MINE

A central media hub for all the things *you* love

CS 40700 - Senior Design Project

DESIGN DOCUMENT

Team 6
Utkarsh Agarwal *uagarwal@purdue.edu*
Shivangi Chand *chands@purdue.edu*
Amol Moses Jha *jha8@purdue.edu*
Pooja Tewari *tewarip@purdue.edu*

PURPOSE

Mission Statement

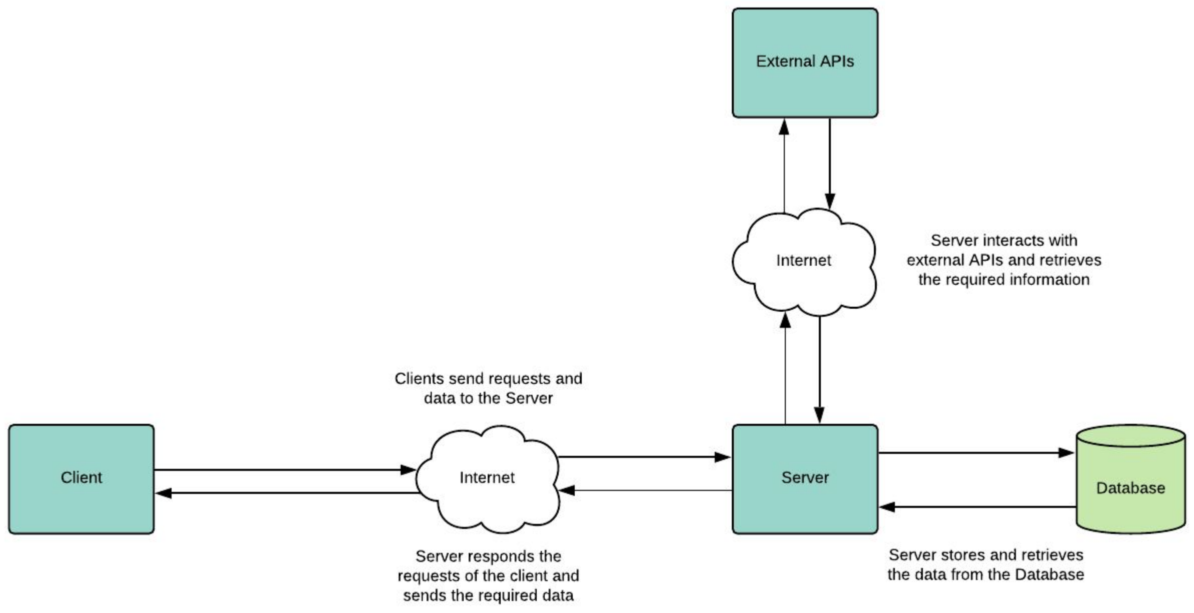
With the rapid proliferation of online content through an ever-increasing number of online streaming services as well as social networks, it can be hard to find something which one might want to devote their time and attention to. With shorter attention spans and ever tighter schedules, there is a huge trove of online content which vies for our attention at any given time. This ever burgeoning store of online content paradoxically leads to indecisiveness in the minds of users, with an additional tedium of logging in and searching every single online providers' portals for something which one would want to consume. This is where Mine comes in, acting as a central hub for searching and discovering content which a user might like.

Project Description

Mine would allow users to search for online content from a myriad of online content providers by specifying a search term and a desired category. Mine would then aggregate search results from a myriad of online content providers to present users with the most relevant results based on their specific search terms. Mine would also allow users to peruse their previous searches in order to find something else they might like, related to content they searched for earlier. Finally, Mine would also promote an online community of users by showing what's trending, so that users can find something that they would want to consume and feel connected to a larger community in the world at large.

DESIGN OUTLINE

We will be using the Client-Server Architecture for Mine. The following Diagram highlights the structure:


External APIs
Internet
Server interacts with external APIs and retrieves the required information
Clients send requests and data to the Server
Client
Internet
Server
Database
Server responds the requests of the client and sends the required data
Server stores and retrieves the data from the Database

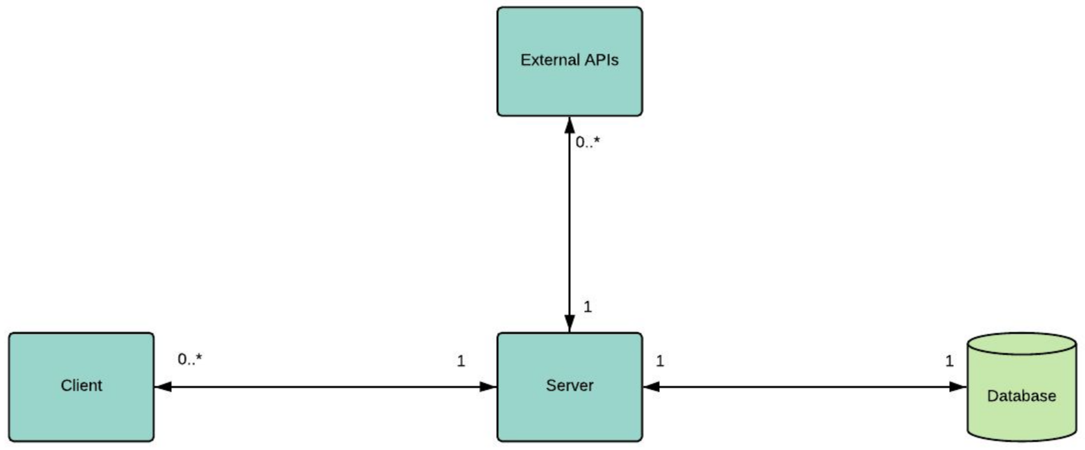


Figure 1: High Level Overview of Components

Components of the Architecture

1. Client:

    The client is responsible for the interaction with the user through the UI and will relay important information to the Server. For instance the client will be responsible for authorizing and authenticating the user, search requests made by the user will be sent to the server and will be done through the client, and the user will be able to update requests for their preferences and personal information to the server through the client.

2. Server:

    The Server is responsible for receiving all the data from the client and doing the required processing on it, for instance let the client know if the user is authenticated to use their respective account. The Server will send out responses to the Client for all the requests made. Other than the interaction with the Client, the Server is also responsible for the interaction with the database to insert, delete, update and retrieve data. The server will also interact with the external APIs to extract the data requested by the user.

3. Database:

    The database will be responsible for storing and retrieving all the data required for the web app. It contains user information such as name, email address, password, profile pictures, preferences, previous searches. It also provides authentication for the user to be able to use their account.

4. External API:

    External APIs will be used to retrieve information pertaining to the search term and category requested by the user. This will enhance the user experience and ensure that we provide better results to them.

Interactions

All the components except for the database and the Server interact via the internet. The decision to run the central server and database on the same machine allows maximal performance between these two components’ interaction. The Server is handling most of the load of the web app as it would be interacting with all the components and acting as a bridge between them. The Client can have multiple users at one point of time hence it will establish a many to one relationship with the server. The server will contact several APIs for getting data for a particular category type; since it will contact several APIs, it has a one to many relationship with the external APIs. The Server interacts with the database to insert, delete, update and retrieve data, since it is going to make one call at one time hence, they have a one to one relation.

DESIGN ISSUES

Functional Issues

1. *Issue: What fundamental architecture should we design our app with?*
    a. **Client-Server Architecture**
    b. Peer-to-Peer Architecture

Since we want our users to run a lightweight client that can be run on a myriad of platforms and computers, we decided that the heavy lifting of interacting with the APIs should be done by a central server which all our clients can connect to.

2. *Issue: What protocol should we use to build the client-server communication?*
    a. **Hypertext Transfer Protocol (HTTP)**
    b. WebSocket Communications Protocol

Since we do not require bi-directional, real-time communication between a client and the server, there is no real need to choose the webSocket communication protocol standard. The HTTP standard is well-understood, supported and documented for our use-case. Furthermore, HTTP is a fantastic choice for developing REST APIs since HTTP requests also encode a required verb for interacting between the client and the server and therefore, furthermore cements the decision to use HTTP.

