AY 2020/2021

Politecnico di Milano

Implementation Document

Luca Pirovano Nicolò Sonnino

Professor
Matteo Rossi

Version 1.1
February 7, 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 Acronyms

- **API:** Application Programming Interface.
- **APK:** Android Package
- **DBMS:** DataBase Management System.
- **DD:** Design Document.
- **DOM:** Document Object Model.
- **DTO:** Data Transfer Object, represents a link between the user input and a Java Object.
- **HTTP:** HyperText Transfer Protocol.
- **IPA:** iOS App Store Package.
- **JPA:** Java Persistence API.
- **JS:** JavaScript.
- **QR Code:** Quick Response Code.
- **REST:** REpresentational State Transfer (see DD).

- **RASD:** Requirements Analysis and Specification Document.
- **S2B:** Software To Be.
- **UI:** User Interface.
- **URL:** Uniform Resource Locator.

1.2.2 Abbreviations

- **CMD:** Command Prompt.

1.3 Revision history

- January 28, 2021: version 1.0 (first release)
- February 7, 2021: version 1.1 (fixed some typos)

1.4 References

- **Dart** https://dart.dev/
- **Easy Loading** https://pub.dev/packages/easy_loading
- **Flutter Barcode Scanner** https://pub.dev/packages/flutter_barcode_scanner
- **Flutter Local Notification** https://pub.dev/packages/flutter_local_notifications
- **Flutter Printing** https://pub.dev/packages/printing
- **Flutter** https://flutter.dev/
- **Font Awesome** https://fontawesome.com/
- **Google Fonts** https://fonts.google.com/
- **Google Maps Flutter** https://pub.dev/packages/google_maps
- **Hive** https://pub.dev/packages/hive
- **Java:** https://www.java.com/it
- **JavaScript:** https://www.javascript.com

- **jQuery:** https://jquery.com
- **Kotlin** https://kotlinlang.org/
- **Leaflet.js** https://leafletjs.com
- **Pull to refresh** https://pub.dev/packages/pull_to_refresh
- **QR Flutter** https://pub.dev/packages/qr_flutter
- **QRcode.js:** https://jquery.com
- **Semantic UI:** https://semantic-ui.com
- **Spring Framework:** https://spring.io
- **Spring Security:** https://spring.io/projects/spring-security
- **Spring Social:** https://projects.spring.io/spring-social
- **Swift** https://www.apple.com/it/swift/
- **Url Launcher** https://pub.dev/packages/url_launcher
- **Vue.js:** https://vuejs.org

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**
- **Periodic notifications**

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. The main functionality of this framework is the versatility of the Dart programming language, which lets compile application both in Swift (iOS) and Kotlin (Android).

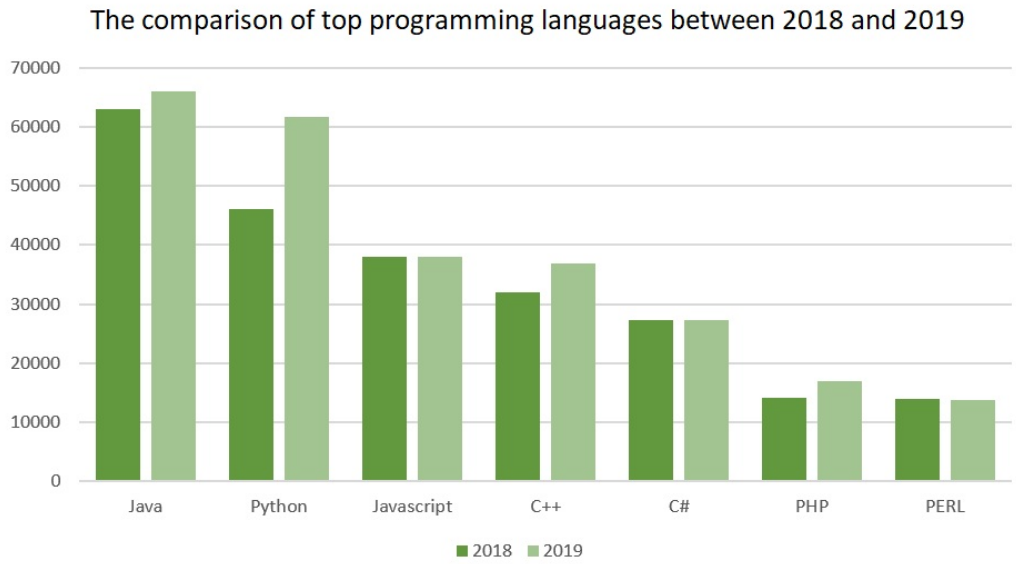

Figure 1: Web Programming languages usage (2018 vs 2019)

