SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

[Requirements] Elicitation

Questions:

NB: The sub questions (1.1, 1.2…) are guideline questions, some may be answered before being asked and some might not need to be asked. The important bit is that we get answers to the three main questions; the sub questions are simply if further understanding is required. We decided this should be more of a conversation than a questionnaire.

1. ***What are the current problems you have with the collaboration and communication applications that you use right now??***

   For example, are there any features not provided in the app that you would want to see, or any features in the app which you think could be improved? We want to know what we can do to make this your go-to communication tool.

   1.1. What is the problem you have/ what is it that you are missing?

   1.2. Where and when does this issue negatively affect collaboration?

2. ***What is the real problem you are trying to address?***

   Perhaps the issue you have raised is a subset of a larger problem that needs to be addressed. So in order to address the actual problem, we pose further questions:

   2.1. So…is there a different way to look at the problem? Use keywords from the answers to question one to try and dig out the bigger problem that is being described.

   Assume we come up with a draft solution to the issue raised (perhaps the user has already suggested what a solution might look like by now, or we as the engineers have suggested one).

   2.2. When will this feature be used?

   2.3. How will the new feature actually affect your collaboration with others? What improvement in communication (requirement) are you trying to achieve?

   2.4. Different consumers have different needs and expectations. Is the new feature something that other users might only use once in a while? If so, is there a larger issue that we need to address which will improve **everyone's** day-to-day collaboration?

   2.5. Get the user to elaborate on the negative effects of **not** implementing the new feature.

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

3. ***What do you think the solution should look like?***

   Describe what you would regard as a successful solution (a feature which solves the problem raised).

   3.1. How may we implement this feature?

   3.2. Where does the process start (e.g. clicking a button)?

   3.3. What are the results we want to see?

   3.4. What are the inputs by the user and outputs by the application?

   3.5. What assumptions are we making about the implementation of this new feature?

   3.6. What would you regard as a pointless, or failed solution?

   3.7. What if…? (suggest implementations that we as the engineers come up with and use the response given to further understand the requirements)

   3.8. How can we judge the success of our solution?

   3.9. Are there any risks/side effects that you want us to make sure we avoid upon implementation?

   3.10. How should we test our solution

NB: We thought that getting the people we are interviewing to talk about the solution they want to see in detail would be the best way to try to understand their problem. If this was not possible, we concentrated more on question 2.

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

Q	Target User 1: Nicole Nanayakkara, n.nanayakkara@ad.student.unsw.edu.au
1	1.1. The group chats list is unnecessarily populated, and I can't organise the group chats into the categories. 1.2. Overpopulated chats bars can put off collaborators who like to be organised, its ugly to look at which can be annoying especially when collaboration needs to be extremely organised.
2	2.1. The idea here is that the channels sidebar is not organised well - a collaboration tool needs to be organised well in order to work well. 2.2./2.3./2.4. Having a better organised sidebar makes it easier for users to communicate. For example, if the user was able to adjust their sidebar to have subheadings so that they could categorise the chats which relate to specific subjects or projects depending on their liking: COMP1531 - Major Project - Tutorial Channel - Course Channel SENG1521 - Major Project - Tutorial Channel - Course Channel 2.5. An unorganised chats bar might minimise the number of chats users want to be a part of (in order to avoid having an overpopulated sidebar); trying to find a chat would be very hard.
3	3.1./3.2./3.3./3.4. It would be great if there was an "Add Category" button upon creating a chat to add subheadings. Once a user clicks this, they can name the section (e.g. COMP1531). 3.5. We assume that there can be as many categories as the user wants, and as many chats within a category as the user wants. 3.6./3.8. An unsuccessful solution would be one where the sidebar is **still** overpopulated; so, perhaps the channels relating to a section can only appear once the section has been clicked on. For example, At the beginning all you see is: COMP1531 SENG1521 But once you click 1531, the channels relating to this section appear. 3.9. You must avoid over complicating the organisation process; users don't want to spend too much time having to organise their channels.

