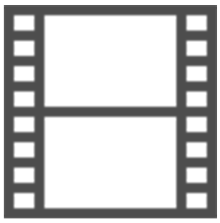


meetzam

Team 4 Design Document

Designed by meetzam-Dev Team:

Sean Chew
Junpu Fan
Zuyuan Fan
Yuting Guo
Qi Meng
Ling Zhang

Date: February 6, 2017

Index

1. PURPOSE

Has anyone ever found any difficulty finding the ideal movie and company to watch a movie with? Oftentimes, just the process of deciding what to watch and who to watch it with can be a time-consuming task. Many people want to watch movies, but nobody wants to watch a movie alone. Even though there are existing apps that recommend the ideal movie for everyone, there is no app out there that brings in the social aspect and allows users to make friends along the way. Think of our app as combining certain aspects of Yelp and Tinder.

We believe that being able to enjoy a good movie with good friends turns the outing into a great experience. By allowing users to find the ideal movie and hosting it with a bunch of friends, our product serves as a simple and convenient way of bringing people together through the classic event of watching a movie. Want to expand your social circle and meet new people? No problem! Our app offers a great function that allows users to match with other users that have the same movie preferences. Then, after chatting on our in-app chatting service, if both parties are satisfied, a movie date is set!

We have two main purposes that we aim to achieve with our app. The first is to connect friends and provide users a convenient way of hosting a movie event with their friends. The second is to match users with new people sharing the same movie interests that allows them to expand their social circles.

FUNCTIONAL REQUIREMENTS

- *Users can create and login to their account*

  As a user,

  - I would like to create and login to an account by using my email address.
  - I would like to create a username which allows other users to find and add me as a friend.
  - I would like to create and login to an account via a third-party site such as Facebook.

- *Users can find movies that are playing in the theater*

  As a user,

  - I would like to know what movies are on and will be out in the next few weeks.
  - I would like to see movies in a rank of popularity.
  - I would like to see short reviews and comments from my friends or public.
  - I would like to leave comments and rate movies I watched.
  - I would like to double tab the movie indicating that I want to watch that movie.

- *Users can choose to watch movies with a group of friends*

  As a user,

  - I would like to see what movies my friends want to watch.
  - I would like to select multiple friends whom I want to watch movie with.
  - I would like to start a group chat with those friends to discuss the details.
  - I would like to receive notifications when I am added to a new group.
  - I would like to keep a list of different active chat rooms.

- *Users can choose to match with new people to watch movies together*

  As a user,

  - If I choose to be matched with new people, I would allow those who have the same movie preferences as me to be able to see my profile.
  - I would like to see the profiles of users who have the same movie interests as me.
  - If I choose to be matched with new people, I would like to have the option of “liking” or “passing” when I see a new person’s profile.
  - I would like to chat with individuals whom I “like” and also “like” me.
  - I would like to keep a list of active chat rooms with users that I’ve been matched with.

- *Users can add other users to their friend list*

  As a user,

  - I would like to have the option of connecting my account to other social media and adding friends on other social media to my friend list on meetzam.
  - I would like to send invitations to my friends on other social media that do not have meetzam account.
  - I would like to receive a notification if I receive a friend request from another user.
  - I would like to “unfriend” a friend if necessary.

✤ NON-FUNCTIONAL REQUIREMENTS

- *Server Side*

  As a developer,

  - I would like to configure and provision AWS resources to support Lambda functions.
  - I would like to deploy Lambda functions onto AWS environment.
  - I would like to have a development environment and a production environment.
  - I would like to provide handlers for client requests.

- *Client Side*

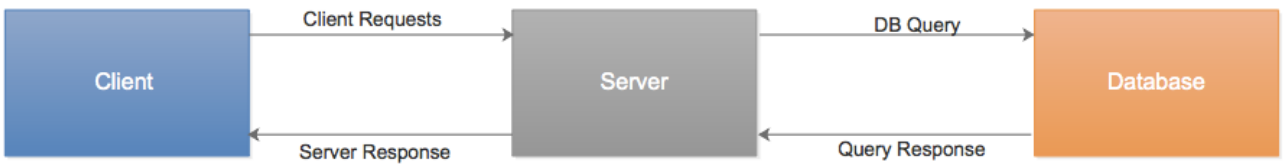
  As a developer,

- I would like the application to run on iPhone.
- I would like to communicate with the backend via SDK and/or API.
- I would like to provide a user friendly UI.

2. DESIGN OUTLINE

✤ HIGH LEVEL OVERVIEW

Our application will use a client-server model. The server will respond to client requests by retrieving relevant information from the database and sending them to the client. The client will then parse and display the data to the users. The figure below demonstrates a high level overview of the system.



- *Client*
  - Client sends a request to the server via SDK and/or API.
  - Client retrieves response from the server via SDK and/or API.
  - Retrieved information will be parsed and then displayed to the user.

- *Server*
  - Event handlers triggered by client requests.
  - Event handlers validate and parse the client requests.
  - Server communicates with the database and query relevant data which is then sent to the client.

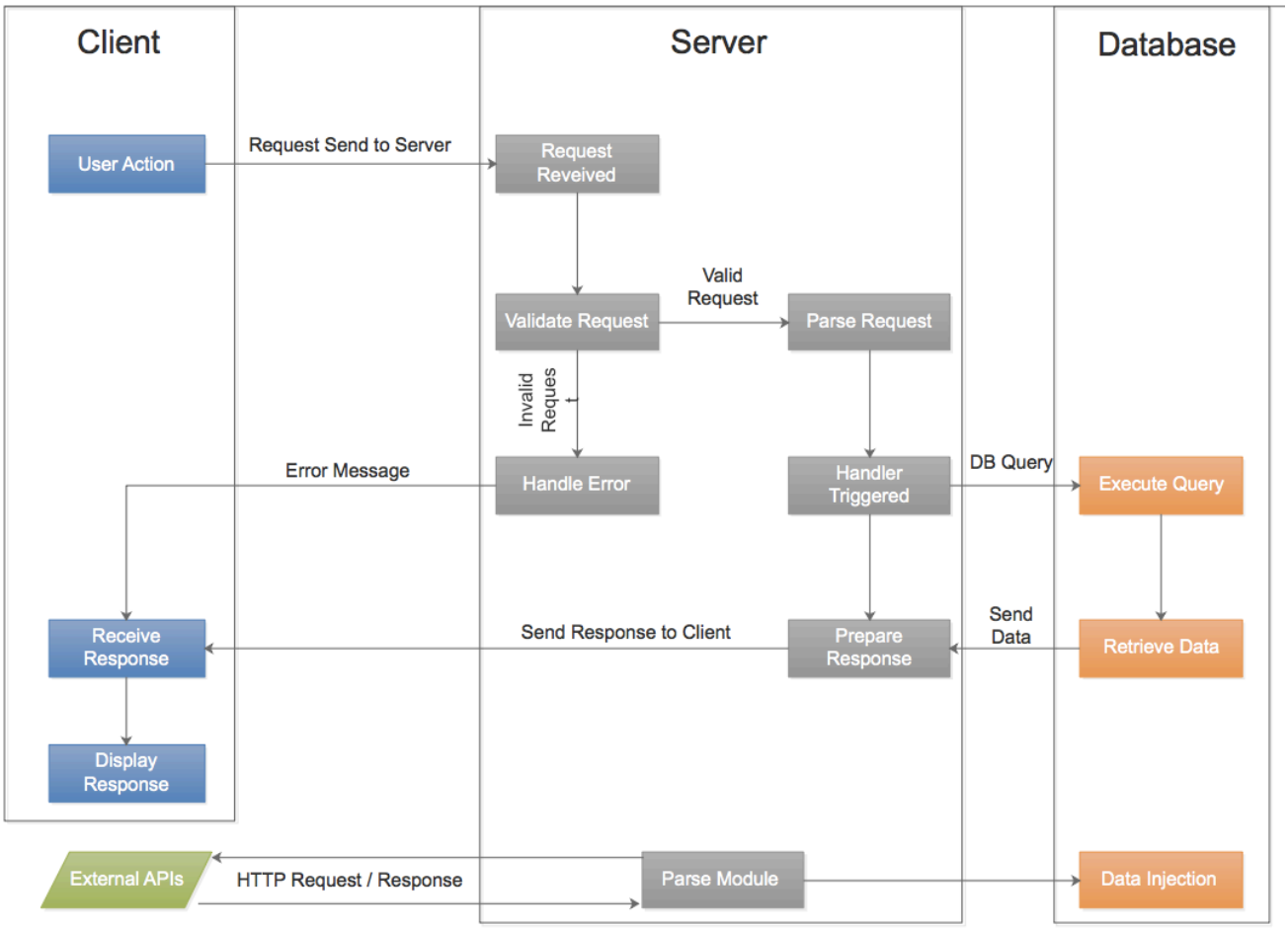
- *Database*
  - Database stores all relevant data our application will use.
  - Database responds to server queries and sends relevant data to the server.

