POLITECNICO
MILANO 1863

Travlendar+
Implementation And Testing

Fumagalli Paolo, Grotti Pietro, Gullo Marco

January 7, 2018

https://github.com/Aestor/FumagalliGrottiGullo

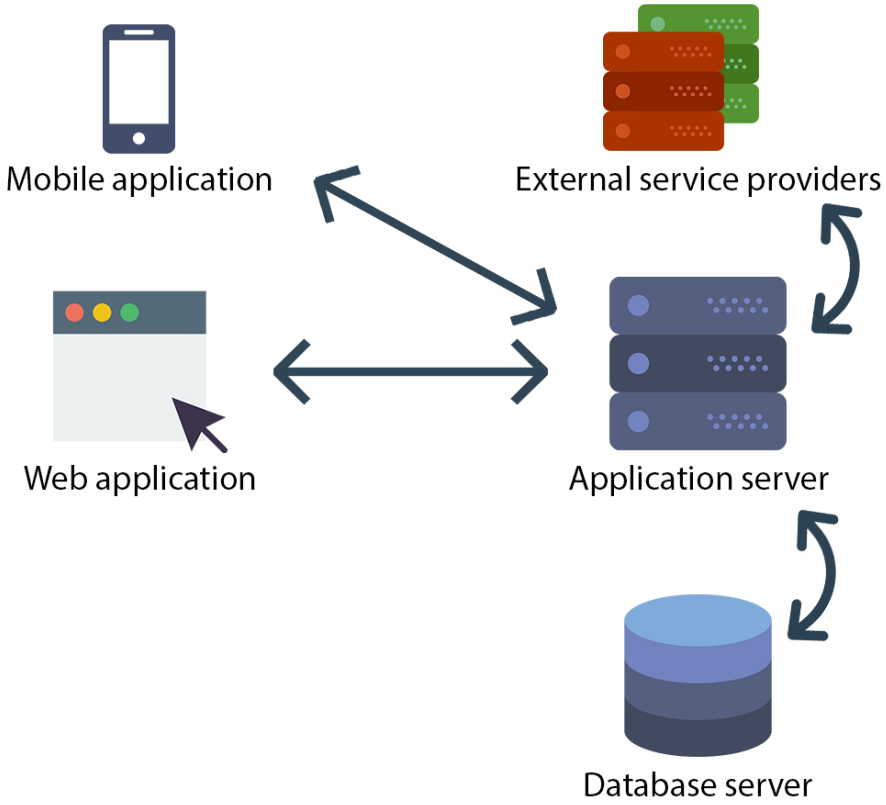
Contents

1 Introduction and Scope

1.1 Purpose

The purpose of this document is to provide a detailed description of what we proudly call the first prototype of the project Travlendar+. Its appearance and design aspects are still raw and not really good-looking, but it contains most of the functionalities we underlined as essential in the previous documents. In the following sections a detailed tour of our work will be given, backed up with brief summaries of the frameworks and tools chosen for the development. There will be also a section dedicated to the software testing we produced.

1.2 Scope



In the development process described in the previous document, the so-called Design Document, it was taken into account that Travlendar+ will be both a mobile and web browser application. Time for developing was limited, it was established by the team to work only on the Web application. Therefore, Mobile application is considered a future expansion of the project. From the picture above it is possible to have a simplified idea of the high-level architecture, all the elements and their interactions. Regarding External Service providers, it was taken advantage of the free APIs available on the market, but it was not possible to find all the ones needed this way. So it was decided to simulate RPC calls for the services left uncovered. In the project repository there are some packages, labeled as fake, written down with this purpose.

The Application Server was entirely developed, but slightly differs from the picture: it is divided into two parts that will run in two different machines (though they will be presented into the same folder). This was done in order to be compliant with our choice of a 4-tier architecture (see the Design Document for further details); that would lighten future mobile version developers job , too. With this bonds, Java EE was considered the best fit, and used for both parts. The Database Server is a normal SQL Server, while queries are handled by a JDBC Connector, provided by Spring . For now it was used only one database, since the amount of storage data is still low. In the future there might be one or more backups, and for management purposes a Data Warehouse. It was made sure that only beans in the back end of the Application Server could interact with the SQL one, but no security for access was provided yet.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

1.3.2 Acronyms

1.3.3 Abbreviations

RASD : Requirement Analysis and Specification Document
DD : Design Document
RPC : Remote Procedure Call
API : Application Programming Interface
Java EE : Java Enterprise Edition, Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software.
JB : JavaBeans
SQL : Structured Query Language
JDBC : Java DataBase Connectivity
CDI : Context Dependency Injection
POJO : Plain Old Java Object
IDE : Integrated Development Environment
RIA : Rich Internet Application
IoC : Inversion of Control

1.4 Revision History

Version 1.0

1.5 Reference Documents

Apache Project's Documentation
Vaadin Documentation
Java EE - "Your First Cup: An Introduction to the Java EE Platform", Oracle 2012.
Spring

1.6 Document Structure

The first part will be about all frameworks used in working at this project, with some screenshots and documentation.
The second section is about how code is divided into the different packages. It will show all the components, views, JBs, UI and so on. Some screenshots are embedded. Here it is written also about the software components designed in the DD.
Then in section 3 will talk about all the goals and requirements, taken from RASD, and their eventual implementation/satisfaction, evaluated according to the design promises of the DD . Section 4 will cover all the Testing part, while the other sections will complete the project schema with some details. To get more confident with all the frameworks and tools, it is possible to read an appendix at the end with some background knowledge.

