AY 2020/2021

Politecnico di Milano

Implementation Document

Luca Pirovano Nicolò Sonnino

Professor
Matteo Rossi

Version 1.0
January 29, 2021

Contents

1 Introduction

1.1 Purpose

This document aims to describe how the implementation and integration testing took place. Implementation is the last step of the CLup application development cycle.

Testing, instead, means check that the critical parts of the application works in a correct way, as described in the DD document.

The code and the releases can be find on the official CLup repository hosted on GitHub, reachable at this link:
https://github.com/PiroX4256/SE2-Piemonti-Pirovano-Sonnino.

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Definitions

1.2.2 Acronyms

- **DD:** Design Document
- **RASD:** Requirements Analysis and Specification Document
- **S2B:** Software To Be
- **DTO:** Data Transfer Object, represents a link between the user input and a Java Object.

1.2.3 Abbreviations

1.3 Revision history

1.3.1 References

- **JavaScript:** https://www.javascript.com
- **Java:** https://www.java.com/it
- **Spring Framework:** https://spring.io
- **Spring Security:** https://spring.io/projects/spring-security

- **Spring Social:** https://projects.spring.io/spring-social
- **Vue.js:** https://vuejs.org
- **jQuery:** https://jquery.com
- **QRcode.js:** https://jquery.com
- **Semantic UI:** https://semantic-ui.com
- **Leaflet.js** https://leafletjs.com
-

2 Development

2.1 Implemented Functionalities

With respect to the RASD and DD documents, we decided to implement the following functions:

- **Sign Up**
- **ASAP: As Soon As Possible**
- **Hand out tickets on spot**

For more details regarding the specific functionalities, you are invited to read the RASD document, which contains a very detailed description of them.

We chooses to implement these functionalities in order to simulate a Module 1 product purchase. In fact, we remind that the application is divided into three modules; the first one is also called *MVP*, and contains the basic functionalities of CLup, the second one contains the *Book a Visit* feature and the last one includes the custom deploy on organization's servers.

In a real-world scenario, the most used module would be of course the first one (due to its simplicity), so that we decided to focus on it.

2.2 Adopted Development Frameworks

As we said in our DD, the application should follow a four-tier architecture, with a fully REST interface and a lot of client scripting. Finally, we decided to adopt the Model-View-Controller pattern, which is one of the most used in distributed applications.

In the following pages you will find a list of adopted frameworks and technologies in order to accomplish to this requirements.

2.2.1 Programming languages

For sake of standards and application speed we decided to use the Java™Programming Language, which is one of the most used languages in web and distributed applications.

Of course, there are some pros and cons about using this type of language:

- **Pros:**
  - + Speed: of course, since it is a compiled language, Java permits to have very good performances on these elaborations;
  - + Standard: as shown in figure 1, Java is the *De Facto standard* in enterprise web development and represents a very good solution for portable applications;
  - + Stability: Java is a mature language that has immensely evolved over the years. Hence it's more stable and predictable.
  - + Object-Oriented: The object-oriented nature of Java allows developers to create modular programs and write reusable codes. This saves lots of efforts and time, improving the productivity of the development process.
  - + Well-documented
- **Cons:**
  - – High verbosity: with respect to other programming languages (such as Python), Java contains a more verbose and less-readable syntax.
  - – High memory consumption: since Java Programs run on top of Java Virtual Machine, it consumes more memory.

For the client-side scripting, we decided to adopt JavaScript™for the web app, and Flutter with Dart™language for the mobile app.

JavaScript is a text-based programming language used both on the client-side and server-side that allows to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user.

Flutter, instead, is an open-source UI software development kit created by Google. It is mostly used to develop applications for Android and iOS.

