Project: To-Do-List Application with Flask

Design Document

COMP2011 Web Application Development: Coursework 1

Omar Choudhry [sc20osc]
10-27-2021

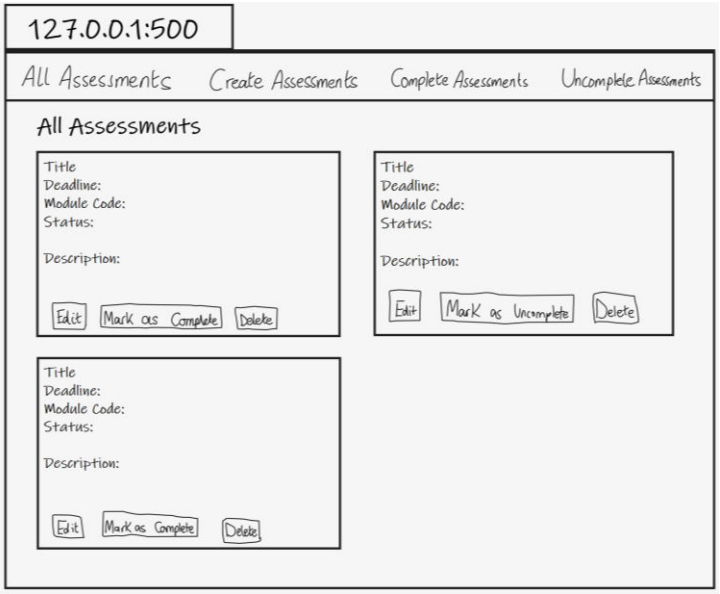
Table of Contents

Development Notes

Wireframe Models

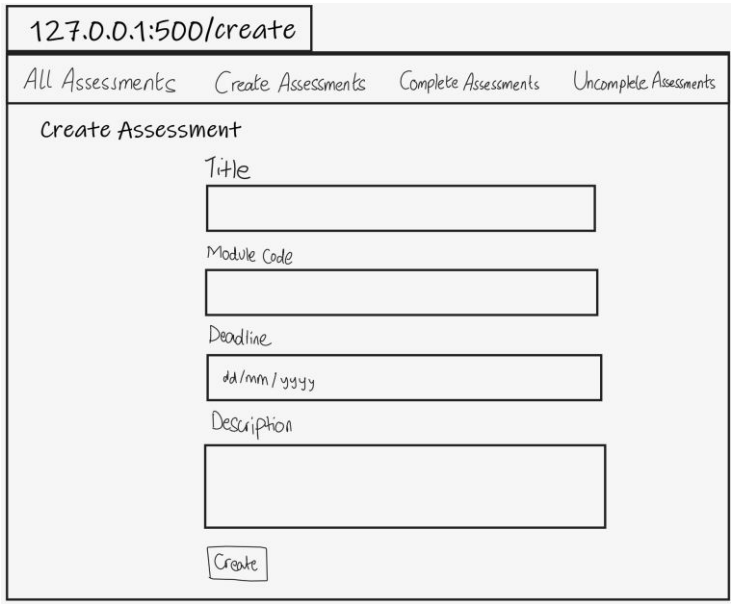
All/Completed/Uncompleted Assessments Page

The wireframe models for all pages displaying assessments are the same. They only differ in the content in which they display (all assessments displays all assessments; complete assessments only displays assessments with status 1 and uncomplete assessments only displays assessments with status 0). Below is a wireframe model of a page with 3 assessments.



Create/Edit Assessment Page

The wireframe models for creating and editing assessments are the same. They only differ in that the submit buttons for the form are different (one is a create button, the other is an edit button) and the form for editing assessments also contains placeholders for the current content. Below is a wireframe model of the create page.



Choice of Layout

I wanted to make a simple application using a consistent colour scheme. To do this I decided to use bootstrap which I have discussed below. Other design choices I made were including the title of the webpage on each page. This is useful for the user to see what page they are on.

Bootstrap

I have used Bootstrap to create a more responsive web page specifically on mobile devices. The main integration with Bootstrap is through the navigation system, cards and buttons used throughout displaying content.

Navigation System

The navigation system I have used is very simple. It contains four links – All Assessments, Create Assessment, Complete Assessments, Uncomplete Assessments. I have also added a toggle navigation button so that on smaller devices (more specifically when the width of the device is too small to display the entire navigation bar) it minimises into a small button, that can then be pressed to reveal the expanded elements in the navigation bar. The toggle button also makes use of Bootstrap JavaScript. When pressing the button, it will reveal the elements in a simple drop-down animation.

Cards

To display assessments, I have made use of Bootstrap cards. Each card contains a title (the assessment title) and then the main card text (the assessment details – module code, deadline, status, and description). Under the card's text I have placed three buttons – edit, change status, and delete. The edit button allows you to edit the assessments, change status will show 'Mark as Complete' for uncompleted assessments and 'Mark as Uncomplete' for completed assessments and the delete button will remove the assessment from the database. Each card also contains a solid slightly curved rectangular border. Assuming there are multiple assessments, up to two assessments can appear side by side occupying a single row. This way I can make use of the extra space on the screen and it does not look empty.