2 Overall Description

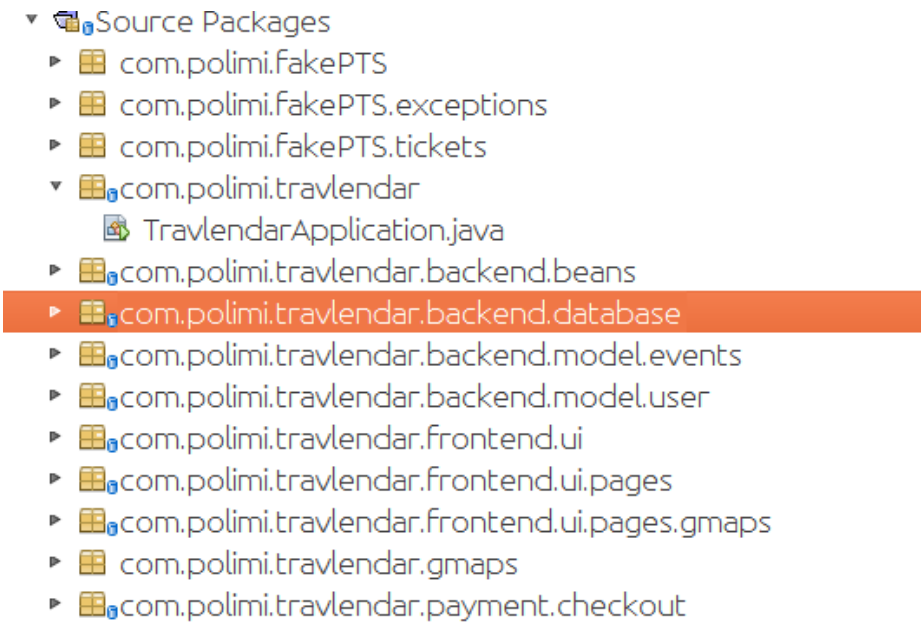
The project was implemented as a SpringVaadin Web Application. The programming language established for this purpose is Java 8, but some small features are written in JavaScript and HTML5. The main reason why we chose to combine Spring with Vaadin is because it is possible to take advantage of both frameworks without any interference between the two. The main benefits were:

- It was possible to avoid working too much on HTML and CSS, but focus more on the application logic. All the code in java is translated in HTML, CSS and JavaScript by Vaadin.
- Vaadin is designed to take care of all of the RIA complexity. From communication to browser quirks and rendering it is all handled by the framework.
- Easy binding to Spring data sources. It was easy to inject existing services into any part of the Vaadin UI component tree and bind the components to the data.
- Thanks to Spring Boot it was possible to get a fully configured environment, including a standalone server, and to start adding content to the UI class right away.
- The IoC container at the ore of Spring libraries makes it easier to write software which is more modular, reusable, maintainable and testable.
- Cloudy-ready applications. Code portability through decouplng and the rich toolbox of features in Spring make it easy to forklift applications from the local development environment into any web container and cloud platform (for the future development of Travlendar+).
- Spring made CDI (*@Autowired*) easy to implement.
- Spring has a wide range of useful libraries.

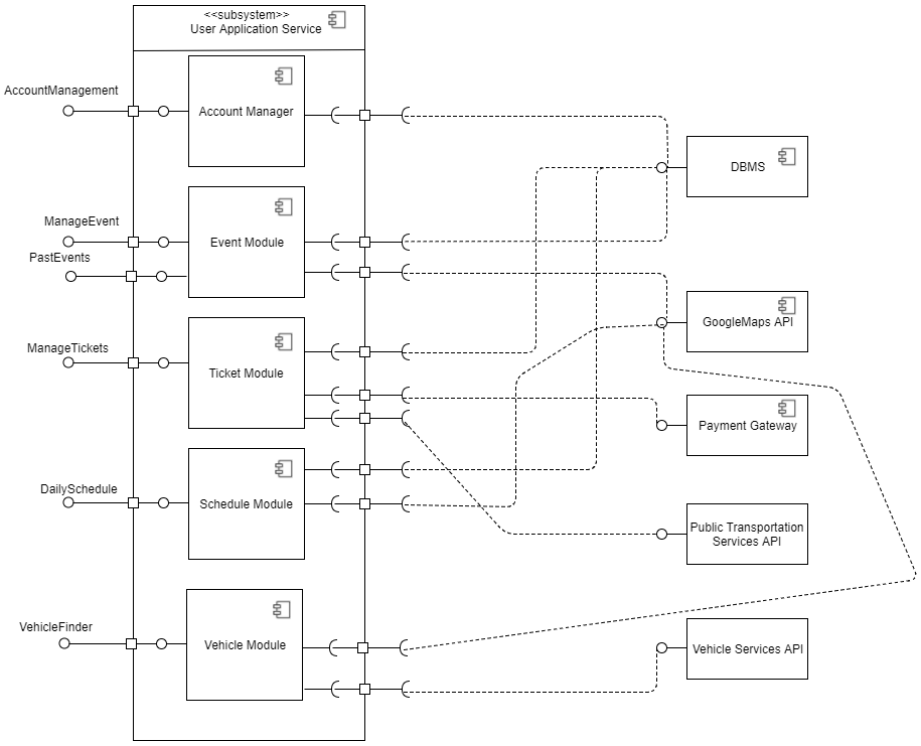
To get a deeper understanding of these modern concepts of web design, such as IoC, CDI and so on, we suggest to look at the links proposed in the Reference Documents section and skim the Appendix. However, a simplified snapshot of the code will be given here.
In Vaadin server-side applications, all pages to be shown to the clients are loaded dynamically in one singleton instance of the class UI (custom UIs like in this project extends it). Anytime a client connects to the server a dedicated UI is instantiated. This UI containts a "navigator" object that allows navigation of the various contents. The navigator is called anytime certain events are triggered: a click for example, or when some response must be shown. The various pages are java classes that are initialized by the navigator. All web components, such as text fields, cheboxes and images, are defined in pages, that

act like views in a Model-View-Presenter pattern. In the code of these classes, which can be found in the package *com.polimi.travlendar.frontend.pages* (see the following section), it is possible to embed Java Beans, Spring Components and Services, that elaborate all the data input given by clients and are responsible of the application business logic. This connection is made possible by the tag *@Autowired*, provided by Spring, similarly to some JavaEE tags. This brief description can be completed by reading the sections on Vaadin and Spring in the appendix.