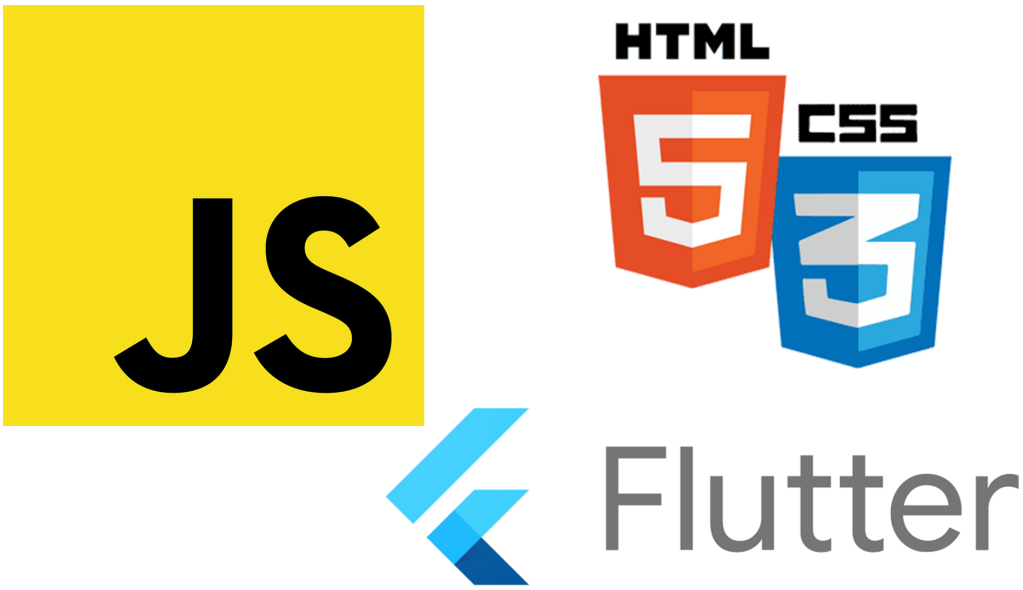

Figure 2: Client-side technologies

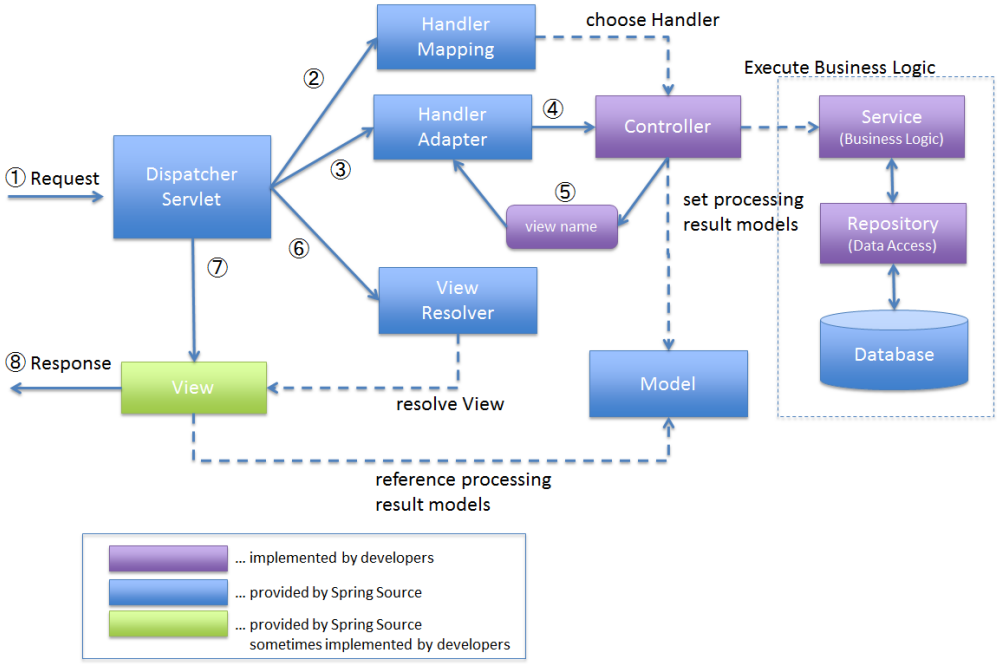

Figure 3: Spring Framework architecture

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 ViewResolver.

- **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.

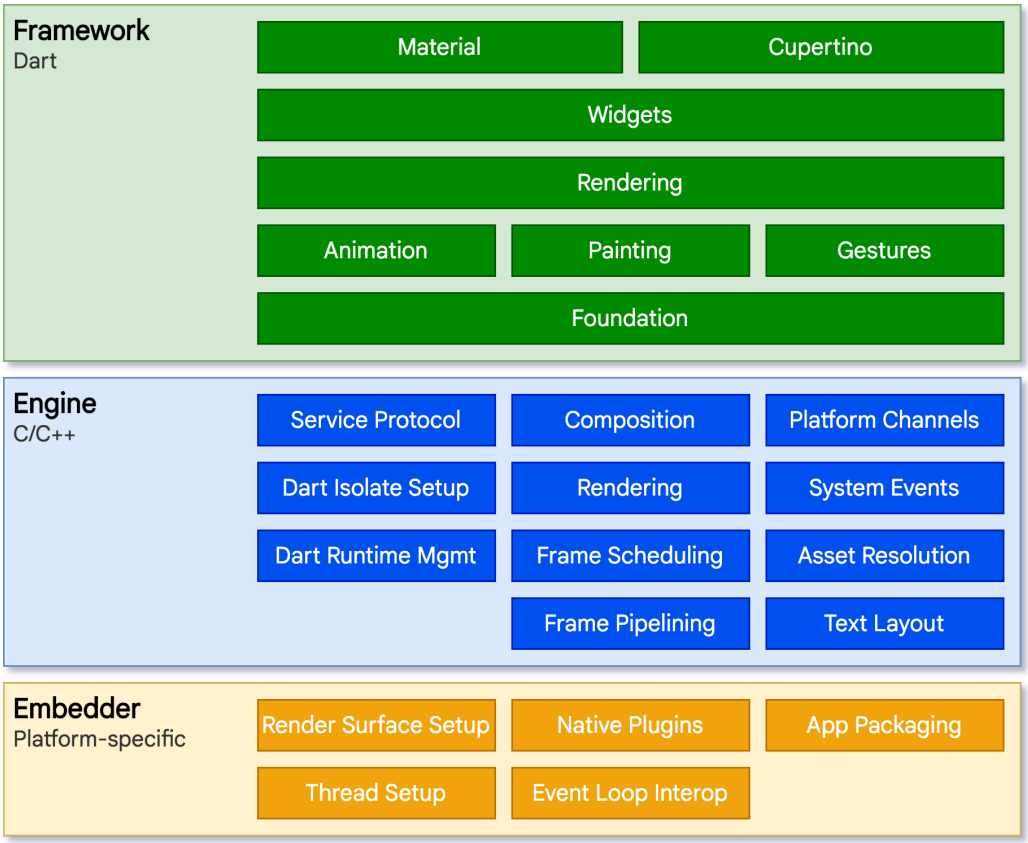

Figure 4: Flutter Framework architecture

2.6 Mobile Frameworks

2.6.1 Flutter

Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.
At the core of Flutter is the Flutter engine, which is mostly written in C++ and supports the primitives necessary to support all Flutter applications.
It includes a rich set of platform, layout, and foundational libraries, composed of a series of layers. Working from the bottom to the top, we have:

- Basic **foundational classes**, and building block services such as ani-

mation, painting, and gestures that offer commonly used abstractions over the underlying foundation.

