Beertastic

Design and Implementation of Mobile Applications

Design Document

Version: 1.0

Andrea Costanzo

September 5, 2020

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of the present Design Document is to provide a complete description of the design of Beertastic's system, within which the main component is the Android application developed for the course of Design and Implementation of Mobile Applications at Politecnico di Milano (Italy). The goal of the course was to design a "mobile" application in which the user experience assumes a central role, starting from how the various elements characterizing the UI should be disposed to how users should interact with them. Beertastic was thought to provide users with a smooth and joyful way to approach to the beer and brewery worlds, by offering the possibility to read articles, see events, search beers and exchange opinions. This document presents a functional specification of the architectural components defining the system as well as their interfaces and interactions, and make use of graphs to expose better their relationships and behaviours.

1.1.1 Goals

The goals of *Beertastic* are the following:

[G1] The application should provide an immediate way to save the credentials of the user, with the possibility to register, sign-in and sign-out.

[G2] Users should be able to read articles.

[G3] Users should be able to check information about the events available in their cities.

[G4] Users should be able to search beers by scanning barcodes and QR codes, or by typing the name of the beer.

[**G5**] Users should be able to review beers.

[**G6**] Users should be able to read the reviews from experts for each beer.

[**G7**] Users should be able to read others' reviews and filter them according to the grade.

[**G8**] Users should be able to save their favourites beers to check them later.

[**G9**] Users should be able to modify their profile information, like nickname, city and profile image.

[**G10**] The application should allow users to delete their accounts.

1.2 Scope

Brewing and craft beers are broad domains, and it isn't easy to approach and explore them without guidance. To better fulfil this need, we developed an Android application aimed to provide users with the possibility to enter in the world of beer and breweries, by offering them an immersive and intuitive experience. Users can read articles written by experts of the field covering different topics, like home brewing and curiosities, look for events available in their cities and obtain more information on what they are drinking. When they move to a different city, they can modify their information to see events belonging to that city. Furthermore, users can review and see what other users said about them, with the possibility to filter results according to the grade.

The software system is divided into two layers, which will be presented further in the document. The architecture must be designed to be maintainable and extensible, while the individual components must have high cohesion and decoupling. For this purpose, the various components must not incorporate several unrelated functionalities and should decrease interdependency between them. Furthermore, design patterns and architectural styles will be used for the same reason.

1.3 Definitions, acronyms, and abbreviations

1.3.1 Definitions

- **Back-end application:** computer program (such as server software) that remains in the background and resides on a server and offers application logic, data access layer and communication interfaces to works with front-end applications

- **Framework:** a semi-finished software architecture for an application domain that can be adapted to the needs and requirements of a concrete application in the domain
- **Front-end application:** user interface or that part of a software or a website that a user sees on the screen, and acts on to enter commands or to access other parts of the software or website
- **Object-relational mapping:** a programming technique for converting data between incompatible type systems using object-oriented programming languages
- **RESTful:** Web services that conform to the REST architectural style
- **Representational state transfer:** an architectural style for developing web services
- **Sign in:** the process in which a user insert his/her to-be-defined data to authenticate into the system
- **Unified Modeling Language:** a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system
- **User:** any person subscribed and logged into the service

1.3.2 Acronyms

- **ACID:** Atomicity, Consistency, Isolation and Durability
- **API:** Application Programming Interface
- **BLoC:** Business Logic Component
- **CI:** Continuous Integration
- **COTS:** Commercial Off-The-Shelf
- **DD:** Design Document
- **HTML:** HyperText Markup Language
- **HTTP:** HyperText Transfer Protocol
- **HTTPS:** HTTP over TLS

- **ID:** IDentification
- **MVC:** Model View Controller
- **MVVM:** Model View ViewModel
- **ppi:** Pixels Per Inch
- **REST:** REpresentational State Transfer
- **SaaS:** Software as a Service
- **TLS:** Transport Layer Security
- **UML:** Unified Modeling Language
- **VCS:** Version control system
- **RASD:** Requirements Analysis and Specification Document

1.3.3 Abbreviations

- **[Gn]:** the n-th goal
- **[Rn]:** the n-th requirement
- **[S.n]:** the n-th software or tool used to realize the document

1.4 Document structure

This DD is structured in 5 chapters:

Chapter 1: Introduction. The first section provides an overall description of the system scope and goals, along with some information about the purpose of the document. In this part are also outlined definitions, acronyms and abbreviation useful to understand the other sections of the paper better.

Chapter 2: Overall description. It presents an overall description of the system's features and constraints, that generally are developer further in a RASD. Furthermore, in this section are outlined the assumption related to users and surrounding behaviours that we considered as valid in the next chapters.

Chapter 3: Architectural design. The third chapter presents a general description of the system components, both at a physical and at a logical level, describing their relationships, along with their static and dynamic design. There are also two sub-sections dedicated to design choices, styles and patterns used for the system to-be-developed.

Chapter 4: User interface design. This section shows how the user interface will look like and behave.

Chapter 5: Implementation, integration and testing. This section provides the guidelines to accomplish the implementation and the test phases, describing how to handle the whole process and outlining the techniques used.

At the end of the document there are an **Appendix** and a **Bibliography** containing references to software, tools and documentation used to write this document as well as additional information on the document itself.

Chapter 2

Overall description

2.1 Product perspective

Beertastic is a system that provides all the functionalities described in the product functions section. It includes all the subsystems needed to fulfil these software requirements. Users can access the service through the application, which should be downloadable and runnable on Android devices. All interfaces must conform to a uniform, intuitive and user-friendly design, that doesn't require the reading of detailed documentation to be used. The mobile application can run on devices that comply with the minimum requirements in terms of memory, computational power and other relevant parameters specified in the subsection 2.5.2. Furthermore, the back-end services are all hosted by Google's firebase and run when triggered by requests from the mobile application.

2.2 Assumptions and dependencies

Domain assumptions

[**D1**] Users can be identified by their email, which is unique for every being

[**D2**] Sign-up information given by users are true

[**D3**] Users complied all terms of agreement

[**D4**] Users' devices can provide precise and correct data on location.

2.3 Functional requirements

In this section are outlined the various the functional requirements of the application dived by goals. If two goals share a requirement, it is cited twice and referenced with the same number. Furthermore, domain assumptions are referenced if pertinent to the context.

[G1] The application should provide an immediate way to save the credentialsof the user, with the possibility to register, sign-in and sign-out.

- [D1] Users can be identified by their email, which is unique for every being.
- [D2] Sign-up information given by users are true.
- [D3] Users complied all terms of agreement.
- [R1] The system has to provide a form to users to register.
- [R2] The system has to provide a form to users to log in.
- [R3] The system must notify a registering user if the inserted email is already assigned to a registered account.
- [R4] The system must notify a signing in user if the inserted password is wrong.
- [R5] The system must notify the user to insert a valid email if the pattern inserted isn't correct.
- [R6] The system must allow users to reset their password using their email.
- [R7] The system must securely store cookies and tokens used for authentication.
- [R8] The system must provide a button to allow the user to log out.

[G2] Users should be able to read articles.

- [R9] The system has to provide a section dedicated to articles.
- [R10] The system has to provide a page for each article.

[G3] Users should be able to check information about the events available in their cities.

- [D2] Sign-up information given by users are true.
- [D4] Users' devices can provide precise and correct data on location.
- [R11] The system has to provide a section dedicated to events.

[R12] The system has to provide a page for each event.

[R13] The system must set as default city the one nearest to the user.

[R14] The system must allow users to modify their city.

[R15] The system must allow users to visits official Instagram event page.

[R16] The system must allow users to visits official Facebook event page.

[G4] Users should be able to search beers by scanning barcodes and QR codes,or by typing the name of the beer.

[R17] The system has to provide a search bar to search beers.

[R18] The system must provide a button to open the camera.

[R19] The system must ask users permissions to use the camera.

[R20] The system must allow users to search a beer by scanning a Qr code.

[R21] The system must allow users to search a beer by scanning a EAN8 or EAN13 barcode.

[R22] The system must allow users to see information about the beer.

[G5] Users should be able to review beers.

[R23] The system has to provide a form to review beers.

[R24] The system must provide a rating bar to rate the beer.

[R25] The system must allow users to delete their reviews.

[G6] Users should be able to read the reviews from experts for each beer.

[R26] The system has to provide a button from the beer page to open the review from the expert.

[R27] The system has to provide a page dedicated to the review from the expert.

[G7] Users should be able to read others' reviews and filter them according to the grade.

[R28] The system has to provide a button from the beer page to open the reviews.

[R29] The system has to provide a page dedicated to the reviews.

[R30] The system must show the reviews related to each beer.

[R31] The system has to provide a selector to filter review by grade.

[G8] Users should be able to save their favourites beers to check them later.

[R32] The system has to provide a button from the beer page to save it.

[R33] The system has to provide a page dedicated to favourite beers.

[R34] The system has to provide a button from to remove the beer from favourites.

[G9] Users should be able to modify their profile information, like nickname, city and profile image.

[R14] The system must allow users to modify their city.

[R35] The system must retrieve the nearest cities to the user location.

[R36] The system has to provide a page for profile information.

[R37] The system has to provide a button or field to modify information.

[G10] The application should allow users to delete their accounts.

[R38] The system has to provide a button to delete the account.

[R39] The system must alert the user that the operation is irreversible.

[R40] The system must verify the identity of the user if credentials were inserted before a defined period.

[R41] The system must have deleted all users data at the end of the process.

2.4 User characteristics

We consider users every person who downloaded the application on his/her android device. We differentiate between registered and unregistered users. The first has also created an account, while the latter has only downloaded the application

2.5 Constraints

2.5.1 Regulatory policies

It's user responsibility to grant that the use of the system complies with the local laws and policies. Users must provide current, complete and accurate account

information when they register. They must also agree to keep their log-in information confidential, do not authorize any third party to use their account and that they are responsible for all the activities that occur under their account. It's also necessary to authorize the system to collect, store and process personal information, authentication tokens and web cookies.

Conditioned on compliance, Beertastic grants to the user a personal, non-exclusive and nontransferable license to download, install and use the mobile or to access the system through the web app. Furthermore the system is capable to delete personal data upon user request.

