ECE 49595 (364) Project Description Document
Phase 1 and Phase 2

Completing this project will satisfy course objectives CO4, CO6, CO2, CO3

Instructions

- You must meet all *base requirements* in the syllabus to receive any credit.
- Work in your Lab12 directory.
- In case you wish to use the helper code for interfacing Sqlite and Flask, use the below command. You can skip this step if you are going to write your own code for this purpose.

  ```
  cp -r ~ee364/DataFolder/Lab12/* ./
  ```

- Remember to add and commit all **required** files to SVN. **We will grade the version of the file that is in SVN!**
- Do *not* add any file that is *not* required. **You will lose points if your repository contains more files than required!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- Name and spell the file, and the functions, exactly as instructed. Your scripts will be graded by an automated or manual process. **You will lose some points, per our discretion, for any function that does not match the expected name.**
- Unless otherwise specified, you cannot use any external Python library, but you can use any module in the **Python Standard Library** to solve this lab, i.e. anything under:

  ```
  https://docs.python.org/3.8/library/index.html
  ```

- Make sure you are using Python 3.8 for your lab.
- For this project, you can use the following guides to implement the requirements, i.e. anything under:

  ```
  http://alexquinn.org/sqlite3_helper/
  https://www.sqlite.org/docs.html
  https://flask.palletsprojects.com/en/1.1.x/
  https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
  https://www.w3schools.com/html/
  https://cssguidelin.es/
  https://developer.mozilla.org/en-US/docs/Web/JavaScript
  ```

Project Description Document

Required Folder Structure

```
Lab12/templates - Store all your HTML files in this folder
Lab12/styles - Store all your CSS files in this folder
Lab12/scripts - Store all your Javascript files in this folder
Lab12/data - Store all your Database files in this folder
```

In order to create and update these folders on svn, you can use these commands

```
$mkdir templates
$mkdir styles
$mkdir scripts
$mkdir data

$svn add templates
$svn add styles
$svn add scripts
$svn add data
```

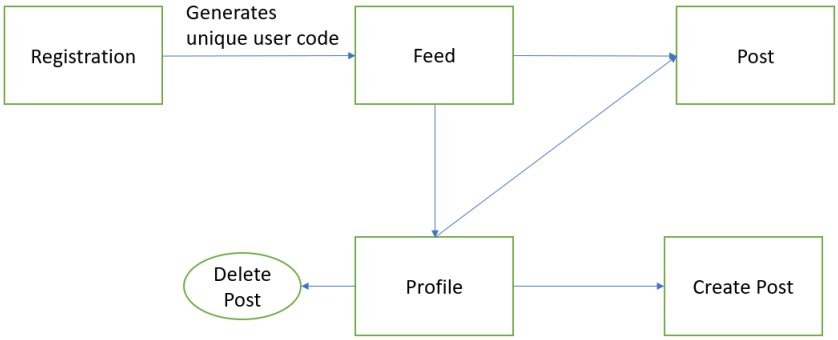
All your Python files, README can be stored inside `Lab12` folder.
You do not need to create a folder to store the `.py` files.

Note

- For the Project, you are only allowed to use the stack we have used till now. Flask and Sqlite for the backend. However, frameworks like Bootstrap, Jquery can be used for the frontend.
- You are responsible for assigning tasks between yourself and your project partner (if you're working as a team). Make sure you communicate clearly among yourselves what tasks each of you are responsible for.
- We will be providing Sqlite helper code (Courtesy Prof. Alex Quinn, Purdue ECE) to help you code server interaction with database. The documentation for the same is mentioned in the instructions. You may use this code for the project if you wish to do so. However, if you wish to code the database interface yourself, feel free to do so.
- The project will be split into 2 sprints, each sprint running for a period of 2 weeks.
- As in a real world agile project, we might have slight changes in requirements at any point of the project. We will ensure you are informed of this in advance if any.
- We estimate an effort of 4-5 hours per student per week to complete the entire project.
- We will have demos delivered online at the end of every sprint. This will serve as the phase 1 and 2 demos respectively.
- Phase 2 demo will be the final demo and will include code submission.
- All submissions will be checked for plagiarism.

