# 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 utilise 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 backend 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.

When registering, the user will have to fill in all of the required details in the form provided before pressing the register button. If the user fails to do this, he/she will be provided the appropriate form validation messages to ensure the right information is filled in.

Upon pressing the registration button, and assuming all appropriate fields have been filled, the form will submit the data based to the route indicated in the action method defined in the forms attributes and the server will also perform the appropriate validation. On successful validation, the server will write the user’s information, with the password encrypted using the Blowfish algorithm, to the database and provide a response to the client side that registration was successful. The client side will display the information and the user will be prompted to use the registration data (email and password) they provided to login.

When logging in, the user will supply the email and password. These details will be sent to the route indicated by the action attribute on the login form. The server will check if the email exists in the database and if found, it will proceed to check if the provided password matches the encrypted one in the database. If the password matches, then the server will begin a session on the browser with the user’s information stored in the session. If any of the checks fail, then the server will give an appropriate error handling message to inform the user of the error on the client side.

CLIENT

Authentication data sent by client

STORAGE

Jwt Stored

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.

**User Roles**

Roles for users in the system have been divided into:

1. Admin
2. Normal

Users with admin role have total control of the system. They can view posts, claims and replies by all users. They can also remove content by users which might not adhere to the website’s operation policy.

Users with normal role can only view posts by fellow users and cannot edit or delete posts by other users.

**Admin Functionality**

Administrators in the system will have mainly view only capability. They will act as moderators and make sure the forum community plays by the rules.

Due to the sensitive nature of being an administrator, an admin will have to be created by default when the system is run for the first time in a server. There will be a seeder file that will automatically be called with the necessary credentials to be persisted in the database.

Once the default credentials have been populated, the entrusted person who has them will have to login and invite other administrators. Invitation will be email based whereby an admin invites another admin by entering their email in the system.

Admins will have access to all the information of all the users including their replies and claims. Admins will login in by entering their email and password in the login pages and this will fire an ajax request to the server side which will check if the credentials provided match the credentials of a user with admin rights. Assuming this is successful, the admin will be redirected to the homepage where they will see a list of all users in the system including other admins except for themselves.

Upon clicking a user row in the client side, the admin will be displayed with a modal that will drop from the top of the page to the middle with the information of the user which will have been fetched asynchronously using ajax from the database by the server.

In the modal the admin will see basic information of the particular user including actions to toggle between suspending or reinstating a user with suspending a user being the default option.

In the modal, the admin will also have a table list view of all the replies the user has ever made with a toggle of either enabling or disabling each reply as the case may be.

**Normal User Functionality**

Upon successful login mentioned in the previous section, the user will be redirected to the all-topics view of the website. This will happen when the browser sends an ajax (Asynchronous JavaScript and XML) get request to the server-side. The server will query the database for all the topics. It will fetch all the topics and their attributes and inner join the count of all claims associated to each topic and count of all replies if available. Once the server does this, it will send the response to the client for it to render the page with all the details. There will be a fade out and fade in animation when rendering the all-topics view when moving from the authentication view.

