UNIVERSIDADE DE AVEIRO

Gamifying Rural Tourism - Ruralia

Alberto Oliveira

Eduarda Aires

Beatriz Agante

Miguel Gomes

Pedro Azevedo

Ruben Lopes

WORKING VERSION

Licenciatura de Engenharia Computadores e Informática

Supervisors: Prof. Daniel Corujo & Prof. Pedro Gonçalves

June 8, 2024

Abstract

Tourism holds a crucial position in supporting Portugal's economy, serving as one of its primary contributors. While the allure of urban and coastal destinations remains prominent, our project recognizes the untapped potential and innate beauty of Portugal's rural areas. Fueled by the desire to shift the focus to these natural landscapes, we seek to enhance the rural tourism experience through the innovative approach of gamification. In pursuit of this goal, we have developed an application designed to immerse users in the heart of rural activities - RURALIA. Users are invited to engage in various fun tasks within farms, such as harvesting fruits and vegetables, or interacting with farm animals, while being awarded with points that unlock an array of enticing goodies belonging to the local business. This gamified approach not only cultivates an interactive and enjoyable rural tourism experience but also contributes to the sustainable growth of this vital sector.

Keywords: Gamification, Rural, Tourism

Resumo

O turismo desempenha uma posição crucial na economia Portuguesa, sendo um dos seus principais contribuidores. Apesar da atração por destinos urbanos e costeiros permanecer proeminente, o nosso projeto reconhece o potencial inexplorado e a beleza inata das áreas rurais de Portugal. Impulsionados pelo desejo de deslocar o foco para estas paisagens naturais, procuramos aprimorar a experiência do turismo rural através de uma abordagem inovadora - a gamificação. Em busca deste objetivo, desenvolvemos uma aplicação projetada para envolver os utilizadores no coração das atividades rurais - a RURALIA. Os utilizadores são convidados a participar em diversas tarefas divertidas em quintas, como a recolha de frutas e vegetais ou interagir com os animais da quinta, sendo recompensados com pontos, que lhes permitem comprar uma variedade de produtos atrativos pertencentes aos negócios locais. Esta abordagem, com o auxílio da gamificação, não só cultiva uma experiência interativa e agradável do turismo rural, mas também contribui para o crescimento sustentável deste setor vital à nossa economia.

Keywords: Gamificação, Rural, Turismo

Contents

List of Figures

Abbreviations

API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
ER	Entity-Relationship
ERD	Entity-Relationship Diagram
IOS	iPhone Operating System
LECI	Licenciatura em Engenharia de Computadores e Informática
MQTT	Message Queuing Telemetry Transport
PV	Persistent Volume
PVC	Persistent Volume Claim
QA	Quick response
SQL	Structured Query Language
UA	Universidade de Aveiro
UC	Unidade Curricular
UI	User Interface
UX	User Experience
WWW	World Wide Web
XP	Experience Points

Chapter 1

Introduction

Based on statistics regarding rural tourism in Portugal from a blog post on Forum Estudante [1], it becomes evident that this activity is emerging as an increasingly popular option. Dating back to 1989, agrotourism establishments comprised a modest 5% of the tourism scenario in Portuguese rural regions. However, over the course of three decades, we have witnessed a **remarkable evolution as this percentage, reaching 17%**.

This data reflects an exponential growth in rural tourism activities. While the overall number of tourist establishments has grown by approximately 700% over the last three decades, **agrotourism ventures stand out with an astonishing increase of around 2500%**.

However, this upward trend brings some challenges, particularly in rural areas. The lack of digital literacy and the variable quality of internet connection emerge as notable obstacles. It is within this context that our project is situated, aiming not only to comprehend but also to overcome these challenges. By integrating gamification and technology, we aspire not only to elevate the agrotourism experience but also to contribute to the socio-economic development of these regions.

1.1 Context

This project, conducted within the Computer and Informatics Engineering degree, aims to develop a system capable of monitoring and tracking the gamification of tourism operations in rural Portugal. Through the development of a mobile application and the aid of gamification, we hope to enrich and elevate the agrotourism experience for users. Our gamification experience focuses on a task-reward system, commonly found in videogames, where users can engage in various farm activities in order to accumulate points or other rewards. This should make the agrotourism a much more engaging and immersive experience. This project was developed by LECI students in UA.

1.2 Motivation

The motivation behind this project was to capitalize on the escalating adoption of technologies and strategical use of gamification in order to improve the experience of agrotourism. Two of the key factors behind this are:

- **Limited Awareness of Rural Areas**: the undiscovered charm and cultural wealth tucked away in rural Portugal often remain overshadowed by urban destinations. This sadly leads to an under utilization of these areas.

- **Technological Integration**: integrating gamification and technology into rural tourism experiences can be a challenge, as communities in these areas often lack digital literacy and the internet connection there is usually not very reliable. However, we hope to face this challenge in order to elevate the agrotourism experience.

1.3 Goals

In pursuit of enhancing the tourism experience in rural Portugal, this project has set various goals that drive the development of the RURALIA mobile application. The goals we hope to achieve in the short term are:

- **Enhance User Engagement**: the implementation of gamification elements in the application should elevate the user's agrotourism experience as they interact with the system by performing tasks and receiving rewards;

- **Improve Accessibility**: making the application user-friendly for people of all ages and tech backgrounds, as tourism should be an engaging activity for everyone. Make RURALIA an easy to understand, straightforward system, ensuring accessibility and a hassle-free experience.

In addition to our immediate objectives, our vision for the RURALIA project extend into the long term. These goals are outlined below:

- **Boost the Agrotourism Activity**: by injecting a dose of fun in the agrotourism scene, we hope to increase the number people interested in this activity, shifting their focus from the cities to the beautiful rural areas Portugal has to offer.

- **Economic Development of Local Communities**: boosting tourism activities in rural areas should support the economical growth of local businesses. Not only does RURALIA attract more people, it also offers a taste of what these businesses have to offer through a reward system, hoping to inspire curiosity and hook more people into them. This should catalyze economic growth for local communities.

- **Expanding Partnerships**: although we should forge some partnerships right from the start, in the long term we hope to expand collaborations with more local businesses, tourism authorities and community organizations, creating a network that supports the growth of agrotourism.
- **Provide Insights**: by monitoring and tracking tourism activities, the long term collection of this data should prove to be a valuable asset when it comes to analyzing agrotourism trends. Providing these insights can be a powerful tool for further enhancement of the tourism experience or aiding local businesses and communities.

1.4 Document Structure

Beyond this introductory section, this report has two more chapters. In Chapter 2, we approach the state-of-the-art, exploring related projects and the technologies that we use in this project. In Chapter 3, we shift our attention to the system's requirements (functional and non-functional), as well as exploring real-life scenarios as use cases with the aid of actors. We also dive into the architecture of the system, presenting various models for better understanding of the project's structure.

Chapter 2

State-of-the-Art

2.1 Related Projects

In this section we've explored other android applications that share a common goal with our project - the gamification of tourism experiences, while briefly explaining how each application approaches this matter in a different way. Exploring these applications was interesting to see how each one of them approaches gamification in a different way - this analysis could also inspire our team to add new gamified features to our app in the future.

2.1.1 Explorial

Explorial [6] flips the script when it comes to typical city sightseeing, transforming it into an engaging treasure hunt. This application invites users to navigate cities by following specific routes and moving to various locations where they can solve puzzles and do tasks. This application mixes tourism with treasure hunting, and awards the users with points in order to maintain a sense of competition. In order to participate in this experience, the user has to buy tickets from Explorial's website.

Figure 2.1: Logo of Explorial

2.1.2 Framie App

Framie [7] transforms tourism into a modern-day "sticker collection" adventure. Users are presented with an interesting challenge: matching pictures of tourism attractions within the app (called "framies") with their own snapshots of these same attractions. This method encourages users to keep a collection of their travel memories through these pictures, crafting a fun travel catalogue that grows with every new adventure.

Figure 2.2: Logo of Framie App

2.1.3 Portugal Play

Portugal Play [24] blends technology, creativity, and a love for Portuguese culture into an interactive augmented reality experience that encourages families to embark on a captivating adventure across the national territory. It's a genuine treasure hunt, unraveling the richness of national icons, Portuguese identity, and tourist landmarks. With this application, participants dive into an immersive journey, exploring the essence of the country while discovering hidden gems of cultural significance.

