Featurama

A social network that features your life.

Ioannis Batsios
Michael McCullough
Christopher Thacker
Hieu Vo
Jamie Weathers

Senior Capstone (CSC490-02)
UNCG - Spring 2020

Table of Contents

Team Structure

Use-Case

Training Plan

Test Cases

Project Description

Overview of why it's needed (Ioannis Batsios)

Currently, the Internet is a bit of a mess. No one knows what is real? What is fake? Up, down, left, right, etc. If we want to know what friends are up to, we go to Facebook or Instagram or Twitter. If we want to watch something, we go to Netflix, or YouTube, or Hulu, or Amazon. If we want to sell ourselves to a company, we go to LinkedIn, or Monster, or Indeed, or Handshake. What if, instead of crawling all over the Internet to get what we wanted, we had the Internet come to us? What if we had our own domain that was fed with our specific interests, wants, and desires?

Welcome, Featurama. Featurama is a website that features an individual user's specific interest. A user creates a profile that asks for what the user is interested in. Based on the user's feedback, a website is created for the user that the user can populate it with what they want. It will marry the idea of social networking on Facebook with the pinned boards of Pinterest, but instead of boards, users can create lists that they share to create social commentary with other like users.

History (Ioannis Batsios)

One night my friends and I were talking about something we were all passionate about: movies. After the conversation had ended, and I'd gone home, I started to remember all the movies that I'd forgotten to tell them about. I went to Facebook and was going to post a message about it and tag them all in the post, but I didn't. I realized I didn't want a bunch of my other friends seeing the conversation, much like I don't like seeing what my friends are doing every second of the day that doesn't interest me. That's when I realized there was something wrong with Facebook. Sometimes I don't want to share something specific to every single person I know, but only to those that are interested in that topic. That's where the idea of Featurama was born.

Originally, Featurama was going to be a website that was only concerned with movies, but I realized I was missing out on untapped markets. Imagine if someone that was interested in quilting had a place to go? Or fishing? So, over the years, Featurama has blossomed and grown into a high-concept web site that features interests of all kinds of people.

System & Platform Requirements

Requirements Overview

Featurama is a web application, which means that it can be accessed from any modern device that can utilize the Internet. Therefore, the device being used would have to be connected to the Internet, whether that's in the form of cellular data, Wi-Fi, or with a wired connection, and must be connected to Featurama's web address through a supported web browser at all times during usage (see "Software Requirements" for supported web browsers). Devices that are officially supported - assuming they are set up to access the Internet - are desktop computers, laptop computers, tablets, and modern cell phones (i.e. smartphones).

Software Requirements

Currently, the only software requirements to use Featurama are regarding web browsers. Supported browsers are Internet Explorer 11, Mozilla Firefox, and Google Chrome. For the best performance, it is highly recommended to use the latest version of any of these web browsers.

Functional Requirements

- **Store and Retrieve User Data**: During registration, prompt the user for their information such as email, name, username, password, etc. and store that information into a database. Once that data is needed - like for user login, authentication, or displaying information - the system will retrieve the stored information to perform the desired task.
- **Store, Retrieve, and Display User Posts**: Users will have the ability to create posts by entering text into a form and submitting that data into the database. Once that post needs to be used, it will be retrieved by the system to be displayed. Comments, which are a subset of posts, will also be available to users; comments can be "left" on posts and will be associated with them within the database. If the comments of a particular post is requested, the system can retrieve and display them.
- **Store and Retrieve Topics**: Each user post will be assigned a topic by the user. A post's topic will determine at which time it will need to be retrieved and displayed; in short, when a user opens a certain topic, the posts that match that topic will be retrieved and displayed (in chunks so the browser / system is not overloaded).

- **Internet Connection**: The supported device of choice will need an Internet connection to connect to, use, and interact with the web application. Without an Internet connection, the device will not be able to reach or use the application at all.

- **Javascript Enabled:** Featurama is developed on Node.js, a javascript runtime environment, therefore JavaScript must be enabled on the user's browser in order for the website to function correctly.

Non-Functional Requirements

- **User Interface**: Featurama's user interface is composed of tabs which will control which data is displayed to the user. This design is intuitive, clean, modern and quick, which will allow the user to swiftly and efficiently navigate the system and enjoy its features more easily.

- **Responsive Design**: In addition to the coherent user interface, the entire interface of the system is responsive. Responsiveness is the key to how so many devices will be able to cleanly and easily utilize the software because it will dynamically change based on the screen size.

- **Modularity**: The system is modular in its design. This not only helps the end-user better compartmentalize different components of the system for easier use, but, from a development standpoint, this helps make the creation process much easier. Modular design helps the developers maintain, reuse, and expand the product to ensure the best possible experience for users.

Implementation

Implementation Overview

Featurama will be implemented using the Node.js framework. By using Node.js we can keep the project modular and responsive because Node.js is asynchronous by default which performs faster than other frameworks because it is single-threaded and non-blocking.

- **Frontend:** Will be developed with a combination of languages and libraries. React.js will be the main library used for the application in addition to HTML. We'll also be using Material UI as a component library for React. Material UI allows us to quickly add CSS styles and components like buttons, tabs, dialog boxes, menus, etc.

- **Backend:** The backend will be developed using NodeJS with the ExpressJS framework. NodeJS is a cross-platform, open-source, JavaScript runtime that allows us to use JavaScript for server-side programming. Running on top of NodeJS, we will use the ExpressJS web application framework to quickly set up server operations such as POST and CRUD, and API routes.

- **Database:** We will use MondoDB to build a database for this project. MongoDB is an open source database management system that uses a document-oriented database model which supports various forms of data. It is one of numerous nonrelational database technologies under the NoSQL banner for use in big data applications and other processing jobs involving data that doesn't fit well in a rigid relational model. Instead of using tables and rows as in relational databases, the MongoDB architecture is made up of collections and documents. MongoDB doesn't require predefined schemas and it stores any type of data. This gives users the flexibility to create any number of fields in a document, making it easier to scale MongoDB databases compared to relational databases. One of the advantages of using documents is that these objects map to native data types in a number of programming languages. Also, having embedded documents reduces the need for database joins, which can reduce costs.

- **Security:** MongoDB Server brings added security features, an in-memory storage engine, administration and authentication features, and monitoring capabilities through Ops Manager. MongoDB offers various methods to verify a client's identity. MongoDB offers network encryption and can pass through disk encryption to help you protect your database and communications. TLS and SSL are both standard technologies that are used for encrypting network traffic.

Developer Roles

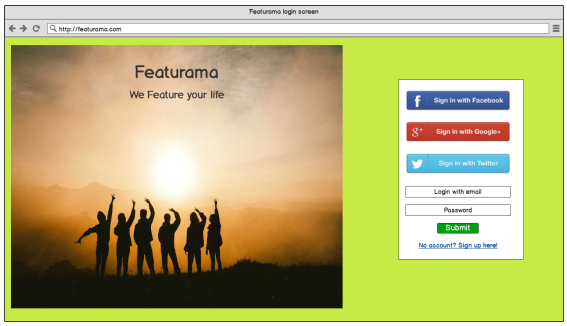
As this project is still in its early stages, we have divided the work according to sections rather than specific features. Features will be assigned to developers over time to help balance the work and they will be assigned to members according to the following list (crossover will occur, but this is a general guide as to who does what).

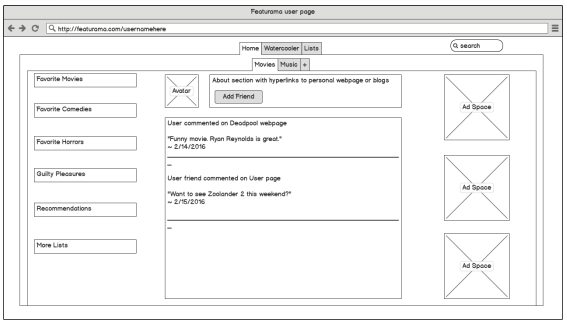
- **Front-End:** *UI, Styling, Input Forms, Views, etc.*
    - Ioannis Batsios
    - Michael McCullough
- **Back-End:** *Business Logic, Send & Retrieve Data, API, Authentication, etc.*
    - Christopher Thacker
    - Jamie Weathers
- **Database:** *DB Design, Store Data, Organize Data, Aid Back-End Developers*
    - Hieu Vo

