POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Science Engineering

Property Booking

A Web Platform Application

Supervisor:
Prof. Sara Comai

Master Thesis of:
Mohsen Faghfourmaghrebi, 894263

Academic Year 2020-2021

To my loving parents,

I salute you for believing in me and my dreams.

"Heav'n but the Vision of fulfill'd Desire,
And Hell the Shadow of a Soul on fire,
Cast on the Darkness into which Ourselves,
So late emerg'd from, shall so soon expire."

- Omar Khayyam, *Rubáiyát, Edward FitzGerald*

Acknowledgments

I would like to thank all of the professors of department of Computer Science of Politecnico Di Milano and specially professor Sara Comai for their immense support throughout my career as a student.

I would like to give my special gratitude to professor Piero Fraternali for building a foundation of knowledge without which this project would not have been possible.

Contents

List of Figures

Sommario

Gli andamenti globali indicano che il numero di utilizzatori di internet nel mondo ha ormai raggiunto la cifra di 4.54 miliardi di persone, con un incremento del 7 per cento (di cui 298 milioni di nuovi utenti) da gennaio 2019 fino a gennaio 2020. Inoltre, a gennaio 2020 sono stati registrati 3.8 miliardi di utilizzatori di social media, cifra che ha avuto un incremento del +9% (321 nuovi utenti) rispetto allo stesso mese dell'anno precedente. In totale, più di 5.19 miliardi di persone utilizzano al giorno d'oggi telefoni cellulari, avendo avuto un incremento di 124 milioni (2.4%) di nuovi utilizzatori rispetto all'anno precedente.

L'impatto dei sopra citati dati hanno dato una grossa spinta alla crescita di un altro settore, quello delle piattaforme digitali. Una piattaforma digitale può essere definita come "un blocco fondamentale che provvedere una funzione essenziale ad un sistema tecnologico e serve da fondamento sopra il quale altri prodotti complementari, tecnologie e servizi possono essere costruiti".

Le industrie che hanno fatto leva sulle piattafrome digitali hanno raggiunto una significante crescita di dimensioni e scala. Per esempio, le piattaforme digitali che operano nelle aree dell'e-commerce e dello sviluppo software hanno superato i 700 miliardi di dollari di valore di mercato. In aggiunta la crescita delle piattaforme digitali ha trasformato il paesaggio di molte industrie come per esempio quella dei trasporti (Uber, Grab), dell'ospitalità (Airbnb, CouchSurfing), e sviluppo software (Apple iOS, Google Android).

Questo report riassume come il processo del design di una piattaforma digitale assuma i connotati dello stesso processo di design che ha l'ingegneria del software. Nel primo capitolo vi sarà una breve discussione della definizione delle piattaforme digitali e delle loro tipologie per poter dare forma al contesto nel quale il nostro progetto si sta sviluppando.

Poichè questo progetto è il riassunto di uno stage, ne vedremo gli stadi di design e sviluppo così come le metodologie e tecnologie che sono necessarie per lo sviluppo di una piattaforma digitale quale è stata il nostro progetto.

Abstract

Global trends indicate that the number of people across the world using the internet has grown to 4.54 billion, an increase of 7 percent (298 million new users) from January 2019 to January 2020. In addition, there are 3.80 billion social media users in January 2020, with this number increasing by more than 9 percent (321 million new users) since same month last year. Globally, more than 5.19 billion people now use mobile phones, with user numbers up by 124 million (2.4 percent) over the past year.

The impact of the aforementioned trends boosted the growth of another industry, the digital platforms. A digital platform can be defined as “a building block that provides an essential function to a technological system and serves as a foundation upon which complementary products, technologies, or services can be developed".

Enterprises that have leveraged the affordances of digital platforms have achieved significant growth in size and scale. For instance, digital platform providers in the areas of e-commerce and software development have attained more than $700 billion in market value. In addition the rise of digital platforms has transformed the landscape of multiple industries such as transportation (e.g., Uber, Grab), hospitality (e.g., Airbnb, CouchSurfing), and software development (e.g., Apple iOS, Google android).

This report summarises the process of designing a digital platform in hospitality sector from a software engineering perspective. In the first chapter we briefly discuss definition of digital platforms and typologies of such platforms for the purpose of providing some context about the software project we are trying to build.

Since this project is a summary of an internship stage body of work, we will discuss thoroughly design and development phases of the project as well as methodologies and technologies which are necessary to develop a digital platform such as our project.

Chapter 1

Introduction

GN Techonomy S.r.l. is a consulting company that, since 1995, offers technological and innovative enterprise solutions. GN techonomy is active in Information technology sector providing ERP (Enterprise resource planning) solutions as well as variety of customized software for its clients. The company's mission statement is to develop a lasting competitive advantage for its customers. I chose this organization because I find their mission to be important and relevant to my career goals. My role at GN Techonomy is JAVA solution architect. This report aims to summarise a project which was developed for a customer of GN Techonomy during a six months period of time.

1.1 Context

Platforms operate in two-or multi-sided markets with distinct groups of end users. [1] Their value lies in creating an interdependence between the different types of users in a way that facilitates transactions, thus improving the welfare of both. [2] Platforms operate online and offline, common offline examples are newspapers and shopping malls,while online ones include search engines and social networks. [3] Recent technological developments–notably computer software, the internet and smart phones have considerably expanded the scope for platforms to lower transaction costs. [4]

According to a study performed by JPMorgan Chase Institute [5] , there is a dramatic growth in number of individuals who are earning income from online platforms, such as Uber[1], TaskRabiit[2], or Airbnb[3]. In addition JP-Morgan&Chase researchers, also make a distinction between "labour" and "capital platforms", which they define as fallowing: *Labour platforms* such as Uber or TaskRabbit, often referred to as "Gig Economy" connect customers with freelancers or contingent workers who perform discrete tasks or projects, *Capital platforms* such as Ebay[4] or Airbnb connect customers with individuals who rent assets or sell goods peer to peer [5].

The result of their research indicates between October 2012 and September 2015, monthly participation in the Online Platform Economy grew 10-fold, from 0.1% to 1.0% (figure1.1). In addition over this three-year period, the cumulative participation grew from 0.1% of adults to 4.7%, a 47-fold growth (figure1.2).

Global trends indicate that the number of people across the world using the internet has grown to 4.54 billion, an increase of 7 percent (298 million new users) from January 2019 to January 2020. Globally, more than 5.19 billion people now use mobile phones, with user numbers up by 124 million (2.4 percent) over the past year [5]. In addition, according to a statistics Portal 2017 survey, 88% of Americans and 78% of French book their hotel using Internet [6].

[1]Uber Technologies, Inc., commonly known as Uber, is an American multinational ride-hailing company offering services that include peer-to-peer ridesharing, ride service hailing, food delivery, and a micromobility system with electric bikes and scooters

[2]TaskRabbit is an American online and mobile marketplace that matches freelance labor with local demand, allowing consumers to find immediate help with everyday tasks, including cleaning, moving, delivery and handyman work

[3]Airbnb, Inc. is an American online marketplace company based in San Francisco, California, United States. Airbnb offer arrangement for lodging, primarily homestays, or tourism experiences

[4]eBay Inc. is an American multinational e-commerce corporation based in San Jose, California, that facilitates consumer-to-consumer and business-to-consumer sales through its website

[5]https://datareportal.com/reports/digital-2020-global-digital-overview

[6]https://www.statista.com/statistics/666643/preference-of-online-or-offline-hotel-

The result of aforementioned study helps us classify our digital platform. An online booking digital platform belongs to the category of *Capital platforms*. In addition as the study suggests, it is evident that the number of adults participating in digital online platforms economy is increasing rapidly year by year. Considering the fact that the number of people across the world using the internet has increased by 7 percent (298 million new users) from Jan 2019 to January 2020 [7] we can confidently assume, existence of a tremendous potential for gaining profit from online platforms by means of taking advantage of the growing user base in order to monetize the platform for interested investors. In this report we will discuss the structure of such platforms in form of an online property booking system from a software engineering perspective.

1.2 Problem Statement

Online booking platforms such as AirBnB offer arrangement for a short period hospitality, primarily connecting hosts (renters) and guests (typically tourists, students, etc.). One observation is that, the services provided by AirBnB or similar websites is not by design a "premium experience", there are cases of high profile, VIP guests which need to rent a property for example a villa, a house or a mansion in its entirety for possibly an extended period of time, in some cases even months.

This opens up opportunity for another digital platform which can connect the high end, luxury seeking guests with property owners which do not intend to enlist their properties on low end platforms such as AirBnB. Providing premium user experience is the main differentiating factor which needs to be embedded in every aspect of our digital platform. In chapter 3 we will discuss some guidelines on how to craft a better user experience.

First part of designing any piece of software project is the identification of already existing business processes among different actors, e.g. agencies

booking-us/

[7]https://datareportal.com/reports/digital-2020-global-digital-overview

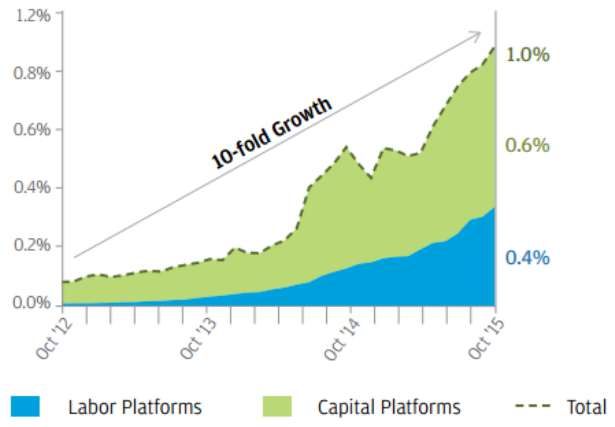


Figure 1.1: Year by year participation percentage
Graph illustrates percentage of adults participating in the Online Platform Economy in each month. Image courtesy JPMorgan Chase Institute[5].

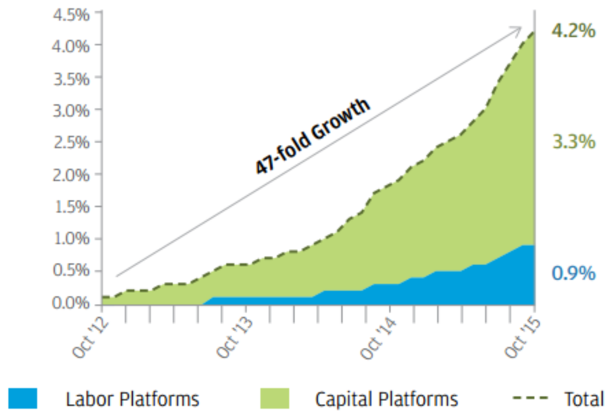


Figure 1.2: Year by year cumulative percentage
Graph illustrates cumulative percentage of adults who have ever participated in the Online Platform Economy. Image courtesy JPMorgan Chase Institute[5]

enlisting properties, Realtor, VIP guest, etc. For this part interviewing the investors of the platform and identifying their requirements and expectations is necessary.

The result of the interview between the project investor and software developer is as follows: A booking platform in essence should be capable of managing interactions between host(s) and guest(s). The host should be able to enlist his/her property on the platform specifying a period of availability and a set of criteria for the guests including number of guests and the price for the property. Alternatively the guest should be able to filter listed properties on the platform based on his/her preferences which includes price,location, desired period, etc. and obviously book the chosen property. Finally there is also an authority figure to manage conflicts between hosts and guests, this is defined as the platform administrator which in this project is defined as the platform manager.

After the identifying the scope and requirements of the project, we need to assess them in order to find out possible candidate technologies and methodologies, in order to design the software architecture solution which can firstly address the requirements, secondly speed up the business processes and finally increase productivity for the software customer as well as end users.

