### 

### **Car Sell and Services API with Flask and MongoDB**

### **1.Introduction**

Car sell and services is a web based application in which services and selling of a car can be made where old or new car will be submitted to sell faster on this platform. New user can register themselves on this application and can submit car information to sell quickly . Information like car model, year, condition, expected price and image of a car can be submitted on the website which directly displays on the used car page where user will come and purchase it directly,

#### **Case study:**

This Project involves the design and implementation of a **Task Management Application** using Flask, an intuitive Python web framework. The application provides an interactive interface for users to manage their daily tasks efficiently by offering features such as task creation, reading, updating, and deletion (CRUD operations). It integrates with an SQLite database for data storage and manipulation, making it a lightweight and portable solution.

This Project involves the design and implementation of a Task Management Application using Flask, an intuitive Python web framework. The application program offers users via an interactive interface to help them efficiently manage their everyday duties efficiently by offering features such as task creation, reading, updating, and deletion (CRUD operations). For data management and storage, it integrates SQLite database, which is a portable and lightweight option.

#### **1. CRUD Operations for Task Management**

* **Create**: Users can add new tasks using the /createTask API endpoint.
  + Includes fields like task name, description, and date.
  + Ensures required fields (taskname and date) are provided.
* **Read**: Users can view a list of all tasks through the /tasklist endpoint.
  + Fetches tasks from the SQLite database and displays them.
* **Update**: Tasks can be updated using the /updatetask endpoint.
  + Supports retrieving task details for editing via a specific ID.
* **Delete**: Tasks can be deleted using /deletetask endpoint.
  + Can delete a specific task based on its name and date.

#### **2. Database Integration**

* Tasks are stored in SQLite database (taskmanager.db) which created at first time using SQL query in an app.
* Function has been created to converts database rows into Python dictionaries for easy manipulation.

#### **3. Dynamic Routing**

* Routes dynamically redirect users based on task operations (addtask, tasklist, deletetask, etc.).
* Supports specific task actions via dynamic route parameters (<string:id> and <string:ID>).

#### **4. Cross-Origin Resource Sharing (CORS)**

* Integration with Flask-CORS enables secure cross-origin requests
* Supports integration with external frontend applications.

#### **5. Error Handling**

* Handles invalid task creation requests with meaningful error messages.
* Logs exceptions and ensures the application doesn’t crash on user errors.

#### **6. User Interface (UI) Support**

* Uses Flask.render\_template to dynamically serve an index.html file for task management views.
* Flexible enough to support different task operations based on query parameters.

#### **Key Features Implemented:**

1. **Task Creation:**
   * Users can create tasks by providing:
     + Task Name (mandatory)
     + Description (optional)
     + Date (mandatory)
   * Input validation ensures required fields are filled.
2. **Task Listing:**
   * Displays a list of all tasks stored in the SQLite database.
   * Each task is shown with its name, description, and associated date.
3. **Task Update:**
   * Allows users to edit task details by selecting a specific task ID.
   * Provides a frontend for users to make updates.
4. **Task Deletion:**
   * Enables users to delete tasks either by task name and date or by ID.
5. **Interactive Interface:**
   * Routes like /addtask, /tasklist, /deletetask, and /updatetask ensure smooth navigation between views.
6. **RESTful API Integration:**
   * Exposes RESTful endpoints for task management:
     + POST /createTask for task creation.
     + GET /tasklist to fetch all tasks.
     + DELETE /delete/<ID> for deleting tasks.
     + PUT /update for task updates.

#### **2. Functional Requirements**

**2.1 User Actions**

* **Add Task:**
  + Input Fields:
    - Task Name (mandatory)
    - Description (optional)
    - Date (mandatory)
  + Backend Endpoint: /createTask (POST)
* **View Tasks:**
  + Display a list of all tasks with fields:
    - Task ID
    - Task Name
    - Description
    - Date
  + Backend Endpoint: /tasklist (GET)
* **Update Task:**
  + Input Fields for Update:
    - Task Name
    - Description
    - Date
  + Backend Endpoint: /update (PUT)
* **Delete Task:**
  + Specify Task Name and Date for deletion.
  + Backend Endpoint: /deletetask/<ID> (GET) or /delete/:ID (DELETE)

**2.2 Application Views**

1. **Home Page (/):**
   * Displays the main interface for task management.
2. **Task List View (/tasklist):**
   * Displays a table of tasks fetched from the database.
3. **Add Task View (/addtask):**
   * Form for creating a new task.
4. **Update Task View (/updatetask):**
   * Interface to select and update existing tasks.
5. **Delete Task View (/deletetask):**
   * Interface to select and delete tasks.

