# ABSTRACT

The aim of this project is to develop a functioning web forum application. The forum serves as a single place where various users can create an account and discuss topics on new and existing claims. The client side will be built using HTML 5 and ECMAScript 6 while the server side will be built using Flask, MySQL and Python 3

# ARCHITECTURAL DESIGN

## Client Side

The client is a program that allows the user to make requests through the web via web browsers. The client sends requests so as to access resources hosted on the server. The requests utilize the HTTP verbs i.e.

* GET: Fetch resource
* POST: Create resource
* PUT: Update/modify resource
* DELETE: Delete/remove resource

To access the website, the client will send a GET request to access the files hosted on the server. The server will then send back hosted files containing HTML and JavaScript code to the client for it to render on the browser.

## Server Side

The server is the back-end of the program

Communication between the client and server will be based on the REST principles using REST APIs. REST API is a way of thinking of how a web server behaves and responds to a request. These REST APIs respond with data and/or resources

Once the URL is keyed in, the client sends a GET request to the server. The server will return a response which will either be an error, HTML code or text.

On the landing page of the forum will be a dialogue box prompting the user to sign in if he/she has an account or to register for one

## Authentication

For a user to log in, he/she must already have been registered and their details must be in our database. Details required to log in are:

* Username or email address
* Password

We shall use Flask to build the web server that will communicate with the client. The data will be stored in a MySQL database.

The purpose of authentication is to ensure that resources on the server are only available to users that are logged in