- The **rendering layer** provides an abstraction for dealing with layout. With this layer, you can build a tree of renderable objects. You can manipulate these objects dynamically, with the tree automatically updating the layout to reflect your changes.
- The **widgets layer** is a composition abstraction. Each render object in the rendering layer has a corresponding class in the widgets layer. In addition, the widgets layer allows you to define combinations of classes that you can reuse. This is the layer at which the reactive programming model is introduced.
- The **Material** and **Cupertino** libraries offer comprehensive sets of controls that use the widget layer's composition primitives to implement the Material or iOS design languages.

2.6.2 Hive

Hive is a lightweight and blazing fast key-value database written in pure Dart.
It has the role of Model in the MVC pattern used in the Mobile App as some information are stored on the client in order to improve performance.
It uses HiveObjects as basic structure which are then implemented or extended through TypeAdapters.

2.6.3 QR Flutter

QR.Flutter is a Flutter library for simple and fast QR code rendering via a Widget or custom painter.
It implements QrImage and QrPainter widgets which are responsible for the correct realization of QR codes.

2.6.4 Flutter Barcode Scanner

Flutter Barcode Scanner is a plugin for Flutter apps that adds barcode scanning support on both Android and iOS.
It's especially useful because it interfaces with the mobile's cameras and implements QR code's scanning out of the box.

2.6.5 Google Maps Flutter

It's a Flutter plugin that provides a Google Maps widget and easy interface with Google API.

2.6.6 Path Provider

A Flutter plugin for finding commonly used locations on the filesystem. Supports iOS, Android, Linux and MacOS (vital for the correct model managing).

2.6.7 Printing & Http

The first plugin provides easy API for accessing native Android printing services, instead the second one lets access HTTP GET/POST requests which are a core functionality in our implementation.

The remaining frameworks help with minor functionalities and can be found in the References section[1.4].

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

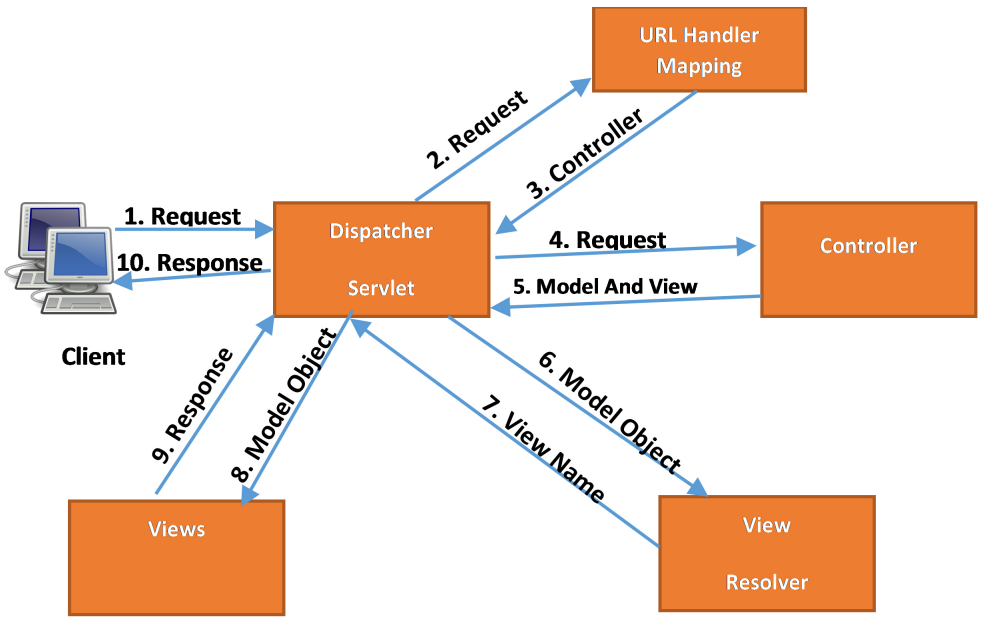

Figure 5: Flow of a client request to backend

As you can see in figure 7, 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, returns the response to Dispatcher Servlet.

Finally, the Dispatcher Servlet returns 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.