3. *What database model should we use on the backend to structure our persistent data?*
    a. **Relational Database**
    b. Document Store Model Database
    c. Key-Value Pair Database
    d. Graph Model

Relational databases are highly performant and extremely well understood and supported databases, with a lot of features to help power modern apps. With great ORM support for a lot of backend frameworks, and the ability to specify and write custom queries for when such extensibility is required, relational databases made the most sense for our application.

4. *Which rendering mechanism would best serve our application?*
    a. **Client-side rendering**
    b. Server-side rendering

Client-side rendering is a well understood and tried and tested rendering mechanism. It also reduces the load on the central server by offloading significant rendering responsibilities to the client as compared to the server. This separation of rendering would furthermore allow our team to easily focus our energies into discrete facets of the application - the frontend can be easily built to consume a well-defined public API, completely disparate from the construction of the backend which would build the web service to serve the consumers of the aforementioned API.

5. *How flexible should our login mechanism be?*
    a. Only allow users to login after they've registered as a user
    b. **Allow users to login via popular online providers such as Google and Facebook, in addition to the option of registering as a new user.**

With a rapid proliferation of online services and applications, it is all too common for a user to have multiple accounts across a multitude of services. This burdens users with unnecessary bookkeeping - by making them track all their logins across various online services. Therefore, allowing users to login via popular online providers would offer them some relief from the aforementioned problem. This is the main reason we decided to support logging in via different popular accounts in addition to the default route of registering as a new user.

Non-Functional Issues

1. *What language should we use on the backend to power our web service?*
    a. **Java**
    b. Python
    c. PHP

With great backend pedigree and extensive support for libraries and frameworks, Java was an instant match. Furthermore, with great potential for scalability and a proven track record of robustness, Java is greatly appreciated in the software engineering community. Finally, with the right ratio of productivity compared with lower level languages such as C/C++, as well as the ability to support different programming paradigms with ease, Java was the correct choice.

2. *What framework should be used on the backend to structure the webservice?*
    a. **Spring**
    b. Struts

Being the most popular framework around, with support for everything which the web development process could entail including, but not limited to an MVC suite, a security suite, and an integrated ORM suite, Spring was an easy choice for structuring and adding functionality to the web application.

3. *Which relational database should we use to power persistent storage for the application?*
    a. **MySQL**
    b. MariaDB
    c. SQLite

MySQL is one of the most well supported and widely known and used DBMS systems for relational databases around. It has great reliability and scalability, and with great support with the Spring framework, it ended up being our DBMS of choice.

4. *What framework should be used on the frontend to power the user interface of the application?*
    a. **React**
    b. Angular
    c. Vue

React has great documentation, great community support and support for a whole suite of features. It is also not very opinionated, leading to great flexibility for designing user interfaces. JSX furthermore allows for rapid prototyping and allows for great productivity, all factors leading it to be the best choice for the frontend.

5. *How should the UI of the application be structured?*
    a. **Multi Page application**
    b. Single Page application

Since we need to have dynamic applications with multiple facets - like multiple searches across multiple categories, user navigation and code organization is vastly superior in a multi page application. Since we have very feature-rich and powerful frameworks on the frontend and a completely decoupled backend, it will be very easy to support a multi-page application while reducing any complexity which stems from our decision to do so.

DESIGN DETAILS

Class Descriptions

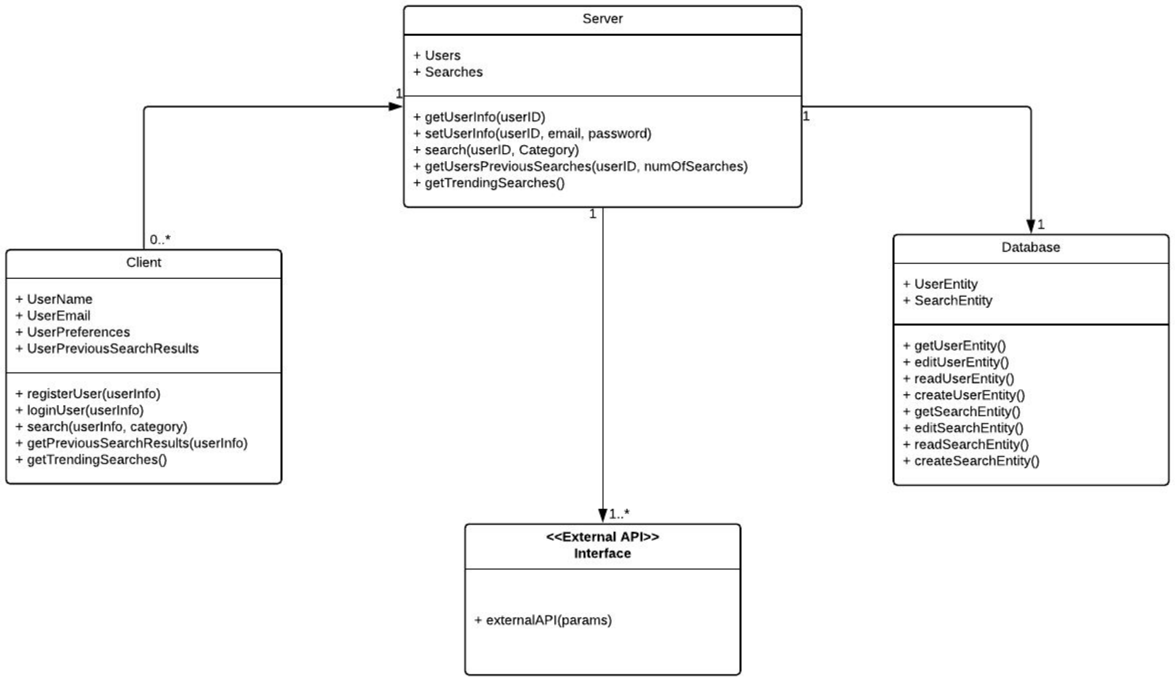


Figure 2: Class Diagram

Figure 2 shows an overview of the major components in our classes and how they will be organized and will interact with each other. These following are the details of these classes themselves:

1. Client
   - Each user will load the website on a device and browser of their choice, and will therefore be treated as a client in our architecture.
   - Every client can communicate with the server register a new user with a user's information input by the user.
   - A client can communicate with the server to login an existing user.
   - A client can allow a user to search for online content split across different categories.
   - A client also enables the user to view their previous searches.

2. Server
    - The central backend server serves all clients connecting to it and acts as the central connection between external APIs and the database.
    - The server allows clients to create and get a user's information.
    - The server also allows clients to search external content providers.
    - The server interacts with external APIs to populate searches from online content providers.
    - The server further allows clients to get previous searches for a specified user.
    - The server enables clients to get current trending searches.
3. Database
    - The database stores all persistent data about users and their searches.
    - It allows the server to insert new user records.
    - It allows the server to insert new search records.
    - It allows the server to query it to provide data about the users and the searches.
4. External API
    - External APIs provide us with search results for a specified search terms for a specific category.

Class Interactions

The core components of our application are related in the following manner:

- There can be many concurrent clients connecting to the central server. However, there can also be the case that there are no clients - our server is designed to be standalone. Therefore, there is a zero to many cardinality between potential clients and a server. The Client calls the Server for getting searches, user details, etc.
- There are many external APIs which the central server interacts with. Therefore, there is a many to one cardinality between the server and external APIs. The Server gets all the searches from External APIs.
- There is only one central server which interacts with the central database. Therefore, there is a one to one cardinality between the server and the database. The Server interacts with the Database to store searches and user details.