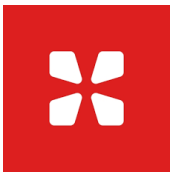

Figure 2.3: Logo of Portugal Play

2.2 Distinguishing Aspects

What makes RURALIA stand out from the three applications we've looked into is primarily the specific tourism niche it caters to. While these apps also aim to make tourism a fun and interactive experience, they mainly target urban tourism—exploring and sightseeing in cities. RURALIA, on the other hand, is pioneering the gamification of rural tourism experiences, bringing users fresh adventures and a chance to discover the charm of the countryside. It's not just a different application; it's a doorway to a whole new way of exploring and enjoying the beauty of rural Portugal.

Figure 2.4: Logo of Ruralia

2.3 Gamification

Our approach to gamification is done through an action-reward system. The user must complete an activity (by engaging in all sorts of tasks, from harvesting to taking care of animals) in order to get a reward - in this case, they are awarded with points. These points can then be spent on vouchers for various farm goodies. Many games have taken this kind of approach, since it motivates the user to play and get rewarded for their effort. According to Wang and Sun [8], reward systems with support of mobile technology have been used to encourage a range of activities, including shopping, traveling, and exercise. As stated in the section "REWARD SYSTEM DESIGN CONSIDERATIONS" of their article, the idea of promoting physical contact and healthy behaviors through gamification has been shown to increase user engagement. Our gamification approach aligns with these findings, as it leverages mobile technology to incentivize users to engage in physical world activities and promotes positive health outcomes.

Figure 2.5: Gamification - How does it work?

2.3.1 Forest

Forest is a productivity app that helps users stay focused and reduce distractions while working or studying. Users start by planting a virtual tree and setting a timer for their focus session. During the session, they're encouraged to stay off their phones and avoid distractions. If they exit the app, the tree will wither and die. Successfully completing a focus session allows the tree to grow and flourish in the user's digital forest, and awards them with coins. These can be used in the game's shop to unlock different species of virtual trees. This application follows a gamified approach to improve productivity. Similar to RURALIA, it encourages the user to complete an activity outside of the application (a studying session, a workout...) to win points (coins) and then redeem rewards (more trees). While in this case the user gets rewarded with in-game products, in our application they can win real-life goodies from the farms.

Figure 2.6: Forest's shop where the users can spend points.

This figure shows the in-game shop that Forest has, which allows the users to spend the coins they've earned on new trees for their forest.

2.4 Technologies

In this section we explore the technological framework of our project, sharing the thought process behind our selection of these specific technologies.

2.4.1 Front-End framework: Flutter

Our choice of front-end framework, Flutter, stems from its exceptional capability of simplifying the development of applications that can run on multiple platforms (Android, iOS) using a single codebase. This allows developers to write the code once and use it for both platforms - which proves handy when it comes to saving time and elevating efficiency. Flutter's strength also lies in crafting visually appealing and responsive user interfaces, ensuring a consistent experience across various devices. With RURALIA we hope to deliver a user-friendly and visually engaging experience, and we feel that Flutter best aligns with our wishes.

2.4.2 Back-End Framework: Flask

For back-end framework, we've chosen Flask, due to its simplicity and efficiency in building robust web applications. Flask is commonly celebrated for its minimalist design, allowing developers to focus on specific project requirements without the weight of unnecessary components. This framework's lightweight nature does not compromise functionality, as it instead empowers us to craft a scalable and maintainable back-end system. Although there were other popular back-end options such as Django, a powerful back-end framework packed with features, we feel that it could be excessive for a project like ours, where we wish to seek a more light-weight and adaptable back-end structure.

2.4.3 Database: SQLite

We've chosen SQLite to meet the demand for a well-organized and reliable data management system. SQLite excels in providing a clear and organized approach to storing and retrieving information, and the relational nature of SQLite ensures the integrity and consistency of data, making it an optimal choice for our project. Additionally, our decision to use SQLite relies on our own experience gained in previous lectures (of another UC - Base de Dados), making it a familiar and comfortable option for our team to work with. This familiarity contributes to a smoother development process, which aligns with our goals.

2.4.4 Containers and Management: Docker [5] and Kubernetes [12]

Our adoption of Docker [5] and Kubernetes [12] on our project came as a valuable suggestion from our supervisors, introducing us to something we had never worked with before. This recommendation prompted us to look into these technologies, conducting research in order to understand how they work and how to use them in our project. Docker [5], serving as our containerization platform, simplifies the packaging and deployment of our application and its dependencies, ensuring consistency across various environments. Kubernetes [12], chosen as our orchestration tool, further enhances efficiency by automating the deployment, scaling, and management of these containers. Its robust features, including load balancing and automatic scaling, contribute to the reliability and performance of our application. Together, Docker [5] and Kubernetes [12] form a powerful duo, empowering us to streamline our development and deployment processes, thanks to the insightful guidance of our supervisors which lead us to explore innovative solutions for our project.

Chapter 3

System Requirements and Architecture

3.1 System Requirements

In this section, we present the system requirements that we obtained in our initial phase after discussing with the project supervisors about the scope and scalability of the application. During these discussions, we identified the fundamental elements that will guide the successful development of the application.

To organize more clearly, we will divide the requirements into two parts: the functional requirements, which describe the specific capabilities/interactions that the application must have, and the non-functional requirements, which outline the attributes of performance, reliability, and security of the system.

This elicitation is crucial because it establishes the foundations for the effective development of the application. By fully understanding the project's expectations and constraints from the start, we can optimize the development process, reducing potential rework and ensuring that the application efficiently meets the needs of the end user.

This preliminary understanding also facilitates better communication among stakeholders, ensuring that all involved parties have a clear view of the project's scope and goals. In this way, the requirements gathering process not only informs technical development but also contributes to the overall success of the project.

3.1.1 Functional Requirements

These functional requirements define the specific behaviors and functionalities that the application must possess to meet the needs of its users, which in our case are the farmers and tourists. For farmers, the focus is on managing their farm's resources and creating activities. For tourists, the emphasis is on providing a gamified experience: by discovering fun activities and completing them - to win points and redeem rewards (vouchers for products).

3.1.1.1 Farmer's Functionalities

- **User/Farm Registration**

  — Ability to create an account, including the management of personal information and account details;

- **Manage the farm's resources**

  — The farmer should have the capability to manage the resources of their farms, by adding and editing activities, products, vouchers, animals and crops to the farm;

- **Activity creation**

- — The farmer should be able to create activities for the tourists to take part in, specifying the details and how many points the activity should award when completed. Upon activity creation an unique QR code will be generated.

- **Voucher creation**

- — The farmer should be able to create vouchers, specifying details such as the percentage discount and the products. Additionally, tourists must be able to redeem these vouchers, and use them via a code generated upon redemption;

3.1.1.2 Tourist's Functionalities

- **Activity track**
  - — The app must verify the completion of activities, through the scan of a QR Code associated with the activity, that the farmer will provide;
  - — In the future, the tracking mechanism will also include user localization and time data;

- **Browse/Customize Activities**

  — Search for activities and filter the results based on individual preferences;

- **Voucher redemption**

- — Enable users to search for vouchers associated with specific farms or products, and redeem them using the points they won by completing activities in the farm;

- **User Reviews and Ratings**

  — Give the user the ability to rate an activity and leave a review about what they experienced;

- **Notification System**

  — Provide a notification system to alert users about upcoming trips and promotions;

- **Interactive Map**

  — Display locations with available activities through an interactive map.

