Carter Williams

9/25/22

Dr. Zhonghang

**CS 396 Phase 1**

**Project Description**

Phase 1 of the CS 396 term project includes building a website that provides services of forum, documentation, searching, linking, and others. This project was implemented with the Python web framework Django along with SQLite as a database. Users can sign up for the forum with a username and password and then can post under discussion topics. They can create a title, provide textual content, and upload files to these posts. Other users can then comment under these posts. Only admins of the website can create discussion topics through the ./admin/ web interface. Admins can also adjust/delete posts, topics, users, and anything else stored in the database. Admins can be created by another admin through the ./admin web interface or by passing the “createsuperuser” argument to manage.py.

The front end of this project was implemented using Django templates. Templates allow for webpages to be created using HTML, CSS, and JavaScript, but it also adds some functionality with Python code inside HTML files. This will be explained further in the Technical Details section.

Anybody can use the codebase for their own purpose by following the ReadMe inside the project’s files.

**Functions Implemented**

Many functions have been implemented in order to make the forum website perform various actions. Most of these functions are inside the ./fintech/views.py file, but some of Django’s models also have overridden methods. These functions will be split into sections based on which file they are implement

./fintech/views.py

1. def login\_(request)

The function login\_ handles the url ./login/ and this is where users get authentication. The template login.html provides the UI for users to interact with this function. If the user is already authenticated, the user is redirected to the home page. If the user is not authenticated, the inputted username and password are retrieved through request.POST.get, and Django’s authenticate method (django.contrib.auth) authenticates the user. If it is a valid login, the user is logged in and redirected to the homepage, but if they are not, a message is displayed that the username and password are not found.

1. def register(request)

The function register handles the url ./register/ and this is where users go to add a new user in the database. The template register.html provides the UI for the users to interact with this function. If the user is already authenticated, the user is redirected to the home page. If they are not authenticated, the fields username, password, and password confirm are retrieved through request.POST.get. The password and password confirm are checked for equality, and if they match, a new user is created through the User.objects.create\_user function (django.contrib.auth.models). The user is then redirected to the login page in order to login to their new user.

1. def logout\_user(request)

The function logout\_user handles the url ./logout/ and simply logs the user out of their account. This is accomplished through the logout function (django.contrib.auth). The user is then redirected to the login page.

1. def index(request)

The function index handles the base url and displays some initial information to the user. Boards, topics, someposts, and the total number of posts are gathered from the database. All of this is then passed to index.html through the context variable. Index then displays the board name, the total number of posts on the website, the topics, and two posts from each topic. Users can navigate to new topics or posts by clicking their titles.

1. def view\_topic(request, topic\_id)

View\_topic is the function that handles the url ./topic/<int:topic\_id>/, and it displays all relevant information to the topic. Topic\_id is the primary key of the topic from the database. The topic and posts under the topic are retrieved from the database and passed to topic.html to be displayed to the user. The posts can then be clicked on by the user in order to view the individual post. Topic.html is the template used for this view function.

1. def view\_post(request, topic\_id, post\_id)

View\_post is the function that handles the url ./topic/<int:topic\_id>/<int:post\_id>/. Topic\_id and post\_id are the primary keys of the related topic and post. View\_post retrieves the topic and post based on the primary keys from the url, and the comments are retrieved from the database based on the foreign key of post. Files that the user uploaded with the post are also retrieved from the foreign key of post. Post.views is also incremented since someone viewed the topic. Post.html is the template used for this view function.

1. def create\_post(request, topic\_id)

Create\_post is the function that handles the url ./topic/<int:topic\_id>/createpost/, and the user must be logged in to interact with this function. The topic and old notification are retrieved from the database. The post’s title, text content, user, and files are retrieved from request.POST.get. A new post object is created and added to the database, and the notification record in the database is updated to display this post. Files are then saved to the OS’s file system, and the path is saved to the database. The function handle\_uploaded\_file handles saving the file to the correct directory. Once all these objects are saved to the database, the user is redirected to the new post. Create\_post.html is the template that allows the user to interact with this function.

1. def edit\_post(request, topic\_id, post\_id)

Edit\_post is the function that handles the url ./topic/<int:topic\_id>/<int:post\_id>/editpost/, and the user must be logged in to interact with this function. The topic, post, post’s user, and files are retrieved form the database, so the user can view the fields to edit. The user is then checked to ensure it was the post’s user or a superuser, and if they are not, they are redirected back to the post. The form’s fields are then retrieved from request.POST.get and changes are uploaded to the database. New files are handled the same way as create\_post, and files checked with the radio button are deleted from the database and file system. Edit\_post.html is the template that allows the user to interact with this function.