3.3 Mobile App

The mobile code implemented follow the MVC pattern with minor differences.
In addition to the architecture, Flutter creates different directories and files:

- **android**, containing native Kotlin code;
- **ios**, containing native Swift code;
- **lib**, the folder where your app is placed (Dart);
- **pubspec.yaml**, file specifying your app's version, plugin imported and assets used.

3.3.1 Package it.polimi.clup

In the lib folder, our mobile app is organized with the MVC pattern as follows:

- **model**: it contains all HiveObject used to store information on the client.
- **pages**: package containing all mobile app's screens, divided in **StatefulWidgets** or **StatelessWidget**. It represents the View in the MVC pattern.
- **widgets**: analogous to the above package, it contains individual widgets (used in pages) in order to increase readability and reuse components multiple times in the mobile app (FormFields, AppBars etc.).
- **utils**: this package contains the controller component specified in the MVC pattern; it contains the following components which sends REST API to the Application Server: **AuthService** (authentication), **Generator** (booking/ticket manager) and minor utility classes.
- **theme**: folder containing files for theming.

3.3.2 Platform channels

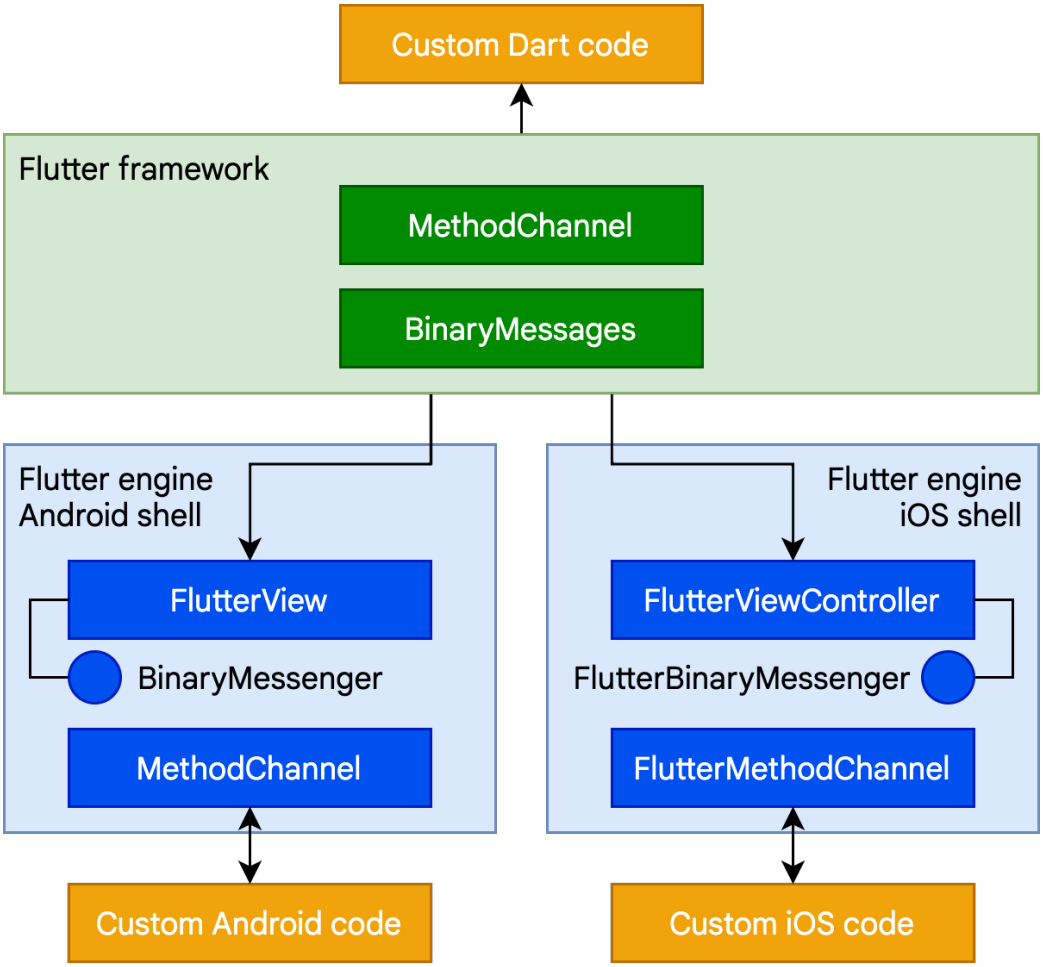

Figure 6: Platform channels

For mobile and desktop apps, Flutter allows you to call into custom code through a platform channel, which is a simple mechanism for communicating between your Dart code and the platform-specific code of your host app.
By creating a common channel (encapsulating a name and a codec), you can send and receive messages between Dart and a platform component written in a language like Kotlin or Swift.
Data is serialized from a Dart type like Map into a standard format, and then