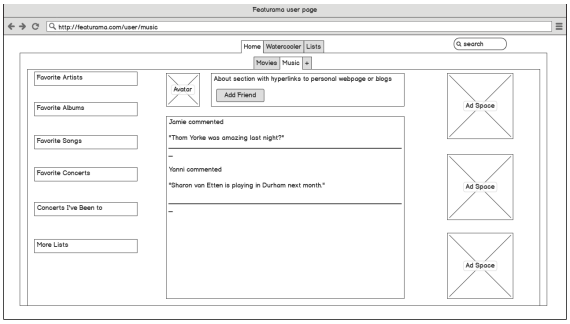
Project Timeline

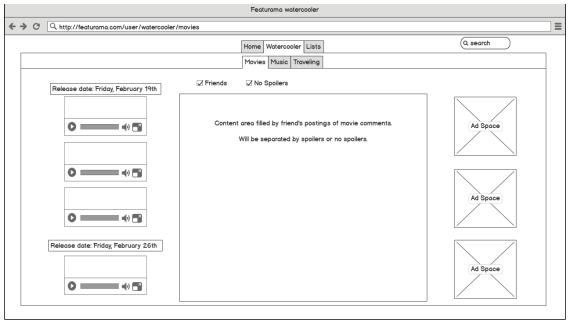
February 6th	Initial Project Presentation
	Functional Requirements
	Non-Functional Requirement
	System Design
	Timeline
February 20th	Project Presentation
	Use Case Diagrams
	UML Diagrams
	Data Flow
	Entity Relational Diagram
April 9th	Preliminary Demonstration
	Unit Testing with Normal, Edge, and Error Cases
	Implementation of a Training Plan
April 28rd	Final Demonstration
	Project Poster
April 30th	Final Report Due

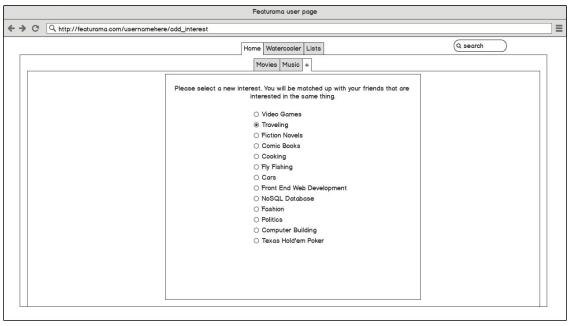
Wireframes (Ioannis Batsios)

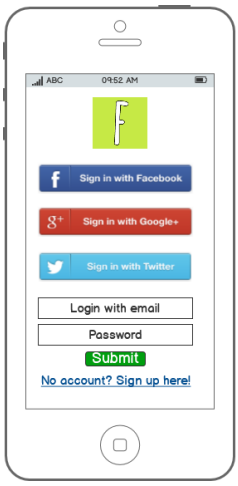












Data Dictionary (Ioannis Batsios)

Below is a list of definitions to help developers and users understand what certain terms mean inside of our application. The following definitions are arranged by the Parent Page and drilled down. Definitions will be repeated, but it's only for clarity on how they are used by the Parent Page.

- **Home Page**: This is the first page that is opened when the site is launched in a browser. It is the User's homepage and is used to modify their Bio and Lists, add or delete Interests, and can be used to launch comments.
    - **Bio:** This is a general synapse of the user. It is modified by the user. It will contain the user's first name, last name, and a profile picture. The user can add to it with an email, their birthdate, social media links, personal webpage, etc.
    - **Interests:** This is the heart and soul of the website and what makes Featurama different from other social media networks. Once a user adds an interest the interest is linked to the List component, the Post component, and the Ad component, meaning that if a user adds an interest in Movies, and once Movies is selected, a user can only see Lists, Posts, and Ads related to Movies.
    - **Lists:** Lists are similar to a Pinterest board. Users create a List by giving it a title. Then they populate the list with Items that relate to the List. The Lists can be sorted by the user manually or in ascending/descending order.
    - **Items:** These are added to a List to populate it. The Item can be created with either a hyperlink and/or a title. The user can also add a description to the Item. The Items can be sorted by the user manually or in ascending/descending order.
    - **Posts:** Posts are generated by the user. Posts will be moderated to make sure they are appropriate to the Interest. Some Posts will have additional attributes based on the Interest, e.g., Movie and Book Interests will have a spoiler button to

warn other users that the Post contains a spoiler. Posts will be linked to the Interest and the specific Post will only be displayed when a specific Interest is selected. There will be no limit to the size of the Post as they can act much like a Blog. Order is by date ascending. On the Home Page, only the user's Posts can be seen. For friend's or everyone's Posts, see Watercooler below.
    - **Comments:** Comments are made on Posts. Comments will also have additional attributes based on the Interest and inherited from the Post's attributes. Comments will also be moderated to make sure they are Interest specific and can only be seen when the specific Interest is selected.
    - **Ad Space:** Ad Space will be targeted by the Interest selected and will change once a user has selected a different Interest. Ads will not target a specific user, but their specific Interest.

- **Watercooler Page:** The Watercooler Page is where all the users get together to discuss an event, item, topic of Interest. Much like the Home Page, the Watercooler Page is modified by the selection of the Interest tab. If Movies is selected by the user, then Posts, Releases, and Ads will change in accordance.
    - **Releases:** This section is to help the user see what is upcoming in their Interest. If the user has selected Movie, then upcoming trailers of movies will be displayed by release date. The same with Music, Books, etc. If the user has selected an Interest that isn't governed by something that can be fulfilled by goods, for example, Politics, then the Releases will be filled with articles, television events, etc. that will satiate their needs in that topic. Releases are created and filled by the developers and not users. Space will be rented out to companies wanting to advertise their wares there.
    - **Interests:** Acts the same as the Home Page Interest tab. Once a user adds an interest the interest is linked to the Releases component, the Post component, and the Ad component, meaning that if a user adds an interest in Movies, and once Movies is selected, a user can only see Releases , Posts, and Ads related to Movies.
    - **Posts:** This acts exactly the same as the Home Page, but instead of seeing just the user's Posts, the user can see everyone's Posts (unless the friend's button is selected where they'll only see their friends). Posts only shown from a specific Interest. Order is by date ascending.
    - **Comments:** Exactly as Comments in the Home Page.
    - **Ad Space:** See Home Page / Ad Space above.

- **Lists Page:** This page is an extension of the List component in the Home Page. The Home Page will have limited space for the List component, so the Lists Page will display as many Lists as the user wants to create.
    - **Lists:** See Home Page / Lists

- **Items:** See Home Page / Items

System Capabilities

Stakeholders (Christopher Thacker)

- General public: people who wish to use the platform to socialize or view interests
- Administrators: those who manage the platform/system

Activities of Stakeholders (Christopher Thacker)

- General public
  - Select interests to view
  - Create, edit, read, and delete their own content
  - Create, edit, read, and delete content in response to other peoples' content
  - Report other peoples' content that is deemed inappropriate
  - Create, read, edit, and delete personal lists for favorite content
- Administrators
  - Review reported content
  - Remove inappropriate content
  - Ban users that break the terms and conditions

Context (Christopher Thacker)

- The system will be deployed to a server which will allow the general public and administrators to access it from any device capable of connecting to the internet. This can physically take the form of smartphones, tablets, laptops, or desktop computers and extends to any location that is authorized to access the web address.

Technology (Christopher Thacker)

- Internet Service
- Desktop Computers / Laptops
- Tablets
- Smartphones
- HTML, CSS, JavaScript, React, Material UI
- Node.js, Express, JavaScript, MongoDB

Non-Functional Requirements (Christopher Thacker)

- Security: stakeholders will be required to login to create, edit, or delete content
- User interface: UI will be tab-based, attractive, and quick for easy use by stakeholders
- Modularity: features will be modular for easy integration with maintenance
- Responsivity: interface will adapt to the device that is being used by stakeholder(s)

Business Benefits

Monetize Specificity (Ioannis Batsios)

