**Final Report: Game of Thrones Blog Forum**

**CPSC 408 Database Management**

**Spring 2019**

**Background**

With the growing popularity of HBO’s hit show Game of Thrones, there has been more demand for a common platform to host discussions about latest events and analysis of weekly episodes. Many blog forums allow users to create accounts to publicly post content to forums that could lead to spoilers and explicit content. While other free discussions such as personal messaging threads do not allow users to manage discussions by updating or editing their own content. Adding these features to forums will give more control to the user, as well as making it easier to add others to the conversation to have more meaningful discussions. The goal of this project is to create a web application that will allow users to join blog forums with their friends by registering accounts and managing discussions through the implementation of CRUD features and a user-friendly interface.

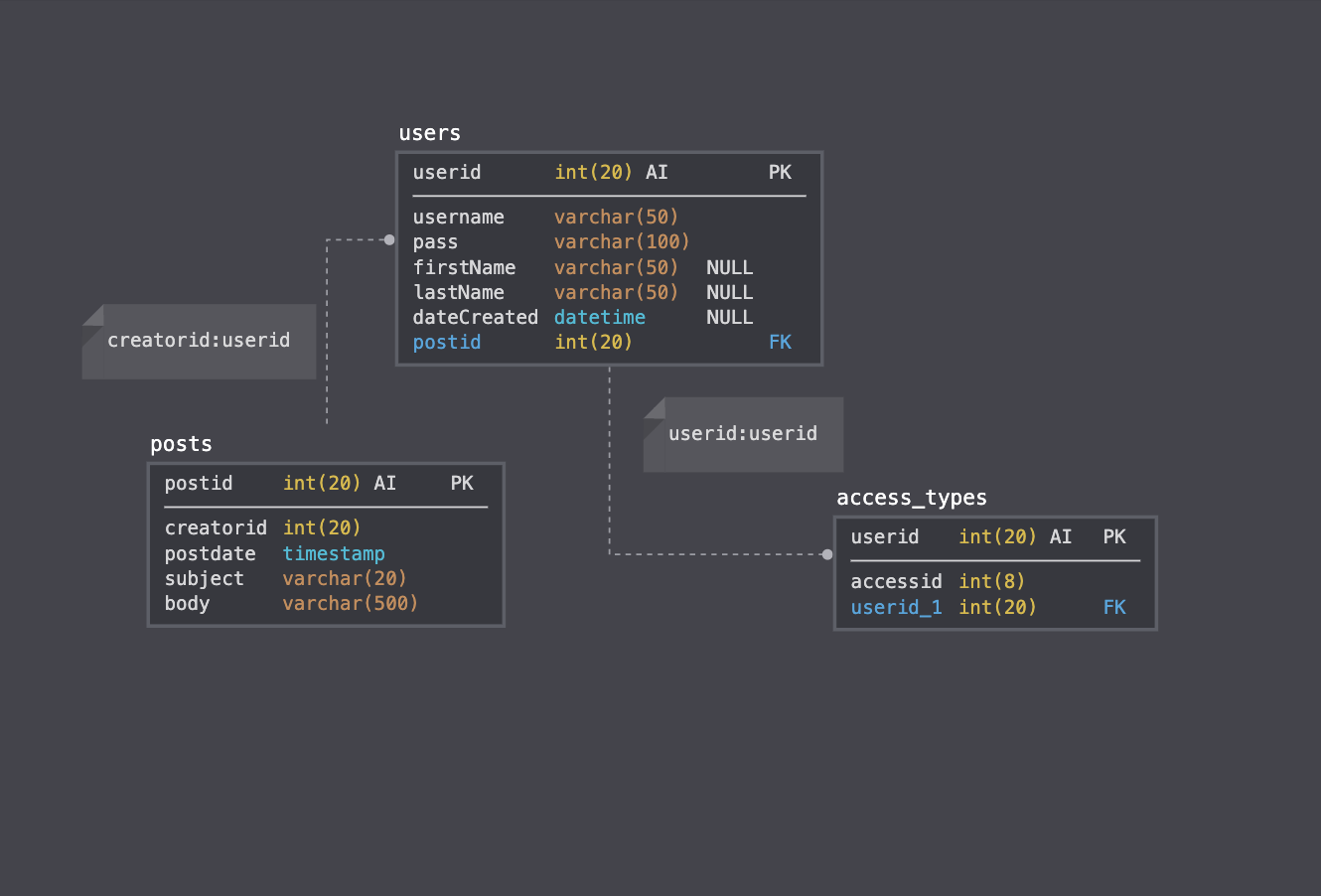
**Solution**

The Python Flask web framework was used for the backend of this project. Although Flask is a relatively young framework, it is a very effective and minimalistic, and powerful for smaller applications. If Python 3 is already installed, the rest of the set up is easy to get up and running quickly. Flask strips many software plug-ins and libraries from the more popular Python web framework Django, making the learning curve much easier for developers new to web development.

Flask serves as a useful tool for the purpose of this application that is meant to be straightforward and user friendly. Flask provides documentation with a clear project layout that makes it easy to organize the required functionalities of the blog forum. HTML and CSS were used in the front end where all HTML views are organized into templates that extend a base super template to create a consistent HTML structure for the entire user interface. The backend required a mysql-connector-python driver to establish a connection to a local MySQL server. To ensure that all sensitive data loaded into the database was not stored in raw form, hashed passwords were generated. A python CGI library called Werkzeug was used as a security helper to generate hashed passwords with a salt, and to verify correct passwords by checking it against a given salted and hashed password value. The python CSV module was also used to generate reports into csv format preferred by Excel. Flask provides a useful built-in development server that provides a debugging tool and tracks all requests made to the application.

