, we can build infrastructure around the semantic meaning of each operation, and make performance, and scalability improvements when possible. This turns out to be extremely advantageous when building highly scalable web applications and services. We also need to be able to represent resources using a concrete representation, a message format, such as XML, RSS, JSON, etc. So when we work with our resources, when we request them, or update them, or create them, we're going to be passing a representation of that resource at a particular point in time. Now, when you build services this way, the assumption is that HTTP can provide you with all the necessary features that you need around security, and scalability, and it has proven to be a very successful design pattern used throughout the web in a wide variety of web applications, and highly scalable web services.

Why WCF? The most fundamental reason is that it's bound to increase your developer productivity. Thanks to WCF's unified programming model, developers only have to worry about a single programming model for writing all their communication logic. Another reason is that if you move to WCF, you will immediately increase your interoperability potential. Like we discussed, WCF is capable of doing basic web services, and all the way up to advanced SOAP and WS-* communications or REST, giving you a wide range of communication options on the wire. WCF also provides increased flexibility, not only in terms of the communication, but also in terms of being able to plug in your own code.

PayPal [https://www.paypal.com/] is an online banking service. As one of the oldest and most recognized names in the internet banking business, PayPal enjoys widespread acceptance as a payment medium for all kinds of online transactions. In addition to basic online payment services it offers additional payment services via mobile phones, browser add-ons, and more. Because it is so easy to use, PayPal is the favorite of millions of amateur sellers and buyers around the world. PayPal doesn't fundamentally change the way merchants interact with banks and credit card companies, it just acts as a middleman. PayPal promotes its presence as an extra layer, as a security feature, because everyone's information, including credit card numbers, bank account numbers and address, stays with PayPal. With other online transactions, that information is transmitted from the buyer to the merchant to the credit card processor. For buyers who want to make payments online, the PayPal platform is a financial gateway that allows simple, private, and secure payments on a large number of websites. Unlike paying directly with credit cards online, paying with PayPal does not require sharing sensitive information such as credit card numbers on merchants' websites and simplifies For vendors who want to sell products online and collect payments electronically, the PayPal platform is a payment processing solution that allows them to accept secure online payments from customers in over 190 markets and in 19 currencies. Unlike online solutions provided by traditional merchant accounts, PayPal's product can be set up.

The PayPal Windows 8 Checkout SDK gives the ability to integrate PayPal payment functionality into the apps created for Windows 8 Store and Windows 8 Phone. Windows 8 Checkout SDK uses Mobile Express Checkout for its underlying technology, and it offers the same functionality with the same pricing model. PayPal's Mobile Express Checkout (MEC) gives the ability to place a Check Out with PayPal button on a mobile website/application, which enables customers to check out via their PayPal accounts. MEC is based on Express Checkout. It gives mobile users a streamlined checkout process—they don't have to enter their banking or shipping information into their mobile devices because this information is contained, and shielded, in their PayPal accounts. In addition, MEC is flexible in that it can give customers without PayPal accounts the option to check out using their credit or debit cards.

Why PayPal? In order to motivate our choice, we will compare PayPal with another online payment system, Fortumo. Fortumo allows any merchant to set up payment processing for web and mobile services, games or

apps. Users with a mobile phone are then able to make one-click payments using Fortumo without the need for a credit card-payment are charged to their mobile operator bill instead. Fortumo payments are cross-platform and work in PC applications, web services and HTML5, Android, Windows Phone and Windows 8 apps. Fortumo offers self-service setup with no monthly fees or minimum volume commitments, allowing any developer regardless of size and location to sign up for an account and get started with mobile payments. Fortumo In-App Payments for Windows 8 and Windows Phone is designed to be extremely simple and straightforward. The payment flow uses Premium SMS messages and HTTP requests to process payments. Payment is considered to be successful after your application has received payment confirmation from Fortumo server, and virtual goods can be granted to the end-user. From this, we can conclude Fortumo is a viable and promising option for our solution, but the disadvantage of Fortumo, that ultimately gave PayPal the upper hand, is the fact that all transactions are billed to the mobile operator. For our current needs, this is not an option, because many people have a prepaid SIM card, and overlooking that side of the market would be a bad choice.

Entity Framework. Entity Framework is an Object Relational Mapper (ORM). ORMs are aimed to increase developer productivity by relieving of the tedium and redundant task of persisting the data that you use in your applications. Taking advantage of its default behaviors, Entity Framework can generate the necessary database commands for reading or writing data and execute them for you in the database. If you're querying, you can express your queries against your own domain objects using LINQ to entities. Entity Framework will execute the relevant query in the database and then materializes results into instances of your domain objects for you to work within your application. So you focus on your domain and Entity Framework takes care of the database work. Most ORMs typically map domain types directly to the database schema. Entity Framework has a more granular mapping layer so you can customize mappings, for example, by mapping the single entity to multiple database tables or even multiple entities to a single table. Microsoft recommends that you use Entity Framework over ADO.NET or LINQ to SQL for all new development.

Entity Framework lets you focus on your business domain instead of database development. With Entity Framework, the focal point is referred as a conceptual model. It's a model of the objects in your application, not a model of the database you use to persist your application data. Your conceptual model may happen to align with your database schema or it may be quite different. You can use a Visual Designer to define your conceptual model which can then generate the classes you'll ultimately use in your application. Or you can just define your classes and use a feature of Entity Framework called Code First, and then Entity Framework will comprehend the conceptual model. Either way, Entity Framework is able to work out how to move from your conceptual model to your database. So you can query against your conceptual model objects and work directly with them.