2.5.2 Hardware constraints

These are the minimum hardware requirements needed to access the system from your personal computer or mobile phone:

- Mobile application
  - 3G UMTS connection at 2 Mb/s
  - 173 MB of available space on disk
  - 1 GB of RAM
  - 440ppi (screen resolution)

2.5.3 Availability

The service offered by *Lesson* has to be available 24/7 for the 99% of the time. The remaining 1% takes into account the time spent on ordinary maintenance sessions.

2.5.4 Reliability

The system reliability, which is the probability of operating without failure for a specific period, must be 99%.

2.5.5 Security

To protect sensitive information, different security measures have been considered while designing the system. All personal information stored in *Beertastic*'s databases has to be securely stored, using a proper hashing and salting mechanism, to avoid unauthorized access at any chance. All the connections, between users and servers and between servers and docker instances, should be encrypted using at least TLS v1.2.

2.5.6 Parallel operations

The system must support concurrent operations from different users.

2.5.7 Criticality of the application

The system is not employed in life-critical applications.

Chapter 3

Architectural design

3.1 Overview

This chapter presents a general description of the system components, both at a physical and at a logical level. The high-level design will be fully covered and detailed in section 3.2, while section 3.3 will focus on the deployment of the system on physical tiers and section 3.4 will describe the dynamic behaviour of the software. The selected architectural styles and patterns used, along with other design decisions, will be respectively presented and discussed in section 3.5 and 3.6. From a perspective of high-level of abstraction, the components of the system are:

Application layer: it includes most of the functional business logic which drives the system's core capabilities. It comprehends all the firebase modules discussed later in this chapter and allow to access the database and the storage.

Mobile application: it serves as the presentation layer of the system, communicates directly with the application layer and contains mainly presentation logic.

Database: it is a NoSQL DB, which includes all structures and entities responsible for data storage and management. No application logic is found at this tier, aside from the one belonging to the DBMS, which must ensure the proper functioning of the data storage and retrieval while ensuring the ACID properties of transactional databases.

Storage: it is a storage space used to store and save collections of large files.

3.2 Component view

In this section, a more detailed view of the various components of the system is presented with a top-down approach. The various components will be described in terms of needed sub-parts, their function and how they interface among them.

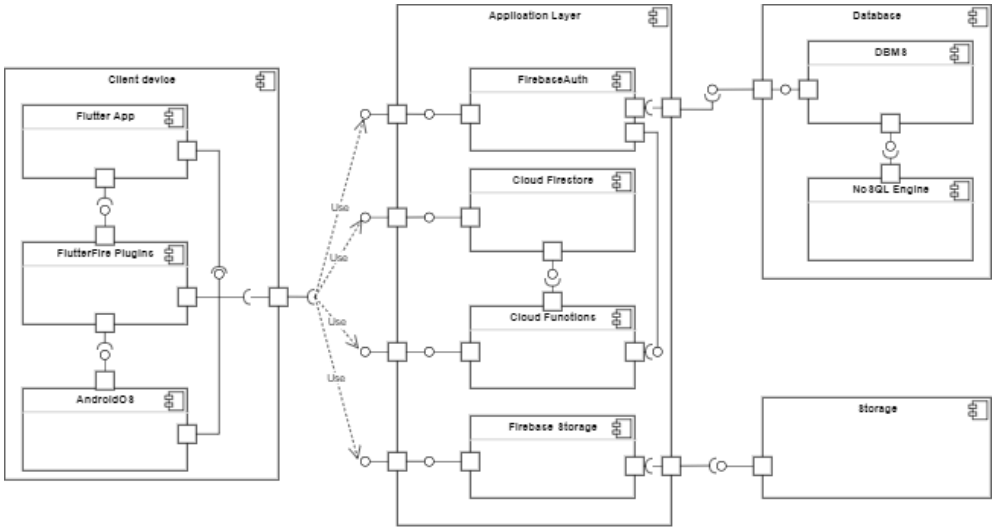


Figure 3.1: Component diagram

3.2.1 Application Server

The application server must handle the business logic, the connections with the data layer, and also have to manage the different ways of accessing the services from different clients. To comply with all these requirements, it must provide interfaces to connect with all the previously mentioned components. To speed up the development process of the mobile application, we decided to use COTS components supplied by the Firebase platform. In particular we used the following modules:

- **FirebaseAuth:** this module let all kinds of users authenticate into the system
- **Cloud Firestore:** this module allows accessing data available in the database

- **Cloud Functions:** this module allows running server-side scripts and perform, if necessary, data processing that would be too heavy for a client-side application.
- **Firebase Storage:** this module allows accessing data available in the storage

These modules use interfaces to communicate between them and with other components of the system. To have a better vision of how they interact, see the section 3.4.

3.2.2 Mobile application

The implementation of the mobile application must be autonomous from the structure of the application server. The UI must respect the design guidelines provided by Google's Material Design. To have the possibility in future to implement an iOS version of the service, without renouncing to the benefits of native execution in Android, we chose to realize the majority of components of the front-end software using Flutter 1.17.5[2], which is a framework based on the Dart[1] programming language. The only features implemented using Android[3] native code (based on Java or Kotlin) are the ones that need to access the hardware. Since Flutter is a very new framework, it still has some issue in delivering all the features to use some hardware components efficiently. The architectural pattern selected to develop the Flutter components is BLoC, while we used the MVVM pattern in Android code.

To develop the application, we also employed external libraries:

- **firebase plugins:** used to access the various services, provided by firebase, used in the application. We included firebase core 0.5.0, firebase auth 0.18.0+1, cloud firestore 0.14.0+2, firebase storage 4.0.0 and cloud functions 0.6.0.
- **google maps flutter 0.5.30:** used to implement a google map widget in the event page.
- **geolocator 5.3.2+2:** used to retrieve the location when searching for near cities.
- **permission handler 5.0.1+1:** used to request permissions to the user when needed.
- **flutter rating bar 3.0.1+1:** used to implement a rating bar widget for reviews.

- **flare flutter 2.0.6:** to add custom animated components to the app.
- **async 2.4.1:** used to implement asynchronous operations in BLoCs.
- **provider 4.3.2:** used to pass down object in the widget tree without having to complex systems and redundancy.
- **auto size text 2.1.0:** used to automatically resize text by providing a minimum and a maximum font size.

Furthermore, in the android part of the application, we employed firebase-ml-vision:24.1.0 library to handle the scan of the barcode and the CameraX API to obtain the preview of the image and the data necessary to the ML module to perform the analysis.

3.2.3 Database and Storage

Firebase entirely hosts the database and the storage. As we already mentioned, the data layer comprises a DBMS and a NoSql Database Engine to manage insertions, modifications, deletions of data inside the storage memory. The advantage of the NoSql engine it indexes queries by default: this means that the performance is proportional to the size of the result set and not of data set. Despite the implementation, for which we choose to rely on the service hosted by Firebase, this layer must respect the ACID proprieties while executing transactions. The database can interact only with the business logic layer (also called application layer) using the standard network interface with the Application Server. Before releasing the service, the team must ensure to implement efficient security measures to avoid unauthorized access to the information stored in the database. First of all, the database must be physically protected while communication has to be encrypted. Personal information needs to be encrypted too, and access must be granted only upon provision of valid credentials and when users reach the minimum level of privilege required to perform the operations. On the other side, the storage is a stand-alone solution for uploading and storing user-generated content like images and videos from an iOS and Android device, as well as the Web.

3.3 Deployment view

The deployment diagram for the system is presented in figure 3.2. Potentially, the logic related to the various modules of the application server can be executed in the same machine but, in most cases, they are allocated on different hosts. The mobile application will be available only for android devices.

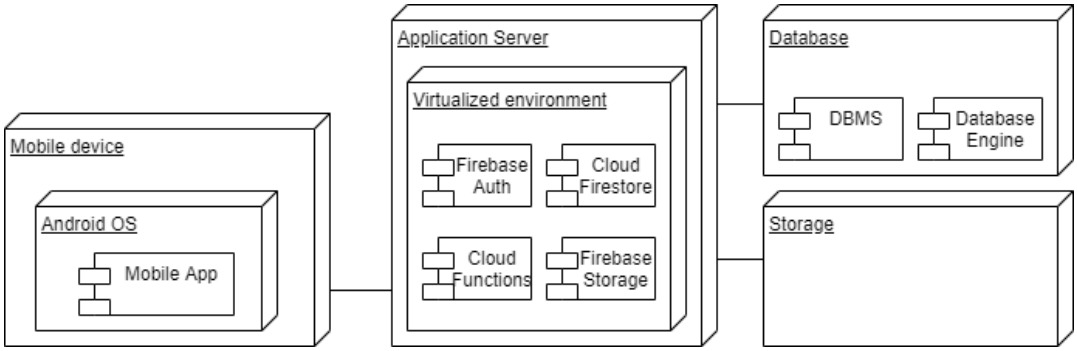


Figure 3.2: A high-level abstraction of the deployment of components

3.4 Runtime view

The purpose of this section is to describe the runtime behaviour of the most meaningful functionalities of the system by highlighting the interactions between different components.

These interactions are described, howsoever, maintaining a certain level of abstraction because we don't know precisely how the application server will instantiate its processes. Furthermore, we omitted the database representation, because the application layer provided by Firebase doesn't show how it interacts with the DBMS, and the application layer, since the plugins provided by Firebase handles all the processes.

Sign up This sequence diagram displays the process of signing up for the first time to the service. At the first opening of the application, the FirebaseAuth plugin notifies the UI that there's not a user logged in the service. The UI renders the authentication page. At this point, the user can select the register form, insert the credentials (email and password) he (or she) want to use, and press the button "sign-in" to start the registration process. If there's an error during registration the user receives a notification, otherwise the UI renders the home page of the service.