1.3 Proposed Solution

After analyzing the investors requirements, as well as identifying end user needs through analysis of similar software, we chose a candidate solution, a website which addresses needs of the high end property owners as well as guests looking for luxury/high valued properties for rent. The website platform is capable of making profit for the investors through monetizing transactions between guests and hosts. Another proposed solution was development of a mobile application encapsulating the same functionalities as the website, however throughout interviews with investors we found that, they prefer development of a web application due to the existence of previously built assets and capabilities, however they are in favor of extending the platform to mobile devices in the future.

In a nutshell, our solution boils down to development of a fully functional responsive website [8], hosted by one of the well known cloud providers, which is capable of managing booking, reservation, handling transactions, making invoices and finally build reports for future analysis. The emphasis on "UX Design" in order to provide premium experience for the users is considered as well.

1.4 Thesis Structure

We already provided some context and back ground info about digital platforms in general and booking digital platform specifically throughout this chapter. In he second chapter we will focus on technologies used for the implementation of our digital platform as well as methodologies which were chosen by the software development team. In the third chapter we will go through software engineering blue print of the project and discuss the software architectural design thoroughly. Finally in the fourth chapter we will discuss the implementation experience and process of validation by the customer and end users. The conclusion and possibility for future improvement of the software is discussed in the last chapter.

[8]Responsive web design is an approach to web design that makes web pages render well on a variety of devices and window or screen sizes.

Chapter 2

Background

In Section 2.1 we will go through software development methodologies used to assess the requirements. The methodologies mentioned in 2.1 hep us create a framework upon which we build our solution.

2.1 Relevant Methodologies Used

2.1.1 Agile Software Development

Agile [1] software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development.

Agile by nature focuses on customer collaboration, interactions between individual members of a development team, and most importantly responding to unforeseen changes (that may arise during development cycle) in requirements. The goal of agile is to produce a working software rather than a comprehensive documentation.

Some of the more famous agile based frameworks are as follows:

- Scrum
- Rapid application development(RAD)

[1]https://www.agilealliance.org

- lean software development
- lean startup
- feature driven development
- extreme programming(XP)

We chose Scrum for the purpose of this project and throughout 2.1.2 we describe how it works.

The Agile software development life cycle is an iterative process. Each iteration delivers a piece of working software available for use by the customer until the final product is complete. The duration of each iteration is usually two to four weeks in length and by definition has a fixed completion time. As it is evident due to the iterative nature of Agile software development life cycle, multiple iterations take place during development, with each iteration completed the customers and business stakeholders provide additional feedback in order to ensure the features delivered meet their requirements.

Figure 2.1 illustrates a typical iteration process flow of agile methodology. Each iteration is composed of phases which are described briefly below:

1. Requirements: Define the requirements for the iteration based on the product backlog, sprint backlog, customer and stakeholder feedback.
2. Design & Development: Design and develop software based on defined requirements.
3. Testing: QA (Quality Assurance) testing, internal and external training, documentation development.
4. Deployment: Integrate and deliver the working iteration into production.
5. Review: Accept customer and stakeholder feedback and work it into the requirements of the next iteration.

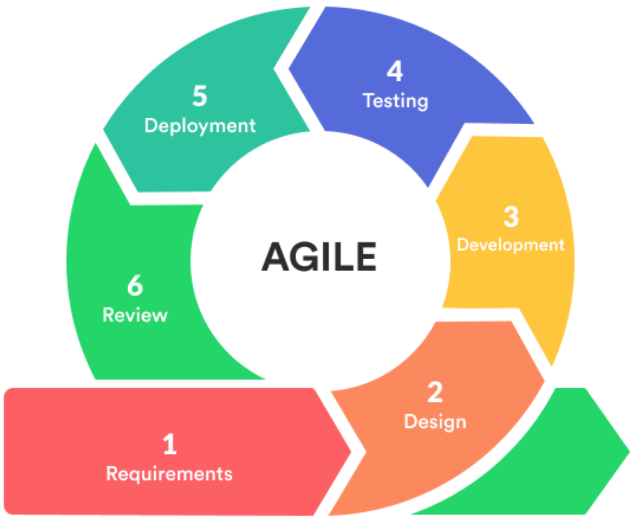


Figure 2.1: Agile Framework
Figure illustrating phases of an iteration in Agile methodology.

2.1.2 Scrum

Scrum [2] is an agile process framework for managing complex knowledge work, with an initial emphasis on software development. We choose Scrum because it is adaptive,it uses iterative cycles, and it is fast and flexible therefore this methodology allows us to deliver significant value to the customer early on. Scrum ensures transparency in communication and creates an environment of collective accountability and continuous progress.

We start by gathering information about the product and the requirements in form of User Stories. A user story is the smallest unit of work in an agile framework. It is an end goal, not a feature, expressed from the software users perspective. User stories are usually developed through discussions with stakeholders. A common template for a user story is as follows: As a<role>I can<capability>, so that<receive benefit>. User stories are grouped together in order to from Epics. Epics are a collection of user stories which are related to each other either implicitly or explicitly. The scrum

[2]https://www.scrum.org

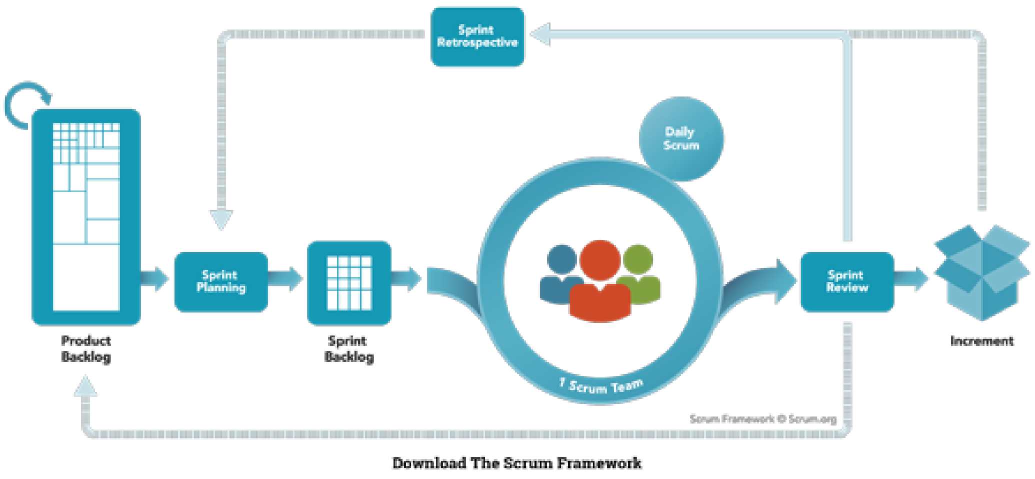


Figure 2.2: Scrum Methodology
Figure illustrating various phases of scrum methodology.

team usually choose epics based on a list of prioritized requirements defined by the customer. The chosen epics enter the product backlog[3]. In scrum, the Product Backlog is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The scrum team identifies a list of tasks from the product backlog, these tasks must be completed by the end of the Scrum Sprint. Sprint[4] is a time-box of one month or less during which a "Done", usable, and potentially releasable product Increment is created.

2.2 Relevant Technologies

In this section we introduce technologies used for the development of this project. Figure 2.3 shows the main elements of our web application. We will go through main elements of proposed architecture for our web application in section 2.2.1. Once we defined the architecture we plan to use, in section 2.2.2 we will discuss the technology used for the front-end and finally in 2.2.3 the back-end platform of choice is examined.

[3]https://www.scrum.org/resources/what-is-a-product-backlog
[4]https://www.scrum.org/resources/what-is-a-sprint-in-scrum

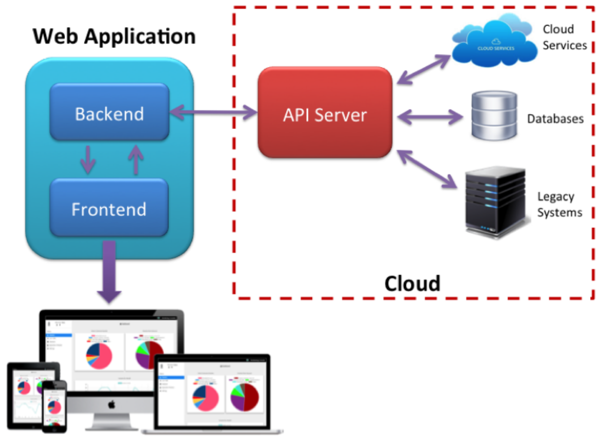


Figure 2.3: Model for Web application
Figure illustrating main elements of our web application and how they work together.

2.2.1 Three tier architecture

In this section we will examine our proposed architecture for the application, which is three tier architecture.

The three tier architecture is a client-server architecture, in which tier represents physical separation and layer represents logical separation. In this architecture each layer can potentially run on a different machine. In addition each tier is developed and maintained as independent modules, most often on separate platforms.

Some aspects concerning design of three tier architecture are as follows:

- Unconnected tiers should not communicate.
- Change in platform affects only the layer running on that particular platform.
- Data transfer between tiers is part of the architecture. Protocols involved may include one or more of SNMP, CORBA, Java RMI, .NET Remoting, Windows Communication Foundation, sockets, UDP, web services or other proprietary protocols.
- Three tier architecture follows component-oriented approach, generally

the architecture uses platform specific methods for communication instaed of a message based approach.

Now we briefly discuss role of each tier and how tiers work together. Figure 2.4 illustrates elements discussed below.

1. **Presentation Tier/Front-end:** This is the topmost level of the application. It provides user interface, handles the interaction with the user. Sometimes called the GUI or client view or front-end. It sends content to browsers in the form of HTML/JS/CSS. This tier may use frameworks such as React, Angular, Ember, Aurora, etc. It communicates with business logic tier in form of http request/response, which business logic tier can handle.

2. **Application Tier/Back-end (Business Logic or Middle Tier):** This tier contains set of rules for processing information(business logic) and is able to accommodate many users. This tier is sometimes also called as middle-ware. Middle-ware processes the inputs received from the clients and interacts with the database. The logic tier will have the JSP, Java Servlets, Ruby, PHP, C++, Python and other programs. The logic tier runs on a Web server.

3. **Data Tier:** A database, comprising both data sets and the database management system or RDBMS [5] software that manages and provides access to the data (back-end). It provides security, data integrity and support application. The data tier is usually a kind of database, such as a MySQL, SQLite or PostgreSQL database running on a server.

Now that we have defined the three tier architecture and how it works, we discuss the main advantages as well as disadvantages of using this architecture.

Advantages:

- **Maintainability:** Because each tier is independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.

[5]Relational Database Management System

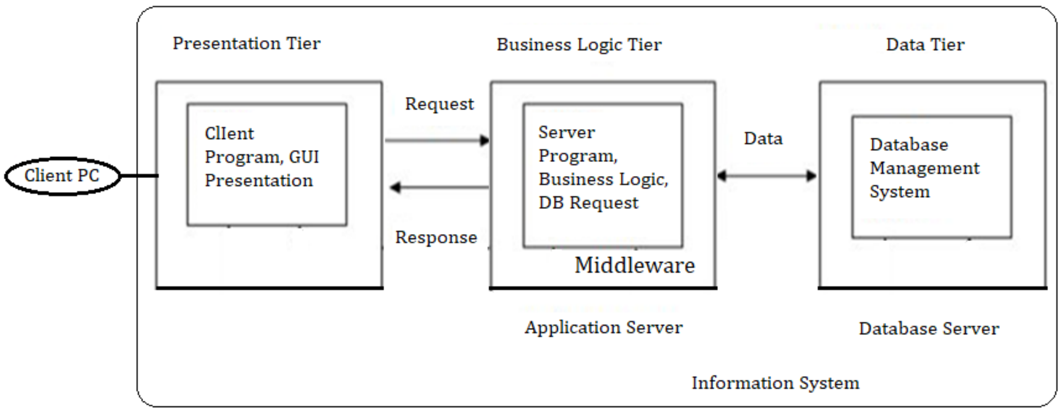


Figure 2.4: Taxonomy of three tier architecture
Figure illustrating the three tier architecture.