3.1.2 Non-Functional Requirements

- **Performance**

  — Ensure a smooth user experience with minimal latency, even under varying network conditions common in rural areas;

- **Scalability**

  - The system should handle an increase in users and visits, with a specific emphasis on supporting a minimum of 40 farms across 4 different zones and approximately 200 users accessing the application simultaneously;
  - Design the architecture with microservices [20] to ensure scalability and easy expansion as the company grows;

- **Security**

  — Implement robust security measures to ensure the secure storage of personal information and payment details;

- **Reliability**

  - Achieve a system uptime of at least 98%, minimizing system downtime and maximizing availability through robust architecture , considering bad internet connectivity in rural areas;
  - Regularly perform automated backups to prevent data loss, ensuring quick recovery in case of unforeseen events;

- **Accessibility**

  - Design an intuitive interface to make the application easy to understand and navigate, considering potential users with varying levels of digital literacy due to some older peoples in rural areas;
  - Ensure that users can find an activity with less than 4 clicks and can scan the QR code within 4-10 seconds;
  - Provide adjustable features, such as a color-blind mode, night mod and interface with adjustable size, to enhance accessibility for users with different needs;

- **Offline Functionality**

  — Develop a robust offline mode, allowing users to access essential features and information even when internet connectivity is limited or unavailable. Implement a mechanism to synchronize the data once the internet connection is reestablished, ensuring a continuous and valuable user experience in rural areas.

3.2 Actors

The application is designed for individuals with an interest in rural tourism across all age groups. Given this large scope of the people involved, the levels of technological knowledge is very different across the board. In that light the needs to be designed to accommodate this diversity. The goal is to ensure a seamless and inclusive interaction with the system, not limiting the users to their previous experience

Prior experience with applications dedicated to the booking of various services would be advantageous, that existing understanding of those systems enhances the user ability to navigate with ease through the features of the application

The main actors will be the following:

- **Tourist:** Represents the tourist role, granting access to the core features of the applications. It allows the user to search and explore the details of the available farms, subsequently viewing the available activities offered on each farm, as well as search for especific hotels and all the farm that are associated with it. The tourist can choose an specific activity and see it's associated rewards, that can be later redeemed by scanning a QR code at the conclusion of the task. He can also give a review on the activities that he has participated and concluded, leaving it available to all that visit the page of the farm

- **Farm supervisor:** Represents the user responsible for creating and managing a farm within the application. This user has the task of inputting all the relevant details, including the range of available activities offered up by the farm. Additionally, they define what are the products available as well as the specific rewards associated with each activity. A supervisor can create a QR code that will be scanned to redeem the rewards by the tourist, they can also create a voucher that can be redeem for products or other perks agreed with the hotel.

- **Tourist supervisor:** Represents the tourist supervisor, who his responsible for facilitating the connection between the farm and the tourist. This role involves managing reservations made by a third-party application, communicating with the farm supervisor regarding the attendance of the tourist at the hotel. Furthermore, the supervisor facilitates the sale of products that the farm is willing to make a deal of.

3.3 Use Cases

Figure 3.1 presents the use case model of the whole system, representing the a basic usage of the application.

The use cases that extend from another can be independent - for instance the "Search for destination" use case is optional and can be used or not. If it is used it adds specific functionalities to the extended case. Despite being distinct roles, the tourist, farm supervisor, and touristic supervisor interact with the same system, each with a different version and functionalities tailored to their respective roles.

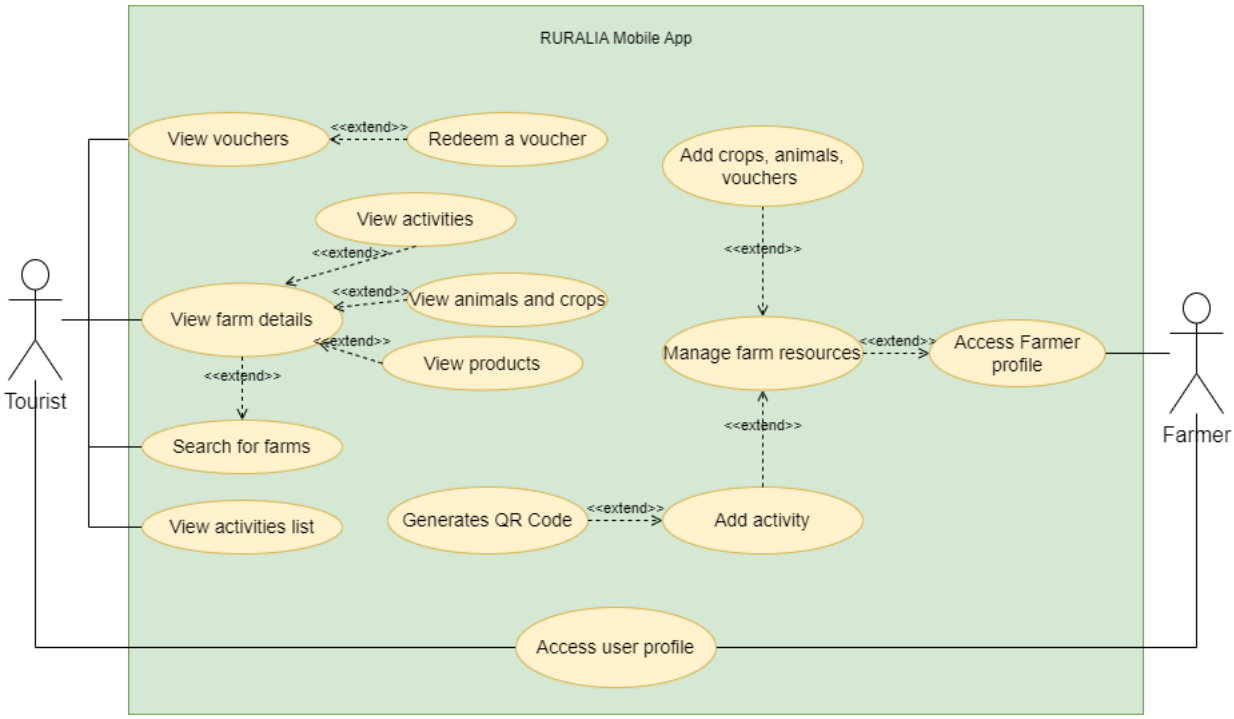

Figure 3.1: Use Case Model

The tourist has access to a home-page that showcases recommended farms and activities according to their tastes, taking into account what places they visited before and activities they've participated in. It will also take into account the user's localization by prioritizing farms that are closer.

The tourist can also search for farms, through a search bar or by accessing an interactive map. This use case extends other two use cases - "View Farm Details" and "View Hotel Details". Upon selecting a farm they will be shown the respective farm's page, where they can view everything the place has to offer: such as the list of crops and animals present in the farm. Knowing this information in advance might encourage them to visit the place. The user can also view all the details about the activities such as description, location and rewards. The farm's contact information is also displayed when the user attempts to make a booking by clicking the 'Book Now' button. This use case is available through the previous one - view farm details, or through the navigation bar present in the application, which takes the user to a list of available tasks.

When the user completes an activity, they can use their phone's camera to scan the QR Code provided by the farmer or the tourist supervisor in charge of the task as soon as it is concluded. This can be done by accessing the QR Code scanning page through the navigation bar. This QR code serves to confirm that the user has completed the task successfully and that the points should be transferred to their account. Each activity has a fixed amount of awarded points, chosen by the farmer upon activity creation.

After the user has participated in some activities and accumulated enough points, they are able to redeem vouchers featuring discounts for farm goodies. As soon as they redeem a voucher, a

code will be shown and the points will be deducted from their account.

The farmer is presented with a different UI from the regular user - the tourist, being able to access their farmer profiles and manage their farm's resources, such as the crops, animals, and products. They can also create activities, by writing a fitting description and choosing how many points they award the users with upon completion. As soon as the activity is created, a prompt will appear showing a QR Code which is exclusive to that one activity - this code can be saved to their phone's gallery if they wish to print it for easier access. The farmer can also create vouchers for the users to redeem, by selecting which products are featured in the voucher, as well as the discount offered on them.

Chapter 4

System Architecture

4.1 Docker Microservices

Microservices is an architectural style in software development where a application built as a group of small, independent and loosely coupled services. Each service is designed to fulfill a specific business function and can be developed, deployed and scaled independently. To manage this services in a scalable in efficient manner, tools like Docker, for containerization, and Kubernets, for orchestration, were used.

All the docker containerized modules present in our Microservices Architecture are listed below, as well as a small description of each one in this project.