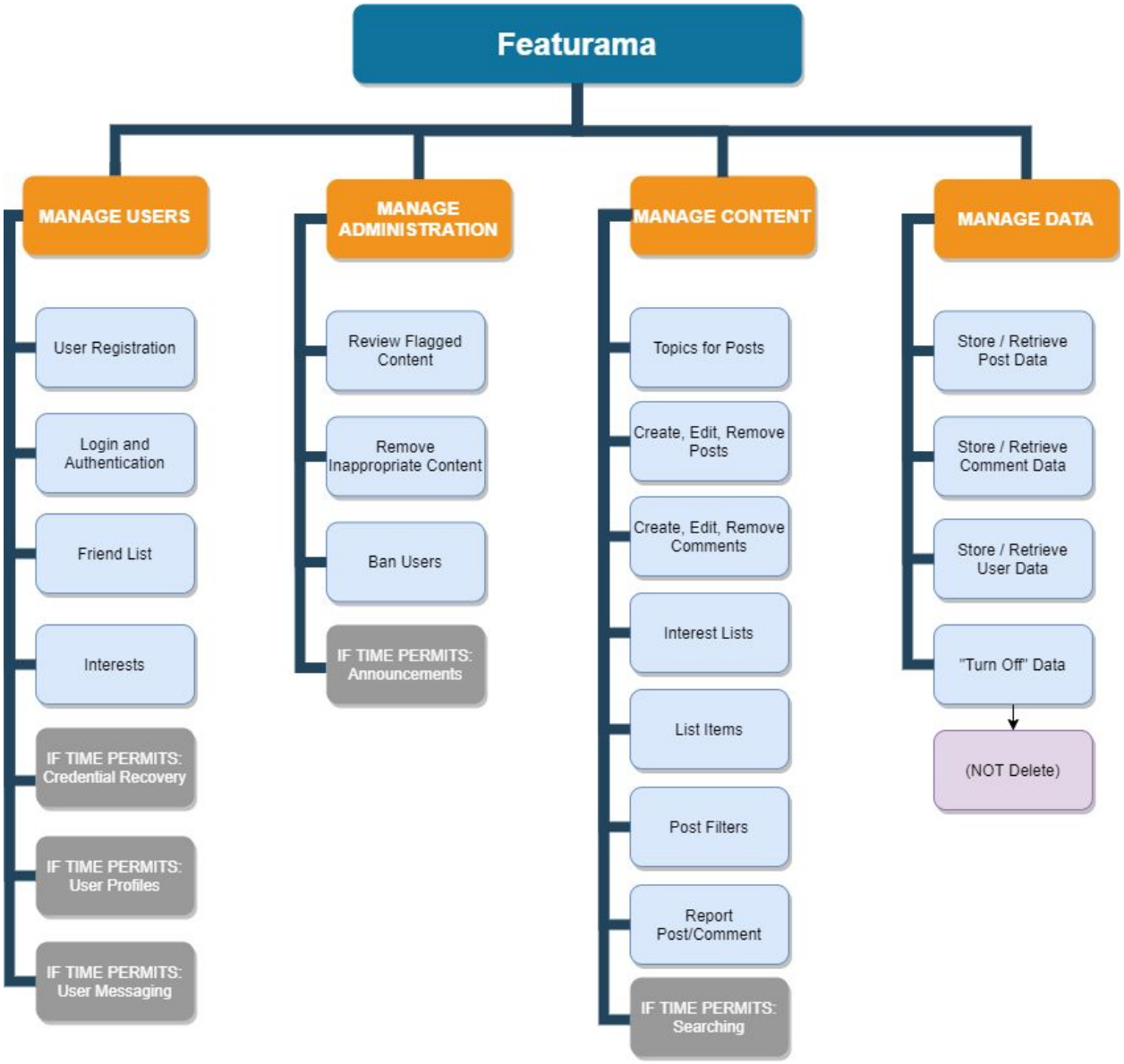
- Advertisers
    - Advertisers will be able to target a user's general interest, e.g., a user has an interest in movies; production companies will be able to offer deals on Blu-rays, DVDs, online-rentals, etc. through targeted advertisement space.
- Companies / Corporations
    - By having a calendar feature inside of the Watercooler page, companies will be able to display presentations to interest users into buying their product. That space will be rented by the companies wishing to advertise.

Time-Saving (Ioannis Batsios)

- Users
    - Users will have their interests and desire for consumption fed more easily by saving them time and energy by not having them looking for more sources for their interests.
    - Users will be able to communicate with like-minded users and be able to offer deals on items of interest to them.

Major Subsystems

Diagram (Christopher Thacker)


Featurama
MANAGE USERS
User Registration
Login and Authentication
Friend List
Interests
IF TIME PERMITS: Credential Recovery
IF TIME PERMITS: User Profiles
IF TIME PERMITS: User Messaging
MANAGE ADMINISTRATION
Review Flagged Content
Remove Inappropriate Content
Ban Users
IF TIME PERMITS: Announcements
MANAGE CONTENT
Topics for Posts
Create, Edit, Remove Posts
Create, Edit, Remove Comments
Interest Lists
List Items
Post Filters
Report Post/Comment
IF TIME PERMITS: Searching
MANAGE DATA
Store / Retrieve Post Data
Store / Retrieve Comment Data
Store / Retrieve User Data
"Turn Off" Data
(NOT Delete)

Constraints

Implementation Constraints (Jamie Weathers)

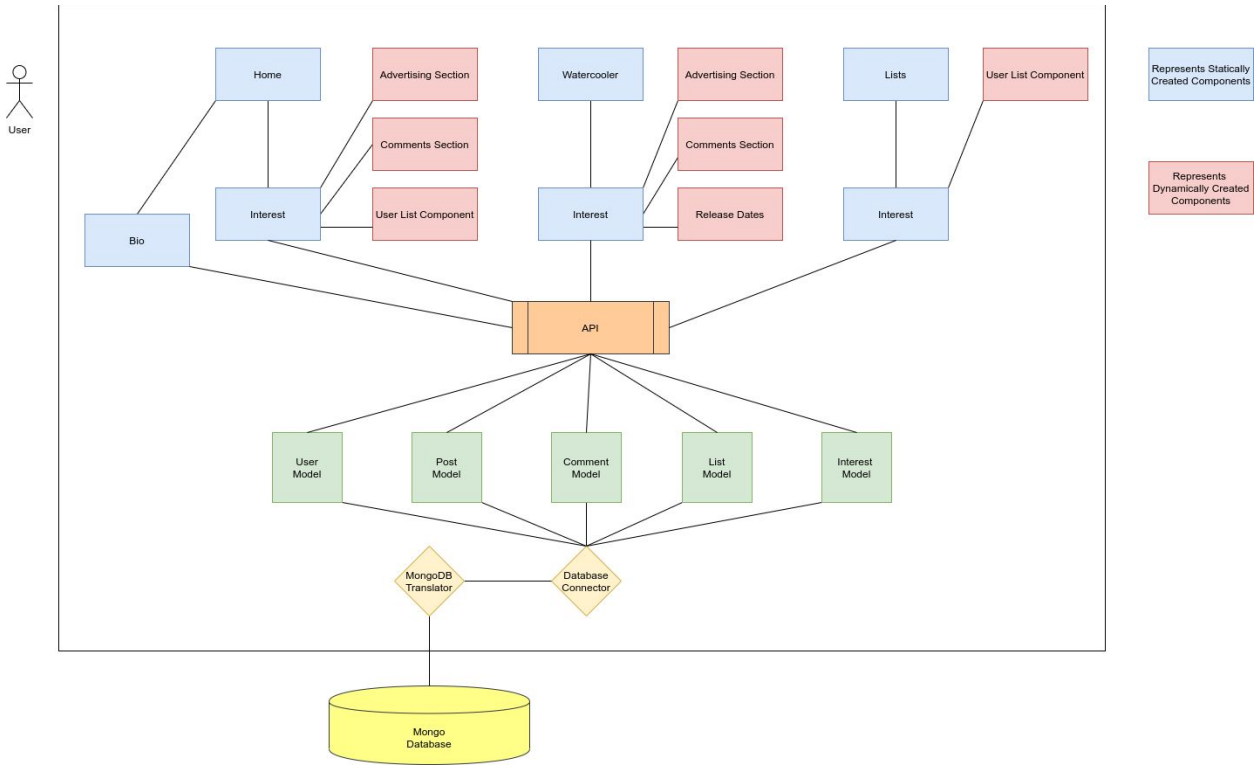
- Limited time
    - With a deadline of April 24th, we have to work fast and concisely to deliver a Minimum Viable Product on time.
        - There are 64 days at the time of this writing until April 24th, 2020.
- Minimum financial resources available
    - A college project made by college students means no money. We'll be optimizing free or close-as-possible-free servers to host our database.
- Appearance, quality and presentation concerns
    - Given the lack of resources and time, we'll focus on implementing a barebones product with the hope that fast execution means more features and better optimization.

Project Constraints (Jamie Weathers)

- User
    - Users will not be able to add their own interests in the tab.
        - Interests are created by the developer and rolled out once the planning phase has gone through issues and developed attributes to the specific interest.
    - Users will not be able to upload photos, videos, or other media into the Lists section.
        - This reduces upload time and increases the website's dynamism.
    - Users can not modify the layout of the website.
    - Users can not delete any data from the database.
        - When they request to remove an item, the item is hidden from view, but not deleted.
- Admin
    - Admin will have no discernible effect on any of the layout, features, advertisement, or functionality of the site.
    - Admin cannot delete a user.
    - They can only block content the user has posted.
        - The removal of a User goes through a committee, which may be implemented later if time allows.
    - Admin does not have access to the database, or read/write permissions.