3 Structure of the source code



The picture above displays the packages and their organization. FakePTS folder represents the simulation of a generic Public Transportation System's java API (such as *ATM*, *AMT*, etc...). It contains all the classes and methods to generate valid tickets (which are POJOs) and validate them electronically.
The Spring Boot Application starter class is called *TravlendarApplication* and is in the main folder *travlendar*. Running it from any Java IDE will start the application. The code of the application is divided into two subfolders: *backend* and *frontend*.
The first one hosts the code that will be deployed in the Application Server, which includes the model of the objects necessary for the functionalities, along with the Business tier (the Beans). It also contains the folder *database*, where all the elements to extract and insert data into the DB server are contained.
The latter contains all the elements for the user interface: the UI class, the classes for the web pages and other UI components.
The package *gmap* contains all the elements to enhance the dialogue with the Google Maps API and services, while the *checkout* package contains the code for the collaboration with Stripe APIs.
In the picture below it is possible to se the modules that were defined in the DD:

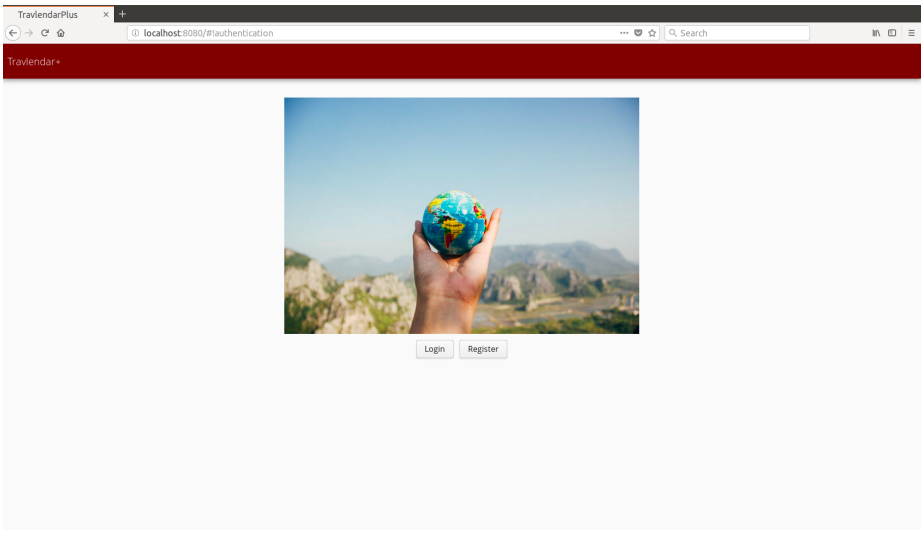


Each module of the subsystem *User Application Service* was implemented in the same way: one single session bean with all the methods to handle requests formulated by the front end components, backed up by some model elements. Therefore, modules are not visible in the packages' structure at first sight. For example, the first module on top *Account Manager* is made of the bean named *UserService* and some calsses in the model folder such as *User*, *UserSettings*, *UserPreferences* etc.
For now, the *Event Module* and the *Schedule Module* are fused together.
The external modules (shown on the right side of the picture) were defined independently and can easily be mapped 1:1 with the packages described above.
The module *Vehicle Module* was not implemented into the project.

3.1 Pages

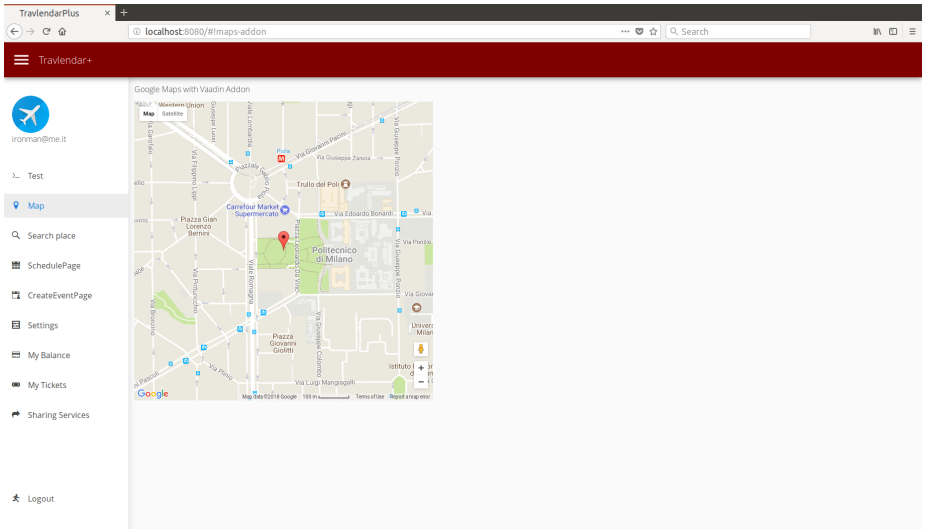
3.1.1 Login/Register

This section will show screenshots of the website's pages.



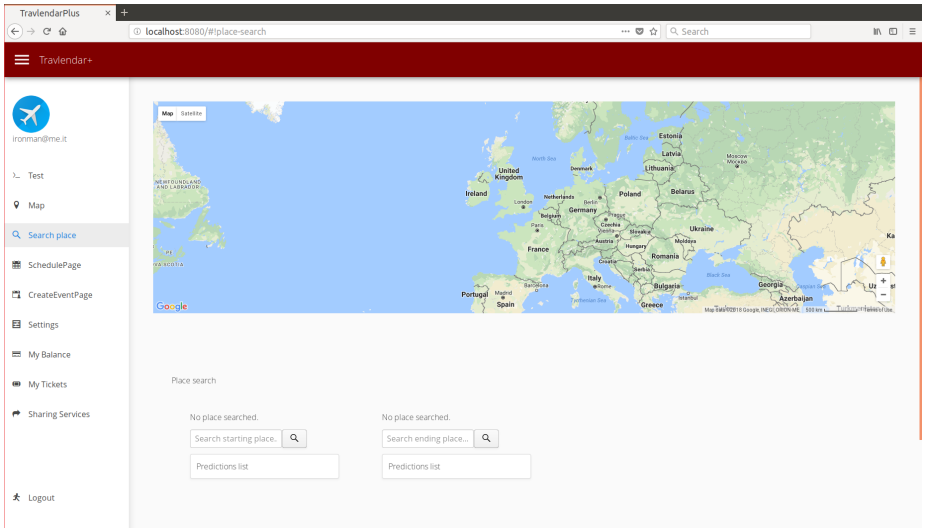
This is the first page that will appear when the application is started. It allows registration and login.