- **API Gateway** - entrypoint for clients, routing requests to the authentication module and the RabbitMQ message broker.
- **Authentication** - creation and verification of JWTs on the incoming requests of the clients
- **Database** - relational database to store all the needed informations
- **RabbitMQ** - message brokers connecte to the 4 modules below, each one with is own queues employing a RPC system
- **Farms** - module to handle farm related business logic
- **Activities** - module to handle activities related business logic
- **Marketplace** - module to handle marketplace related business logic
- **User Profile** - module to handle user related related business logic

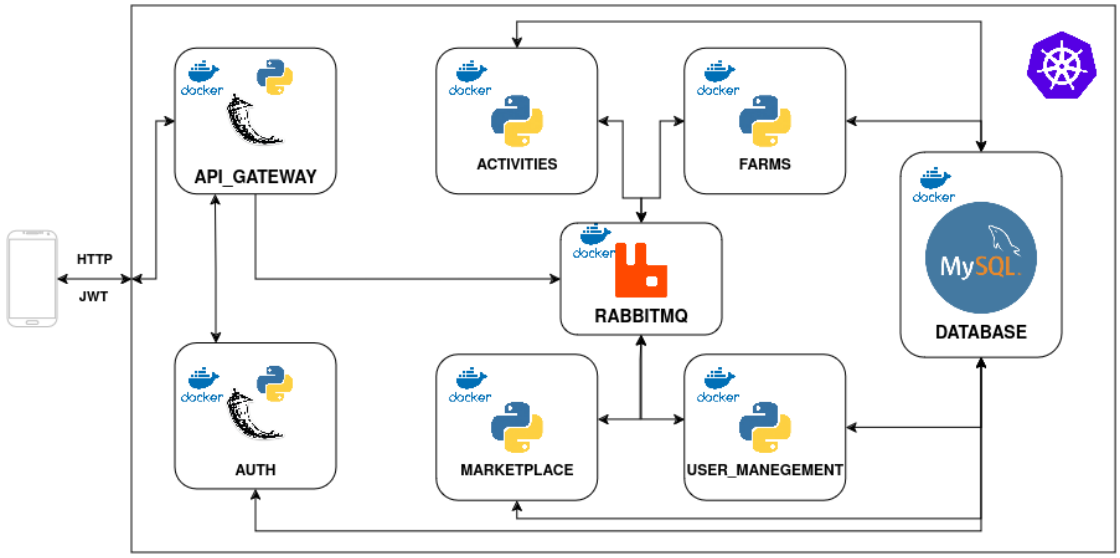

Figure 4.1: Microservices Architecture Overview

4.1.1 API Gateway

The API Gateway is a Flask that handles the requests made by the client, all the requests between the application and the API will need to have a JWT token attached to the headers of the HTTP connection (except the login and register requests). When a request is received, the module creates a RPClient class that handles the connection with RabbitMQ and creates a message that the modules connected with RabbitMQ can process. This model acts on an producer/consumer model, where the API produces the messages that the other modules will consume.

```
1  FROM python:3.10.14
2
3
4  WORKDIR /usr/app
5  WORKDIR /usr/src/app
6
7  COPY requirements.txt ./
8  RUN pip install --no-cache-dir -r requirements.txt
9
10 RUN groupadd -g 999 python && \
11     useradd -r -u 999 -g python python
12
```

```
13 USER 999
14
15 COPY . .
16
17 EXPOSE 5000
18
19 CMD python ./api.py
```

This Dockerfile is constructing on top of the base python 3.10.14. It begins by copying the 'requirements.txt' which lists all the dependencies required by the application, into the container, after that 'pip' command installs all the libraries using the file.

After the installation of the dependencies, a non-root user is created, this security measure helps to mitigate potential risks associated while running processes as the root user, enhancing the overall posture of the application

Next, all the files on the Dockerfile file directory are copied to the container, additionally the port 5000 is exposed, default port used by Flask, to enable the external communication.

Finally the Dockerfile concludes by executing Flask, initializing the application and making it accessible via the exposed port.

The container has 5 environmental variables that are not specified in the Dockerfile:

- **Flask_HOST** : Host of the Flask aplication
- **Flask_PORT** : Port of the Flask aplication (default = 5000)
- **RabbitMQ_USER** : RabbitMQ authentication username
- **RabbitMQ_PASSWORD** : RabbitMQ authentication password
- **RabbitMQ_HOST** : RabbitMQ host name
- **AUTH_SVC_ADDRESS** : Authentication module address

4.1.1.1 Remote Procedure Calls

The architecture of all routes follows a RPC (Remote Procedure Call) logic, which delegates information processing to other microservice modules. This approach ensures modularization and scalability within the system.

To facilitate this RPC communication, the RCPClients class was created. This class serves as a crucial intermediary, managing connections and orchestrating RPC requests. Upon instantiation, the RCPClients class establishes a secure connection to RabbitMQ, using the API's username and password for authentication. This connection acts as a gateway, enabling communication with other microservices.

This class provides predefined set of functions, designed to carry out the to various processing needs of the API. These functions encapsulate common tasks and operations, streamlining the development process and promoting code reusability.

```python
class RpcClient(object):
    """
    Creates a Remote Procedure Call Client object to handle all the messages from
        the API module to the
    desired module.

    Args:
        rpc_queue (str): The name of the queue to send the message to.
    """
    def __init__(self, rpc_queue):
        """
        Initializes the RPC Client.

        Args:
            rpc_queue (str): The name of the queue to send the message to.
        """

    def on_response(self, ch, method, props, body):
        """
        Function to handle the messages received on the queue.

        Args:
            ch (pika.channel.Channel): Channel object.
            method (pika.spec.Basic.Deliver): Method object.
            props (pika.spec.BasicProperties): Properties object.
            body (bytes): Body of the message received with JSON.
        """

    def call(self, message):
        """Function to send a message to the queue

        Args:
            message (dict): Message to be sent to the queue for the desired module

        Returns:
            _type_: Message response from the module
        """

    def close(self):
        """
        Close the RabbitMQ connection.
        """
```

Each module as it's own class, where each class extends the RPCClient class and communicates via different queues, sending messages in a predefined format. The message includes a dictionary where the 'func' field specifies the function of the module to be invoked. The remaining fields in the dictionary contain the necessary arguments for that function. Below is an example

of that structure

```
class MarketplaceRpcClient(RpcClient):

   def redeemVoucher(self, userID, voucherID, code, used):
        message = {'func': 'redeemVoucher',
                   'userID': userID,
                   'voucherID': voucherID,
                   'code': code,
                   'used': used}
        return self.call(message)
```

4.1.2 Authentication Module

The Authentication module is based on a Python-3.10.4 Docker image that is running Flask to handle requests made by the API.

The Dockerfile used to create this module is basically the same as the API. This module has eight environmental variables that are being used:

- **MYSQL_HOST** : MySQL host name
- **MYSQL_PORT** : MySQL port
- **MYSQL_USER** : MySQL username
- **MYSQL_PASSWORD** : MySQL password
- **MYSQL_DATABASE** : MySQL database name
- **Flask_HOST** : Flask host name
- **Flask_PORT** : Flask port
- **JWT_SECRET** : secret key used to sign JSON Web Tokens

There are 3 type of requests the module can receive: login requests, register requests and verify requests.

4.1.2.1 Login

This route logs in a user by verifying their credentials against the database and providing a JWT token upon successful authentication.

It expects the following fields in the request form

– name: The username or email of the user.

– password: The password of the user.

If the verification is successful a JWT token is created containing the username, expiration date, issued time and the user's public id.

4.1.2.2 Register

This route registers a new tourist by adding their information to the database and it expects the following fields in the request form:

- username: The username of the tourist.
- password: The password of the tourist.
- name: The name of the tourist.
- email: The email of the tourist.

4.1.2.3 Verify

This route checks the Authorization header for a JWT, verifies its validity, and decodes it to extract the user information.If the information is valid a tuple with the decoded data and a 200 code is returned.

This route is called in every request made to the API (with the exception of the register and login were the JWT is non existent).