Sequential Diagrams

Register

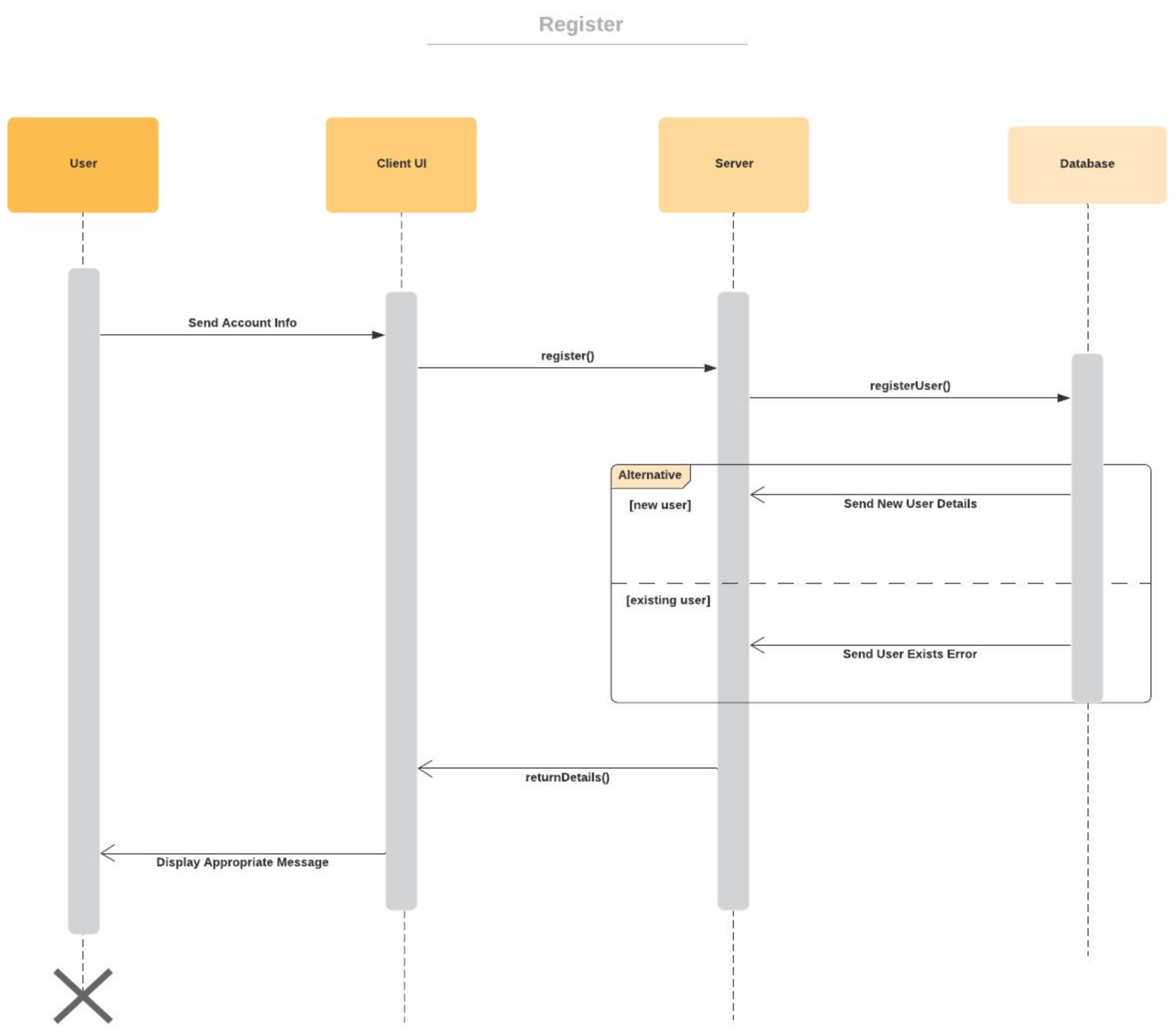


Figure 3: Sequence Diagram for Register

When a User has to register, their account information is sent through the Client UI to the Server. The server then sends this data to the Database to register the User. If it is a new user, the Database returns the New User details back to the Server. Otherwise, it sends User Already Exists Error to the Server. The Server then sends these details back to the Client UI which displays it to the User.

Login

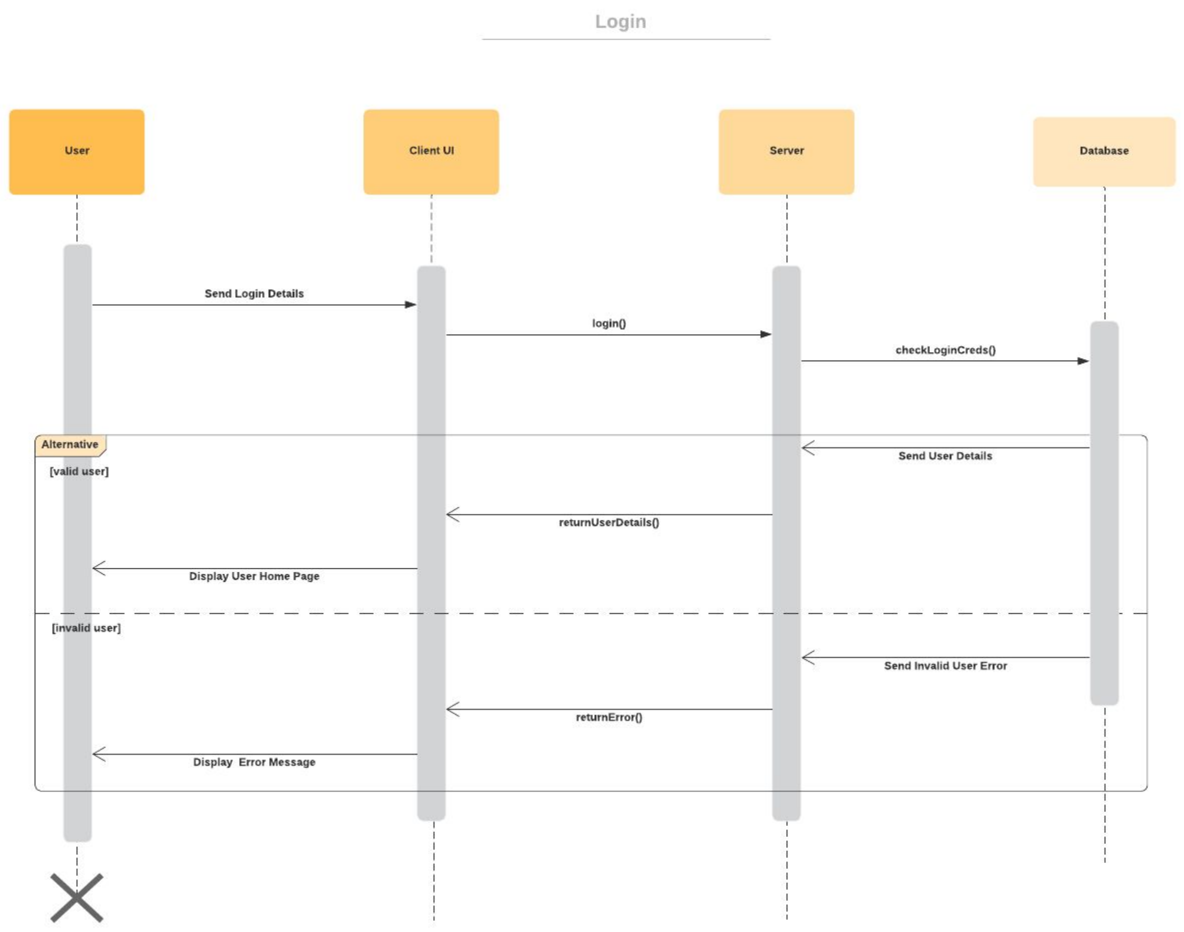


Figure 4: Sequence Diagram for Login

When a User has to login, their login information is sent through the Client UI to the Server. The server then sends this data to the Database to check login credentials of the User. If it is a valid user, the Database returns the User details back to the Server. Otherwise, it sends Invalid User Error to the Server. The Client UI displays the Home Page to the User in case of no error. Otherwise it displays error message.