CLIENT

Jwt Stored

STORAGE

Authentication data sent by client

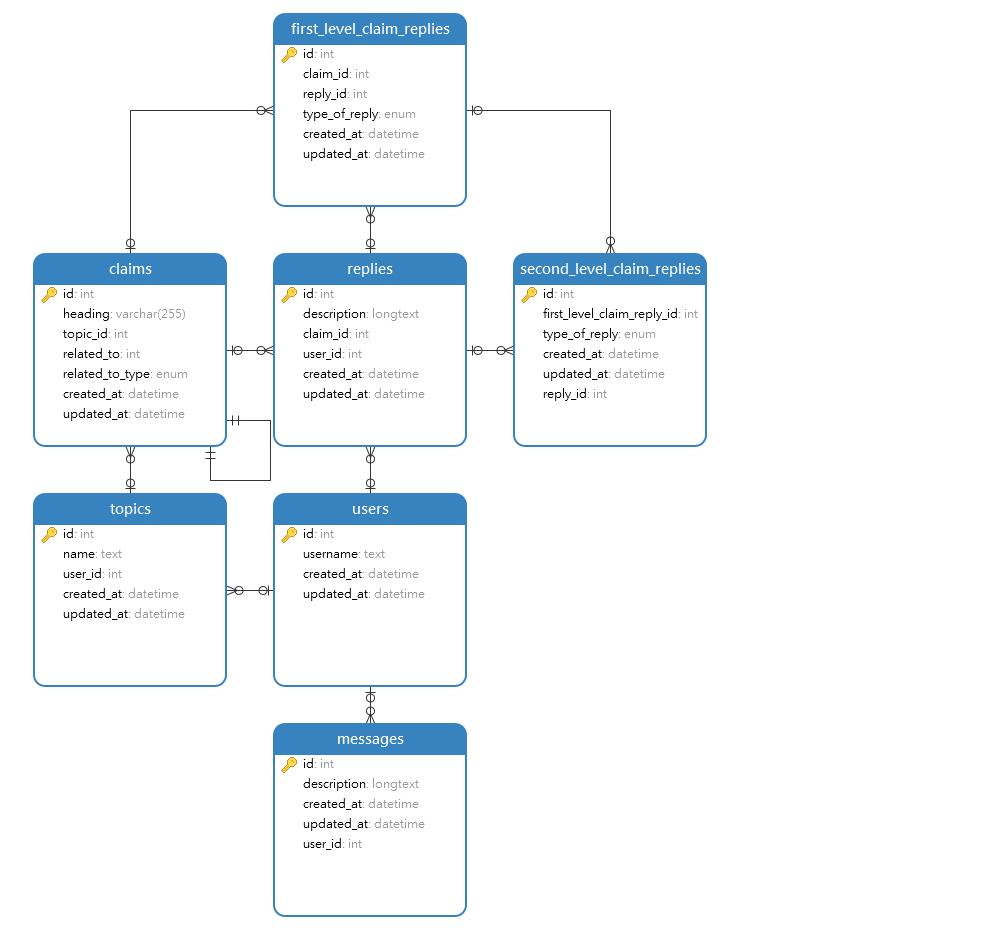
JSON Web Token

SERVER

Authentication works by the client sending the data to the server. In the server, the data will be verified to check if the user details actually exist. Once verification is complete, the server will return a JSON Web Toke (JWT). This is an object or a piece of data that contains a signature that can be verified on the server. The token can be stored by the client and can be attached to future requests

Once the above are keyed in, the client sends a GET request to the server. The server contains resources which interact with our request. The server will then interact with our MySQL database which will check whether the user actually exists and communicate to the server which will then respond to the client.

## Database

****

The database was designed based on the functionality required for the forum. Key elements required their own tables to capture the relevant information that would be collected while the forum is in operation i.e.,

* Users
* Messages
* Topics
* Replies

***Users*** table consists of persons who register on the forum. Attributes of a user include:

* **Id,** an integer value which will also be the primary key for the table
* **Unique username** that users can identify each other on the forum
* Created\_at which, a datetime value showing when a user is created
* Updated\_at, a date value showing the date when the attribute of a user is modified

The Users table will have a one-to-many relationship with the following tables:

* Messages
* Replies
* Topics

***Messages*** table consists of all messages posted by a user. It has a one-to-many relationship with the *users* table, with the latter serving as the parent table. Attributes of a message include:

* Id, an integer value to uniquely identify each message. This also serves as the primary key
* Description, a longtext which is the message posted by the user
* Created\_at, a datetime value showing the day and time a message was created
* User\_id, an integer value linking the *users* table from the one-to-many relationship

***Topics*** table consists of topics posted on the forum. Like the *messages* table, it has a one-to-many relationship with the *users* table and the *claims* table, the former serving as the parent table and the latter as the child table. Its attributes are:

* Id, an integer value to uniquely identify each topic.
* Name, a text value that shows the topic name.
* User\_id, an integer value linking the *users* table
* Created\_at, a datetime value showing when a topic was created
* Updated\_at, a datetime value showing when any attribute of the topic is changed

***Claims*** table consists of claims posted in a topic. It has a one-to-many relationship with the *topics, replies* and *first\_level\_claim\_replies* tables with the former being the parent table and the latter 2 child tables, as well as a one-to-one relationship with itself. Its attributes are:

* Id, an integer value to uniquely identify each claim
* Heading, a varchar value that shows the claim heading
* Topic\_id, an integer value that links the *claims* and *topics* tables
* Related\_to, an integer value that shows the relationships between other claims.
* Related\_to\_type, an enum value that shows the link between claims. The claims can be related as opposed or equivalent. In the table, these 2 options are enumerated.
* Created\_at, a datetime value shows when the claim was created.
* Updated\_at, a datetime value shows when an attribute of the claim was changed.

***Replies*** table consists of message replies by users. It has one-to-many relationship with the *second\_level\_claim\_replies,* *users* and *claims,* with the former being a parent table and the latter 2 child tables. Its attributes are:

* Id, an integer value that uniquely identifies each reply
* Description, a longtext value that shows whether the reply posted by a user.
* Claim\_id, an integer value that links the *claims* to *replies*
* Created\_at, a datetime value showing when a reply was posted
* Updated\_at, a datetime value showing when the reply was modified

***First\_level\_claim\_replies*** table consists of type of replies, where the reply is in response to a claim i.e., clarification, supporting argument or counterargument. It has a one-to-many relationship with *claims* and *replies*, and in each relationship, it is the child table. Its attributes are:

* Id, an integer value that uniquely identifies each type of reply
* Claim\_id, an integer value that links *first\_level\_claim\_replies* to *claims.*
* Reply\_id, an integer value that links *first\_level\_claim\_replies* to *replies.*
* Type\_of\_reply, an enum value that shows the enumerated values of the type of replies
* Created\_at, a datetime value that shows when the type of reply was created.
* Update\_at, a datetime value that shows when the type of reply was modified.