✤ WORKFLOW OF EVENTS

Below is a demonstration of a typical workflow with events triggered by user action. The workflow begins with some user action with the client, such as a button being pressed, a page being swiped etc. If such a user action requires additional data from our backend system, then a corresponding request will be sent from the client to the server.

The server receives the incoming request and validates it. If the request contains an error, an appropriate error message will be sent to the client. If the request was properly formatted with the appropriate permissions, then the parser will validate and parse the request and a corresponding event handler will be triggered. The event handler then sends a query to the database. The database then executes the requested query and sends the relevant data back to the handler. The handler receives the database response, generates a response and sends it back to the client. The client then properly displays the information to the user.

There will be a number of parse modules that are running on the server. These parse modules request relevant raw data from the various external APIs periodically and populates the database to support client requests.



3. DESIGN ISSUES

✤ FUNCTIONAL ISSUES

- *Issue: Will the users be matched with a new person randomly?*

  - Option 1: Users are matched with a new person randomly
  - **Option 2: Users are matched with a new person according to their selection**

Reason: We choose to let the users select new people based on their movie interests. Users tend to be more willing to use the app when they are allowed to have a choice in who they pick as their movie buddies.

- *Issue: When should users be added to the matching pool to meet new people?*

  - Option 1: Once the user "likes" a particular movie
  - **Option 2: Once the user selects the "Matching" icon indicating that he/she wants to start the matching process with new people**

Reason: We made this decision because of security reasons. We don't want to expose users' profiles to people who are not on their friend lists if they are not interested in watching movies with new people. We will include in our terms and conditions that once users choose to match with new people, they agree to expose their profiles to people who have the same movie preferences and are willing to match with new people too.

- *Issue: How should users add friends?*

  - Option 1: By username only
  - **Option 2: By username and have the option of importing friends from other social media**

Reason: It would be very convenient for users to add friends from other social media if they choose to link their accounts. We also considered that some users might not want to add all their friends on other platforms to meetzam. So our solution is to show users a list of their friends from other social media but it is up to them to decide whether or not to add them on meetzam.

- Issue: What should be the source of popularity ranking and reviews of movies?

  - Option 1: From other platform
  - Option 2: Solely dependent on our own users
  - **Option 3: Combination of other platform and our user base**

Reason: At the initial stage of our application, there might only be a few users. Thus importing comments from other platforms is necessary for our users to decide on which movies to watch. However, as our user base grows larger and users start leaving reviews, we could rank the movies based on a combination of other platforms' and our reviews.

- *Issue: Should users be able to see people whom they "passed" in the matching process?*

  - **Option 1: Yes, but those people are at the end of the matching pool**
  - Option 2: No, once the user pass a person, he/she will never be able to see that person again

Reason: Considering that people might accidentally "pass" on other users, we think it is not user friendly to completely delete that person from a user's matching pool. Having those "passed" people at the end of the queue prevents users from seeing the same person repetitively.

- *Issue: How will movies be displayed to the users?*

  - Option 1: The movies are displayed in a scrollable table with posters and basic information
  - **Option 2: The movies are displayed using a single poster and additional information is displayed on another page**

Reason: Compared to a scrollable table, a single poster is much simpler and directly gives a great first impression to new users.

- *Issue: Should the movies that are no longer showing be kept in the list of "liked movies" and be considered in the matching process?*

  - Option 1: No. Movies that are not showing are removed from database.
  - Option 2: Yes. Movies in different showing status are considered to behave equally.
  - **Option 3: Yes. Movies that are not showing reside in our users' list of liked movies, but are not directly used in matching process.**

Reason: By keeping the "liked movie" history, users can have a richer profile and keep track of other users' interests as well. Also, since the order of displayed individuals during the matching process is based on the number of common liked movies currently in theaters, when there is a tie, the common movie histories will be used as a tiebreaker. This way, we will provide users with more precise matching results.

✤ NON-FUNCTIONAL ISSUES

- *Issue: What platform will the app be on?*

  - **Option 1: Mobile**
  - Option 2: Web

Reason: Considering that the purpose of this app is for mobile recreational purposes, users are unlikely to have their laptops with them. It is more likely for them to carry phones by their sides as a form of daily habit. While meeting and hanging out with old or new friends, users always want to use a handy and simple app in their phones. Therefore, we came to a conclusion that our app should be built on a mobile platform.

- *Issue: What IOS development language will we use for the app?*

  - **Option 1: Swift**
  - Option 2: Objective-C

Reason: We choose to use Swift as client side language under several considerations. Swift is generally a more intuitive and expressive language compare to Objective-C, thus learning Swift is easier. Also, Swift Introduced playground, which enable developers to test new algorithms or graphic flows with only about 20 lines of code, instead of creating an app. Moreover, Swift as a language was created with the purpose of creating apps on Apple platform, and the Swift community is still actively improving the language. Thus it is reasonable to assume Swift will be the go-to language for creating any Apple App in the future, therefore it is worthwhile to learn this language.

- *Issue: What is our choice of backend service for the app?*

  - Option 1: Physical Server
  - **Option 2: Amazon Web Services**

Reason: By building the backend on matured platforms such as AWS, we leverage many AWS services that make our infrastructure infinitely scalable, and resolve many problems that arise from a traditional server-based architecture such as network and server management, compatibility and dependency management, and inefficient use of resources.

