**CS 1530 - Group 4**

**Milestone 4**

**Alec, Tim, Joey, Jeremy, Bert**

**Object Oriented Analysis for Online Testing System  
1. System Overview**

CodeFeed is a social media interface designed for software engineers, by software engineers. This web application will implement certain functionalities from already popular social-media platforms, including: Twitter, Reddit and Stack Overflow. However, CodeFeed will combine and modify the best features of these platforms to create a unique and refreshing user experience. This platform will be sure to attract developers who wish to create quality and inspirational codebases. The application will run on the most stable and newest versions of Python, Flask, SQLAlchemy, Google Chrome, JavaScript, HTML5, CSS3, and PostgreSQL. This will ensure the application is modern, scalable, and professional across all aspects of the stack, while also being easy to test and maintain. To test the application, we will use a combination of unit, integration, and mutation testing.

**2. The Class Model** 2.1. The Classes

User (id, username, password\_hash name, email, biography, creation\_date, isValidated, last\_login)

Category (id, name, description)

Thread (id, category\_id, user\_id, title, body, creation\_date)

Comment (id, thread\_id, user\_id, body, creation\_date)

Friendship (id, user1\_id, user2\_id, creation\_date)

ThreadVote (id, thread\_id, user\_id, value)

CommentVote (id, comment\_id, user\_id, value)

Message (id, user1\_id, user2\_id, body, creation\_date)

2.2. The Class Diagram

**3. The Dynamic Model** 3.1. The Scenarios

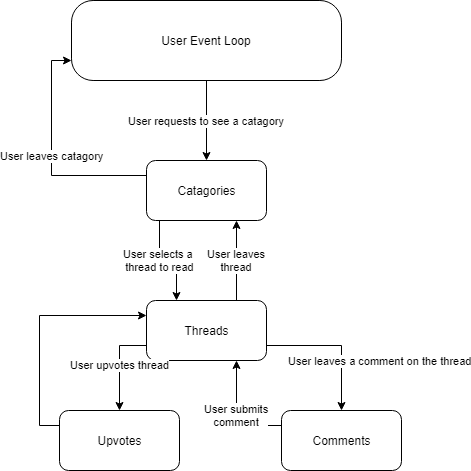
Scenario 1:

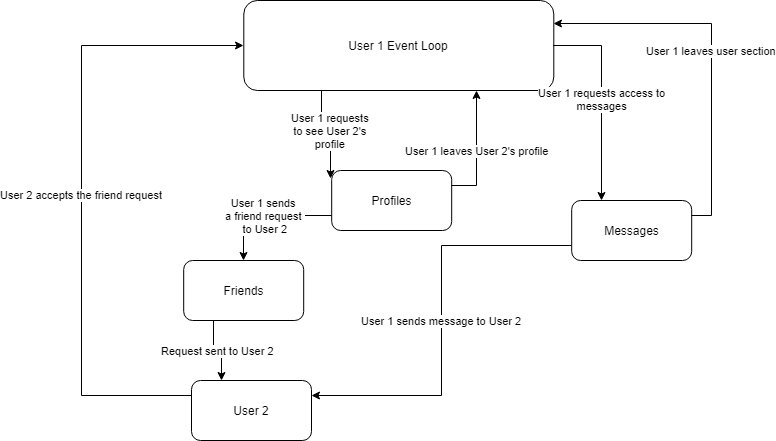
* User requests to see a category
* User selects a thread to read
* User upvotes the thread
* User comments on the thread

Scenario 2:

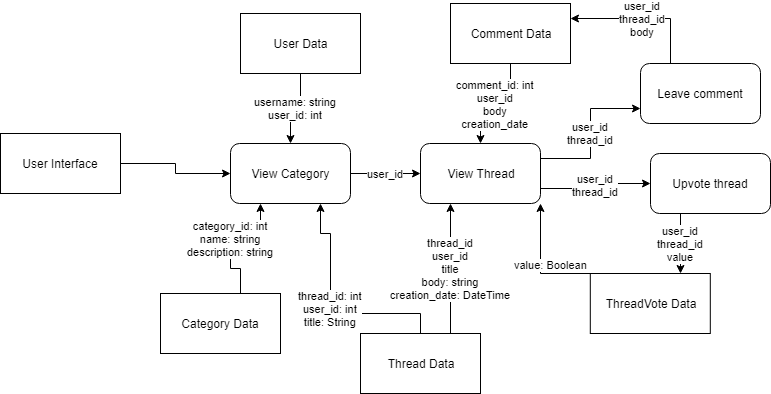
* User 1 requests to see User 2’s profile
* User 1 sends a friend request to User 2
* User 2 accepts the friend request
* User 1 sends a message to User 2
* User 2 receives the message.

