Team 1	Team 2	Team 3	Team 4	Team 5	Team 6	Team 7	Team 8	Team 9	Team 10
Ankith	Karthik Kumar	Sidvi	Sanjay	Karthik Nair	Hisana	Julius	Nitta	Sneha	Megha
Minu	Shaun	Akshaya	Lakshmi	Chaitanya	Ashin	Aiswarya	Jissa	Amal Jose	Reema
Swathi	Nandini	Augustine	Lena	Hemandh	Nabeel	Lahin	Shibin	Archana	Anurag
Govind	Hanniya		Ajay						Elvin

Core Requirements for All Projects:

- **Python (Flask) Backend:** Implement a RESTful API with CRUD (Create, Read, Update, Delete) operations and appropriate data models. The API will be built using the Flask framework.
- **Database (Optional):** Use a lightweight database for persistence. A great choice for Flask is **SQLite**, which is included with Python. You could also use a local MySQL or PostgreSQL instance.
- **Angular Frontend:** Develop a responsive user interface to interact with the backend API, displaying, adding, editing, and deleting data.
- **API Consumption:** The Angular application must consume the REST APIs built in Python using Flask.
- **Error Handling:** Implement basic error handling on both the backend (e.g., for invalid inputs, not found resources) and frontend (displaying user-friendly messages).
- **Data Validation:** Implement basic input validation on both the frontend and backend.
- **Routing:** Implement basic routing in Angular to navigate between different views.

Using SQLite with a Python Flask API is straightforward with the **Flask-SQLAlchemy** extension. Here's a single, comprehensive example that demonstrates how to set up, define a model, and perform basic CRUD (Create, Read, Update, Delete) operations.

---

1. Project Setup and Configuration

First, you'll need to install the necessary libraries. Flask-SQLAlchemy simplifies the process of integrating SQLAlchemy with your Flask application.

```
None
pip install Flask Flask-SQLAlchemy
```

Next, configure your Flask application to use an SQLite database file. We'll create a file named `site.db` in your project directory.

Python

```python
# app.py

from flask import Flask, jsonify, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

2. Define the Database Model

Now, define your data model. This Python class will represent a table in your SQLite database. In this example, we'll create a `User` model.

Python

```python
# app.py (continued)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)

    def to_dict(self):
        return {
            "id": self.id,
            "username": self.username,
            "email": self.email
        }
```

3. Create the Database and Tables

Before running the API, you need to create the database file and the `User` table. You can do this by running a small script or from a Python shell.

Python

```python
from app import app, db
with app.app_context():
    db.create_all()
```

---

4. Implement CRUD Operations

Here are the API endpoints for each of the CRUD operations. We'll use a `to_dict` method in the `User` model to easily convert objects to JSON.

Create (POST): Add a New User

This endpoint accepts a `POST` request with JSON data to create a new user.

Python

```python
# app.py (continued)

@app.route('/users', methods=['POST'])
def create_user():
    data = request.get_json()
    new_user = User(username=data['username'],
email=data['email'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify(new_user.to_dict()), 201
```

Read (GET): Retrieve Users

This endpoint fetches all users from the database and returns them as a JSON list.

Python

```python
# app.py (continued)

@app.route('/users', methods=['GET'])
def get_users():
    users = User.query.all()
    return jsonify([user.to_dict() for user in users])
```

Update (PUT): Modify an Existing User

This endpoint updates a user's details based on their `id`.

Python

```python
# app.py (continued)

@app.route('/users/<int:user_id>', methods=['PUT'])
def update_user(user_id):
    user = User.query.get_or_404(user_id)
    data = request.get_json()
    user.username = data.get('username', user.username)
    user.email = data.get('email', user.email)
    db.session.commit()
    return jsonify(user.to_dict())
```

Delete (DELETE): Remove a User

This endpoint deletes a user from the database based on their `id`.

Python

```python
# app.py (continued)

@app.route('/users/<int:user_id>', methods=['DELETE'])
def delete_user(user_id):
    user = User.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    return '', 204
```

To run the application, simply execute the following command:

```
None
flask run
```

Shared Usecases for Assignments

1. Task Management System for a Small Team

Ankith Ramesh
Minu Mariam
Swathi Prasad
Govind Nair

- **Business Problem:** A small startup needs a simple way for team members to create, assign, track, and complete tasks. Existing solutions are too complex or expensive.
- **Solution:** Develop a web application where users can:
  - Create new tasks with a title, description, due date, and assigned team member.
  - View all tasks, filter by assigned person or status (e.g., "To Do", "In Progress", "Done").
  - Mark tasks as completed.
  - Edit existing task details.
  - Delete tasks.
- **Key Learning Points:** Basic CRUD, data relationships (tasks to users), date handling, simple filtering.

2. Product Catalog and Inventory Viewer

Karthik Kumar
Shaun Thomas
Nandini Ranjunath
Aishathul Haniyya

- **Business Problem:** A small online store wants a simplified internal tool to view their product catalog and current stock levels without logging into a complex e-commerce platform.
- **Solution:** Build an application that allows:
  - Displaying a list of products with details like name, description, price, and current stock.
  - Searching/filtering products by name or category.
  - *Optional (harder):* A simple form to "update" stock levels (e.g., increase/decrease by a fixed amount) for a given product (no full inventory management).

- **Key Learning Points:** Displaying lists of data, searching/filtering, simple update operations, potentially basic form handling for stock adjustments.

3. Simple Blog/Article Management System

Akshaya Somaraj
Augustine Kadavan
Nagalapuram Sidvilaasa

- **Business Problem:** A content creator wants a straightforward way to manage their articles online, including creating new posts, editing existing ones, and publishing them.
- **Solution:** Create a system where users can:
  - Create, read, update, and delete blog posts (title, content, author, publication date).
  - View all blog posts in a list.
  - View a single blog post in detail.
  - *Optional (harder):* Implement a basic markdown editor for the content.
- **Key Learning Points:** Textual data management, date formatting, potentially integrating a third-party library for markdown preview (frontend).

4. Expense Tracker for Personal Finance

Sanjay Prakash
Lakshmi Lisha
Lena Joseph
Ajay Vinod

- **Business Problem:** Individuals need a better way to track their daily expenses and categorize them to understand spending habits.
- **Solution:** Develop an application where users can:
  - Add new expenses with amount, description, category (e.g., Food, Transport, Entertainment), and date.
  - View a list of all expenses.
  - Filter expenses by category or date range.
  - Calculate total expenses for a selected category or period.
- **Key Learning Points:** Numerical data handling, aggregation (summing expenses), filtering by multiple criteria, date pickers on the frontend.

5. Basic Student/Course Management System

Karthik Nair
Chaitanya Sudheer
Hemandh Jiji

- **Business Problem:** A small training institute needs a system to manage students and the courses they are enrolled in.
- **Solution:** Create an application to:
  - Manage students (add, edit, delete student details like name, ID).
  - Manage courses (add, edit, delete course details like name, code, credits).

    - Enroll students in courses and remove them from courses (many-to-many relationship).
    - View a list of students and the courses they are enrolled in.
    - View a list of courses and the students enrolled in each.
- **Key Learning Points:** Managing relationships between entities (many-to-many), complex CRUD involving multiple resources.

6. Recipe Management Application

Hisana Thasneem
Ashin Bijosh
Nabeel Manjalamkuzhi

- **Business Problem:** A home cook wants to organize their recipes, search through them, and easily view ingredients and instructions.
- **Solution:** Build an application that allows:
    - Creating recipes with a name, description, list of ingredients, and instructions.
    - Searching recipes by name or ingredients.
    - Viewing a detailed recipe.
    - *Optional (harder):* Implement a "favorite" feature.
- **Key Learning Points:** Handling lists within a data object (ingredients, instructions), text search, potentially more complex UI for recipe display.

7. Simple Library Book Catalog

Julius Shaji
Aiswarya Sasidharan
Lahin Saleem

- **Business Problem:** A small community library needs a digital catalog to keep track of its books, including basic lending information.
- **Solution:** Develop a system where:
    - Librarians can add, edit, and delete books (title, author, ISBN, publication year, genre).
    - Track the availability of books (e.g., "Available", "Borrowed").
    - *Optional (harder):* Implement a simple "borrow" and "return" function (no user authentication needed, just changing book status).
    - Users can search for books by title, author, or ISBN.
- **Key Learning Points:** CRUD with status updates, searching, potentially date tracking for borrowed/returned.

8. Fitness Activity Tracker

Nitta Sogal
Jissa Johnson

Shibin Sajeev

- **Business Problem:** Individuals want to track their daily fitness activities (e.g., running, cycling, walking) including distance, duration, and calories burned.
- **Solution:** Create an application where users can:
  - Log new activities with activity type, date, duration, distance, and estimated calories burned.
  - View a list of all logged activities.
  - Filter activities by type or date range.
  - Calculate total distance or calories burned for a selected period.
- **Key Learning Points:** Numerical and date input, aggregation, charting/visualization (even basic text-based summaries on the frontend can be a challenge here).

9. Simple Appointment Scheduler

Sneha James
Amal Jose
Archana Sudhakar

- **Business Problem:** A small service provider (e.g., a salon, a tutor) needs a basic system to manage appointments with clients.
- **Solution:** Build an application to:
  - Create appointments with client name, service type, date, and time.
  - View all upcoming appointments.
  - *Optional (harder):* Check for time conflicts when creating new appointments.
  - Cancel/reschedule appointments.
- **Key Learning Points:** Date and time handling, conflict checking (logic on backend or frontend), managing dynamic data based on time.

10. Customer Feedback/Support Ticket System

Megha Nambiar
Reema Shathick
Anuraag Pillai
Elvin John

- **Business Problem:** A small business needs a way for customers to submit feedback or support tickets, and for staff to track their resolution.
- **Solution:** Develop a system where:
  - Customers can submit new tickets (title, description, contact info, priority).
  - Staff can view all tickets.
  - Staff can update ticket status (e.g., "New", "In Progress", "Resolved").
  - Staff can add comments/notes to a ticket.
  - *Optional:* Implement a simple search/filter for tickets based on status or keyword.
- **Key Learning Points:** Handling multi-line text input, status management, adding sub-entities (comments) to a main entity (ticket), potential for slightly more complex UI for ticket details and comments.

@All

11. The Assignment: Online Bookstore Data Analysis

You are given a CSV file named `book_sales.csv`. Your task is to perform a series of data analysis steps using the Pandas library to answer a few questions about the sales data.

Dataset (`book_sales.csv`)