Entity Framework vs. traditional ADO.Net. The highlights are that you can write code against the Entity Framework and the system will automatically produce objects for you as well as track changes on those objects and simplify the process of updating the database. The EF can therefore replace a large chunk of code you would otherwise have to write and maintain yourself. Further, because the mapping between your objects and your database is specified declaratively instead of in code, if you need to change your database schema, you can minimize the impact on the code you have to modify in your application, so the system provides a level of abstraction which helps isolate the application from the database Finally, the queries and other operations you write into your code are specified in a syntax that is not specific to any particular database vendor. ADO.NET, prior to the EF, provided a common syntax for creating connections, executing queries and processing results, but there was no common language for the queries themselves; ADO.NET just passed a string from your program down to the provider without manipulating that string at all, and if you wanted to move an application from Oracle to SQL Server, you would have to change a number of the queries. With the EF, the queries are written in LINQ or Entity SQL and then translated at runtime by the providers to the particular back-end query syntax for that database

Entity Framework vs. nHibernate. The big difference between the EF and nHibernate is around the Entity Data Model (EDM) and the long-term vision for the data platform we are building around it. The EF was specifically structured to separate the process of mapping queries/shaping results from building objects and tracking changes. This makes it easier to create a conceptual model which is how you want to think about your data and then reuse that conceptual model for a number of other services besides just building objects. So the differentiator is not that the EF supports more flexible mapping than nHibernate, it's that the EF is not just an ORM, it's the first step in a much larger vision of an entity-aware data platform.

Why EF? While Microsoft provides Entity Framework support in the SQL Server provider, there are many

third party providers that allow to you to use Entity Framework with a variety of databases, from commercial databases like Oracle, to Open Source databases like MySQL and Firebird. Entity Framework is able to keep track of changes made to objects that it's aware of, including adding new objects or deleting some. If you're designing disconnected apps where Entity Framework won't be around at the time the objects are actually being edited, there are patterns you can follow to let Entity Framework know what changes it should be aware of when you pass the objects back, for Entity Framework to save back to the database. Entity Framework has a variety of ways that it supports toward procedures and a complex API that let's you have granular control over everything from its modeling to its runtime behavior.

AutoMapper [https://automapper.codeplex.com/]. Many times we need to map objects between different application layers, we need to translate data from one object type to another. Common examples include DTOs (Data Transfer Objects), View Models, or even just some request or response object from a service or Web API call. The solution to these problems is to create some mapping mechanism with minimal configuration that we can use in a fairly generic way in our code, but thankfully, we don't have to actually do that: third-party libraries exist that we can lean upon, specifically, AutoMapper.

Why AutoMapper? You can write extension methods to move data between properties (perform the mapping manually), but why write countless mapping lines just for shipping data across types? AutoMapper will automatically move data from your business entities into the corresponding properties on your DTO, saving you many, many lines of repetitious assignment statements. It also means that if you add a new property to your DTO, AutoMapper will automatically pick up the corresponding property from the entity; no further changes to the code are required. AutoMapper provides all the convention you need to get the simple cases done for free, and simple patterns for managing the more complex scenarios where you are reducing shapes or changing types.

NDEF Library for Proximity APIs/NFC [https://ndef.codeplex.com/]. NFC tags and the content sent in device-to-device communication when tapping two phones is based on certain standards by the NFC Forum (called NDEF-NFC Data Exchange Format). When it comes to storing data on NFC tags that can have as little writable storage as around 40 bytes, very efficient and complex data storage schemes are necessary. The downside is that most operating systems do integrate the NFC data transmission at the base level, but offer developers very little support for the NDEF standards on top. Obviously, creating an own implementation of a message that stores a simple URL on a tag for example, is not an easy job.

The open source NFC/NDEF Library contains a large set of classes that take care of formatting data according to NDEF standards, so that these can be directly written to NFC tags or sent to other devices. The library creates an NDEF message out of the data, which you can directly send to the NFC stack in your operating system as a byte array, which takes care of *writing it to a tag or publishing it to another device*. Additionally, the library can parse NDEF byte arrays that you read from tags or receive from other devices and create a list (NDEF Message) of data classes (NDEF records) that you can easily analyze and use in your application. For Windows Phone 8, the NFC stack is represented through the Proximity APIs-they encapsulate NFC hardware communication and basic NDEF formatting for a very limited subset of the NDEF standards. This missing part is added by this NDEF library. The NFC/NDEF library is written in C# and can therefore be used on any operating system that supports C# development. The library is available as a ready-made portable class library, which can be used on the Windows 8 platform, as well as on Windows Phone 8. Both platforms provide support for interacting with the NFC hardware through the Proximity APIs.

3.3. System Overview

In this section, we describe the system and present its architecture, and then we will break it into its main components and discuss each component's functionality, how it accomplishes its goals and what are some strengths and weaknesses. Before diving into the system's architecture, we need to understand how the system's components will interact. In **Figure 1**, we present the involved entities in the processes of validating a ticket, buying tickets, etc.

The ticketing application is installed on the NFC-enabled device and it communicates with the application server and the payment server through an active Internet connection (mobile data). The device also communicates with the validator (tag) from the bus/tram/trolleybus, through NFC. When buying tickets or a monthly pass, or when validating a ticket using the application installed on the NFC-enabled mobile device, the result of each of these operations is stored in the database, on the application server. Furthermore, the tag writing application,

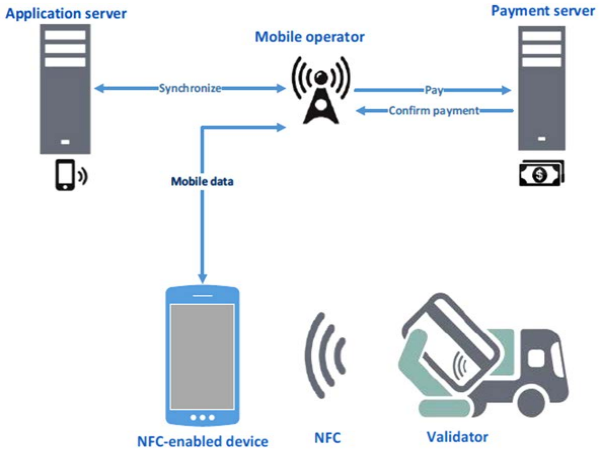


Figure 1. System overview.

also installed on the NFC-enabled device, needs to communicate with the application server to retrieve the data that will be written to the tag, and this communication is performed in the same manner, through an active Internet connection. The tag writing application will write the tag using NFC.

