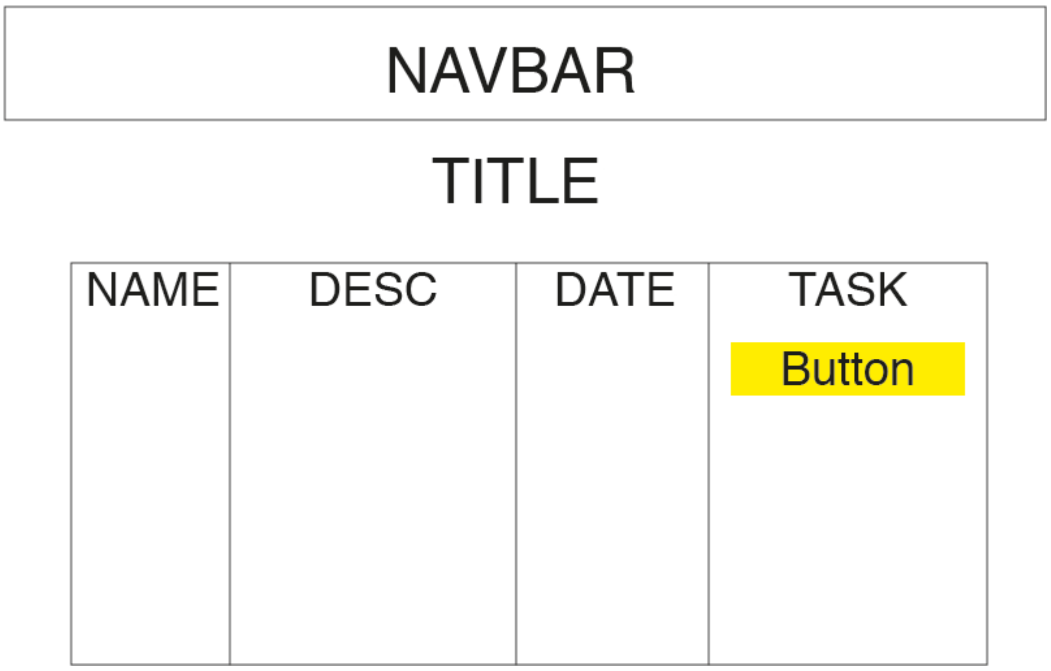
COMP2011 – Report

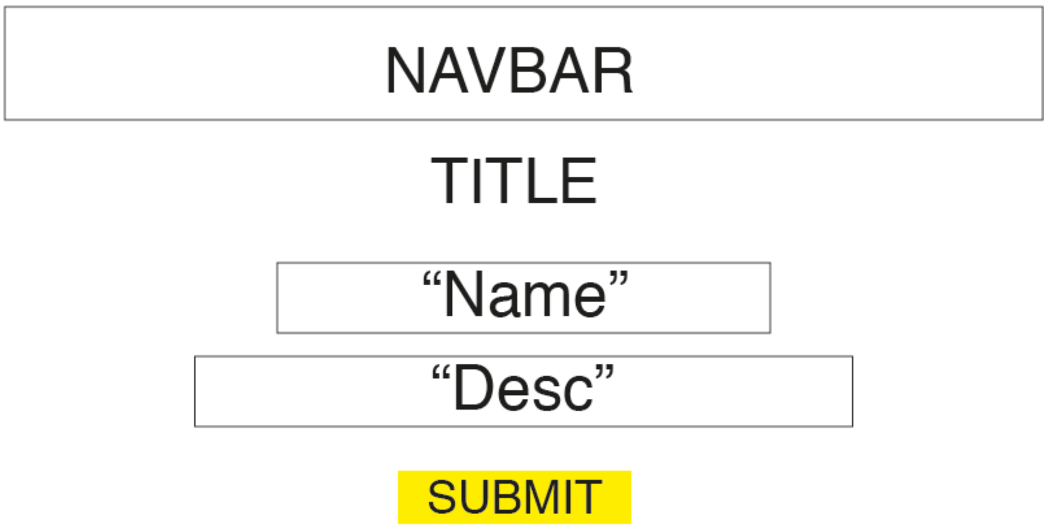
Jacob Holland – sc15j3h

Wireframe Model

List View (Homepage/Pending & View Complete)



"Add new Task" Page



References

This project uses the Flask-Material Framework, this modification to my styling was authorised by Sam Wilson and may need to be installed appropriately if not already done.

Justification of Layout

For the list view, the To Do list has been represented in the form of a table – this allows for clear separation between tasks and allows for each task to be clearly viewed, as it is represented on a single line. The alternative of using bulleted lists would've worked in terms of functionality, but it wouldn't be as intuitive to follow.

For the "Create Task" page, the form has been centred to clean up the layout of the page. Both form entries have received labels and placeholder values so the user knows what to enter and where. The submit button should follow the same type of layout, and by having a button instead of a link tag (<a>), It makes it clear that the user needs to press this button in order to submit the data.