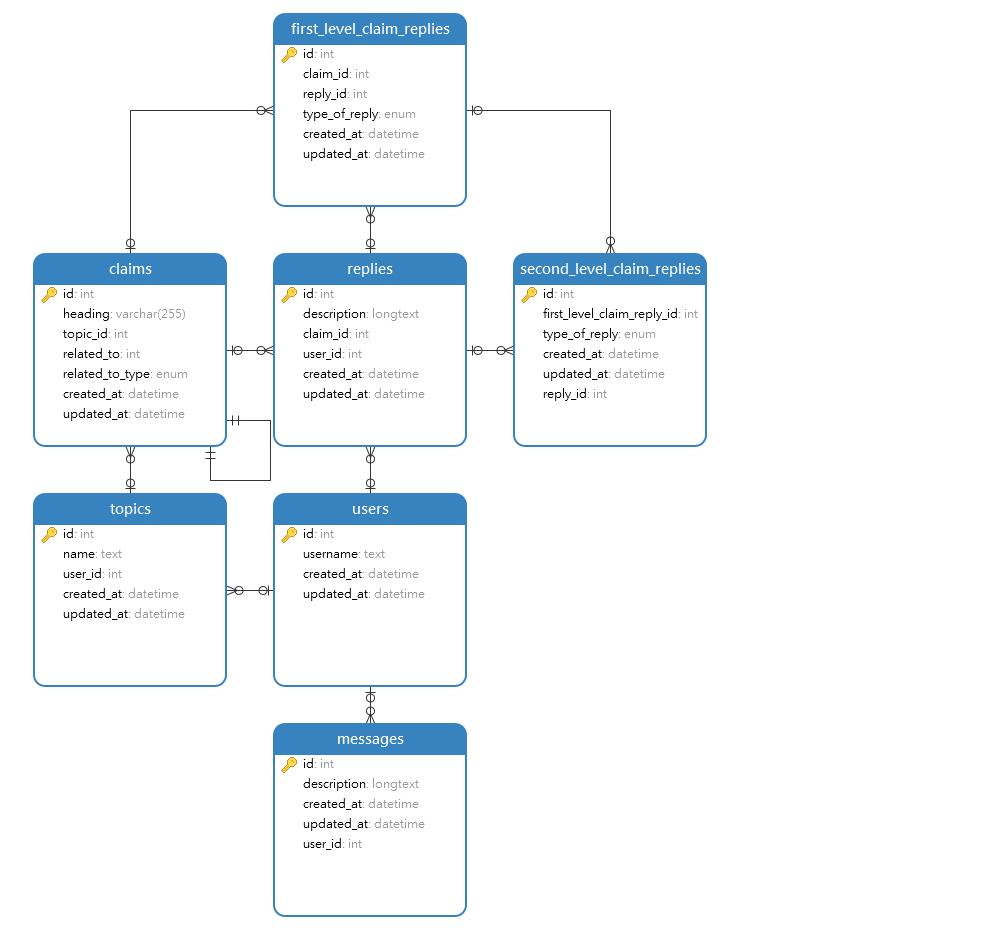
On the all-topics view, if a user clicks a single topic an ajax GET request will be fired to the server with the topic id as a parameter. The server will query all the necessary claims that are associated with the topic id that was passed as a parameter. Once the server gets this information from the database, it will render the all-claims view and render all the claims related to the topic id parameter. There will be a fade out and fade in animation when rendering the all-claims view when moving from the all-topics view.

When a single claim is clicked, an ajax request will be sent to the server which will query the database for all comments and replies associated to a particular claim. The server will then send the data to the client which will fade out the all-claims view and fade in single claim view using the jQuery fade functions.

Another function is that a user will be able to search the entire site using the search text box. To achieve this the user will have to enter the keyword he or she would like to search. The user will then press the enter or search button which will send an ajax POST request to the server with the request body set as the keyword the user entered. The server will then query the database for list of the columns that match the particular keyword and return the response to the client which will render the searched result in a list for the user to click the targeted content.

## Database

### Entity Relation Diagram

****

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.