deserialized into an equivalent representation in Kotlin (such as HashMap) or Swift (such as Dictionary).

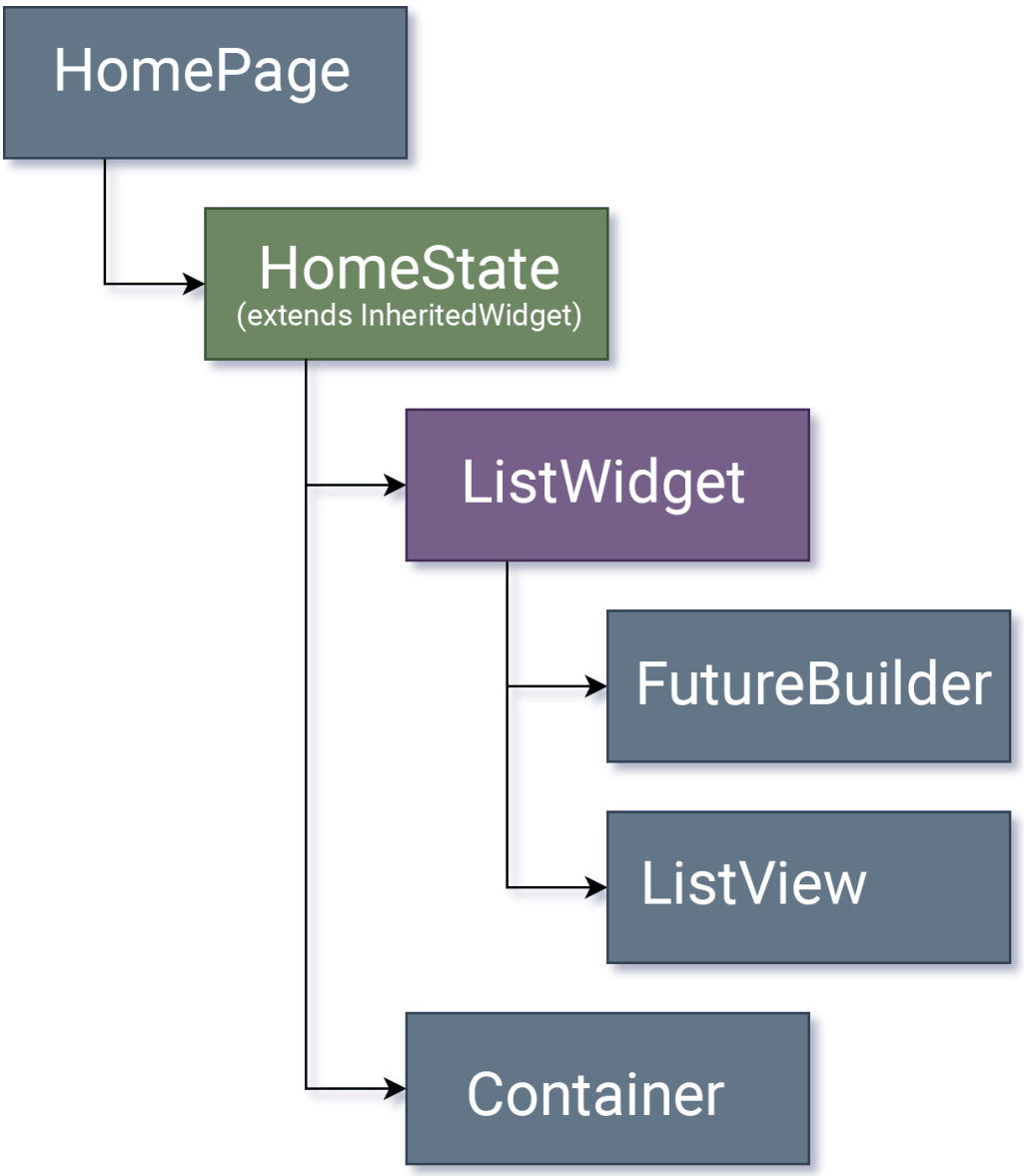

Figure 7: Platform channels

The framework introduces two major classes of widget: *stateful* and *state-*

less widgets.
Many widgets have no mutable state: they don't have any properties that change over time (for example, an icon or a label). These widgets subclass **StatelessWidget**.
However, if the unique characteristics of a widget need to change based on user interaction or other factors, that widget is *stateful*. For example, if a widget has a counter that increments whenever the user taps a button, then the value of the counter is the state for that widget. When that value changes, the widget needs to be rebuilt to update its part of the UI. These widgets subclass **StatefulWidget**, and (because the widget itself is immutable) they store mutable state in a separate class that subclasses **State. StatefulWidgets** don't have a build method; instead, their user interface is built through their **State** object.
Whenever you mutate a State object (for example, by incrementing the counter), you must call **setState()** to signal the framework to update the user interface by calling the State's build method again.
Having separate state and widget objects lets other widgets treat both stateless and stateful widgets in exactly the same way, without being concerned about losing state. Instead of needing to hold on to a child to preserve its state, the parent can create a new instance of the child at any time without losing the child's persistent state. The framework does all the work of finding and reusing existing state objects when appropriate.

3.4 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.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 20, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.507 s