1. def delete\_post(request, topic\_id, post\_id)

Delete\_post is the function that handles the url ./topic/<int:topic\_id>/<int:post\_id>/editpost/deletepost/, and the user must be logged in to interact with this function. The topic, post, and post’s user are retrieved from the database, and then the user is checked to be the creator or a superuser. If the user is authenticated, the post is deleted from the database. There is no template for this function since it does not need to display any information.

1. def post\_comment(request, topic\_id, post\_id)

Post\_comment is the function that handles the url ./topic/<int:topic\_id>/<int:post\_id>/postcomment/, and the user must be logged in to post a comment. The comment’s information is retrieved from request.POST.get and it is saved to the database. The template post.html has a form that allows the user to interact with this function.

1. def display\_images(request, topic\_id, post\_id, image\_name)

This function handles the url ./topic/<int:topic\_id>/<int:post\_id>/media/<str:imange\_name>/. The file is retrieved from the file system using the topic\_id, post\_id, userid, and image name. This file is then returned to the user with the FileResponse() function. If it is not found, a 404 error is thrown.

1. def authenticate\_for\_update(request, user)

This is a helper function that returns true if the user is not authenticated to update or delete a post. If the user is authenticated, it returns false. It simply checks if the request.user and post’s user are the same, or if the request.user is a super user.

1. def return\_post\_files(user, post)

This is a helper function that retrieves all of the files associated with a post. It filters all the records in the DocumentFile table with the foreign key of the associated user and post. The files (if found) are then returned.

1. def handle\_uploaded\_file(f, topic\_id, post\_id, user\_id)

This is a helper function that saves a file to the OS’s file system. The location is picked based on the topic\_id, post\_id, and user\_id. The file (f) is then saved to the file system in the correct location.

1. def get\_notifications()

This is a helper function that retrieves the latest notification from the database. This row is updated to the latest post posted to the website. The text from Notification is then returned.

./fintech/models.py

1. def delete\_post(post)

This is a helper function that is called when a post is deleted. It finds all files associated with the post and deletes then from the database and file system.

1. def delete(self, \*args, \*\*kwargs) Post

Overridden method for Post that calls delete\_post(post) in order to delete the files. Calls the models.Model delete function.

1. def delete(self, \*args, \*\*kwargs) DocumentFile

Overridden method for DocumentFile that deletes the directory containing the post’s files. It then calls the models.Model delete function.

**Technical Details**

This section will discuss any technical details associated with the database, software, and language.

Database

Django provides almost all functionality related to interacting with the database. For this project, no SQL had to be written because Django has libraries that handle that for you. Models had to be defined that became associated with tables in the SQLite database. This section will discuss the database tables and columns. Each of the listed class has an associated table in the SQLite database, and they inherit the models.Model class. Most of Model’s methods are kept, and the ones that were redefined were explained in the previous section.

1. Board

The table that represents the forum board. It has a name.

1. DiscussionTopic

The table that represents a topic that users can post under. There is a title for the topic and a date for when it was created. There are also two foreign keys: one for the board it is under and for the user that started it.

1. Post

The table that represents a user’s post under a topic. Posts have a title, textual content, a creation date, and the number of views. There are also two foreign keys: one for the topic the post is under and one for the user who created the post.

1. Comment

The table that represents a comment under a post. Comments have a creation date and textual content. There are two foreign keys: one for the post it is under and one for the user who created the comment.

1. Notification

The table that represents the newest post that was made. It has one row that holds the text that describes what the latest post is. The single row is updated every time a new post is created.

1. DocumentFile

The table that represents the file associated with a post. It has the relative path to the file, and the name of the file. It also has two foreign keys: one that links it to the user who uploaded the file and one that links it to the post it is under.

Software

This section will outline the technical details of the software, in general. Overall, the software follows the model-view-controller and server-client pattern.

Models represent records database tables, and can be used to store their information. Columns in the tables are represented through class attributes, and database actions are represented through methods in the model. These are held within ./fintech/models.py. Models are updated through the controller, which are held in ./fintech/views.py. These functions serve as the middleman between the models and view by updating the model(s) and passing information to the view. Views describe how to display information to the user, and these are held within the ./fintech/templates/fintech directory. In this project’s case, views are HTML files that display information and pass information back to the controller.

Django defines how the server serves information to the client. In this case, the SQLite database and Django Python files are the server. HTML files are then sent to the client to be rendered by the browser. This keeps sensitive data away from the clients machine, and puts most of the load on the server.

**Language/Scripts**

This project was implemented with Python, HTML, CSS, and JavaScript. Python comprises the bulk of the program because Django is used for most of the database and back end logic. HTML and CSS are used for displaying and styling the view served by the backend. JavaScript is used for displaying the notification on the page.