Data Model

Diagram (Michael McCulloch, Ioannis Batsios)

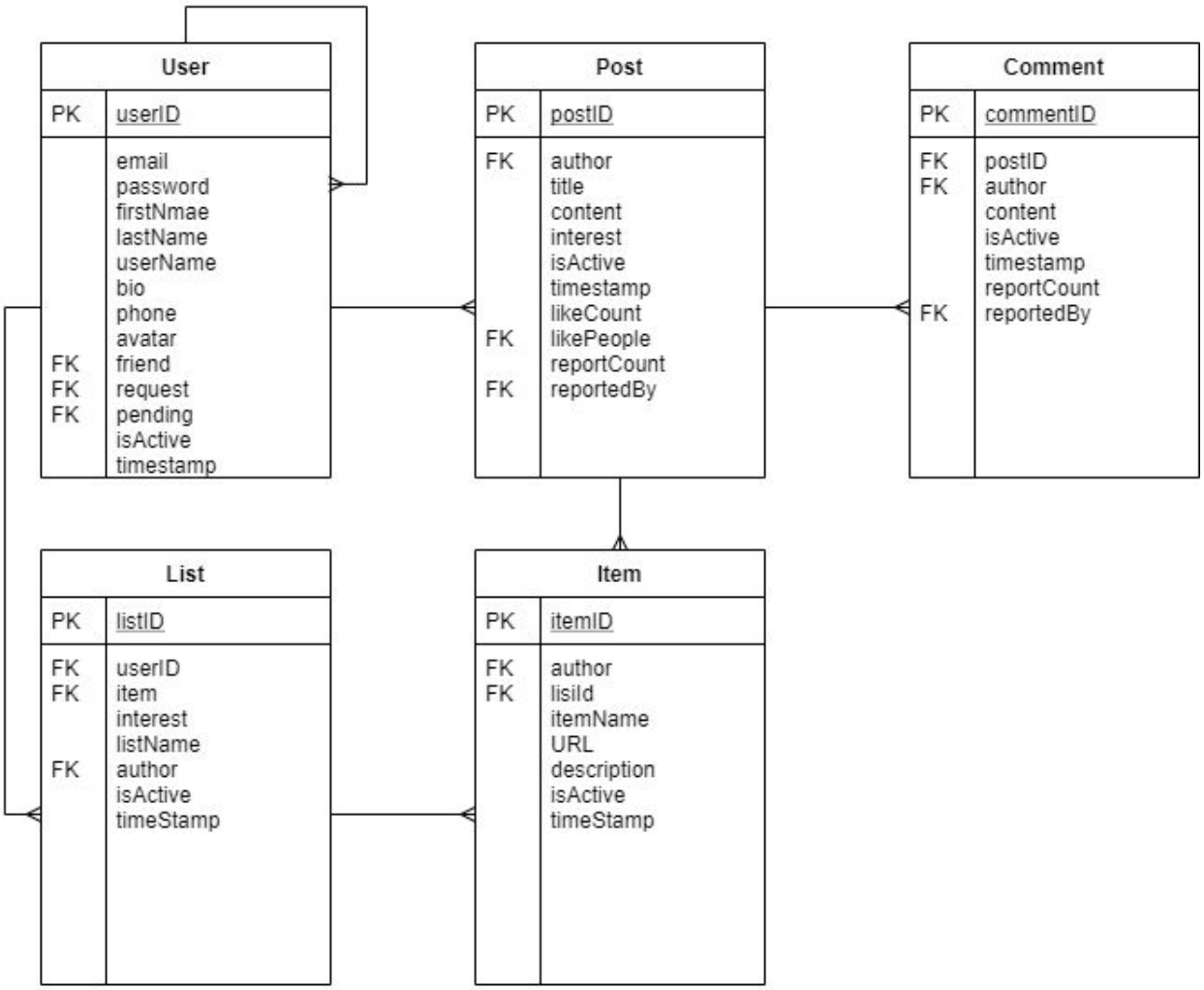


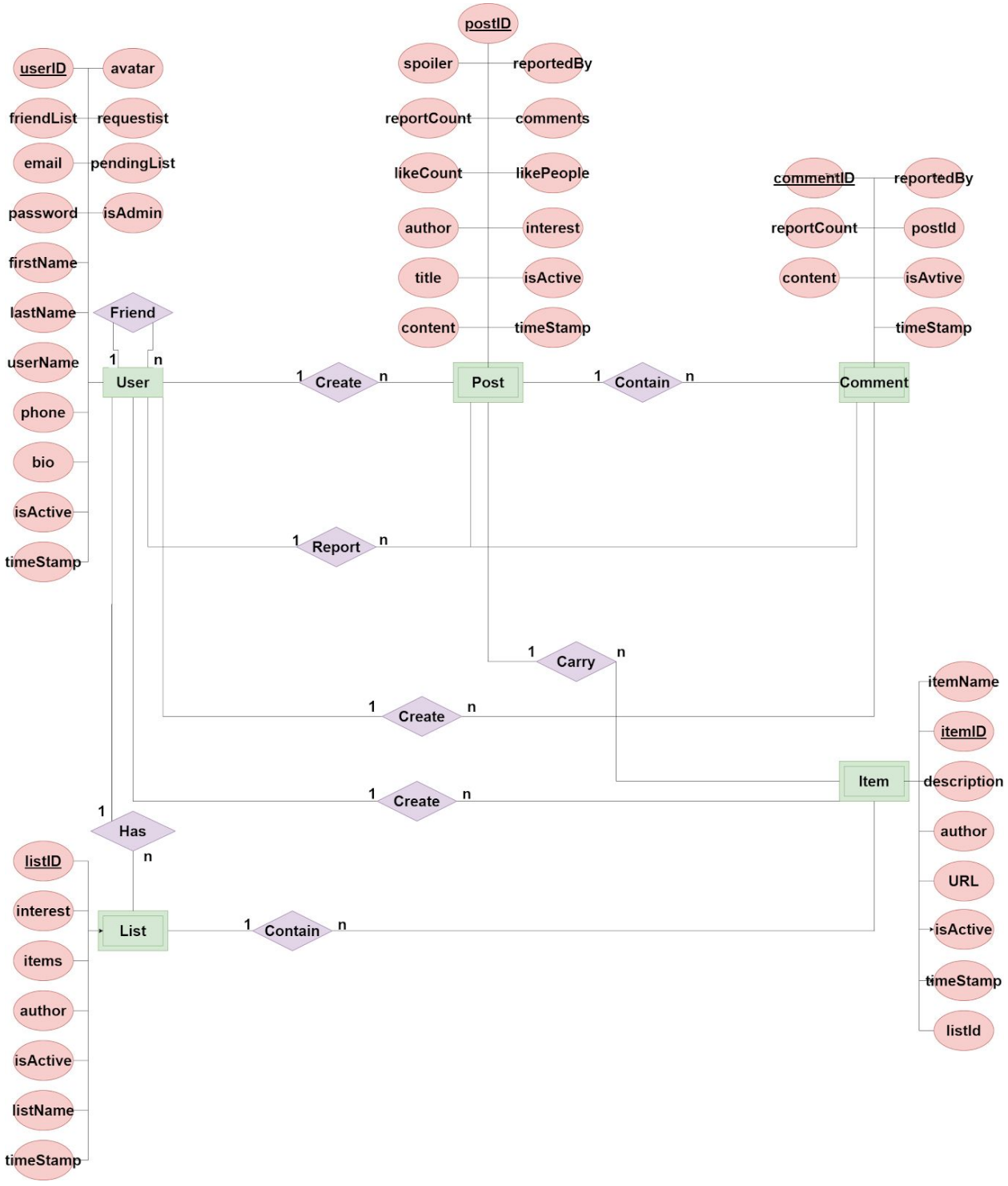
Overview (Michael McCulloch)

This data model diagram represents how the system communicates with itself. Components are called when other components have called them. The data is routed from the models that call the data through the connector. The translator translates the calls and is used in case we want to change our database in the future.