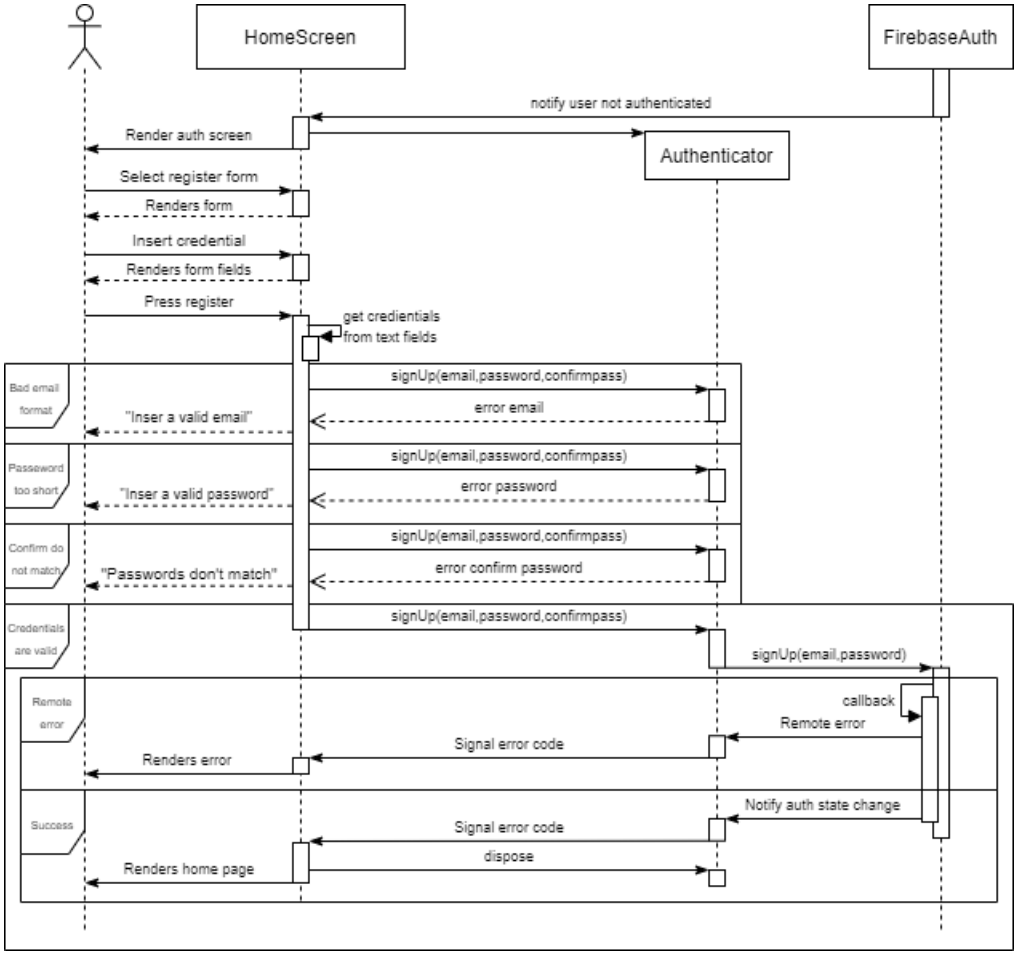


Figure 3.3: Sequence diagram about signing up

Sign in This sequence diagram displays the process of logging in after having registered to the service. When the user re-open the application, the FirebaseAuth plugin checks if there is an authentication token saved locally and, if it founds one, it immediately sends a signal to the UI that there's an authenticated user, which will display the home screen. Otherwise, it notifies the UI that there's not a user logged in the service. In this case, the UI renders the authentication page. At this point, the user can select the login form, insert the credentials (email and password) he (or she) want to use, and press the button "log in" to start the authentication process. If there's an error during authentication the user receives a notification, otherwise the UI renders the home page of the service. The firebase plugin manages both the authentication and the refresh tokens internally. In particular, the refresh token, which is used to obtain new access tokens, is deleted

only upon logout request from the client.

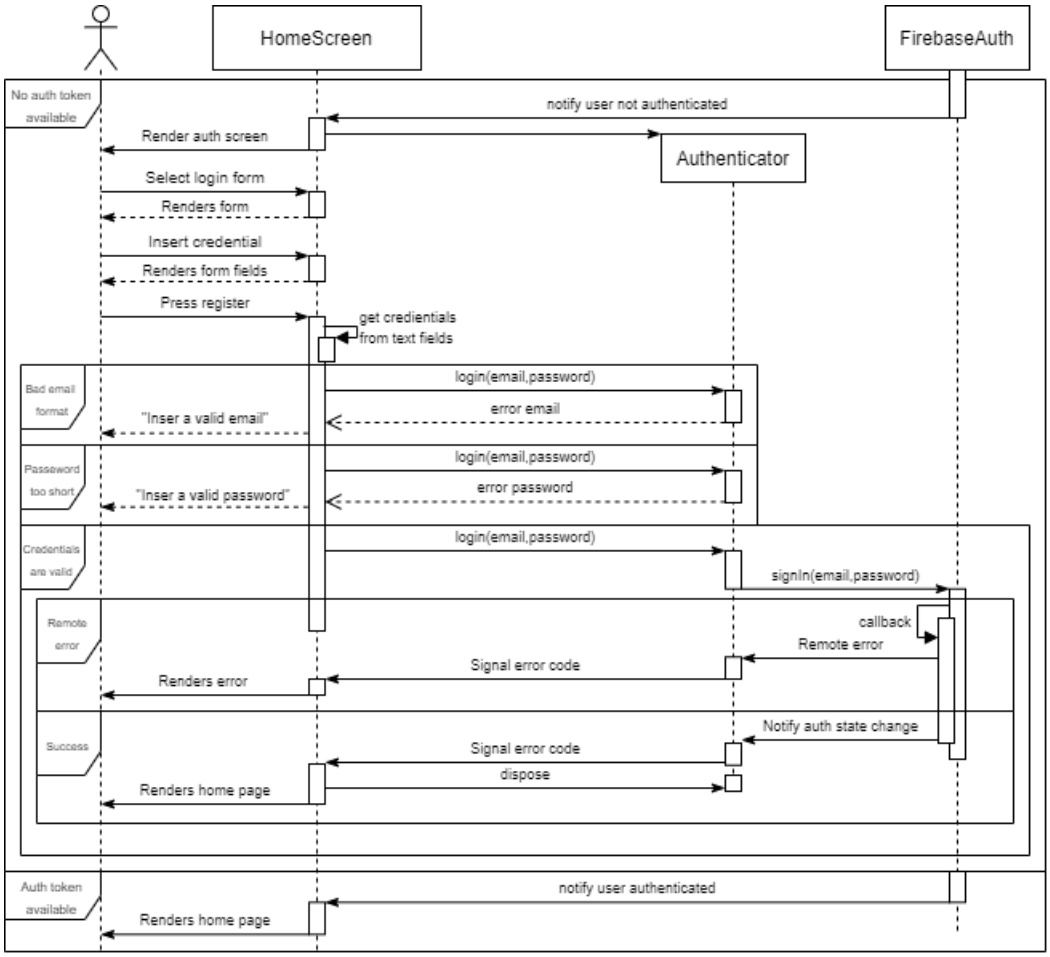


Figure 3.4: Sequence diagram about signing in

Scan a barcode This sequence displays how a user can scan a barcode: after pressing the scan button, the app uses a method channel to run native android code. The activity, if the user grants the permission to access the camera, start showing a preview to the user and analyzes each frame in the background. After finding a barcode or a QR code, the activity ends and returns its value as a result. If the user doesn't grant the permission to access the camera, the app notifies that he (or she) can't use the feature.

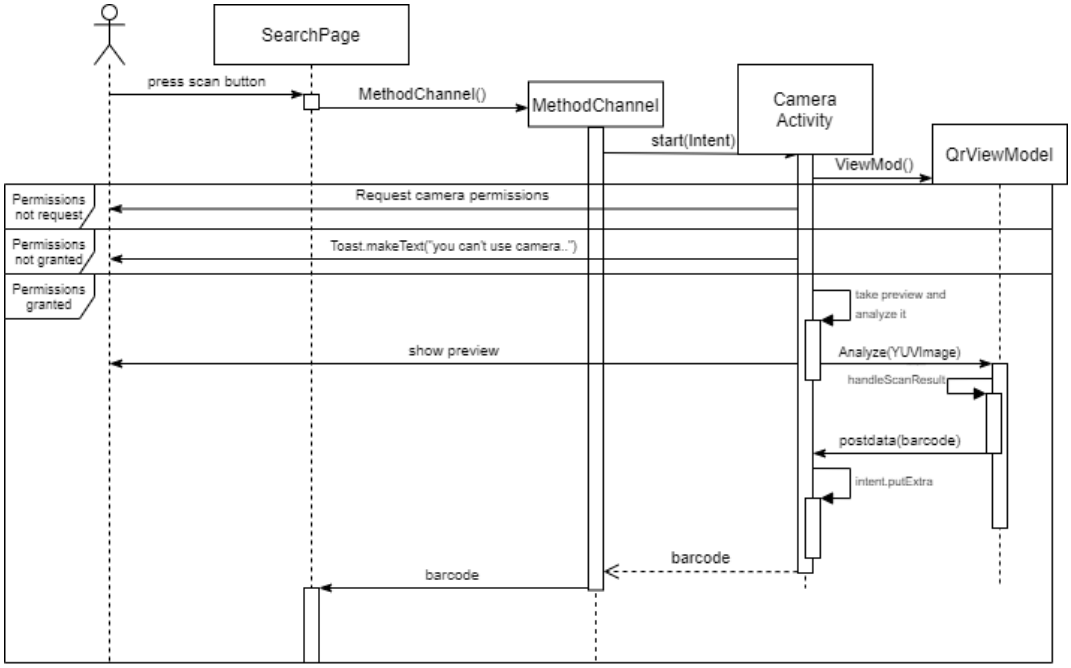


Figure 3.5: Sequence diagram about scanning a barcode

Access beer data After searching a beer using barcode or search bar, or after clicking on the image of a beer, a user can open a page containing its details. The page will retrieve from the database and the storage respectively information and the picture, using a background task. After the download process ends, the BLOC uses a stream to notify the UI. In the specific case of the barcode scan, it may happen that the barcode doesn't belong to any beer in the database. In this case, the user is notified that the beer is not available.