Introduction

For the project, you need to implement a basic version of `Reddit`. Like the real `Reddit`, you need to implement the work flow as defined by this diagram :



Note: This is not a complete representation of the system, it is just to give you a big picture idea of the main system features and how they interact.

The implementation details for the Frontend and Backend are given below:

Frontend Implementation Requirements

The workflow should start with a `Registration` page in which users can register/signup for the product. Keep in mind the following:

- Create an API (Eg: /register) for this registration page.
- Once a user registers, you need to write the user's information in the database.
- Use a `hashing` function to generate a unique code for each user. You can use any hash you deem fit for this purpose.
- Any API in the product must only expose this `hashed` code. You will need to map the user ids in the database to the unique hashed code. This is to ensure no user can access another user's API by changing the id in the request.
- If an already registered user registers again, link him/her with the old data,
- Your registration page need not have a `Login` option. You do not need to think about ACLs (access control lists) for the project.

Once the user registers, he/she should be directed to the `Feed` page where all the posts created by all the users except his/her own posts are seen. Keep in mind the following:

- Create an API (Eg: /{hashedcode}/feed) for the feed.
- In the feed, the posts must be shown in reverse chronological order i.e, most recently created posts on the top of the page.
- Use `pagination` to ensure not more than 5 posts are showed per page in the feed.
- At the top right of the feed, show the user's name. Clicking on the user's name should take him/her to his/her profile page.

- Include a logo of your product in the top left corner. At any point of time, clicking on the logo should redirect to the `Feed` page.
- Clicking on any of the post's titles in the feed should redirect to the post's page - /posts/{postid} (Defined later in the document).
- Each post should have an upvote and downvote button. Also, display the number of votes on the post next to it.
- Clicking the upvote or the downvote button should dynamically change the vote count shown and update the database without reloading the page.
- A user must be able to upvote or downvote only once for each post.
- Do not display the posts created by the user in his/her own feed.

On clicking the user's name in the feed, the user must be redirected to his/her `Profile` page. Keep in mind the following:

- Create an API (Eg: {hashedcode}/profile) for the profile page.
- The profile page should show all the posts created by the user in reverse chronological order.
- A common terminology used in systems design is CRUD (Create, Read, Update and Delete). From the profile page, a user should be able to perform all the CRD operations. Update need not be implemented for the project.
- At the top right, add a `create` button that allows a user to create a new post. On clicking the button, redirect to create API (Eg: {hashedcode}/create).
- In the create API, capture the following and write the information in the database.
    1. `Post title` - Entered by the user when creating the post.
    2. `Post content` - Entered by the user when creating the post.
    3. `Post timestamp` - Date and Time (Eg: Oct 10, 11:59pm) using timestamp generated by the server at time of creation (doesn't have to be captured from the user).
    4. `Post vote counter` - Initialized to zero (doesn't have to be captured from the user).
    5. Upon creating a post, generate a unique post id. The post should be accessible from an API (Eg: /posts/{postid}).
    6. The API should display the post's information along with the username of the user who created the post.
    7. You do not need to implement any sort of comments/replies for a post just the voting system.
- Once the post is created, redirect to the user's profile page.
- The posts displayed in the profile page must must have a `delete` button next to them.
- Clicking the delete button should delete the post upon additional confirmation from the user. Upon deletion, the post should vanish from the user's profile page without having to reload the page.
- The deleted post should no longer be shown on the profile page or on the feed of other users.

Additional Notes:

- For most of the Frontend webpage components, using a framework like Bootstrap will help you save a lot of time.

- You can take inspiration from the real `Reddit` website for choosing colors, fonts etc.
- One suggestion would be to use Cards and Pagination directly from bootstrap instead of coding everything from scratch.
- Similarly, a lot more components required for the frontend need not be coded from scratch. Go through the framework documentation and think of how to implement the components using the framework.
- Using Javascript/Jquery will also help you define actions for the UI events like button clicks etc.
- Make sure to use AJAX (Asynchronous Javascript and XML) when required to send asynchronous data.
- Make sure to test all your button actions, links thoroughly.