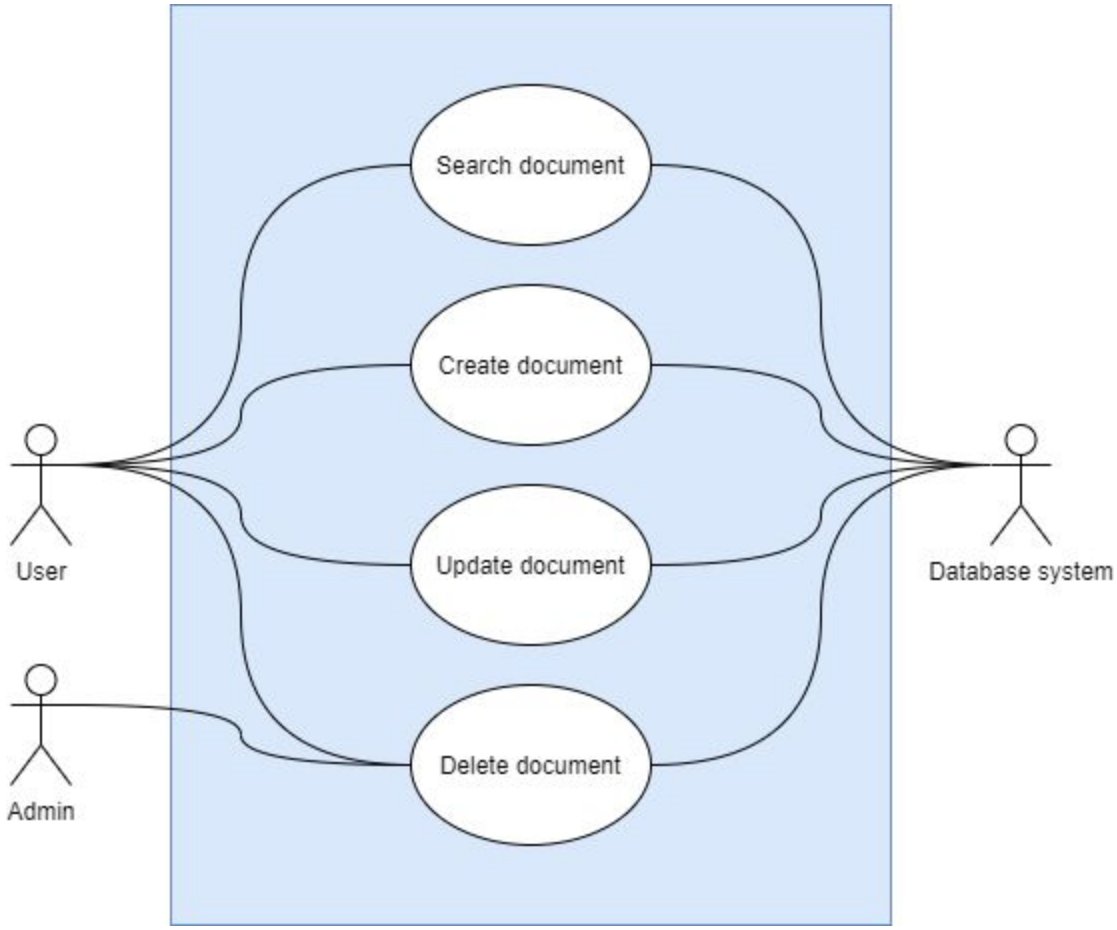
Database Design

ER Diagram (Hieu Vo)





Use-Cases (Hieu Vo)



Create new account

Use-case 1	New account creation
Actor	User
Overview	A user wants to open a new account to start using Featurama
Trigger	A user is interested in Featurama
Precondition	none

Search for an account

Use-case 2	Account searching
Actor	User
Overview	A user wants to search for other user’s

	accounts, their information and interests.
Trigger	A user is interested in someone else's profile or interests.
Precondition	User is authenticated

Update an account

Use-case 3	Account’s information updating
Actor	User
Overview	A user can change and update his/her existing information and profile.
Trigger	A user wants to update and edit his/her account’s information
Precondition	User is authenticated

Delete an account

Use-case 4	Account deactivation
Actor	User or Admin
Overview	A user or admin deactivate account which removes all of their information from the database.
Trigger	A user is not interested in Featurama anymore or wants to stop using Featurama Admin deactivates accounts that violate the web’s policies.
Precondition	User is authenticated

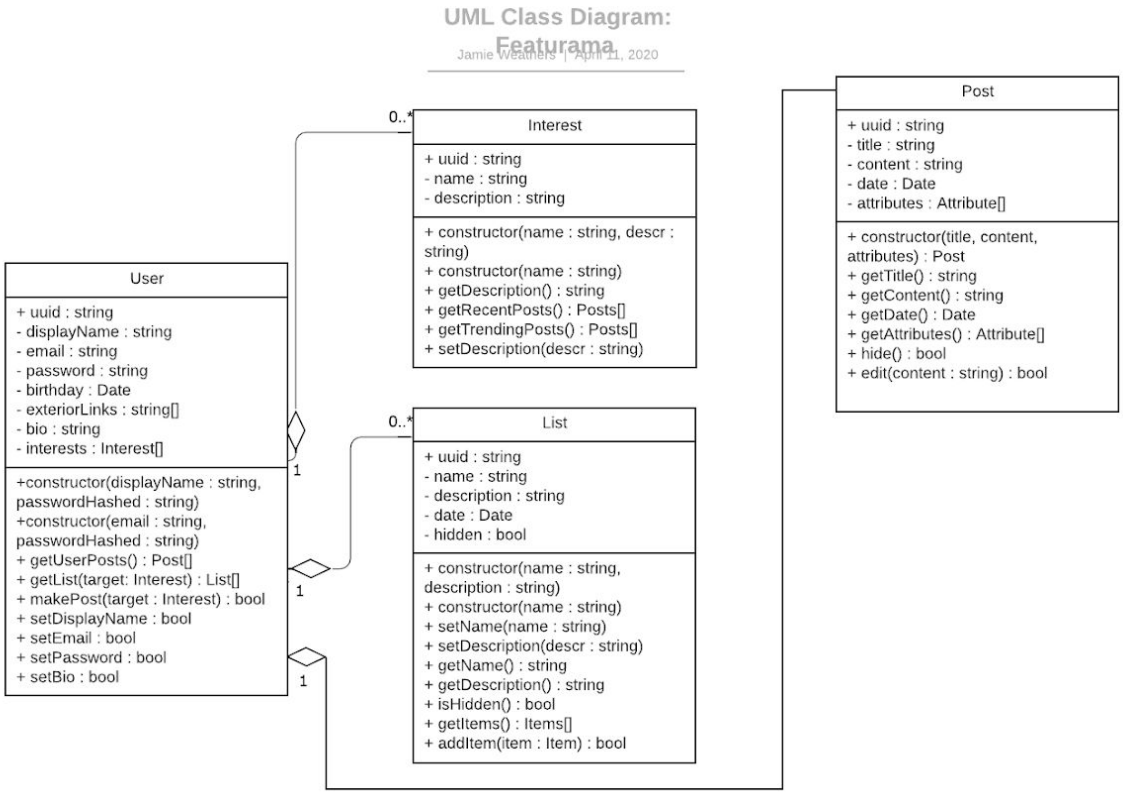
Database System Time Complexity (Hieu Vo)

MongoDB is using btree for indexing. So the order of the algorithm is O(log n). O(logn) means that the algorithm's maximum running time is proportional to the logarithm of the input size. So for a million records we only have two levels (assuming that tree is balanced) and that is why it can find the record so fast.
log2 (1,000,000) = 20.

UML

Class Diagram (Jamie Weathers)

Adjustments and additions to the UML class diagrams will be updated as the project grows.



Build Plan

Overview (Christopher Thacker)

Upon completion of the initial documentation and diagrams (i.e. this document), we will immediately begin work on optimizing and finalizing the skeleton code that will be the foundation of the remaining features to be added. Once completed, we will begin working on a single major subsystem from the *Functional Decomposition Diagram* at a time.

Priorities (Christopher Thacker)

1. Content Management
2. Data Management
3. User Management
4. Administration Features
5. Any extra features for the above categories

Although these categories have been given priorities, they each will communicate with each other and thus will require forethought during the development of any feature to allow for easy integration with any existing or future pieces.

Development Approach (Christopher Thacker)

The proposed development process for Featurama is an Agile approach. Given the short timeframe of the semester and the learning curve with some of the technologies being used, this methodology will allow us to learn by doing and will give us the opportunity to change design flaws or the direction of features on the fly, if necessary. This will be achieved with small teams of two or three people and consists of planning, developing, launching, and maintaining assigned features throughout the week coupled with consistent communication remotely and with meetings at the end of each week to review our work, and to decide where we should focus our resources into next.

Team Structure

Overview (Christopher Thacker)