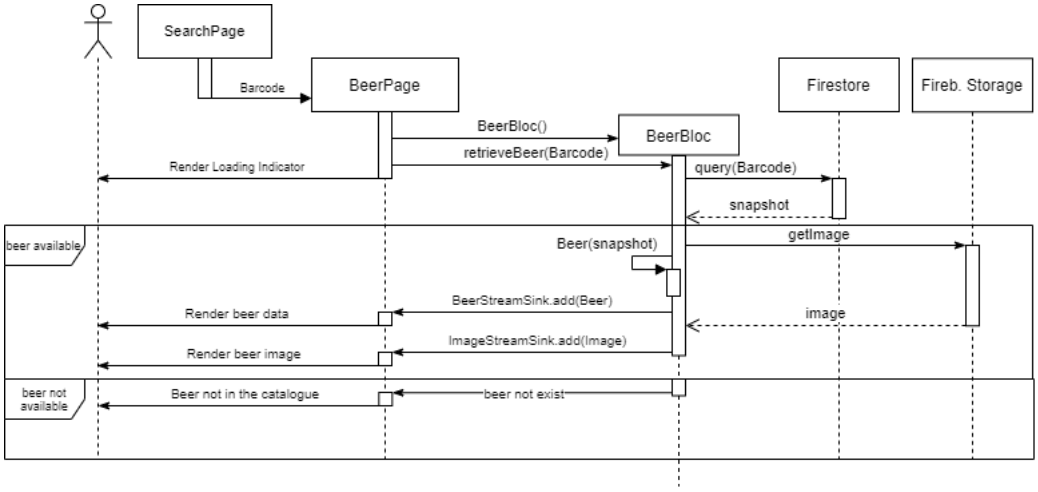


Figure 3.6: Sequence diagram about accessing beer data

Retrieve neighbouring cities Another fascinating element to analyze is how the application gets the nearest cities in the settings page. After using the geolocator plugin to obtain the coordinates of the user, the application computes the geo hash associated with that location. After that, given a specific precision (in Km), it calculates the geo hashes of the neighbouring areas to retrieves the cities in these zones. If the number of towns is lower than 2, the app increases the radius of the search and repeat this task. In the end, the app will render the widget containing the cities retrieved from the DB.

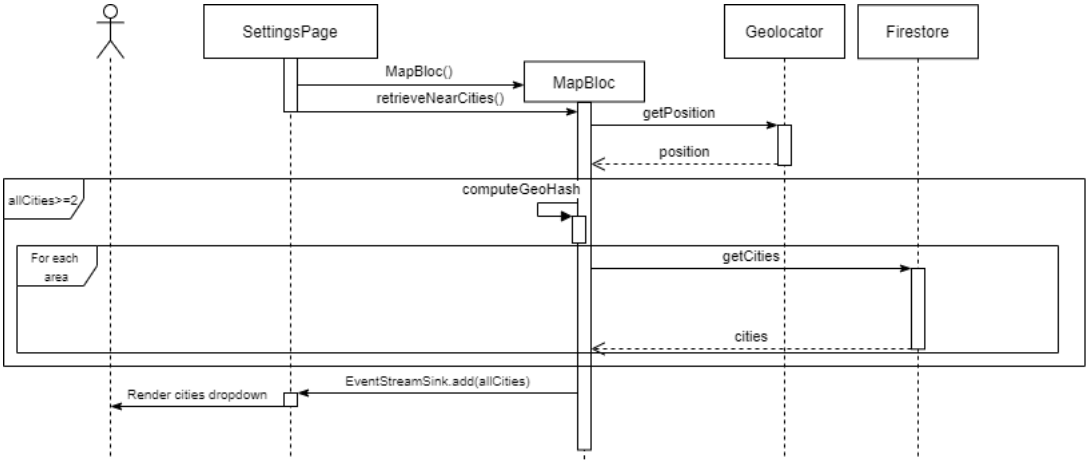


Figure 3.7: Sequence diagram about retrieving neighbouring cities

Retrieve events for a user We decided to put this sequence as the last in our analysis process since it is the last one that involves a little more complex interactions aside from queries to a database with some filters. When retrieving the event list in the home page, the first task done by the application is to get the user id from the authentication plugin and retrieve the user's record from the DB. After that, the app uses the city reference belonging to the user to select from which collection retrieve the events available in that town. Even if not directly shown by the sequence diagram, we implemented pagination to avoid the usage of too much bandwidth.

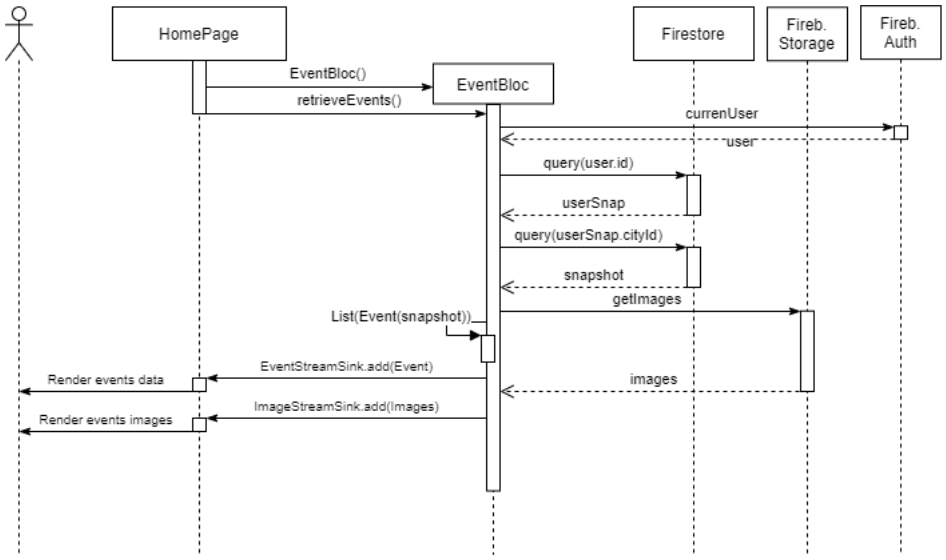


Figure 3.8: Sequence diagram about retrieving events in a city

3.5 Selected architectural styles and patterns

This section displays the various architectural styles and pattern that we decided to follow, to develop the system, during the design phase. In the next lines, we will explain the different pattern used, in what they consist, and why we chose them.

Business Logic Component BLoC is a state management system for Flutter recommended by Google developers. It helps in managing state and makes access to data from a central place in your project. This pattern helps to maintain the various components of the mobile application independent to the structure of the UI, leading to a more maintainable and testable

application. Bloc has his foundations in reactive programming and consists of using streams of events to update the UI and respond to changes inside the application.

Model View ViewModel MVVM is a software architectural pattern that helps to decouple the structure of the user interface (the view) from the business logic or back-end logic (the model), leading to a more maintainable and testable application. It is responsible for exposing and converting the data coming from the "model", to allow the "view" to process that information without being dependent on the model's structure.

Client-Server The client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. This pattern is used in different ways into the system: the application server behaves like a client when it communicates with the database and the docker server (which, in this case, held the role of the server), while it acts as a server when the web server and the mobile application perform operations using the provided RESTful APIs. Furthermore, the web server acts as a server when the web client query it using HTTPS.

Multi-tier architecture In the Multi-tier Architecture pattern, the various components of a system are allocated on different physical layers, which are not isolated, but they communicate through interfaces with at least another layer of the system. There are many benefits to using a multi-tier architecture: modularizing different tiers of an application gives development teams the ability to develop and enhance a product with higher speed than developing a unique code base because a specific layer can be upgraded with minimal impact on the other layers. Scalability is another great benefit of a multi-tier architecture. By distributing the different layers you can scale each autonomously depending on the need at any given time. Furthermore, having disparate layers can also increase the system's reliability and availability by hosting different parts of your application on different servers.

Representational state transfer REST is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. This improves the independence of the various tier of the system. Metadata about the resource

is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control. This patter is used to develop APIs for communications between the application server with the mobile client and the web server and between the docker server and the application server.

Thin client A thin client is a lightweight software that has been optimized for establishing a remote connection with a server-based computing environment. Having a thin client in our case is an advantage because all the application logic is allocated on the application server, which has enough computational power to manage concurrency efficiently.

3.6 Other design decisions

3.6.1 Security

To comply with the security constraints defined in subsection 2.5.5, we entirely rely on the security measures implemented in Firebase modules. After in-depth analysis, we have agreed that they were enough to secure users personal data and communication. Users' data are encrypted using a version of the SCRYPT algorithm, which implement a proper hashing and salting mechanism. For what concerns communication, firebase implement HTTPS or TLS (when its modules use WebSockets) to secure data transfers over the internet.

Chapter 4

User interface design

4.1 General design

In this section are mentioned further details on the design of the various UIs. User interfaces must conform to a uniform, intuitive and user-friendly design, that doesn't require the reading of detailed documentation to be used. They should comply with the guidelines provided by Google Material Design[5] to unifies the user experience across platforms, devices, and input methods. Text and important elements, like icons, should meet legibility standards when appearing on colored backgrounds, across all screen and device types. Material surfaces have consistent, unchangeable proprieties and behaviors across Material Design:

- **Dimension:** Material components has varying horizontal and vertical dimensions (measured in dp) and a uniform thickness (1dp).
- **Shadows:** Material surfaces at different elevations cast shadows
- **Resolution:** Material has infinite resolution
- **Content:** content is displayed in any shape and color on Material. Content does not add thickness to Material. Content is expressed without being a separate layer

Material design constraints also the navigation across screens, that can be lateral (refers to moving between screens at the same level of hierarchy), forward (refers to moving between screens at consecutive levels of hierarchy) and reverse (refers to moving backwards through screens either chronologically or hierarchically). For further details see Material guidelines [5].

4.2 Mobile interface

Design is a crucial topic in developing a mobile application. A key issue when conceiving a design is thinking that having a "nice-looking" app is enough to achieve it. The first error while approaching the design phase is not to put the user at the centre. When software is intuitive, users have a more pleasant experience and tend to use it again while, if they find difficulties, their painful experience will make them abandon the application. With this concept in mind, we tried to exploit the notions gained during the course, the design guidelines provided by Google Material Design[5] and also some graphic source available on Dribbble[6], to realize a pleasant experience for our users.

