COMP1021

Programming Project

**Web Application Development**

**Coursework 1 Written Report**

Samuel Barnes

Sc18sjb

Word Count: /

Character Count /

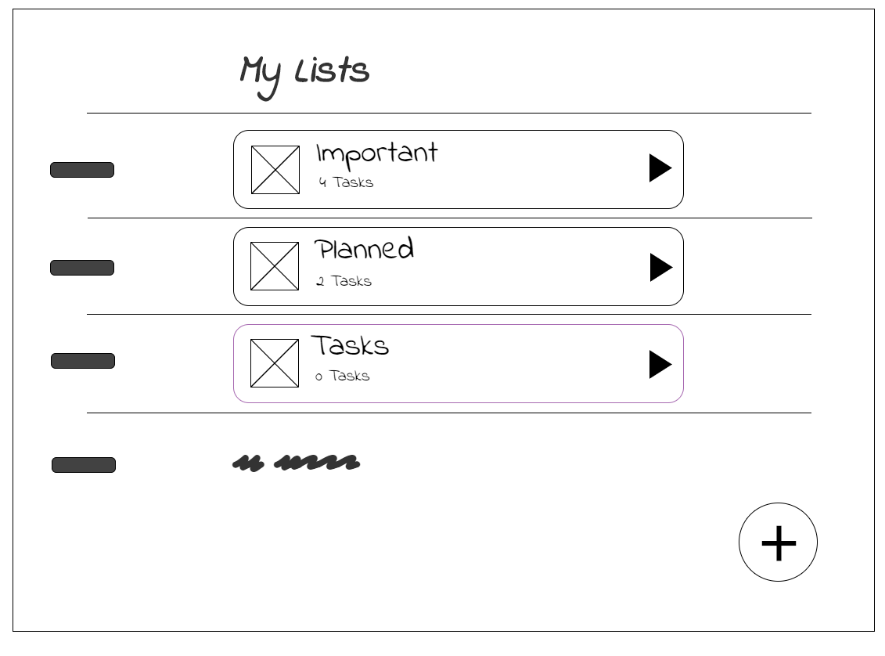
# **Wireframe Model Designs**

To design the layout of the web application, Wireframe Models have been designed for each page layout within the application. These can then be followed during the implementation of the application, as a guideline towards the styling and layouts. Each Wireframe Model has been annotated with a justification of the choice of layout for each specific design.

Links to each list available to the user. These will be styled appropriately with the colour and icon of the list, as chosen by the user (when creating the list).

# **Index layout**

The website will be styled to look like a notebook, these will represent the binders of the page, with the underlines representing the page lines.



‘create’ button; will redirect the user to the create list template, where they can create a new list to be inserted into the database.

The name, tasks completed / uncompleted and icon of each list will be presented.

Presentation Tier:

The index layout of the web application will allow users to view all the lists available to them (which they will have created). Therefore, the page is styled in a way to clearly present each list as a button, which can be clicked on to take them to that list; showing all the tasks within it. I have chosen the layout of the webpage to be styled like a notebook, with each line representing a different list. The reason for this is to try and present my to-do application as a digital representation of a notebook (which is typically used for writing down tasks to-do). The layout embodies this by using ‘lines’ to split up content, and having the fonts styled to look like handwriting. Furthermore, I have included decorative elements to represent the binders of the page.

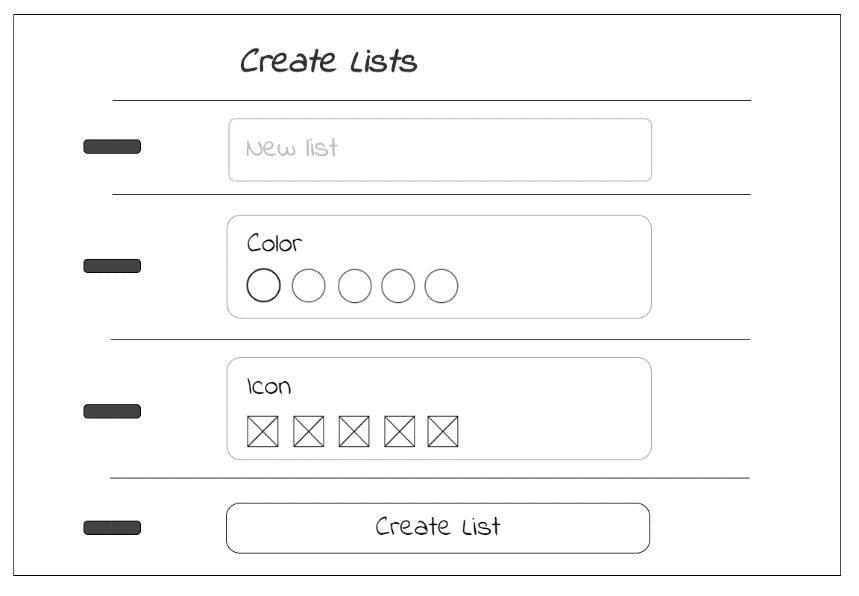
In terms of colour, the background of the page (further trying to symbolise a notebook) will be styled off-white, which will also add some clarity to the page. As mentioned, each list button will be styled with the colour represented to that list; chosen by the user. Having these colours will allow each list to ‘pop’ and stand out; as they are the main component of the webpage, I want users to be drawn towards them, so having them ‘pop’ will achieve this.