3.1.2 Map



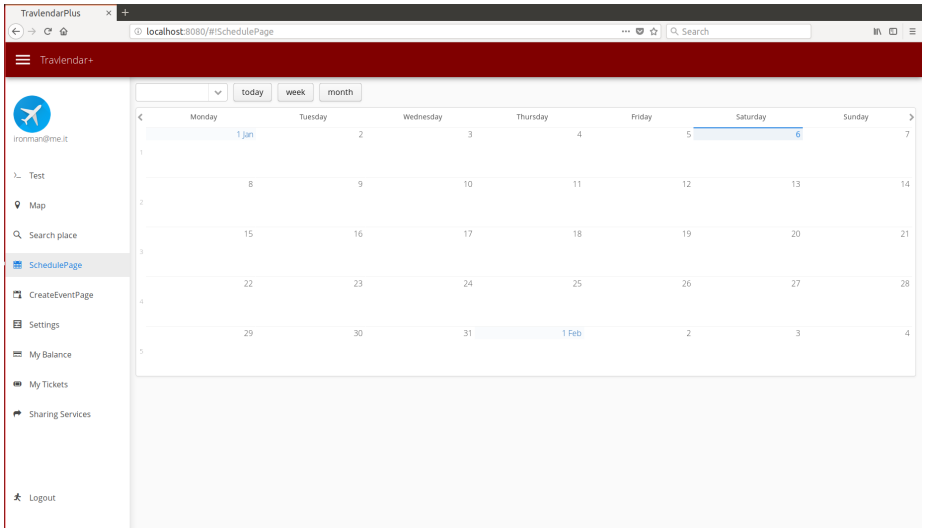
Once logged in, the menu bar appears on the left, showing all the pages of the website. The first one is the map: here the user can see his roundabouts.

3.1.3 Place Search



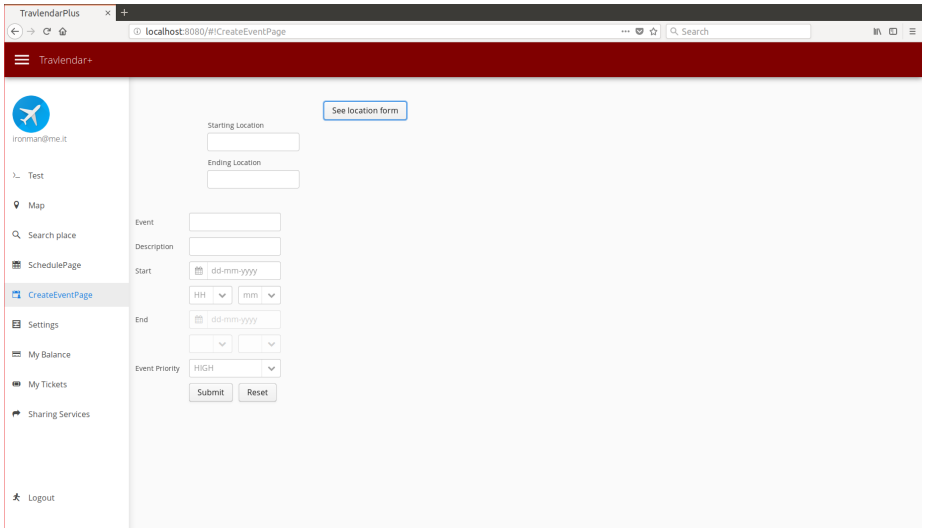
The second page is for place search.

3.1.4 Schedule



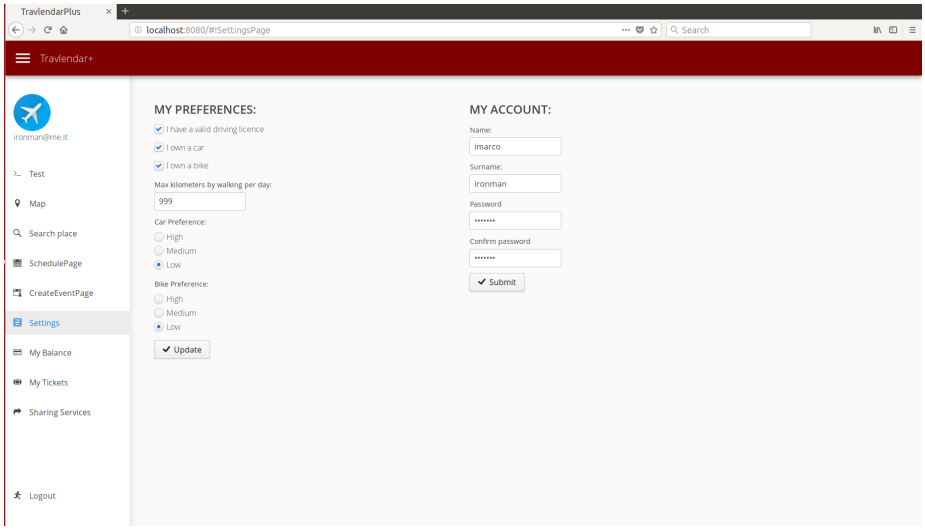
The third page contains the schedule. Here the user can see his agenda and the events he created. Selecting an event with the mouse will give you the option to modify or delete it.