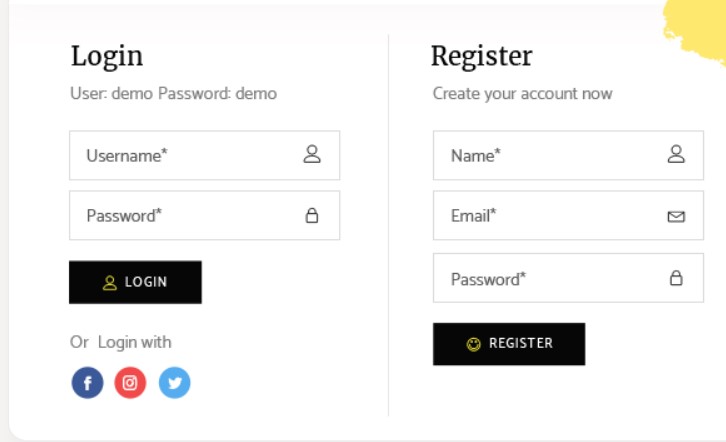
***Second\_level\_claim\_replies*** table consists or types of replies, where the reply is made in response to another reply. The type of replies are evidence, support or a rebuttal. It has a one-to-many relationship with *first\_level\_claim\_replies* and *replies,* where each of the 2 is a parent table to *second\_level\_claim\_replies.* Its attributes are:

* Id, an integer value that uniquely identifies each second level reply
* First\_level\_claim\_reply\_id, an integer value that links the *second\_level\_claim\_replies* to *first\_level\_claim\_reply*
* Type\_of\_reply, an enum value that shows the enumerated values of the types of replies
* Created\_at, a datetime value that shows when the second level reply was made
* Updated\_at, a datetime value that shows when the second level reply was modified
* Reply\_id, an integer value that links *replies* to *second\_level\_claim\_replies*

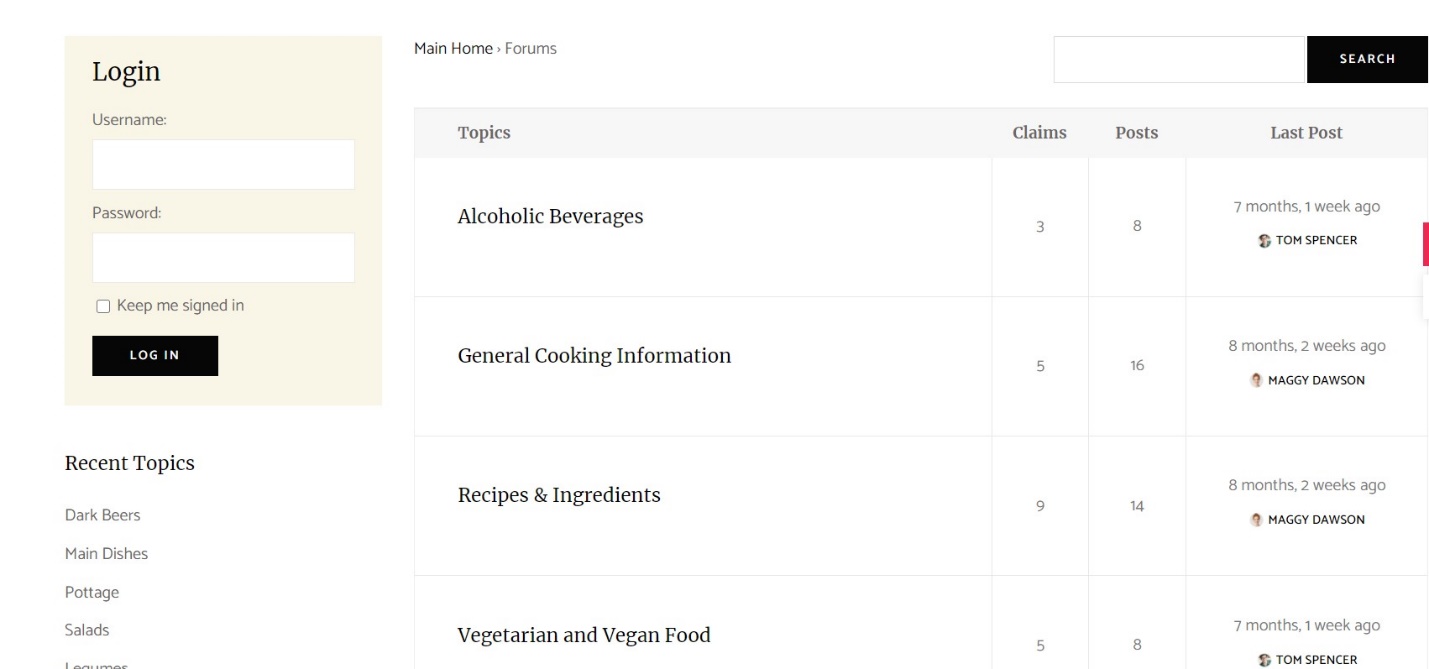
# High-Level Design (Wireframes)