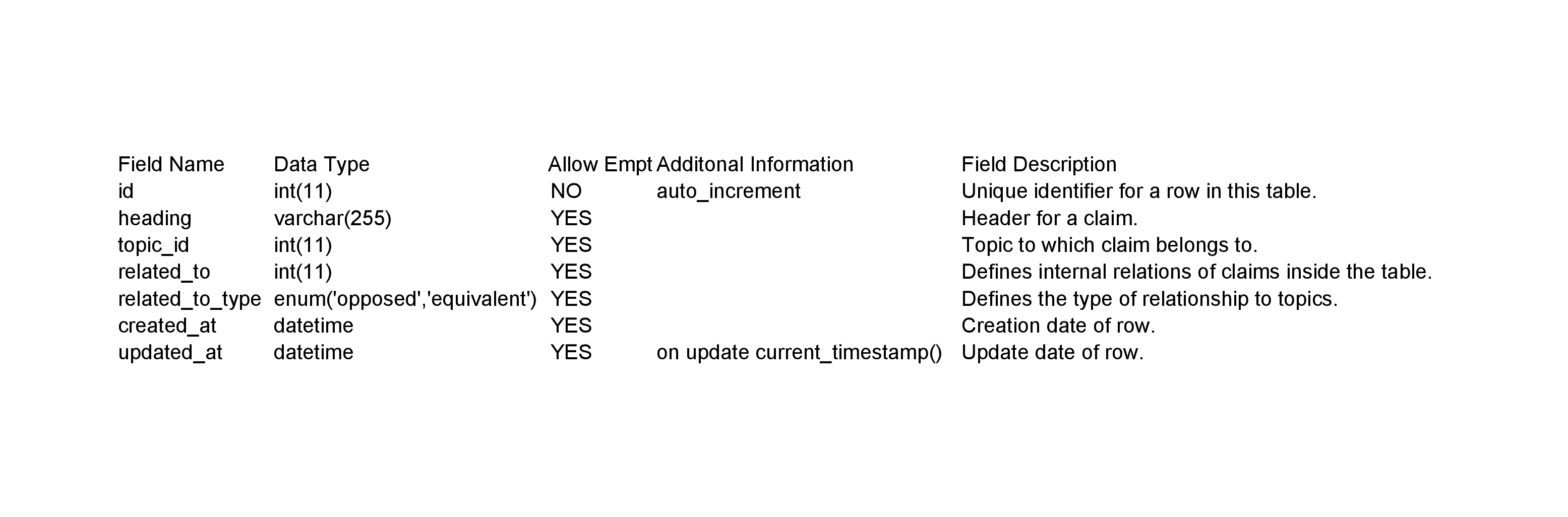
***Second\_level\_claim\_replies*** table consists or types of replies, where the reply is made in response to another reply. The type of replies are evidence, support or a rebuttal. It has a one-to-many relationship with *first\_level\_claim\_replies* and *replies,* where each of the 2 is a parent table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

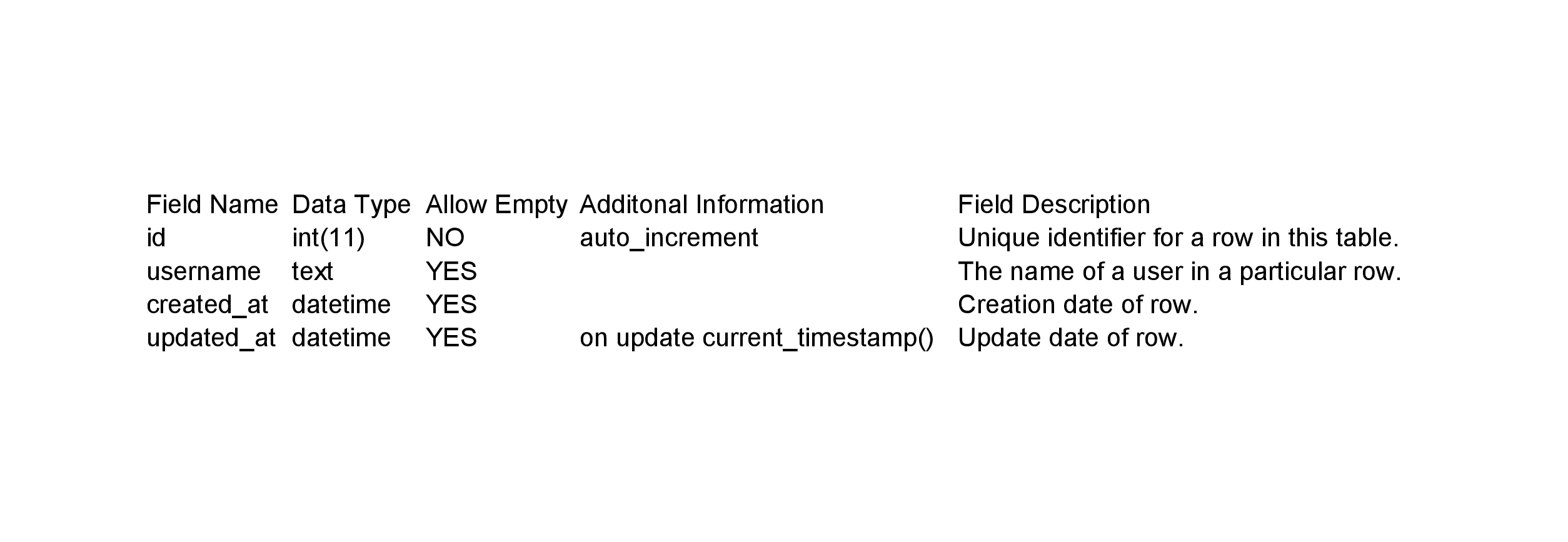
### Database Dictionary

The database dictionary of the forum consisted of the below content.