- *Issue: What type of database will we use for the app?*

  - Option 1: MongoDB
  - **Option 2: DynamoDB**

Reason: By building the backend on matured platforms such as AWS, we leverage many AWS services that make our infrastructure infinitely scalable, and resolve many problems that arise from a traditional server-based architecture such as network and server management, compatibility and dependency management, and inefficient use of resources.

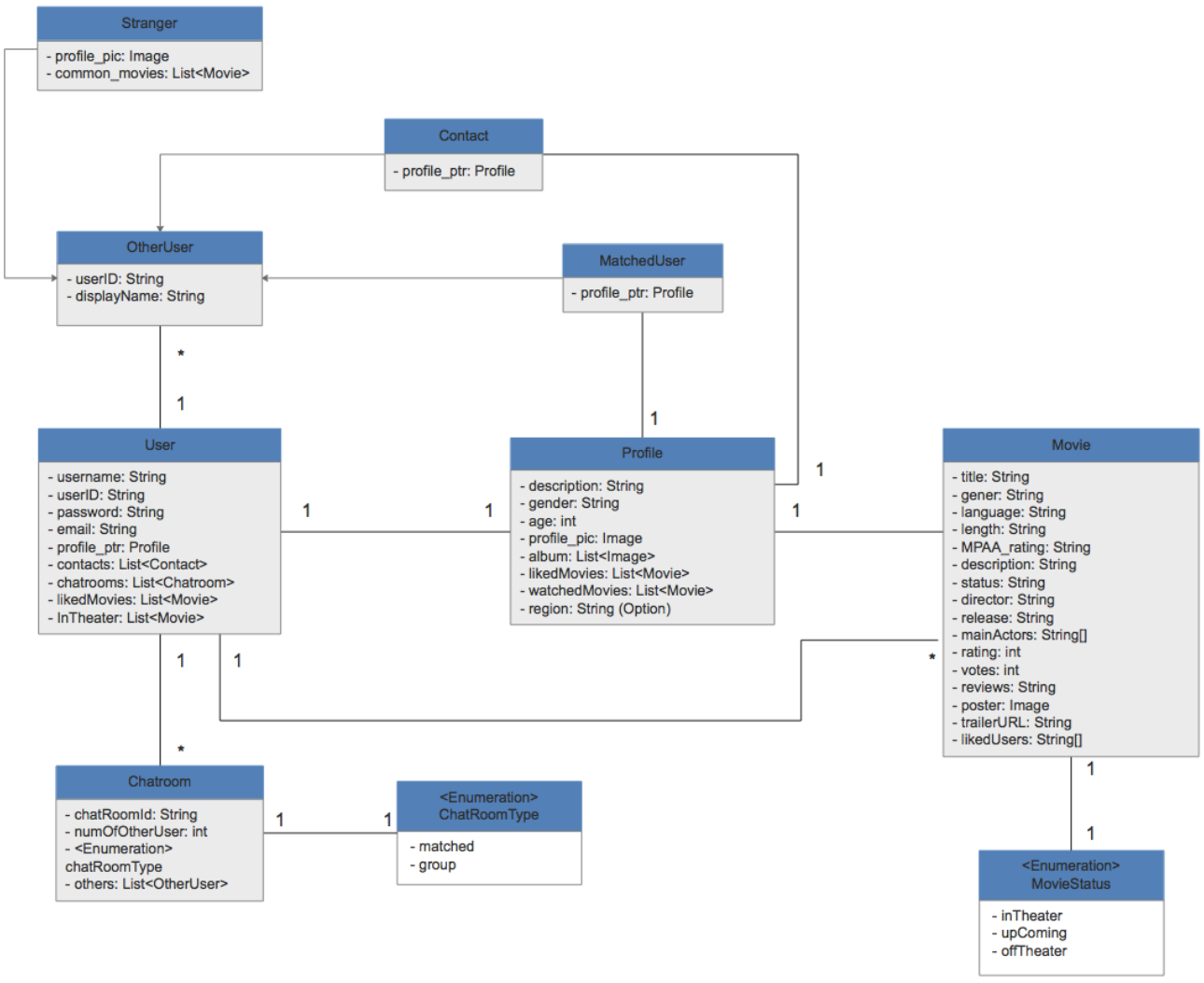
- *Issue: How is our app going to keep the movie data up to date?*

  - Option 1: Users will manually refresh the movie data
  - **Option 2: Our app will automatically update the movie data everyday**
  - Option 3: Refresh the movie data when users enter the movie main page

Reason: Since it is common for movies to be released every two to three days, there is no need to refresh the movie data in response to user requests, doing so not only slows down the process of satisfying requests, but also makes lots of redundant API calls. Instead, we adopt the idea of pre-fetching. Since we know that the type of data we need are generally static, we can choose to fetch data periodically into our database. During client request, we simply satisfy the request from our database, such practice will boost overall application performance and reduce waste of resources.

4. DESIGN DETAILS

✤ CLASS DESIGN



✤ DESCRIPTIONS OF CLASSES AND MODELS

- *User:*
  - Represents the current user.
  - Created when a user first signs up successfully.
  - Contains authentication information of the user, such as username, password, and email.
  - Contains current user's related users, such as contacts, matched users, and potential matching options.
  - Contains current user's pointer to his or her profile.

- *OtherUser:*
  - Represents the parent class of other users except the current user.
  - Created when the current user establishes a relationship with another user.
  - Has subclasses: contact, matched user, and stranger.
  - Contains the base information, including user id and display name.

    - *Contact:*
      - Represents the contacts of the current user.
      - Created when the current user add someone to his/her contact list.
      - Contains a pointer to profile class, meaning that the current user could see the contact's profile.

    - *Matched User:*
      - Represents the matched users of the current user.
      - Created when the current user is matched with a stranger.
      - Contains a pointer to profile class, meaning that the current user could see the contact's profile.

- *Stranger:*
  - Represents the matching option of the current user.
  - Created when the current user chooses to start matching with strangers.
  - Contains only profile picture and a list of movies that the current user and the stranger both like.

- *Profile:*
  - Contains all the detailed information of a user, including description, gender, age, profile picture and region.
  - Contains a user's photo album, liked movies that are currently showing, liked movies that are no longer showing.

- *Chat_room:*
  - Created when a user starts a group chat with his/her contacts or when a user is matched with a stranger.
  - Updated when a user re-enter or leave the chat room.
  - Contains basic chat room information, including room ID, room type, number of people, and a list of pointer to the people in the chat room.

