# SEG Lab-Project
(King's Connect Forum)

Alin Fulga
Dhairya Mehta
Aflal Asker
Jakub Uher
Jack Hinshelwood

The purpose of this design document is to discuss the requirements of our question and answer forum and design decisions conducted for the forum. This will help to successfully document an existing system and finally this will help us to produce a description from which we will implement our system. For this design document, we will explore different types of models that will give additional details about our project and how we have explored all different possibilities for our project. In this document, we have implemented different models such as the use case diagram, class diagram, and activity diagram.

Use Case Diagram
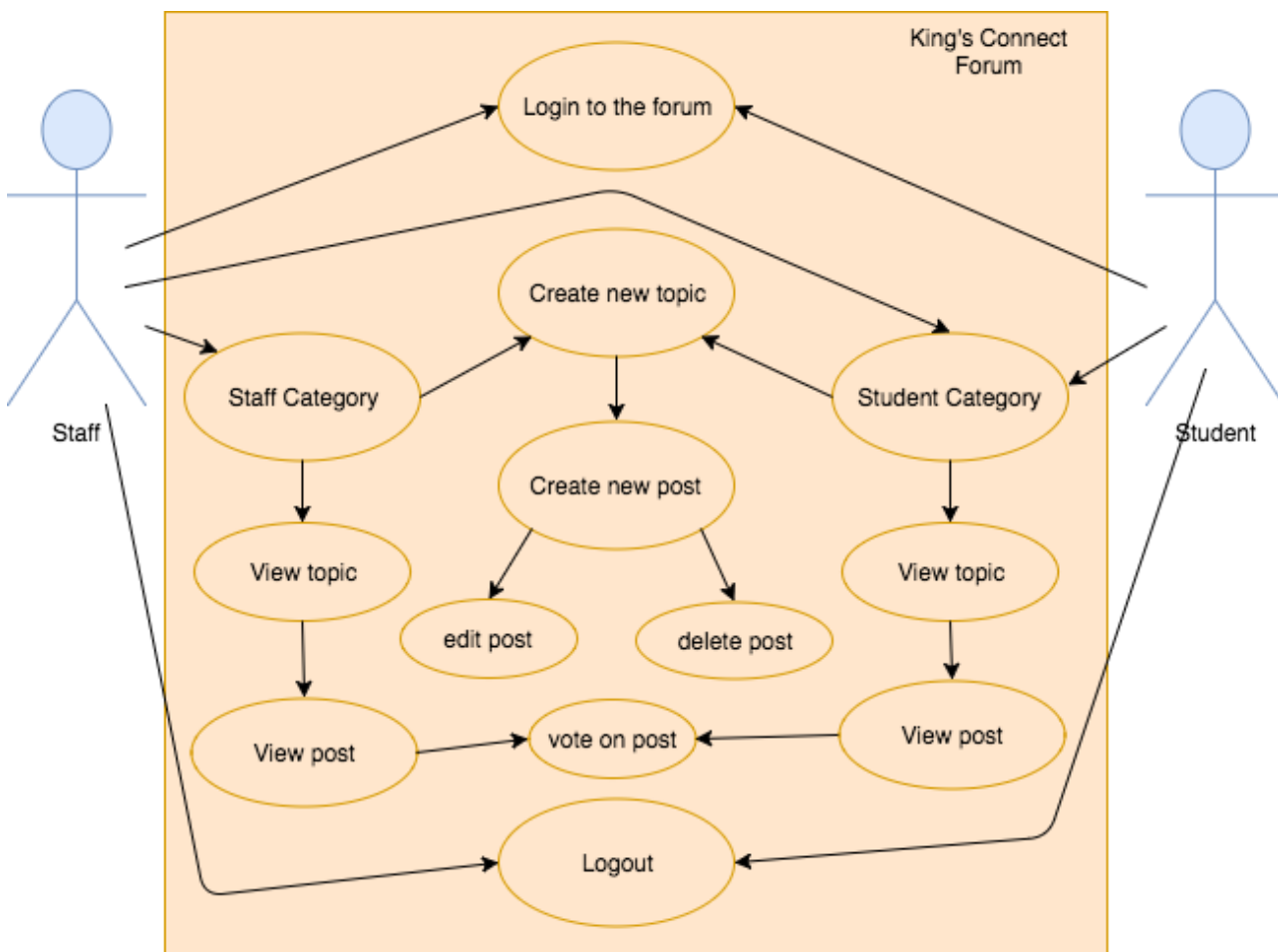
This shows a simple representation of the staff and student's interaction with the question & answer forum in the system that shows the relationship between the user and all the different use cases in which the user is involved in. As you can see the 'actors' have the functionality to login to the system as well as log out of the system. The 'actors' can only login if they have the correct KCL credentials. When they are logged in they will be able to enter one of two categories which is the staff or the student category, this

will depend on the who the user is. We assumed that staff members can also view the student category to keep an eye of the posts conducted in that section of the forum. They can then create a new post which could be a specific subject such as "Computer Science" and then within that post they can create different topics such as "Programming" or "Software Engineering group project". The author can then edit the post or delete the post if only he is the one who created it. Another user can also comment on another post. They can also vote on a post. Finally, the 'actors' can logout of the forum to keep their account secure.