- **Scalability:** Because tiers are based on the deployment of layers, scaling out an application is reasonably straightforward.
- **Flexibility:** Because each tier can be managed or scaled independently, flexibility is increased.
- **Availability:** Applications can exploit the modular architecture of enabling systems using easily scalable components, which increases availability.
- **Re-usability:** Components are reusable
- **Faster development:** Because of division of work web designer does presentation, software engineer does logic, DB admin does data model.

Disadvantages:

- **Cost:** High installation cost.
- **Complexity:** Structure is more complex as compare to 1 & 2 tier architectures.

The usage of Cloud environments as Database solution provider is optimal for our architecture. The dynamic distributed database over Cloud

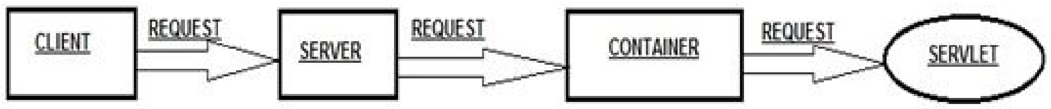


Figure 2.5: Request from Client to Server
Figure illustrating a request made by a client toward the web application server.

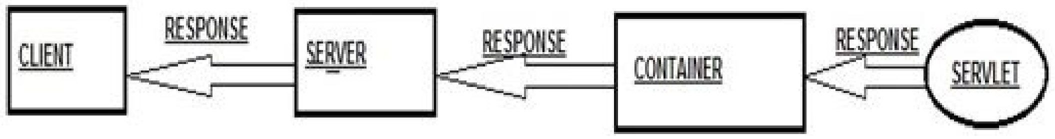


Figure 2.6: Response from Server to Client
Figure illustrating server response to the client request.

Environment improves accessibility and response time for our clients as well as providing scalability in case of future expansions of our digital platform. Figure 2.3 illustrates this concept.

2.2.2 Spring for Web applications

The Spring Framework is an application framework and inversion of control container for the Java platform.

There are a few key concepts which need declaration. Firstly we explain the concept of "Web Container" briefly. A Web Container is a java application that controls servlet. Servlets do not have a main() method, therefore they require a container to load them. In essence A Web Container is a place where servlets are deployed. Figure 2.5 shows a request made by a client to the application server, when a client sends a request to the web server which contains a servlet, the web server redirects that request to the container rather than to the servlet directly. Figure 2.6 shows the response received by the client. In this case the Web Container finds out the servlet which requested a response and passes the Http Request as well as the response to the servlet and loads the servlet methods i.e. doGet() or do Post().

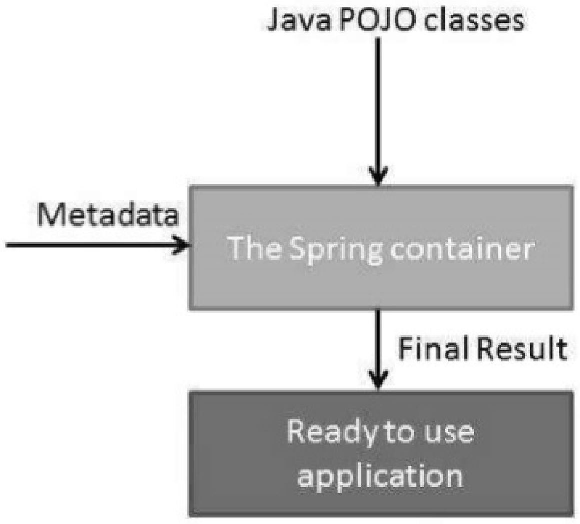


Figure 2.7: Spring container dependency injection
Figure illustrating usage of user supplied Metadata and JAVA POJO classes for the purpose of dependency Injection into the application.

Secondly we introduce the concept of "Inversion of Control". IOC [6] is a process whereby objects define their dependencies, that is, the other objects they work with,only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.The container then injects those dependencies when it creates the bean. In other words Inversion of control indicates the control flow of the program is inverted meaning that the control flow of the program is delegated to an external source, a.k.a. the container.

In addition the Spring IoC container consumes a form of configuration metadata which can be seen in figure 2.7; this configuration metadata represents how an application developer instructs the Spring container to instantiate, configure, and assemble the objects in an application. Configuration metadata is traditionally supplied in a simple and intuitive XML format provided by the programmer.

[6]https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-introduction

Some essential tasks which the web container is responsible for are listed below.

- Managing objects/Dependency Injection
- Managing the servlets life cycle
- Mapping URLs to a particular servlet
- Directing/redirecting requests
- Verifying the URL requester has correct access rights

The Spring[7] Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications. Spring web framework core technologies are: dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP. We use JDBC for our data access,Java Database Connectivity(JDBC) is an application programming interface(API) for the programming language Java, which defines how a client may access a database. JDBC is a Java-based data access technology used for Java database connectivity which is part of the Java Standard Edition platform, from Oracle Corporation.

2.2.3 Sencha Ext-JS

Sencha Ext-JS is a JavaScript framework for building cross-platform HTML5-based web applications. Ext-JS includes pre-integrated and tested UI components which make it a suitable framework for data visualization across a wide array of browsers.

Ext-JS allows us to build a single page web application. A single-page application is a web application or website that interacts with the web browser by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages.

[7]https://spring.io/projects/spring-framework

Sencha Ext-Js libraries support a variety of data visualization elements including charts, grids, trees, as well as other common user interface components such as buttons, tabs, tool tips, gauges which can be customized for various purposes according to the programmers needs. Some features of Ext-Js which make it a viable choice for a client side web application are as follows:

- Browser Compatibility
- Support of applications across multiple devices
- Increased developer productivity
- Rapid Prototyping
- Multi Platform
- No extra Plugin installation on the browser
- Sencha Support

In this project we use "Sencha Cmd" command line tools suit which builds the front end web application for us.

Chapter 3

Related work

There are a number of other similar digital platforms on the internet which provide booking services online. Among them Booking[1] and AirBnB are the most well known. Observation of similar platforms help us in understanding the shortcomings and lacking features/functionalities which can potentially differentiate our platform from the existing platforms. In addition we can identify the most essential functionalities that our platform shall provide for the end users (a.k.a. "Must-have features").

Since development team did not have access to Software code and assets of the competitor platforms, the team analyzed their software from User Experience(UX) perspective. The team found out, practices used by the competitor platforms toward their customers are not always aligned with end users expectations from these platforms, this concept is defined as *Dark patterns* among UX design researchers community.

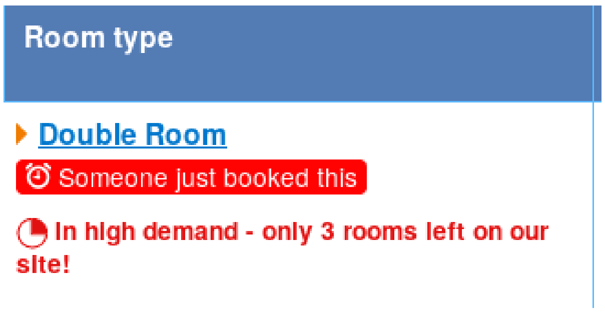
According to Gray et al. the term *Dark patterns* refers to instances where designers use their knowledge of human behavior (e.g., psychology)and the desires of end users to implement deceptive functionality that is not in the user's best interest[6]. In addition Gray et al. classify *Dark patterns* as follows:

- Bait and Switch disguised Ad: User sets out to do one thing, but a different, undesirable thing happens instead. Adverts that are disguised

[1]www.Booking.com is a travel meta search engine for lodging reservations.

as other kinds of content or navigation, in order to get User to click on them.

- Forced Continuity: When Users free trial with a service comes to an end and Users credit card silently starts getting charged without any warning. In some cases this is made even worse by making it difficult to cancel the membership.
- Friend Spam: The product asks for Users email or social media permissions under the pretence it will be used for a desirable outcome (e.g. finding friends), but then spams all Users contacts in a message that claims to be from the user.
- Hidden Costs: User get to the last step of the checkout process, only to discover some unexpected charges have appeared, e.g. delivery charges, tax, etc.
- misdirection: The design purposefully focuses users attention on one thing in order to distract him/her attention from another.
- Price comparison prevention: The retailer makes it hard for user to compare the price of an item with another item, so user cannot make an informed decision.
- privacy zuckering: Users are tricked into publicly sharing more information about themselves than they really intended to. Named after Facebook CEO Mark Zuckerberg.
- roach model: The design makes it very easy for user to get into a certain situation, but then makes it hard for user to get out of it (e.g. a subscription).
- Sneak into basket: User attempts to purchase something, but somewhere in the purchasing journey the site sneaks an additional item into the users basket, often through the use of an opt-out radio button or checkbox on a prior page


Room type

Double Room

Someone just booked this

In high demand - only 3 rooms left on our site!

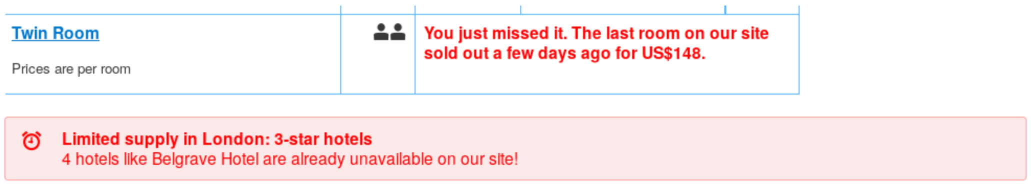
Figure 3.1: Example of a Dark pattern in UX design
Figure illustrating one type of Dark Patterns used by a booking platform

- Trick questions: User responds to a question, which, when glanced upon quickly appears to ask one thing, but if read carefully, asks another thing entirely.

The development team analyzed competitor platforms, by conducting an online survey from a community of one hundred digital platform users among company employees. The result showed 85% already experienced either one or multiple forms of *Dark patterns* practices and they were dissatisfied with their experience. The dominant categories of malpractices from survey are as follows:

- Misleading price sorting
- Incoherent rating system, some platforms categorize multiple aspects of hospitality experience, however they only show the best average rating
- Misleading reviews on front/first page, some platforms tend to sort reviews in a way which prioritize positive reviews on the first page

Here there are some examples of usage of *Dark patterns*. Figure 3.1 and 3.2 illustrate a case of *Misdirection* by a digital platform, in this case viewers are mislead into believing that the offer the platform purposes is time limited (sense of emergency) and the viewer may lose the chance in case of hesitation, which is not the case.


Twin Room

Prices are per room

You just missed it. The last room on our site sold out a few days ago for US$148.

Limited supply in London: 3-star hotels
4 hotels like Belgrave Hotel are already unavailable on our site!

Figure 3.2: Example of a Dark pattern in UX design
Figure illustrating one type of Dark Patterns used by a booking platform

Final conclusion drafted from the survey indicates, while in the short run usage of *Dark patterns* practices might improve short term profitability of the platform through increased number of bookings, in the long run it will damage reputation of company as well as customer loyalty to the platform, which lead customers into migrating to another platform. The development team concluded that the differentiating factor for our platform is "customer first" approach. By adopting "customer first" approach the *Dark patterns* practices are avoided altogether, while the emphasis is to build a trust and log lasting relationship between platform end users and the platform owners.

Huang and Lee study [7] examines various modes of attacks on websites including cross site scripting(XSS), Sql Injection as well as mitigation techniques against such attacks which author classifies as extensive testing of security and software verification techniques. The important viewpoint which needs to be addressed is the *insecure information flow* of web application which can introduce vulnerabilities. For this purpose we used mitigation techniques against SQL injection attacks by using *Spring* framework's prepared statement. Furthermore the design of the back-end system follows guidelines which incorporate cleansed(validated) data which is only accessible through tested APIs which are controlled by *spring* authentication mechanisms in order to deny access to unauthorized usage of back-end data. Figure 3.3 illustrates one of the mitigation techniques against Unauthorized requests. The steps which are implemented by Spring Security is as follows:

1. the FilterSecurityInterceptor obtains an Authentication from the SecurityContextHolder.