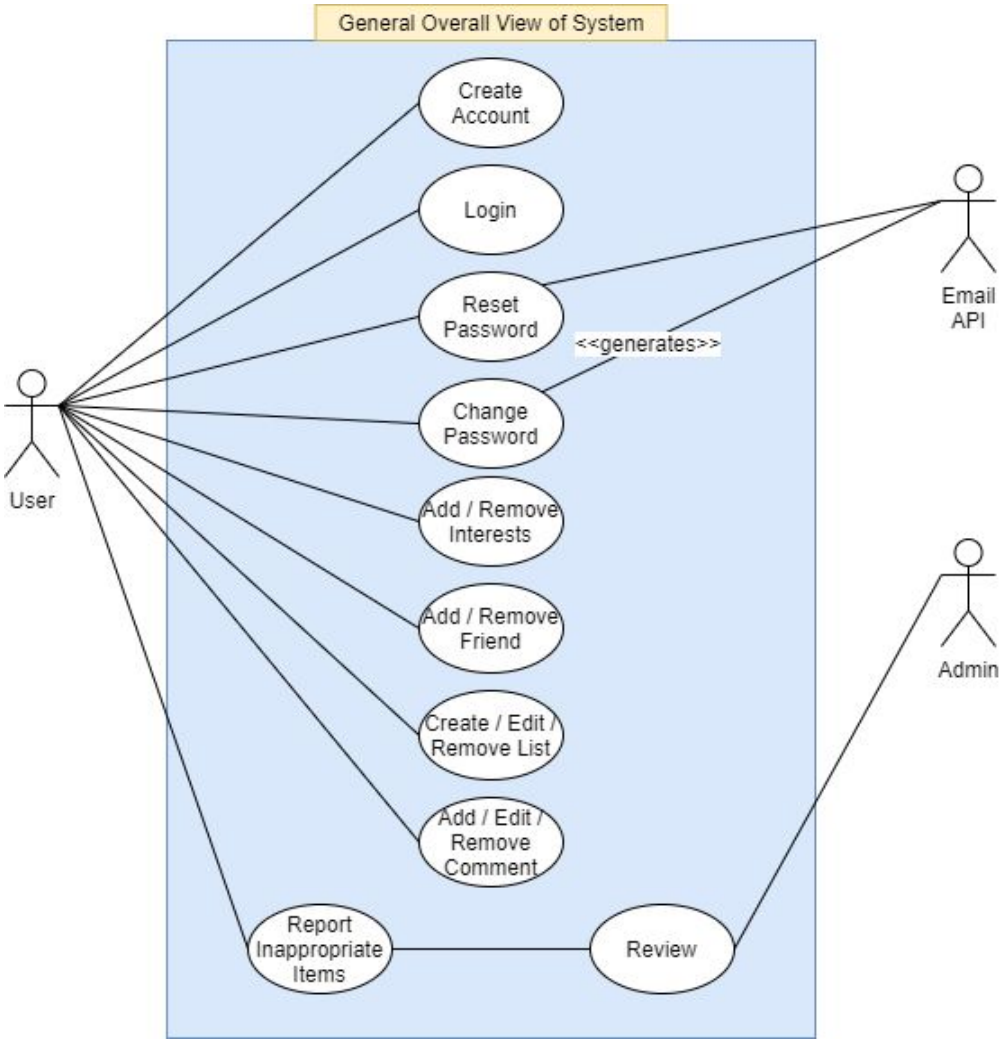
Team Members:

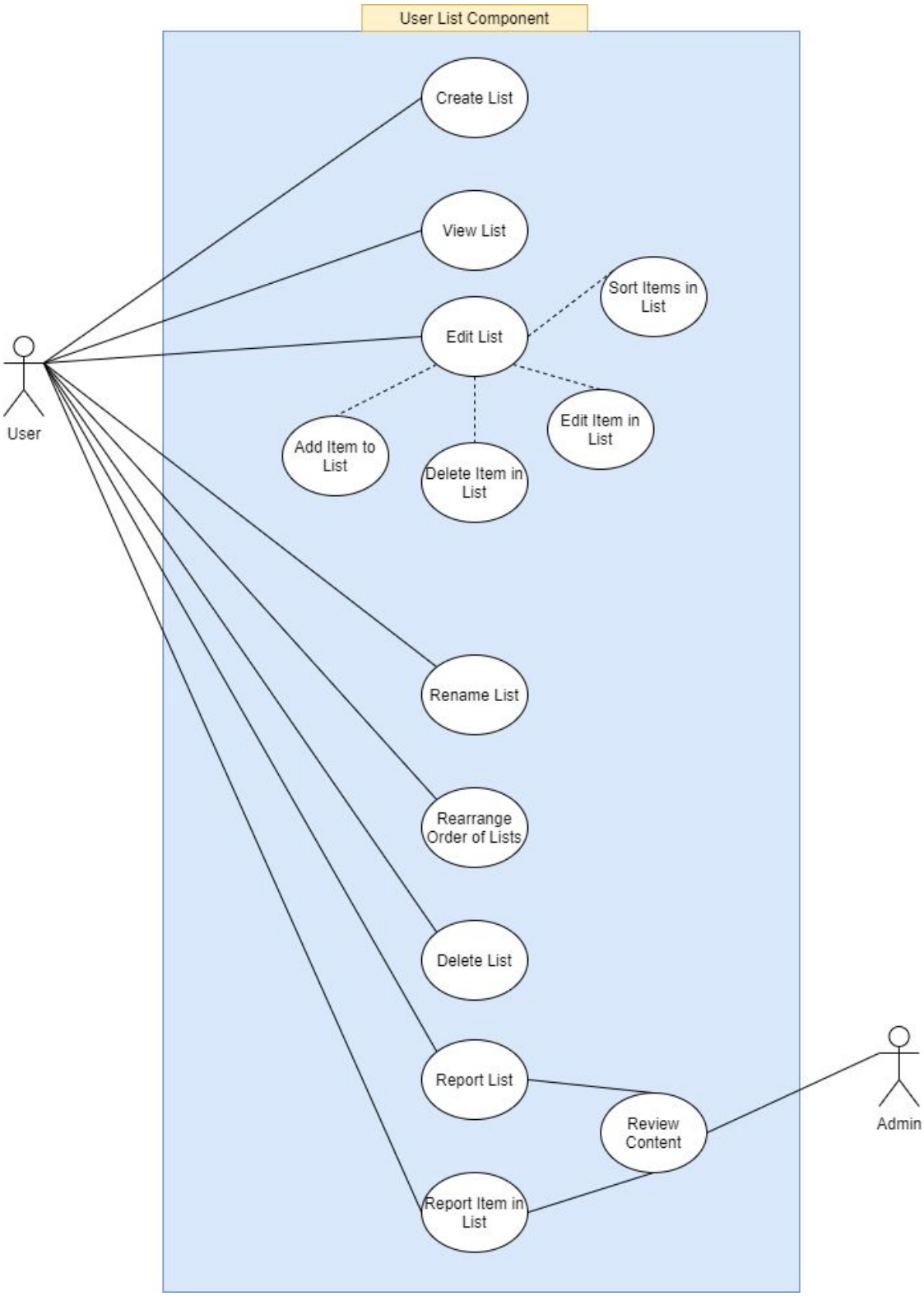
- Ioannis Batsios *(Visionary; Front-end Developer)*
- Michael McCulloch *(Front-end Developer)*
- Christopher Thacker *(Back-end Developer)*
- Hieu Vo *(Database Design; Back-end Developer)*
- Jamie Weathers *(Back-end Developer)*

As a team, every person listed above will contribute to the project (code, documentation, presentations), will keep in constant communication with other team members and will attend the weekly meetings if they are able to. All ideas and suggestions will be taken seriously, will be considered, and will be available for discussion with the entire group to create the best possible outcome or decision. Each member will be responsible for their assigned area of duty listed next to their name above, but there will be some cross-over between roles depending on the task at hand.

Use-Case

Diagrams (Michael McCulloch, Ioannis Batsios)




User List Component
Create List
View List
Sort Items in List
Edit List
Edit Item in List
Add Item to List
Delete Item in List
User
Rename List
Rearrange Order of Lists
Delete List
Report List
Review Content
Admin
Report Item in List

Create List Use-Cases (Michael McCulloch)

Use Case 1	User creates a new list for an interest
Actor	User
Use Case Overview	A user is interested in creating a list containing their favorite Movies, to share with friends and other users. The user navigates to a desired interest and selects add option, then gives the list a name and list items.
Trigger	A user wants to create a list of interests.
Precondition	User is authenticated

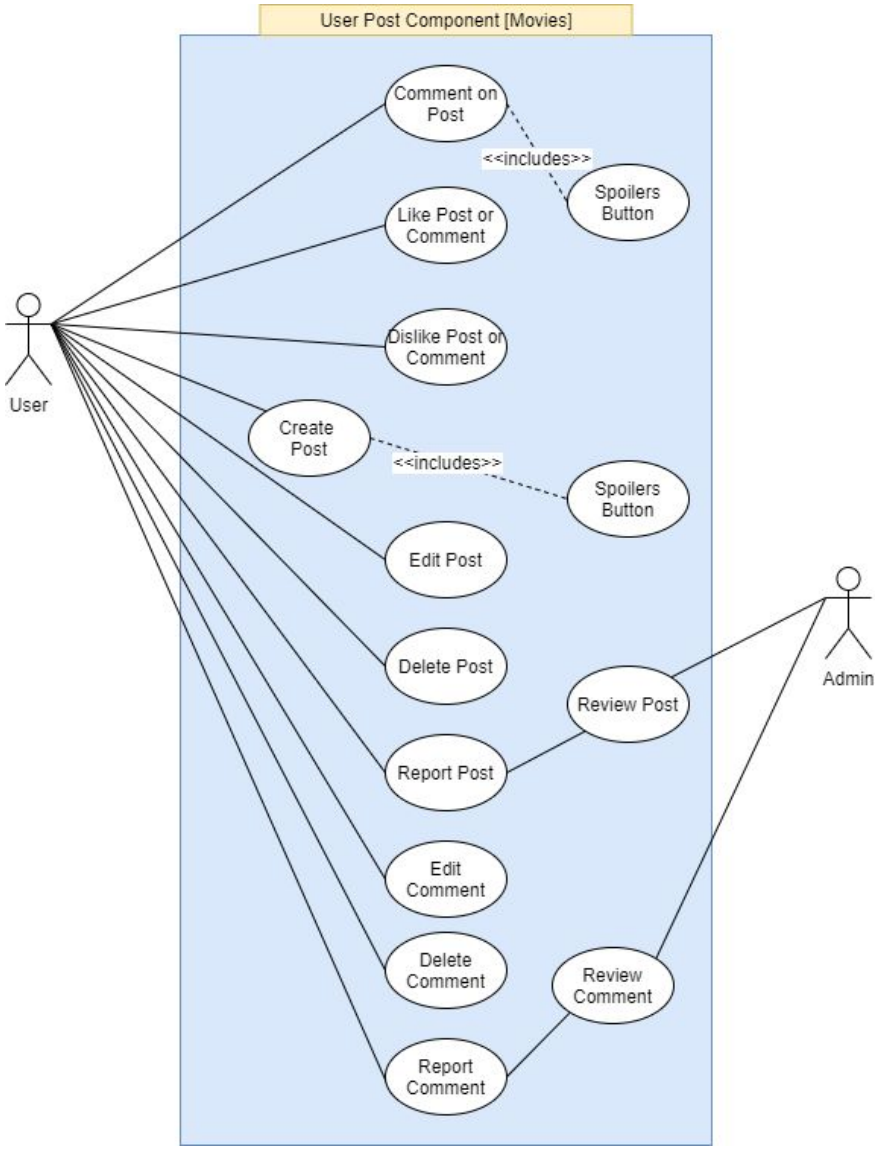
Edit List Use-Cases (Michael McCulloch)

