Project Submission Guideline

- **Tasks**: Each student will be assigned 2-3 tasks based on their role. All tasks are mandatory.
- **Task Folders**: After completing each task, create a folder named task-1, task-2, etc.
- **Final Submission**: Once all tasks are completed, zip the folders.
- **Upload**: Upload the zip file to Google Drive and set the sharing permission to "Anyone with the link."
- **Form**: Fill out the submission Google Form.

  https://docs.google.com/forms/d/1ZtciVwIsOg54CQ8uQjIj6BSQZx6tPabPO0MBAHmaNvI/edit

Prerequisites

BACKEND: Django, Github, Promt Engg., Database

Backend/Database 1st Year

This is not a group assignment, you will have to individually submit this task

1) to design a database for a personal finance management system that tracks users, their bank accounts, and transactions.

Requirements:

4. Users:
    - ◯ EachuserhasauniqueID,name,email,andcontactinformation.
    - ◯ Ausercanhavemultiplebankaccounts.
5. BankAccounts:
    - ◯ Eachbankaccountisassociatedwithoneuserandhasaunique account ID, account type (e.g., savings, checking), and balance.
    - ◯ Ausercanhavemultiplebankaccounts,buteachaccountbelongsto one user only.
6. Transactions:
    - ◯ Eachtransactionisassociatedwithonebankaccountandhasa unique transaction ID.
    - ◯ A transaction can either be a deposit or a withdrawal and includes the amount, transaction type (deposit/withdrawal), and date.
    - ◯ Abankaccountcanhavemanytransactionsovertime,buteach transaction is linked to only one account.

Tools:

Use Lucidchart or DrawDB for designing the database.

2) You need to develop a django api/backend service which accomplishes the following

Technologies Used: Python, Django, html, css, github, sql

Batch 1 - Assigned Students

Anushka Rathore, Ojas Gangrade

Task:

Design a Django-based backend API for a small library management system. This system should support managing books, members, and borrowing activities. Use the following specifications to guide your development:

Requirements

1. **DatabaseModels** :
    - ◯ `CreateDjangomodelsforBook,Member,andBorrowingTransaction` based on the following:
        - ■ **Book**:EachbookshouldhaveauniqueID,title,author,andgenre.A book can be borrowed multiple times but only by one member at a time.
        - ■ **Member**:EachmembershouldhaveauniqueID,name,email,and phone number. A member can borrow multiple books, but each borrowed book can only be checked out by one member at a time.
        - ■ **BorrowingTransaction**:EachtransactionshouldrecordthememberwhoborrowedthebookthedateborrowedandareturndateindicatingwhenthebookisdueIfabookisntreturnedbytheduedateitisconsideredoverdue.
2. **API Endpoints**:
   Implement the following endpoints:
    - ◯ POST /books/:Createanewbookwithtitle,author,andgenre.
    - ◯ POST /members/:Createanewmemberwithnameandcontact information.
    - ◯ POST /borrow/:Recordaborrowingtransaction,specifyingthemember, the book, and the borrowing date. Ensure a book can only be borrowed by one member at a time.
    - ◯ POST /return/:Updatethedatabasetomarkabookasreturned.
    - ◯ GET /overdue-books/:Listalloverduebooks,includingthememberwho borrowed them, the borrowing date, and the due date.
    - ◯ **Youcanalsocreatemoreendpointsbasedonyourinterpretation.**
3. **Authentication:**
   Use Django session authentication to restrict task creation, updating, and deletion to authenticated users.
4. **Tools:** Django models, Django REST Framework, API views.
5. **You are free to use any kind of database or required software such as postman for api testing.**

Batch 2 - Assigned Students

Aditya Koul, Manas Gupta

Task:

Design a Django-based backend API to support a digital clothing store. The API should handle clothing item management, user authentication, filtering, and shopping cart functionalities.

Requirements

Models

The system will have two main models: **Product** and **CartItem**. The **Product** model will store details about each clothing item, including its name, a description of the item, its price (as a decimal), the category (such as men's, women's, or kids'), size (using a CharField for options like S, M, or L), the available stock (as an integer), and a URL linking to an image of the item. The **CartItem** model will track the items added to a user's shopping cart. It will reference the authenticated user via a ForeignKey, associate the cart item with a product via another ForeignKey, and store the quantity of that product in the cart.

API Endpoints

- GET /api/products/:Listallproductswithoptionalfiltersforcategory, size, and price range.
- GET /api/products/{id}/:RetrievedetailsofaspecificproductbyID.
- POST /api/cart/:Addaproducttotheuser'sshoppingcart(authenticated users only).
- PUT /api/cart/{item_id}/:Updatethequantityofanitemintheuser's cart.
- DELETE /api/cart/{item_id}/:Removeanitemfromtheuser'scart.
- GET /api/cart/: Retrieve all items in the current user's cart along with the total price.
- **Youcanalsocreatemoreendpointsbasedonyourinterpretation.**

2. **Authentication:**
   Use Django session authentication to restrict task creation, updating, and deletion to authenticated users.
3. **Tools:** Django models, Django REST Framework, API views.
4. **You are free to use any kind of database or required software such as postman for api testing.**

Batch 3 - Assigned Students

Kuldeep Mujalde, Krishna Rathi, Mohammad Muntasir Multani

Task:

Create a backend API for a clinic where users can select a doctor, choose an available time slot, and book an appointment. The system should also allow doctors to list their available times and specialisations.

Requirements:

Models:

The system will have three main models: **Doctor**, **Appointment**, and **Patient** (optional). The **Doctor** model will store the doctor's full name, specialization (e.g., Cardiology, Pediatrics), average rating (a decimal between 1 and 5), and available appointment times, which can be managed using a JSONField or related table. The **Appointment** model will record essential details about the appointment, including the patient's name, contact information (phone or email), reason for the visit, the associated doctor (linked via a ForeignKey), the selected appointment time, and the current status of the appointment (such as "Booked," "Completed," or "Cancelled"). The **Patient** model, though optional, can store the patient's personal details, including their full name, contact information, and account creation timestamp, allowing for separate tracking of patient records within the system.

API Endpoints:

- GET /api/doctors/: List all doctors with their specializations and ratings.
- GET /api/doctors/{id}/:Retrievedetailsforaspecificdoctor,including available times.
- GET /api/appointments/:Listallbookedappointments(adminonly).
- POST /api/appointments/:Createanappointmentbyselectingadoctor,time slot, and providing patient details (name, contact, and reason).
- GET /api/appointments/{id}/:Retrievedetailsofaspecificappointment.
- PUT /api/appointments/{id}/:Updateanexistingappointment(e.g., reschedule or cancel).
- `DELETE /api/appointments/{id}/`:Cancelanappointment.
- `GET /api/doctors/{doctor_id}/timeslots/`:Retrieveavailabletimeslotsfor a specific doctor.
- `POST /api/doctors/{doctor_id}/timeslots/`:Addavailabletimeslotsfora specific doctor (admin or doctor only).
- `PUT /api/doctors/{doctor_id}/timeslots/{slot_id}/`:Updatean existing time slot (e.g., mark as unavailable).

(optional)
3)Additionally you can also create a frontend for the assigned backend component.(For all batches of Backend)