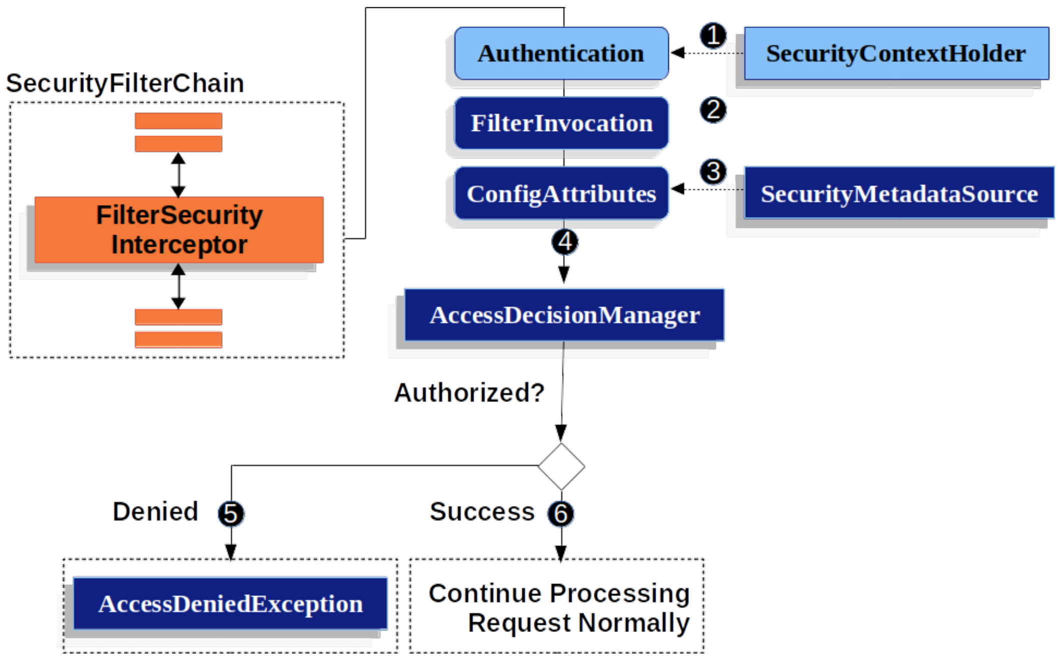


Figure 3.3: Spring Authorization of HttpServletRequest
Figure handling of Unauthorized HttpServletRequest.

2. FilterSecurityInterceptor creates a FilterInvocation from the HttpServletRequest, HttpServletResponse, and FilterChain that are passed into the FilterSecurityInterceptor.

3. lNext, it passes the FilterInvocation to SecurityMetadataSource to get the ConfigAttributes.

4. Finally, it passes the Authentication, FilterInvocation, and ConfigAttributes to the AccessDecisionManager.

5. If authorization is denied, an AccessDeniedException is thrown. In this case the ExceptionTranslationFilter handles the AccessDeniedException.

6. If access is granted, FilterSecurityInterceptor continues with the FilterChain which allows the application to process normally.

Chapter 4

Designing Web Application Platform

In this chapter we will go through necessary phases required to develop a software solution. This chapter is entirely focused on the design and modelling of the software solution, regardless of the underlying programming languages or development platform of choice. We use UML [1] in order to make a model of our proposed solution. UML is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. In addition we need to make model of business processes and main activities, we will see BPMN [2] diagrams throughout this chapter.

4.1 Software Modelling

4.1.1 Use Case Diagram

UML use case diagrams describe relevant functionalities of the business process, the users/actors involved in execution of the business process, and the assignment of functionalities to users/actors.

[1]The Unified Modeling Language(UML)

[2]Business Process Model and Notation(BPMN)

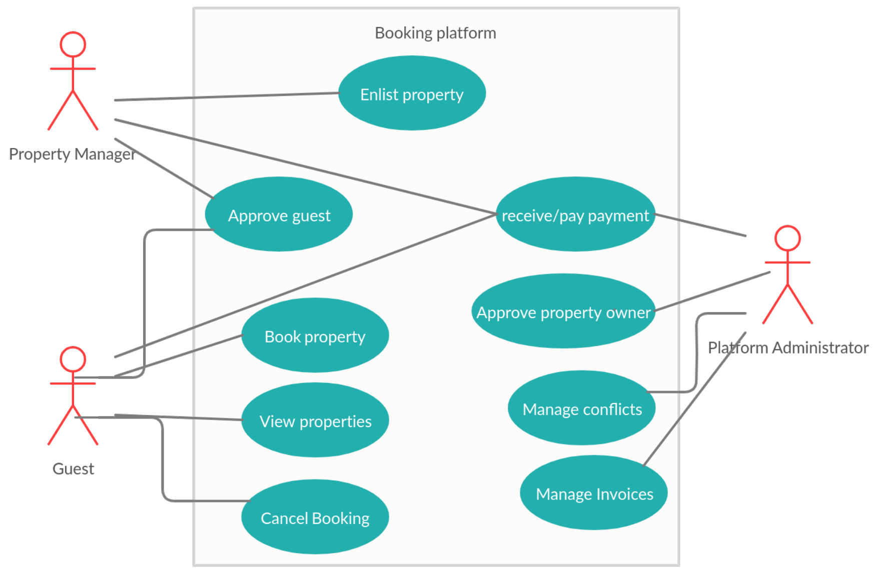


Figure 4.1: Use case Diagram
Figure illustrating Use case diagram, part of UML modelling diagrams.

Use case diagram visualizes functional requirements of the system which were previously defined in requirement analysis section of chapter one. As it is evident in figure 4.1, some use cases are linked only to one actor, while others involve two or more actors based on nature of the task.

Figure 4.1 illustrates use case diagram for our digital booking platform. Main actors of the system are "Platform Manager", "Property Owner" and "Guest". These actors are connected to their respective functionalities which are indicated as oval shaped figures in the use case diagram.

4.1.2 Class Diagram

UML class diagrams report the schema of the underlying information model. We need class diagram in order to describe the structure of our system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. Figure 4.2 illustrates our digital booking platform classes and the relationship among objects. As the figure 4.2 illustrates, there are four essential classes identified for our system namely

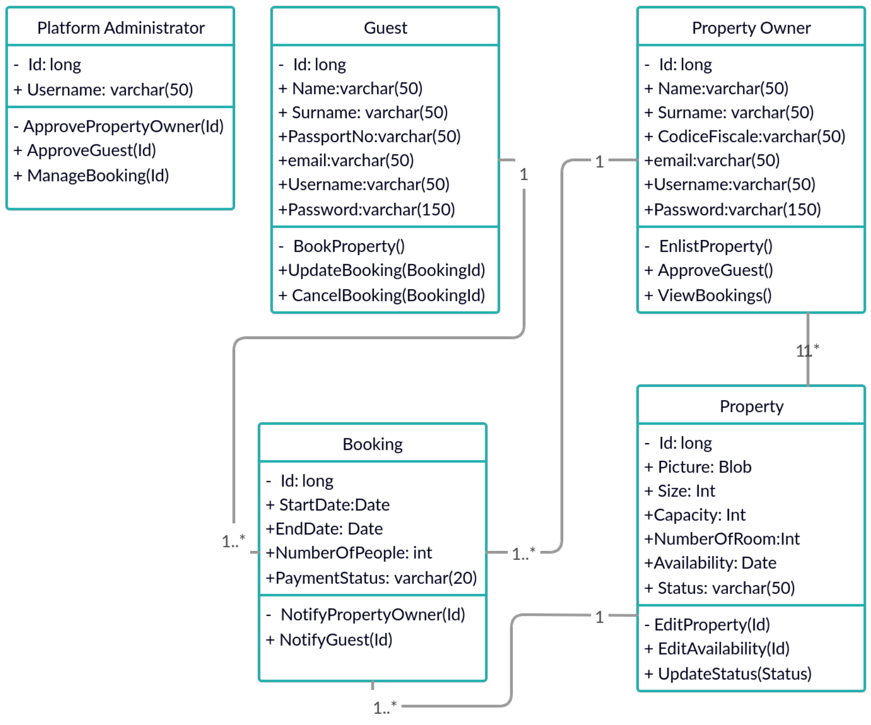


Figure 4.2: UML Class Diagram
Figure illustrating UML class diagram, part of UML modelling diagrams.

"Guest", "Platform Administrator", "Property Owner" and "Booking". In addition figure 4.2 also shows association relationships among identified classes which are shown by the numbers on both ends of the association relationship. These associations are essential for designing database tables and placement of foreign keys later on. One of the important aspects of modelling UML class diagram is the fact that, class diagram provides a solid foundation about the structure of classes which later on we will use when in the implement ion of our system.

4.1.3 Business Process Modeling Diagram

After identifying the main actors and functionalities of the system, we need to further examine the flow of the business processes which occur inside the system. For this purpose we use Business Process Modelling in particular

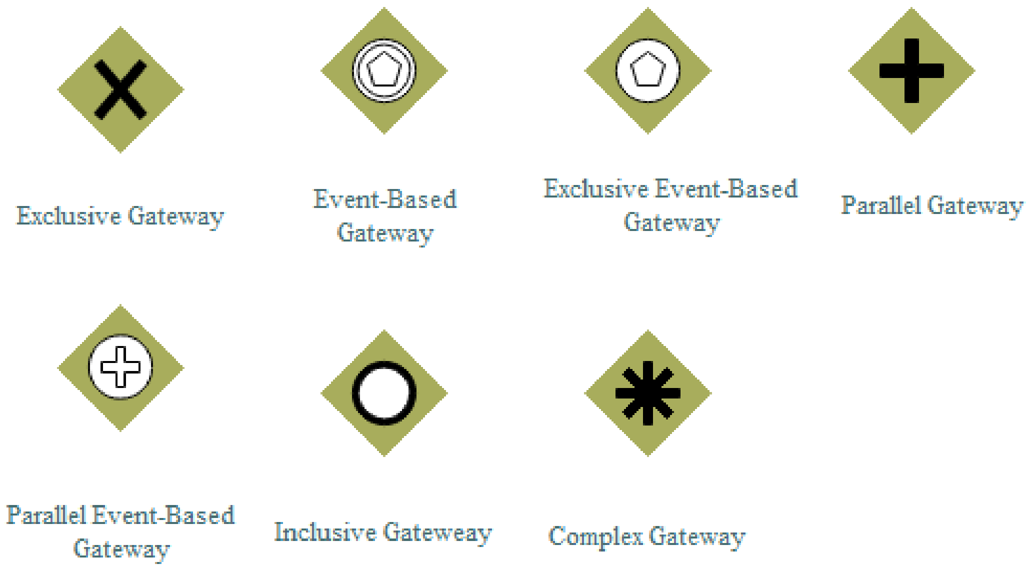


Figure 4.3: BPMN gateways
Figure illustrating BPMN gateways.

BPMN [3]. BPMN is a widely used standard for process modeling. In BPMN, activities are represented as round rectangles, Control nodes (called gateways) are represented using diamond shapes finally Activities and control nodes are connected by means of arcs (called flows) that determine the order in which the process is executed. Figure 4.3 shows various types of gateways used in our business process diagrams.