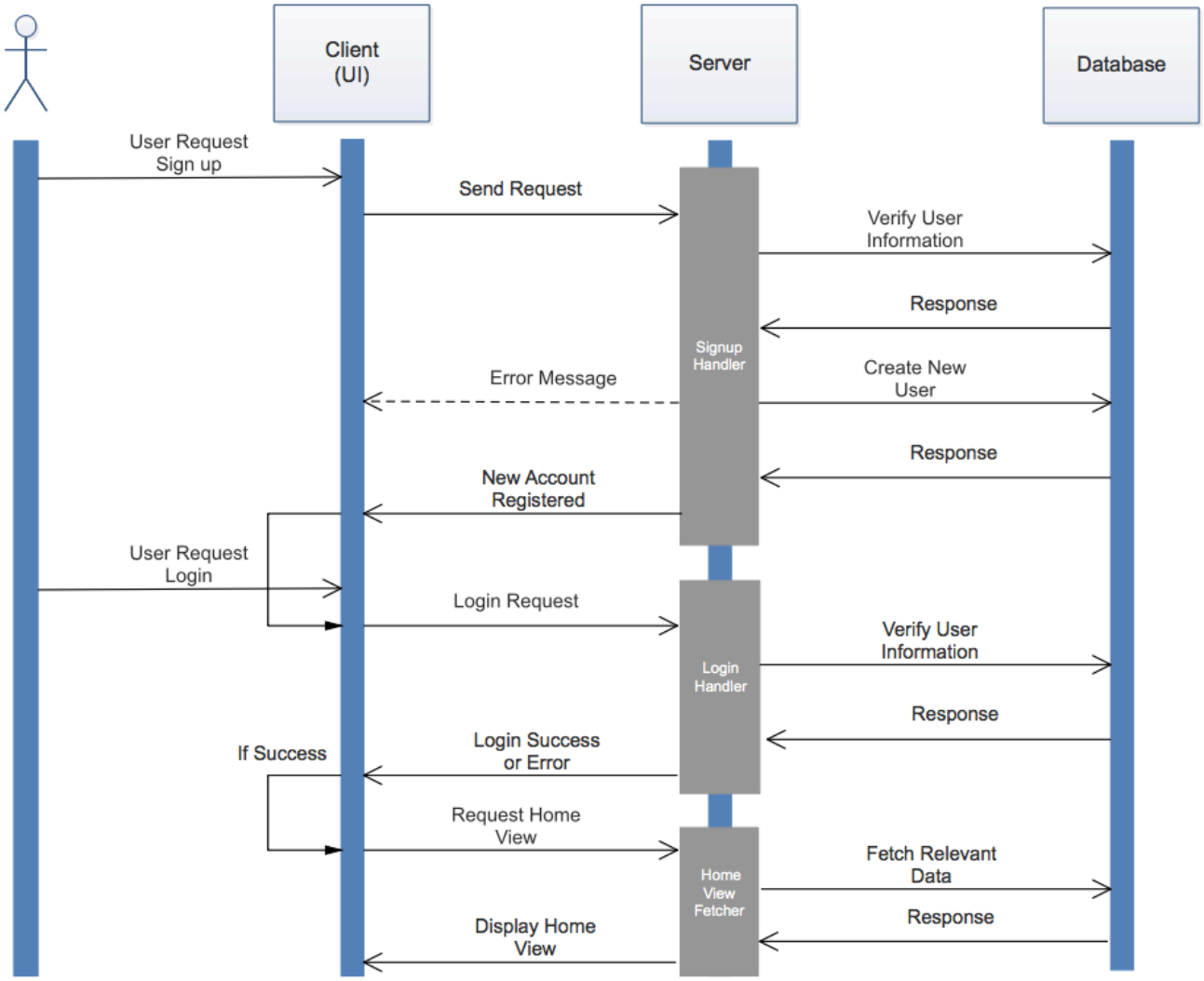
- *Movie:*
  - Represents the movies.
  - Created when the user first log in and the movie data is retrieved from API.
  - Updated when the movie data is kept up to date by querying API.
  - Contains all viewable movie information.

- *<Enumeration> chat room Type:*
  - Used to identify the type of chat room (with strangers or friends).
  - Created when chat room is created when a match established or being invited to a group chat.

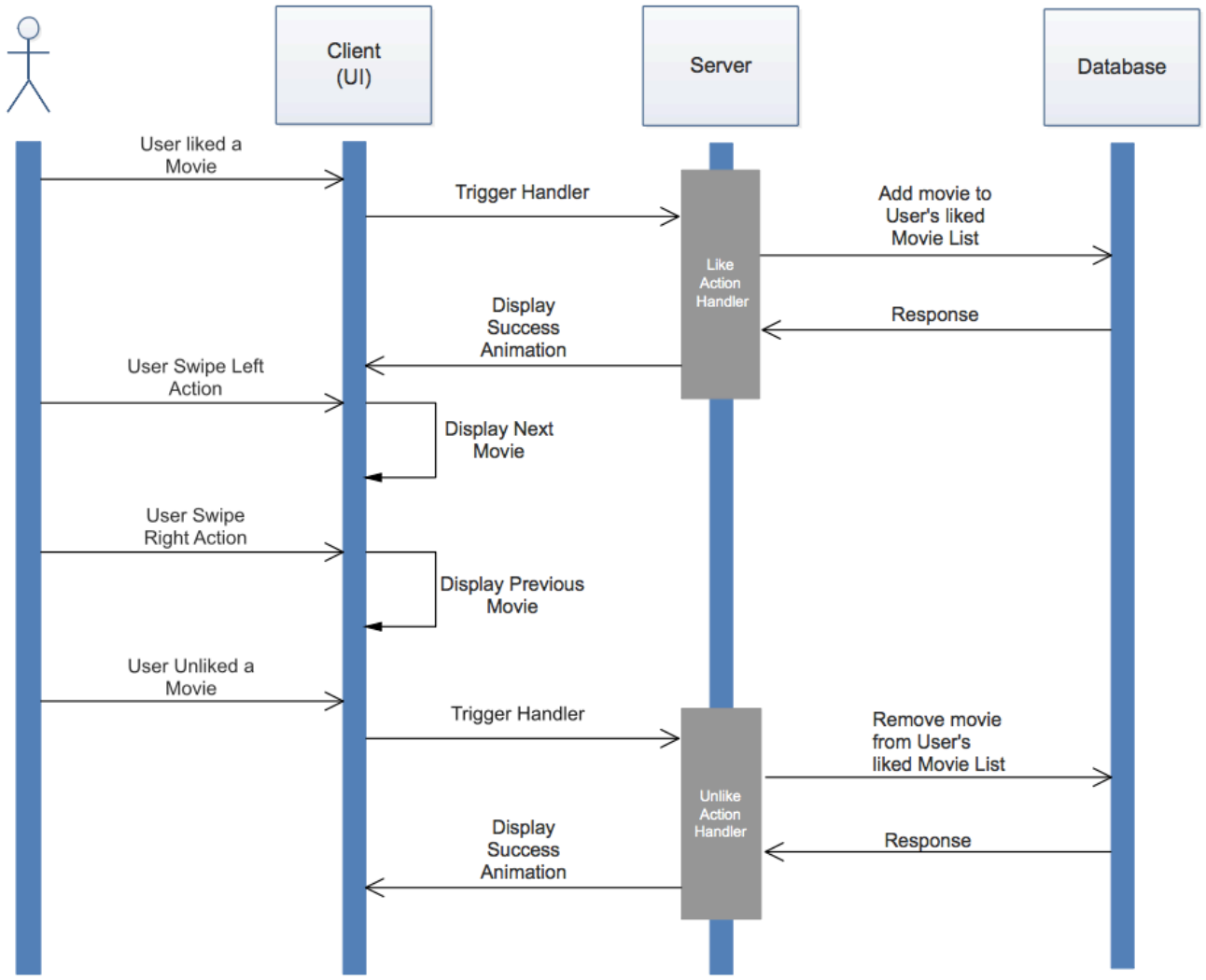
- *<Enumeration> movie status:*
  - Used to identify the type of chat room (with strangers or friends).
  - Used to identify the type of movies (on theater or out or upcoming).
  - Created when movie data is returned to client side.