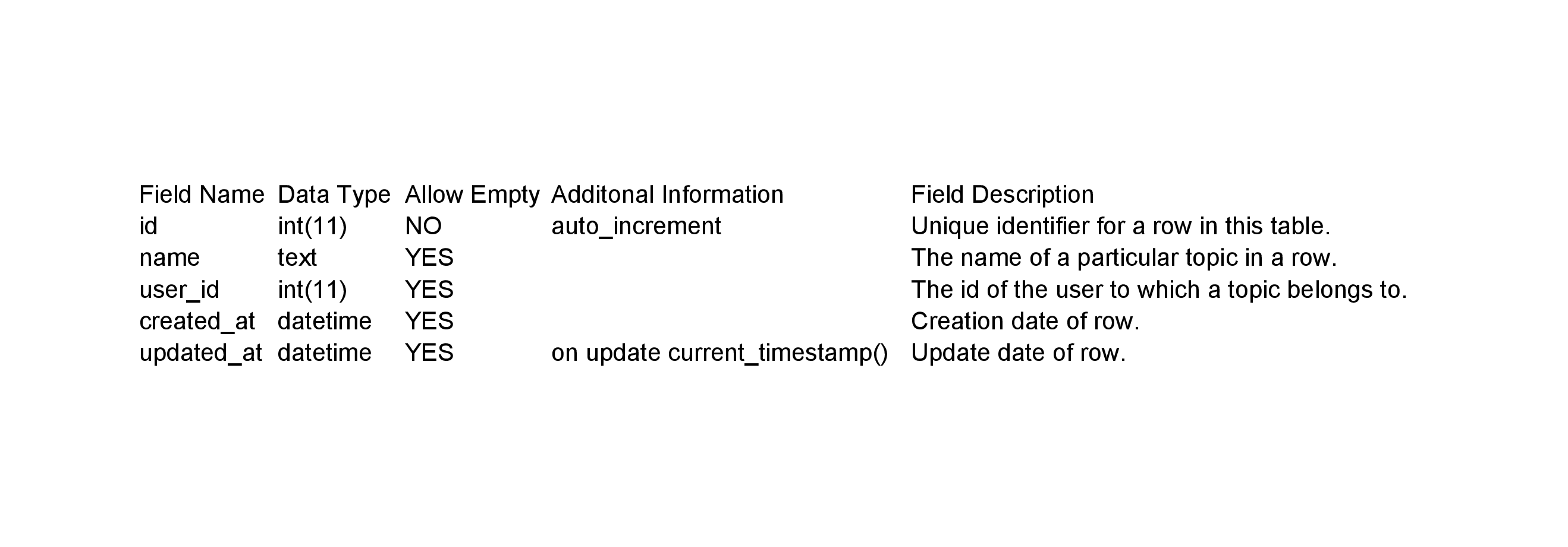
Claims



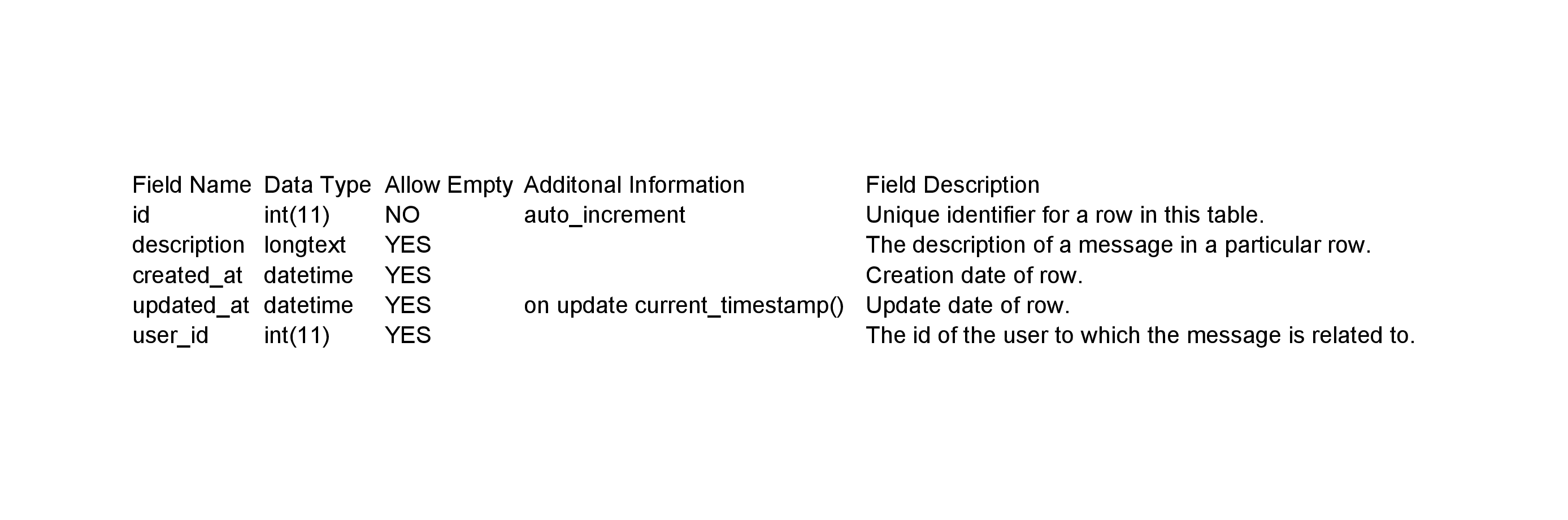
Users