3.1.5 Create Event



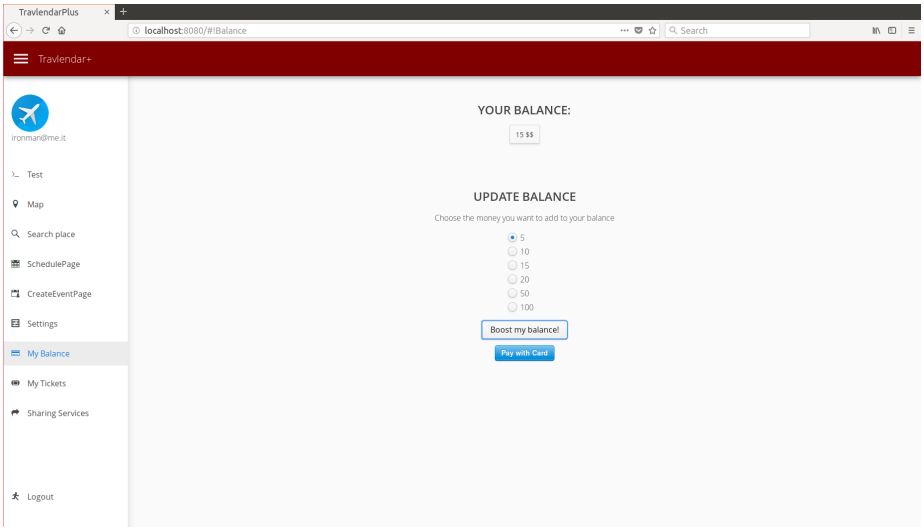
In this page the user can create new events. By clicking on the *See location form* a window pops up with the possibility of selecting where you will be located when you head out to go to the event and where the event will be.

3.1.6 Preferences



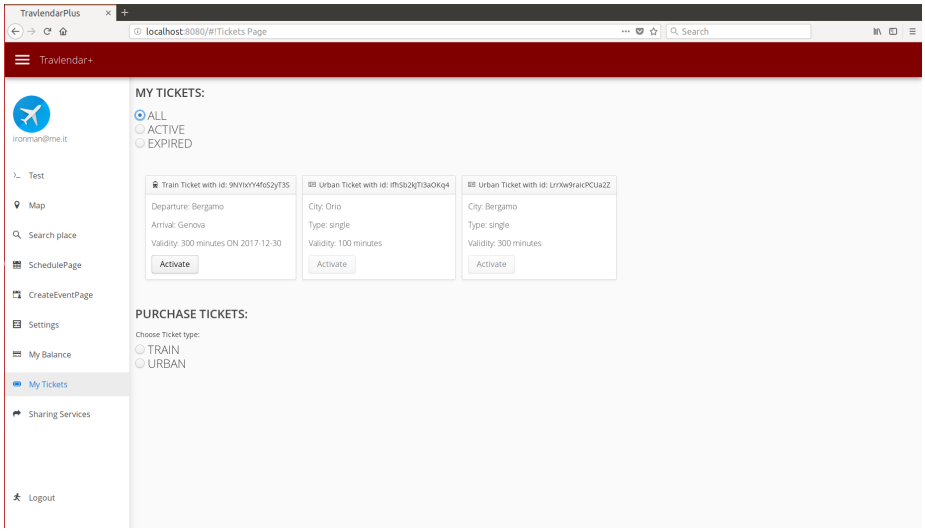
Here are loaded all the user's customizable preferences and account details. Update and Submit buttons will make the system save the selected ones.

3.1.7 Balance



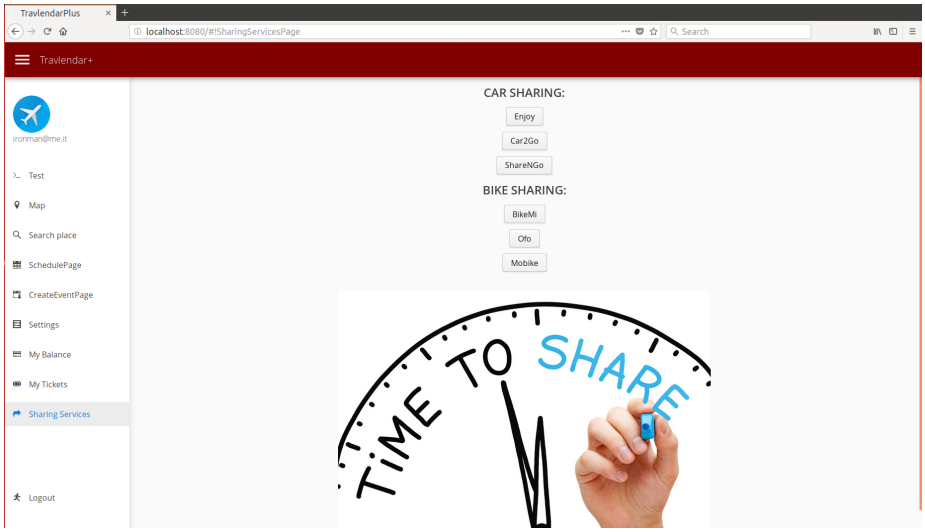
My Blanace Page concerns Travlendar's currency. Every user will have a balance which can be increased by clicking on *Pay with Card* and filling all the credit card required fields that pop up afterwards.

3.1.8 Tickets



This is the area where it is possible to purchase tickets through the application (balance currency is needed for this operation) and to access the ones owned (that may be active or not).

3.1.9 Sharing Services



The last page is to advertise the car and bike sharing services Travlendar+ has agreements with. In the future this page will directly show vehicles on the map.

4 Requirements and Functionalities

4.1 Functional Requirements

In this section we will map the first prototype of Travlendar+ with all the goals and requirements, specified in the RASD, in order to highlights which functionalities are ready-to-use and which ones will be implemented or improved in the future.

4.1.1 Goal 1

G1: Allow a guest user to register to Travlendar+ by filling the registration form with the data needed.

- **R1:** A guest user can create a new account through the registration process by providing his credentials to the system. An e-mail will be necessary and it must be unique.
- **R2:** At the end of the registration process the system will send an e-mail to the user to verify the account.
- **R3:** The account must be verified with the link provided by the system before it becomes active.
- **R4:** A guest user can log in the application with his credentials.