As a conclusion, we can establish that our system requires NFC-enabled devices with an active Internet connection, NFC configurable tags and an application server to persist our data. The payment service will be provided by a third party.

3.4. System Architecture

For the needs of our application, a classic client-server architecture will suffice, without adding any overhead. The major components of the system implementing this architecture are shown in **Figure 2**. We can observe this is a modular architecture which represents a strong point for reusability and maintainability. The major downfall of this architecture is its single point of failure: the application server. If the application server is offline, then the whole system is impaired, but yet this is a classic problem in the client-server architecture and can be resolved by adding a backup server. The backup server will kick in when the main server is down.

An issue that is generated by the solution to the first problem is that we need to keep the data from the two servers synchronized. This is a subject that we will not discuss, since it is not in the scope of our project. The next major challenge we need to consider is the performance of the system. The application server represents the bottleneck of the system, because assuming multiple concurrent users access the server simultaneously. This will be reflected in the performance of the system. The solution to the single point of failure problem can help us also to improve the performance, because by having multiple server we can use a load balancer to share the requests to the servers, thus minimizing the stress on each server. We will shortly describe how each individual component communicates with the others, and after that, we will see the implementation details of each of these components.

We begin by showing how the *tag writing application* communicates with the application server and the tag.

The purpose of the tag writing application is to configure the NFC tag with the needed data from the server. The communication with the server is done via an Internet connection (mobile data or WiFi), and the communication with the tag is done via NFC.

The purpose of *the ticketing application* is to provide the traveler with tickets and monthly passes in an electronic format, along with the ability to use these tickets in an intuitive way. The device on which the ticketing application is installed needs to be tapped against the tag in order to validate a ticket. When this tap occurs, the data from the tag, containing information about the bus the user is currently traveling on, is transferred to the NFC-enabled device via NFC, which in turn sends this information to the application server for further processing. The communication of the ticketing application to the application server is also performed via Internet.

To conclude, the ticketing application uses the tag for read operations only, while the tag writing application uses the tag for write operations only. The tag writing application only retrieves data from the application server, while the ticketing application performs all CRUD operation on the database located on the application server.

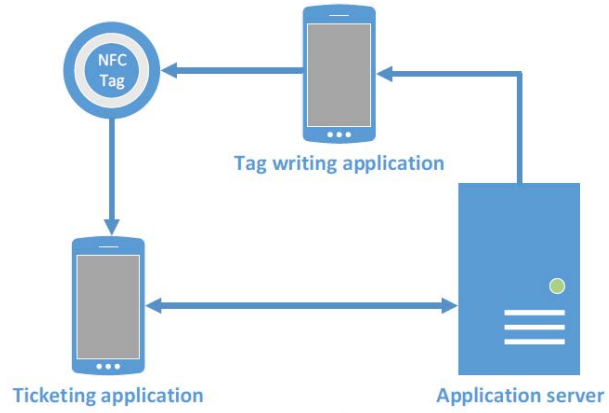


Figure 2. Conceptual architecture.

Application server. The application server exposes a web service implemented on the.NET Framework. The web service is implemented with Windows Communication Foundation (WCF). In order for the web service to be online and functional it needs to be hosted, which in our project is done via IIS Express. The architecture of the application server is a 3-tier architecture. For a better understanding of the application server, we will discuss all three tiers separately, but we will start with the detailed description of the Data Source.

Data Source. Our system is database-centric, hence the modeling of the database is crucial to our application. We need to model the database such that we can encapsulate change and provide maintainability. Furthermore, it is important to add new functionalities to our system with ease, thus we need to take this into account when designing the database.

Data Access Layer. The purpose of this layer is to ensure and implement the connection to the database. When building this layer, we had in mind the following: avoid duplicated code, decrease potential for programming errors and have a strong typing of the business data. To achieve this, we used a repository to separate the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer.

The repository mediates between the data source layer and the business layers of the application. It queries the data source for the data and persists changes in the business entity to the data source. A repository separates the business logic from the interactions with the underlying data source. The separation between the data and business tiers has the following benefits: it centralizes the data logic, it provides a flexible architecture that can be adapted as the overall design of the application evolves, improves the code's maintainability and readability by separating business logic from data or service access logic, uses business entities that are strongly typed so that we can identify problems at compile time instead of at run time.

The Repository Pattern has been implemented using Entity Framework. EF allows operations on generic types, hence the implementation of a generic Repository Pattern is natural and brings along reuse, extensibility, maintainability and avoids duplicate code. From the generic repository, several repositories were derived, which perform specific functionalities on the database. **Figure 3** shows the class diagram of the Data Access Layer.

Business Layer. The purpose of this layer is to encapsulate the logic of the application. This layer should be totally decoupled from any other layer such that it can be reused in other applications. This results in a cost efficient implementation. We accomplished this by totally encapsulating the Business Layer. The approach to do this was to create a set of Data Transfer Objects, which contain the necessary information needed by the upper layer. By using AutoMapper, we created a component in this layer that handles the mapping between the Data Transfer Objects and the entities objects returned by Entity Framework. This component should not impact memory performance, nor CPU performance (it has no functional value), hence it was implemented as a static component. The Business Layer needs to access the Data Access Layer in order to retrieve information from the Data Source. By looking at the current implementation of the Data Access Layer, we can infer that a generic DataDervice is in order. This approach allows us to reduce duplicate code in this layer, and makes this component flexible, maintainable and reusable.