Figure 4.4 illustrates business process of booking by Guest. In this process we use *exclusive gateway* which evaluates the state of the business process and, based on the condition, breaks the flow into one of the two or more mutually exclusive paths. In addition we can see parallel tasks which execute concurrently, as shown in figure 4.4 "Check Availability" and "Check approval by Owner" tasks are executed concurrently in the booking process. Final point which we need to address is usage of time based events. A clock icon represents the timer event, we used this technique in order to indicate the payment is bound to a time limit in this case 60 minutes, in this way guest must finalize the booking by doing the "payment transaction" task otherwise

[3]business Process Model and Notation(BPMN)

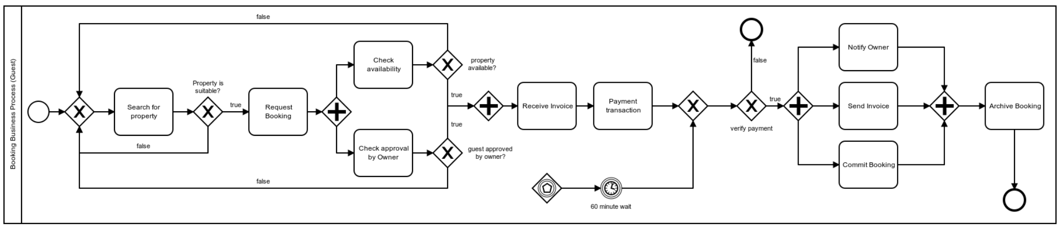


Figure 4.4: BPMN guest Diagram
Figure illustrating BPMN guest Diagram.

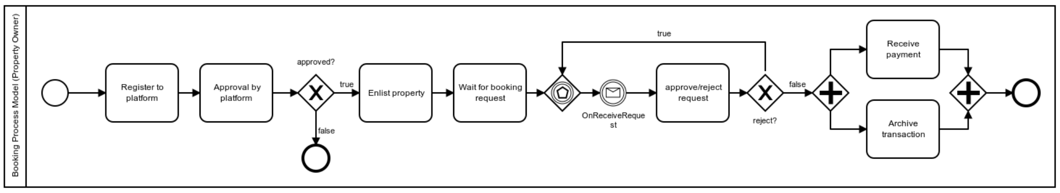


Figure 4.5: BPMN property owner Diagram
Figure illustrating BPMN property owner Diagram.

the booking process ends.

Figure 4.5 illustrates business process of booking from perspective of property owner. Property owner firstly registers to the platform, the platform then validates property owner and property owner is given permission to enlist his/her property for booking. After enlisting the property the business process stops until there is a *signal* indicating a request for booking, the property owner has a choice of approving or rejecting the request. Upon accepting the request "Receive payment" and "Archive transaction" processes will execute simultaneously and finally the business process ends.

Figure 4.6 illustrates the process of "Managing dispute" between "Property owner" and "Guest" from perspective of platform administrator. The process begins by simultaneous validation of Property Owner's clam and Guest's claim. Once this is done, one of the claim's is chosen as a "valid" claim. If we consider Owner claim as the valid claim, the status of payment by the "Guest" is evaluated, and the security deposit of the Guest is collected in case the Guest has already paid the security deposit. Then the guest is notified and the process ends. On the other hand if the guest's claim turns

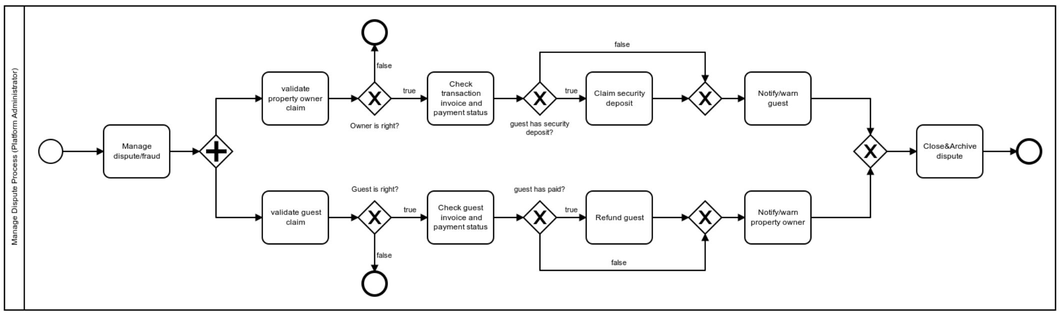

Manage Dispute Process (Platform Administrator)
Manage dispute/fraud
validate property owner claim
Owner is right?
false
true
Check transaction invoice and payment status
guest has security deposit?
true
Claim security deposit
false
Notify/warn guest
Close&Archive dispute
validate guest claim
Guest is right?
true
false
Check guest invoice and payment status
guest has paid?
true
Refund guest
false
Notify/warn property owner

Figure 4.6: BPMN platform owner Diagram
Figure illustrating BPMN platform owner Diagram.

out to be valid, guest will receive a refund from the platform and property owner will receive a warning.

Figure 4.7 illustrate guest update/cancel booking business process. As it is evident from the diagram, firstly we must check for the status of booking in case the booking is not locked the process continues and the guest is given options to either change date/duration of stay, edit number of guests or simply cancel the booking altogether. Upon choice of edit by guest, the platform checks for availability of the property in the specified period, in case property is available the change made by Guest is confirmed and the process continues by sending invoice and wait for payment. In case the Guest decides not to commit payment, the timer will continue the process and as a result verify payment check returns false and consequently the process ends.

Now that we have seen the main structure of our system throughout UML use case diagram, UML class diagram and BPMN business process diagrams, we arrive at design of database structure for our digital booking platform project. We complete this chapter by representing our database structure represented in Figure 4.8. As evident from the picture, the tables are similar in structure to the class diagram represented in figure 4.2. In addition the choice of primary and foreign keys is a direct result of the UML class diagram relationships.

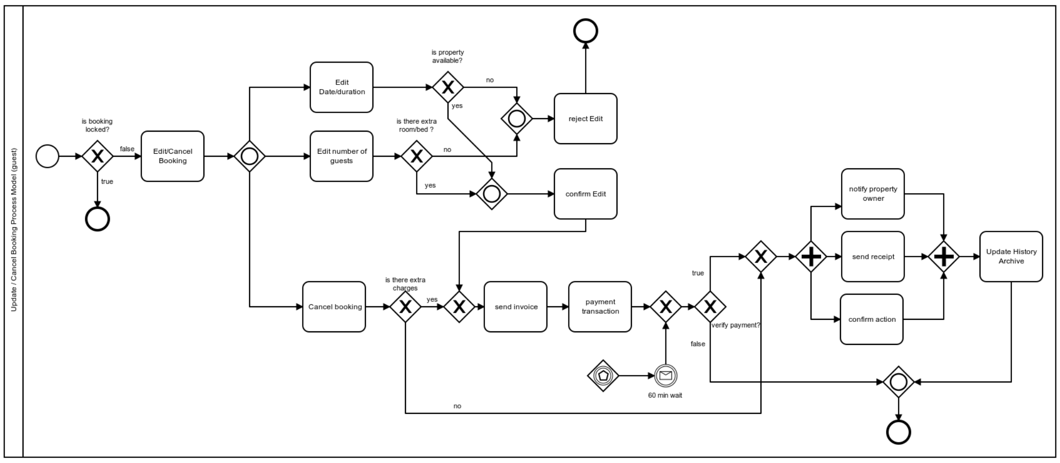


Figure 4.7: BPMN guest update booking Diagram
Figure illustrating BPMN guest update booking Diagram.

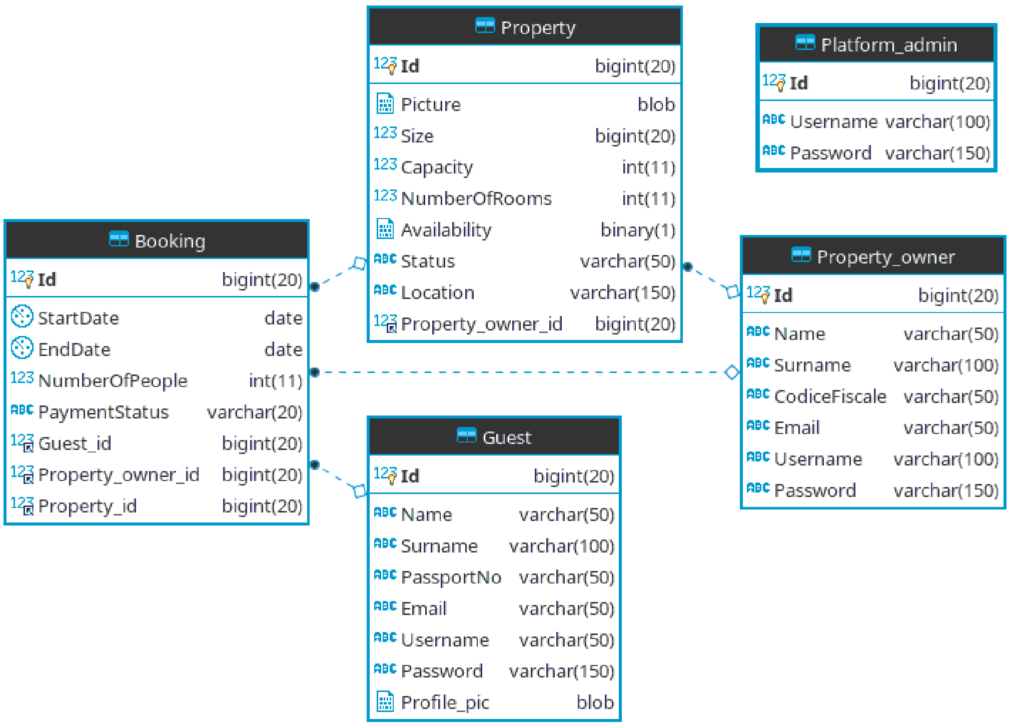


Figure 4.8: Database structure of digital booking platform
Figure illustrating main tables as well as relationship among tables through usage of foreign keys.

Chapter 5

Implementation

Throughout this chapter we will go through the development phase of our proposed solution. In section 5.1 we will see design patterns commonly used for development of web applications, as well as some examples from our application code in order to clarify the concepts. In section 5.4 we will discuss methods that we use for testing and debugging our application.

5.1 Design patterns

A design pattern is a general solution that addresses common software-design challenges. While not a finished design, you may think of a design pattern as a template or set of best practices. [8]

5.1.1 MVC

The model-view-controller (MVC) pattern is a software-design pattern used for creating data-driven web applications. In the design pattern of Model-View-Controller (MVC) the presentation of information (View) is separated from the information itself (Model) and the control or manipulation of the information (Controller).[9]

In an MVC architecture, most classes are either Models, Views or Controllers. The user interacts with Views, which display data held in Models. Those interactions are monitored by a Controller, which then responds to the interactions by updating the View and Model, as necessary. The View and the Model are generally unaware of each other because the Controller has the sole responsibility of directing updates. Generally speaking, Controllers will contain most of the application logic within an MVC application. Views ideally have little (if any) business logic. Models are primarily an interface to data and contain business logic to manage changes to said data.

The goal of MVC is to clearly define the responsibilities for each class in the application. Because every class has clearly defined responsibilities, they implicitly become decoupled from the larger environment. This makes the app easier to test and maintain, and its code more reusable, since it is not integrated with a specific presentation format. We choose MVC as design pattern of choice for our back-end application. Figure 5.1 shows the adaptation of MVC architecture to the web application, using JAVA as a reference platform.