Logic Tier:

To retrieve each list to be displayed upon the webpage, there will need to be a query to get all the lists within the database; I can use the query.all() operator on the list class model to achieve this. Once each list has been returned, I will need to allow the user to ‘access’ this list, to show all of the completed / uncompleted tasks within it (add & complete layouts). Each list can therefore be presented as a form within the HTML, with the list ID presented as a hidden field within the form. This form (once submitted) can then redirect to a view handler, which can parse the list ID and redirect the user to the add layout, setting the URL to the ID (which can then be handled through the GET method).

# **Create Layout**

Name of the list; Input Box



List icon selection. As with the colours, these will also be available to select by clicking on them.

List colour selection; all the colours will be available to select by clicking on them (as a colour selector).

Submit button to request the list to be inserted within the database.

Presentation Tier:

The create layout will allow users to insert new lists within the application, these lists can be given a unique name, colour and icon for representation. The page will follow a similar layout to the index page, using the same handwritten aesthetic to present a notebook. The form a layout to improve its usability. For example, when picking the colour of the list (instead of just having a drop-down of the different colours available) I will use a specific colour-selector to display all the colours available to them, which the user can simply click on to select. This can be implemented using JavaScript and will vastly improve the interactivity of the webpage. The same layout design will be implemented on the icon selector of the list.

For colour, again I will use the same off-white background to convey this is a page in a notebook. Each section of the form should stand out from the background; border shading can be used around the elements to achieve this, with the actual elements being styled white. The submit button however, will be styled green to differentiate it and convey you are submitting the data.

Logic Tier:

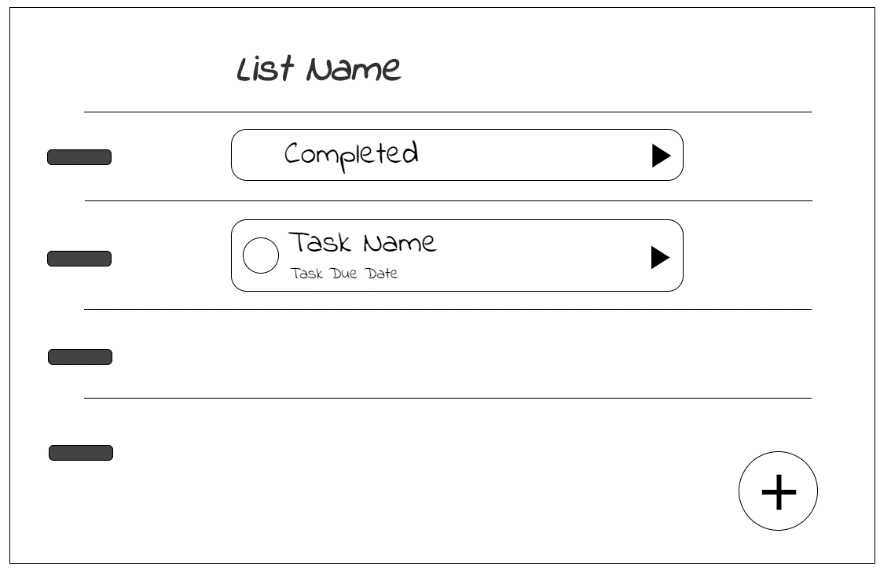
Logically, the webpage will need to use JavaScript for both the colour and icon selectors. Since I am using no pre-defined rules (such as bootstrap) within my web application, I can achieve the functionality by simply differentiating each colour uniquely (with its name), and then pulling the requested colour and returning a class to show it has been selected. Another hidden field within the form can represent the actual colours data, with the value of this field being changed within the JavaScript, dependant on the name of the pulled colour. To summarise, whatever the name of the colour the user selects, the data of the field will be changed to that name.

Upon submitting the form, an insertion query will need to take place on the database to input a new record with the forms data. This can be achieved by redirecting the form to a view which will handle the submit request, creating a new list object with the forms data as the attributes, then inserting the new record within the database with the add(record) command (making sure to commit the databases changes).

# **Add Layout**

Allow users to mask tasks as complete by ‘ticking’ the checkbox.

When a task is marked as complete, a button allowing the user to view the completed tasks of that list will appear.



‘add’ button will allow users to add a new task to the list by opening a modal form. This form will be styled identically to the create layout.