Resembling the Repository implemented in the Data Access Layer, the DataService is built in a similar man-

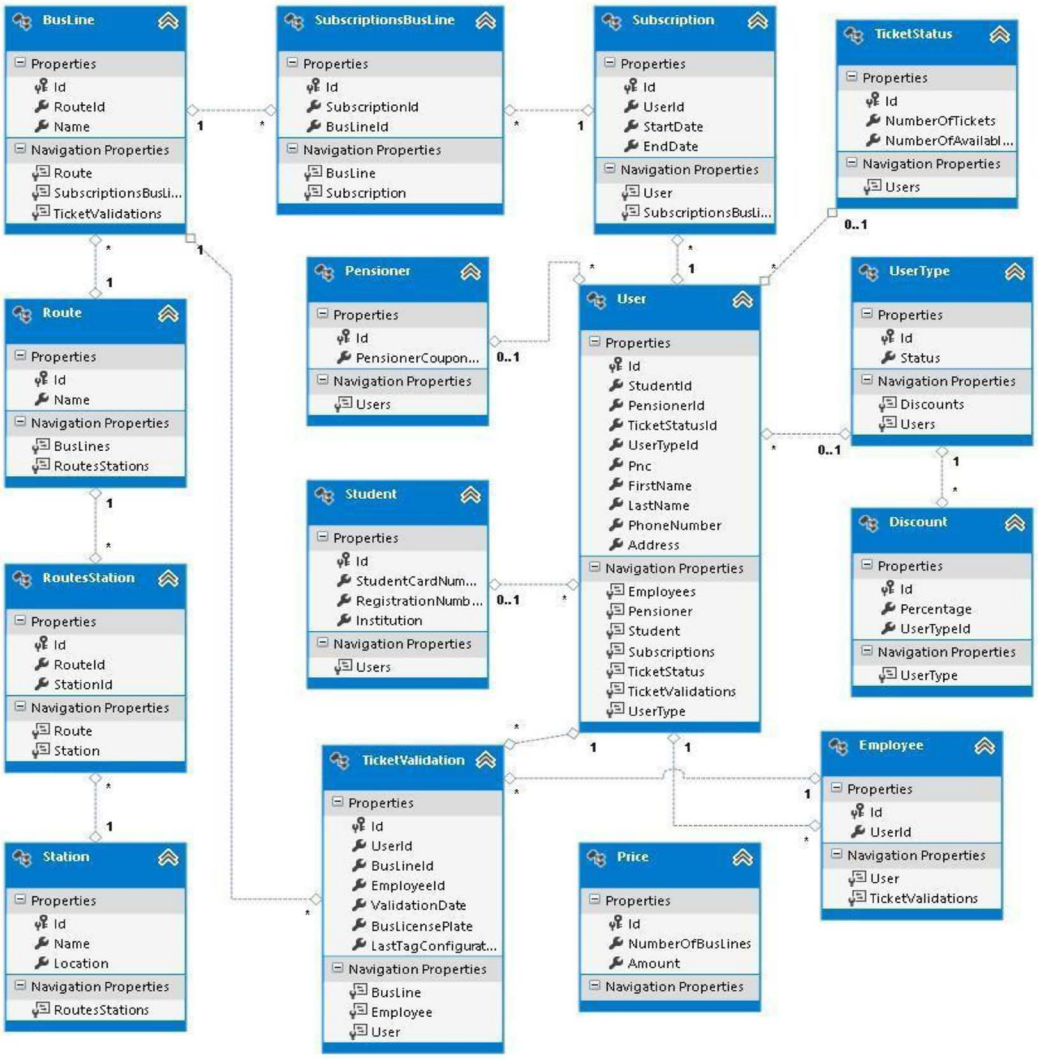


Figure 3. Data model.

ner. The differences are that the Business Layer has as data source the Data Access Layer and that it works with Data Transfer Objects instead of entity objects. An aspect worth mentioning here, is the use of the dynamic type provided by the.NET Framework The dynamic type enables the operations in which it occurs to bypass compile-time type checking. Instead, these operations are resolved at run time. Type dynamic behaves like type object in most circumstances.

However, operations that contain expressions of type dynamic are not resolved or type checked by the compiler. Variables of type dynamic are compiled into variables of type object. Therefore, type dynamic exists only at compile time, not at run time. This capability was needed for the following reason: because the DataService is a generic class and all of its methods revolve around the generic parameter T, at compile time we do not know the type of T. This implies we do not know how to instantiate our repository, which is done by providing an actual type. To solve this, we declare the Repository as a dynamic type. If we were to declare it as type object, VisualStudio's IntelliSense would alert us that the object does not have the specified methods. Instead, if we declare the Repository as dynamic we avoid this checking. The Repository is instantiated in the constructor of the Data Service, when the type of T can be found.

The way we managed to instantiate the Repository (shown in the above snippet code) with the appropriate type is the following:

1) We decorate each Data Transfer Object with a CustomAttribute previously implemented, which basically

specifies the type of the respective class as a string.
2) We retrieve the MemberInfo object of the type of T (remember, T is known when the method is executed), which discovers the attributes of a particular member and provides access to member metadata.
3) We retrieve our Custom Attribute from the MemberInfo.
4) We find the type of the T entity by using the string representation of the type and the name of the assembly where this type is declared.
5) We retrieve the type of the Repository<> class.
6) We create an array of types, to which we add the type of the entity found in step 4.
7) We call the make generic type method of the type class on the type retrieved in step 5, which substitutes the elements of the array of types for the type parameters of the current generic type definition and returns a Type object representing the resulting constructed type.
8) We finally create the Repository by calling the Create Instance method of the activator class, which creates an instance of the specified type using the constructor that best matches the specified parameters.

An important difference between the Data Access Layer and the Business Layer is that the Business Layer needs to be able to accommodate any new business rule. To do so, we need to just simply derive from the generic Data Service.

Web Service. The Web Service's role is to expose the functionality from the Business Layer to the other components of the system. When implementing the functionality of the Web Service, we need to be careful to not include any business logic, because we might end up compromising the maintainability of the entire system, *i.e.* it will be hard to replace the Web Service with other components, such as a Web API. The Web Service was implemented using Windows Communication Foundation. This implies the declaration of the service contract and the data contract, which will be used in generating the Web Service proxies.