4.1.3 Database

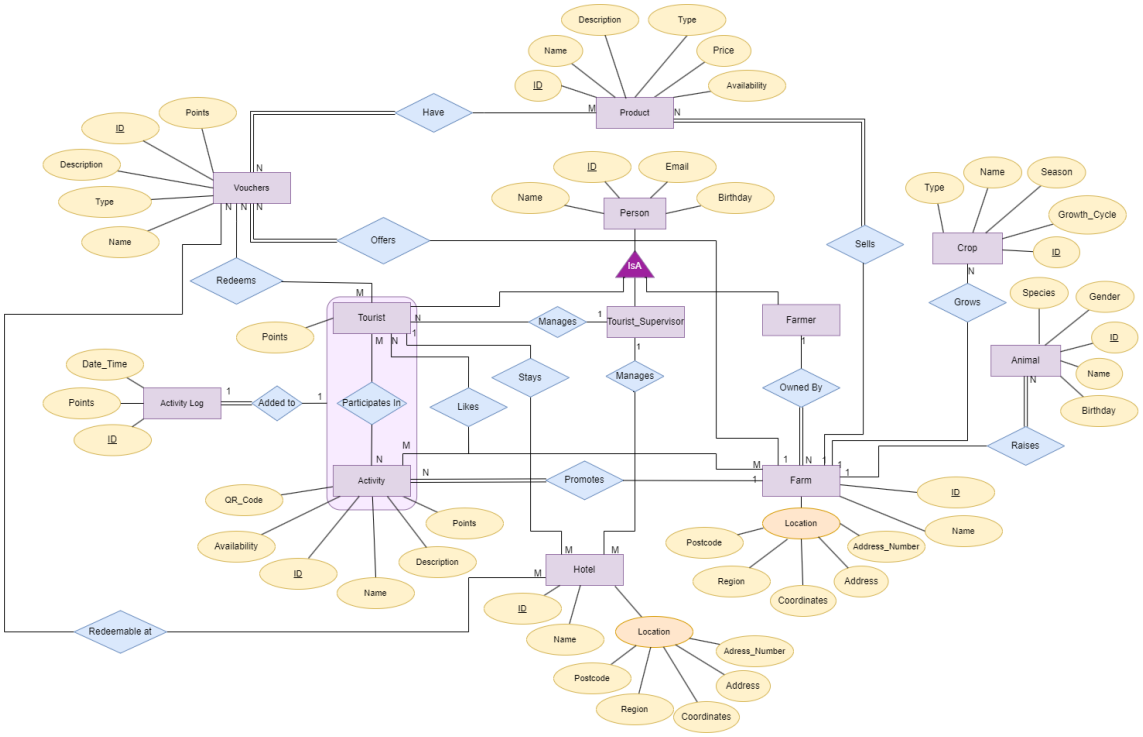

Figure 4.2: DER

The DB have the follow entities:

- **Person** - In project context this are tourists, farm owners and tourist managers.
- **Activity** - Rural Activities, for example pick blueberries.
- **Animals** - Farm Animals.
- **Crop** - Farm crops.
- **Product** - All farms have a set of specific products.
- **Activity Log** - This entity are in charge of store the activities done by tourists.
- **Farm** - Local where the activities are done.
- **Voucher** - A discount product or a discount in one next activity, a decision that will be made by the farm owner.
- **Hotel** - The place where the tourists will stay.

Based on the ERD, the tables that needed to be created and present in the ER were extracted. In addition to those directly representing the entities in the diagram, five more tables: Redeem, Voucher_Product, FavoriteFarms, FavoriteActivities and Redeemable_at had to be created, that are because N to M relationships between entities.

4.1.3.1 MySQL[21]

MySQL is a widely-used open-source relational database management system (RDBMS) known for its reliability, performance, and ease of use.

Within the system, five modules possess access to the MySQL database: Activities, Farms, Marketplace, User Management, and Authentication. These modules, primarily focused on internal operations, each one having it's own credentials, being easy to log and track which module execute some specific task. They form the backbone of the system's backend functionality, handling crucial tasks such as data manipulation and user authentication.

```
1  FROM mysql:8.4.0
2
3  ENV MYSQL_ROOT_PASSWORD= #insert the root password here
4  ENV MYSQL_DATABASE=ruralia_db
5
6  ENV MYSQL_CHARSET=utf8mb4
7  ENV LANG=C.UTF-8
8
9  COPY init.sql /docker-entrypoint-initdb.d/
10
11 EXPOSE 3306
```

This Dockerfile utilizes MySQL version 8.4.0 as its base image. It begins by defining the root password for accessing the MySQL database, ensuring secure access and administration.

Next, the Dockerfile proceeds to specify the name of the database to be created within the MySQL environment.

After that, it configures the character set to ensure full support for UTF-8 characters within the database, facilitating the storage and retrieval of multilingual data.

Subsequently, the Dockerfile copies the 'init.sql' file into the container. This SQL script is instrumental in initializing the database, creating necessary tables, users, and populating initial data.

Finally, the Dockerfile exposes port 3306, the default port used by MySQL for client connections, enabling external applications to communicate with the MySQL database server.

4.1.4 RabbitMQ[25]

RabbitMQ is a widely used open-source message broker that facilitates communication between distributed systems and applications. It implements the Advanced Message Queuing Protocol [2](AMQP), among other messaging protocols. The main points applicable to this project is the reliable delivery of messages using various acknowledgment mechanisms and persistent storage options and scalability.

RabbitMQ is used to carry the messages between the API and the modules, ensuring that messages are not lost and all the requests are fulfilled.

```
1  FROM rabbitmq:3.13-management
2
3  EXPOSE 5672 15672
4
5  COPY definitions.json /etc/rabbitmq/
6  COPY rabbitmq.conf /etc/rabbitmq/
7
8  ENV RABBITMQ_DEFAULT_USER=#string for username
9  ENV RABBITMQ_DEFAULT_PASS=#string for user's password
10
11 RUN chown rabbitmq:rabbitmq /etc/rabbitmq/rabbitmq.conf /etc/rabbitmq/definitions.
       json
12 CMD rabbitmq-server
```

This Dockerfile configures the RabbitMQ image, specifically tagged with version 3.13, and integrates the RabbitMQ Management Plugin. This plugin augments RabbitMQ with an HTTP-based API, enhancing its capabilities for management and monitoring tasks.

The Dockerfile commences by exposing two crucial ports: port 5672, utilized for the Advanced Message Queuing Protocol (AMQP) communication within RabbitMQ, and port 15672, designated for the HTTP API and Management UI, providing convenient access for administration purposes.

Next, two custom configuration files for RabbitMQ is introduced into the container: one is the 'definition.json' this file as a list of users their respective passwords and queues that are automatically loaded when the application starts and 'rabbitmq.conf' this file includes essential settings and configurations to optimize RabbitMQ's behavior. Additionally, the Dockerfile sets the username and password required for accessing the Management UI, ensuring secure access to administrative functionalities.

Permissions are granted to the previously copied files within the container and finally the Dockerfile concludes by executing the RabbitMQ application, initializing the messaging broker and making it ready for usage

Each module as it's unique queue, where he and it's replicas, can consume the messages and execute their business logic. There are 4 queues:

- **activities_rpc_queue** - queue for all the messages from the API module to the Activities module.
- **market_rpc_queue** - queue for all the messages from the API module to the Market module.
- **farms_rpc_queue** - queue for all the messages from the API module to the Farms module.
- **user_rpc_queue** - queue for all the messages from the API module to the User module.

4.1.5 Modules

The remaining modules within the system function as workers, responsible for executing queries in the database. Operating behind the scenes, these modules receive information and perform various processing tasks.