✤ SEQUENCE DIAGRAMS

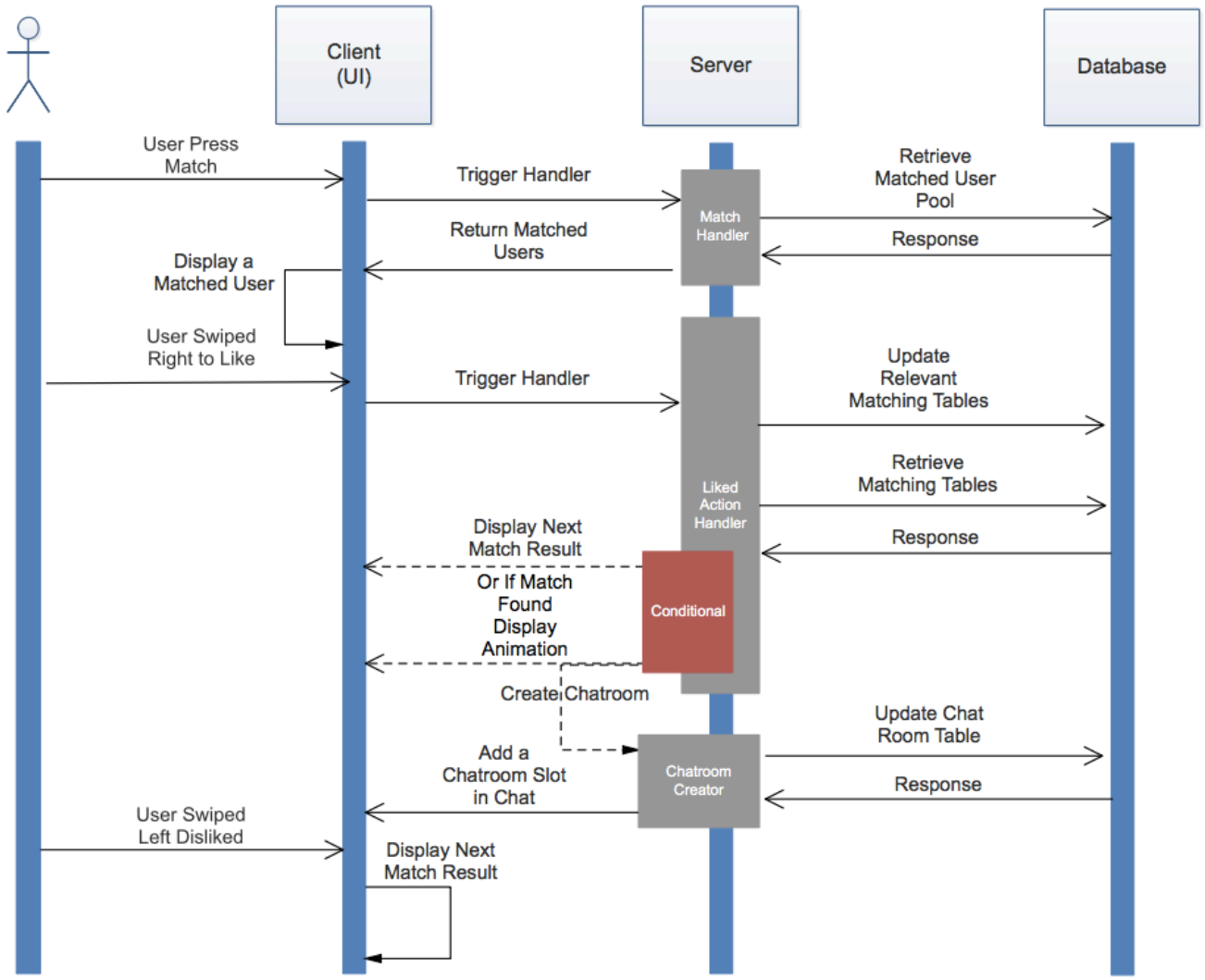
- *Sequence of events after the user starts the application*



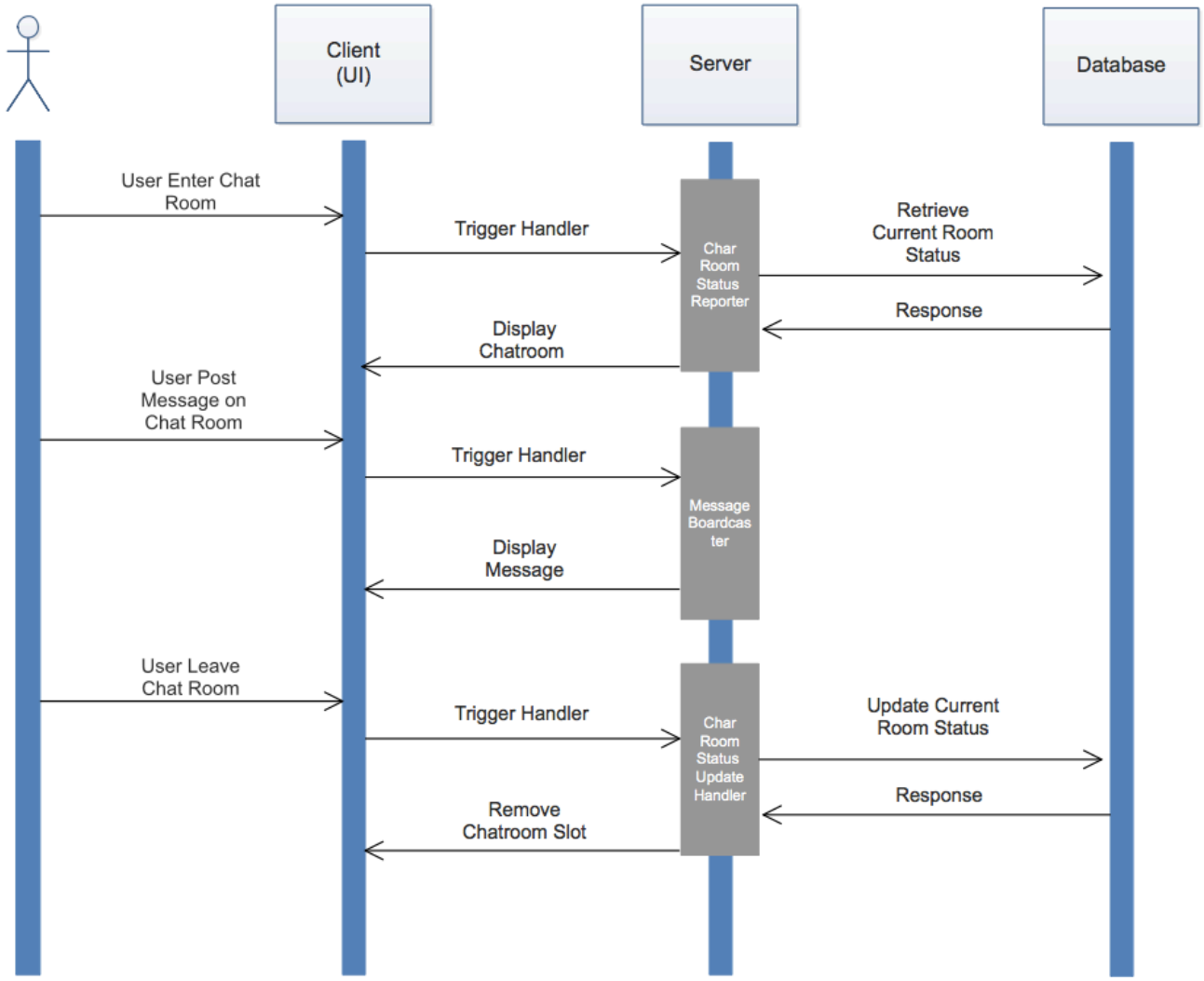
- *Sequence of events when the user browses and responds to movies*