The layer represented by the Web Service communicates directly with the business layer and establishes a composition relationship with a series of wrappers that encapsulate the Data Service and holds business logic. These wrappers are the place where the business rules meet the information retrieved from the Data Service, and can be easily manipulated. In **Figure 4**, we emphasize the relationship between the Web Service and the Business Layer. We can see that the web service accesses only the Business Layer classes and does not perform any logical operations, just method calls.

3.5. Ticketing Application

The purpose of this component is to offer the traveler all the functionality he/she requires, meaning the ability to buy tickets and monthly passes, to validate tickets, to consult the application for bus related information (bus lines) and to display an overview of his/her validations and subscriptions. The architecture of this component is based on the Model-View-View Model architectural pattern. This is a best practice recommended by Microsoft for building Windows Phone 8 applications.

The Model-View-View Model (MVVM) pattern helps to cleanly separate the business and presentation logic

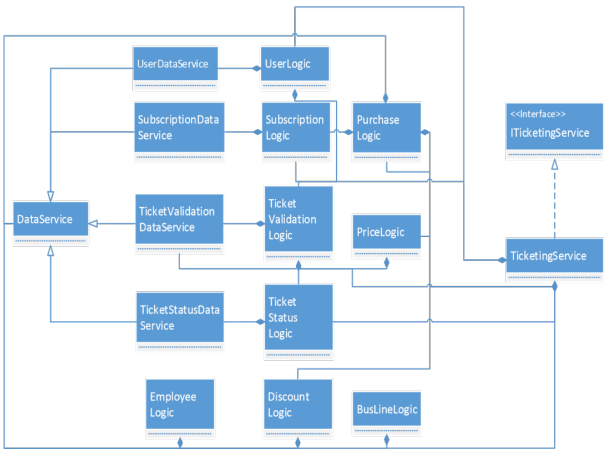


Figure 4. Web service relationship with bussiness layer.

of the application from its user interface (UI). Maintaining a clean separation between application logic and UI helps to address numerous development and design issues and can make the application much easier to test, maintain, and evolve. It also improves greatly code reuse opportunities and allows developers and UI designers to collaborate more easily when developing their respective parts of the application. The diagram in **Figure 5** shows the basic architecture of an application built using the MVVM design pattern.

The MVVM is a way to separate the data from the UI. In the MVVM design pattern, developers can code application logic, and designers can create the UI. XAML is designed to make it easy to build apps using MVVM, and they complement each other in a way that makes separation of UI and application logic a natural thing to do. Having a clear understanding of the MVVM pattern, the way the ticketing application was implemented becomes more clear and easy to understand.

In what follows, we will detail the way MVVM was used in our application, by presenting the ViewModels and their connection to the Views and Models, as well as the motivation behind this implementation.

When starting the application, the first page we come in contact with is the LoginPage. This page has a password box where the user needs to enter his/her Personal Numerical Code (PNC) in order to authenticate into the application.

If the user comes in contact with the application for the first time, he/she has the option to register, by pressing the “register” button from the application bar, which will redirect the user to the Registration Page. The class diagram in **Figure 6** shows the interaction between the Login View Model, the views associated with it (Login-

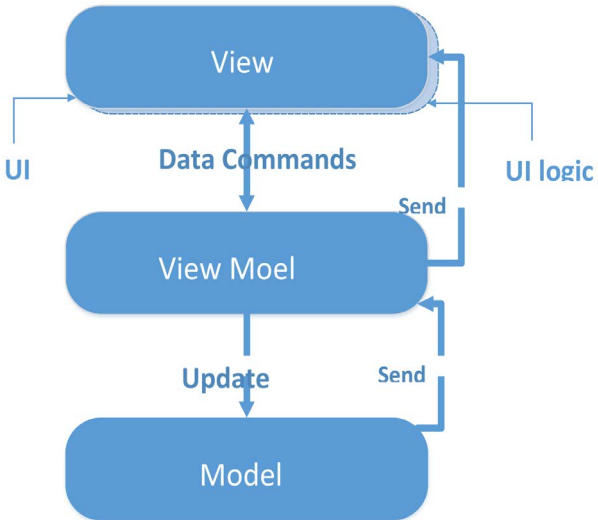


Figure 5. MVVC architectural pattern.

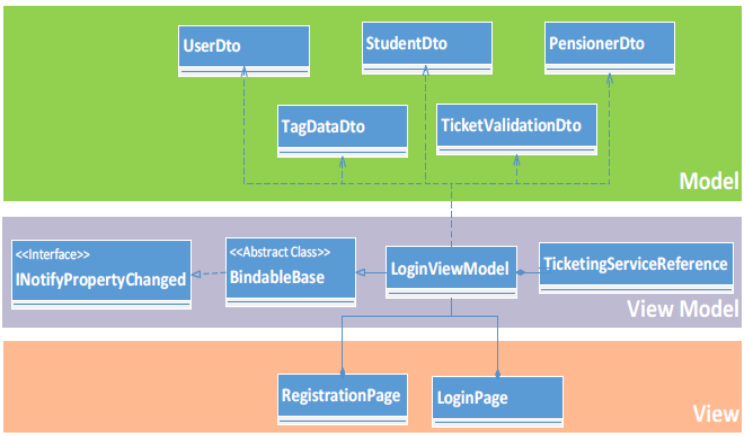


Figure 6. Login View Model interaction.

Page and Registration Page) and the Models it uses. The Login View Model inherits the Bindable Base abstract class, which is an implementation of the INotify Property Changed interface. This is needed in order for the View Model to notify the View when changes occur.

What is worth mentioning in the **Login View Model** (and also the other View Models), is the fact that it uses commands and bindings to have a clear separation of the UI from the business logic. This is done by data binding and the use of the ICommand interface.

The ICommand interface was implemented by the Command class, and the command is binded in the XAML markup language to specify the method a certain UI action should trigger. The Login View Model gets the information it displays to the View by making asynchronous calls to the Web Service, through the TicketingService Reference.