3.2. The State Diagrams



****

**4. The Functional Model**

****

**Object Oriented Design for Online Testing System**

**Class Name:** Category

**Attributes:** id: int

name: string

description: string

**Module Name:** Category::createCategory **Module Type:** Method **Return Type:** void **Input arguments:** name, description **Output arguments:** None **Error messages:** If a category with the same name already exists **Files accessed:** None **Files changed:** Category database **Modules called:** None **Narrative:** Creates a new category with the given name and description

**Module Name:** Category::getThreads **Module Type:** Method **Return Type:** list of tuples **Input arguments:** category\_id **Output arguments:** None **Error messages:** If the category doesn't exist **Files accessed:** Category database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all threads within the given category. The list is comprised of tuples containing the information of each thread.

**Class Name:** Comment

**Attributes:** id: int

user\_id: int

thread\_id: int

body: string

creation\_date: DateTime

**Module Name:** User::commentOnThread **Module Type:** Method **Return Type:** void **Input arguments:** user\_id, thread\_id, body **Output arguments:** None **Error messages:** If the thread or user doesn’t exist or the body is empty **Files accessed:** Thread database **Files changed:** Comment database **Modules called:** None **Narrative:** Add a new comment to an existing post

**Module Name:** Comment::editComment **Module Type:** Method **Return Type:** void **Input arguments:** comment\_id, user\_id, body

**Output arguments:** None **Error messages:** If the user trying to edit the comment did not create it **Files accessed:** None **Files changed:** Comment database **Modules called:** None **Narrative:** Checks to see if the user created the comment. If so, updates the comment’s body in the database

**Module Name:** Comment::getComments **Module Type:** Method **Return Type:** list of comments **Input arguments:** user\_id **Output arguments:** None **Error messages:** If the user doesn’t exist **Files accessed:** Comment database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all comments created by the given user

**Class Name:** CommentVote

**Attributes:** id: int

user\_id: int

comment\_id: int

value: boolean

**Module Name:** CommentVote::deleteVote **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, comment\_id **Output arguments:** None **Error messages:** If the user or comment do not exist or if no vote by the given user on the given comment exist **Files accessed:** None **Files changed:** CommentVotes database **Modules called:** hasVoted **Narrative:** Deletes the vote by the given user on the given comment

**Module Name:** CommentVote::hasVoted **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, comment\_id **Output arguments:** None **Error messages:** If the user or comment do not exist **Files accessed:** CommentVotes database **Files changed:** None **Modules called:** None **Narrative:** Checks the database for a vote by the given user on the given comment

**Module Name:** CommentVote::vote **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, comment\_id, value **Output arguments:** None **Error messages:** If the user or comment do not exist **Files accessed:** None **Files changed:** CommentVotes database **Modules called:** hasVoted **Narrative:** Creates a new CommentVote for the given user on the given comment. If the user already voted on the comment, change the value of the vote to the new value

**Class Name:** Friendship

**Attributes:** id: int

user1\_id: int

user2\_id: int

creation\_date: DateTime

**Module Name:** Friendship::areFriends **Module Type:** Method **Return Type:** boolean **Input arguments:** user1\_id, user2\_id **Output arguments:** None **Error messages:** If one of the users doesn’t exist **Files accessed:** Friendship database **Files changed:** None **Modules called:** None **Narrative:** Checks if there is a friendship between the two users

**Module Name:** Friendship::createFriendship **Module Type:** Method **Return Type:** void **Input arguments:** user1\_id, user2\_id **Output arguments:** None **Error messages:** If one of the users don’t esist or a friendship between them already exists **Files accessed:** User database **Files changed:** Friendship database **Modules called:** areFriends **Narrative:** Creates a friendship between the two users in the database

**Module Name:** Friendship::deleteFriendship **Module Type:** Method **Return Type:** void **Input arguments:** user1\_id, user2\_id **Output arguments:** None **Error messages:** If the users are not already friends **Files accessed:** None **Files changed:** Friendship database **Modules called:** areFriends **Narrative:** Deletes the friendship between the two users in the database

**Module Name:** Friendship::getFriends **Module Type:** Method **Return Type:** list of users **Input arguments:** user\_id **Output arguments:** None **Error messages:** If the user doesn’t exist **Files accessed:** Frienship database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all users that are friends with the given user.

**Class Name:** Message

**Attributes:** id: int

user1\_id: int

user2\_id: int

body: string

creation\_date: DateTime

**Module Name:** Message::getMessages **Module Type:** Method **Return Type:** list of messeages **Input arguments:** user\_id **Output arguments:** None **Error messages:** If the user doesn’t exist **Files accessed:** Message database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all messages sent to the given user

**Module Name:** Message::getSentMessages **Module Type:** Method **Return Type:** list of messages **Input arguments:** user\_id **Output arguments:** None **Error messages:** If the user doesn’t exist **Files accessed:** Message database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all messages sent by the given user.