Use Case 2	User edits an existing list.
Actor	User
Use Case Overview	A user wants to rename a list and add a new item to the list. The user navigates to the list to be edited and enters edit mode. After providing a new name for the list, the user adds a new item to the list.
Trigger	A user wants to rename a list for an interest and add new list items.
Precondition	User is authenticated

Delete List Use-Cases (Michael McCulloch)

Use Case 3	User deletes a list.
Actor	User

Use Case Overview	A user no longer wants to share a particular list with friends and other users of the site. The user navigates to the undesirable list and removes it.
Trigger	A user no longer wants to share a list.
Precondition	User is authenticated



Posts Use-Cases (Michael McCulloch)

Use Case 4	User Post
Actor(s)	User
Use Case Overview	A user makes a comment on the site.
Trigger	A user has something worth commenting on.
Precondition	Users are authenticated
Alternative Flow 1	If the user wants to modify their post, the user can edit the post.
Alternative Flow 2	If the user decides to remove the post, the post can be removed by the user.

Comments Use-Cases (Michael McCulloch)

Use Case 5	User Comment
Actor(s)	User, Admin
Use Case Overview	A user finds a comment on the site.
Trigger	Another user has made a comment.
Precondition	Users are authenticated
Alternative Flow	If the user thinks the comment is offensive, the comment can be flagged for review by the admin.

Report Use-Cases (Michael McCulloch)

Use Case 6	User reports offensive content

Actor(s)	User, Admin
Use Case Overview	A user finds an offensive list, post, or comment. The user flags the content, then the content is made available to the Admin for review. The admin decides whether or not to remove the content.
Trigger	A user finds content offensive.
Precondition	User is authenticated

Training Plan

Overview (Ioannis Batsios)

Below is an overview of how the user interacts with our site.

- How to register
    - User must give a first name, last name, email, and password
    - If first name and/or last name field is unfilled
        - User will be prompted to put fill out empty fields
    - If email is already in database
        - User will be notified that email already exists
    - If password is too short
        - User will be notified to select a longer password
- How to login
    - User is not registered
        - There is a link to go to the registration page
    - User does not have a valid username
        - There will be a message for the user to create a valid username
    - User does not remember their password or too many failures
        - User will be redirected to a create new password page through their email
    - User has a valid username and password
        - Is redirected to their Home / General page
- How to access pages
    - User must be logged in

        - User selects the pages he/she wants to access using the tab system navigation
        - Tabs are nested with the parent being the page user wants to visit and the child being the interest of that page
- Posting
    - How to add a post
        - In the top left corner, there is a menu toggle. The toggle has an option to add a post. After selecting, you need to fill a post title, post content, and select which interest it pertains to.
    - Mark as Spoilers checkbox
        - What is it?
            - If your post has the potential of spoiling the outcome of a film, movie, or other event/content, then you should select the spoiler checkbox.
        - What if I forget?
            - Users may report the post and it will be removed.
    - How to edit a post
        - After a post has been created, there will be vertical ellipses in the component. Clicking on them brings up the post menu. From there you can select Edit post and edit it.
    - How to delete a post
        - After a post has been created there will be vertical ellipses in the component. Clicking on them brings up the post menu. From there you can select Delete post. Careful though, you will not be able to recover your post once it has been completed.
        - Also to note: deleting a post will delete all of the comments of that post as well.
    - How to add post to interest
        - See “How to add a post” above
        - Adding an interest to a post is important so users and yourself can access only the content in which you are interested in.
    - How to report a post
        - Click on the vertical ellipses to bring up the post menu. From there you can select Report. Report only works on other user’s posts, not your own.
- Commenting
    - How to add a comment
        - Comments can only be added to original posts. Click on the vertical ellipses on the post and click the add comment icon. A text field will come up allowing you to comment.
    - How to remove a comment
        - Click on the vertical ellipses of the comment you wish to delete. Click the delete icon. This can not be undone and does not delete the original post. It can not be undone.

    - How to edit a comment
        - Click on the vertical ellipses of the comment. Click on the edit icon. Edit and save the new comment.
    - How to report a comment
        - Click on the vertical ellipses to bring up the comment menu. From there you can select Report. Report only works on other user's comments, not your own.
- Lists
    - How to add a list
        - In the top left corner, there is a menu toggle. The toggle has an option to add a List. After selecting, you need to fill a list name and select the interest it pertains to.
    - How to view a list
        - Click on the Lists tab in the navigation. From there you can view any list in your interest.
    - How to edit a list
        - Once a list has been created, click on the vertical ellipses to bring up the list menu. From there select the edit icon. You will be able to change the interest and name of the list.
    - How to delete a list
        - Once a list has been created, click on the vertical ellipses to bring up the list menu. From there select the delete icon. Once a list has been deleted, it can not be undone.
    - How to report a list
        - Once a list has been created, click on the vertical ellipses to bring up the list menu. From there select the report icon. If the list is deemed to be against Featurama's policies of allowable content, it will be removed and the user may be banned.
- Items
    - How to add an item
        - Once a list has been created, click on the vertical ellipses to bring up the list menu. Click on the add item icon. A prompt will be introduced for you to fill in the item name, item URL, and a description. The name is required, the others are not. Click save
    - How to view an item
        - Click on the vertical ellipses of a list to bring up the menu. Click the view icon. A prompt will show you the list items of the list.
    - How to edit an item
        - Click on the vertical ellipses of a list to bring up the menu. Click the view icon. A prompt will show you the list items of the list. Click on the edit icon. A prompt will let you edit all of the text fields of the item. Click save.
    - How to delete an item

- Click on the vertical ellipses of a list to bring up the menu. Click the view icon. A prompt will show you the list items of the list. Click the delete icon of the item you want to delete. This can not be undone.

Test Cases

Database Layer Testing (Christopher Thacker)

The approach I have taken is black-box testing for the Database Connector.

Connector

Functionality	Test Case	Expected	Result	Case Type
DB Connector	Connect to translator	The translator will be located and DB connection established	Pass	Normal
DB Connector	Connect to translator that doesn't exist	The translator will not be located and the connection will not open.	Pass	Error
DB Connector	Only one instance of the connector is present	Only one instance of the Database Connector will be instantiated even if the "new" keyword is used more than once	Pass	Edge

Translator

Functionality	Test Case	Expected	Result	Case Type
MongoDB	Connect to DB with valid URI	MongoDB connection will be established	Pass	Normal
MongoDB	Connect to DB with invalid URI	MongoDB connection will not be established and an "Invalid URI" error will be	Pass	Error

		returned		
MongoDB	Connect to DB with no URI	MongoDB connection will not be established and a “No URI” error will be returned	Pass	Edge
MongoDB	Create an object	The object will be stored in the database	Pass	Normal
MongoDB	Read one object	The single object will be returned from the database	Pass	Normal
MongoDB	Read multiple objects	The list of objects will be returned from the database	Pass	Normal
MongoDB	Update an object	The object’s old data will be replaced with the new data	Pass	Normal
MongoDB	Delete an object	The object will be removed from the database (this will be replaced with setting a flag)	Pass	Normal

User Data Testing (Christopher Thacker)

The approach I have taken is black-box testing for the User system.