```
None
Book_Title,Author,Genre,Price,Quantity_Sold,Sale_Date,Customer_Ra
ting
The Cybernetic Prophet,Jules
Verne,Sci-Fi,19.99,150,2025-01-15,4.5
Mystery of the Scarlet Mansion,Agatha
Christie,Mystery,12.50,200,2025-01-16,4.8
The Last Alchemist,Jules Verne,Fantasy,24.99,105,2025-01-17,4.2
A Tale of Two Cities,Charles
Dickens,Classic,10.00,300,2025-01-18,4.9
The Cybernetic Prophet,Jules Verne,Sci-Fi,19.99,75,2025-01-20,4.5
The Last Alchemist,,Fantasy,24.99,90,2025-01-21,4.2
A Tale of Two Cities,Charles
Dickens,Classic,10.00,150,2025-01-22,4.9
Cooking with Confidence,Julia
Child,Cookbook,35.00,80,2025-01-23,4.7
Mystery of the Scarlet Mansion,Agatha
Christie,Mystery,12.50,120,2025-01-24,4.8
```

Detailed Tasks

Complete the following steps in a single Python script or Jupyter Notebook.

Task 1: Data Loading and Initial Inspection

- Import the **pandas** library.
- Read the `book_sales.csv` file into a Pandas DataFrame.
- Display the first 5 rows of the DataFrame to get a quick overview of the data.
- Print a concise summary of the DataFrame using the `.info()` method to check data types and non-null values.

Task 2: Data Cleaning and Preparation

- Check for **missing values** in the DataFrame.
- You'll notice a missing value in the `Author` column. Replace this missing value with the string `'Unknown'`.
- Convert the `Sale_Date` column to a proper datetime format.
- Remove any duplicate rows in the DataFrame. A duplicate row is one where all column values are identical to another row.

Task 3: Data Analysis and Aggregation

- Calculate and print the **total quantity of books sold** across all sales.
- Find the **average price** of all books.
- Group the data by `Genre` and calculate the **total quantity sold for each genre**. Display the result.
- Determine the **top 3 best-selling books** based on the total `Quantity_Sold`.
- Find the `Book_Title` with the highest `Customer_Rating`.

Details and Tips

- **Libraries:** You only need `pandas` for this assignment.
- **Creating the CSV:** Before you start, save the data provided in the "Dataset" section into a file named `book_sales.csv` in the same directory as your Python script.
- **Solving the Tasks:** Approach the tasks one by one. Use a fresh DataFrame copy for each major task if you're worried about accidental data modifications.
- **Key Pandas Functions to Consider:** `pd.read_csv()`, `.head()`, `.info()`, `.isnull()`, `.fillna()`, `pd.to_datetime()`, `.drop_duplicates()`, `.sum()`, `.mean()`, `.groupby()`, `.sort_values()`.