Albert Khaidarov
Enis Nuredini
Nedim Šišić

Web App for Private Accommodation Reservations

Project Seminar

Technical Documentation

University of Primorska

Faculty of Mathematics, Natural Sciences and Information Technologies

July 2022

Contents

1. Introduction

This document contains the technical documentation for the Web App for Private Accommodation Reservations (Spletna aplikacija za rezervacije zasebnih nastanitev) Project Seminar project. The document gives a description of the application problem, the technical requirements of the application, the architecture of the system, and the software choices.

2. Problem description

The Web App for Private Accommodation Reservations is a web application that hosts an online marketplace for lodging, primarily for offering and renting of vacation homestays. In recent decades, partially due to the widespread use of internet and rise of social media, the homestay lodging model has become very popular among the general public – and has caused concern in the hotel industry. The Web App for Private Accommodation Reservations offers registered users the opportunity to both host and rent residences for monetary compensation. The users are able to register and customize their profiles, to create and display their residence offers, and to rent the offers of other users. Users search for offers by location, by specifying constraints regarding the timeframe, cost, and amenities, and are able to use a map. The application uses a database for storing user profiles, residences, and reservations, provides users with an intuitive user interface, and implements the logic for data manipulation, routing user requests, and implementation of different use cases.

3. Technology Stack choices

3.1 Web Framework

Web App for Private Accommodation Reservations is implemented in Ruby on Rails. Ruby on Rails is a server-side web application framework written in Ruby. The framework provides rapid development and prototyping of applications by enforcing "Convention over Configuration" (COC), providing scaffolding of system components, and by offering a variety of widely used *gems* (Rails libraries). As such, Ruby on Rails was used to build the entire application "from scratch".

Ruby on Rails is a model–view–controller (MVC) framework, where "model" denotes data logic, "view" denotes the user interface (UI), and "controller" denotes the web service and the connection between the data and the UI. Web App for Private Accommodation Reservations uses the Devise gem, a library for user registration, authentication, and profile management. Leaflet, an open-source JavaScript library for interactive maps, is used for displaying and manipulating the map.

3.2 Database

The application uses the SQLite database engine. SQLite is a database library written in C, and is one of the most widely deployed database engines. It is not a client-server engine; rather, the database is linked to the program and does not require service management or access control. Because SQLite is also the default database engine for Ruby on Rails, it serves as a good choice for the application.

3.3 User interface

Like most applications, Web App for Private Accommodation Reservations uses HTML, CSS, and JavaScript as its front-end tech stack. The jQuery JavaScript library is used to simplify HTML DOM tree traversal and manipulation. The application uses Bootstrap, a popular front-end framework, to provide a grid-based layout and basic style definitions by its CSS classes. Ajax is used to make the application asynchronous, thus enabling client-server communication without the need to reload web pages. And emphasis is placed on intuitive and elegant design.

Figure 1. Some of the technologies used

4. System specification

A user is able to perform the following:

- Create an account
- Login using a password
- Search for residences, and
    - specify location, timeframe, cost, and amenities constraints
    - use a map
- Book residences
- Create and display residences
- Leave reviews

The basic functionalities are presented by a use case diagram in Figure 2. An activity diagram of making a reservation is given in Figure 3. The user is modeled by the User class, which contains the basic data of a user. A single user may act both as a host and as a renter at any given time; a user is not able to rent one of their own rooms. The Room class contains a user ID as foreign key, a name/title, and address and coordinates, Boolean indicators for amenities such as TV, AC, and internet, and the nightly price. A single room may contain multiple photos, which are kept in the Photo class. There is a many-to-many reservation relationship between users and rooms (of other users), implemented by the Reservations class. A class diagram is given in Figure 4.