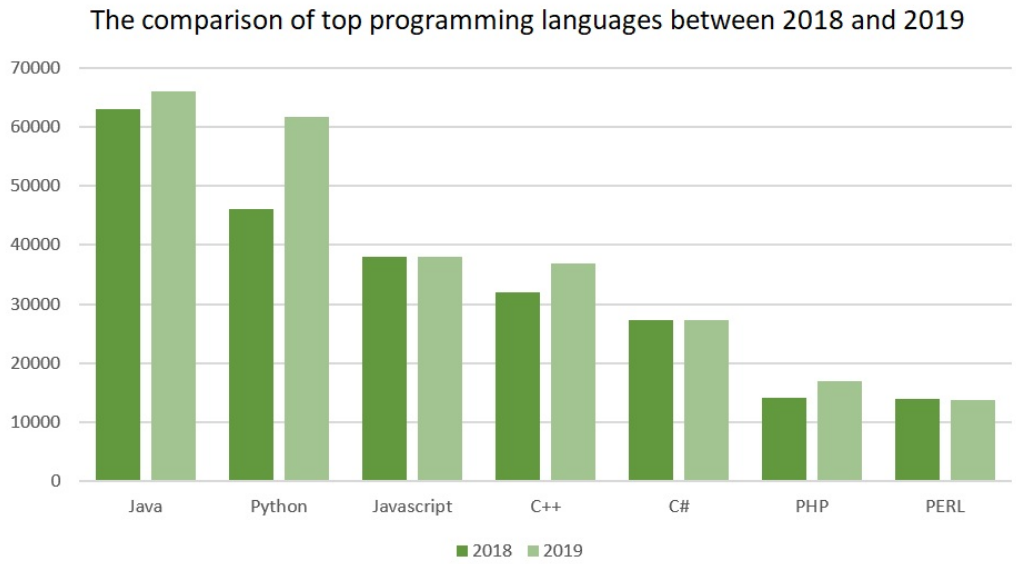

Figure 1: Web Programming languages usage (2018 vs 2019)