Navigation The first things to understand during the design phase was how users would have moved inside our service. We wanted to group the various elements in characterizing our offer by domain, to make as most intuitive as possible for users to find an event or a beer. To achieve our objective, we divided the application into three main sections:

- **Home page:** containing all the things that can explore and discover, like events and articles;
- **Search page:** where users can find beers by searching them or by chosing one of the proposals we selected to inspire them;
- **Profile page:** where users can log out and change their profile information.

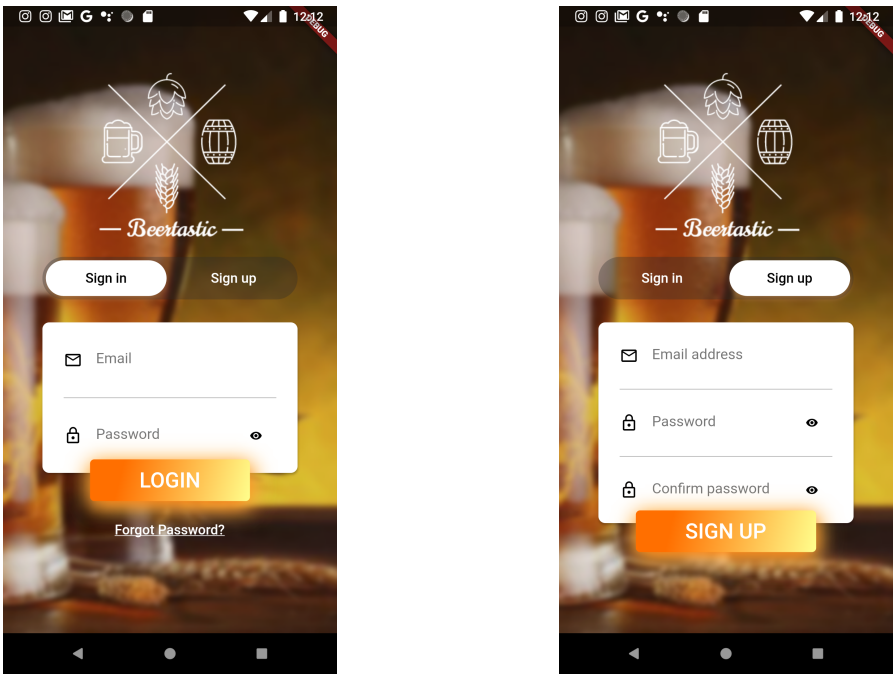
To let the user navigate among these sections, we used a Tab selector, which provides some buttons on the bottom of the display, where each button corresponds to a single screen, and which is the solution suggested by Material design in the case screens are from 2 to 5.

Interactive Components Always with the goal in mind of developing an app easy to handle for our users, we decided to put particular focus on how interactive elements, like buttons and cards, should behave. The first thing that we wanted to achieve was the possibility for our users to understand the state of these objects quickly. Each of these elements is animated when pressed to let users interact better with them. Furthermore, also in this context, we kept particular attention on the Material Guidelines for these objects.

Readibility In our project, we also paid particular attention to the legibility of the text. We put effort into ensuring that the weight of fonts and the contrast between text and background was adequate for our users to read with ease. In particular, we also made use of animations to grant legibility without renouncing to a beautiful appearance of the UI.

4.2.1 Authentication screen

The authentication screen, containing the forms to login and register in the service, is the one that appears when the user opens for the first time the application. The design of the interface is straightforward: there is the logo, a button to switch between forms used to sign in or sign up and the controls needed to run these processes. Furthermore, in the login screen, there is a button to open the page dedicated to the reset of the password

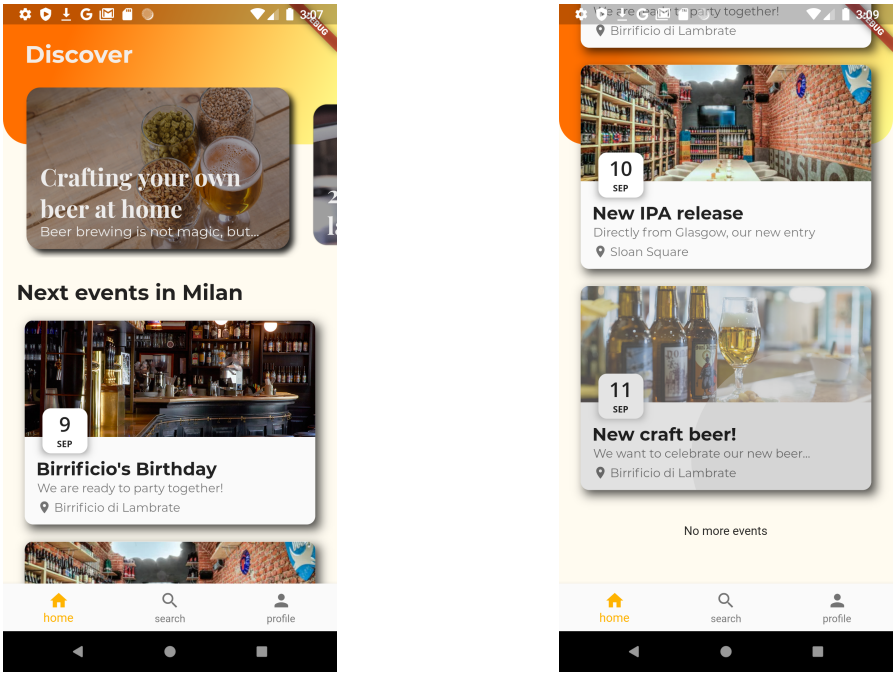


(a) Login screen (b) Register screen

Figure 4.1: Authentication screens

4.2.2 Home screen

The home page was designed to contain all the events and the article the users may want to explore. In the top, we used a pageview to let the user navigate among articles by scrolling the various widgets horizontally and can click on cards to open them. Just below articles, there is a section dedicated to the events in the city. User can scroll down the page to see more events and tap on them to see their details. Since pagination is implemented, if the user scrolls down to the very bottom of the page, the system will download other events, or it will notify him (or her) that they are finished. Furthermore, a gesture to refresh the page is integrated.



(a) Home page perspective 1

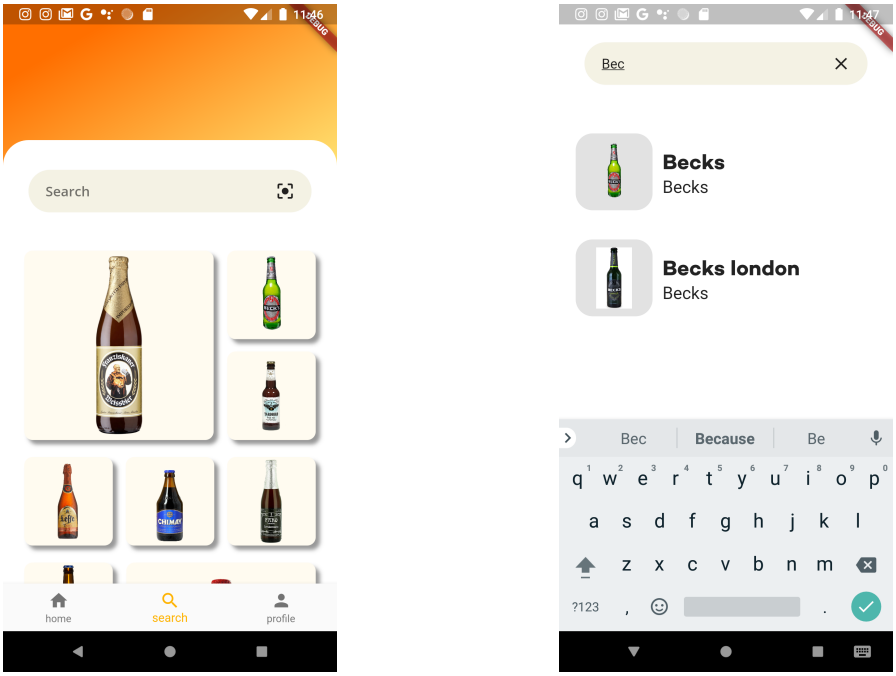
(b) Home page perspective 2

Figure 4.2: Home page screen

4.2.3 Search page

The search screen is the second main screen of the application. From this screen, users can access beers by searching them or by being inspired by our proposals. By scrolling down the page, users can explore unique beers belonging to our catalogue. Since pagination is implemented for beers too, when users reach the bottom of

the page, the system will download new beers if they are available in the DB. Otherwise, the UI will show the user a message that no more beers are available. Always from this screen users can start to research a beer by tapping on the "scan button", which is rendered using a proper icon that reminds to the user its purpose, or by clicking on the search bar. In the last case, a fluid animation will open clear the screen, allowing the user to see the results of his (or her) research in real-time. The user can dismiss the screen by clicking on the "X" button available in the search bar.



(a) Search page perspective 1

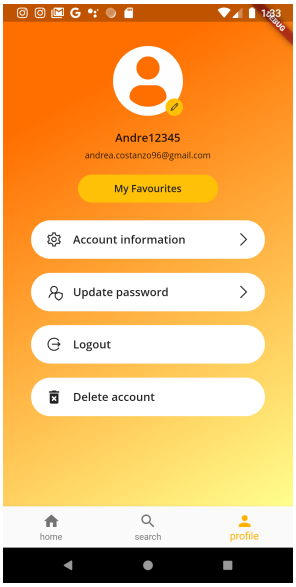
(b) Search page perspective 2

Figure 4.3: Search screen

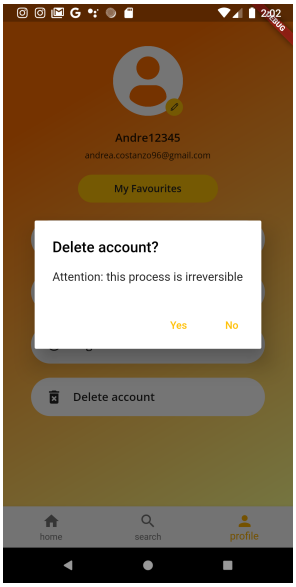
4.2.4 Profile page

The profile screen is the last that users can access through the tab selector. It shows the profile image of the user at the top, with his (or her) nickname and email. Below there are different buttons: the main one, in yellow, is to access favourites while the others can be used to access settings, log out from the service or delete the account. If the user presses the last button, the screen will pop up an alert dialogue to notify the user about the irreversibility of the process. It may

also happen that, if the users authenticated with credentials too long ago, the page redirects the user to the authentication page to confirm his (or her) identity.