**2.3 REST API Endpoints**

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Method** | **Description** |
| /createTask | POST | Creates a new task. |
| /tasklist | GET | Fetches all tasks. |
| /deletetask/<ID> | GET | Deletes a task by Task Name and Date. |
| /delete/:ID | DELETE | Deletes a task by ID. |
| /update | PUT | Updates an existing task. |

#### **3. Non-Functional Requirements**

**3.1 Performance**

* The system should handle up to 50 concurrent users without noticeable delays.
* API responses should be returned within 500ms under normal conditions.

**3.2 Usability**

* The application must be easy to navigate with intuitive naming conventions for routes.
* Error messages should be user-friendly and descriptive.

**3.3 Scalability**

* SQLite is used for simplicity but should be replaceable with a more robust database like PostgreSQL for scaling.

**3.4 Security**

* Ensure proper validation for user inputs.
* Protect the application from SQL injection and other common attacks.

**3.5 Portability**

* The application should be platform-independent and deployable on any environment supporting Flask.

#### **4. Data Requirements**

**4.1 Database Schema**Table: tasks

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Constraint** |
| id | INTEGER | Primary Key |
| taskname | TEXT | Not Null |
| description | TEXT | Optional |
| date | TEXT | Not Null |

#### **5. Constraints**

1. SQLite is used for data storage, suitable for small-scale use cases.
2. Flask application will be run locally or on a small web server.
3. No authentication system is included in the initial phase.

#### **6. Assumptions**

1. Tasks are user-independent, meaning no user management system is required.
2. The application will primarily be used on modern web browsers.

#### **7. Deliverables**

* Fully functional Task Manager Application with:
  + Web interface for task operations.
  + REST APIs for task operations.
* Deployment instructions and user documentation.

# **1.** **Requirements Specification:**

### **3.1** **System Users:**

#### **USERS ACTION:**

● Browse the Tasks

● Add task

● VIEW TASK

● UPDATE Task

* Delete Task

#### **Extra Features:**

Searching

Sorting

Validation

● Manage Tasks(add, update, delete)

● View and manage all Tasks

### **3.2** **Software Requirements:**

Based on the components identified in the case study, the software requirements for the Task Management system can be split into the following functional and non-functional requirements

#### **3.2.1** **Software Tools and Technologies: Backend:**

● Python

● Flask(Web framework)

#### **Frontend:**

● HTML

● CSS

● JavaScript

#### **Database:**

● SQLITE

#### **3.1.1** **Functional Requirements:**

#### **1.** **User Authentication:**

● Users can register with a username, fullname, mobile number, email, and password.

● Users can log in and log out of the system.

● Passwords are securely stored and encrypted.

#### **2.** **Task Management:**

● Display tasks in the task list

* Each task includes a Task, Description, date,month and year and “Add task” button

● Users can search for Task items by name.

#### **3.** **Creating a Tasks:**

● Users can create a tasks by giving task name, description, and date.

● Only logged-in users can make Tasks.

● User can view easily all CreatedTask .

#### **3.2.3.Non-Functional Requirements:**

#### **Usability:**

● Intuitive and user-friendly interface.

● Clear form instructions and validations.

#### **Performance:**

● Handle multi simultaneous users efficiently.

● Minimal page load times.

#### **Security:**

● Secure storage of user data and passwords.

● Measures to prevent unauthorised access and data breaches.

* The application must be easy to navigate with intuitive naming conventions for routes.
* Error messages should be user-friendly and descriptive

#### **Scalability:**

● Accommodate an increasing number of users and Tasks.

● Efficient database management for growing data volumes.

### **3.3. Data Storage and Access:**

#### **Database: SQLITE**

● SQLITE is a portable and light weighted database which is use to create db locally on devices. So whenever app will be install locally it fetch and manipulates data there only.

# **5.** **Implementation:**

### **5.1** **Development Environment:**

The development environment for the Task management system involves a combination of softwares tools, frameworks, and libraries used to create and manage the project. The key components of the development environment include:

#### **Backend:**

● Python was chosen for its simplicity, readability, and extensive support for web development through flask framework

● Flask is a lightweight and flexible web framework for python that provides the necessary tools and libraries to build web applications.

#### **Frontend:**

● HTML: Used to create the structure and content of the web pages.

● CSS: Used for styling the web pages, ensuring a visually appealing design.

● JavaScript: Used for adding functionality to the add to Add task button in the home page.

#### **Database:**

● MAn open-source, serverless, lightweight relational database management system (RDBMS) is called SQLite. Because of its ease of use, low resource requirements, and little configuration, it is frequently used to integrate database functionality into applications.

#### **Development Tools:**

● Visual Studio Code

● Git and GitHub