User Profile Updates

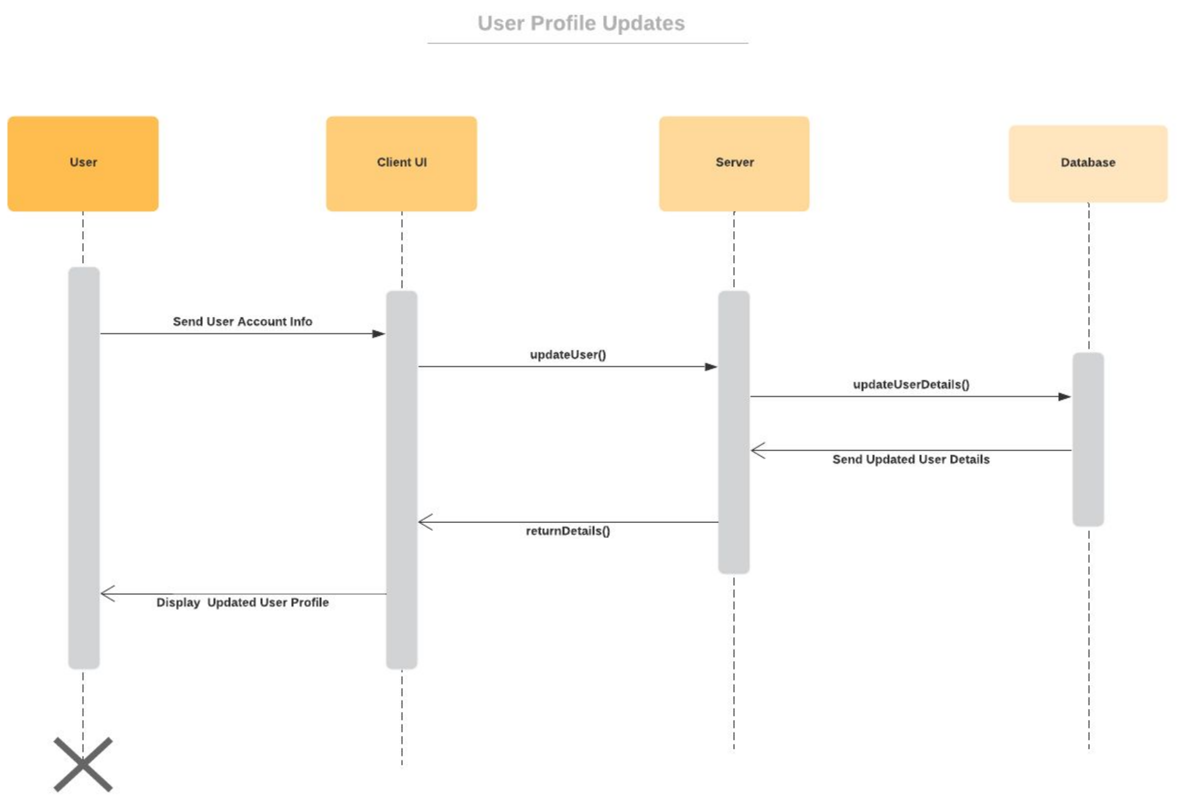


Figure 5: Sequence Diagram for User Profile Updates

When User wants to update their profile information and preferences, they will send new account information to the Client UI. Client UI will then send the details to the Server which will in turn ask the Database to update the User. The Database will then send the updated User details to the Server which will send it to Client UI and display it to User.

Search

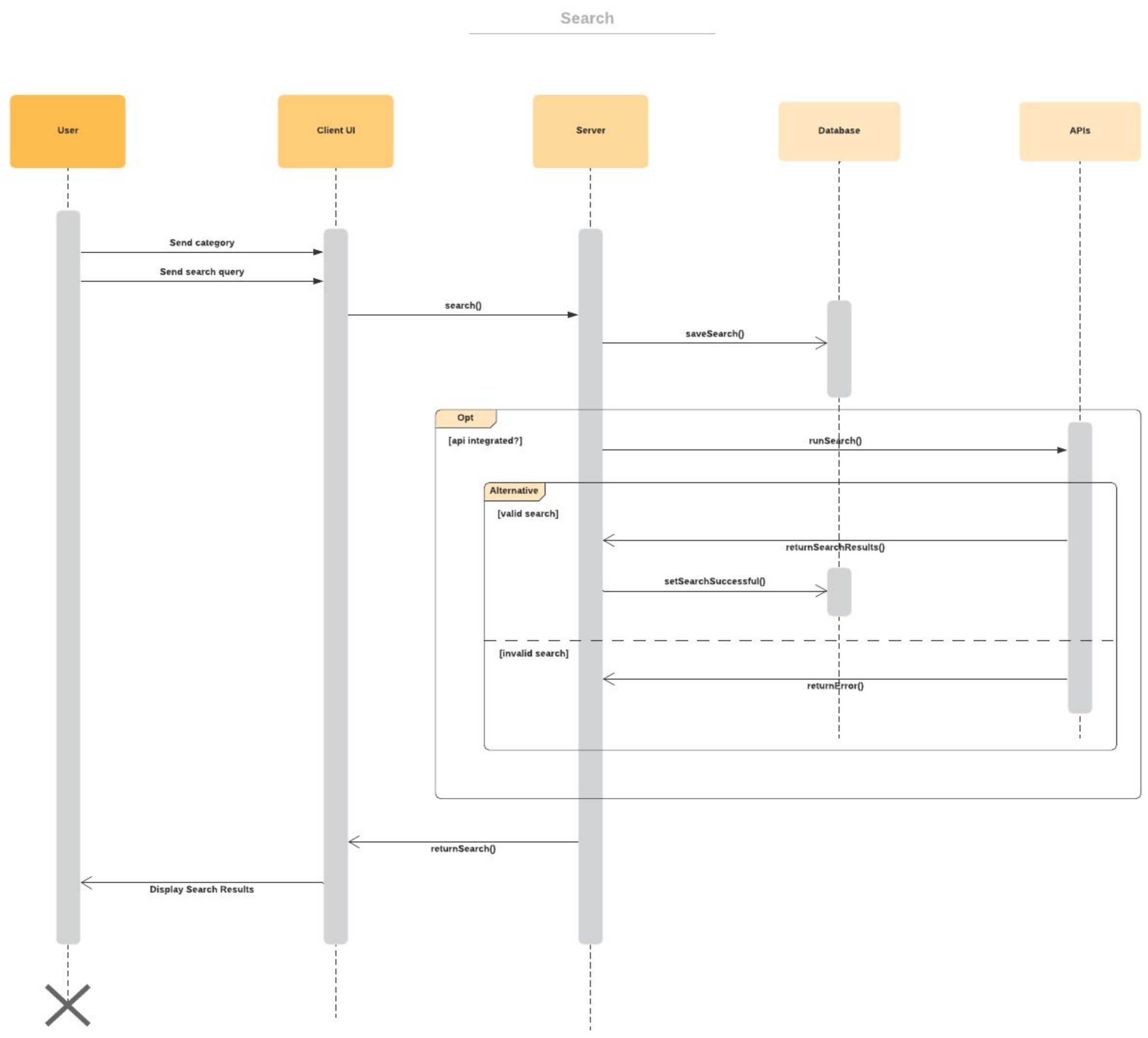


Figure 6: Sequence Diagram for Search

The User will send the category and search query details to the Client UI which will send these details to the Database for storage. If the Third-Party APIs are integrated for the User, then the search will be sent to respective APIs. If it is a valid search, the API will return the result back to the Server. The Server will also update the Database to record that the search was successful. If the search was invalid, then the API will return error to the Server. Finally, the Server will send appropriate payload to the Client UI which will display it to the User.