**Schema Diagram**

The database schema consists of three tables containing user-specific data, blog post data, and data of user access privileges. The userid attribute is assigned as the primary key of the Users table which stores all user account data. The Posts table stores updated blog post content and timestamps. The attribute creatorid from the Posts table references the primary key userid from the Users table to manage post ownership of each unique user. To ensure that posts are linked and can only be managed by users who create them, a composite key, or pair of unique primary keys are created with creatorid and postid. The third table stores access types to each foreign key userid by assigning each an access id. A temporary view was created as a requirement for this project, and to also consolidate user and blog post data onto a single table to generate meaningful reports to a csv file.

****

**Functionality**

When navigating to the localhost address, the index page displays all blog posts made by users with registered accounts. A new user will not be able to create a new post until they create a username and password. Clicking on the register icon on the top right of the window will prompt the user to enter a new username, password, first name and last name. Once the account is created, the user will be redirected to the login window, where they enter their new username and password. After doing so, the same index page will appear, but will now give the user options on the top right of the window to create a new post. Clicking on the New Post icon prompts the user to enter a title and body for the new blog post. To post the blog to the forum for other users to view, the user can click Save, or to remove it, the user can simple click Delete. Either action will take the user back to the index page and list the updated posts. The Edit icon will only appear to the right of any posts that belong to them, which will allow them to edit or delete them. Users may also search for blog posts by entering any string at the top right of the window where the Search bar is located. This will filter out any blog posts containing the input string within the blog post title or body.

**Future Works**

Although the web application includes all the necessary features to create, update, edit and delete blog post content, there ways to make the application more customizable and fun to use. Using other Flask modules such as Flask\_Avatars provide access to Avatar URLs for users to personalize their user profiles. Adding more personalization features will allow users to be more creative and provide a fun environment for discussions in the future. While lightweight and easy to use, Flask’s built-in server is not suitable for production as it does not scale well, and only serves one request at a time by default. As this application becomes more polished, it would be interesting to deploy it on a production server to make it public and accessible for friends to use as well in the future.