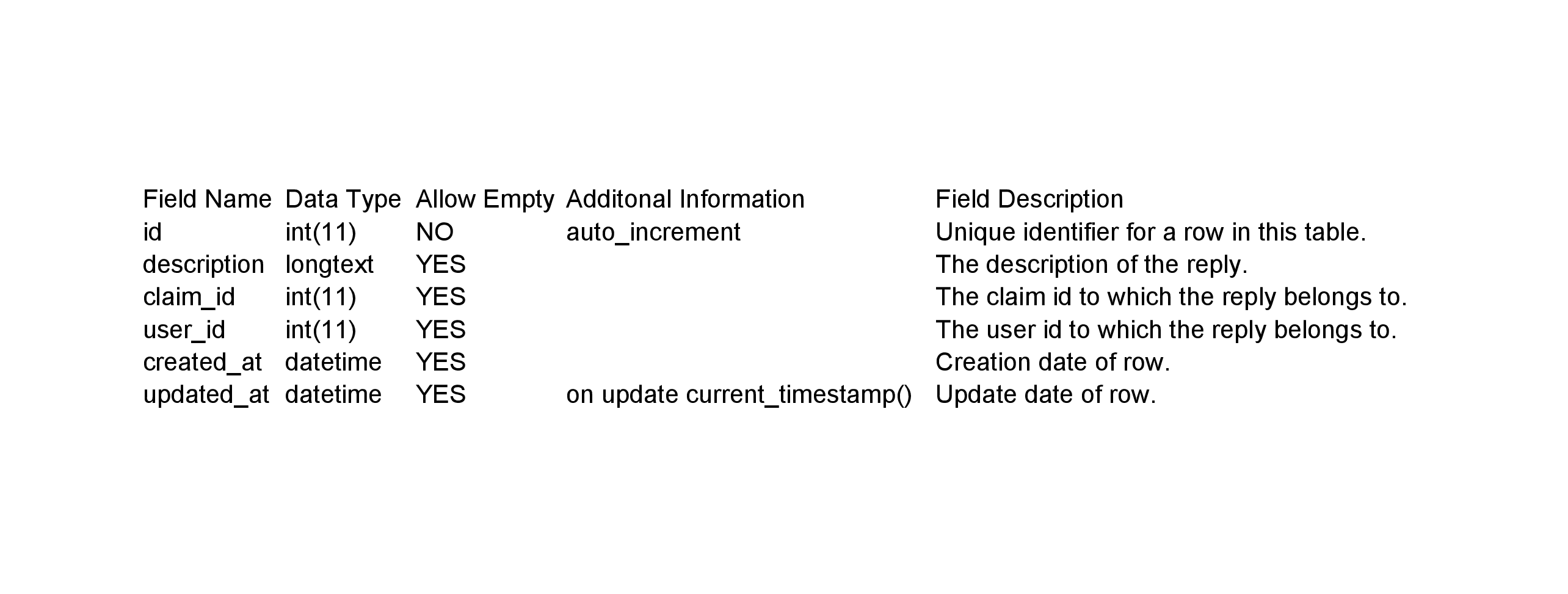
Topics



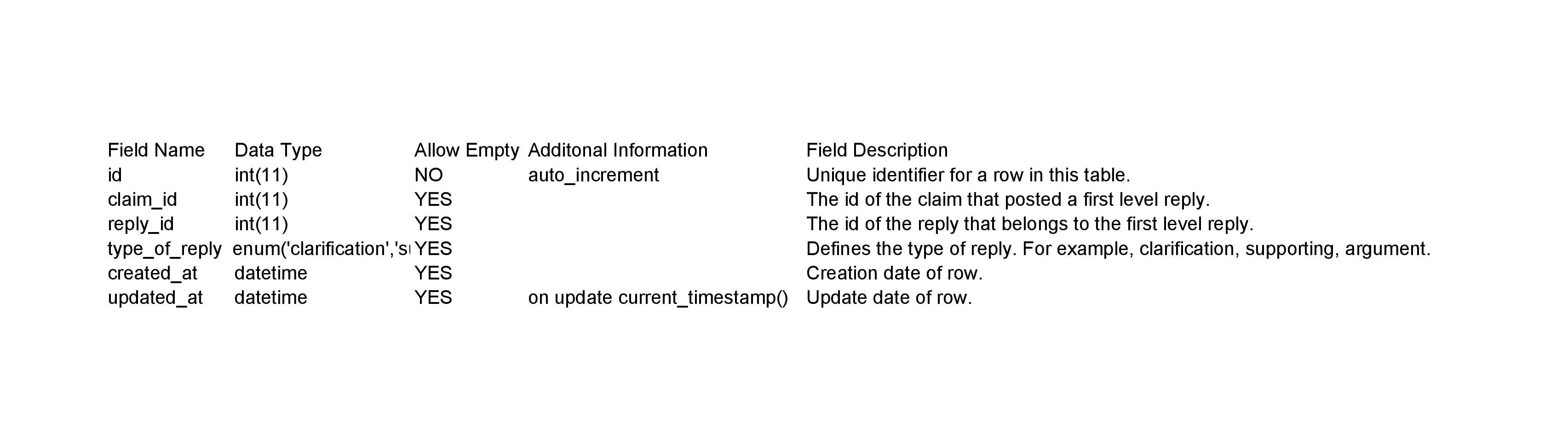
Messages