Backend Implementation Requirements

Use `Flask` to design your web server with the following specifics:

- You are free to design as many APIs as you think your product might require. Make sure to at minimum create the APIs mentioned in this document.
- You are free to name your APIs as you like but keep in mind aspects of security. Eg: A user should not be able to access another user's profile just by changing the url. (The hashing function will help you here)
- You can use arguments or dynamic routes in case you need any sort of infomation in the url.
- Make sure to specify the correct methods for each API. Internal APIs should not have their GET methods exposed.
- Use Jinja template render HTMLs with dynamic data.
- Do not directly return any data for any API. Always render HTML files.
- You can use the Sqlite helper files to interact with the database from the web server. You can code it yourself too if you wish to do so.

Use `Sqlite` to design your database with the following specifics:

- Depending on how you implement it, you will have to decide what columns to include and how many tables your product will require.
- You will have to design the schemas for all your tables that you use.
- Put all your table creation commands in a schema and use the schema to initialize your database. We will be using the schema to intialize a new database during the demo if required, so ensure your schema files are up to date. We will use the below command to initialize your database:

  ```
  sqlite3 database.db < schema.sql
  ```

- Be sure to identify Primary keys for all your tables.
- Make sure to always store timestamps in UTC format. Your front end code should handle time conversion from UTC to browser local time.
- You will have to also identify foreign keys for your implementation.

- Ensure the PRAGMA foreign_keys is set to ON to ensure foreign key constraints are checked every time data is inserted in the database.
- Add an API (Eg: /getTSVdump) that renders a HTML with a 'Download TSV' button. Upon clicking the button, the page should download a TSV file with all the information contained in the database in the same format as the database. You can have multiple 'Download TSV' buttons for multiple tables in your database. Make sure to name them correctly.

Phase 1 Demo Requirements

- Phase 1 demo is planned for the end of sprint 1 - **13th/14th of April**.
- For phase 1, you are not required to submit code.
- Inline with Agile way of work, we will not deduct points for any bugs found in phase 1. However, you will have to fix these bugs in your phase 2 demo.
- The demo should showcase the following implementations:
    1. Registration Page and the API should be complete. Registering on the webpage should be reflected in the database.
    2. User Profile Page and API should be complete. You should demo creating a post. The created post information should reflect in the database.
    3. You should show us the database schemas that you plan to use for the project. You are free to change this later on, this is just to ensure you are headed in the right direction.
    4. Based on the demo, we will notify you of bugs if we find any. You will have to fix them for the next demo.
    5. Hashing function must be implemented and make sure you ensure your APIs cannot be modified to access other user's information.
    6. Please ensure you test your frontend behavior thoroughly.
    7. The exact testing rubric will be posted one week prior to the demo date.
    8. Feed, implementation of Delete post and implementation of the Download TSV API are not required for Phase 1 demo.

Phase 2 Demo Requirements

- Phase 2 demo is planned for the end of sprint 2 - **27th/28th of April**.
- **Final code submission for phase 2 should be done through svn before 26th April 11:59pm.** You will have to give us a demo with this version of code during your demo slot.
- This will include all the requirements mentioned in this document. You should be able to showcase the full fledged system.
- Any bugs raised in Phase 1 demo need to be implemented in Phase 2 demo..
- Phase 2 submission should include a README file which explains all the APIs of your product and the database schemas. Also mention your team name in the README file.

Extra Credit

- Implement adding pictures for users during registration. This picture will be shown on their profile page. For simplicity, you can store these pictures in folders in your backend.
- Instead of showing the timestamp on a post, show the duration of time since post creation. Eg: Instead of showing `'22 Oct 11:30 pm'`, show `'8 hours ago'`. Implement this with a granularity of 30 mins.

Final Check!

Before submitting, ensure you have all your Python, HTML, CSS, Javascript and database files in the correct folders as specified in this document. Please test your code on the ECE Grid machine to ensure the behavior is as expected.

You can check if your files are correctly committed to svn by using the command `$svn list`. If this does not work, use the command `$svn up` followed by `$svn list`.

It is upto you to check if your files are correctly uploaded to svn.