**Module Name:** Message::sendPrivateMessage **Module Type:** Method **Return Type:** void **Input arguments:** user1\_id, user2\_id, body **Output arguments:** None **Error messages:** If the users are not friends **Files accessed:** None **Files changed:** Message database **Modules called:** areFriends **Narrative:** Check if the two users are friends. If they are send the message to user 2

**Class Name:** Thread

**Attributes:** id: int

user\_id: int

category\_id: int

title: string

body: string

creation\_date: DateTime

**Module Name:** Thread::createThread **Module Type:** Method **Return Type:** void **Input arguments:** user\_id, category\_id, title, body **Output arguments:** None **Error messages:** If the creation fails **Files accessed:** None **Files changed:** Thread database **Modules called:** None **Narrative:** Creates a new thread

**Module Name:** Thread::deleteThread **Module Type:** Method **Return Type:** void **Input arguments:** user\_id, thread\_id **Output arguments:** None **Error messages:** If the user didn’t create the thread **Files accessed:** None **Files changed:** Thread database **Modules called:** None **Narrative:** Deletes an existing thread

**Module Name:** Thread::editThread **Module Type:** Method **Return Type:** void **Input arguments:** user\_id, thread\_id, body **Output arguments:** None **Error messages:** If the user did not create the post. **Files accessed:** None **Files changed:** Thread database **Modules called:** None **Narrative:** Modify an existing thread

**Module Name:** Thread::getThreads **Module Type:** Method **Return Type:** list of threads **Input arguments:** user\_id **Output arguments:** None **Error messages:** If the user doesn’t exist **Files accessed:** Thread database **Files changed:** None **Modules called:** None **Narrative:** Returns a list of all threads created by the given user

**Module Name:** Thread::voteCount **Module Type:** Method **Return Type:** int **Input arguments:** thread\_id **Output arguments:** None **Error messages:** If the thread doesn’t exist **Files accessed:** Thread database **Files changed:** None **Modules called:** None **Narrative:** Checks each vote on the given thread and tallys them, returning the final score for the thread.

**Class Name:** ThreadVote

**Attributes:** id: int

user\_id: int

thread\_id: int

value: boolean

**Module Name:** ThreadVote::deleteVote **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, thread\_id **Output arguments:** None **Error messages:** If the user or thread do not exist or if no vote by the given user on the given thread exist **Files accessed:** None **Files changed:** ThreadVotes database **Modules called:** hasVoted **Narrative:** Deletes the vote by the given user on the given thread

**Module Name:** ThreadVote::hasVoted **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, thread\_id **Output arguments:** None **Error messages:** If the user or thread do not exist **Files accessed:** ThreadVotes database **Files changed:** None **Modules called:** None **Narrative:** Checks the database for a vote by the given user on the given thread

**Module Name:** ThreadVote::vote **Module Type:** Method **Return Type:** None **Input arguments:** user\_id, thread\_id, value **Output arguments:** None **Error messages:** If the user or thread do not exist **Files accessed:** None **Files changed:** ThreadVotes database **Modules called:** hasVoted **Narrative:** Creates a new ThreadVote for the given user on the given thread. If the user already voted on the thread, change the value of the vote to the new value

**Class Name:** User

**Attributes:** id: int

username: string

password\_hash: string

name: string

email: string

creation\_date: DateTime

is\_validated: boolean

last\_login: DateTime

**Module Name:** User::friendOtherUser **Module Type:** Method **Return Type:** void **Input arguments:** user1\_id, user2\_id **Output arguments:** None **Error messages:** If one of the users declines **Files accessed:** None **Files changed:** Friendship database **Modules called:** createFrienship **Narrative:** Allows a user to friend another

**Module Name:** User::login **Module Type:** Method **Return Type:** void **Input arguments:** username, password **Output arguments:** None **Error messages:** If username and password aren't valid **Files accessed:** None **Files changed:** User **Modules called:** None **Narrative:** Allow the user to login if their credentials are correct and update their last\_login

**Module Name:** User::modifyUserDetails **Module Type:** Method **Return Type:** void **Input arguments:** name, email, password, verify\_password, biography **Output arguments:** None **Error messages:** If the two passwords don’t match or the email is not of valid form **Files accessed:** None **Files changed:** User database **Modules called:** None **Narrative:** Get the new details for the user and update them in the database

**Module Name:** User::register **Module Type:** Method **Return Type:** void **Input arguments:** username, password, name, email **Output arguments:** None **Error messages:** If the creation fails **Files accessed:** None **Files changed:** Users database **Modules called:** None **Narrative:** Creates a new user in the database