[INFO] Finished at: 2021-02-01T00:21:43+01:00
[INFO] ------------------------------------------------------------------------
```

Figure 8: Maven test report

4.2 Mobile Application

Integration testing is possible in Flutter, but Widgets have very strict assertions on input data; another very useful functionality of this framework is *hot reload* and *hot restart* which let runtime testing while coding it and without rebuilding it.
We decided that integration on widgets was very time consuming and not worth it compared to the effort spent.
Instead we tested our application on daily basis but more importantly we deployed our application *APK* and *IPA* on **Google Play Store** and **TestFlight** (Apple's App Store for beta builds).
Once the builds were approved by Google/Apple we invited about 20 testers in order to stress the app and receive constant feedbacks, using them as the baseline for each new update.

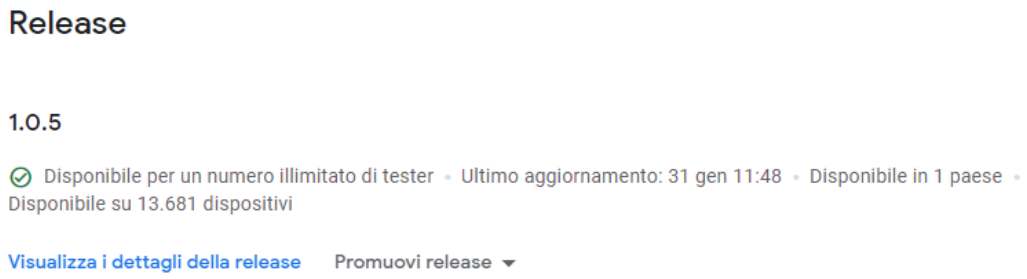

Figure 9: Google Play build report

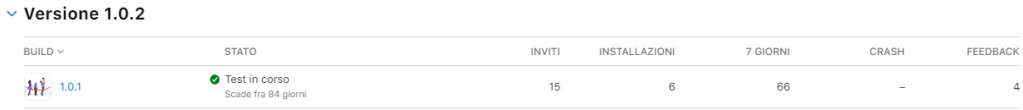

Figure 10: Apple's TestFlight report

5 Build

5.0.1 Requirements

- Java SE JDK 15 (**OracleJDK**, **OpenJDK**)
- **Maven** framework version 3.0 (or newer)

5.1 Windows

If you use **Windows Subsystem for Linux (WSL)** or **MinGW** you can skip to Linux installation steps.

5.1.1 Installing Java

Download **OracleJDK** or **OpenJDK**.

Extract content from the zip folder to your preferred location, then go to `Start>Edit the system environment variables>Environment Variables ...`

In the User Section select `Path` variable and click on `Edit`; select `New` and type `C:\Users\<your-user>\<path-to-extracted-folder>\bin`, then save and exit.

Open `Start>cmd.exe` and verify your Java version by typing the following command:

```
java -version
```

5.1.2 Installing Maven

Download **Maven**.

Extract to preferred location and repeat above steps.

Verify your Maven version by typing the following command:

```
mvn --version
```

5.2 Linux

If you want to download Maven and JavaJDK with your package manager be sure to fulfill the system requirements otherwise follow the next steps.

5.2.1 Installing Java

Download JavaJDK based on your distro or OpenJDK 15 here.

Linux Platforms

Navigate to your preferred install location and extract the .tar.gz archive file using:

```
tar zxvf jdk-15.<version-number>-x64_bin.tar.gz
```

Now let's set the `PATH` variable by typing the following code:

```
cd $HOME
nano .bashrc
```

Add the following line to the end of `.bashrc`:

```
export PATH=/<path-to-extracted-folder/bin:$PATH
```

Verify your Java version by typing the following command:

```
java -version
```

Debian-Based Linux Platforms
Type:
`sudo apt install /path/to/package/name.deb`
Verify your Java version by typing the following command:

```
java -version
```

RPM-Based Linux Platforms
Type the following command to install the package:

```
rpm -ivh jdk-15.<version-number>-x64_bin.rpm
```

Upgrade the package using the following command:

```
rpm -Uvh jdk-15.<version-number>-x64_bin.rpm
```

You can now delete the `.rpm` file and verify your installation by typing:

```
java -version
```

5.2.2 Installing Maven

Download **Maven**, extract it with:

```
tar zxvf maven.<version-number>-x64_bin.tar.gz
```

Set `PATH` variable by appending this command to `.bashrc` as previous steps:

```
cd $HOME
nano .bashrc
export PATH=/<path-to-extracted-folder/bin:$PATH
```

5.3 MacOS

5.3.1 Installing Java

Download JavaJDK 15, double-click on `.dmg` file and **Install** it.
Verify your Java version by typing the following command:

```
java -version
```

5.3.2 Installing Maven

Follow Linux installation steps.

5.4 Compiling the source files

1. Open the terminal and navigate to the root application folder (containing pom.xml).
2. Type `mvn package -Dmaven.test.skip=true` and hit `Enter`.

Why am I skipping maven test phase?

Because since there is no data source connected, it cannot perform tests on it.
If you want to include the test phase, you will need to edit the *application.yaml* file located under `src/main/resources` as specified in the Installation section.

6 Installation

Follow compiling installation steps without **Maven** section.

6.0.1 Requirements

- Java SE JDK 15 (**OracleJDK**, **OpenJDK**)

6.1 Backend

6.1.1 Windows self contained installer

1. Download the executable file from **Release page** on our Github repo.
2. Install it following the wizard installation tool.
3. Navigate to installation directory (usually `C:\Program Files\clupServer`).
4. Edit the *application.yaml* file under **app** folder:

Spring.datasource

- **url**: replace <url> with your database domain address and <db_name> with the database's name.
- **username**: your database's administrator username.
- **password**: your database's administrator password.

Spring.mail

- **host**: your SMTP server hostname.
- **username**: your SMTP sender username.
- **password**: your SMTP sender password.

Server.port

- **port**: application server port.

5. Save and exit.
6. Launch clupServer shortcut from your Desktop.

6.1.2 Cross-platform JAR Package

1. Download the zip archive from **Release page** on our Github repo.
2. Extract the two files in the same folder.
3. Edit the *application.yaml* as stated in the Windows self contained installer section.
4. Launch it from your terminal with the following command:
   `java -jar clup<version>.jar -spring.config.location=application.yaml`

6.2 Mobile Application

6.2.1 Store Builds

Download the application based on your platform following the provided links:

6.2.2 Github Builds (Android only)

Download the application by accessing the Release page on our Github repo or by clicking the link **here**.

7 Effort spent

Student	Time for Implementation Project
Luca Pirovano	70 hours
Nicolò Sonnino	70 hours