IS211 Course Project

For your course project, you will be responsible for developing a small web application, using the knowledge you gained from the course so far. You can choose one of the following options, with more details below on each option:

1. **Book Catalogue** - A web application that helps a user record details about the books they own. The application allows a user to login and type in an ISBN, which will instigate a search using the Google Books API. The server will save some of the information to a database, which will be used to show the user their list of books.
2. **A Blogging Application** - A web application that allows a user to manage their own blog. A blog is simply a collection of "posts" (textual content), typically displayed in reverse chronological order.
3. **Custom** - If you have an idea for your own project, thats great! As long as it meets the criteria listed below, there should be no problem. However, you will need to pass this idea by me, and have my approval before starting.

Useful Reminders

1. Read the project descriptions over a few times. At least twice. It always helps to have a clear picture of the overall assignment when understanding how to build a solution.
2. Think about the problem for a while, and even try writing or drawing a solution using pencil and paper or a whiteboard.
3. Make sure to put in a lot of effort to think about the 'model', or the database schema, before you start working on the code. What is it that you are trying to model? What relationships and properties do you need to model?

Project Option 1: Book Catalogue Details:

The purpose of this web application is to provide the end user with a fuctional interface that will allow the user to keep track of the books they own. The primary screen, after logging in, will show the user the books they have saved so far. Initially, this list will be blank. Users are also presented a form to search for books by ISBN. Once filled out, the application will use the Google Book API to search for the book via ISBN.

To do this, let us take a look at an example. To search for a given ISBN, like 9781449372620 , you can navigate to https://www.googleapis.com/books/v1/volumes?q=isbn:9781449372620 and receive a JSON response that includes all sorts of metadata on the book. For now, we only need to *show* and *store* for the user the title, the author, page count and average rating. If there is any error in processing the JSON, you should return an appropriate error to the user. Sometimes, the API will come back with multiple results. For now, you can use the first result that comes back.

The application should also allow the deletion of books. This will allow the user to delete a book from their list (either because they made a mistake, or they sold the book, etc).

Extra Credit:

1. Extend the application to support multiple users: You do not have to worry about supporting a full registration workflow. Use a table in the database to store users and their credentials, and use this table to support logins from many accounts. **Please note:** storing passwords that are unencrypted is a bad idea, security wise. However, there is no need for us to delve into that.
2. Save links to thumbnails: Use the JSON data to save the URL of a thumbnail, and show this thumbnail when listing out a user's set of books
3. Handling multiple responses from Google: If the Google API gives you many responses for a single ISBN number, show the user the list of results and allow them to choose which one is the one they want to add to their list
4. Allow searching by title: Figure out how to use the Google API to search by title.

Project Option 2: Blog Application

Details

The purpose of this web application is to give a user the ability to run their own blog. A blog is just a series of posts made by an author. Each post has a title, a published date, an author, and textual (HTML) content. When first loading the application at the root URL, the user should be presented with a list of available posts listed in reverse chronological order (newest posts first). The application should allow a user to login via the '/login' URL, which gives that user the ability to make changes to the blog.

After logging in, the user should be sent to the '/dashboard' page, which presents the user with an interface that:

1. Shows a list of their posts in a table. This list just shows the title of the post, with buttons labeled 'Edit' and 'Delete'. This edit button will take the user to a page that allows them to update the post. The delete button will simply delete that post.
2. Allows the user to add a new post (This can be done either directly on the dashboard page or on a new page).

Extra Credit

1. Extend the application to support multiple users: You do not have to worry about supporting a full registration workflow. Use a table in the database to store users and their credentials, and use this table to support logins from many accounts. **Please note:** storing passwords that are unencrypted is a bad idea, security wise. However, there is no need for us to delve into that.
2. Perma-links: Most blogs have a feature called 'permalinks', which is a unique URL for every blog post. Construct URLs for each blog post, and display this perma-link for each blog post.
3. Allow a user to 'un-publish' a post: Lets say the user would like to make changes to a post but would like for the post to not show up on the website. Create another button called 'Unpublish', which keeps the post in the database but stops it from being published on the website.

4. Add a category feature: Posts may have an associated ‘category’ name. Allow the user to set up their own list of categories. Update the ‘add’ and ‘edit’ forms for posts to allow the user to create posts in that category. When displaying the post, make sure to display the category as a link; this link should go to a view that shows only posts in that category

Project Option 3: Custom

Details

You can work on your own idea for your own project, as long as it meets certain criteria:

1. You **must** pass this idea by me, and have my approval over email before starting. Please do this as early as you can, as to maximize the time you have to work on the project.
2. The project **must** be an web based application written in Flask.
3. The project **must** include a database model.
4. The project **should** incorporate as much knowledge from the class as possible. Downloading data from the web, parsing HTML using BeautifulSoup, regular expressions, etc are all things you can incorporate into your project

Requirements

1. You need to make sure to deliver all needed files in a github repository.
2. I should be able to checkout your project repository, run `python app.py` and expect a working version of your project. Opening a web browser to the application main URL should have it bring up your project.
    a. For extra credit, you can host your project on PythonAnywhere (which is covered in Week 15 of the lectures)
3. Please include a `README` file with a few paragraph explaining your solution, how it works and details about your model.