Each of these worker modules adheres to a consistent pattern, using the Pika Python library to establish connections with RabbitMQ. This integration enables them to consume messages from the message queue, following a predefined format established by the RPCClients class.

```
1  FROM python:3.10.14
2
3  WORKDIR /usr/src/app
4
5  COPY requirements.txt ./
6  RUN pip install --no-cache-dir -r requirements.txt
7
8  COPY . .
9
10 RUN groupadd -g 999 python && \
11     useradd -r -u 999 -g python python
12
13 USER 999
14
```

```
15 CMD python ./'#name of the python file'.py
```

This Dockerfile is constructing on top of the base python 3.10.14.It begins by copying the 'requirements.txt' which lists all the dependencies required by the application, into the container, after that 'pip' command installs all the libraries using the file. After the installation of the dependencies, a non-root user is created, this security measure helps to mitigate potential risks associated while running processes as the root user, enhancing the overall posture of the application

There are 7 environmental variables present in all the remaining modules

- **MYSQL_HOST** : MySQL host name
- **MYSQL_PORT** : MySQL port
- **MYSQL_USER** : MySQL username
- **MYSQL_PASSWORD** : MySQL password
- **MYSQL_DATABASE** : MySQL database name
- **RabbitMQ_USER** : RabbitMQ host name
- **RabbitMQ_PASSWORD** : RabbitMQ port

4.1.5.1 Activities Module

These are the function available in this module and what they do

- **getAllActivities** : retrieves all the activities from the database
- **addFavActivity** : adds a favorite activity to a user <- needs

4.1.5.2 Farms Module

These are the function available in this module, what they do and their arguments

- **getAllFarms** : retrieves all the farms from the database
- **getFarmActivities** : retrieves all the activities from a farm <- FarmID
- **getFarmProducts** : retrieves all the products from a farm <- FarmID
- **getFarmerFarms** : retrieves all the farms where the farmer is the owner <- UserID
- **getFarmAnimals** : retrieves all the animals from a farm <- FarmID
- **getFarmCrops** : retrieves all the crops from a farm <- FarmID
- **getFarmSugested** : retrieves suggested farms from the database

- **getFarmDetails** : retrieves all the information from a farm <- FarmID
- **addFavFarm** : adds a favorite farm to a user <- UserID, FarmID
- **addFarmActivity** : adds an activity to a specif farm <- FarmID, name, qrcode, points, description, availability
- **addFarmCrop** : adds a crop to a specif farm <- FarmID, name_crop, type_crop, growth_cycle, season

4.1.5.3 User Management Module

These are the function available in this module and what they do

- **getUserInfo** : get all the user info from the database <- UserID

4.1.5.4 Marketplace Module

These are the function available in this module and what they do

- **getAllProducts** : retrieves all the products from the database
- **getAllVouchers** : retrieves all the vouchers from the database
- **redeemVoucher** : redeems a voucher from the database <-
- **addVouchers** : adds a voucher to the database <-

4.2 Kubernetes [12]

We created a Kubernetes cluster to house our backend. This cluster is composed of four nodes (Virtual Machine in the IT servers), a master node and three worker nodes.

The master node is made up of four main components[13]:

- **API Server** - This is the entry point for the clusters control plane. Administrators and components communicate with the API server to manage the state of the cluster.
- **Controller Manager** - This component has the responsibility of ensuring the desired state is matched by the actual state. It can replicate pods and control nodes and the lifecycles.
- **Scheduler** - The scheduler distributes the workload through the worker nodes, according to factors like resource availability and node affinity rules.
- **ETCD** - This component stores all of the clusters data, including configurations and state information.

Worker nodes are also made up of four main components:

- **kubelet [11]** - This is the primary node agent. It receives pod specifications from the API server and makes sure the containers described in those pod specs are running and healthy.
- **kube-proxy [10]** - The kube-proxy works as a network proxy and load balancer. It maintains network rules on the node, to allow communication to and from the pods inside the node.
- **Container Runtime [4]** - It is the software responsible for running containers. It will pull images from a container registry and manages the container images and storage. In our case, we used Docker Engine as our container runtime.
- **Pods [16]** - This is the simplest Kubernetes object. It can run one or more containers. Each pod is assigned a unique IP address within the cluster and can contain multiple containers that share the same network namespace.

4.2.1 Deployments [14]

For all of the modules in our system architecture, we created a Kubernetes deployment (that will instantiate a number of pods), and a service (that allows communication to pods from that deployment). Deployment and services are created using .yaml files, and here is an example of one of our deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
    spec:
      containers:
        - name: rabbitmq
          image: our image container registry
          imagePullPolicy: Always
          ports:
            - containerPort: 5672
      imagePullSecret:
      - name: dockerconfigjson-github-com
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
```

```
27              - matchExpressions:
28                - key: nodename
29                  operator: In
30                  values:
31                  - node2
```

As you can see in the yaml file above, all the information regarding the creation of the deployment is specfied in the file, from the kind of Component we are creating, the number of replicas needed, and in cases where it is necessary, what sort of nodeAffinity is required.

This will ensure the creation of this deployment in its specific node. In the case of RabbitMQ, nodeAffinity was used so that its deployment will always be created in node2, as specified in the affinity section. This was done by first setting a label for each of the worker nodes specifying a nodename.

The other two cases where we used nodeAffinity were with the MySQL deployment and with the API Gateway deployment. This allows us to make sure these three main components will have enough resources and not compete for resources in case they were created in the same nodes.

4.2.2 Services [18]

Services are used to allow communication inside the cluster, and define rules regarding the type of routing and the specific port numbers used. Here is the service configuration for the RabbitMQ Deployment:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: rabbitmq
5  spec:
6    selector:
7      app: rabbitmq
8    ports:
9      - protocol: TCP
10       port: 5672
11       targetPort: 5672
12   type: LoadBalancer
```

This is a simple configuration and does not require too much complexity, but there are some important things to consider. First, the type of service. In our case, we used the LoadBalancer, since in a scenario where we have multiple replicas of RabbitMQ pods, we would want the traffic to be distributed equally by the service. If we wanted the service to expose this pod outside of the cluster, we would have to configure a NodePort service, that would assign a static port in each node's IP. Services are shared by all nodes of the cluster, unless they are configured in different

namespaces, and these are used when it is necessary to restrain communication throughout the whole cluster.

4.2.3 Secrets [17]

Secrets are important in the configuration of a Kubernetes cluster, because they allow the cluster administrator to hide sensitive information in a more secure way, by storing the secrets in the API server, where they are encrypted at rest, and only accessed by authorized users and pods.

```
apiVersion: v1
kind: Secret
metadata:
  name: rabbitmq-api-pw
type: Opaque
stringData:
  password: #actual password string
```

Since both RabbitMQ and MySQL require a username and password to connect, secrets were created for each module to fulfill these authentication requirements, while storing the data safely.

4.2.4 Volumes

To allow for reliable and stable storage of the database, we added a Persistent Volume [19]. Pods and their storage are ephemeral, meaning their storage and information are lost if the pod dies. PVs allow the pods to retain data, regardless of the lifecycle of the pod. The PV represents a physical storage resource inside the cluster, much like a local disk.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath
    path: # path to store db
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
```

```
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

This is a basic configuration of the PV. There is also a configuration of a Persistent Volume Claim, which serves as a connector from the pod that needs to use the PV to the PV itself. After creating these configurations in the cluster, the deployment that will require the PV will have a specific mention of the PVC on its configuration file, which will go inside the spec configuration of the deployment.

```
    volumes:
    - name: mysql-data
      persistentVolumeClaim:
        claimName: mysql-pvc
```

4.2.5 Ingress [15]

To allow for outside communication coming in, since the goal is having the application backend reachable for any user in any farm, a few steps were taken. First, adding a Ingress controller. This Ingress controller routes external HTTP/HTTPS traffic to services in a Kubernetes cluster based on Ingress resources. There are multiple options for Ingress Controller such as Traefik, HAProxy and the one we used, NGINX. This tool was easy to set up, and does the job well. The Ingress resource will be another configuration in a YAML file, that set the rules for the routing the controller will need to do.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  namespace: default
spec:
  rules:
  - host: agroturgame.av.it.pt
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: api
            port:
```

