John Li
99661787

Design of Closet App

Part 1:

Front End:

Module Name	**Purpose**
Clothes View	Display clothes categories, clothes items, and outfits
User Profile View	Display user information, such as login information and user preference
Calendar View	Display user's calendar (should match the connected Google Calendar)
Weather View	Display weather information of the current day and the next day
Notification Center	Display notification messages
Navigation Bar	A navigation bar that navigates to different views
Notification Controller	A controller that manages notifications
Request Manager	The controller that connects front-end views and back-end APIs

Back End:

Module Name	**Purpose**
Clothes DB	Database to save all the clothes and outfits information
User DB	Database to save user information, such as user preference
Clothes Category	Clothes category entity
Clothes Item	Clothes item entity
User	User entity
Outfit Creator	Receive all the required information (clothes, user preference, calendar event, weather) and create outfits for the user. (The complex logic is included in this module)
Scheduler	A controller to set notification time and push notifications
Google Calendar APIs	External APIs to obtain events information
OpenWeather APIs	External APIs to obtain weather information

Part 2:

Clothes DB

- DBRecord findAllCategories (SQLQuery)
    - Find all clothes categories
- DBRecord findOne(SQLQuery)
    - Find one clothes category
- void addCategory(SQLQuery)
    - Add one clothes category
- void updateCategory(SQLQuery)
    - Update one clothes category
- DBRecord deleteCategory(SQLQuery)
    - Delete one clothes category
- DBRecord findAllItems(SQLQuery)
    - Find all clothes items
- DBRecord findOneItem(SQLQuery)
    - Find one clothes item
- void addItem(SQLQuery)
    - Add one clothes item
- void updateItem(SQLQuery)
    - Update one clothes item
- DBRecord deleteItem(SQLQuery)
    - Delete one clothes item

User DB

- DBRecord save(SQLQuery)
    - Save the user information into database
- void update(SQLQuery)
    - Update user information in database

Clothes Category

- ClothesCategory[] getCategories()
    - Return a list of clothes categories
- ClothesCategory getOneCategory(Long:id)
    - Return one clothes category by its id
- void addCategory(String:name)
    - Add a new clothes category with the category name
- void updateCategory(Long:id)
    - Update a clothes category by its id
- Category deleteCategory(Long:id)
    - Delete a clothe category by its idand return the deleted category

Clothes Item

- ClothesItem[] getClothesItemsByCategory(Long:categoryId)
    - Return a list of clothes items that belong to one category
- ClothesItem getClothesItemById(Long:itemId)
    - Return one clothes item by its id
- void addClothes(String:name)
    - Add a new clothes item with the new clothes item name
- void updateClothes(Long:itemId)
    - Update a clothes item by its id
- Clothes deleteClothes(Long:itemId)
    - Delete a clothe item by its id and return the deleted clothes item

User

- Profile getProfile(userId)
    - Get user profile by his/her id
- void updateProfile(userId)
    - Update user profile

Outfit Creator

- OutfitMaterial materialCollect()
    - Collect all the required material to create outfits, including user preference, weather, calendar events, and clothes items
- Outfit createOutfit(OutfitMaterial)
    - Create one outfit
- Outfit[] createOutfits(OutfitMaterial)
    - Create a list of outfits

Scheduler

- void setNotification()
    - Set a notification time
- Notification pushNotification()
    - Push a notification

