HOSTEL MESS MANAGEMENT

By:
Name: Tanzeel Khan
Course: B.Tech
Branch: AI&DS
University Roll Number: 2017677

[Tanzeel Khan]

Hostel Mess Management

Problem Statement

Create a Hostel Management Web Application using a technology of your choice. The application should provide proper functionalities for signing-up and signing-in. It should be connected to a database.

Motivation

When the bookings got open for hostels in my second year, I booked a room first thing in the morning, but I was not assigned the same room I chose. This happened because the ‘first come first serve’ policy of my hostel was implemented manually which gave room for errors. Therefore, to remove the manual part of this policy, I designed a web-application that only lets us choose room from currently available rooms.

Introduction

Web development is a field with several ways to reach the same outcome. We have various technologies like ReactJS, NodeJS, ExpressJS, WAMP, LAMP. I have adopted one such technology called ‘Ruby on Rails’. It is helpful in making light-weight web applications with easy communication with database and easy front to back-end integration.

Objective

The main goal of this project is to make a fast and efficient hostel managing web-application that allows students to sign-in and book a room to live in a hostel or book a meal plan if they don’t want to live in the hostel but want to enjoy food from the hostel mess.

Technology Used

This project is made with the help of a programming language called `Ruby`. It is an open-source object-oriented scripting language invented in the mid-90s by Yukihiro Matsumoto. It provides a framework called `Ruby on Rails`. This framework helps in easy development of applications. It has already written Ruby code for things like file handling, communication, database connections and more. It takes care of all the tedious tasks and follows DRY (Don’t Repeat Yourself) principle.

Functionalities

In the web-application that I have created, users have two roles, namely: 'student' and 'admin'. While a normal sign-up will give us the role of a 'student', we can be promoted to 'admin' by any other admin.

Functionalities of a Student:

- Booking a room: Students can book a room of their choice. The rooms are allotted on 'first come first serve' basis. Rooms can be distinguished on the number of roommates, availability of air-conditioner, on which floor it is, and in which hostel it is.
- Booking a meal plan: Students who do not wish to live in the hostels but want to eat in the mess for some reason can purchase meal plans. They can choose mealtimes (breakfast, lunch, or dinner) and if they want a tiffin service or not. They will be charged according to the purchased plan specifications.

Functionalities of an Admin:

- Manage Rooms: They admin can see which student has booked what room and may cancel the booking if they wish so.
- Manage Meal Plans: They admin can see which student has purchased what meal plan and may cancel the plan if they wish so.
- Add new hostels: The admins can add new hostels in the database. As soon as a new hostel is added, three room of that hostel with different specifications will be created. More rooms must be manually created by the admin as per the hostel.
- Add a new admin: A current admin can look at the list of all the users and promote their role to admin.
- Manage raw materials: The admin can check current status of each raw material, add new items, and purchase more of existing items.

Security of this application

Following are the security features of this web-application:

- BCrypt password storage: Before storing any user password, they are converted into a one-way hash key, which is unique for every possible combination of letters, numbers or special characters.
- Cross Site Request Forgery disabled: Every form in Rails has a unique authenticity token which enables forms to be submitted only via a Rails page/website. This completely reduces the chances of any cross-site forgeries.
- 'has_many" and 'belongs_to': Rails provides us with these two functions with the help of which the currently logged-in user will only be able to edit, update or delete the data which solely belongs to them.

- Schema.rb and database migrations: Rails maintains a version control system for our database. If we add a new table, delete a row, add a row etcetera, all of this needs to be migrated on Rails just like pushing our code on GitHub.

About the database used

In this web-application, I have used PostgreSQL. It is an advanced, enterprise class open-source relational database that supports both SQL (relational) and JSON (non-relational) querying. It is a highly stable database management system, backed by more than 20 years of community development which has contributed to its high levels of resilience, integrity, and correctness. PostgreSQL is used as the primary data store or data warehouse for many web, mobile, geospatial, and analytics applications.

Some features of this database are:

1. It has rich extensions.
2. It is reliable.
3. It is open-source.
4. It is a general purpose OLTP Database.
5. It supports Geospatial Database.
6. It supports federated hub database.

What is MVC pattern in Rails?

MVC stands for Model, View, Controller. To explain this, suppose you have one utility in your web-application called users. Now, to handle this, we will create a model named ‘user.rb’ [.rb is the extension for Ruby files], in it we will define a class which will have methods that can be applied to each and every object or user. All the piece of Graphical User Interface which is related to the utility [example: signup page, change password page] will be placed in a folder called ‘users’. Each template will have a unique named, for example the template for ‘sign-up’ page can be called ‘signup.html.erb’.

Let's have a look at the database schema

```ruby
ActiveRecord::Schema[7.0].define(version: 2022_07_13_103401) do
  enable_extension "plpgsql"
  create_table "hostel_rooms", force: :cascade do |t|
    t.string "studentname"
    t.string "hostelname"
    t.bigint "seater"
    t.boolean "ac"
    t.bigint "number"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.boolean "available"
    t.bigint "price"
    t.bigint "occupants"
  end
  create_table "hostels", force: :cascade do |t|
    t.string "name"
    t.bigint "floors"
    t.bigint "four_seater"
    t.bigint "three_seater"
    t.bigint "two_seater"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "for"
  end
  create_table "meals", force: :cascade do |t|
    t.boolean "breakfast"
    t.boolean "lunch"
    t.boolean "dinner"
    t.boolean "tiffin"
    t.bigint "price"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end
  create_table "users", force: :cascade do |t|
    t.string "email"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "first_name"
    t.string "password_digest"
    t.string "last_name"
    t.string "gender"
    t.bigint "room_id"
    t.bigint "meal_id"
    t.string "role"
  end
end
```

Here, we have following 4 tables:

1. hostel_rooms: It has 8 columns:
    a. studentname: Name of the student who has booked this room. (type: string)
    b. hostelname: Name of the hostel to which this room belongs. (type: strinng)
    c. seater: Number of students the room can accommodate. (type: bigint)
    d. ac: If air-conditioner is available in that room. (type: boolean)
    e. Number: Room number. (type: bigint)
    f. available: If this room is available for booking. (type: boolean)
    g. occupants: Number of students currently living in this room. (type: bigint)
2. hostels: It has 6 columns:
    a. name: Name of the hostel. (type: string)
    b. floors: Number of floors in the hostel. (type: bigint)
    c. four_seater: Number of rooms with four seats. (type: bigint)
    d. three_seater: Number of rooms with three seats. (type: bigint)

    e. two_seater: Number of rooms with two seats. (type: bigint)
    f. for: If this hostel is for boys or girls. (type: string)
3. meals: It has 5 columns:
    a. breakfast: If the student has added breakfast in their meal plan. (type: boolean)
    b. lunch: If the student has added lunch in their meal plan. (type: boolean)
    c. dinner: If the student has added dinner in their meal plan. (type: boolean)
    d. tiffin: If the student has added tiffin service in their meal plan. (type: boolean)
    e. price: Monthly price for the plan chosed. (type: bigint)
4. users: It has 8 columns:
    a. first_name: First name of the student. (type: string)
    b. last_name: Last name of the student. (type: string)
    c. email: Emailof the student. (type: string)
    d. password_digest: Encrypted password of this account. (type: string)
    e. gender: Gender of the student. (type: string)
    f. role: If the user is a student or an admin. (type: string)
    g. room_id: Room ID of the booked room of the student. (type: bigint)
    h. meal_id: Meal ID of the booked room of the student. (type: string)

Check out the code on my GitHub: tanzeyl/hostel-management (github.com)