- *Sequence of events when the user enters the matching process*

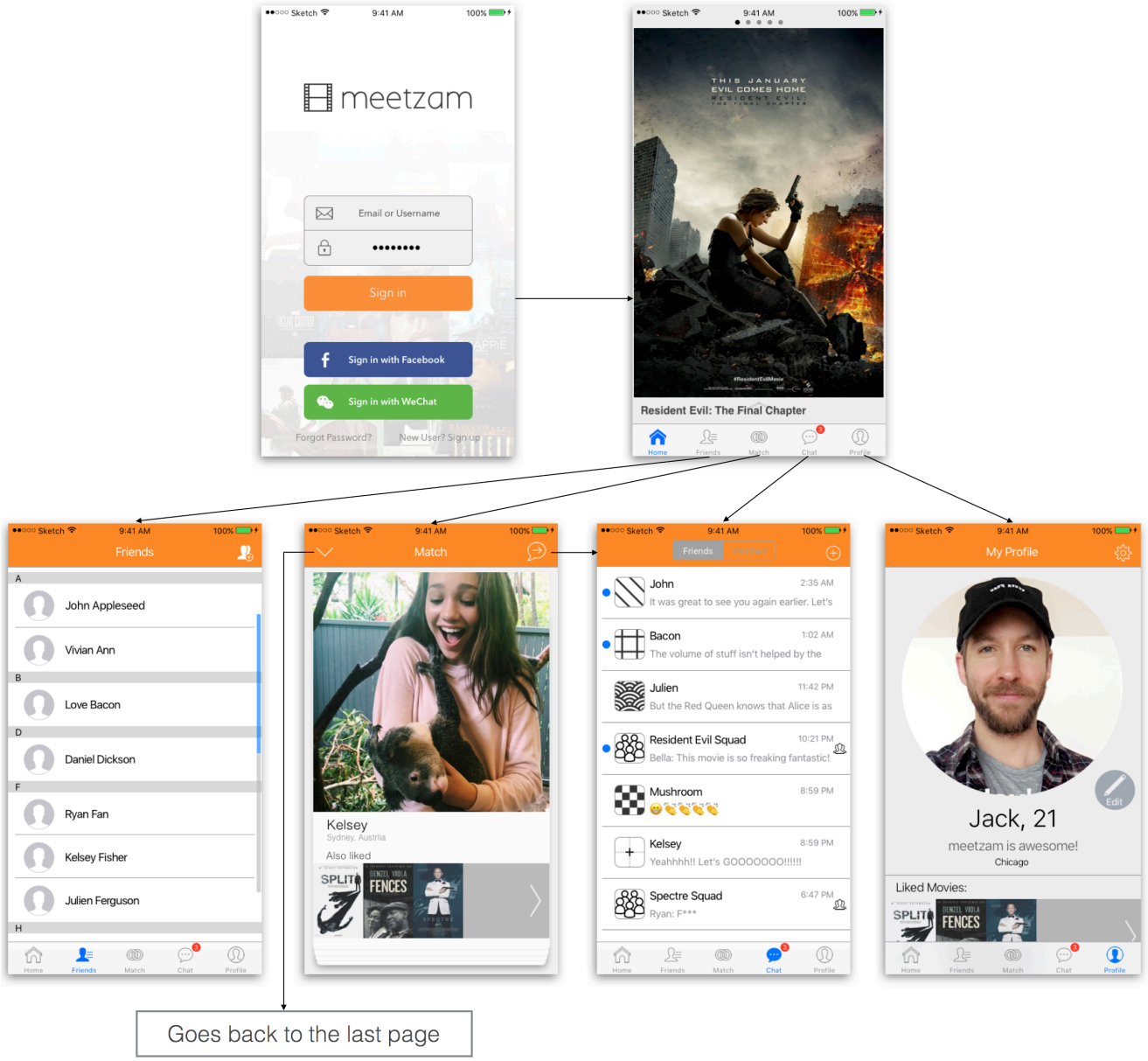


- *Sequence of events when the user enters a chat room*

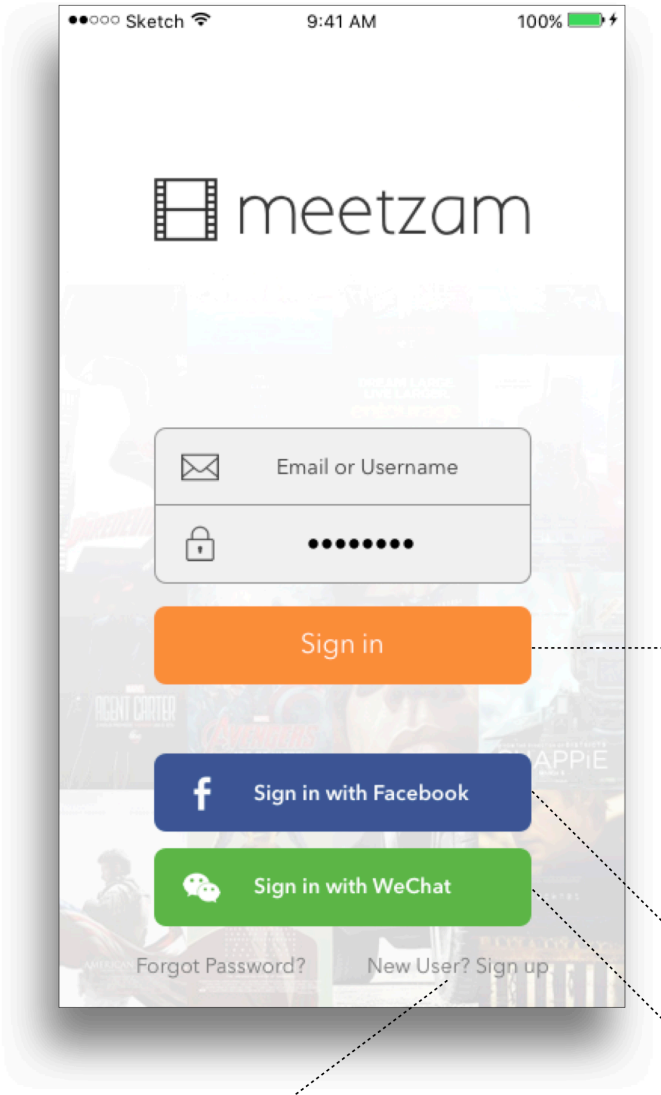


UI MOCKUP

UI Overview



Login Page



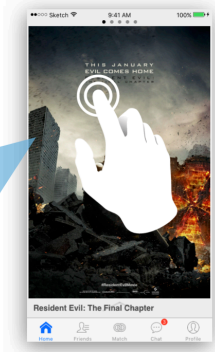
User's information is verified and will take user to the homepage