(a) Profile page perspective 1



(b) Profile page perspective 2

Figure 4.4: Profile screen

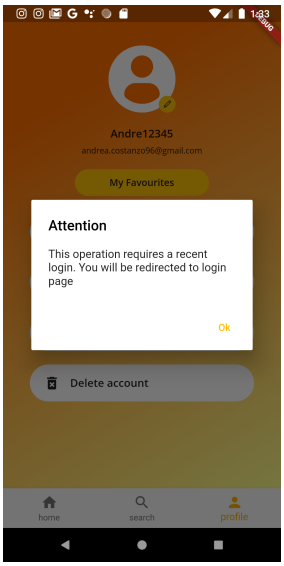
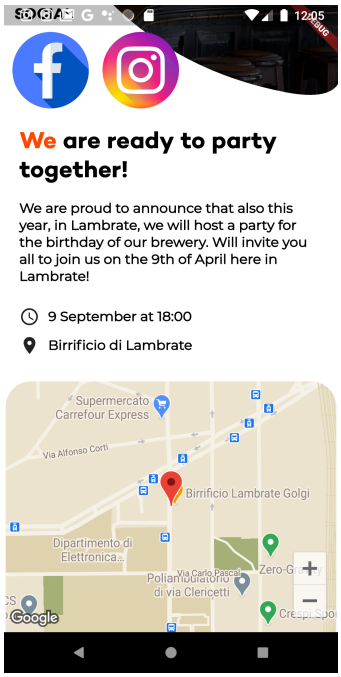


Figure 4.5: Profile page perspective 3

4.2.5 Event page

In the event page, we can find all the information about it. The design used is straightforward: on top, we can see the title and the name of the place. Below there are the links to the social pages of the business, and then we can find, in order, the punchline, the description, the information (place and time) and a map that can be used to Google Maps.

(a) Event page perspective 1



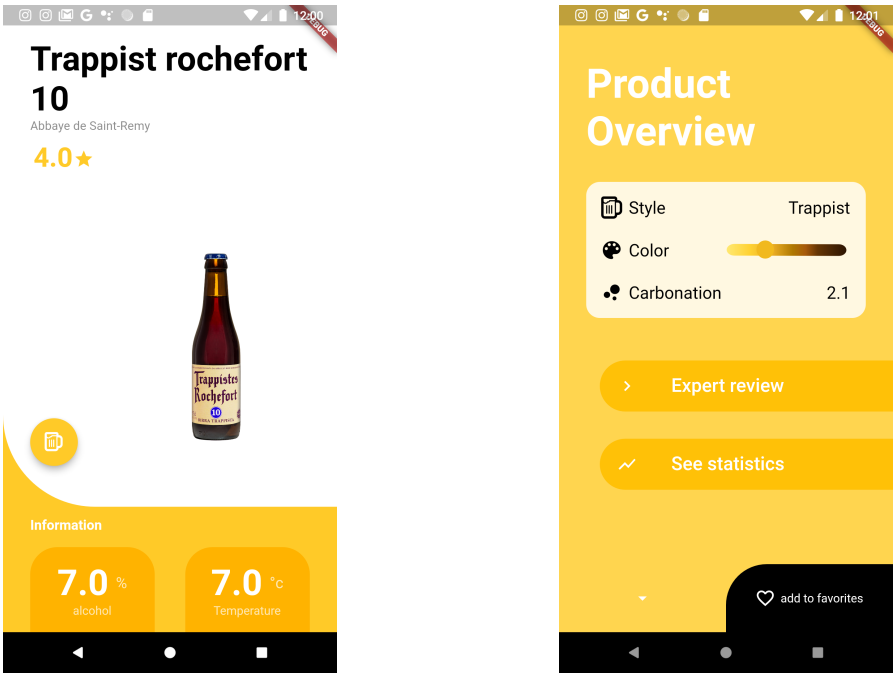
(b) Event page perspective 2

Figure 4.6: Event screen

4.2.6 Beer and beer details pages

The pages dedicated to a beer's information are 2: the one we will call "beer page" and the one we will define as "beer details page". The first contains some preliminary information about the beer, like name, producer, temperature, alcohol and rating. The interactive elements in this screen are two: the rate, positioned in the top-left of the UI, which allow accessing the reviews, and the main button (with a beer icon) that allows opening the details of the beer. After opening the screen dedicated to the main characteristics of the beverage, we can find in the top a small square dedicated to attributes like the type, the colour and the carbonation.

Below we can see three different buttons: the first two, in yellow, allow to access respectively the expert review and the statistics of the beer. Statistics are a small container that expands from the bottom and can be dismissed by tapping on the blurred screen or with a "swipe down" gesture. Lastly, the black button allows adding a beer to favourites. When clicked it became red, and it can be pressed again to remove a beer from favourite, causing its colour to become black again.



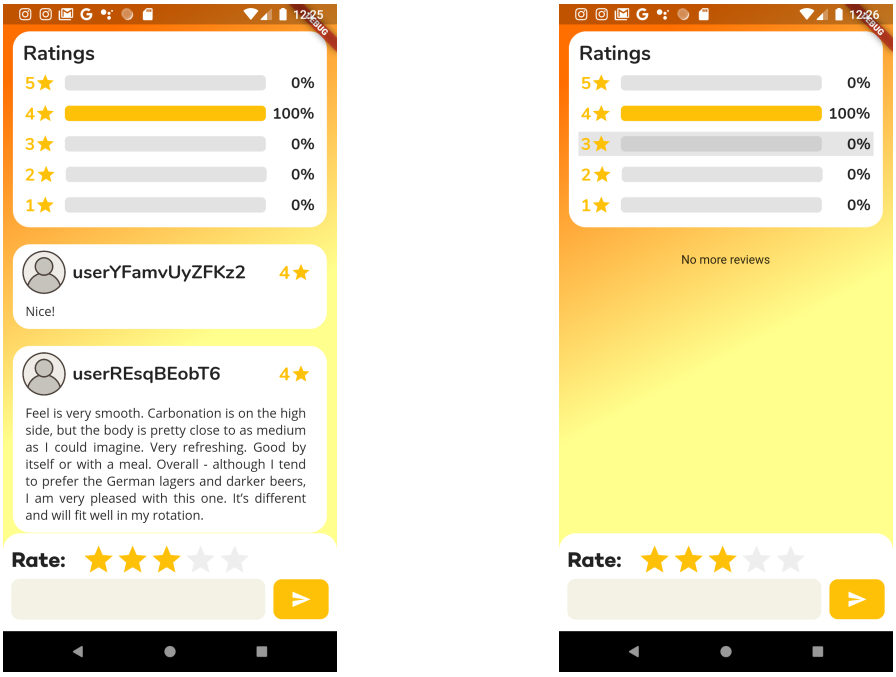
(a) Beer page (b) Beer details page

Figure 4.7: Beer pages

4.2.7 Reviews page

In the review page, we can find two main fragments: the first, which occupy most of the screen, contains the review of the other users, while the latter is a component placed at the bottom of the screen. In the section dedicated to others review, we can find a summary of the percentages of votes that the beer obtained for each rate. Users can tap on each row to filter reviews by grade and can click on the row again to return to the original state. Pagination and a "refresh gesture" are also implemented in this screen, allowing the user to scroll till the bottom to load new reviews or "swipe down" to reload reviews. For what regards the component at

the bottom of the page, users can use it to submit a rating for the beer. Then, the component updates and show our text and vote for that specific beer with a grey icon that can be used to delete our review and return the component appearance to the original state. If users don't delete their judgment, when they come back, they will find the widget in the condition they left it.



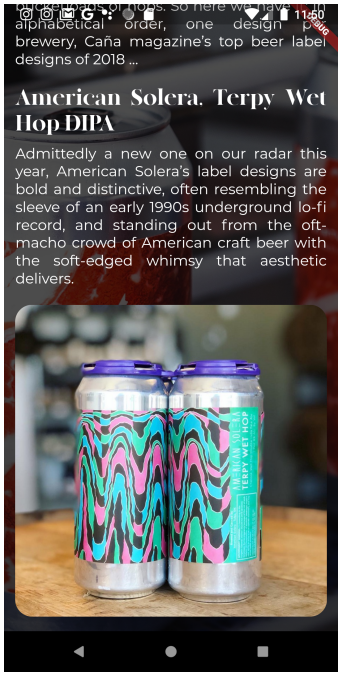
(a) Reviews page perspective 1 (b) Reviews page perspective 2

Figure 4.8: Reviews screen

4.2.8 Article and expert review pages

The article page and the expert review page are very similar. In designing them, we placed particular emphasis on legibility, to grant our users the best experience possible while they are reading. Both pages become darker when the users start to scroll the article or the review, making the text stands out from the background thanks to a higher contrast. The page can be divided into small subsections, each with a title, a description and, if available, an image. At the end of the page are inserted the author and a link to the source hosting that content. The only difference between articles and review is the tag showing the category of the article, which is not available for reviews.