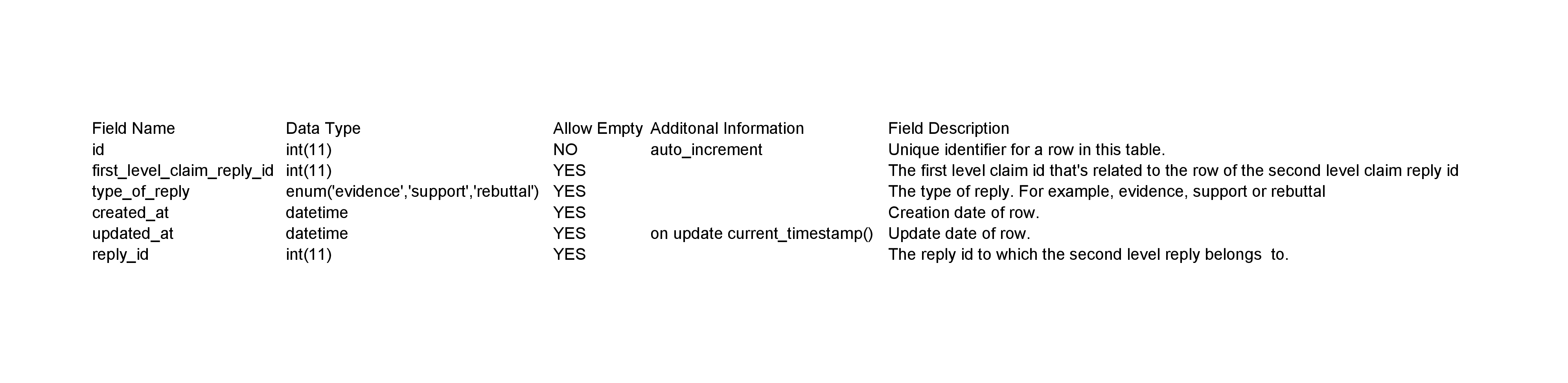
Replies



First level replies



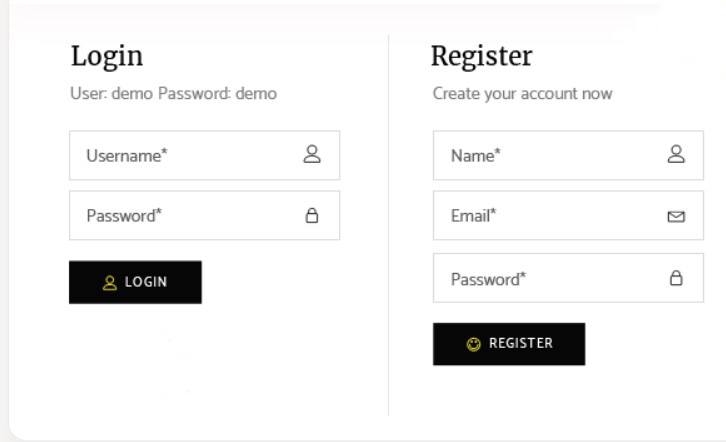