The application has its own register/login component that works. For now only input data are verified and not personal email existence. We didn't configure a server mail yet, which means that the system does not send a confirmation email but immediately registers the new user into the database.

4.1.2 Goal 2

G2: Allow the user to select preferences and modify them whenever he wants.

- **R5:** A registered user can modify his preferences by choosing the preferred movement system at any time.
- **R6:** A registered user can select a specific time during the day (or night) after which he doesn't want to take public transportation.
- **R7:** A registered user can choose a maximum distance that he is willing to walk to move from one place to the other.

Travlendar+ contains a dedicated page, called *Settings Page*, where the user can customize his preferences. All the requirements are satisfied.

4.1.3 Goal 3

G3: Allow the user to easily create an organized and customizable agenda based on his preferences.

- **R8:** A registered user can create an event by filling the time, date, duration and location of the event.
- **R9:** A registered user can select a time slot during the day in which he wants to take a break, he can also choose the length of the break (which doesn't have to necessarily be equal to the time slot).

This goal has been achieved, the application contains a calendar in which users can create events and modify them, by giving the application the information about where they will be before going to the event, where the event will be, and the starting and ending time of the event. Lunch breaks are considered a future development feature.

4.1.4 Goal 4

G4: To help the user plan his movements in a clever and efficient way.

- **R10:** The user has to declare if he has a driving license.
- **R11:** The user can declare if he owns a personal vehicle and indicate which kind of vehicle (car, motorbike, bike).
- **R12:** The system has to calculate a route to move araound the city based on the user's preferences.
- **R13:** The system will have to adjust accordingly to unexpected events (accidents, strikes, weather, natural disasters) changing the route of the day and notifying the user.

The algorithm for path suggestion is not optimal, but it still gives the fastest path to the selected location. The other requirements are achieved.

4.1.5 Goal 5

G5: To guarantee the user no to be late for his appointments.

- **R14:** The system will always try to have the ETA to an event at least 10 minutes earlier than the beginning of the event.
- **R15:** The system will notify the user if one appointment is not reachable, this feature has to update the user in real time, which means that if an event becomes unreachable it will notify the user immediately.

This goal was not achieved and will be developed in the future. The system does currently deal with real time changes.

4.1.6 Goal 6

G6: To let the users buy bus or train tickets (both single or seasonal).

- **R16:** The system will support the acquisition of valid tickets for public transportation.
- **R17:** The user can link h is account to a credit card to pay for tickets.
- **R18:** The user can open a balance connected to his account that can be used to pay for tickets.
- **R19:** The balance can be charged with credit cards or with cash, by going to shops that are certified by the application.
- **R20:** After being bought a ticket will be flagged as "Available".

This goal has been achieved. The application has a page in which the user can buy new tickets that can be used. Even though the PTS APIs are faked, the system is open and ready for a real collaboration about payments. The credit and debit cards are not saved in the database, the user will have to submit the details everytime. Tickets can be purchased only with Travlendar+'s balance value. This way all the money transactions pass through the system (in particulare the payment gateway module).

4.1.7 Goal 7

G7: To let the user find vehicles from vehicle sharing systems.

R21: The system must be able to locate vehicles from vehicle sharing systems.

This feature was not implemented. At the moment the application will redirect the user to external websites of different car and bike sharing systems.

4.1.8 Goal 8

G8: Allow the user to buy in advance tickets which can be used later on.

R21: An "Available" ticket can be activated.

R22: After a ticket is activated it will be flagged as "Active".

R23: Once the time expires on an "Active" ticket it will be flagged as "Expired".

R24: An "Expired" ticket can't be activated.

This goal has been achieved. Tickets are stored in Travlendar+'s databases and can be activated at any moment after their purchase. The expiration is handled too.

4.2 Non-functional Requirements

3.4.1 Extensibility

Since Travlendar+ will be developed from scratch, it will be first launched with only the essential characteristics, its core goals, but is meant to be open to future additional features and improvements. Therefore, the system shall provide an API, a way for programmers to create own application which can interact with the systems data and functionalities. This choice implies a more complex security system, in order to avoid developers to access sensitive users data. More specific information will be given in the design document.

3.4.2 Performance

The aim of Travlendar+ is to become a large-scale used application. The system must be able to handle a great number of simultaneous requests and respond correctly in the best time possible.

3.4.3 Reliablity

System must work ideally 24/7, but a short period of time should be dedicated to daily maintenance.

3.4.4 Security

Users credentials, personal information, payment data will be stored and must be protected.

3.4.5 Accuracy

System must be accurate in computing routes and in estimating time necessary of any movement from a place to another. For this purposes very precise maps must be used, the system also must take account of GPS accuracy (GPS technology is theoretically able to provide locations with an error less or equal to 7.8 meters).

Regarding non-functional requirements, the project has successfully satisfied the first three. The code is shareable on open-source, can ideally run non-stop and does not generate performance issues. The other two have not been implemented only because we didn't have to deal with them yet. They are flagged as open issues, but the developers are confident that with a little improvement they can be easily achieved.

5 Testing

5.1 Premise

Due to delays in the applications development, allowed time for testing happened to be really short. In the Design Document there is a Gantt diagram showing a prediction of project timeline, which could not be satisfied due to a project setup (framework choice and time for getting confident with them) that took more that the developers expected. Therefore, we focused on set up a good testing base, to start with. Jacoco report tool was added. Unit and Integration testing can be run separately. JMeter is present in the dependencies as well, but could not be completely configured yet.

5.2 Results

The unit test base is made of about 230 tests, that cover around 70-75% of the java classes (via SonarQube report).
Integration test base, since java beans are not many, contains about 20 test.

5.2.1 Unit Testing

Unit testing was made with an automatic tool called EvoSuite. This software is for JUnit, which worked fine for a java web project. Some tests were slightly changed but the majority was considered good enough for the purpose.

5.2.2 Integration Testing

As explained in the section 4 (source code), modules are made of JBs and some support classes. In the design document we speculated that various modules would need integration, but in the actual project they work separately all the time. Only integration was with the DMBS Module.