Previous Searches

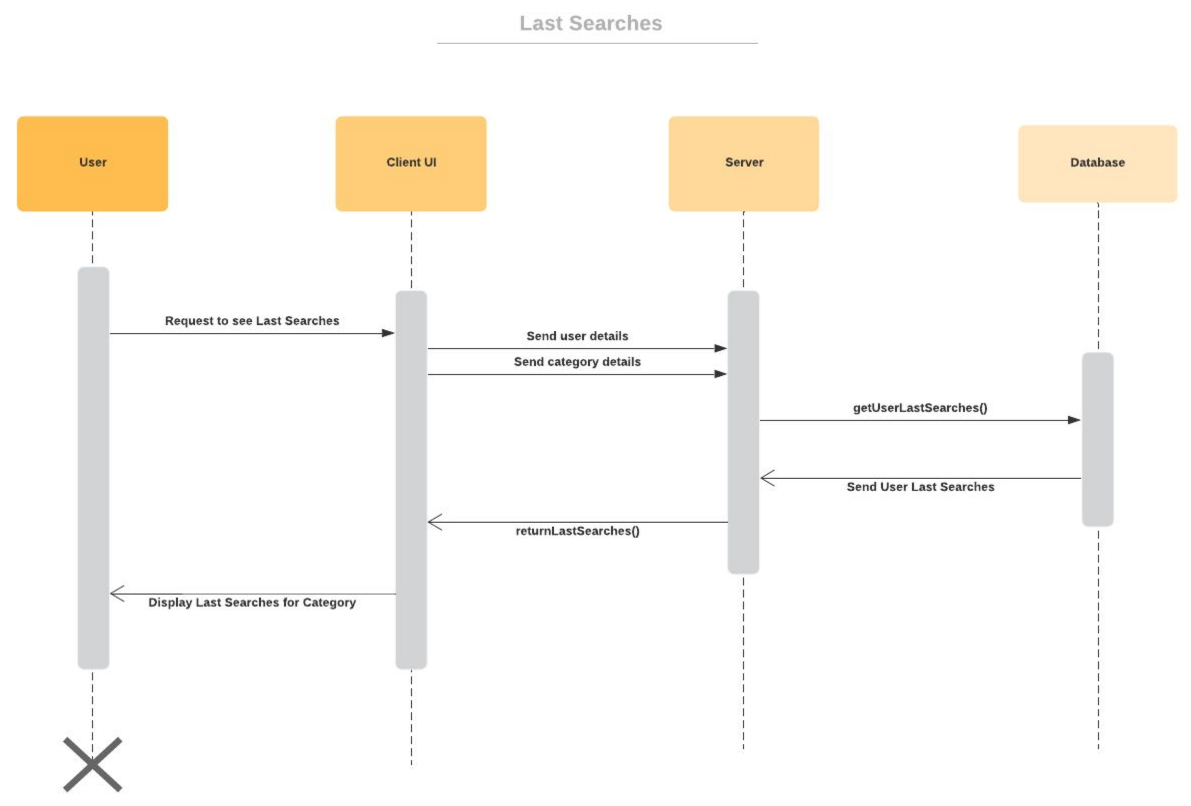


Figure 7: Sequence Diagram for Last Searches

The User will request to see Last Searches to the Client UI. The Client UI will then forward user details and category details to the Server. Based on these details, the Server will query the Last Searches from the Database. The results will be sent to the Client UI which will display it to the User.

Trending Searches

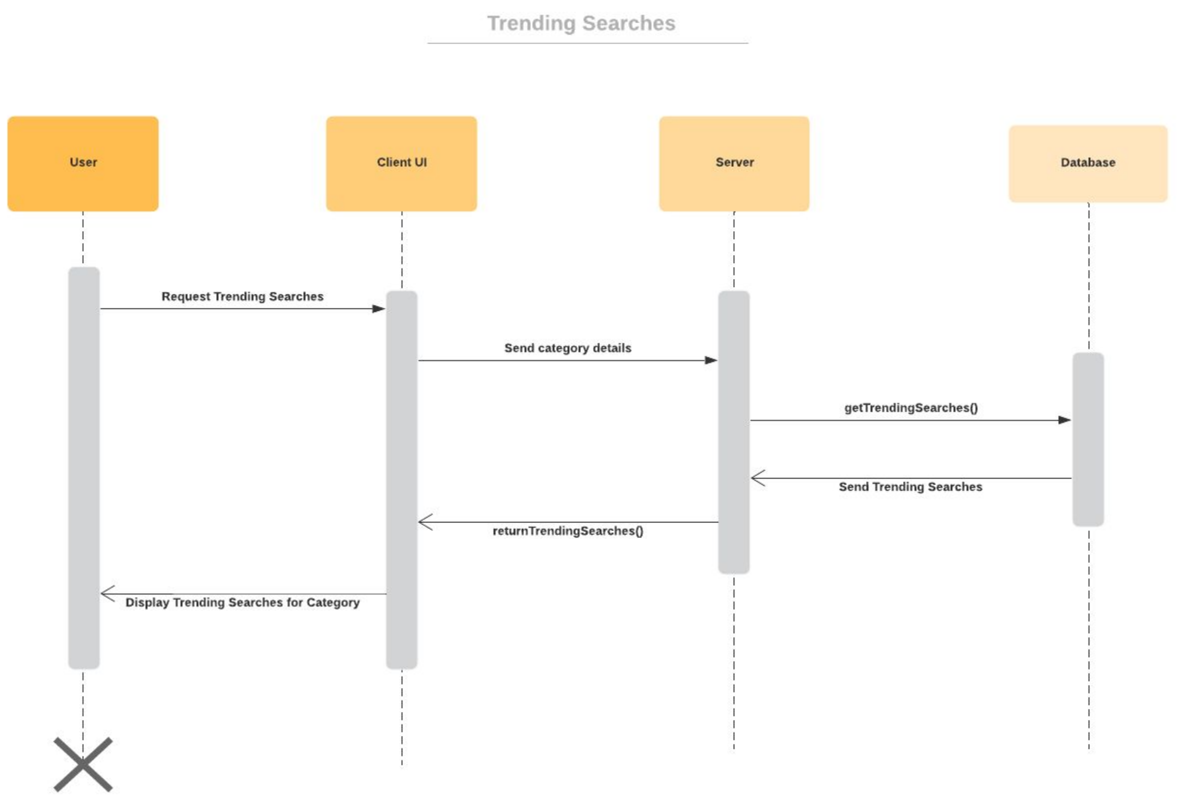


Figure 8: Sequence Diagram for Trending Searches

The User will request to see Trending Searches to the Client UI. The Client UI will then forward user details and category details to the Server. Based on these details, the Server will query the Trending Searches from the Database. The results will be sent to the Client UI which will display it to the User.