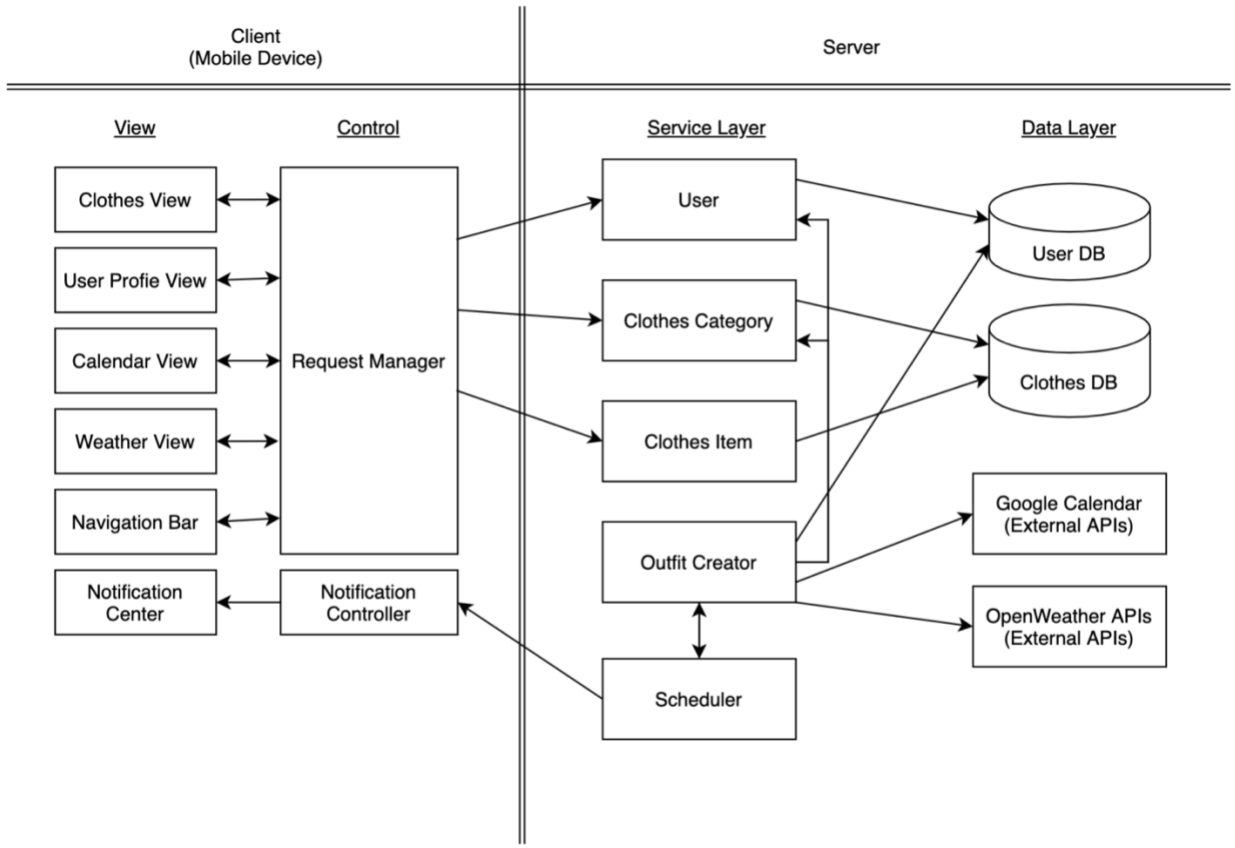
Google Calendar APIs

- GET /calendars/***calendarId***
    - Returns a calendar from the user's calendar list
- GET /calendars/***calendarId***/events
    - Returns events on the specified calendar

OpenWeather APIs

- GET api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
    - Get the current weather data by city name

Part 3:



Part 4:

I used both Client-Server architecture and Model-View-Controller architecture.

The client-Server architecture fits the app very well. The mobile device is client-side, and the logic and data operations are on the server-side.

I chose Model-View-Controller since it is one of the most popular architecture for web development, and I can apply it to the front-end part of the app.

Part 5:

I will use React Native as the front-end framework, MySQL as the database, and AWS as the Cloud provider. I choose them because I have used those technologies in my personal projects before, and those experiences can save me much time when developing my app in CPEN 321. For the database section, MySQL is not the only option. A non-relational database, such as MongoDB, is also a good choice for the app.

Part 6:

In M3, I listed two non-functional requirements.

- Users should not need more than 5 steps to a clothes item.
- Users should wait at most 2 seconds when requesting for an outfit suggestion.

For the first requirement, each step can be a different UI component. So, I need to make sure that there should be at most 5 different UI components involved when adding a clothes item.

For the second requirement, we have to design our backend API wisely by separating the task "request for an outfit suggestion" into several smaller sub-tasks. Then, we need to write a unit test for each API function to ensure the response time is acceptable. Finally, we re-test the whole task and calculate the cumulative time.

Part 7:

Inputs: Clothes items, calendar events, weather condition, user preference

Outputs: One suggested outfit

Main computational logic:

1. Check Calendar Events
    a. If the calendar has formal events, such as interviews and company parties, search for formal clothes items such as suits. If it has, then return the suit as the outfit.
    b. If there are outdoor events, such as hiking and camping, search for sportswear and sports shoes. If they are available, then return the sportwear as the outfit.
    c. This is an "else" case. If the user has special events but no suitable clothes available in the database, then include the content of the special events in the notification and recommend other outfits using the logic below.
2. Check Temperature
    a. If the temperature is below 5 °C, then select warm clothes, such as down jackets and sweaters.
    b. If the temperature is between 5 °C and 20 °C, the select shirts and long sleeves.
    c. If the temperature is above 20 °C, then select T-shirts.
3. Check User Preference
    a. For the all selected clothes item, delete the ones that do not match the user's color preferences and return the outfit.