|  |
| --- |
| detail of persons hands with scissors, markers, workingSystem Design |

|  |  |  |
| --- | --- | --- |
| Fanlinc |  |  |
| Aamir Haroon Abel Debalkew Prashyen Kanagarajah Carl Alvares  Rahul Ramani Ruidan Ji Tahasun Tarannum |  |  |

TABLE OF CONTENTS

[System Architecture 3](#_Toc501116467)

[Architecture Diagram 3](#_Toc501116468)

[System Architecture Description 4](#_Toc501116469)

Handling Exceptions [5](#_Toc501116470)

[System Interaction with Environment 6](#_Toc501116471)

CRC Cards 7

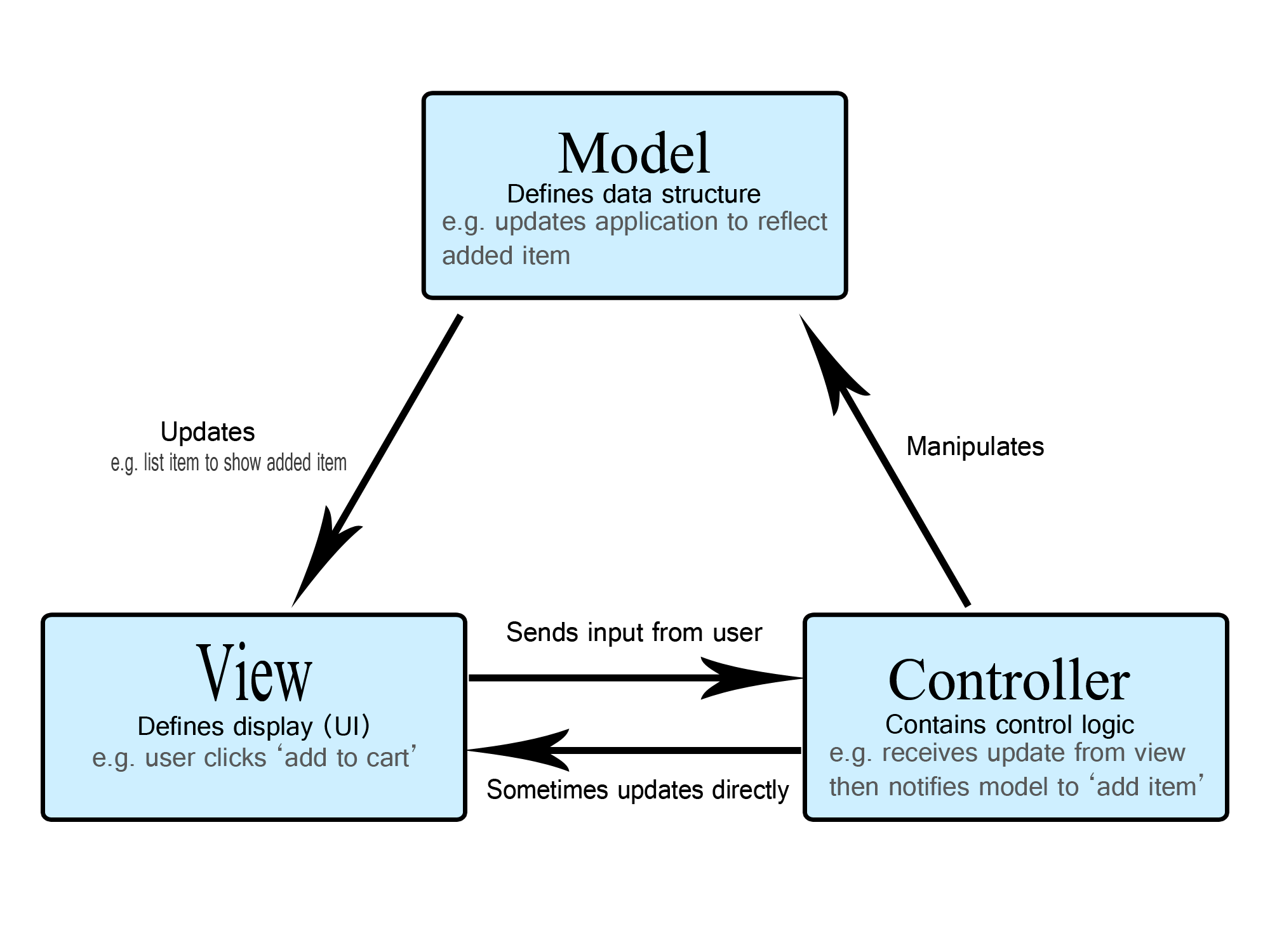
# System Architecture

## Architecture Diagram

A picture containing text, map

Description automatically generated

## System Architecture Description

The architecture of the System is designed based on the Model-View-Controller pattern. More information about the MVC pattern can be found here: https://www.geeksforgeeks.org/mvc-design-pattern/, where the blue boxes indicate the models, the orange boxes indicate the controller and the green boxes indicate the view. The Models are manipulated by the controllers, the controller receives information from the view to determine which models must be manipulated. Once the models are manipulated, the view is updated from the model. A diagram of the MVC pattern is displayed below.