The other View Model in our solution, is the **Main View Model**. This View Model is common for the rest of the Views, due to the fact they all require the same or related information. In **Figure 7** are presented the interactions between the Main View Model, the views associated with it, the involved Models and third party libraries.

The same principle of data binding and commands is applied throughout these pages. What is worth mentioning is the Buy Now class from the PayPal Checkout SDK. The PayPal Checkout SDK exposes the BuyNow class, which contains some properties like the MerchantId, Use Sandbox (if this is set to true, the test environment is used), Payment Method etc., and some event handlers to handle the following events:

- Auth Event Args (triggered when the payment is being authenticated).
- Cancel Event Args (triggered when the payment is canceled).
- Complete Event Args (triggered when the payment is complete).
- Error Event Args (triggered when an error occurs while processing the payment).
- Start Event Args (triggered when the payment is initiated).

The most important method that concerns us is the *IAsyncOperation<bool>Execute*(). This method is executed separately on a different thread asynchronously with respect to the main thread. This is a best practice when dealing with third party services and we do not have a guaranteed response time, or simply the operation is too expensive to execute it synchronously. Without doing this, we would leave the UI unresponsive until the method returns.

The return result of the IAsync Operation is handled via a generic event handler, that returns the needed information. When the method finishes execution, one of the previous events are triggered, based on the execution state of the method.

3.6. Tag Writing Application

This application's whole reasons of being is to provide a way to configure tags with the appropriate data. This application is implemented as a Windows Phone 8 application and thus it respects the Model-View-View Model

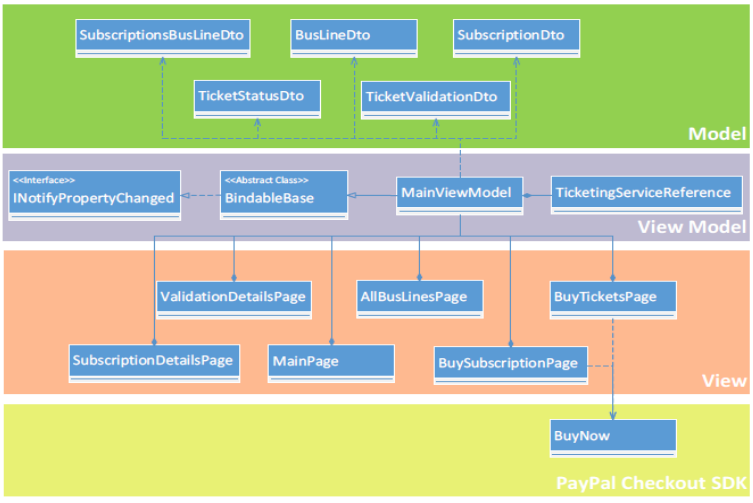


Figure 7. Main View Model interactions.

architectural pattern and Microsoft's recommended design principles.

This application also uses a Ticketing Service Reference to make calls to the Web Service. The data required by this application are: the list of all bus lines, the list of all employees

As discussed previously, NFC tags content is based on the NDEF message standard (NFC Data Exchange Format). In order to use these capabilities, we needed to use the Windows, Networking, Proximity namespace, referencing the Proximity API. By using this API, the application can establish a connection through a tap within wireless range. The Proximity API encapsulates NFC hardware communication and basic NDEF formatting for a very limited subset of the NDEF standards. This missing part, the support for the NDEF standards on top,is added by the NDEF library.

After the data needed to configure the tag is retrieved from the view (it is inputed by the user), we save it in a TagDataDto object. This class does not have a corresponding entity in the database. In order to send this data to the NFC tag, we must transform it in a format known by the NFC tag. For the purpose of our system, we need the ticketing application to be opened when the user taps the NFC-enabled device to an NFC tag. We achieve this by using the Ndef Launch App Record class provided by the NDEF library. The NdefLaunch App Record is a special type of record that will open an application, based on the application ID specified in the record, when a NFC connection is established. This class contains several properties, but the property that concerns us is the "Arguments" property. This property is of type string and we will use it to send the data we need to the tag. We do this by serializing the Tag DataDto object with the help of a JSON serializer. The Tag Data Dto class is marked with the Data Contract attribute and all the members that we want to take part in the serialization are marked with the Data Member attribute.

In **Figure 8**, the interactions between the classes of the tag writing application can be observed.

4. Testing and Validation

The purpose of this chapter is to offer an overview of the test suite. For our project, a test suite has been built for the most important functionalities of our ticketing application. A test suite is composed of multiple test cases. A test case provides the functional knowledge for an application and contains the preconditions of the test case, *i.e.* in what state should the system be in such that the desired functionality is available, all the necessary steps to be performed in order for that functionality to reach its goal, an expected result for each step of the test case, and finally, the expected result of the test case, *i.e.* the goal of the test case. It is important to start developing test cases and test suite early in the application life cycle because it increases maintainability, is cost efficient and we can create Smoke and Regression testing based on which we can start automating the testing process. Smoke testing is a set of basic cheap to run tests that precede actual testing. It aims to verify that the build is deployed

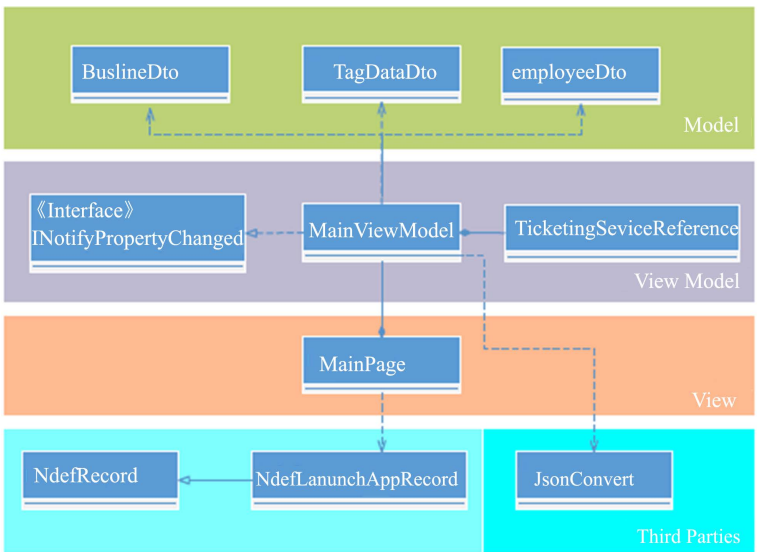


Figure 8. Tag writing application class diagram.

successfully and that all test environment aspects are running and ready for the actual test process. It saves you bringing the full extent of your testing wrath down a faulty build and just realizing that you have been testing on a bad environment or erroneously deployed build possibly too late.