Use Case Description

Use case name	Create a new Task
Goal in context	A staff or student can log in and create a new topic.
Preconditions	The system only allows staff to go the staff category and student to the student category.
Successful end condition	A staff or a student can log out.
Failed end condition	Staff or student login fails if they are not part of the King's college London university.
Primary actors	Staff and Student
Secondary actors	n/a
Trigger	staff or student can view topics, edit topics and delete topics.

Use case name	Creating a new task	
Main Flow	Step	Action
	1	The staff or student attempts to log in with the correct credentials (KCL details)
	2	The staff or student can view current categories, depending on the student they can only go to that category (e.g. student will only be able to access the student category, staff can view both categories)
	3	Staff or student can view topic or create some new topics
	4	Staff or student can create posts, edit posts, delete posts and vote on posts.
	5	student and staff can log out of the question/answer forum.
Extensions	1.1	The staff or student cannot login as they do not have the correct credentials

Use case diagrams are not the complete description of the of the system so all use case diagrams are accompanied by use case descriptions. This shows the context of the system, pre-conditions, successful end condition, failed end condition, primary actors, secondary actors and triggers. The second part of the use case description shows the main flow of the how the forum works and shows extensions which needs to be taken into consideration for the forum.
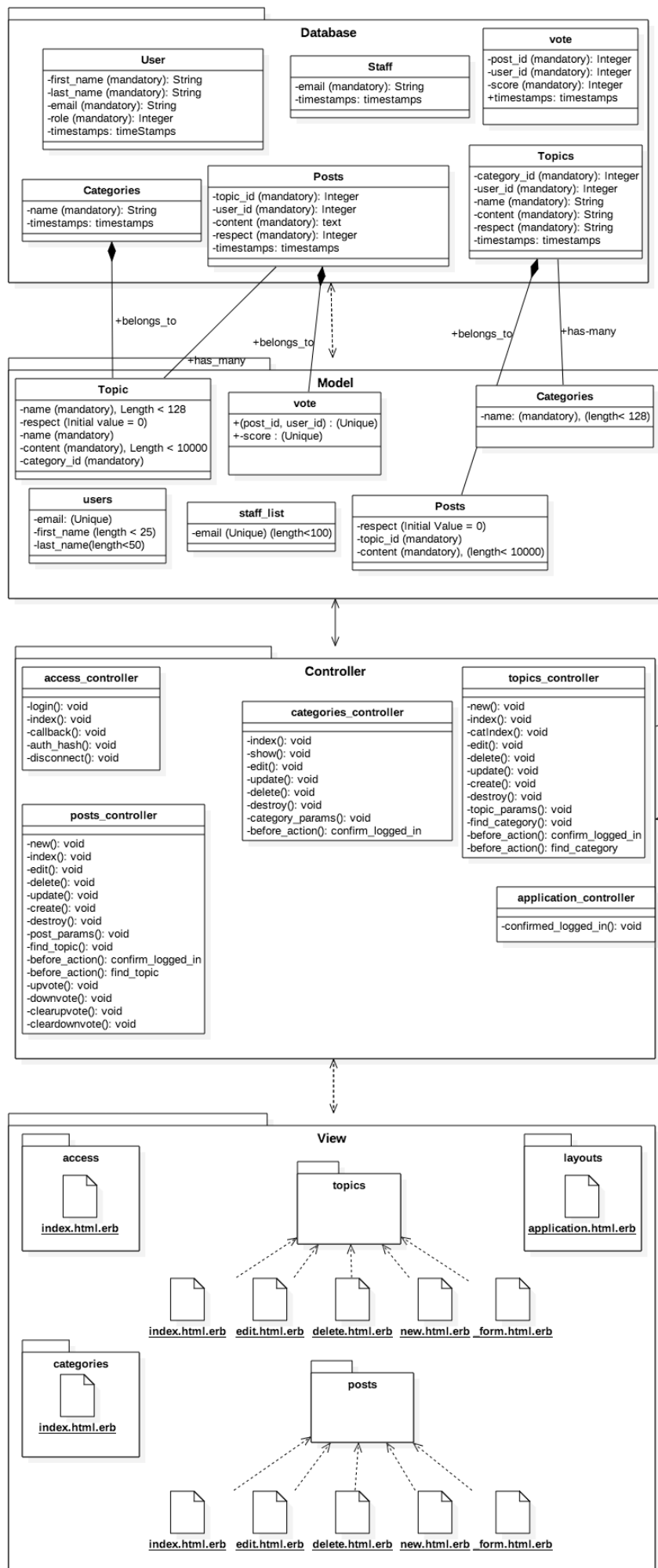
Activity Diagram
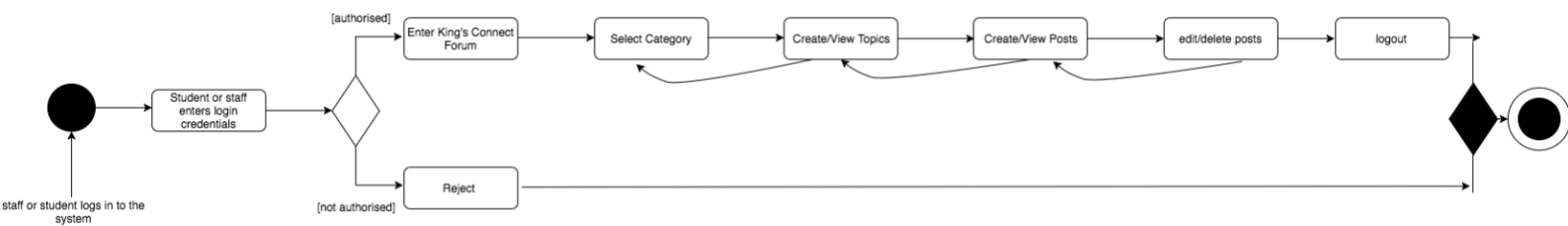
Activity diagram is very like flow charts in the way they are shown as a diagram, as you can see they describe processes in a system. They allow a user to easily document workflow in a system. It helps to capture the dynamic behaviour of the system. As you can see this activity diagram shows the activities, association, conditions and constraints. As you can see the initial node is when the staff or student to logs into the system. The actions which include entering the King's connect forum, selecting category, create and view category etc. You can also see a decision node which authorises the user if they have the right login credentials. Then the final node is shown after the user has logged out.