Buttons

As already mentioned, I have used Bootstrap button within my application. To keep a consistent colour scheme, I have used the dark buttons to match the dark navigation bar and card borders. In addition to the buttons on each card, there are also buttons for submitting the form when creating and editing assessments.

Forms

For the create and edit assessments forms, I decided to keep it simple with input fields and corresponding labels above. I have used a date picker widget for the deadline as to minimise errors and to make validation easier. I also made the description box slightly larger. The entire form is also centred on the middle columns of the webpage to make it feel simple.

Three-tier Architecture

To complete this coursework, I have implemented a three-tier architecture using the Flask framework. The three tiers are presentational, business logic and data access. The basic concepts for all of these combining the facilitation Flask provides allows for the following concepts and specific technologies: forms templating through FlaskWTF and Jinja2 templating engine for the presentational tier, views through Flask routes for the business logic tier and finally models provided by SQLAlchemy for the data access tier.

Implemented Flask Architecture

I have implemented a single CreateAssessment form for creating and editing assessments, multiple templates including a base template, a home template and a create template and a simple Assessment model for the database. These individual components interact together to produce the web application. I have talked more about architecture and specific sections of code below under Problem Solving Technique.

Models

As mentioned I created a simple Assessment model for the database. The model contains 6 columns, the id, title, module code, deadline, description, and status of the assessments. The id is a unique record and a primary key for the table. This way it is easy to pick a specific assessment to mark, edit, unmark, or delete. This is because multiple assessments might have the same name or module code and so we need something to uniquely identify the exact assessments.

Data Types

For data types, I decided that the id should be an integer, as to easily identify any assessment. The title, code and description columns are all of String data types however have different maximum value constraints. The title is limited to 100 characters, module code limited to 10 characters and description is limited to 500 characters. The deadline is of data type date (as we validate every deadline before inputting to the database this is very useful as we keep the data types consistent throughout the application). Finally, I have used a Boolean type for the status of the assessment, 1 to signify the assessment has been completed and 0 to signify it is not completed. This way there is no need to store any excess redundant data.

HTTP Features

At the very basic level the web application is accessed through a web browser at localhost 127.0.0.1 through port 5000. This is important as it makes testing the application easy. However, I also used this to be able to test my application on different devices. By using some additional command line arguments with flask, I used the host parameter at 0.0.0.0 which allowed me to use my mobile phone to test the application. Some browsers, like Google Chrome, are useful in that they show the layout of the webpage and to an extent, functionality of the application. However, using my mobile phone allowed me to see how the application would work on a different operating system and of course, how the console dealt with requests from another device, perhaps multiple devices at the same time.

Request Handling

All the HTTP requests made are between the client (through the web application) and the server. Every request is outputted to the console which displays the unique IP address from the client and the details of the HTTP request. This is useful as when testing the application using my mobile phone (and multiple devices at the same time), I can differentiate the requests from the different devices. It also allows me to see status codes. Through testing, there were many 404 errors and being able to see the requests on the server made it easy to work out what was going on that I need to change or to be fixed.

Application Requests

The two main methods for requests that take place in my application are GET and POST. All my routes use the GET method as every route returns a template using by Jinja2 to display the content of the web page. I use the POST method for two routes, creating and editing assessments. As I am using a form, the POST method is useful to provide the data for the assessment because of submitting the form to a data-handling process (i.e., the python function to add the task to the database).

Evaluation

Overall, I feel like I have completed the task successfully. I have met all requirements of the specification, allowing all assessments to have a specified deadline date, module code, assessment title, a description as well as functionality to mark an assessment as complete or uncomplete. In terms of implementation, I have utilised a database to store the list of assessments, appropriately styled the to-do-list web application and many features including but not limited to permitting a user to add an assessment to the list; marking an assessment as complete; displaying all uncompleted assessments in the list; and displaying all completed assessments.

Testing

Testing was a very important part of developing the web application. The process ensures that the application meets the specification/ user's needs. To begin testing, I used a simple process called requirements gathering to extract the user requirements from the coursework brief as well as what I would expect with a to do list application. Though the specification did not specify, I added some extra functionality such as editing and deleting assessments, as well as marking and unmarking assessments from all pages displaying assessments. I felt this was very important for accessibility which I have mentioned in more detail below. With these requirements I was able to devise a simple test plan.

Tasks included: creating a new assessment with all fields populated, creating a new assessment with missing information (missing title, missing description, missing deadline, and missing module code), marking an assessment as complete or uncomplete, viewing assessments (all assessment, complete and uncomplete on their respective pages), editing assessment, and deleting assessment. Though I never created a formal test plan I was able to work through tests to make such the functionality of the application worked as expected.