During a regression test, we run through the application testing features that were known to work in the previous build. We look specifically for parts of the application that may not have been directly modified, but depend on (and could have residual bugs from) code that was modified. Those bugs (ones caused by bugs in dependent code, even though they were working before) are known as regressions (because the feature was working properly and now has a bug and therefore, regressed). In what follows we will present two most important test cases for the application.

Buy Tickets Test Case

Preconditions: User is logged in, main page is displayed

Step	Expected result
Expand application bar	3 buttons are shown: all bus lines, buy tickets, subscribe
Press button Buy tickets	The ticketing page is displayed, showing the available tickets and the price of one ticket
Select the number of tickets you want to buy	The selector shows the number of tickets that the user wants to buy. Total price is updated by multiplying t number of tickets desired with the price of a ticket
Press button pay now	The application is redirected to the official PayPal webpage.
Enter PayPal credentials	The form is filled
Press button Login	The application is redirected to a cost overview page where detailed information about the transaction is displayed
Press button continue	The account balanced is modified accordingly. The application redirected to the Ticketing page. A pop is displayed showing the transaction id

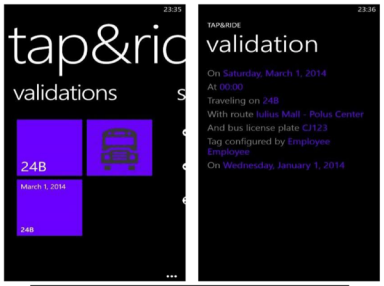
Subscribe Test Case

Preconditions: User is logged in, main page is displayed

Step	Expected result
Expand application bar	Three buttons are displayed: all bus lines, buy tickets, subscribe
Press subscribe button	The application is redirected to the BuySubscription page where a selectable list containing all bus lines is available and a date picker
Select the start date of the subscription	The end date is computed and displayed
Select the line(s) for the subscription	The total price is computed with respect to the number of lines selected
Press pay now button	The application is redirected to the official mobile PayPal webpage
Enter PayPal credentials	The form is filled
Press login button	The application is redirected to a cost overview page where detailed information about the transaction is displayed
Press continue button	The account balance is modified accordingly. The application redirects to the BuyTickets page. A pop-up is displayed showing the ID of the transaction

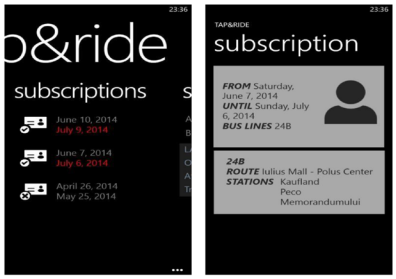
The application can be downloaded from the Store by manually searching for it. The OS can also ask you if you want to download the application from the Store if you previously tapped an NFC tag configured for our application and the application was not installed on the device.

Another way to install the application is to deploy the executable file, the.xap file to the device, by using the Application Deployment tool, a stand-alone application that was installed along with the Windows Phone SDK.



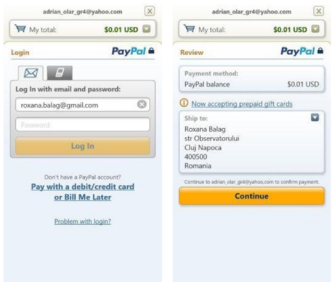
4.1.Details of a ticket validation

From the main page, the validations pane, tap a ticket validation (represented by a tile). The application will redirect the user to a page displaying the details of the tapped validation.



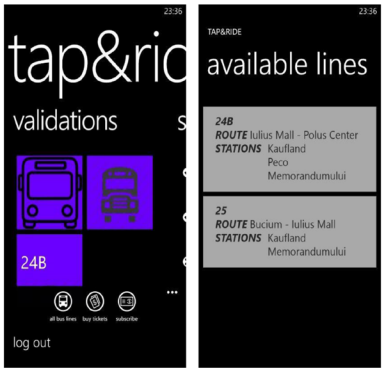
4.2.Details of a subscription

From the main page, the subscriptions pane, tap a subscription. The application will redirect the user to a page displaying the details of the tapped



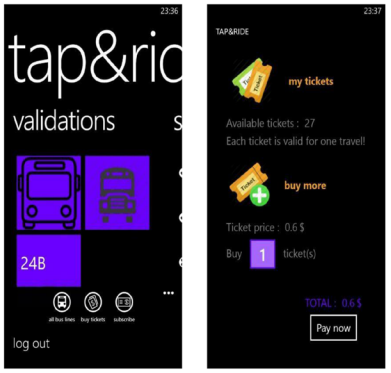
After pressing "Pay Now" the application is redirected to a PayPal page where the user needs to enter his PayPal credentials to login into his PayPal account.

After confirming the payment by pressing "Continue", the application will redirect the user to the buy tickets page and show a pop-up with the ID of the transaction.



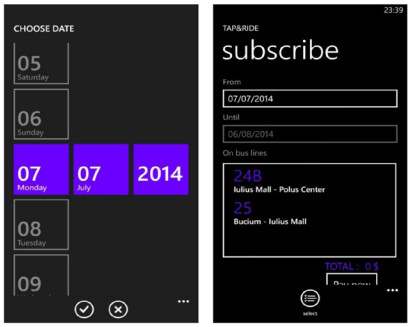
4.3.View the available bus lines

From the main page, open the application bar and press the all bus lines button. The application will redirect the user to another page where all the available bus lines are shown.