Second level replies



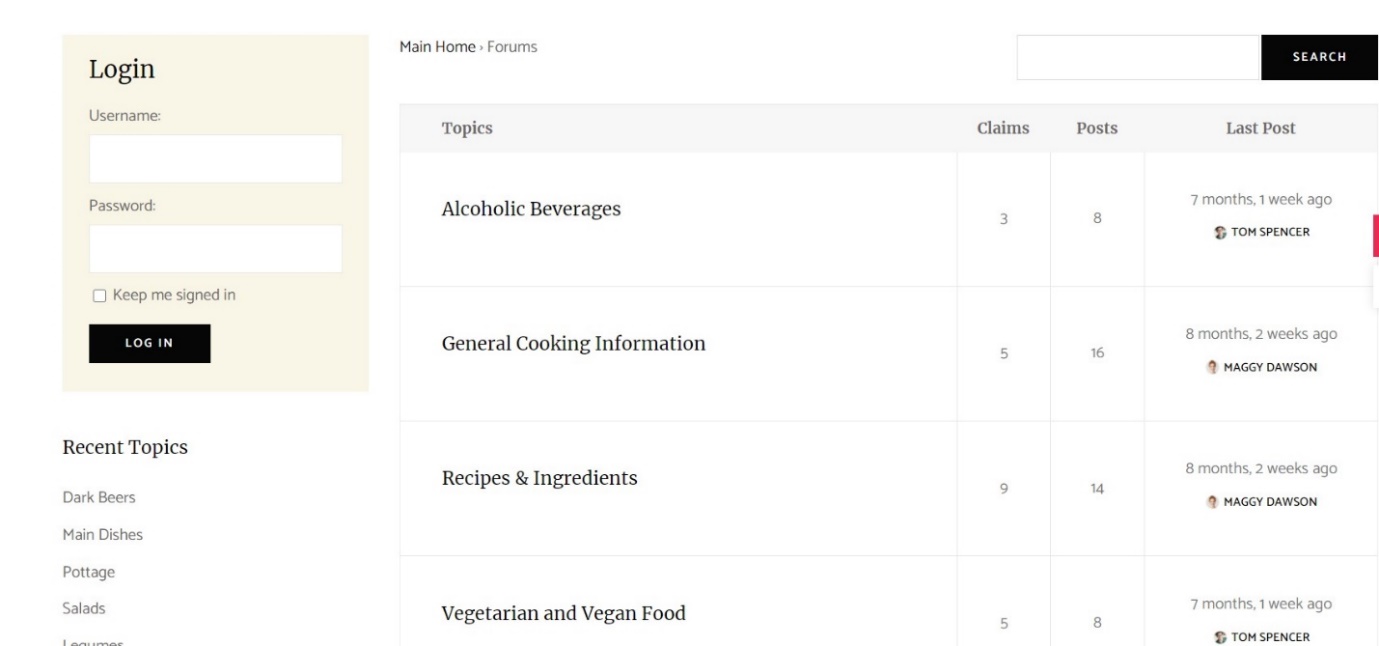
# High-Level Design (Wireframes)