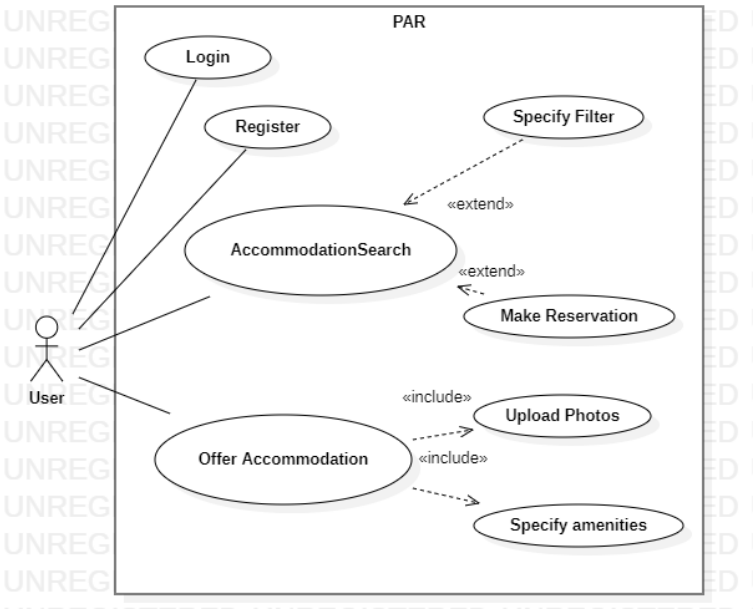

Figure 2. Use Case Diagram

The code for the models (classes) is kept in the app/models folder, containing the photo.rb, reservation.rb, review.rb, room.rb, and user.rb files. Each file contains code for setting the rules for validating data consistency of its model, and for specifying the relations among the classes. The config/routes.rb contains the *routes*, which specify the name of application actions and the urls for running those actions. The links that use those urls and action names are located in the views files. The app/views folder contains multiple subfolders, all of which contain html.erb files for a specific page of the app. The html.erb format allows the use of embedded Ruby code in an HTML document. The devise folder contains the views for user registration, authentication, and profile management; most of the files in the folder were made by the Devise library itself. The view in the app/views/layouts folder is a template that is used by all other views of the app. Similarly, the app/views/shared folder contains files that provide the navigation bar and message and error notifications for all views of the application. All other folders in the app/views folder contain html.erb files that are automatically loaded by the associated actions specified in the controller files (and whose routes are specified in config/routes.rb). Because Ruby on Rails uses Convention over Configuration, the views are loaded automatically when they are named according to Rails

conventions. The code in the views specifies class and id labels for HTML elements, while the CSS styles are defined in assets/stylesheets/application.scss

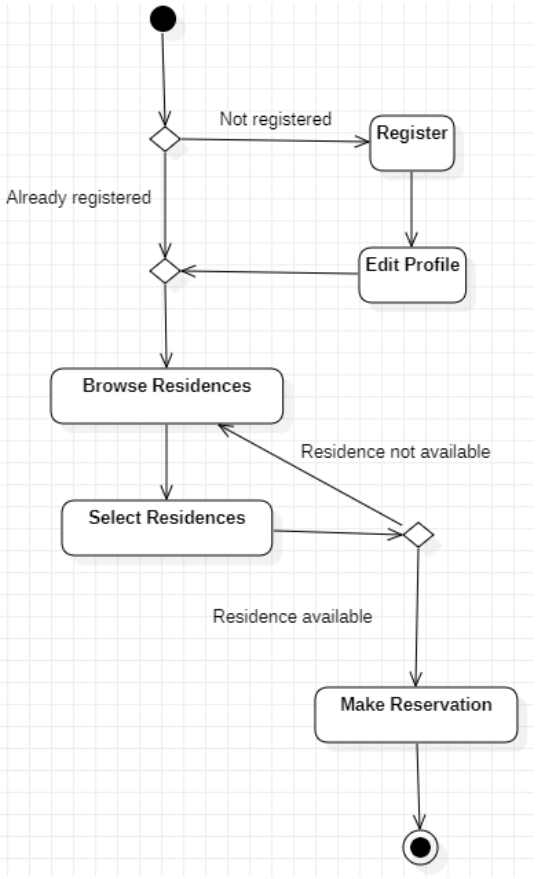

Figure 3. Activity diagram

The controller files in app/controllers contain code that is executed when actions are called. The application controller is the “default” controller, which specifies instructions that are executed before every action in the application. The registrations controller was created by Devise. All other controllers contain code for CRUD operations and follow the Rails naming conventions which allow them to be connected to the views automatically. Additional logic that is implemented in the views using Ajax requests, so the webpages do not need to reload when the requests are made. The structure of application code is shown in Figure 5.