4.4.How to buy tickets

From the main page, open the application bar and press the buy tickets button. The application will redirect the user to the buy tickets page from where the user can select the number of tickets to buy along with other information.



How to buy a monthly pass (subscribe)

From the main page, open the application bar and press the subscribe button. The application will redirect the user to the subscribe page.

Tap the first date picker, select the start date of the monthly subscription and press save.

From the application bar press the select button to start selecting bus lines. The buttons in the application bar will allow you to select all and deselect all.After pressing "Pay Now" the application is redirected to a PayPal page where the user needs to enter his PayPal credentials to

5. Conclusions and Further Developments

In this chapter, we will present some conclusions regarding the developed system. In the first part, we will present an overview of our achievements, followed by an analysis of the obtained results and finally we will talk about future development and improvements.

This project has a lot of place for improvement and future development. We suggest some future development work that can be done:

- Encrypt the sensitive data, like the user's personal information. This will enhance the system's security. A possible encryption algorithm is AES (Advanced Encryption Standard).
- Add offline capabilities to the system. The user should be able to use the system even when an Internet connection is not available. This can be done by possibly storing data on the device and when an Internet connection becomes available, synchronize all the data with the application server.
- Add multiple payment methods. A possibility would be to use SMS-based payment, or implement an API like Fortumo, which makes payments to be charged to the user's mobile operator bill. Possibilities here are countless.
- Manage the lifecycle of the application. Implement a Suspension Manager class that handles the changing of states in the application's lifecycle.
- Add extra validation rules on the inputs provided by the users. Constrain PNC to be numerical.
- Add more business rules. Give the user the option to subscribe for passes on wider range of periods, varying from weeks to even trimesters and on a number of bus lines of his willing, not restricted to one, two, three or all bus lines.
- Offer discounts based on the purchases performed by the user. A fidelity program is a nice idea to keep people interested and attracted to the application.
- Offer the user the possibility to edit his account. The user should be able to see and edit his personal information.

The performance of the system was not rigorously evaluated, due to the fact that until now, the system was tested with a small amount of data and no concurrent users. Under the mentioned conditions, all the system operations did indeed take under 5 seconds.

References

[1] Mut-Puigserver, M. and Payeras-Capella, M. (2012) A Survey Of Electronic Ticketing Applied to Transport. *Computers & Security*, **31**, 925-939. http://dx.doi.org/10.1016/j.cose.2012.07.004

[2] Public Transport ITS Committee, White Paper on the Application of NFC Technology in Public Transport, Asoc. (2013) Foro de NuevasTecnologíasen el Transporte, ITS España. C/ Serrano, 216-1ºdcha. 28016 Madrid, 1st Edition. http://www.fomento.gob.es/NR/rdonlyres/F5BBB37E-F29E-4C47-AE7B-77C3875CAC92/122698/White_Paper_NFC.pdf

[3] Tuikka T. and Isomursu M. (2009) Touch the Future with a Smart Touch. VTT Tiedotteita-Research Notes 2492, 280 p. http://www.vtt.fi/inf/pdf/tiedotteet/2009/T2492.pdf

[4] Stroh, S., Schneiderbauer, D. and Kreft, C. (2007) Next Generation eTicketing. http://www.boozallen.com/media/file/Next_Generation_eTicketing.pdf

[5] European Parliamentary Research Service (2014) Integrated Urban e-Ticketing for Public Transport and Touristic Sites. January 2014. http://www.europarl.europa.eu/RegData/etudes/etudes/join/2014/513551/IPOL-JOIN_ET(2014)513551_EN.pdf

[6] NFC Forum (2006) NFC Forum Technical Specifications. http://members.nfc-forum.org/specs/spec_list/

[7] GSMA (2014) The Value of Mobile Commerce in Transport. White Paper, February 2014. http://www.gsma.com/digitalcommerce/wp-content/uploads/2014/01/GSMA-The-Value-of-Mobile-Commerce-in-Transport-Feb14.pdf

[8] Dobson, B. (2008) NFC in Transport for London. Setting a Context for the Role of NFC Technology in TfL's Ticketing Strategy. http://67.222.41.204/wp-content/uploads/2013/12/Brian-Dobson-Transport-for-London.pdf

[9] Burkard, S. (2012) Near Field Communication in Smartphones. https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/nfc-in-smartphones_burkard.pdf

[10] Desai, A. and Shajan, M.G. (2012) A Review on the Operating Modes of Near Field Communication. *International Journal of Engineering and Advanced Technology* (*IJEAT*), **2**, 322-325.

http://www.ijeat.org/attachments/File/v2i2/B0956112212.pdf

[11] Liebenau, J., Elaluf-Calderwood, S., Karrberg, P. and Hosein, G. (2011) Near Field Communications; Privacy, Regulation & Business Model. London School of Economics, October 2011. http://www.lse.ac.uk/management/documents/LSE-White-Paper_-_Near-Field-Communications-Privacy-Regulation-Business-Models.pdf

[12] Kerschberger, M. (2011) Near Field Communication: A Survey of Safety and Security Measures. https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0156.pdf

[13] Van Anh Pham, T. (2013) Security of NFC applications, June 2013. http://nordsecmob.aalto.fi/en/publications/theses2013/thesis_pham/

[14] GSMA (2012) White Paper: Mobile NFC in Transport. September 2012. http://www.gsma.com/digitalcommerce/wp-content/uploads/2012/10/Transport_White_Paper_April13_amended.pdf

[15] Mezghani, M. (2008) Study on Electronic Ticketing in Public Transport. European Metropolitan Transport Authorities. http://www.emta.com/IMG/pdf/EMTA-Ticketing.pdf

[16] Whitechapel, A. and McKenna, S. (2012) Windows Phone 8 Development Internals. Microsoft Press, USA.