The design for the user interface will be as follows:

## Login and Registration View

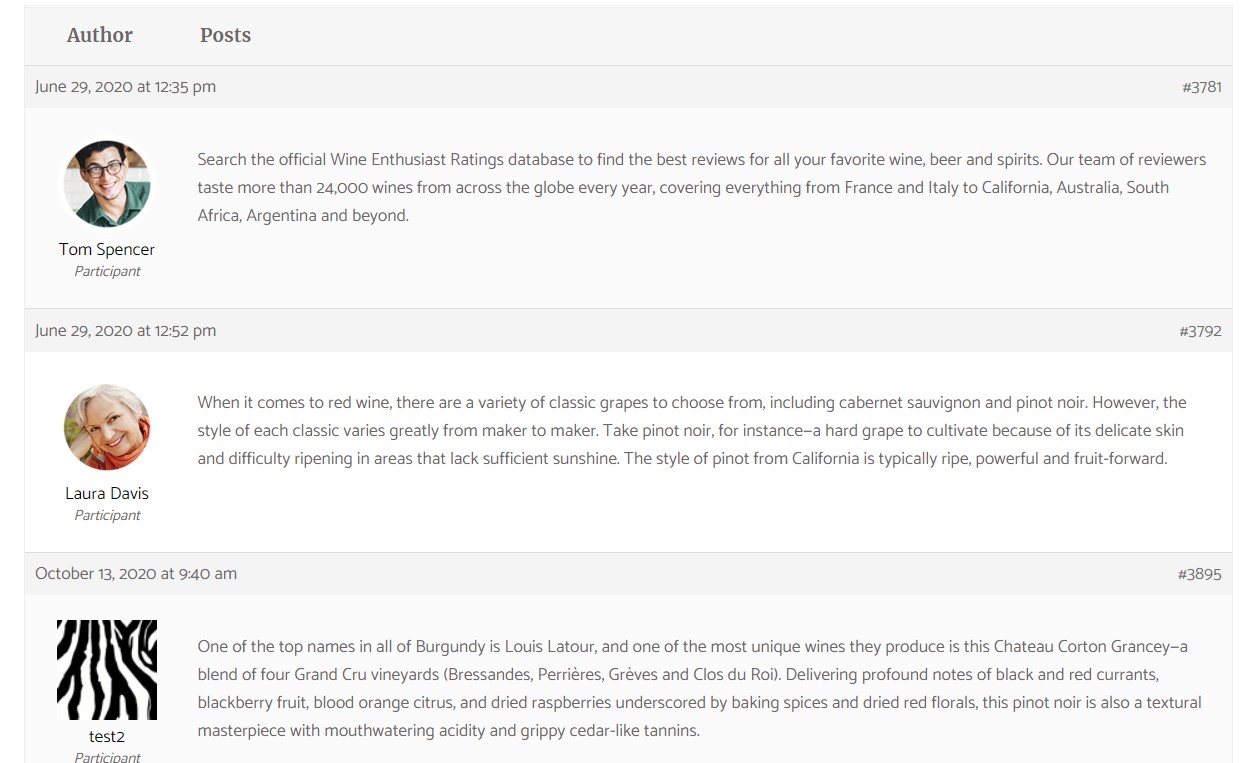
****

## All Topics View

****

## **All Claims views**

## Single Claim View

****