Presentation Tier:

The add layout will display each of the tasks within a specific list and allow users to add a new task to this list. Furthermore, users can mark a task a complete which will then create a sub-section of the list, where users can view completed tasks; completed layout. For continuity through the application, the page will use the same layout and styling as the previous layouts. Main changes to the layout include a checkbox on each task to mark it as completed; the justification for this is that it very clearly marks how to ‘tick’ a task as complete, again thinking of the layout similarly to a notebook.

To add a new task, I have decided to include the ‘add’ form within a modal that will appear when the ‘add’ button is pressed (which has the same styling as the ‘create’ button within the index layout). Some simple JavaScript will have to be implemented to incorporate the functionality of opening and closing the modal. The form will use the same layout as the create layout, with the same styling on each input and button; allowing for continuity throughout the application that improves usability.

Colours of the layout will embody the unique colours of the lists; such that the chosen colour of the list will be the background colour of the page. To make the elements stand out from this background, which will change from list to list, each section will be wrapped in a white container that will use border shading to make it ‘pop’ from the background. We want users to be directed towards the elements on the page, so this is a way to achieve this. The same affect will be applied to the modal when it is open.

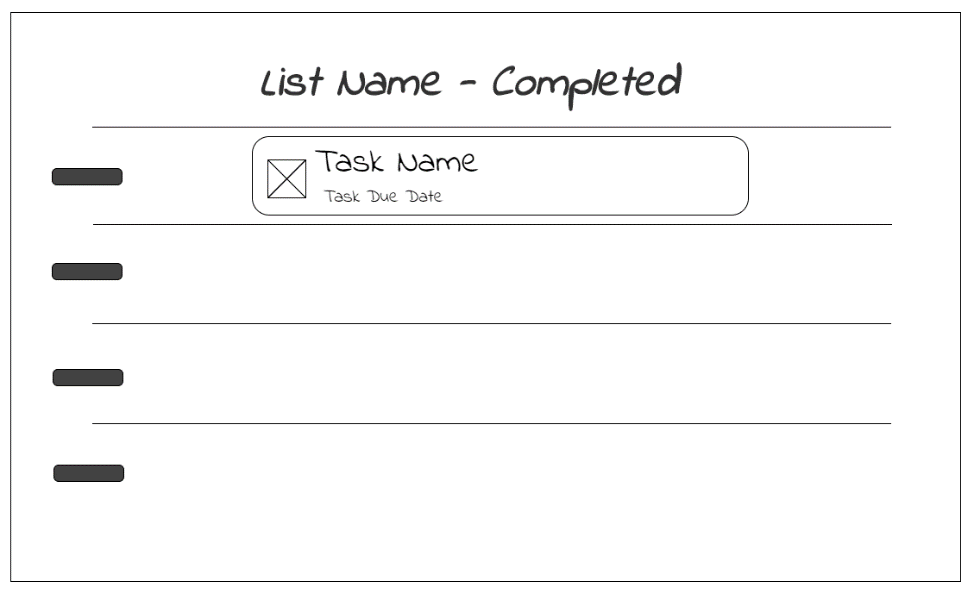
Logic Tier:

As with the index layout, to retrieve each of the lists tasks a request upon the database will need to take place. I can use the query.get\_or\_404(attribute(s)) operator, where the attribute is the ID of the list pulled as a GET request from the URL which was set when redirecting from the index page. From the retrieved list, I can then simply iterate over the lists ‘tasks’ attribute (which is a one-to-many relation between list and tasks) to display each of the tasks within the specific list.

A request to insert data will also need to be facilitated, to allow users to add new tasks to a specific list. The ID of the list will need to be set within the new task’s attributes, along with the data pulled from the form when submitting; all this data will be added to a new task object which can then be inserted within the database using the add(record) command, where the record is the new task object created (making sure to commit the database changes).

Tasks should also have the functionality to be marked as complete. The complete checkbox will be a form with a hidden field set to the specific tasks ID. Once submitted a query.filter\_by(attribute(s)) operation will take place, where the attribute is the ID of the task set to the hidden task ID from the forms data. The tasks complete attribute can then be set to true, with the changes committed (and the page refreshed).

# **Complete Layout**



Each of the completed tasks will be listed in the order they were completed in.

Allow users to mark tasks as uncomplete by ‘unticking’ the complete image next to the task.

Presentation Tier:

Finally, the complete layout will display each of the completed tasks (which users will have marked as complete) within the specific list. From the add layout not much will have changed to the design. This is by choice to create consistency between the web pages, and because the task accomplishes the same functionality as the add layout; just displaying complete tasks instead of those uncomplete. Each task, instead of being able to be marked as complete, can now be marked as uncomplete. Although this feature is not necessary for the requirements, implementing it allows for a better user experience; say a task was accidently marked as complete, or is no longer complete, then it gives users the option to revert their decision.