(a) Article page perspective 1



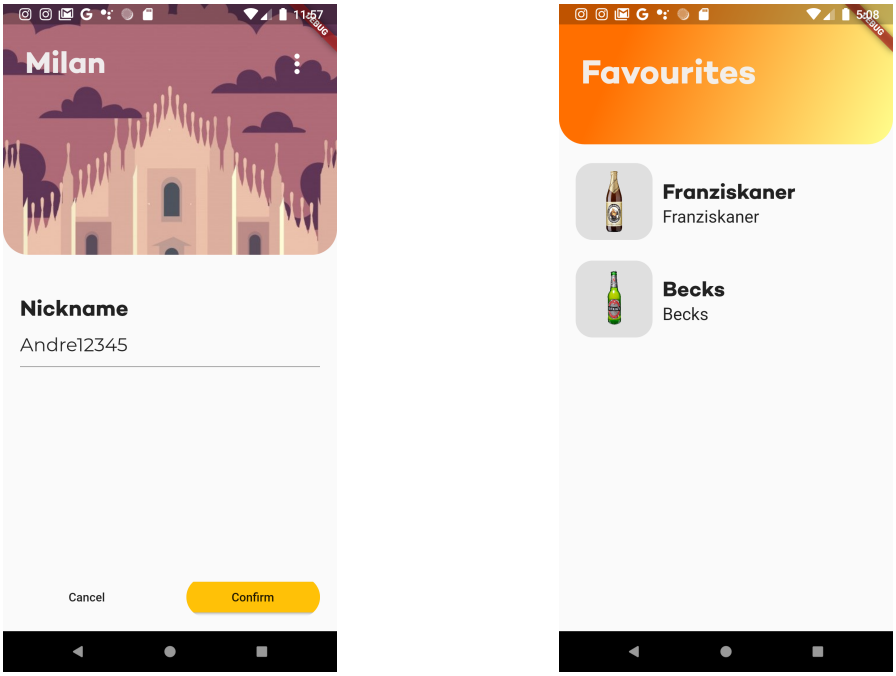
(b) Article/Expert review perspective 2

Figure 4.9: Article and expert review screens

4.2.9 Favourites and Settings page

The favourites screen is straightforward: it contains a title and a list view of all beers saved by the users. Even in this case, we kept our design approach of making interactive elements animated: if users tap on beers, the button will animate. For

what regards the settings screen, at the top we have the city in which the user is registered. Users can change it by clicking on the pop-up menu at the top-right of the screen, which will load the nearest city to the user. Furthermore, users can change their nickname, which can have a maximum of 14 characters. To save each change, the users have to tap on the "confirm" button; otherwise, they can use the "cancel" button to undo them.



(a) Settings page (b) Favourites page

Figure 4.10: Favourites and setting pages

Chapter 5

Implementation, integration and test plan

5.1 Implementation and Integration

5.1.1 Implementation

Beertastic is a complex system composed of different layers, modules and submodules that interacts and communicates among them. They have to be as much independent as possible to enable easy and fast maintenance or upgrade operations. Before starting with the implementation, the VCS and the CI tool must be correctly set up. Then, the database must be created and set up. Once realized this components, it's time to start developing the mobile application: first, the UI is implemented using mock data to understand the flow of the user experience inside the application. Then we develop the blocs that we will employ to retrieve data and talk with the server.

5.1.2 Integration

The integration of the system will be based on its architectural structure. The components that need to be integrated are:

Firebase Authentication It has the objective to perform the authentication process, including the management of authentication tokens and the encryption of users' information.

Cloud Firestore It's purpose is to act as a middleware between the clients and the database.

Firebase Storage This component is used to storage large files and objects.

Functions It is the module used to run server-side code.

ML It is the module that employs ML models to scan bar codes and QR codes.

Mobile application It's purpose is to act as front-end application to connect users with the service.

There is not a specific order of integration between these components since, aside for the mobile application, they are all COTS (provided using a SaaS paradigm) and ready to be used.

5.2 Testing

5.2.1 Introduction

To implement the system, we decide to test the application mainly through debug on both emulators and real devices since most of the system components are offered as SaaS. The only exception is the mobile application, for which we also performed unit testing on the main business logic components. Furthermore, to speed up the development process, the CI tool runs all these tests every time the remote repository is updated and it notifies the developers in case of failures. Furthermore, the results of tests can be seen also from the dedicated badge in the repository:

Figure 5.1: Repository

5.2.2 Unit testing

For what concerns unit testing, the objective is to ensure that the business logic components are behaving correctly. For this reason, the purpose was to test blocs

to reach a coverage at least of the 80%. However, since it was impossible to mock permissions and because of some bugs in third-party packages, we got a little more than the 70%. To keep track of tests and build status of the repository, we implemented a custom pipeline using CircleCI[7]. This choice allowed to speed up the testing activities, ensuring that everything was under control. Furthermore, we integrated Codecov[8] in the pipeline to have a detailed report of the code coverage.

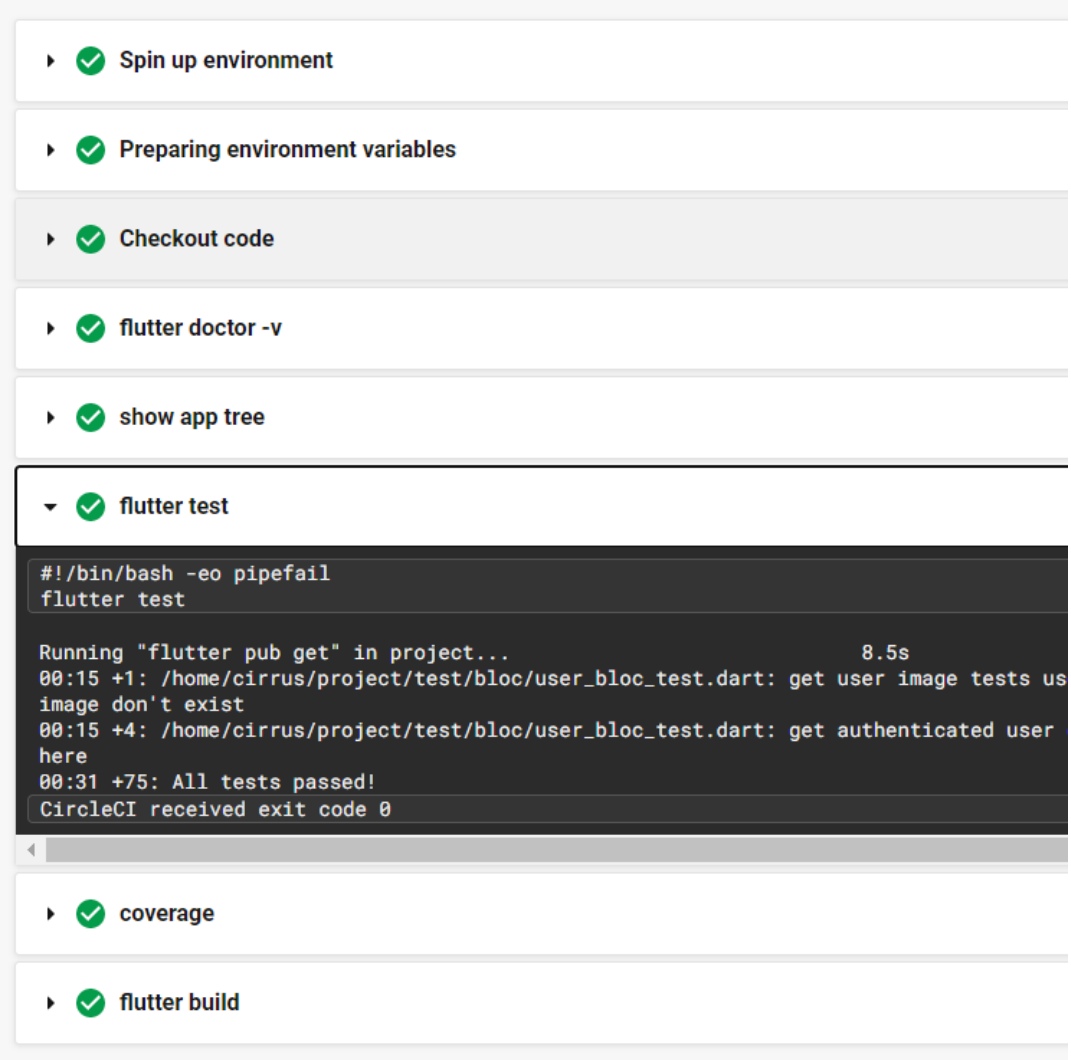


Figure 5.2: CircleCI pipeline

5.2.3 Debug

In this subsection are listed all the activities performed to debug the application. They are organized page by page, and they cover all the interactions that a user can have with the application.

Name	Register [Authentication pages]
Actor	User
Input condition	User has Beertastic installed on their smartphone, and it is not registered.
Event Flow	1. The user opens the application 2. The user clicks on the sign up button 3. The user inserts his/her email 4. The user inserts a password 5. The user confirms the password 6. The user presses the register button
Output condition	The system logs the user into the service
Exception	1. Email patter not respected ⇒ email error reported by the app 2. Password pattern not respected ⇒ password error reported by the app 3. The user already exists ⇒ user notified to use sign in

Name	Login [Authentication pages]
Actor	User
Input condition	User has Beertastic installed on their smartphone, and it is registered but not authenticated.
Event Flow	1. The user opens the application 2. The user inserts his/her email 3. The user inserts a password 4. The user presses the login button
Output condition	The system logs the user into the service
Exception	1. Email patter not respected ⇒ email error reported by the app 2. Password pattern not respected ⇒ password error reported by the app 3. Wrong password inserted ⇒ password error reported by the app 4. The user not exists ⇒ user notified to use sign up

Name	Recover password [Authentication pages]
Actor	User
Input condition	User has Beertastic installed on their smartphone, and it is registered but he/she doesn't remember his/her password.
Event Flow	1. The user opens the application 2. The user click on forgot password 3. The user inserts his/her email 4. The user receives an email 5. The user modify his/her password
Output condition	The system change user's password
Exception	1. Email patter not respected ⇒ email error reported by the app 2. The user not exists ⇒ user notified to use sign up

Name	Open article [Home page]
Actor	User
Input condition	User is logged and has the home page open.
Event Flow	1. The user scrolls articles until he found one interesting. 2. The user taps on the article's card.
Output condition	The system opens the article page.
Exception	-

Name	Open event [Home page]
Actor	User
Input condition	User is logged and has the home page open.
Event Flow	1. The user scrolls events until he found one interesting. 2. The user taps on the event's card.
Output condition	The system opens the event page.
Exception	-

Name	Refresh events [Home page]
Actor	User
Input condition	User is logged and has the home page open.
Event Flow	1. The user swipes down the page until a refresh indicator appears. 2. The user waits until the refresh indicator disappear
Output condition	The system refreshes articles.
Exception	-

Name	Load more events [Home page]
Actor	User
Input condition	User is logged and has the home page open.
Event Flow	1. The user scrolls down the page to the bottom
Output condition	The system load more events if they are available, otherwise it notifies the user that there isn't any additional event.
Exception	-