Model Implementation

The ToDo list application utilizes the three-tier architecture model for a web application

Presentation Tier - This tier is the user interface for which the user properly interacts with. On the ToDo list application, the pending tasks are presented in the format of a table, which is easy for the user to interact with and understand. On the "new task" page, an online form is presented, so the user can easily input data and submit it with the click of a button. The presentation tier is a combination of HTML5 and CSS styling (provided by Flask-Material), which is used for templating and shaping the flask framework.

Logic Tier – This layer co-ordinates the application, it processes commands and sorts logic. The main example is implemented using the Flask-WTF extension. When a user submits data, it does so in the form of a POST request in HTML. Flask-WTF allows us to render this data for the presentation tier, as well as process it for the Data tier. The form is represented as a class in python (refer to "forms.py") with 2 different fields – "name" and "description" – Both are represented as String Fields as the user is expected to input a string of text into the fields so they can articulate what they actually want to do. Both are using the "InputRequired" field, in order to prevent the form from submitting empty fields. For this form, there is a security problem of CSRF, which allows third-party users submit data without the primary users permission, this can be easily prevented using WTF, as in the config.py file, we have enabled CSRF prevention and included an encrypted key on the server. The "*{{form.hidden_tag()}}"* part in "new-task.html" provides a unique ID for the POST request for the form to prevent CSRF.

The Logic Tier interacts with the presentation tier by displaying the correct form elements for which the user can input and submit the correct data (which has been appropriately styled via "new-task.html"). Once the data has been inputted by the user and they have clicked "Submit", then the data is processed by Flask-WTF and then moved down and stored into the Data Tier.

Data Tier – this is where the data from the form is processed and retrieved in the form of a database system. Using SQLAlchemy in flask, we are able to implement a database with relative ease. It is represented as a class in Python (check "models.py") and has more fields than the form.

- ID = This is the primary key of the table and is used for easily representing an entry in the table, this is sequentially generated in the database upon each new entry as an integer value
- Name – This is the name of a task, this is stored as Unicode as it allows for unrestricted input, if we want to restrict the input, we can either do it in the Form class, or by changing the Type in the model to String.
- Name – This is the description of a task, this is stored as Unicode as it allows for unrestricted input, if we want to restrict the input, we can either do it in the Form class, or by changing the Type in the model to String.
- Creation Date, this is the date on which the task was created, this is typically represented as a value, it has a default value of "UTCNOW" so it generates the correct date upon submission
- Is_Done – this is a Boolean variable to see if the task is complete or not, and is primarily used for filtering. It is set to default upon creation, but can be altered by the user afterwards.

The database file is created using "db_create.py". This creates a file called "app.db" where the entries are stored. If there are changes to the database, then this can be altered using the db_migrate, which allows the DB model to change, or even be rolled back to an earlier iteration if something were to go wrong – these are stored in the db_respository directory in the file system.

In short, the three layers interact in an operation like so – the User inputs data via the presentation layer and submits a POST request, this data is then processed through the Logic Tier, where it creates a new database entry using the specific string inputs submitted by the user, this change is then committed and stored to the Database file, when a GET request is submitted, the data is retrieved from the Data Layer by the logic layer in the form of a query (which is handled by SQLAlchemy) This data is then represented sequentially in the form of a table on the presentation layer, which the user can interact with.

If the user wises to mark a task as done, he submits another POST request. The logic layer will then obtain the ID of the entry that the user wishes to change. The Boolean value of that entry in the Data layer is changed to "true" and the logic layer then refreshes the page to show the change has been made for the user on the presentation layer.

Evaluation

Testing Table

What I'm Testing	**How**	**Expected Result**	**Actual Result**
If hyperlinks work	Click every hyperlink in the navigation bar	Every link should navigate to the page advertised	As expected
If the website scales appropriately	Use developer tools to view the website at various resolutions	The contents should resize	The page typically scales well, but there seems to be an issue with the mobile navigation bar not activating.
If the form accepts an empty entry	Submit a new task form with no data entered	The form should not validate and the webpage will refresh	As expected
If entries add to the database correctly	Add a new entry via the "new task" form	The form should validate and add the entry – displaying it on the home screen	As expected
Database filters work correctly	View Pending and complete tasks correctly, and view them against the database values	The pending tasks page should only show entries where is_done = False, completed tasks should display the opposite	As expected
If the "mark as done" button works correctly	Create a task, and click on the "mark as done" button	The pending tasks page should refresh and no longer show that specific task - the completed tasks should show this entry	As expected

From these brief test cases, the application typically works as expected, however it has become apparent there is a scaling issue in my application. This primarily seems to be because of the discrepancies in the flask-material plugin and its failure to load in the proper scripts and icon files for the application. If I were to rewrite this application, I would not be using the wrapper files for flask, but instead load the files manually and link it in my own base file for complete control and hopefully get a better expected output.

The primary challenge in this application was getting the data from the form to properly process into the database. In hindsight, the process was relatively simple, but the transition from using SQLite syntax to SQLAlchemy plagued me with errors. After reading the correct documentation, this problem was eventually fixed.

There are some changes to my initial wireframe document, the table was supposed to be centred on inception, but when attempting to implement this, only the header of the table would successfully align. To keep consistency, I left aligned the tables.