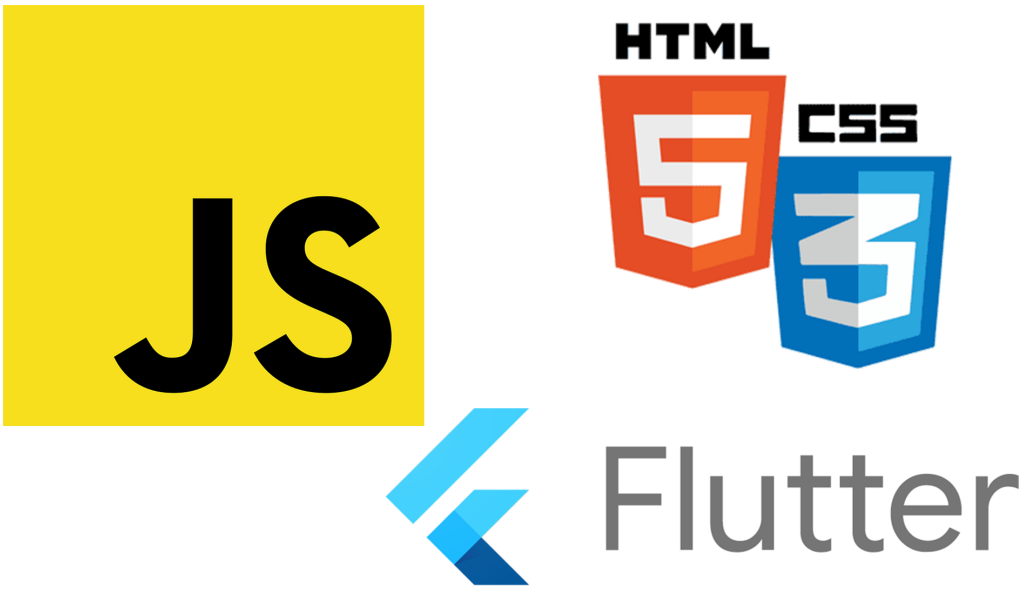

Figure 2: Client-side technologies

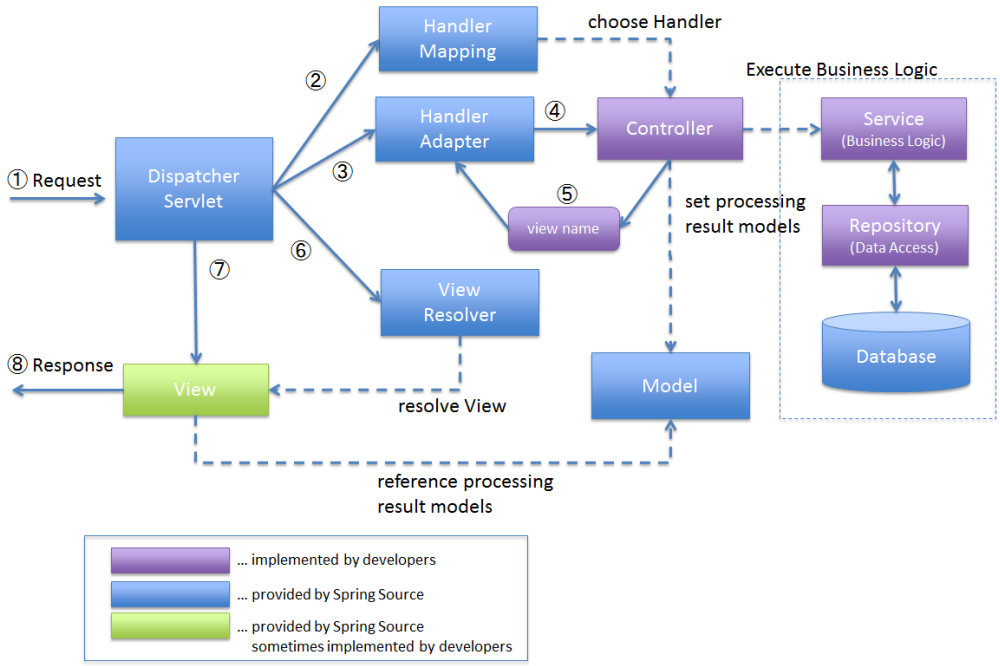

Figure 3: Spring Framework architecture

2.3 Java Frameworks

2.3.1 Spring

In order to accomplish the S2B requirements, we decided to adopt the Spring development framework.

Spring is an open source framework, used for RESTful Java application development. It's built on top of the Java Enterprise Edition (JEE) and represents an efficient and modern alternative to the classic Enterprise Java Bean (EJB) model.

Of course, using a framework means works on a solid base, which is well tested and documented. In fact, Spring contains a proper paradigm in order to build web services on its API.

As shown in figure 3, the architecture of Spring is very simple and easy-to-use.

In fact, it is composed by:

- **Dispatcher Servlet:** its job is to route all the incoming requests to the

correct Spring controller, which is properly mapped with an appropriate annotation. This is done through the severals components of Spring, which are HandlerMapping, HandlerAdapter and ViewResorver.

- **Controller:** the controller acts as an interface between the user and the services. It catches the requests coming from the dispatcher and makes actions relying on what user passed to it.

- **Model:** it contains the application logic about the transfer objects (also called DTO) which maps a user input (which is encoded in JSON format) and a Java object. The DTOs can also be used in the opposite direction (server to client), so they are encoded in JSON format and then sent to the client.

- **Services:** they act as an intermediate between the entities (database objects) and the controller, containing some useful methods in order to manipulate data sent by controllers.

- **Repositories:** they are interfaces that contains the query methods in order to fetch data from database. They are managed from Spring engine and it suffices to specify what to retrieve in order to get a response object from the DBMS.

- **Entities:** they represent database objects. They are declared with @Entity annotation, which maps the object to a database table (or set of them). Inside an entity object it is possible to specify constraints and foreign references through proper annotations.

2.3.2 Spring Data JPA

Spring Data JPA, part of the larger Spring Data family, permits to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed.

In fact, through services and repositories, interfacing with the database becomes very simple. Useless to say that, furthermore, the security of the queries from SQL injection is high, because it is totally managed by Spring.

2.3.3 Spring Security

Spring Security is a plugin of the Spring Mvc suite, which manages the application security.

In fact, it is role-based and permits, through several annotations, to grant access to controller only to authorized people.

In CLup, Spring Security is also the responsible for the REST authentication, which is made through a pattern called JWT (Json Web Token).

Practically, it is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. In our case, there is an exchange of a token which authorizes each client request, so the client needs to attach it to request's headers each time it calls the server.

2.3.4 Spring Social

Spring Social is another plugin of the Spring family which permits the user authentication through several identity providers (IdPs).

For sake of simplicity, we decided to start from the Facebook one, which is one of the most used social network nowadays. Once called, Facebook servers authenticate the user and return back to the CLup backend its information, which generally consists of name, surname and profile id.