In order to use MVC pattern in our project we need to take a look at class diagram 4.2. As it is evident from diagram 4.2, we have five classes which we consider as our model objects. A model is a simple POJO (Plain Old Java Object) class, as an example we model Guest class as:

```
->Guest.java
public class Guest {
  private int id;
  private String name;
  private String surName;
  private String passportNo;
  private String email;
  private String username;
  private String password;

  public String getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  //...
  // Above code is repeated for other fields for generating
  // getter and setter methods.
}
```

Now that we have seen how a model class is created, we will see a Controller example. Example shown here is called *RESTFUL* controller which we will see in section 5.2. The controller receives HTTP requests and responds to them, according to the defined path. This controller is in charge of API requests made by the front-end application.

```
->Controller.java
@RequestMapping("/api")
@RestController
class ApiController {
  //... A repository containing the fetched data from database
  @Autowired
  private final GuestRepository repository;
  //... Mapping which connects a path to a method defined in
  //... controller.
  @GetMapping("/guests")
  List<Guest> all() {
    return repository.findAll();
  }
}
```

The final piece of our MVC example is "View". We used Sencha ExtJS framework for our GUI application, however any plain HTML or JSP file is suitable for representing of a "View" to the client as long as an appropriate path and method for retrieving that view is defined in a Controller.

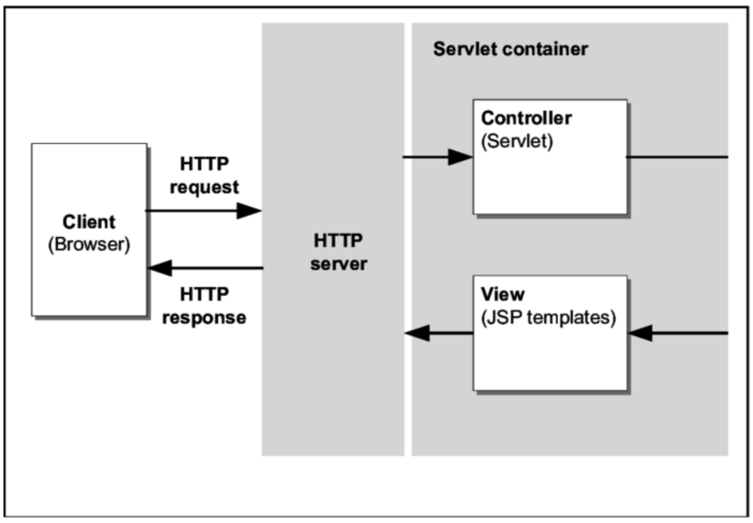


Figure 5.1: MVC Architectural design pattern
Figure illustrating the MVC architecture applied to Web application. In this section we will see two design patterns for our back end and front end applications.

5.1.2 MVVM

Another software-design pattern used in the development of our front-end application is Model View ViewModel (MVVM) design pattern. In MVVM, the View layer is concerned only about the graphical user interface, while the Model layer only about the business logic. All communication between them is realized by the ViewModel layer. [10]

The key difference between MVC and MVVM is that MVVM features an abstraction of a View called the ViewModel. The ViewModel coordinates the changes between a Model's data and the View's presentation of that data using a technique called "data binding". The goal of MVVM is that the Model and framework perform as much work as possible, minimizing or eliminating application logic that directly manipulates the View.

In this section we go through each part of MVVM architectural pattern and see some examples:

- **Model**: This is the data for our application. A set of classes (called "Models") defines the fields for their data (e.g. a Guest model with username and password fields). Models know how to persist themselves through the data package and can be linked to other models via associations. **Store**: Models are normally used in conjunction with Stores to provide data for grids and other components. Models are also an ideal location for any data logic that you may need, such as validation, conversion, etc.

- **View**: A View is any type of component that is visually represented. For instance, grids, trees and panels are all considered Views.
- **Controller**: Controllers are used as a place to maintain the view's logic that makes the application work. This could entail rendering views, routing, instantiating Models, and any other sort of app logic.
- **ViewModel**: The ViewModel is a class that manages data specific to the View. It allows interested components to bind to it and be updated whenever this data changes.

Figure 5.2 shows the adaptation of MVVM architecture for the Graphical User Interface (GUI) of our web application which lives inside the end user's browser.

Example code shown below represents a MVVM Model for our Booking class, defined in our front-end application.

```
-> Booking.js
// Definition of a Model class according to ExtJS framework
Ext.define('BookingApplication.model.Booking', {
//We need to extend the Model super class defined by ExtJS
extend: 'Ext.data.Model',
requires: ['Ext.data.proxy.JsonP'],
//We define model fields here
fields: [
{
    name: 'id',
    mapping: 'id' //-> Mapping our model fields to the Object
// fields, coming from JSON response of back-end
},
{
    name: 'startDate',
    mapping: 'startDate'
},
{
    name: 'endDate',
    mapping: 'endDate'
},
{
    name: 'numberOfPeople',
    mapping: 'numberOfPeople'
},
{
    name: 'paymentStatus',
    mapping: 'paymentStatus'
}],
proxy:
{
    type: 'ajax',
    url: '/api/bookings',
    reader: {
        type: 'json',
    }
}});
```

As we can see in above code, "proxy" defines the type of call which is required to fill the model fields as well as the end point url on the back-end which responds to the call made by the object. The response which "reader" for this model expects is in from of a JSON message which we will discuss in section 5.3.

Now that we have seen how a model is defined in Sencha ExtJS framework, the next step is to define a ViewModel for our model. In the Below code section we can see that the Model for which we are defining a View Model must be referenced, there is also a *Store* type of object which holds a collection of items of the type specified type. This store object is used inside the View to show the collection of objects to the user.

```
Ext.define('BookingApplication.view.main.BookingViewModel', {
  extend: 'Ext.app.ViewModel',
  alias: 'viewmodel.bookingviewmodel',
  requires: [
    'BookingApplication.model.Booking'
    ],
  stores: {
    bookings: {
      model:'BookingApplication.model.Booking',
      autoLoad: true
    }
  }
});
```

We need a Grid object from the ExtJS framework libraries in order to show the collection of objects in the "Store" we have created. The Grid example shown below is later used inside our View.

```
Ext.define('BookingApplication.view.BookingGrid',
{
    extend: 'Ext.grid.Grid',
    xtype: 'bookinggrid',
    cls: 'booking-grid',
    requires: [
        'Ext.grid.column.Column',
        'Ext.grid.cell.*'
    ],
    defaults: {
        height: 54
    },
    columns: [{
    text: 'id',
    dataIndex: 'id',
    flex: 1
    }, {
    text: 'startDate',
    dataIndex: 'startDate',
    flex: 1
    }, {
    text: 'endDate',
    dataIndex: 'endDate',
    flex: .5
    }, {
        text: 'numberOfPeople',
        dataIndex: 'numberOfPeople',
        flex: .5
    },{
        text: 'paymentStatus',
        dataIndex: 'paymentStatus',
        flex: .5
    }
]});
```

Finally we proceed to insert the Grid we have created above, in our View. As we can see in the code section below, we include a reference to our custom grid inside our View, we also have to reference the store object that we have created so that ExtJS framework loads the custom Grid with the store content.

```
Ext.define('BookingApplication.view.main.MainView', {
  extend: 'Ext.tab.Panel',
  xtype: 'mainview',
```

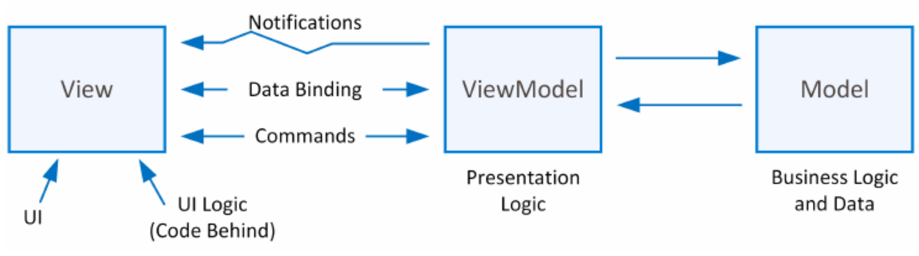


Figure 5.2: MVVM Architectural design pattern
Figure illustrating the MVVM architecture which is applied to the front-end application.

```
  requires: [
    'ModernTunes.view.main.BookingViewController',
    'ModernTunes.view.main.BookingViewModel',
  ],
  controller: 'bookingviewcontroller',

  viewModel: {
    type: "bookingviewmodel"
  },
  items: [
    {
        title: "Grid",
        xtype: 'tunesgrid',
        bind: {
          store: '{bookings}'
        }
    }
  ]
});
```

Now we have created our ViewModel and View. As the above code indicates, we also added BookingViewController. This controller handles interactions between the user and GUI application for this specific View.

5.2 RESTFUL Web Services

REpresentational State Transfer (REST) was originally introduced as an architectural style for building large-scale distributed hypermedia systems. REST leverages existing well known W3C/IETF standards (HTTP,XML,URI,MIME).[11] The REST architectural style is based on four principles: [11]

- **Resource identification through URI**: A RESTFUL Web service exposes a set of resources which identify the targets of interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.
- **Uniform interface**: Resources are manipulated using fixed set of four create, read, update, delete operations: PUT,GET,POST and DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.
- **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats (e.g., HTML, XML, plain text, PDF, JPEG, etc.). Meta data about the resource is available and used for example control caching, detect transmission errors, negotiate appropriate representation format, and perform authentication or access control.
- **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless, i.e., request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, e.g. URI rewriting, cookies, and hidden form fields. State can be embedded in response message to point to valid future states of the interactions.

We use RESTFUL web services to create back-end APIs which respond to HTTP requests created by the front-end GUI application. These APIs allow us to create communications between the front-end client application and the back-end server application. An example request is provided below:

```
->An example request
GET localhost:8080/api/guests
```

The server responds to this request by sending HTTP 200 (OK) status code which indicates that the request has been processed successfully on the server. In addition the response contains A JSON message corresponding to the requested source. We will discuss JSON in the next section.

5.3 JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999.[1]

JSON is used primarily to transmit data between our web application server and our front-end GUI application. Following on the example we made through this chapter, the JSON respond to the sample request made in section 5.2 is shown below.

```
->JSON response
 {
 "id":"1", "name":"Max",
 "surName":"Wayne","passportNo":"T51617181",
 "email":"max.p@server.com","username":"maxwayne",
 "password":"0d0a96fa021ccd3fac05df1a584e3185"
 }
```

5.4 Testing & Validation:

In this section we will go through tools and methods used for testing our application. There are various testing methodologies which cover different aspects of either functional or non functional requirements.We will focus

[1]https://www.json.org/json-en.html

mainly on "Unit testing" and "System testing" due to the nature of the project.

The goal of utilizing testing methodologies in development process is to make sure the software can successfully operate. Testing methodologies can typically be broken down between functional and non-functional testing. Functional testing involves testing the application against the business requirements. It incorporates all test types designed to guarantee each part of a piece of software behaves as expected by using uses cases provided by the design team or business analyst.

5.4.1 Unit Testing with JUnit

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process. [2]

JUnit is a unit testing framework for the Java programming language. We can write unit tests manually or use already existing framework tools to generate automated tests. The simple test shown below checks whether our "ApiController" is not *null*.

```
package com.example.testingweb;
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
```

[2]https://smartbear.com/learn/automated-testing/software-testing-methodologies/

```
@SpringBootTest
public class ControllerTest {
@Autowired
private ApiController controller;

@Test
public void contexLoads() throws Exception {
assertThat(controller).isNotNull();
}
}
```

As mentioned above we can also write custom unit tests. Example shown below is a custom Unit test which checks whether "Guest" user has "PassportNo" field or not.