5.2.3 System Testing

JMeter still needs configuration. For future testing it is suggested to add Gatling tool, since it is the one referenced by Vaadin documentation.

6 Installation Instruments

6.1 Configuration

In short

Import in your IDE as a Maven project.
You need to execute mvn install. Some IDEs (e.g.: Eclipe JEE) automatically execute it when you run the project. Run the class TravlendarApplication.java as a Java Application.
From your browser navigate to localhost:8080.

6.2 Database Configuration with MySQL Workbench

To import the database in your PC:

1. Install MySQL Workbench.
2. In the program: Server -> Data Import.
3. Select "Import from Self-Contained File" and browse Dump.sql.
4. Create a new schema for the Default Target Schema and name it "travlendar" (important).
5. Refresh schemas to see travlendar.

6.3 Database Configuration with Command Line

Alternately, create manually a new schema with MySQL Command Line:
Copy and Paste commands from the file *"sql_entries"* that is accessible in the folder *Implementation*.

6.4 Configure JDBC

To access the database, the application reads the information in the file *src/main/resources/application.properties*. By default, the application tries to connect to localhost:3306 with username "admin" and password "root", which are the standard settings of MySQL Workbench.

6.5 Usage

More useful advice on how to navigate through the application can be found in the README.md of the project.

7 References

- Vaadin + MySQL
- Vaadin layout components
- Vaadin server-side app
- SQL Syntax
- Vaadin login
- Integration testing in Maven
- Jacoco
- Testing in Spring: the following documents
  https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html
  http://www.baeldung.com/spring-boot-testing
  https://techbeacon.com/how-extend-java-unit-tests-components-spring-mockito
- EvoSuite Maven Plugin Tutorial
- Stripe for Spring
- App-Layout Vaadin Addon Documentation
- EvoSuite

8 Effort Spent

- Team work: $\sim$ 120 hours.
- Individual work: $\sim$ 110 hours each.

A Background Knowledge

In this section we provide an idea of the usage of all the main frameworks and software tools which are responsible of Travlendar+ first prototype's shape and code. We will also give an insight on the impact each framework had on the project.

A.1 Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the comncept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

- Making the build process easy.
- Providing an uniform build system.
- Providing quality project information.
- Providing guidelines for best practices development.
- Allowing transparent migration to new features.

Version: 3.5.2

Advantages:

- It was very easy for the developers to add APIs, Vaadin Elements, Spring features and all kind of pre-built packages.

Drawbacks:

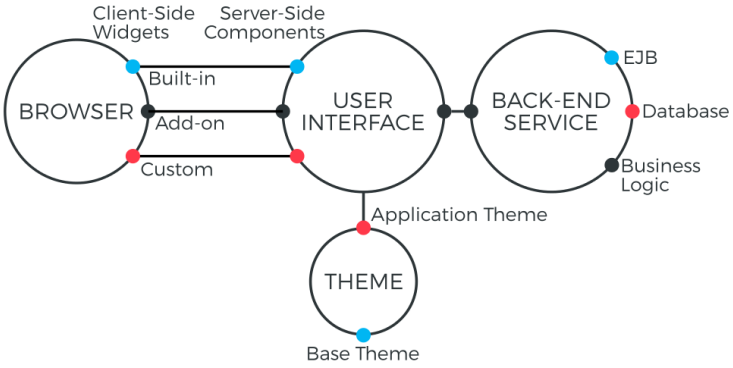
- None in our project.

In our project: see pom.xml file in the code's folder.

A.2 Vaadin

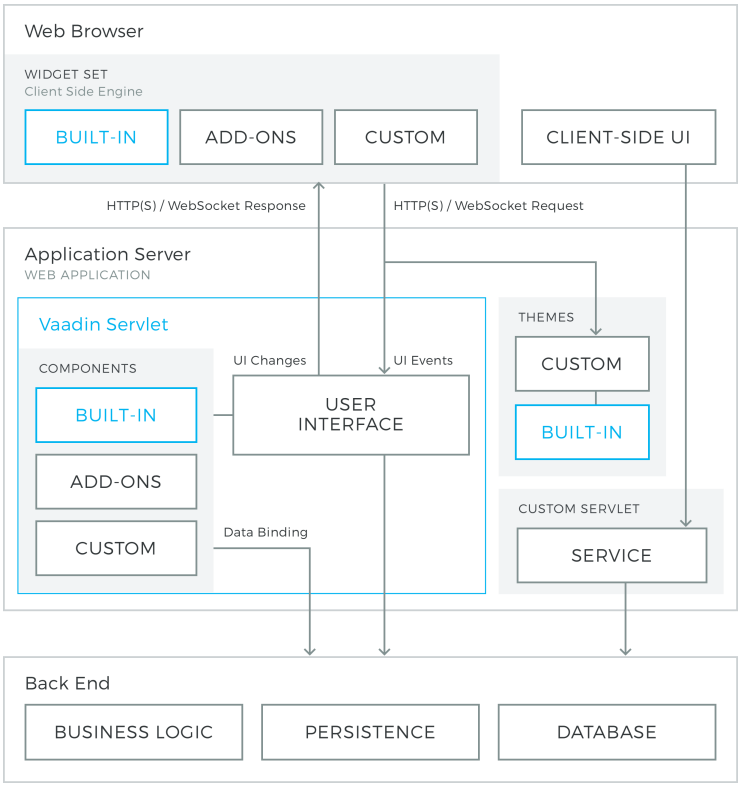
Vaadin Framework is a Java web application development framework that is designed to make creation and maintenance of high quality web-based user interfaces easy. Vaadin supports two different programming models: server-side and client-side. The server-driven programming model is the more powerful one. It lets you forget the web and program user interfaces much like you would program a desktop application with conventional Java toolkits such as AWT, Swing, or SWT. But easier.