Obviously, the same can be done with the other identity providers, such as Google, Twitter, GitHub, etc.

2.4 Other Frameworks

2.4.1 Vue.js

Vue is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

We used Vue.js to manage the client-side rendering of the pages on the web app. In fact, each page calls its scripts, which are responsible for the correct orchestration.

2.4.2 jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

We used jQuery to implement the network layer of the client-side web application. In fact, it is responsible (through the Ajax plugin) of the communication with the server, managing all the requests and response.

2.4.3 QRcode.js

QRCode.js is javascript library for making QRCode. QRCode.js supports Cross-browser with HTML5 Canvas and table tag in DOM. QRCode.js has no dependencies.

We used this library in order to generate a visual representation, through a Quick Response Code (also known as QRCode), of the ticket unique id, in order to easily permit its validation at the supermarket entrance.

2.4.4 Semantic UI

Semantic UI is a graphical framework which is useful to construct good-looking web interfaces with less effort than making them manually.

It includes several css styles, together with a full JavaScript set of functionalities (such as dynamic management of the page).

2.5 API Integrations

2.5.1 Leaflet.js

Leaflet is an open source JavaScript library used to build web mapping applications.

We used it to render the maps containing stores position, in the user and manager private area.

3 Source Code

3.1 Backend

The backend is packaged through the Maven framework, which gives also a standard for organizing the code. Of course, we followed this standard, merged with the Spring Framework one.

More details about the maven projects standard organization can be found at https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

3.1.1 Packages

As previously said, the application backend is built on top of the Spring Framework, which follows an architectural pattern called Model View Controller (MVC). The extension of the MVC provided by Spring defines also a standard structure of the project classes.

Package it.polimi.se2.clupapplication

- **Config:** this package contains the configuration classes of Spring framework, which manage security and general application configuration.
- **Controllers:** it contains all the controllers of CLup application, following the Spring controllers paradigm. See section 3 for more details.
- **Entities:** this package contains all the relational database objects, which are properly mapped to a SQL server through the JPA APIs. Each table has its own entity class, with the exception of the bridge (OneToOne, OneToMany, ManyToMany) relations.
- **Model:** following Spring standard, this package includes all the DTO (Data Transfer Objects) for data exchange between client and server.
- **Repositories:** it contains the Spring repositories, which are the interfaces required in order to query the DBMS and the related entities.
- **Security:** part of the Spring plugins, this package includes the security classes of CLup application. In fact, these classes manages the process of login and sign up through the exchange of the JWT (Json Web

Token). Furthermore, they also implement the IdP sign in feature with social networks (such as Facebook).

- **Services:** this last package contains the service classes, which are the intermediates between controllers and repositories, following the Spring approach described in section 3.