Create

Functionality	Test Case	Expected	Result	Case Type
Register	Valid format in “Email” field	Does not return an email error	Pass	Normal
Register	Invalid format in “Email” field	Returns email format error	Pass	Error
Register	Blank “Email” field	Returns empty field error	Pass	Error
Register	Valid password	Does not return a	Pass	Normal

	input	password error		
Register	Passwords don't match	Returns mismatched password error	Pass	Edge
Register	Valid username input	Does not return an error	Pass	Normal
Register	Username field is blank	Returns empty field error	Pass	Error
Register	Username is already being used	Returns username in use error	Pass	Edge
Register	All required fields filled with valid input	User with that data is added to the database	Pass	Normal

Read

Functionality	Test Case	Expected	Result	Case Type
Login	Valid credentials	JSON Web Token is generated	Pass	Normal
Login	Invalid credentials	JWT is not generated and user is redirected to login again	Pass	Error
Login	Empty "email" field	Returns empty field error	Pass	Error
Login	Empty "password" field	Returns empty field error	Pass	Error

Update

Functionality	Test Case	Expected	Result	Case Type
User Data	Update on object with valid ID	Object is updated in database	Pass	Normal
User Data	Update on object with invalid ID	Returns "invalid ID" error	Pass	Error
User Data	Update on object that doesn't exist	Returns "user not found" error	Pass	Error

Delete

Functionality	Test Case	Expected	Result	Case Type
User Data	Delete on object with valid ID	Object is removed from the database (this will be changed to updating a flag instead)	Pass	Normal
User Data	Delete on object with invalid ID	Returns “invalid ID” error	Pass	Error
User Data	Delete on object that doesn’t exist	Returns “user not found” error	Pass	Error

List Data Testing (Hieu Vo)

The approach I have taken is black-box testing for the List system.

Create

Functionality	Test Case	Expected	Result	Case Type
AddNew	listName provided	Does not return error	Pass	Normal
AddNew	Blank listName provided	Returns empty field error	Pass	Error
AddNew	item provided	Does not return error	Pass	Normal
AddNew	Topic provided	Does not return error	Pass	Normal
AddNew	Topic blank	Returns empty field error	Pass	Error

Read

Functionality	Test Case	Expected	Result	Case Type
GetById	Valid list ID	Post data	Pass	Normal
GetById	Invalid list ID	Error	Pass	Error

Update

Functionality	Test Case	Expected	Result	Case Type
Update	Valid list ID	Update Post data	Pass	Normal
Update	Invalid list ID	Error	Pass	Error
Update	Blank update data	Original post data	Pass	Normal

Delete

Functionality	Test Case	Expected	Result	Case Type
Remove	Delete on object with valid ID	Object is removed from the database (this will be changed to updating a flag instead)	Pass	Normal
Remove	Delete on object with invalid ID	Returns “invalid ID” error	Pass	Error
Remove	Delete on object that doesn’t exist	Returns “user not found” error	Pass	Error

Item Data Testing (Hieu Vo)

The approach I have taken is black-box testing for the Item system.

Create

Functionality	Test Case	Expected	Result	Case Type
AddNew	ItemName provided	Does not return error	Pass	Normal
AddNew	Blank ItemName provided	Returns empty	Pass	Error
AddNew	URL provided	Does not return error	Pass	Normal
AddNew	URL blank	Returns empty		

AddNew	Topic provided	Does not return error	Pass	Normal
AddNew	Topic blank	Returns empty	Pass	Error
AddNew	Description provided	Does not return error	Pass	Normal
AddNew	Description blank	Returns empty	Pass	Error

Read

Functionality	Test Case	Expected	Result	Case Type
GetById	Valid item ID	Post data	Pass	Normal
GetById	Invalid item ID	Error	Pass	Error

Update

Functionality	Test Case	Expected	Result	Case Type
Update	Valid item ID	Update Post data	Pass	Normal
Update	Invalid item ID	Error	Pass	Error
Update	Blank update data	Original post data	Pass	Normal

Delete

Functionality	Test Case	Expected	Result	Case Type
Remove	Delete on object with valid ID	Object is removed from the database (this will be changed to updating a flag instead)	Pass	Normal
Remove	Delete on object with invalid ID	Returns “invalid ID” error	Pass	Error
Remove	Delete on object that doesn’t exist	Returns “user not found” error	Pass	Error

Friends Data Testing (Hieu Vo)

The approach I have taken is black-box testing for the Friend system.

Request

Functionality	Test Case	Expected	Result	Case Type
friendRequest	requesterID, receiverID	Does not return error	Pass	Normal
friendRequest	requesterID	Returns false	Pass	Error
friendRequest	receiverID	Returns false	Pass	Error
friendRequest	Either requesterID or receiverID	Returns false	Pass	Error

Accept

Functionality	Test Case	Expected	Result	Case Type
acceptFriendt	requesterID, receiverID	Does not return error	Pass	Normal
acceptFriendt	requesterID	Returns false	Pass	Error
acceptFriend	receiverID	Returns false	Pass	Error
acceptFriend	Either requesterID or receiverID	Returns false	Pass	Error

Reject

Functionality	Test Case	Expected	Result	Case Type
rejectFriend	requesterID, receiverID	Does not return error	Pass	Normal
rejectFriend	requesterID	Returns false	Pass	Error
rejectFriend	receiverID	Returns false	Pass	Error
rejectFriend	Either requesterID or receiverID	Returns false	Pass	Error

Remove

Functionality	Test Case	Expected	Result	Case Type
removeFriend	requesterID, receiverID	Does not return error	Pass	Normal
removeFriend	requesterID	Returns false	Pass	Error
removeFriend	receiverID	Returns false	Pass	Error
removeFriend	Either requesterID or receiverID	Returns false	Pass	Error

Cancel

Functionality	Test Case	Expected	Result	Case Type
cancelFriend	requesterID, receiverID	Does not return error	Pass	Normal
cancelFriend	requesterID	Returns false	Pass	Error
cancelFriend	receiverID	Returns false	Pass	Error
cancelFriend	Either requesterID or receiverID	Returns false	Pass	Error

Posts Data Testing (Jamie Weathers)

Black box test approach

Create

Functionality	Test Case	Expected	Result	Case Type
AddNew	Title provided	Does not return error	Pass	Normal
AddNew	Blank title provided	Returns empty field error	Pass	Error
AddNew	Content body provided	Does not return error	Pass	Normal
AddNew	Content body blank	Returns empty field error	Pass	Error
AddNew	Topic provided	Does not return	Pass	Normal

		error		
AddNew	Topic blank	Returns empty field error	Pass	Error
AddNew	Author provided	Does not return error	Pass	Normal
AddNew	Author blank	Returns empty field error	Pass	Error
AddNew	All required fields filled with valid input	Post with that data is added to the database	Pass	Normal

Read

Functionality	Test Case	Expected	Result	Case Type
GetById	Valid post ID	Post data	Pass	Normal
GetById	Invalid post ID	Error	Pass	Error

Update

Functionality	Test Case	Expected	Result	Case Type
Update	Valid post ID	Update Post data	Pass	Normal
Update	Invalid post ID	Error	Pass	Error
Update	Blank update data	Original post data	Pass	Normal

Delete

Functionality	Test Case	Expected	Result	Case Type
Hide	Valid post ID	Set isActive field to false	Pass	Normal
Hide	Invalid post ID	Error	Pass	Error

Comments Data Testing (Jamie Weathers)

Black box test approach

Create

Functionality	Test Case	Expected	Result	Case Type
AddNew	Content data provided	Get Comment data	Pass	Normal
AddNew	Content data blank	Error	Pass	Error
AddNew	Author data provided	Get Comment data	Pass	Normal
AddNew	Author data blank	Error	Pass	Error

Read

Functionality	Test Case	Expected	Result	Case Type
GetById	Valid comment ID	Post data	Pass	Normal
GetById	Invalid comment ID	Error	Pass	Error

Update

Functionality	Test Case	Expected	Result	Case Type
Update	Valid comment ID	Update comment data	Pass	Normal
Update	Invalid comment ID	Error	Pass	Error
Update	Blank update data	Original comment data	Pass	Normal

Delete

Functionality	Test Case	Expected	Result	Case Type
Hide	Valid comment ID	Set isActive field to false	Pass	Normal
Hide	Invalid comment ID	Error	Pass	Error

User-interface Testing (Michael McCulloch)

No specific framework or methodology was used for testing the user-interface. Instead mock data was used to manually test UI for expected results.

Integration Testing (Michael McCulloch)

No specific framework or methodology was used for testing integration between server and client. Instead server and client code were tested separately. Following these tests, mock data for individual components was removed and replaced by server provided data. The completion of this incremental process allowed for testing the integration between client and server.

Quality Assurance Testing (Michael McCulloch)

Quality assurance testing was accomplished through continued use of the site by each member of the development team. Each member created a user, logged-in, and assumed the role of a user. This allowed for testing site functionality in real-world scenarios.