Class Diagram (Please see class diagram below)

This class diagram is a structure diagram which describes the components of a system. Class diagrams are a type of structure diagram because they describe what must be present in our system being modelled, this is typically used to document software architecture of our program. The class diagram is adhered to a Rails framework which uses the MVC (Model, View, Controller) structure. The first part is the database which is data storage access, this includes the following classes which are user, categories, staff, posts, vote and topics. The second part is the model which is the actual interface behind the application. The model consists of the following classes which are topic, users, vote, staff_list, categories and posts. There is a relation between our database and model implementation. For example, a topic belongs to a category, a post also belongs to a topic etc. The third part is the controller which acts as an interface between the view and the model. The controller consists of access_controller, post_controller, categories_controller, topics_controller and application_controller. The last component is the view which is used for all the User interface logic of the application. The view consists of five different packages which are access, categories, topics, posts and layouts and these packages embedded ruby html files which renders the UI of the application.

Database

User
-first_name (mandatory): String
-last_name (mandatory): String
-email (mandatory): String
-role (mandatory): Integer
-timestamps: timeStamps

Staff
-email (mandatory): String
-timestamps: timestamps

vote
-post_id (mandatory): Integer
-user_id (mandatory): Integer
-score (mandatory): Integer
+timestamps: timestamps

Categories
-name (mandatory): String
-timestamps: timestamps

Posts
-topic_id (mandatory): Integer
-user_id (mandatory): Integer
-content (mandatory): text
-respect (mandatory): Integer
-timestamps: timestamps

Topics
-category_id (mandatory): Integer
-user_id (mandatory): Integer
-name (mandatory): String
-content (mandatory): String
-respect (mandatory): String
-timestamps: timestamps

+belongs_to

+belongs_to

+has_many

+belongs_to

+has-many

The user makes a request for a page using a browser and the routes.rb file directs the request to the appropriate html file.

Model

Topic
-name (mandatory), Length < 128
-respect (Initial value = 0)
-name (mandatory)
-content (mandatory), Length < 10000
-category_id (mandatory)

vote
+(post_id, user_id) : (Unique)
+-score : (Unique)

Categories
-name: (mandatory), (length< 128)

users
-email: (Unique)
-first_name (length < 25)
-last_name(length<50)

staff_list
-email (Unique) (length<100)

Posts
-respect (Initial Value = 0)
-topic_id (mandatory)
-content (mandatory), (length< 10000)

Controller

access_controller
-login(): void
-index(): void
-callback(): void
-auth_hash(): void
-disconnect(): void

categories_controller
-index(): void
-show(): void
-edit(): void
-update(): void
-delete(): void
-destroy(): void
-category_params(): void
-before_action(): confirm_logged_in

topics_controller
-new(): void
-index(): void
-catIndex(): void
-edit(): void
-delete(): void
-update(): void
-create(): void
-destroy(): void
-topic_params(): void
-find_category(): void
-before_action(): confirm_logged_in
-before_action(): find_category

Web browser

config

routes.rb

posts_controller
-new(): void
-index(): void
-edit(): void
-delete(): void
-update(): void
-create(): void
-destroy(): void
-post_params(): void
-find_topic(): void
-before_action(): confirm_logged_in
-before_action(): find_topic
-upvote(): void
-downvote(): void
-clearupvote(): void
-cleardownvote(): void

application_controller
-confirmed_logged_in(): void

View

access
index.html.erb

topics

layouts
application.html.erb

index.html.erb   **edit.html.erb**   **delete.html.erb**   **new.html.erb**   **_form.html.erb**

categories
index.html.erb

posts

index.html.erb   **edit.html.erb**   **delete.html.erb**   **new.html.erb**   **_form.html.erb**