Name	Darken background [Article page/Expert review page]
Actor	User
Input condition	User is logged and has the article page open.
Event Flow	1. The user scrolls down the page.
Output condition	The system will make the background progressively darker.
Exception	-

Name	Open source [Article page/Expert review page]
Actor	User
Input condition	User is logged and has the article page open.
Event Flow	1. The user scrolls down the page to the very bottom. 2. The user taps on "Source here" 3. The user selects a web browser
Output condition	The system opens the web browser with the source page.
Exception	-

Name	Open Facebook [Event page]
Actor	User
Input condition	User is logged and has the event page open.
Event Flow	1. The user taps on the button with the Facebook icon
Output condition	The system opens the Facebook app with the event/location page.
Exception	-

Name	Open instagram [Event page]
Actor	User
Input condition	User is logged and has the event page open.
Event Flow	1. The user taps on the button with the Instagram icon
Output condition	The system opens the Instagram app with the event/location page.
Exception	-

Name	Open maps [Event page]
Actor	User
Input condition	User is logged and has the event page open.
Event Flow	1. The user scrolls down the page. 2. The user taps on the landmark icon in the map widget 3. The user click on google maps icon
Output condition	The system opens google maps with the location of the event.
Exception	-

Name	Open beer [Search page]
Actor	User
Input condition	User is logged and has the search page open.
Event Flow	1. The user clicks on the beer card
Output condition	The system opens the beer page.
Exception	-

Name	Search beer with barcode [Search page]
Actor	User
Input condition	User is logged and has the search page open.
Event Flow	1. The user clicks on the button with the scan icon 2. The user gives to the app camera permissions. 3. The user scans the bar code
Output condition	The system opens the beer page if a the beer is available, otherwise notifies the user that the beer is not available
Exception	-

Name	Search beer by typing [Search page]
Actor	User
Input condition	User is logged and has the search page open.
Event Flow	1. The user clicks on the search bar 2. The user types the name of the beer 3. The user clicks on the beer he/she wants to open
Output condition	The system opens the beer page.
Exception	-

Name	Load other beers [Search page]
Actor	User
Input condition	User is logged and has the search page open.
Event Flow	1. The user scrolls down to the bottom of the page.
Output condition	The system loads new beers if available, otherwise it notifies the user that there isn't any beer available
Exception	-

Name	User don't agree with camera permissions [Search page]
Actor	User
Input condition	User is logged and has the search page open.
Event Flow	1. The user clicks on the button with the scan icon 2. The user doesn't give to the app camera permissions.
Output condition	The system tells the user he can't use the feature without giving permissions
Exception	-

Name	Open details page [Beer page]
Actor	User
Input condition	User is logged and has the beer page open.
Event Flow	1. The user taps on the button with the beer icon
Output condition	The system opens the page containing further details on the beer.
Exception	-

Name	Open reviews [Beer page]
Actor	User
Input condition	User is logged and has the beer page open.
Event Flow	1. The user taps on the rate of the beer
Output condition	The system opens the page containing the reviews of the beer.
Exception	-

Name	Add to favourites [Beer details page]
Actor	User
Input condition	User is logged, has the beer details page open and the beer is not among his/her favourites.
Event Flow	1. The user taps on the button with the love icon
Output condition	The system saves the beer among user's favourites and turn the color of the button to red
Exception	-

Name	Remove from favourites [Beer details page]
Actor	User
Input condition	User is logged, has the beer details page open and the beer is among his/her favourites.
Event Flow	1. The user taps on the red button with the love icon
Output condition	The system removes the beer among user's favourites and turn the color of the button to black
Exception	-

Name	Open stats [Beer details page]
Actor	User
Input condition	User is logged, has the beer details page open.
Event Flow	1. The user taps on the statistics button.
Output condition	The system opens the statistics box
Exception	-

Name	Close stats [Beer details page]
Actor	User
Input condition	User is logged, has the beer details page open and the beer stats are opened.
Event Flow	1. The user taps on the blurred background or swipes down.
Output condition	The system closes the statistics box
Exception	-

Name	Open expert review [Beer details page]
Actor	User
Input condition	User is logged, has the beer details page open.
Event Flow	1. The user taps on the expert review button.
Output condition	The system opens the page of the expert review
Exception	-

Name	Add review [Beer reviews page]
Actor	User
Input condition	User is logged, has the beer reviews page open, and he/she didn't review the beer.
Event Flow	1. The user taps on the text input (optional) 2. The user writes his review (optional) 3. The user selects a rate using the rate bar 4. The user taps the button with the "send" icon
Output condition	The system adds the review and show the user the box containing his review
Exception	-

Name	Delete review [Beer reviews page]
Actor	User
Input condition	User is logged, has the beer reviews page open, and he/she reviewed the beer.
Event Flow	1. The user taps the button with the "delete" icon
Output condition	The system removes the review and show the user the box to write another review
Exception	-

Name	Filter reviews [Beer reviews page]
Actor	User
Input condition	User is logged and has the beer reviews page open.
Event Flow	1. The user taps on the the bar of a specific rate
Output condition	The system shows only the reviews with that specific grade and the background color of the bar becomes darker
Exception	-

Name	Update filter [Beer reviews page]
Actor	User
Input condition	User is logged, has the beer reviews page open and is filtering reviews for a specific grade.
Event Flow	1. The user taps on the the bar of a rate different from the one he/she selected before
Output condition	The system shows only the reviews with the new grade. The background colour of the bar of the previous grade turns back to normal while the one related to the new rate becomes darker
Exception	-

Name	Stop filtering [Beer reviews page]
Actor	User
Input condition	User is logged, has the beer reviews page open and is filtering reviews for a specific grade.
Event Flow	1. The user taps on the the bar of the rate he/she selected
Output condition	The system shows all the reviews and the background color of the bar turns back to normal
Exception	-

Name	Load more reviews [Beer reviews page]
Actor	User
Input condition	User is logged and has the beer reviews page open.
Event Flow	1. The user scrolls down to the bottom of the page
Output condition	The system load more reviews or shows the message no more reviews available
Exception	-

Name	Update profile image [Profile page]
Actor	User
Input condition	User is logged and has the profile page open
Event Flow	1. The user taps the button with the pen icon just below the profile image widget 2. The user selects an image from storage
Output condition	The system updates user's profile image.
Exception	-

Name	Open favourites [Profile page]
Actor	User
Input condition	User is logged and has the profile page open
Event Flow	1. The user taps the button "My favourites"
Output condition	The system opens the "Favourites" page
Exception	-

Name	Open account info [Profile page]
Actor	User
Input condition	User is logged and has the profile page open
Event Flow	1. The user taps the button "Account information"
Output condition	The system opens the "Account information" page
Exception	-

Name	Open account info [Profile page]
Actor	User
Input condition	User is logged and has the profile page open
Event Flow	1. The user taps the button "Account information"
Output condition	The system opens the "Reset password" page
Exception	-

Name	Logout [Profile page]
Actor	User
Input condition	User is logged and has the profile page open
Event Flow	1. The user taps the button "Logout"
Output condition	The system redirects the user to the authentication page
Exception	-

Name	Delete account [Profile page]
Actor	User
Input condition	User is logged and has the profile page open.
Event Flow	1. The user taps the button "Delete account" 2. The user taps "Yes" on the alert dialog that appears on the screen
Output condition	The system delete user's account and all its data and redirects him to the "Authentication page"
Exception	1. User did not authenticated recently: • Systems open an alert dialog telling to the users that a recent authentication is needed • Users taps on "OK" • The system redirects the user to the authentication page • The user authenticate in to the service • The user repeat the deletion process

Name	Users change his/her mind about deleting [Profile page]
Actor	User
Input condition	User is logged, has the profile page open and authenticated himself/herself recently
Event Flow	1. The user taps the button "Delete account" 2. The user taps "No" on the alert dialog that appears on the screen
Output condition	Nothing happens and the users remains in the profile page
Exception	-

Name	Update city [Account information page]
Actor	User
Input condition	User is logged and has the account information page open.
Event Flow	1. The user taps on the button with three dots 2. The user select a city 3. The user press confirm
Output condition	The system updates user's city.
Exception	-

Name	Update city [Account information page]
Actor	User
Input condition	User is logged and has the account information page open.
Event Flow	1. The user taps on the text field of the nickname 2. The user inserts a nickname 3. The user press confirm
Output condition	The system updates user's nickname.
Exception	-

Name	Open beer [Favourites page]
Actor	User
Input condition	User is logged and has the favourites page open
Event Flow	1. The user taps on a beer widget
Output condition	The system opens the dedicated "Beer" page
Exception	-

Appendix A

Appendix

A.1 Software and tools used for the DD

[**S.1**] Draw.io - *Used to realize the sequence diagrams* - `https://www.draw.io/`

[**S.2**] Git - *Distributed Version Control System* - `https://git-scm.com/`

[**S.3**] Github - *web-based hosting service for version control using Git* - `https://github.com/`

[**S.4**] Overleaf - *Used to realize the pdf version of this document* - `https://www.overleaf.com/`

A.2 Other references

- Freepik.com - *Used for some pics in the mobile application* - `https://it.freepik.com/`

A.3 Effort Spent

- **Mobile application & back-end:** 250 hours
- **Design document:** 35 hours
- **Slides and Video:** 20 hours

Bibliography

[1] Dart 2.9.1 - *client-optimized language for fast apps on any platform* - `https://dart.dev/`

[2] Flutter 1.17.5 - *Google's portable UI toolkit for building beautiful, native applications for mobile, web, and desktop from a single codebase* - `https://flutter.dev/`

[3] Android - *Android 11, the newest version of Android, sets you up to take advantage of a range of new experiences, from foldable devices to stronger protections for your users.* - `https://developer.android.com/`

[4] IEEE Standard 1016:2009 - *IEEE Standard for Information Technology–Systems Design–Software Design Descriptions*

[5] Material Design - *A design system that helps teams build high-quality digital experiences* - `https://material.io/design/`

[6] Dribbble - *Dribbble is the leading destination to find & showcase creative work and home to the world's best design professionals.* - `https://dribbble.com/`

[7] CircleCI - *Automate your development process with CI hosted in the cloud or on a private server.* - `https://circleci.com/`

[8] Codecov - *The leading, dedicated code coverage solution.* - `https://codecov.io/`