The colours within the layout will also remain identical to the add layout; having the background be the unique colour chosen for the list, with each task being in a white container with border shading to make it ‘pop’ from the background and draw users towards them.

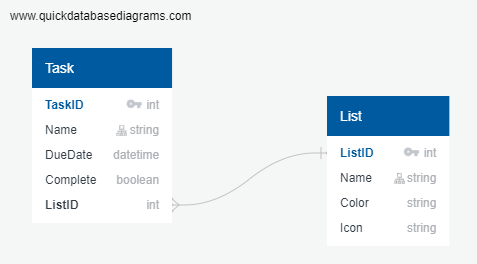
Logic Tier:

Instead of requesting tasks to be marked as complete, there will now have to be a request to unmark complete tasks. This functionality will be identical to that within the add layout, using the same form for each task containing a hidden field with the tasks ID. However, instead of marking tasks complete attribute as true, it will instead change it back to false, with the changes committed (and page refreshed).

# **Security**

To ensure the web application is secure, each of the forms discussed in each page layout will need to include either a csrf\_token() or hidden\_tag() tag dependant on if the form has any hidden fields; if the form does the first tag will have to be used, as to not duplicate hidden fields. The purpose of these tags is to prevent cross site request forgery by including a secret token within the forms data that is submitted. This can be configured within the web applications config file, setting WTF\_CSRF\_ENABLED to true and setting a SECRET\_KEY variable will enable cross site request forgery prevention within the web application, therefore securing it against any of these attacks.

# **Database Design (Data Tier)**



There will be two models implemented within the web application; a Task model and a List model. There will be a one-to-many relationship between the List and Task models, meaning one list can have many tasks (modelled above). This allows for the functionality of the web application, so that each list can have multiple tasks associated with it. Both models with have ID attribute as their primary key. These will be unique integers to identify each record of the specific model type within the database. Both models will also have ‘Name’ attributes to represent the name of the task / list. These will be of type string as to allow for text to be stored within them. They will also likely have a max string length of 20 to limit the length of the names; there is no need for a task / list with a name longer than 20 characters, and this takes up more space within the database, especially if there are a lot of records. The rest of the attributes differ between the two models.

Within the Task model there will be a ‘DueDate’ attribute to represent the date that a task is due (i.e. if it is important, to make sure it is done). This will be an optional DateTime attribute formatted as ‘dd/mm/yyyy’; using the current standard for date representation (in the UK). The attribute has to be of type DateTime to allow it to store dates, whilst a string could be used, it would not be the most effective method as further computation would need to be done to interpret these strings, whilst the DateTime does this for us. The ‘Complete’ attribute is simply a Boolean field representing true / false value for if the task is marked as complete. Finally, the ‘ListID’ attribute facilitates the relation to the List model; it is foreign key of the ‘ListID’ which the task is related to. The attribute must be of type integer because that is what the ‘listID’ attribute within the list model is for the relationship to take place.

The List model has two attributes meant as unique customizers for the specific list. These are the ‘Color’ and ‘Icon’ attributes which are both of type string. The ‘Color’ attribute simply holds the name of the colour of the list within the string, which can then be used when rendering the form as a style class, so that the list is of the chosen colour. The ‘Icon’ attribute will also simply hold the name of the icon within the string, which can then be concatenated onto the src of the ions image when rending the form. Since all the icons are within the same directory and of the same type (PNG), there is no need to specify the path to the icon or the type, since this info can just be concatenated onto the name of the icon.

# **Evaluation**

With the forms, say about how I will secure them (such as with hidden\_tag()), also say about what the forms do and how they will input data into the database – show an understanding of handling requests.

Include designs of the database, including the relationships, the attributes in each table and the data type of each attribute and WHY it is that data type.

# **References**

[1] *Star Icon*, downloaded free from Iconfinder, <https://www.iconfinder.com/icons/326487/rate_star_icon>

[2] *Sun Icon*, downloaded free from Pinterest, <https://www.pinterest.co.uk/pin/700169073290018075/?autologin=true>

[3] *Delete Icon*, downloaded free from MaterialUI, <https://www.materialui.co/icon/delete>

[4] *Calendar Icon*, downloaded free from MaterialUi, <https://www.materialui.co/icon/calendar>

[5] *Arrow Icon*, downloaded free from MaterialUi, <https://www.materialui.co/icon/keyboard-arrow-left>

[6] *Return Icon*, downloaded free from All-free-download, <https://all-free-download.com/free-vector/download/gray-arrow-icon-vector_280878.html>

[7] *Tick Icon*, downloaded free from MaterialUi, <https://www.materialui.co/icon/complete>

[8] *Writing Icon*, downloaded free from Flaticon, <https://www.flaticon.com/free-icon/school-material_182148>