Modern Application Development II

Project Statement

Vehicle Parking App - V2

It is a multi-user app (one requires an administrator and other users) that manages different parking lots, parking spots and parked vehicles. **Assume that this parking app is for 4-wheeler parking.**

Frameworks to be used

These are the mandatory frameworks on which the project has to be built.

- Flask for API
- VueJS for UI
- VueJS Advanced with CLI (only if required, not necessary)
- Jinja2 templates if using CDN **only for entry point** (Not to be used for UI)
- Bootstrap for HTML generation and styling (No other CSS framework is allowed)
- SQLite for database (No other database is permitted)
- Redis for caching
- Redis and Celery for batch jobs

Note: All demos should be possible on your local machine.

Roles

The platform will have **two** roles:

1. **Admin - root access** - It is the superuser of the app and requires no registration
    - Admin is also known as the **superuser**
    - There can be only one admin for the app
    - Admin can create a new parking lot

decrease the number of parking spots inside the lot.

2. **User - Can reserve a parking space**
   - Register/Login
   - Choose an available parking lot
   - Book the spot (automatically allotted by the app after booking)
   - Release or vacate the spot

Terminologies

User: The user will register/login and reserve any parking spot.

Admin: The superuser with full control over other users and data. No registration is required, i.e. the admin should exist whenever the database is created.

Parking lot: It's the physical space where the collection of parking spots are available for an area. The parking lot may contain the following attributes.

1. id - primary key
2. prime_location_name
3. Price
4. Address
5. Pin code
6. number_of_spots
7. etc: Additional fields (if any)

Parking spot: The physical space for parking a 4-wheeler parking. The parking spot may contain the following attributes.

1. id - primary key
2. lot_id (foreign key-parking lot)
3. status(O-occupied/A-available)
4. etc: Additional fields (if any)

Reserve parking spot: Allocates parking spot as per the user requests.

4. Parking_timestamp
5. Leaving_timestamp
6. parking_cost
7. etc: Additional fields (if any)

Note: The above tables and fields are not exhaustive, students can add more tables and fields as per their requirements

Similar apps

https://www.secureparking.co.in/

Application Wireframe

Vehicle Parking App

Note:
The provided wireframe is intended **only to illustrate the application's flow** and demonstrate what should appear when a user navigates between pages.

- **Replication of the exact views is NOT mandatory.**
- Students are encouraged to work on their front-end ideas and designs while maintaining the application's intended functionality and flow.

Core Functionalities

1. Admin login and User login
   - A login/register form with fields like username, password etc. for the user and a login form for the admin
   - The application should have only one admin identified by its role
   - You can either use **Flask security (session or token)** or **JWT based Token** based authentication to implement role-based access control
   - The app must have a suitable model to store and differentiate all types of users
2. Admin Dashboard - for the Admin
   - The admin should be added, whenever a new database is created.
   - The admin creates/edits/deletes a parking lot. **Note:** Delete only if all spots in the parking lot are empty.

3. User dashboard - for the User

- The user can choose an available parking lot and allocation is done as per the first available parking spot. **Note:** The user can't select a parking spot.
- The user changes the status of the parking spot to ***occupied,*** once the vehicle is parked.
- The user changes the parking spot status to ***released***, once the vehicle is moved out of the parking.
- The timestamp is recorded between parking in and parking out.
- Shows the summary charts on his/her parking.

Note: The database must be created programmatically (via table creation or model code) or through shell. Manual database creation, such as using DB Browser for SQLite, is **NOT allowed.**

4. Backend Jobs

a. Scheduled Job - Daily reminders - The application should send daily reminders to users on g-chat using Google Chat Webhooks or SMS or mail

1. Check if a user has not visited or parking lot is created by the admin
2. If yes, then send the alert asking them to book a parking spot if required by them
3. The reminder can be sent in the evening, every day (students can choose the time)

b. Scheduled Job - Monthly Activity Report - Devise a monthly report for the user created using HTML and sent via mail.

1. The activity report can include parking spots booked per month, Most used parking lot by the user, amount spent on parking for a month or any relevant information to the user etc.
2. For the monthly report to be sent, start a job on the first day of every month → create a report using the above parameters → send it as an email

c User Triggered Async Job - Export as CSV - Devise a CSV format details for the parking spots used by the user till date

1. This export is meant to download the parking details (slot_id, spot_id, timestamps, coset, remarks etc.)
2. Have a dashboard from where the user can trigger the export
3. This should trigger a batch job, and send an alert once done

5. Performance and Caching

- Add caching where required to increase the performance
- Add cache expiry
- API Performance

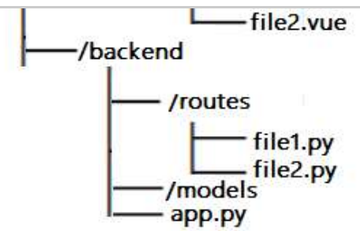
- Well-designed PDF reports for Monthly activity reports (Students can choose between HTML and PDF reports)
- External APIs/libraries for creating charts, e.g. ChartJS
- Single Responsive UI for both Mobile and Desktop
  - Unified UI that works across devices
  - Add to desktop feature
- Implementing frontend validation on all the form fields using HTML5 form validation or JavaScript
- Implementing backend validation within your APIs
- Provide styling and aesthetics to your application by creating a beautiful and responsive front end using simple CSS or Bootstrap
- Incorporate a proper login system to prevent unauthorized access to the app using Flask extensions like flask_login, flask_security etc.
- Implement a dummy payment portal (just a view taking payment details from user for paid quizzes
- Any additional feature you feel is appropriate for the application.

Evaluation

- Students have to create and submit a project report (not more than 5 pages) on the portal, along with the actual project submission
- The report must include the following things;
  - Student details
  - Project details, including the question statement and how you approached the problem statement
  - Frameworks and libraries used
  - ER diagram of your database, including all the tables and their relations
  - API resource endpoints (if any)
  - Drive link of the presentation video
- **If a student has used any form of AI/LLM for the project, you will need to mention the extent of use in your report**

Click here for project report demo

Possible folder structure:



- All code is to be submitted on the portal in a single zip file (zipping instructions are given in the project document **-** Project Doc T22025
- Video Presentation Guidelines (Advised):
    1. A short Intro (not more than 30 sec)
    2. How did you approach the problem statement? (30 sec)
    3. Highlight key features of the application (90 sec)
    4. Any Additional feature(s) implemented other than core requirements (30 sec)

  Note:
    1. The final video **must not exceed 5-10 minutes.**
    2. Keeping your video feed on during recording (like in a screencast) is optional but recommended.
- The video must be uploaded on the student drive with **access to anyone with the link** and the link must be included in the report:
    - This will be viewed during or before the viva, so it should be a clear explanation of your work.
- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions that the examiner asks:
    - This includes making changes as requested and running the code for a live demo.
    - Other questions may be unrelated to the project itself, but are relevant to the course.

Instructions

- This is a live document and will be updated with more details (wireframe)
- We will freeze the problem statement on or before 20/04/2025, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.