All of the back end’s logic is implemented with Python through the Django framework. Interacting with the database, serving HTML files to the user, and writing business logic are all done with Django and Python. Django provides a manage.py file that allows for arguments to be passed to perform actions. Running the server, migrating model changes to the database, creating a superuser, etc. are done by passing arguments to this file.

Information viewed by the user will be served using HTML, CSS, and JavaScript. HTML makes up the bulk of the front end code because it lays out the information. CSS styles HTML elements by changing their color, size, position, etc. There is one JavaScript function implemented in every HTML file that hides or displays a div. When the user presses the notification button, the JavaScript function is called to either hide or display the notification.

**Features**

1. Create an administrator who can delete other users, delete/edit all posts, create new discussion topics.

There are two ways this can be done: through manage.py or through the ./admin/ web interface. The argument “createsuperuser” can be passed to manage.py to create a new admin, or an existing admin can create a new user and give them admin privileges through the ./admin/ interface. Once the admin has been created, new discussion topics can be made through the ./admin/ interface and all other models can be edited/deleted here. Admins can also edit/delete posts through ./topic/topic\_id/post\_id/editpost/ interface.

1. Provide login function, and sign-up function for new users.

Users can sign-up through the interface at ./register/ by entering a username and password into a form. They can then login at ./login/ by entering their username and password. You must be logged in to make posts and comment, so this is necessary unless you are just viewing the site. These URLs are accessed by clicking the login button in the navigation bar and the signup text under the login form.

1. Any user can create a new post and reply to other users’ posts.

Users can create a post through the interface at ./topic/topic\_id/post\_id/createpost/ and this can be accessed via a button on the navbar while viewing a topic or post. Other users can then comment on that post while viewing it by typing the comment in a text are and submitting through a button.

1. Any user can upload files (could be multiple files).

While creating a post, users can select multiple files by clicking a button and navigating through their computer’s file system. Multiple files can be selected by CTRL clicking on multiple files.

1. Show the numbers of posts and visits, and keep these numbers updated.

The total number of posts on the website are displayed on the index page. This number is retrieved after each GET request. The total number of posts in a given topic is also displayed and updated. Posts also have a view counter, and this number is stored as an integer in the Post table. After each GET request of the designated post, this number is incremented.

1. You can store some files on your website.

Files can be stored after being uploaded by users. The actual file is stored in the ./media folder with other subdirectories related to the topic, post, and user. The database then stores the path and name of the file. These files can then be deleted and viewed by users.

1. Include a link to google calendar in a drop list.

Each individual post has a drop list at the bottom of the post. When this is clicked, a link to Google Calendar appears. When this link is clicked, a new tab opens at https://calendar.google.com/calendar/.

1. Show a warning a message/sign to remind users that there is an update or something interesting posted in the forum.

The navigation bar has a button that displays the newest notification. This button is a bell image, and it triggers a JavaScript function that displays a div. This div contains information on where the newest post is.

1. Your website interface should be friendly.

There is a navigation bar that helps users easily navigate around the website. Also, topics and posts can be navigated to just by clicking their title. Forms are very concise and clear on what information they are asking for. These factors create an interface that is easily navigable by the user.

**Discussion**

For the purpose of this project, Django and Python are a great option for database interactions as well as back end logic. Python provides an easy syntax with a dynamically typed variables, but this could hurt the project in the long run.

Dynamically typed languages provide a much easier programming experience because no variables must have their types declared. There are two problems with this: types are not checked until runtime and processes usually take more time to complete. Statically typed languages like C# and Java would allow for type checking at compile time as well as generally faster processing time. Bugs are usually easier to spot because types are explicitly declared, but dynamically typed languages, like Python, cause bugs to be harder to spot. If there was more time on this project, I would likely use a framework like Spring boot (Java) or ASP.NET (C#). These languages would also provide portability similar to Python since they are compiled to an intermediate language.

In terms of the database SQLite is the best option for this project, currently. SQLite has a simple access algorithm because it is an embedded into the application. SQLite’s problem begins to show itself when many users are trying to access the database. SQLite struggles with throughput, so if the application’s userbase grew significantly, the database would likely need to be migrated to MySQL or SQL Server.

Finally, the front end could be implemented with a JavaScript framework, like React of Angular JS, instead of Django’s template. These frameworks allow for more interactive and dynamic UIs to be implemented with more functionality. Pages would likely take longer to load due to most of the UI being interpreted from JavaScript code, but it also has benefits: UI components could be reused easily throughout the website. If this website was a commercial product, its UI would need to be more robust, which would require a better front end framework.