**This hack-a-thon will extend over 2 days and will include the following high-level areas of focus for providing an account management system for bank customers:**

* **Creation of an SQLite database and tables to provide persistent data storage for your application**
* **Creation of a set of model classes in Python which will be used to represent the domain entities in your application**
* **Creation of a set of data repository classes in Python which will be used to manage activity between the domain entities in your application and your database tables, supporting CRUD (create, retrieve, update, and delete)**
* **Creation of a set of controller classes in Python which will act as the entry point for your API and will provide routes with which consumers can integrate to leverage the back-end functionality described above**
* **Creation of a set of Angular feature modules and components providing the front-end for your application and integrating with the Python API for back-end business logic and data access**
* **Creation of a set of Angular Reactive forms providing the graphical user interface for user view, interaction, and maintenance of data**
* **Leverage of Angular routes for coordination of various user workflows within the application**

Diagram

Description automatically generated

**Technical Requirements**

* **Angular 10+**
* **Python 3.8+**
* **FastAPI and Pydantic**
* **SQLite**
* **VS Code with Angular extensions and Python extensions installed and enabled**

Create SQLite Database

* Use the provided code to generate the “accounts.db” SQLite database or utilize the “accounts.db” file as provided (see starter files in <https://github.com/KernelGamut32/hat-starter>)
* Address
  + ID (autoincrementing integer) – primary key
  + Address (text)
  + City (text)
  + State (text)
  + Zip code (text)
* Customer
  + ID (autoincrementing integer) – primary key
  + First name (text)
  + Last name (text)
  + Address ID (foreign key)
  + Email address (text)
* Account
  + ID (autoincrementing integer) – primary key
  + Account number (text)
  + Customer ID (foreign key)
  + Current balance (decimal value)

Create Python Model Classes

Create Python model classes for each of the entities described above, including class hierarchies to reflect the proper dependencies between the domain entities (i.e., Customer will have an Address reference and Account will have a Customer reference).

Create Python Repository Classes

Create Python repository classes for each of the entities described above, including class hierarchies to reflect the proper dependencies between the domain entities. Include methods to support CRUD operations (create, retrieve, update, and delete).

Create Python Controller Classes

Create a Python controller class providing routes for each of the following operations, leveraging repository classes for actual database operations:

* Open account
* Retrieve all accounts
* Retrieve specific account
* Account withdrawal
* Account deposit
* Close account

Create Angular Components

* Create Angular components (modules, components, forms, and routes) to provide support for the following operations:
  + Open account
  + Retrieve all accounts
  + Retrieve specific account
  + Account withdrawal
  + Account deposit
  + Close account
* Include required HTTP client components in Angular for interface with the Python back-end API endpoints

Add Error Handling, Styling, and Automated Unit Tests (If Time Permits)

If time permits, add error handling to address potential error conditions, improve styling, and/or add automated unit tests. For example:

* Prevent a withdrawal that would result in an overdraw
* Require that a minimum of $25.00 be included to open a new account
* Prevent specification of invalid values on a withdrawal or deposit