Program States

Mine

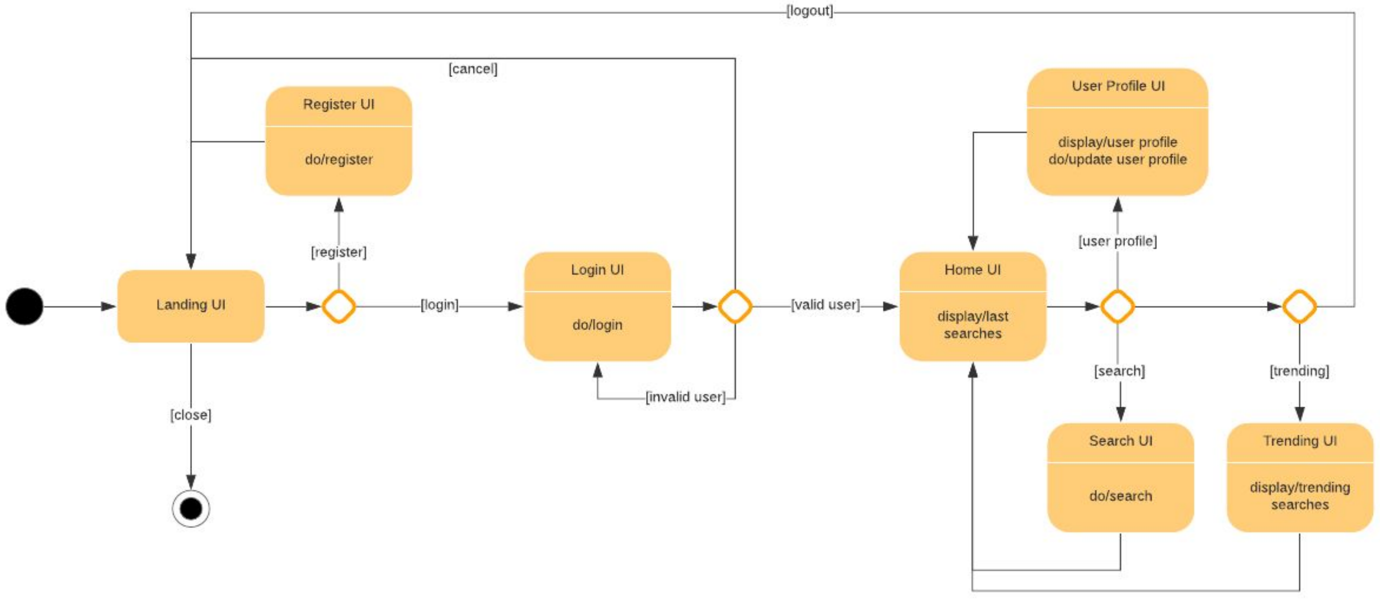


Figure 9: State Diagram

When the User first open the application, the initial state is the Landing UI. Here the User sees information about the application and the options to Register or Login. If the User clicks Register, the state of the application changes to Register UI. Here the User can register and then he will be redirected to Landing UI. If the User clicks Login, the state of the application changes to Login UI. Here the User can login or cancel. If cancelled, the state changes back to Landing UI. Otherwise, the state changes to Home UI where the User can see previous searches. If the User clicks profile picture, the state changes to User Profile UI where the User can edit User preferences. If the User clicks Search, the state changes to Search UI. If the User clicks Trending, the state changes to Trending UI. When the User clicks logout, the state changes back to Landing UI. The state of Mine closes finally when the User closes the browser.