Within the Model aspect, there are two types of objects, a node, and a relationship. Fandom, User and Post are all nodes whereas LIKED, POSTED\_IN, POSTED, and JOINED are all relationships. In this specific design, the relationships play the role of interconnecting the nodes. The information stored by each object is defined in the architecture diagram on page 3.

This design consists of three controllers, the AccountController, FeedController, and FandomController. The AccountController is responsible for profile or account related API calls such as creating an account or editing a profile. This controller interacts/manipulates the user model to acquire/update information about the user, as well as interacts or manipulates the LIKED and JOINED models, to keep track of posts liked by the user and fandoms joined by the user respectively. The FeedController is responsible for generating the feed with posts, it interacts and/or manipulates the post model to get or edit information about posts, and interacts with the POSTED\_IN model to determine the fandom the post belongs in. The FandomController is responsible for acquiring information about the fandoms, which interacts and/or manipulates the fandom model to get information about the fandoms.

The design focuses on four main views, the LoginPage, the SignUpPage, the ProfilePage, and the FeedPage. The LoginPage deals with scenarios in which the user attempts to click the sign-up button or inputs user info and clicks the login button. When the user clicks the signup button, the user gets redirected to the signup page. However, if the user attempts to log in, the AccountController is sent the input, which interacts with the user model and returns the response back to the view on whether the login was successful or not. The SignUpPage deals with the user attempting the create a new account, this scenario also interacts the AccountController, manipulating the user, and JOINED model and returning the response back to the view. The ProfilePage is for the user to either view or edit the profile, hence it interacts with the AccountController which manipulates or interacts with the user, JOINED, and LIKED models. The FeedPage is responsible for displaying posts, allowing the user to like and create posts. Thus, the FeedPage interacts with the FeedController, which manipulates the post, and POSTED\_IN models when creating or liking a post and interacts with the post and POSTED\_IN models when displaying posts.

## Handling Exceptions

There are numerous Exceptions that must be handled across the various pages. These Exceptions will be displayed in the UI with a dialog message describing the error. The status code will be based on the type of exception which can found at this link: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> .The LoginPage handles error caused by the username and/or password being incorrect using a UserNotFoundException which returns a 404 status code. If the user attempts to log in with the username, or password or both fields missing, a 400 stats code is returned by Spring. The SignUpPage handles errors caused by an account with the same username already existing when attempting the create an account with a UsernameNotUniqueException, which return a 409 Status code. Missing field errors when signing up are handled with a 400 status code by Spring. For the ProfilePage, when attempting to edit the profile the same exceptions are thrown as the SignUpPage, for the username not being unique and for missing fields. The FeedPage will throw an exception with 400 status code missing fields when attempting to post. For the case where the database can’t be reached, a 500 status code is returned, and the UI will display a Pop up with the message “Please restart the application”.

## System Interaction with Environment

Prior to running the application, certain assumptions must be made. A Neo4j database must be running on http://localhost:8080 and bolt://localhost:7687 with database username set to “Neo4j” and password set to “secret”.

# cRc cARDS

Class name: UserBuilder

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Creates a user

Collaborators:

\*j User

Class name: FandomBuilder

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Creates a Fandom

Collaborators:

\*j Fandom

Class name: PostBuilder

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Creates a Post

Collaborators:

\*j Post

Class name: AccountController

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Profile or account related API calls such as creating an account or editing a profile

\*h interacts/manipulates the user model to acquire/update information about the user

\*i interacts or manipulates the LIKED and JOINED models, to keep track of posts liked by the user and fandoms joined by the user respectively.

Collaborators:

\*j AccountService

Class name: FeedController

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Generates the feed with posts

\*h interacts and/or manipulates the post model to get or edit information about posts

\*i interacts with the POSTED\_IN model to determine the fandom the post belongs in

Collaborators:

\*j Feed

Class name: FandomController

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Responsible for acquiring information about the fandoms,

\*h interacts and/or manipulates the fandom model to get information about the fandoms.

Collaborators:

\*j Fandom

Class name: UsernameNotUniqueException

Parent class (if any): RuntimeException

Subclasses (if any):

Responsibilities:

\*g Exception error for throwing when a Username is not unique

Collaborators:

\*j

Class name: UserNotFoundException

Parent class (if any): RuntimeException

Subclasses (if any):

Responsibilities:

\*g Exception error that is thrown when a User is not found

Collaborators:

\*j

Class name: User

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Holds all the variables and methods of a user class

Collaborators:

\*j

Class name: Fandom

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Holds all the variables and methods of a Fandom class

Collaborators:

\*j

Class name: Post

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Holds all the variables and methods of a Post class

Collaborators:

\*j

Class name: UserRespository

Parent class (if any): Neo4jRepository

Subclasses (if any):

Responsibilities:

\*g Interface with method to find a user given a username

Collaborators:

\*j UserRespository

Class name: AddUserRequest

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Request to add a user

Collaborators:

\*j

Class name: ValidateUserRequest

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Request to validate User

Collaborators:

\*j

Class name: AddUserResponse

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Response when user added

Collaborators:

\*j

Class name: ValidateUserResponse

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Response when user validated

Collaborators:

\*j

Class name: AccountService

Parent class (if any):

Subclasses (if any):

Responsibilities:

\*g Username and password validation

\*h Adds Users in the database

Collaborators:

\*j UserRespository