Sign-up button for new users who do not want to sign-in via Facebook or WeChat

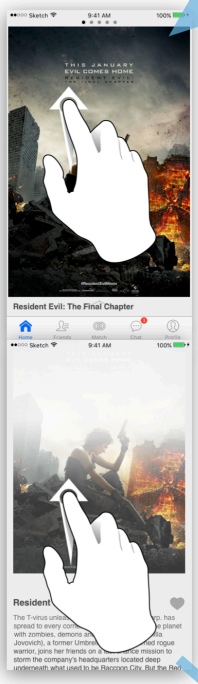
Alternative sign-in methods. Performs the same function as the basic sign-in button. Allows user the convenience of skipping the sign-up process

Home Page

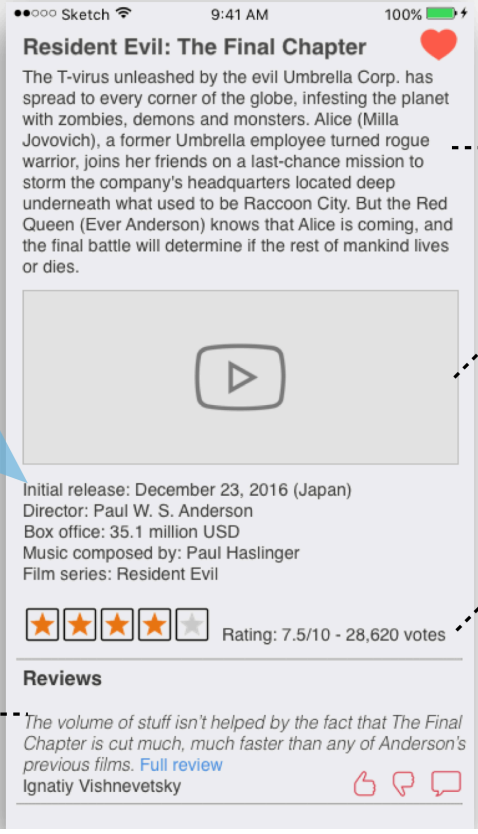
Double tap to like a movie!