The design for the user interface will be as follows:

## Login and Registration View

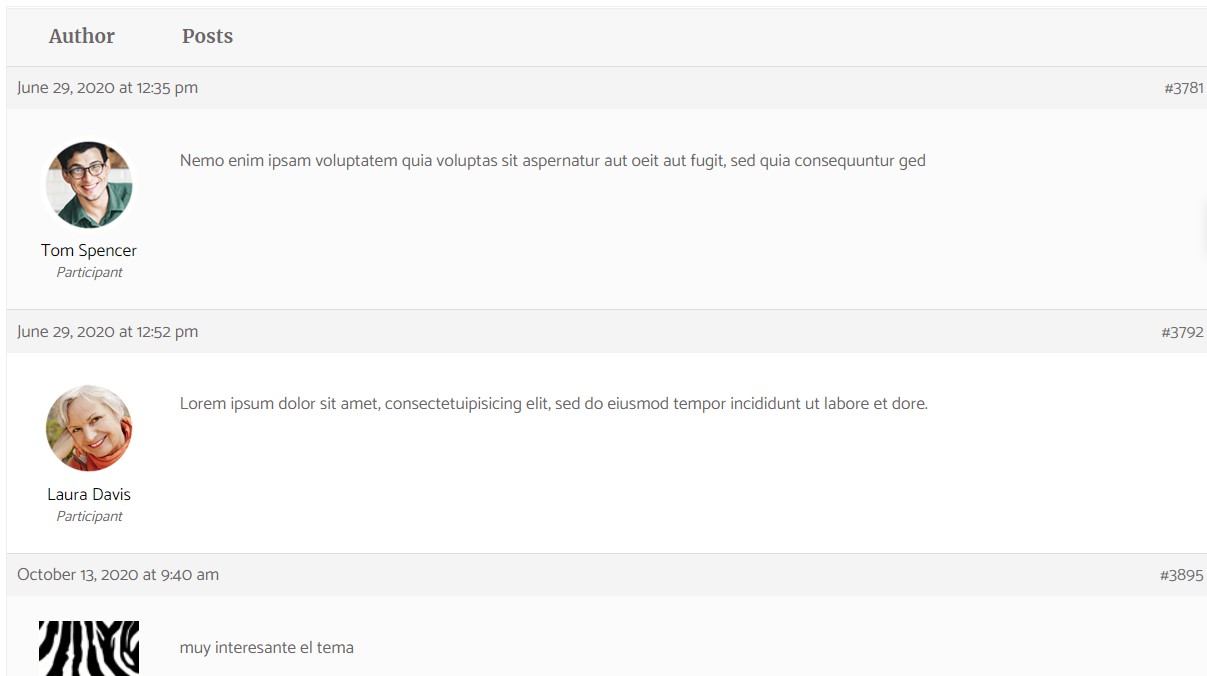
****

## All Topics View

****

## **All Claims views**

## Single Claim View

****