UI Mockups

Registration

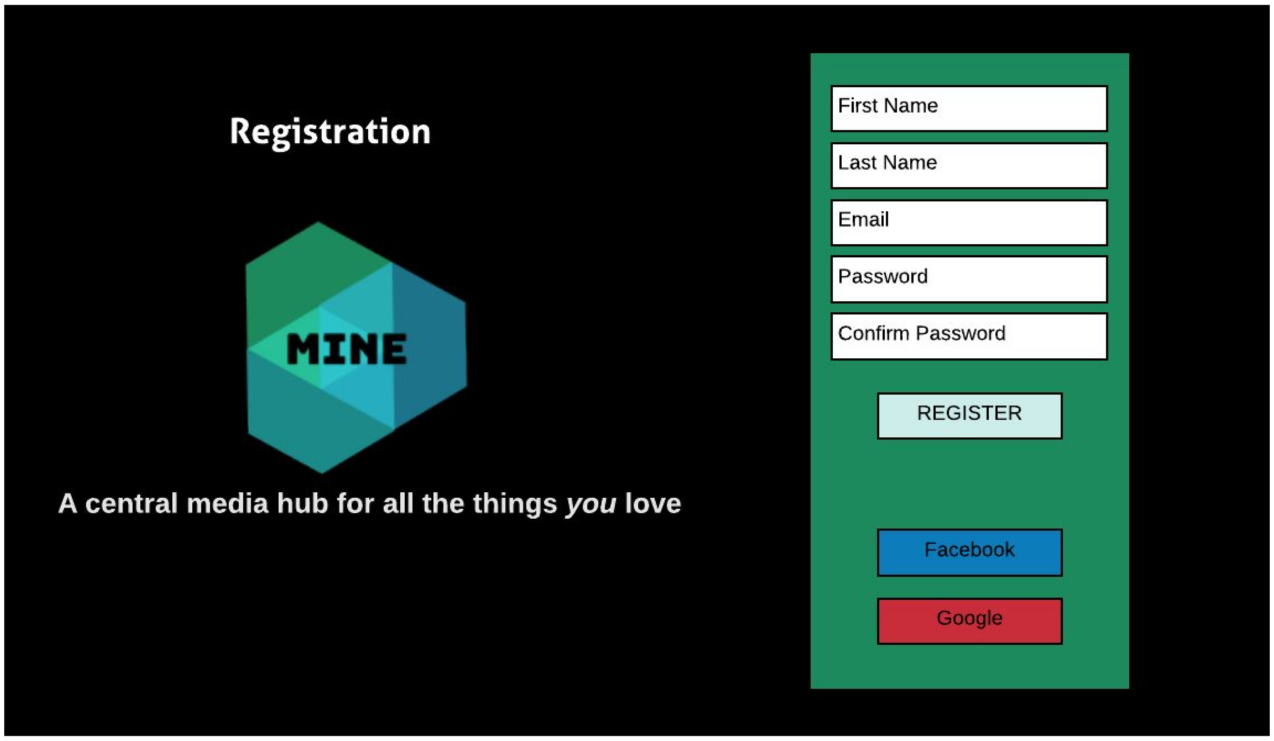


Figure 10: UI Mockup for Registration

Login

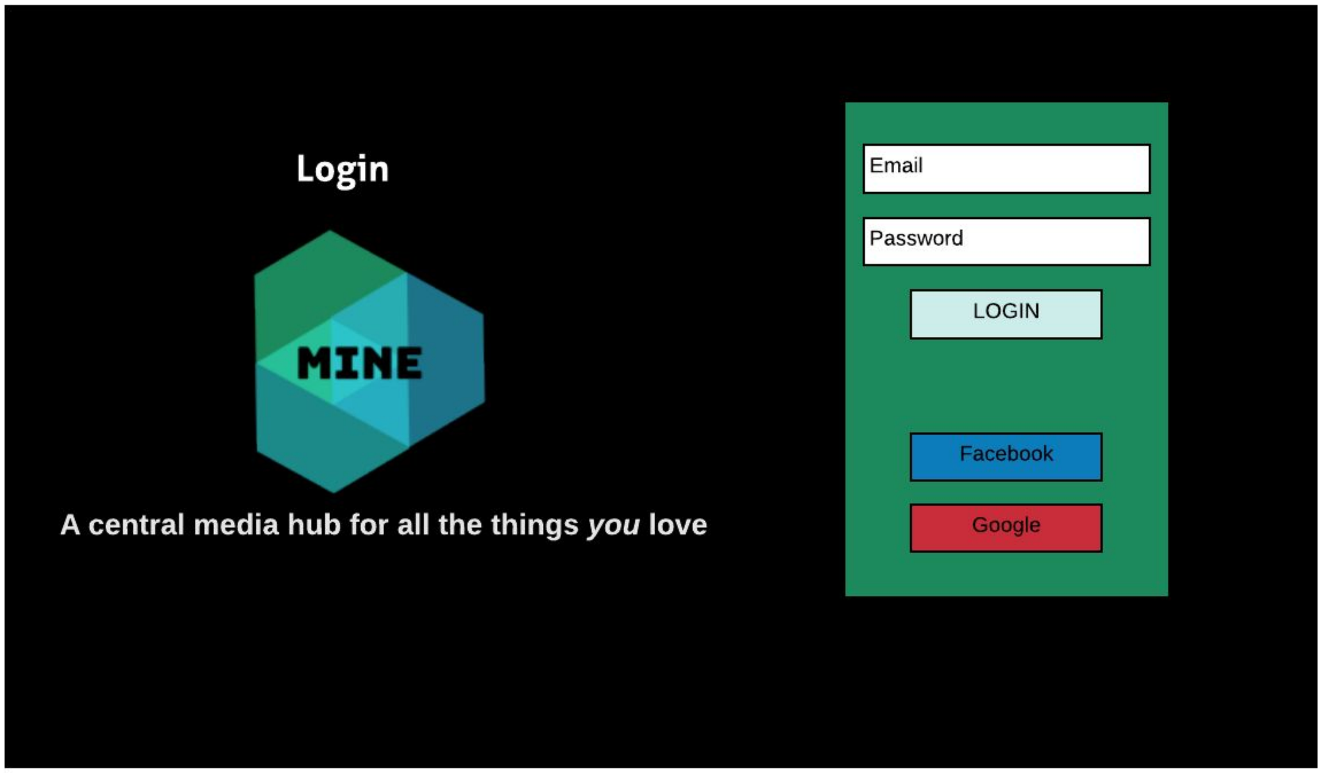


Figure 11: UI Mockup for Login

Forgot Password

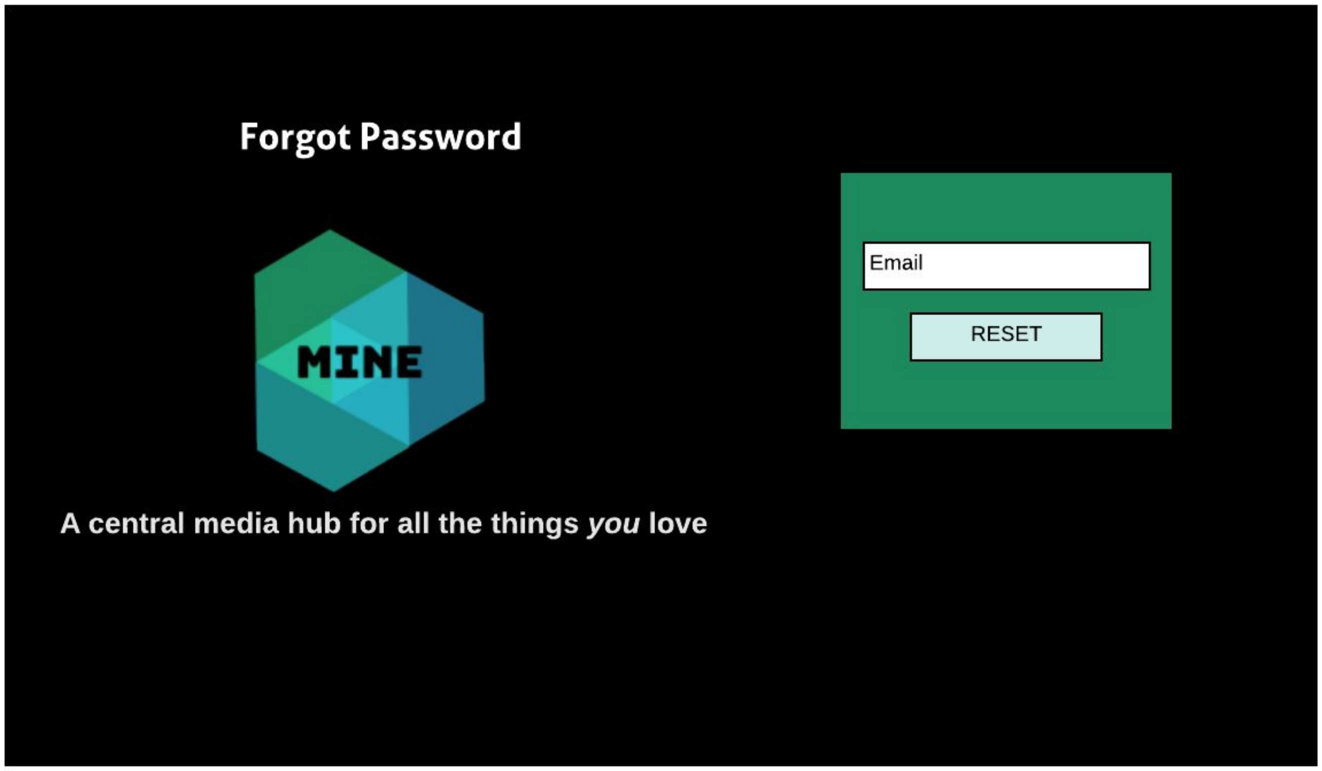


Figure 12: UI Mockup for Forgot Password

Home

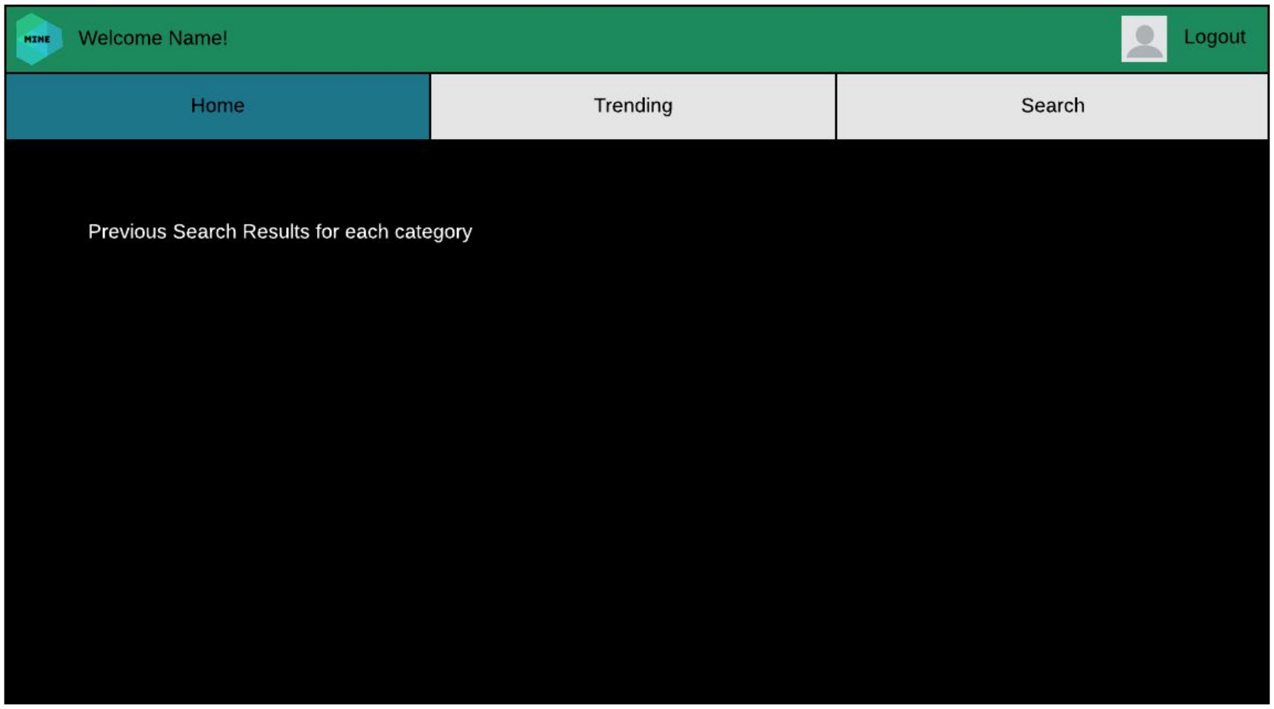


Figure 13: UI Mockup for Home