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
class GuestRegisterTest {

  @Autowired
  private GuestService guestService;

  @Test
  void savedGuestHasPassportNumber() {
    Guest guest = new Guest("Max","Payne", "zaphod@mail.com");
    Guest savedGuest = guestService.registerGuest(guest);
    assertThat(savedGuest.getPassportNo()).isNotNull();
  }
}
```

5.4.2 System Test with LOG table

In order to monitor the performance and availability of our application we need to implement a logging mechanism. We use the logging mechanism for troubleshooting issues and to make decisions about maintenance tasks.

Some of the required functionalities of a logging system are listed below:

- Distinction between front-end and back-end application for logging errors/exceptions.
- Each application writes its own log using internal API, making sure that user requests are not blocked while logs are being written.
- The logging API collects log information produced by the application and sends it to the log database table.

We need to take into consideration that our application is using multi tier architecture, hence logs are separated by type of application. The decision of persisting logs into separate database tables or on a local log file located on the web application server depends on the level of autonomy and access given to developers by system administrator.

Another factor to consider is the number of concurrent requests to a certain database for accessing data (READ, WRITE operations) and the impact of log writing operations in case the log table is located on the same database server. We have decided on a log table located on the same server as our main database server, since the number of requests is not considerably large, later on this can be changed by adding a separate log server.

LOG4J is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License.

We use LOG4J to identify and collect exceptions occurring during an API call to our "ApiController". LOG4J allows modification of layout of the log message, persistence in database and/or local file. In addition we can set *level* of log, allowing for easy identification and categorization of events. A log request of level p in a logger with level q is enabled if $p >= q$. For the standard levels, we have $ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF$. In order to log information into our database we need to create a LOG table. The SQL code below indicates the structure our LOG table.

```
CREATE TABLE LOGS
   (USER_ID VARCHAR(20)    NOT NULL,
    DATED   DATE           NOT NULL,
    LOGGER  VARCHAR(50)    NOT NULL,
    LEVEL   VARCHAR(10)    NOT NULL,
    MESSAGE VARCHAR(1000)  NOT NULL
   );
```

We also need to modify the LOG4J.properties file. This file contains settings which LOG4J uses to persist log data into the LOG table.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>
<appender name="DB" class="org.apache.log4j.jdbc.JDBCAppender">
   <param name="url" value="jdbc:mysql://localhost/BOOKINGDB"/>
   <param name="driver" value="com.mysql.jdbc.Driver"/>
   <param name="user" value="root"/>
   <param name="password" value="****"/>
   <param name="sql" value="INSERT INTO LOGS
   VALUES('%x','%d','%C','%p','%m')"/>
   <layout class="org.apache.log4j.PatternLayout">
   </layout>
</appender>
<logger name="log4j.rootLogger" additivity="false">
   <level value="DEBUG"/>
   <appender-ref ref="DB"/>
</logger>
</log4j:configuration>
```

Now LOG4J is ready to persist log information. We can modify our "ApiController" by adding a logger which detects SQL Exception and persists it into our LOG table.

```
import org.apache.log4j.Logger;
import java.sql.*;
import java.io.*;
import java.util.*;
@RequestMapping("/api")
@RestController
public class ApiController{
   @Autowired
   private final GuestRepository;

   /* Get actual class name to be printed on */
   static Logger log = Logger.getLogger(ApiController.class.getName());

   @GetMapping("/guests")
   List<Guest> all() throws IOException,SQLException{
   try {
         return repository.findAll();
    } catch (SQLException e){
        log.debug("ERROR");
   }
}
```

The same principles shown above is applied to the other methods inside "ApiController". We can set the log level shown in the above code to "DEBUG" or "INFO" in case we need to debug our code or retrieve a variable. The lines below show the actual result of execution of above code.

```
    mysql >  select * from LOGS;
+--------+------------+-------------+-------+---------+
| USER_ID| DATED      | LOGGER      | LEVEL | MESSAGE |
+--------+------------+-------------+-------+---------+
|  root  | 2019-05-13 | ApiController |ERROR |  ERROR  |
+--------+------------+-------------+-------+---------+
1 row in set (0.00 sec)
```

5.4.3 Debugger tools

Now that we have seen the bug tracing via LOG table for our back-end application, we examine available tools to debug and test our front-end application.

Secnha Inspector is a debugging tool for troubleshooting and improving performance of Ext JS applications. Since *Secnha Inspector* is a proprietary tool developed by Sencha we will not go through details of this tool in this report.

As an alternative we can debug our front end application, using developer tools found in modern browsers.

Image shown in A.2 illustrates developer tools used to print some information on the debug console.

There are two ways to debug JavaScript code.

1. The first way, is to place console.log() in the code and see the value of the log, which will be printed in the console of the development tool.

2. The second way is by using breakpoints in the development tool. Following is the process.

   - Open the file in all the available scripts under script tag.
   - Now place a breakpoint to the line you want to debug.
   - Run the application in the browser.
   - Now, whenever the code flow will reach this line, it will break the code and stay there until the user runs the code by keys F6 (go to the next line of the code), F7 (go inside the function) or F8 (go to the next breakpoint or run the code if there is no more breakpoints) based on the flow you want to debug.
   - You can select the variable or the function you want to see the value of.
   - You can use the console to check the value or to check some changes in the browser itself.

Chapter 6

Conclusion and future work

Throughout this Report we discussed how to design a booking platform web application from a software engineering point of view. We discussed frameworks used inside Software companies in order to manage a group of software developers working on various parts of the software. After that we have seen dominant architecture for designing Web Applications and how different frameworks incorporate that architecture in order to make the code resusable and more efficient, as well as a logical separation between different parts of the code. Some of guidelines which were used during the development process came directly from User Experience research which we have included as part of this report. In addition we had to design a secure website therefore the usage of commonly tested platforms such as Spring and ExtJS helped us achieve that goal. Although this report summarizes the main aspects of developing a software solution from scratch, we had to sacrifice some details and technical perspectives which were not related to the main topic of this paper. One area that seems fruitful for further research and development is usage of Mobile Development platforms in order to further extend the availability and accessibility of the application, since websites by nature run on browsers and mobile platforms provide more robust tools for user interaction which leads to creation of a superior user experience. Another area which requires further studying is the usage of cross platform frameworks for development of digital platforms. These cross platform frameworks reduce the development workload by providing ready to use modules which work across variety of devices. In recent years *Google* and *Facebook* created their own cross platform development frameworks in form of *Facebook REACT* and *Google Flutter*. Considering the attractive nature of these frameworks for developers, the impact of such frameworks on digital platform development requires additional research.

Appendix A

Application Screens

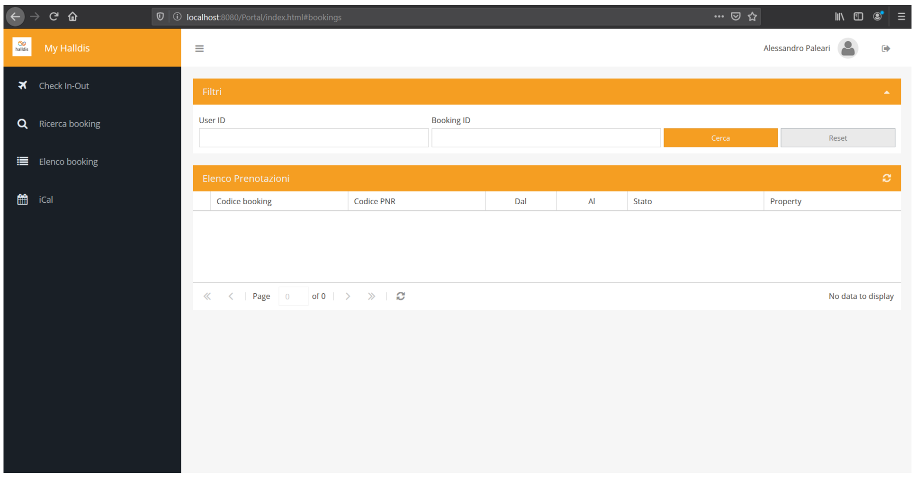
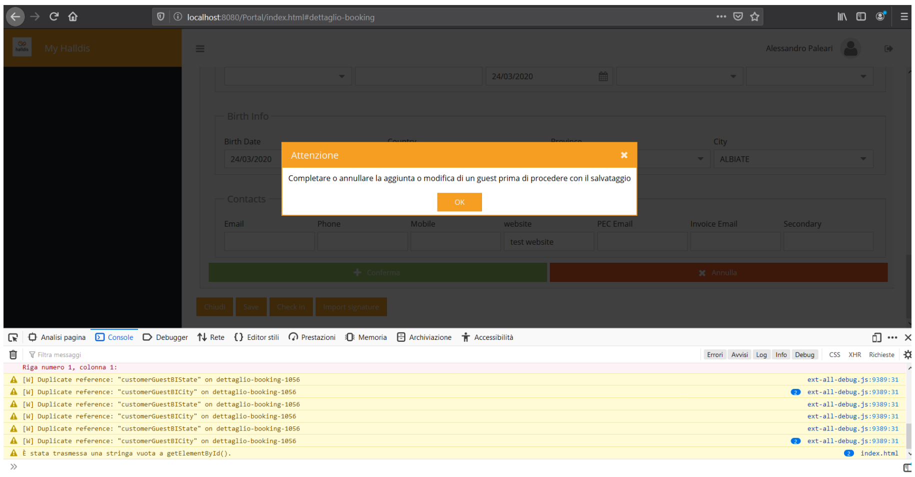


Figure A.1: Bookings page
Image illustrates Bookings management screen for platform administrator user.



Attenzione

Completare o annullare la aggiunta o modifica di un guest prima di procedere con il salvataggio

OK

```
Riga numero 1, colonna 1:
[W] Duplicate reference: "customerGuestBIState" on dettaglio-booking-1056        ext-all-debug.js:9389:31
[W] Duplicate reference: "customerGuestBICity" on dettaglio-booking-1056       2 ext-all-debug.js:9389:31
[W] Duplicate reference: "customerGuestBIState" on dettaglio-booking-1056        ext-all-debug.js:9389:31
[W] Duplicate reference: "customerGuestBICity" on dettaglio-booking-1056         ext-all-debug.js:9389:31
[W] Duplicate reference: "customerGuestBIState" on dettaglio-booking-1056        ext-all-debug.js:9389:31
[W] Duplicate reference: "customerGuestBICity" on dettaglio-booking-1056       2 ext-all-debug.js:9389:31
È stata trasmessa una stringa vuota a getElementById().                                     2 index.html
```

Figure A.2: Warning message for Booking details page
Image illustrates a warning message, indicating some information is missing from a booking detail.