```
17                number: 5000
```

This simple configuration will redirect traffic labeled to host `agroturgame.av.it.pt` to the API Gateway service, which will then route it to one of the pods of the API Gateway deployment. Currently, no requests outside the IT local network are being attended, since the application is not deployed. We did this to be able to test the application locally and verify the correct routing and functionality of the different APIs before deploying to production.

By testing the application locally within the IT network, we make sure that all components are working correctly. Once we have verified that the application behaves as expected, we can then proceed to deploy it for external access, thus reducing the risk of encountering issues in a live environment.

Chapter 5

Development and Implementation

5.1 Previous Story Board

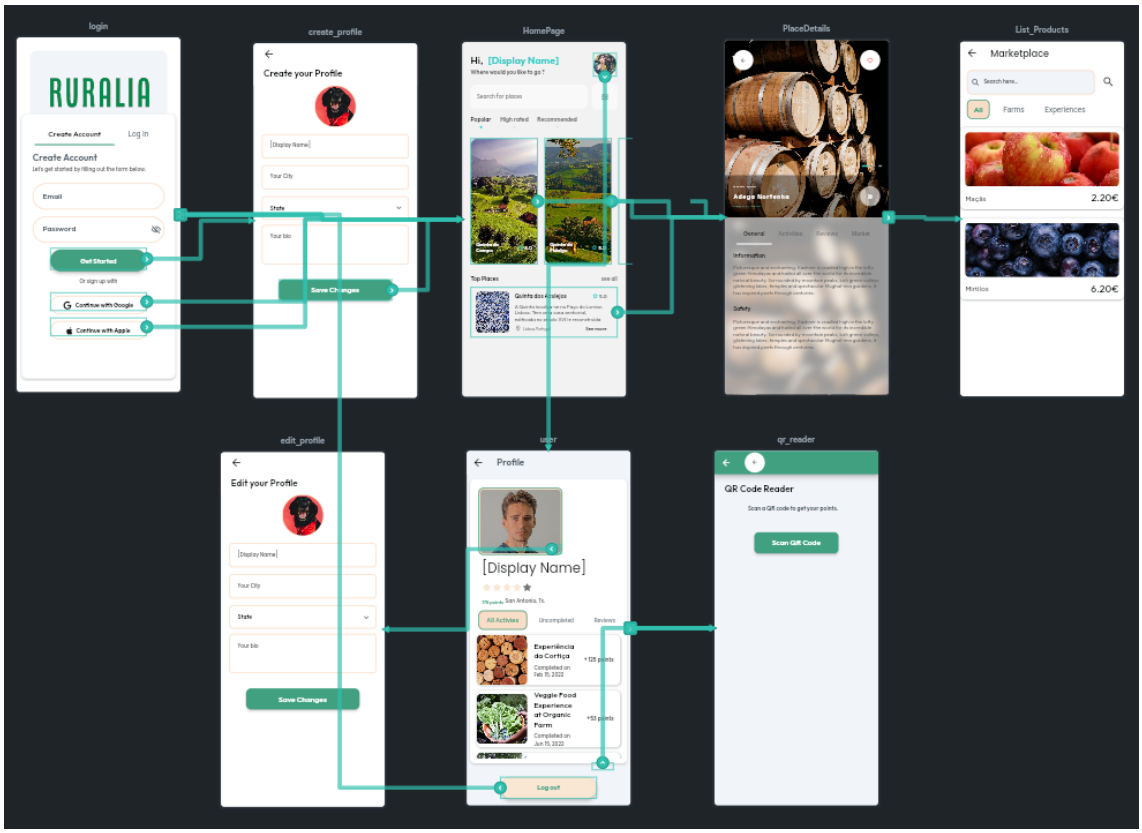

Figure 5.1: Mock up Storyboard

The initial application prototype and design mock up served as a valuable foundation for planning the database integration phase. Furthermore, the identification of an implementation issue during this phase allowed for a more cohesive setup. While the initial deployment of the mock up encountered challenges, it provided crucial insights that ultimately informed the decision to redevelop the application from scratch.

5.2 Development

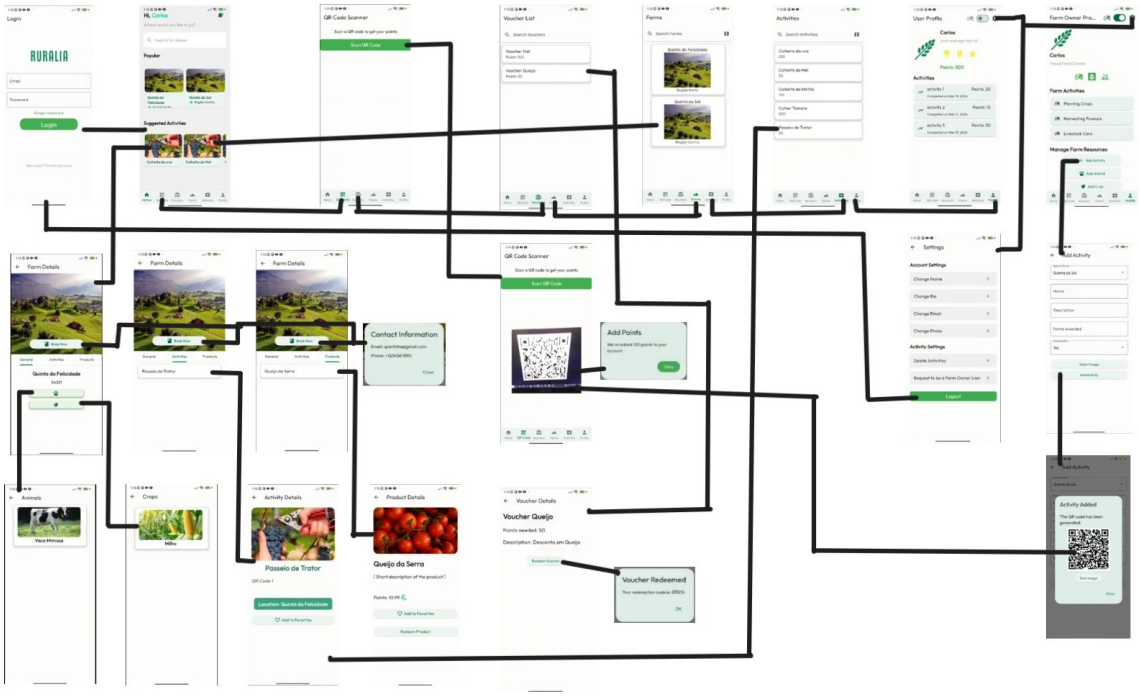

Figure 5.2: App Storyboard

Building upon the initial design mock up, deployment revealed several challenges. Consequently, the decision was made to develop the application from the ground up. User interaction was prioritized, and the previously presented storyboard was incorporated with additional refinements. Core functionalities were largely preserved, allowing users to interact with activities, farms, and scan QR codes associated with experiences, thereby earning points. These points can be redeemed for products defined by farmers or tourism representatives.

For Farm Users, a dedicated profile switch mechanism was implemented using a simple slider. This ensures access to the Farm User view only for authorized users. Additionally, Farm Users can toggle between profiles, granting them the ability to experience the application from a regular User perspective. This approach fosters a more cohesive design and strengthens the connection

between the application and both user groups.

Chapter 6

Future Work

6.1 Future Work

In this section we will talk about the future work - features that are missing but we would like to implement on our application at some point in the future. Most of these features are related to gamification and are meant to help provide an even more fun experience to the tourists. We've also taken inspiration from existing platforms or games that have features that users appreciate, in order to enhance our users' experience.

- **Introduce a new kind of reward - achievements:** many games feature achievements, badges or trophies that the users can strive for and collect. These rewards could be given when the user completes a set number of activities, when they spend a certain amount of points, or visits different places... The possibilities are endless. Their achievements would be showcased on the user's profile as a proof of their mastery. A good example of this achievement system is the browser game Pokerogue [23], a fan-made rogue-like Pokémon game which awards users with achievements when they have completed a certain task. The player can then access their achievements tab and see the fruits of their efforts!

Figure 6.1: Pokerogue's achievements tab

An extremely popular gaming platform that features this kind of reward is Steam [26] - users can earn achievements when playing games, and showcase them on their profile. Many users strive for 100% completion of achievements on games they enjoy. As proven by Steam, badges and achievements are very popular among users who enjoy completing goals, collecting things or simply like to show off their experience!

Figure 6.2: Steam Achievement Progress

- **Implement reviews and farm / activity rating:** reviews often help users decide if they wish to visit certain place or not, or help them know what to expect of such place. By allowing users review farms and activities, we hope to provide a better user experience as well as a tool for farmers to know if there is room for improvement and how their tourists feel about the place. Google Maps [9] is an example of an application that uses the review system, as users are allowed to leave a review and a rating ranging from zero to five stars on the places they visit.

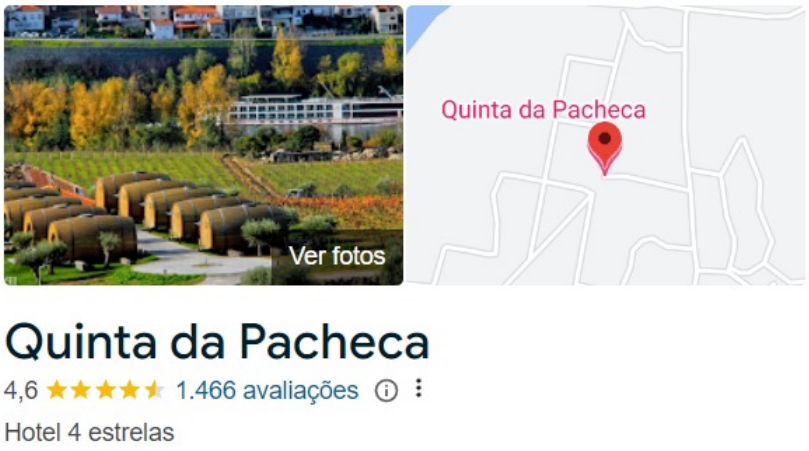

Figure 6.3: Google Map Reviews

- **Introduce post-activity insights:** when the user is done with their vacation and leaves the farm, we don't want them to forget about RURALIA or the place they visited! So, we would

like to give them insights on what's happening on the farm even after they've left : have the crops grown? How are the animals doing? Are there any new activities and products? These are all topics that we would like to explore using insights, that could be given to the user though push notifications. Perhaps knowing that the farm they visited has a new activity could be an incentive to book another vacation there! Or they're lying on their couch and feel like knowing how the cute calf they took care of is doing, all they have to do is open the application and check the farm's page for insights on the animals.