Trending

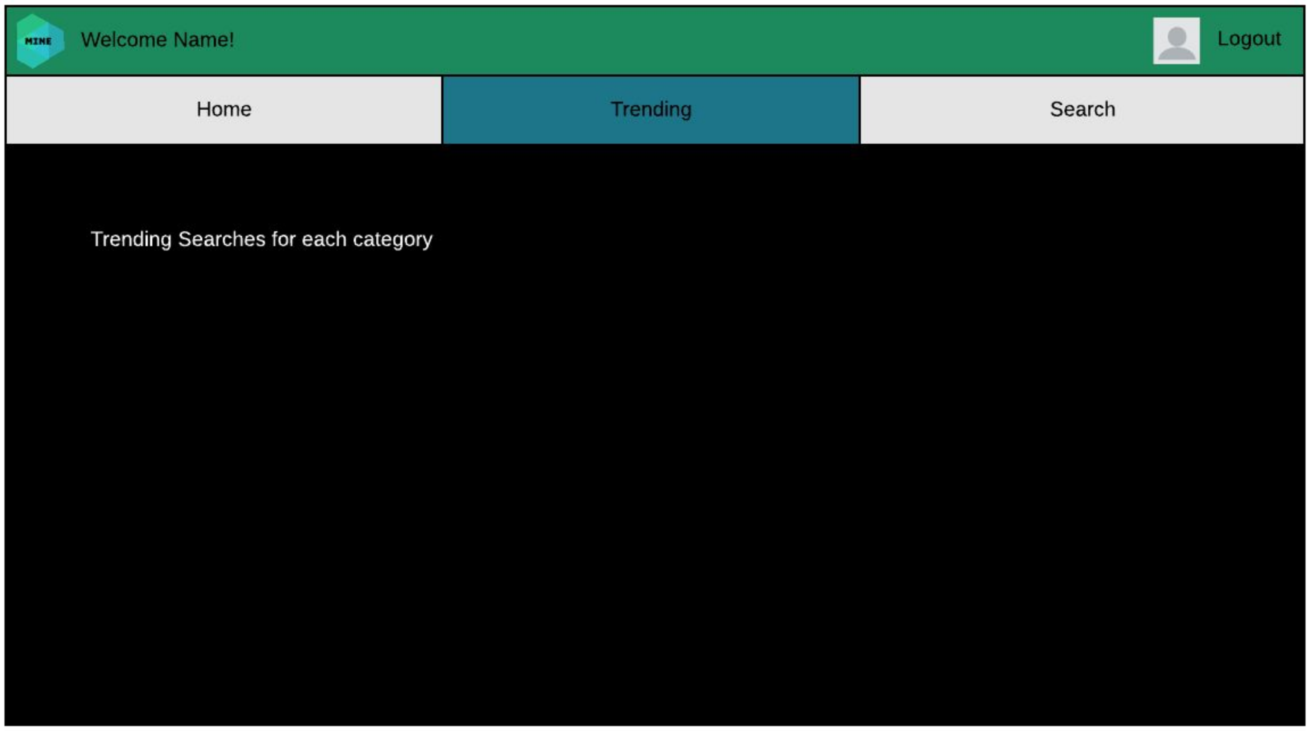


Figure 14: UI Mockup for Trending

Search

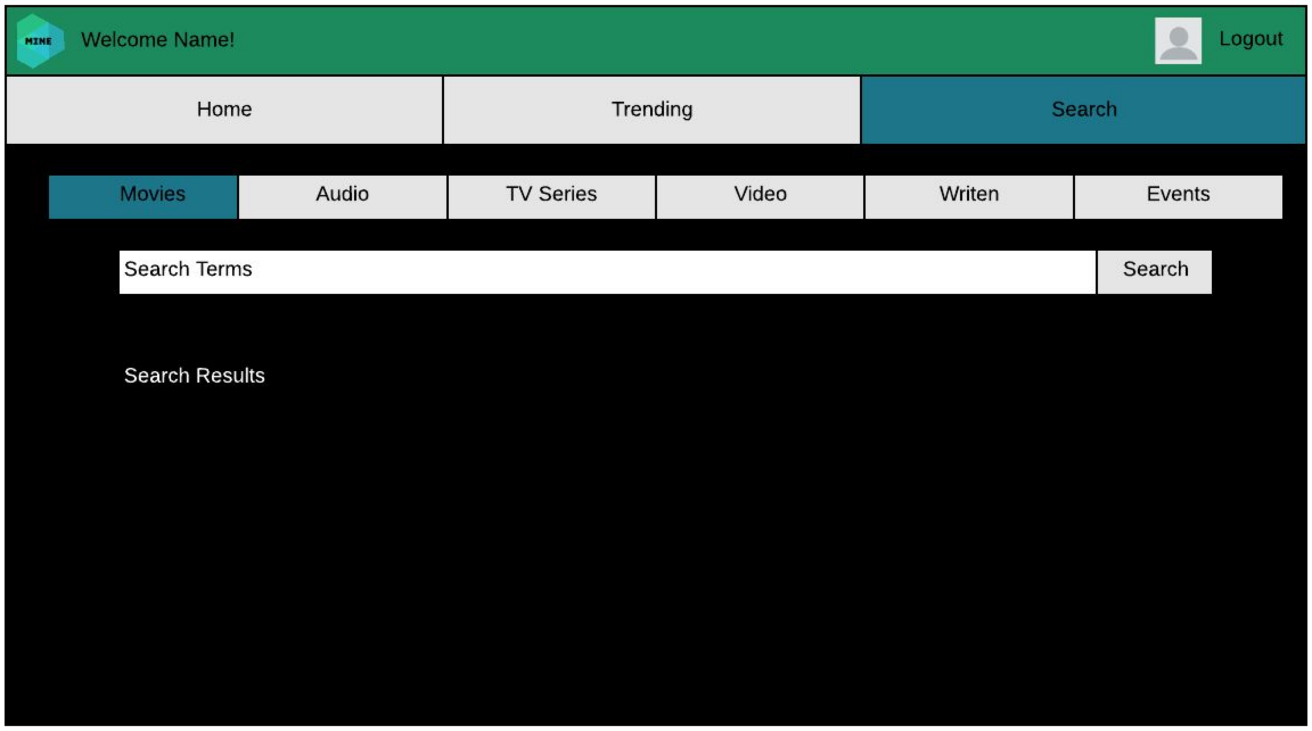
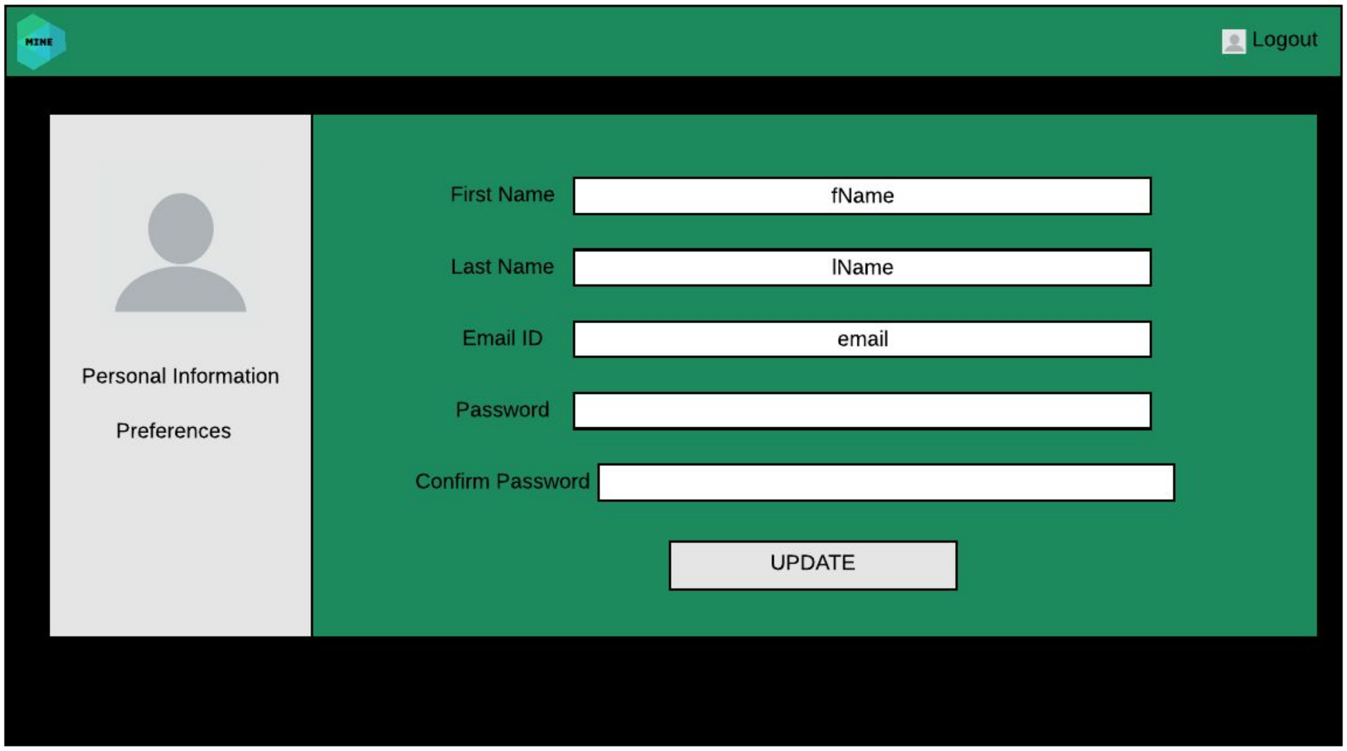


Figure 15: UI Mockup for Search

User Profile



First Name: fName
Last Name: lName
Email ID: email
Password:
Confirm Password:

Figure 16: UI Mockup for User Profile