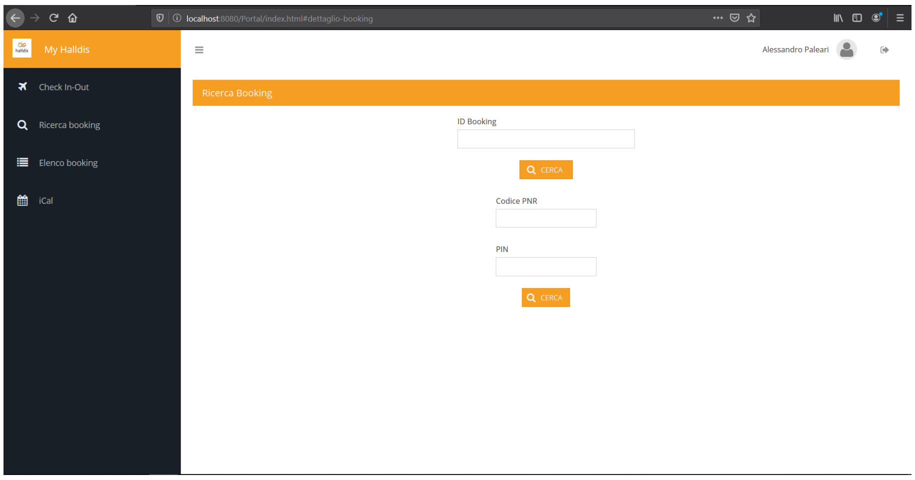
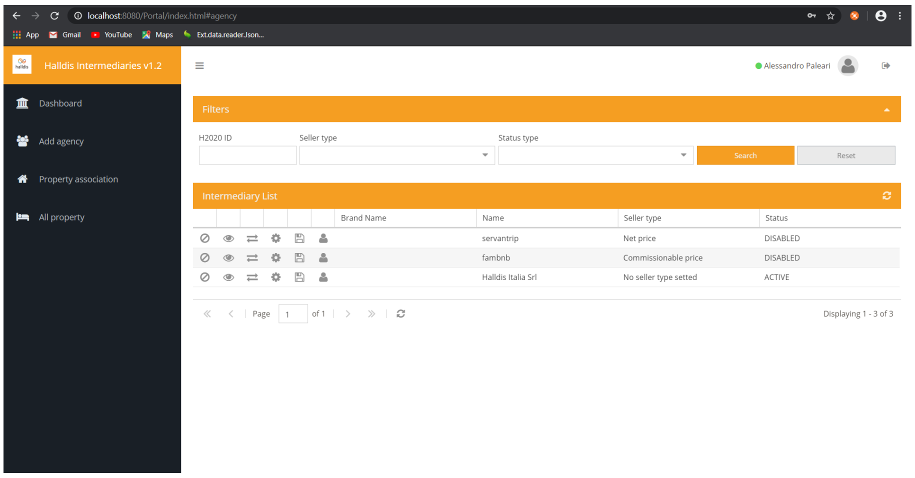
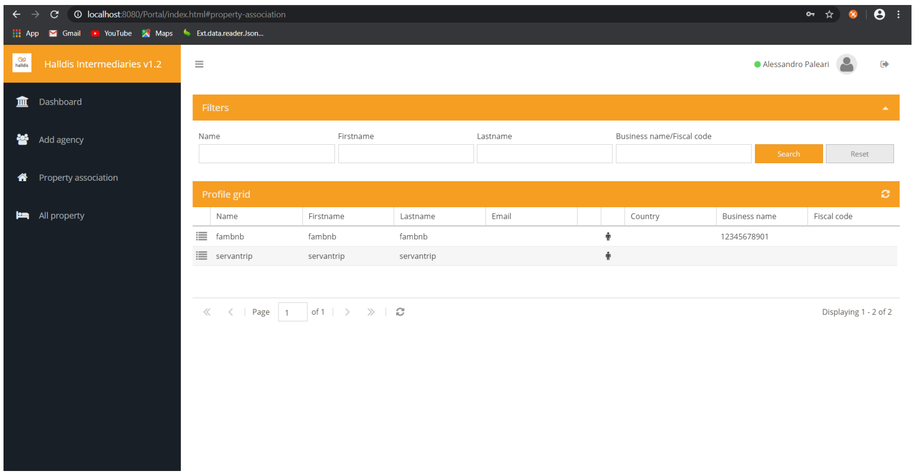


Figure A.3: Search Booking
Image illustrates Booking Search functionality for platform administrator user.



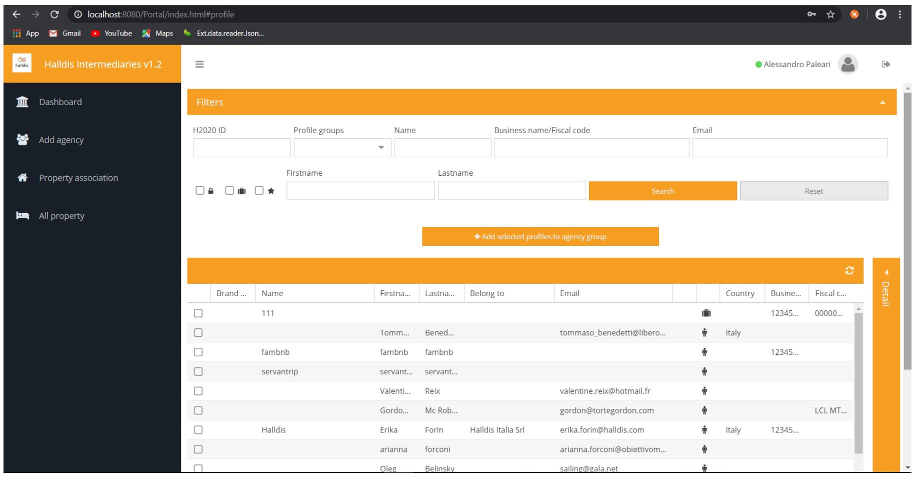
						Brand Name	Name	Seller type	Status
							servantrip	Net price	DISABLED
							fambnb	Commissionable price	DISABLED
							Halldis Italia Srl	No seller type setted	ACTIVE

Figure A.4: Property Owner(Agency) list
Image illustrates a list of Property Owners already registered, for the platform administrator user.



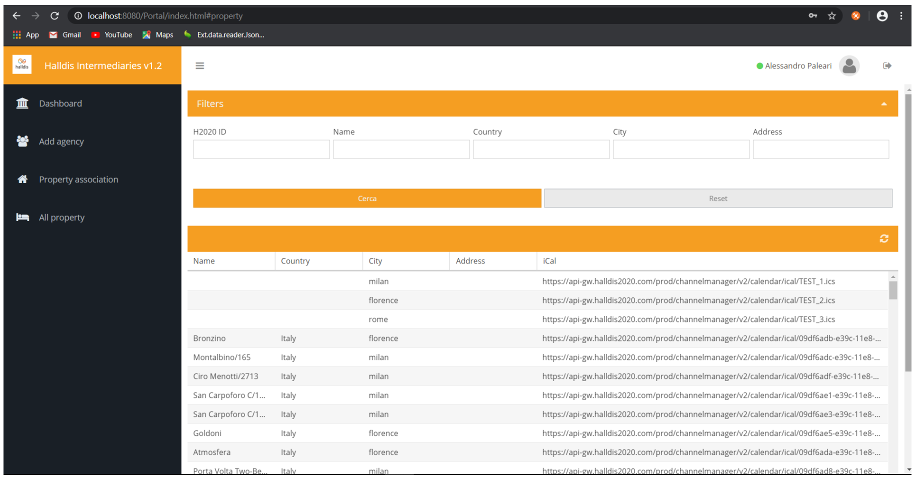
Name	Firstname	Lastname	Email		Country	Business name	Fiscal code
fambnb	fambnb	fambnb				12345678901	
servantrip	servantrip	servantrip					

Figure A.5: Property Owner Association
Image illustrates Property Owner association page for the platform administrator user.



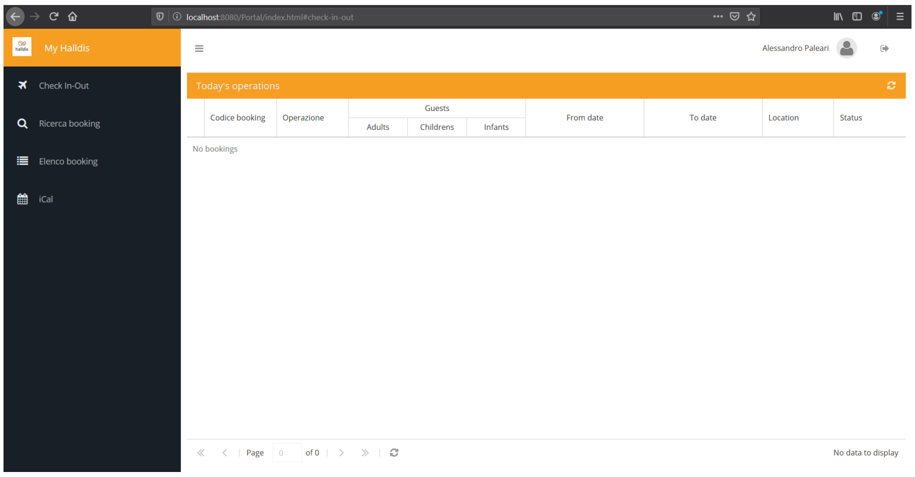
Brand ...	Name	Firstna...	Lastna...	Belong to	Email		Country	Busine...	Fiscal c...
	111							12345...	00000...
		Tomm...	Bened...		tommaso_benedetti@libero...		Italy		
	fambnb	fambnb	fambnb					12345...	
	servantrip	servant...	servant...						
		Valenti...	Reix		valentine.reix@hotmail.fr				
		Gordo...	Mc Rob...		gordon@tortegordon.com				LCL MT...
	Halldis	Erika	Forin	Halldis Italia Srl	erika.forin@halldis.com		Italy	12345...	
		arianna	forconi		arianna.forconi@obiettivom...				
		Oleg	Belinsky		sailing@gala.net				

Figure A.6: Guest Profiles
Image illustrates list of Guests (profiles) for the platform administrator user.



Name	Country	City	Address	iCal
		milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/TEST_1.ics
		florence		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/TEST_2.ics
		rome		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/TEST_3.ics
Bronzino	Italy	florence		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6adb-e39c-11e8-...
Montalbino/165	Italy	milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6adc-e39c-11e8-...
Ciro Menotti/2713	Italy	milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6adf-e39c-11e8-...
San Carpoforo C/1...	Italy	milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6ae1-e39c-11e8-...
San Carpoforo C/1...	Italy	milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6ae3-e39c-11e8-...
Goldoni	Italy	florence		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6ae5-e39c-11e8-...
Atmosfera	Italy	florence		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6ada-e39c-11e8-...
Porta Volta Two-Be...	Italy	milan		https://api-gw.halldis2020.com/prod/channelmanager/v2/calendar/ical/09df6ad8-e39c-11e8-...

Figure A.7: Properties List
Image illustrates list of properties registered for the platform administrator user.



Codice booking	Operazione	Guests			From date	To date	Location	Status
		Adults	Childrens	Infants				
No bookings								

Figure A.8: Check In/Check out Management
Image illustrates Check-In Check-Out management screen for platform administrator user.

Bibliography

[1] Jean-Charles Rochet and Jean Tirole. «Platform Competition in Two-Sided Markets». In: *Journal of the European Economic Association* 1 (Feb. 2003), pp. 990–1029. DOI: 10.1162/154247603322493212.

[2] Alfonso Lamadrid de Pablo. «The double duality of two-sided markets». In: *Swedish Competition Pros and Cons conference* (Feb. 2014), p. 5. URL: https://antitrustlair.files.wordpress.com/2015/05/the-double-duality-of-two-sided-markets_clj_lamadrid.pdf.

[3] Jean-Charles Rochet and Jean Tirole. «The RAND Corporation Two-Sided Markets : A Progress Report». In: 2005.

[4] M Munger. «Coase and the 'sharing economy.In Forever Contemporary, ed. Cento Veljanovski». In: 2015, pp. 187–209.

[5] Fiona Greig Diana Farrell. «The OnlinePlatform Economy Has Growth Peaked?» In: *JPMorganChase Institute* (Feb. 2016), p. 4. URL: https://www.jpmorganchase.com/corporate/institute/document/jpmc-institute-online-platform-econ-brief.pdf.

[6] Colin M. Gray et al. «The Dark (Patterns) Side of UX Design». In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, 2018, pp. 1–14. ISBN: 9781450356206. DOI: 10.1145/3173574.3174108. URL: https://doi.org/10.1145/3173574.3174108.

[7] Yao-Wen Huang and D. Lee. «Web Application Security—Past, Present, and Future». In: Jan. 2005, pp. 183–227. DOI: 10.1007/0-387-24006-3_12.

[8] Dave Wolf and A. Henley. «What Is MVC?» In: Dec. 2017, pp. 23–25. ISBN: 978-1-4842-3194-4. DOI: 10.1007/978-1-4842-3195-1_5.

[9] Stephen Kaisler and Frank Armour. «Design Patterns». In: Jan. 2002. ISBN: 9780471028956. DOI: 10.1002/0471028959.sof089.

[10] Joanna Patrzyk et al. «Towards A Novel Environment For Simulation Of Quantum Computing». In: *Computer Science* 16 (Jan. 2015), p. 103. DOI: 10.7494/csci.2015.16.1.103.

[11] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. «RESTful Web Services vs. "Big" Web Services - Making the Right Architectural Decisions». In: Apr. 2008, pp. 805–814. DOI: 10.1145/1367497.1367606.