3.1.2 Flow of a request

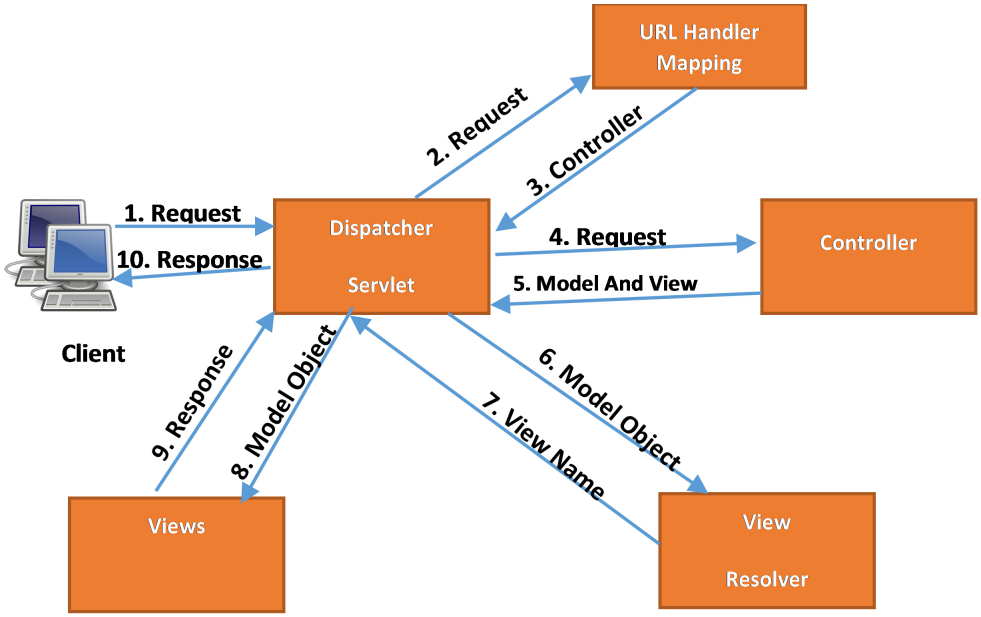

Figure 4: Flow of a client request to backend

As you can see in figure 4, a request coming from a client follows several step into the backend structure.

First of all, it is intercepted by the Dispatcher Servlet, which forwards the request to the URL Handler Mapping. This handler contains the mapped urls from all the application controller and also the method accepted (GET, POST, PUT, etc.).

Once URL Handler has resolved the associated controller, the request is dispatched to the correct controller, which calls the classes needed for elaboration (such as services and, consequently, repositories).

After the elaboration, the controller creates a ResponseEntity object and sends it to the view resolver which, with the help of Views class, return the response to Dispatcher Servlet.

Finally, the Dispatcher Servlet return the created response to the client.

3.1.3 Test cases

The test cases, which are described in details in section 4, are located under the maven *test* folder, into the package it.polimi.se2.clupapplication.

3.2 Frontend Structure

3.2.1 Web Application

The front-end web application is contained into the *resources/public* folder of the Spring application, in order to permit an easy reachability directly from the backend dispatcher without being filtered from the security manager. Of course, following the four tier architecture described in the Design Document, the front-end web application can also be deployed to a dedicated web server, which will then make requests to a different backend server.

The structure of the web app is the following:

- **admin:** this folder contains all the administration HTML web pages, which are served with the */admin* url prefix;
- **attendant:** this folder, instead, contains all the attendant dashboard HTML web pages, which are served with the */attendant* url prefix;
- **static:** this last folder contains all the static files of the web application, which consists of CSS stylesheets, JS scripts, fonts, images and client-side frameworks adopted (see section 2.4);
- all the other HTML files in *public* directory refer to the customer web interface, which will be of course the most used pages.

3.2.2 Mobile App

3.3 Other Useful Information

The application is configured to fetch data from a MySQL database, which credentials must be specified in the application.yaml properties file. This

configuration file can be found in *resources* folder and contains also all the other static configurations of the application, such as jpa configurations, social login credentials, jwt token settings, etc.

4 Testing

In this section we will briefly describe how we tested the application, following the general guidelines given in the Design Document.

We decided to test only the backend and mobile app deployable, because the front-end can be easily tested "by seeing it working".

4.1 Backend

We wrote the test cases using the JUnit 5 suite and, for mocking the user in the requests, the Spring Security Testing Suite.

Following the suggested Spring testing approach, we wrote the integration test cases for the controller classes, because of their primary role in request-response flow.

In fact, testing a controller means test also all the other components (services, repositories, entities, etc.), because they are sequentially called from it.

Up next you'll find a complete report on how we did the test cases, and also their results.

Integration Tests

- **UserController:** we tested the Sign Up and login methods, as defined in the DD. In the sign up tests, we tried with both a Customer and a Store Manager profiles. Finally, we tested the correct exchange of the Json Web Token between the parties.

- **StoreController:** we tested the store and slot creation functions. We defined a mock store manager, in order to correctly test that the permission based access would correctly handle the request.

- **TicketController:** we tested the procedure of ASAP (As Soon As Possible) ticket retrieving functionality. Furthermore, we defined a mock store attendant, in order to test also the **Hand Out on Spot** function.

The above test, which were in total 13, were run all together, obtaining a percentage of success of 100%.

Thanks to the obtained result it is therefore possible to state that the backend is sturdy and built on solid source code.

Unit Tests

Furthermore, we made some tests on the correct DBMS initialization, through a specific JPA Test construct. They all obtained a positive mark.

4.2 Mobile Application

5 Installation Instructions

6 Effort spent

Student	Time for Implementation Project
Luca Pirovano	60 hours
Nicolò Sonnino	60 hours