While traditional web programming is a fun way to spend your time learning new web technologies, you probably want to be productive and concentrate on the application logic. The server-side Vaadin framework takes care of managing the user interface in the browser and the AJAX communications between the browser and the server. With the Vaadin approach, you do not need to learn and deal directly with browser technologies, such as HTML or JavaScript. (Read more at https://vaadin.com/docs/v8/framework/introduction/intro-overview.html)

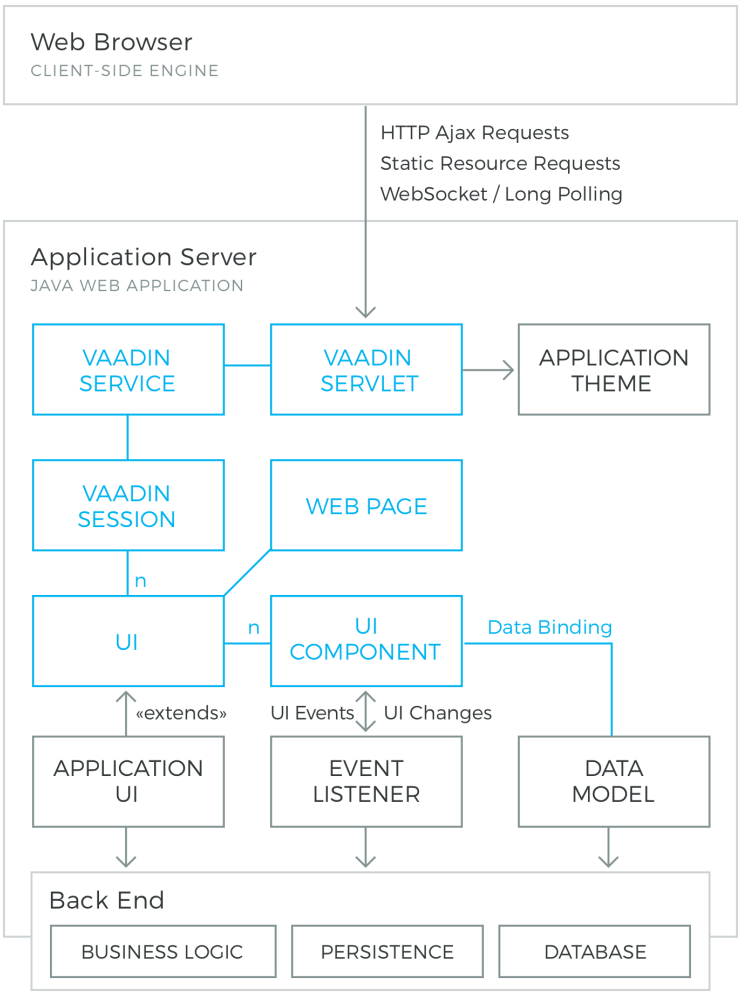


Vaadin Framework consists of a server-side API, a client-side API, a horde of user interface components/widgets on the both sides, themes for controlling the appearance, and a data model that allows binding the server-side components

directly to data. For client-side development, it includes the Vaadin Compiler, which allows compiling Java to JavaScript.



A server-side Vaadin application runs as a servlet in a Java web server, serving HTTP requests. The VaadinServlet is normally used as the servlet class. The servlet receives client requests and interprets them as events for a particular user session. Events are associated with user interface components and delivered to the event listeners defined in the application. If the UI logic makes changes to the server-side user interface components, the servlet renders them in the web browser by generating a response. The client-side engine running in the browser receives the responses and uses them to make any necessary changes to the page in the browser.



Version: 8.1.7

Advantages:

- Vaadin masked many basic but complex and long aspects of Java Web programming, allowing developers to concentrate on the functionalities of Travlendar.
- Some non-functional requirements were given for granted by the framework.
- Vaadin community is big, with a very crowded forum; it was helpful.
- Many Vaadin built-in components were available.

Drawbacks:

- It was hard to make Vaadin and custom Javascript files work together.
- It took time to get confident with the environment and the high number of packages/elements.

In our project: used to build the bones of the Application Server, both presentation and business logic.

A.3 Spring Boot Framework

Spring is framework open source for Java Web Applications. Spring Boot is the starting point for building all Spring-based applications. Spring Boot is designed to get you up and running as quickly as possible, with minimal upfront configuration of Spring:

- Build anything - REST API, WebSocket, Web, Streaming, Tasks, and more.
- Simplified Security.
- Rich support for SQL and NoSQL.
- Embedded runtime support - Tomcat, Jetty, and Undertow.
- Developer productivity tools such as live reload and auto restart.
- Production-ready features such as tracing, metrics and health status.

Version: 1.5.9

Advantages:

- Suits well with Vaadin.
- Added more pre-build features.

Drawbacks:

- It is quite complicated.
- It needed time to be understood.

In our project: used some features, especially the ones concerning CDI, beans and database.

A.4 Stripe Checkout

Stripe builds the most powerful and flexible tools for internet commerce. Whether youre creating a subscription service, an on-demand marketplace, an e-commerce store, or a crowdfunding platform, Stripes meticulously-designed APIs and unmatched functionality help you create the best possible product for your users. Checkout is an embeddable payment form for desktop, tablet, and mobile devices. It works within your sitecustomers can pay instantly, without being redirected away to complete the transaction.

Version: 5.25

Advantages:

- The API has its java version: it was very easy to use.

Drawbacks:

- Checkout presentation is made with a javascript file that was hard to insert in the project.

In our project: the payment gateway module is made with the help of Stripes API and Checkout element.

A.5 SQL

SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system. Originally based upon relational algebra and tuple relational calculus, SQL consists of many types of statements,which may be informally classed as sublanguages, commonly: a data query language (DQL), a data definition language (DDL), data control language (DCL), and a data manipulation language (DML) . The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

Version: 5.7.0

Advantages:

- Developers were familiar with the language.

Drawbacks:

- Nowadays it is considered old technology.

In our project: used in all the database part.

A.6 Java Enterprise Edition

Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community

Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.

IDE Eclipse Oxygen and Netbeans 8.2

Advantages:

- Various and user-friendly IDEs.

Drawbacks:

- None

In our project: it was the development environment and basic toolkit.
.