Debugging

To ensure that all the tasks above were met and that no errors were present, debugging was going to be very important. As the program is heavily reliant on the web page, there is small to no room for error from the user. Apart from the relatively simple things to check like does the uncomplete assessments display all incomplete assessments, the most important parts to debug was creating and editing the assessment. Since the user can input anything into the form, it was important to limit the number of characters of each input field, using a date field that would make inputting an invalid date difficult and displaying appropriate outputs. To do this I utilised Flask's flashes and templating to output errors or validation errors at the top of the webpage (just under the navigation bar). Using Flask was also useful as any errors were displayed along with all information needed to see where the problem was being caused. This was especially useful in working with the Assessment model and the SQL database.

I also used testing for checking the application was being displayed correctly on different browsers, different devices, and different sized displays. Testing was also important making sure contents of all pages were displayed correctly, especially with using Bootstrap and a custom CSS stylesheet.

Security

I kept the security very simple for this application. As there is no sensitive information there was no need for encryption of the data or database, however it is something that could have been implemented to increase the security. I did implement some security by including a secret token in

the form data that is to be submitted. This secret token is provided by the server when the page is delivered to the user/client using the web application. This approach is adopted by Flask, which assumes that the transmission of the HTML page is not being intercepted. To do this, I enabled WTF CSRF in the configuration file and provided the security key, then configured the app in the initialisation python file to configure from the configuration file.

Accessibility

I have added some accessibility in my web application. The ability to mark and unmark assessments from the 'All Assessments' page makes it easy to mark and unmark assessments as otherwise the user would have to find the assessments under the respective complete or uncomplete assessments page. I have also added the functionality to edit and delete assessments. This way if a user makes a mistake in creating the assessment they can go back and edit the assessment or alternatively delete if needed. The functionality to unmark assessments as complete is also useful in case a user accidentally marks an assessment as complete.

I have also added custom styling through the CSS style sheet to reduce font size for smaller screen sizes and as mentioned previously, a toggle navigation button that expands or retracts the navigation menu options to accommodate smaller screens. Once viewing multiple assessments in a single row becomes too compact, it also switches to display assessments on their own row, again allowing all content being presented in an accessible format. This way the application can be presented on different displays and devices without content being displaced.

The colour scheme is also simple, not including colours that visually impaired people would not struggle to distinguish but also adding enough contrast between elements so that all information is clear to see. I also tested the web application on different browsers to make sure all content was displayed accurately. I also used a widget for the date picker to limit the error margin for the user entering an invalid deadline in any format.

To increase the accessibility, I could have added a search feature allowing users to search through assessments. This would make it easy for users who know what assessment they are looking for. This could be expanding to have search filters, such as for module codes, up to a certain date for the deadline, or contain certain keywords in the description.

Challenges

Creating this application came with some challenges. I think the most difficult challenge was finding a way to get the id of an assessment from the html and pass it into a python function to mark as complete or uncomplete. Eventually I was able to do this by adding another route. After completing this I decided to also add extra functionality to edit, unmark and delete assessments, even from the home page.

The other biggest challenge I faced was just getting to grips with new technologies - Flask, forms, SQLAlchemy and templating. It took some time learning the process however I feel much more confident in building web applications especially with a database.

Problem Solving Technique

To overcome these challenges, I had to use a technique, otherwise these problems wouldn't be easy to solve. As mentioned in the testing section, debugging was a very useful way to finding out errors in the application and to make all the functionality work as expected. However, to understand these new technologies, I used a different method. To learn how all the technologies worked and how they should work to build the application I split up the program into three different tasks – one task for

each of the tiers in the three-tier architecture. I took each tier at a time - forms, templates, and models – learning how they worked individually. Then I looked at how they would interact with each other to achieve a successful result. Utilising resources on the Module Website helped me understand the basics of how everything worked, especially the theoretical side, however websites like Stack Overflow were useful in working out how specific problems need to be solved.

For example, trying to work out how to pass a specific assessment id from the html to a python function to mark as complete is not trivial and can take some time to learn. In fact, it utilised some knowledge from all tiers. I had to use the model and database to get the assessments I needed to display. I then had to use templating to get the assessments and display them on the page. I then had to use a technique through templating and forms to get the assessment id and use it in the python function. However, instead of sending it directly to the python function, I ended up creating a route using the assessment id that would utilise the id as a parameter in the function.

Improvements

As mentioned in accessibility there were a few improvements I could have made by adding different features. Apart from these I think the places I could have improved on is adding more functionality to the program. I would do this by giving the user more features, such as adding colour coding to assessments, add custom tags (that can be searched and filtered through), group assessments (by module code or any combination the user wants to). I could have also added different display modes, for example cards (like it is now) or a more compact look with links to each task displaying them in a list format.

I could have also improved the database with these features too. I could have created another table storing module codes instead of repeating them in the assessment table. This way we would store less data and it would be easy for a user to auto pick a module for which assessments already exist.

References

Images:
https://upload.wikimedia.org/wikipedia/en/2/2a/Notepad.png