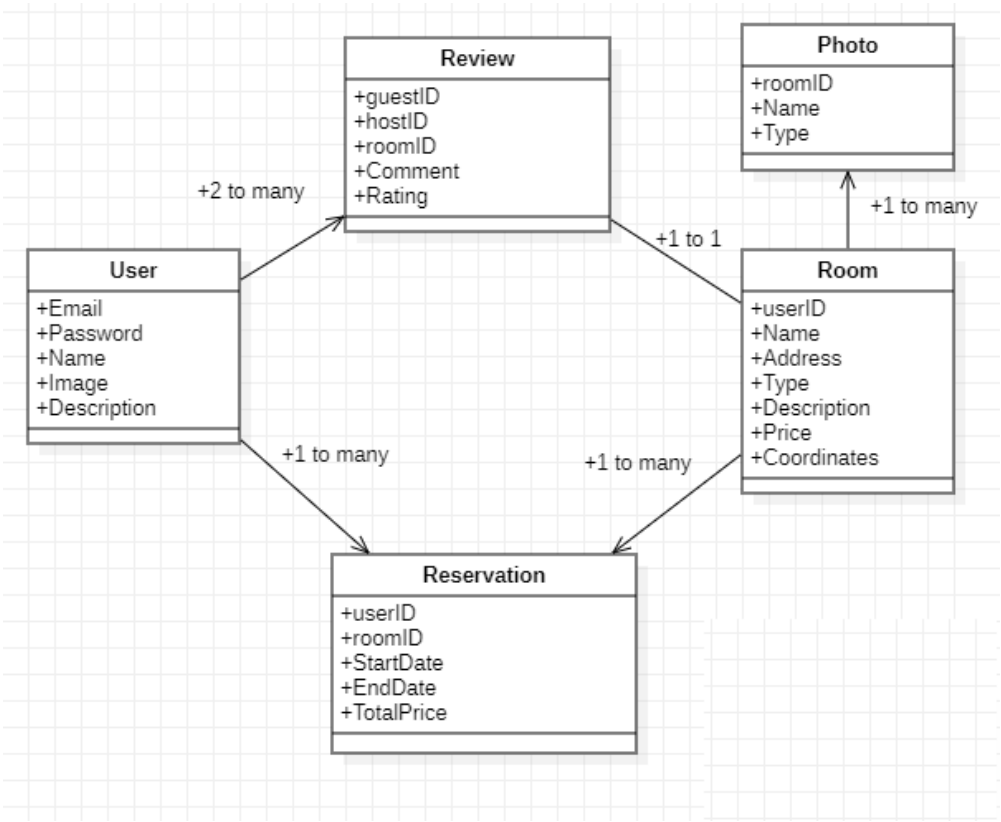

Figure 4. Class Diagram

5. Task division

The different tasks in developing the app were split between the team members as follows:

Albert:

- Requirements analysis
- Data design
- Creating the database
- Creating the User migration and model
- Hashing the passwords
- Creating the Residence migration and model
- Creating the Reservation migration and model
- Connecting the residence locations with the map
- Creating the Review migration and model
- Connecting each model to its controllers

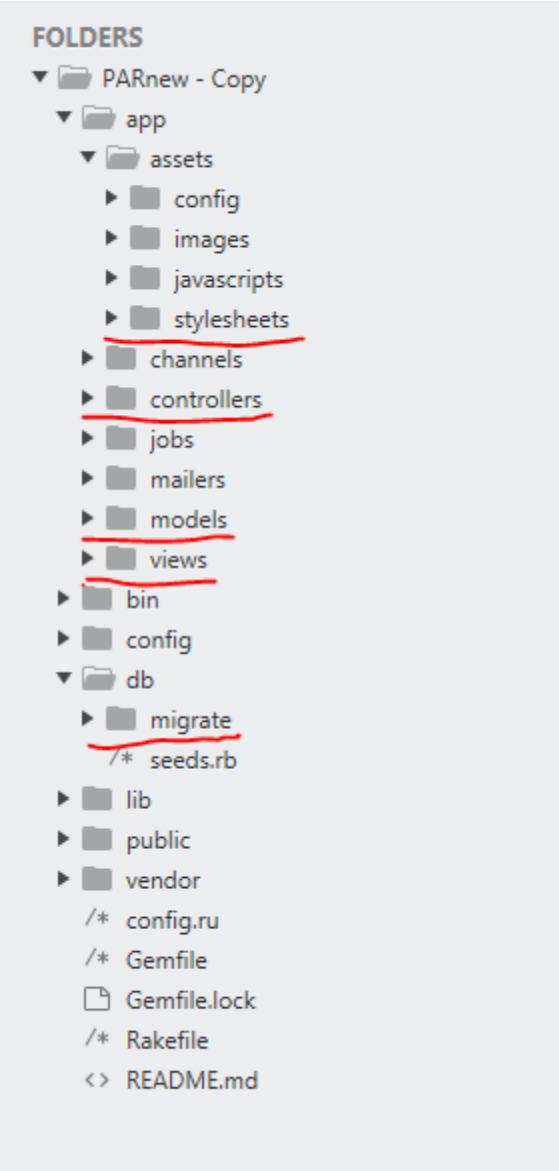

Figure 5. Code structure

Enis:

- Requirements analysis
- Designing the layout of the application's views:
    - User registration and login pages
    - User profile pages
    - Hosting a residence pages
    - Viewing a residence page
    - Home page
    - Search page
- Designing the CSS classes for styling all of the pages
- Writing JavaScript code for dynamic page manipulation

Nedim:

- Requirements analysis
- Creating the structure of the application
- Creating pages
- Creating routes and links between pages
- Configuring gems and dependencies
- Creating the CRUD operations
- Creating the User controller - user authentication and maintaining sessions
- Creating the Residence controller - creating, hosting, and updating the residences
- Creating the Photos controller - uploading and deleting photos
- Implementing search by location
- Implementing search filters
- Creating the Reservations controller - making reservations
- Creating the Reviews controller - creating reviews
- Connecting the models, views, and controllers
- Connecting pages with the map
- Creating AJAX methods for client-server communication without page reloads
- Writing the documentation