- **Allow users to earn XP and level-up:** each user would start at level 1 and could fill a bar with experience points as they complete activities or visit places. Completing an activity would grant a set amount of experience and the user would level up after filling the bar with the required amount of XP. A large amount of games feature an experience point and leveling up system similar to this one, such as Pokémon Go [22] - a game where the user can walk in real life and catch Pokémon in the game, gaining experience points with each catch, and leveling up as they accumulate enough experience. This system provides continuous motivation for users to engage in more activities and explore new places!

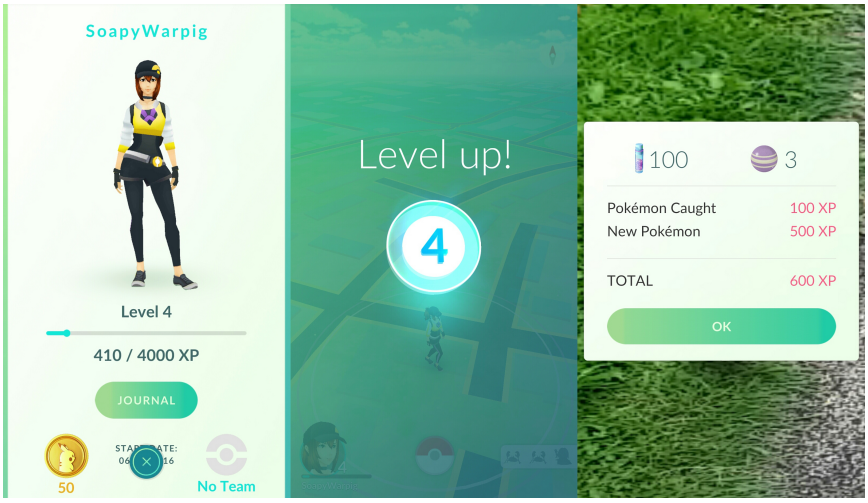

Figure 6.4: Pokemon Go - Player profile, leveling up and a successful catch

- **Introduce a ranking leaderboard:** in the future, we would like to implement a ranking leaderboard to display the relative performance of players, providing a competitive element to our application, by showing who has achieved the highest level, completed the largest number of activities, or acquired most achievements - this aligns well with our other future work goals (achievements and level-up system). It should motivate players to engage in more activities in order to climb higher on the leaderboard. An example of a similar system belongs to a popular mobile game, Candy Crush Saga [3], which features a leaderboard consisting of the player's highest scores, as shown on the figure below.

Figure 6.5: Candy Crush Saga's high score leaderboard

Chapter 7

Conclusion

This project culminates in the creation of RURALIA, a mobile application designed to revitalize rural tourism in Portugal through the power of gamification. Recognizing the inherent beauty and unique character of Portugal's rural landscapes, RURALIA seeks to redirect visitor focus away from the well-trodden tourist path and towards these lesser-known gems.

RURALIA distinguishes itself from existing tourism applications by carving a niche within the broader market. By leveraging gamification principles, the app transforms rural exploration into an engaging and interactive experience. Users participate in enjoyable activities specific to the region, fostering deeper connections with local businesses and communities. This immersive experience transcends mere sightseeing, promoting a more meaningful exchange between visitors and the rural landscape.

The positive ramifications of RURALIA extend beyond the realm of user experience. The gamified approach holds immense potential to diversify Portugal's tourism offerings, steering visitors away from over-saturated urban centers and towards the economic revitalization of rural communities. By strengthening the economic fabric of these regions, RURALIA contributes to a more balanced and sustainable tourism model for Portugal.

On a personal note, embarking on this project has been an enriching and intellectually stimulating experience. We delved into uncharted technological territory, acquiring new skillsets in mobile application development. The knowledge gained throughout this process will undoubtedly serve us well in future endeavors.

In conclusion, RURALIA represents not only a novel mobile application but also a potential catalyst for positive change within the Portuguese tourism industry. By harnessing the power of gamification, RURALIA paves the way for a more immersive, sustainable, and economically diverse tourism experience that celebrates the unique beauty of Portugal's rural landscape.

Bibliography

[1] *Agroturismo. Quando fazer turismo é fazer agricultura.* 2023. URL: https://www.forum.pt/ambiente/agroturismo-quando-fazer-turismo-e-fazer-agricultura.

[2] *AMQP*. URL: https://www.rabbitmq.com/tutorials/amqp-concepts.

[3] *Candy Crush Saga*. URL: https://www.king.com/game/candycrush.

[4] *Container Runtime*. https://kubernetes.io/docs/setup/production-environment/container-runtimes/.

[5] *Docker: Zero to Hero*. https://www.youtube.com/watch?v=3c-iBn73dDE.

[6] *Explorial - Get to know a city. Solve Puzzles. Have fun!* 2023. URL: https://explorial.com/pt-pt/.

[7] *Framie App*. 2022. URL: https://www.facebook.com/framieapp/.

[8] *Game Reward Systems: Gaming Experiences and Social Meanings*. 2012. URL: https://www.researchgate.net/publication/268351726_Game_Reward_Systems_Gaming_Experiences_and_Social_Meanings.

[9] *Google Maps*. URL: https://www.google.pt/maps/.

[10] *Kube-proxy*. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/.

[11] *Kubelet*. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.

[12] *Kubernetes*. https://kubernetes.io/docs/concepts/.

[13] *Kubernetes Components*. https://kubernetes.io/docs/concepts/overview/components/.

[14] *Kubernetes Deployments*. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[15] *Kubernetes Ingress*. https://kubernetes.io/docs/concepts/services-networking/ingress/.

[16] *Kubernetes Pods*. https://kubernetes.io/docs/concepts/workloads/pods/.

[17] *Kubernetes Secrets*. https://kubernetes.io/docs/concepts/configuration/secret/.

[18] *Kubernetes Services*. https://kubernetes.io/docs/concepts/services-networking/service/.

[19] *Kubernetes Volumes*. https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

[20] *Microservices*. https://www.youtube.com/watch?v=lL_j7ilk7rc&ab_channel=5MinutesorLess.

[21] *MySQL*. URL: https://dev.mysql.com/doc/refman/8.0/en/introduction.html.

[22] *Pokemon Go*. URL: https://pokemongolive.com/.

[23] *Pokerogue*. URL: https://github.com/pagefaultgames/pokerogue.

[24] *Portugal Play*. 2022. URL: https://portugalplay.com/.

[25] *RabbitMQ*. URL: https://www.rabbitmq.com/docs.

[26] *Steam*. URL: https://store.steampowered.com/.