Swipe up



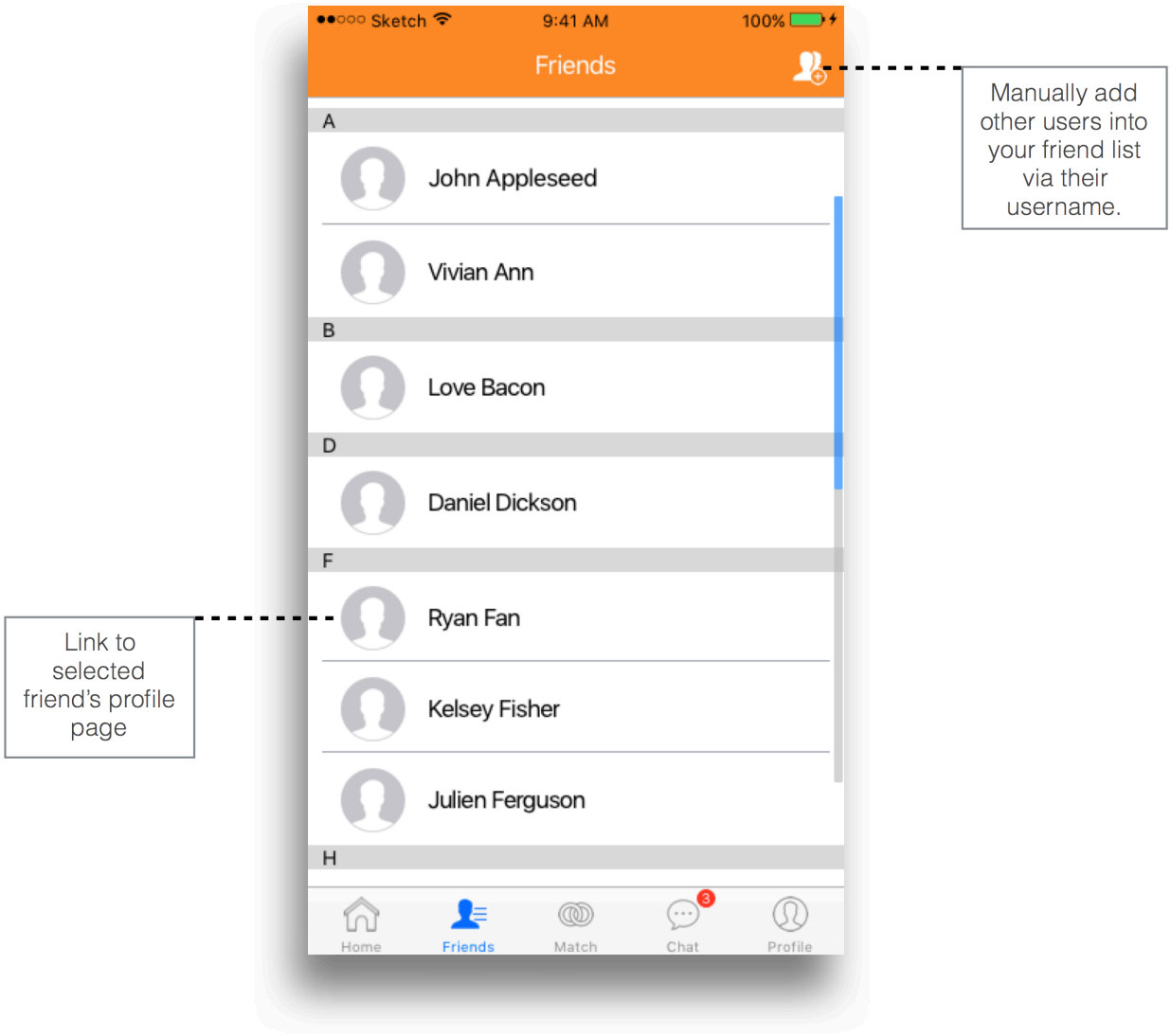
Swipe left/right to view the next movie



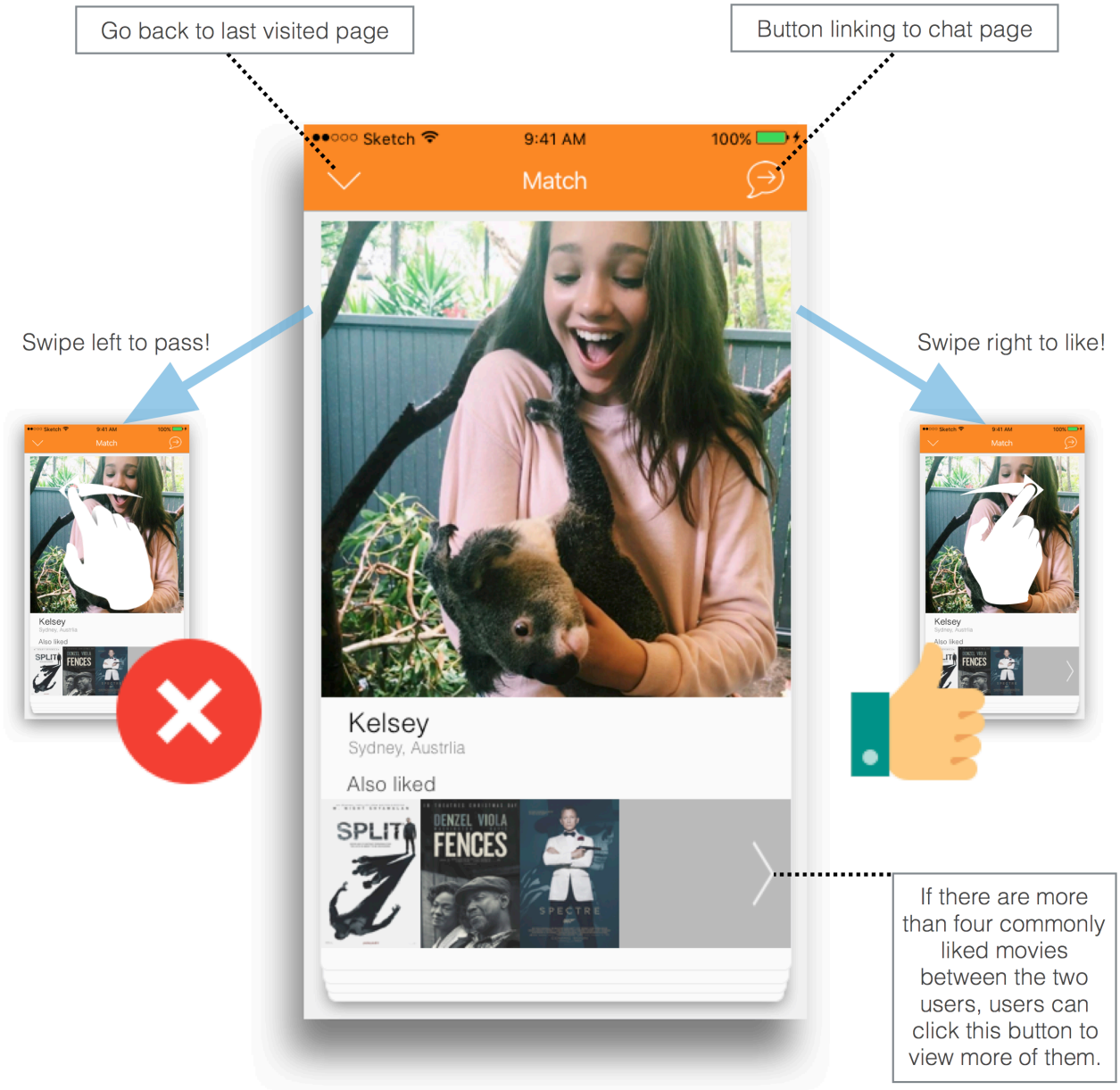
Movie information and trailer taken from external API

Reviews written by meetzam's users.

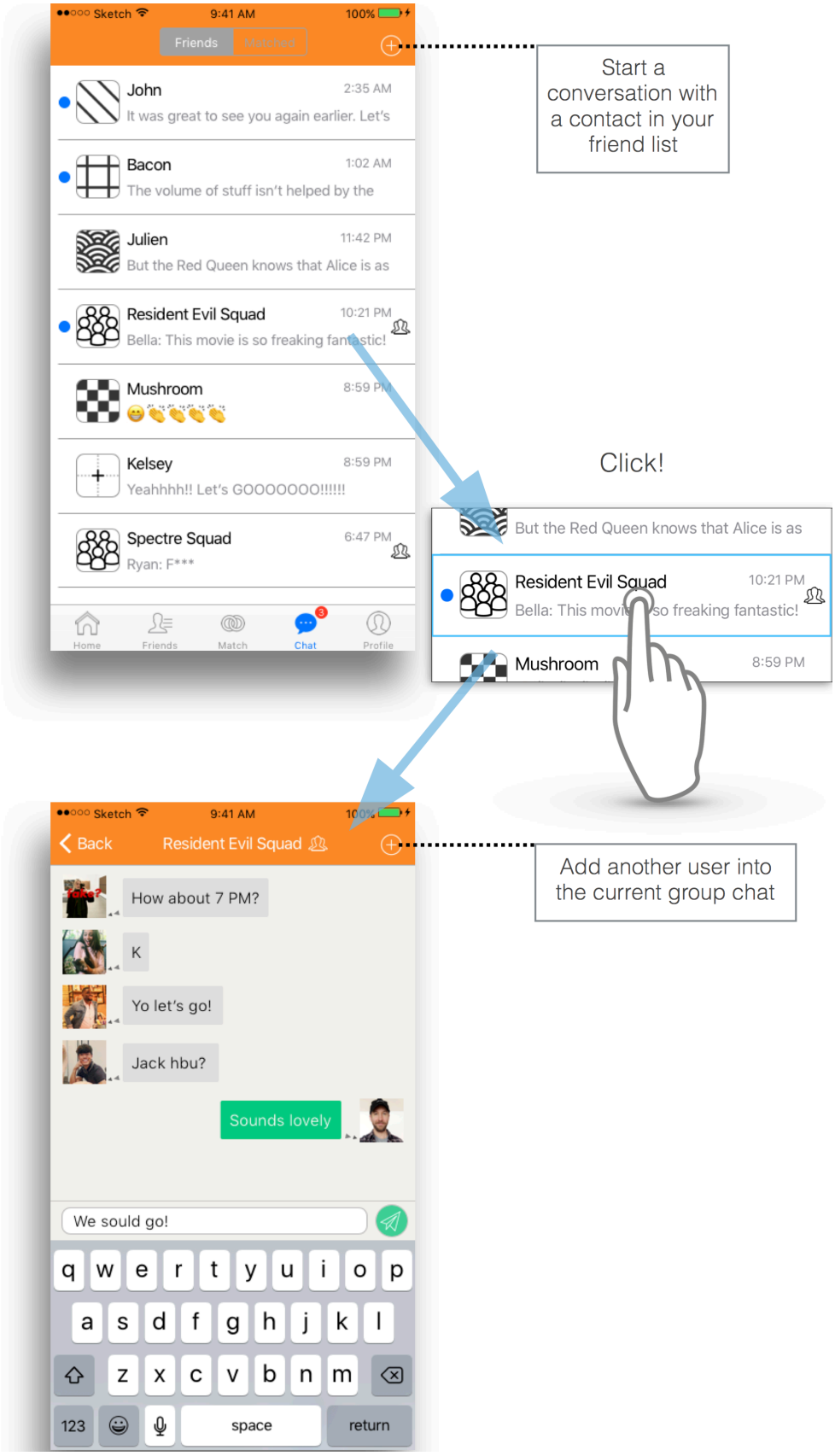
Friend List Page



Match Page



Chat Page



Profile Page