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

Q	Target User 2: Amaan Mohideen, amaanmohideen@gmail.com
1	1.1. Limited collaboration capability; my collaboration application does not have in-built tasking tools - specifically a to-do list. 1.2./1.3. Having to keep up with what I need to do is hard when I have many projects going on and lots of school work. To solve this, I could use a to-do list but I just never get around it because then I have to open an entirely new application/write down on a piece of paper.
2	2.1. This can hinder communication and collaboration; an app which doesn't help with keeping a track of tasks is in no way customised to be a collaborative tool. 2.2./2.3./2.4./2.5. The larger problem is that the app needs to be more creative in the way it engages communication and collaboration to help enhance teamwork. Text messages alone just don't make the cut.
3	**TO-DO LISTS** {Upon the insightful idea of the to-do lists, we informed the user of the previous discussion we had with Nicole (target user 1), and told them about how we will implement a by-subject organisation of channels. We suggested that we could further update this new feature to simply have a 'To-Do' list button available where users may assign the to-do to a specific channel. Both Amaan and Nicole liked this idea.} 3.1./3.2./3.3./3.4. The user clicks the 'To-Do' button and a to do list appears. The user can then add/remove (tick/untick) things on the list. 3.6./3.9. The to-do must be simplistic, and the user should be able to easily switch between the to-do list and their channels. The idea is that the user can quickly add things to their to do lists during discussions.

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

[Requirements] Analysis & Specification - Use Cases

User Story 1: Nicole Nanayakkara, a student.

As a student, I want to be able to categorize the channels in my sidebar, so that it is easy to navigate through channels in an organised manner.

- On the app, categories will be referred to as ‘subjects’
- When the user enters the app, only categories are shown (not channels)
- If the user selects a category, the channels relating to this category are displayed
- The user can then select a channel and start using it
- Initially, there are two buttons available on the sidebar: ‘+ New Channel’ and ‘+New Subject’
- There is also a pre-set category (subject) named ‘General’
- If ‘+ New Channel’ is selected, a channel is created and the user must select the name of the category (subject) they want to add the channel under upon creating the channel. When no category is selected, the channel is automatically added under ‘General’
- If the ‘+ New Subject’ button is selected, a category is created with the name the user inputs
- The user can only have names for Categories which are less than 20 characters
- Each channel may only be under one category

Use Case:

INITIAL TEMPLATE

- Use Case: Create a category (subject) to organise channels
- Goal in Context: Create an organised list of channels which are organized by subject
- Scope: Channels sidebar, Categorization
- Level: Subfunction
- Preconditions: The user creating the channel and all members who are to be invited already have accounts in the Flockr app.
- Success End Condition: A user is able to navigate through channels they are a part of in an organised manner.
- Failed End Condition: No category (subject) is created
- Primary Actor: User
- Trigger: A user clicks on '+ New Subject'

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

MAIN USE CASE SUCCESS SCENARIO

Step 1: A users clicks on '+ New Subject' on the frontend
Step 2: Frontend of the app asks user to declare a name for the channel
Step 3: User types 'COMP1531' as the name
Step 4: Frontend of the app asks backend to verify that the user sending this request is a valid user with a valid token, and if so asks the backend to create the category
Step 5: Backend informs frontend of validity and returns details about the new category (subject) to be displayed on the frontend
Step 6: Frontend asks user what action they wish to take
Step 7: A user clicks on 'create channel' on the frontend
Step 8: Frontend of the app asks user to declare a name for the channel
Step 9: User types 'Major Project' as their name
Step 10: Frontend of the app asks user whether the channel should be public or private
Step 11: User selects private
Step 12: Frontend of the app asks the user to specify the category in which the channel must be associated
Step 13: Frontend of the app asks the backend to verify that the user sending this request is a valid user with a valid token, and if so asks the backend to create the channel and assign it under the category ‘COMP1531’. This is added to the category details stored
Step 14: The user selects the category COMP1531 on the frontend
Step 15: The frontend asks the backend to validate the user, and then return the details relating to this category
Step 16: The backend validates the users and returns this information for the frontend to display
Step 17: The user selects the channel ‘Major Project’
Step 18: The frontend asks the user to validate the backend and return the details relating to this channel
Step 19: The backend validates the user and returns the information about the channel
Step 20: The user starts using the channel

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

User Story 2: Amaan Mohideen, a student.

As a student, I want to be able to use to-do lists, so that I can keep track of the tasks assigned to me when collaborating.

- The ‘To-Do’ button will be above the channels buttons on the sidebar
- When the ‘To-Do’ button is selected the main page displays the to-do list
- Here, a ‘+ New Task’ button is available
- When selected, the user describe the task shortly and can assign this task to a specific channel
- The description must be less than 200 characters
- When the task is created, the To-Do list displays it. It must be organised by both category (subject) and channel (the channel with the subject)
- The user may tick/untick to mark a task as done/not done
- The user may delete a task

Use Case:

INITIAL TEMPLATE

- Use Case: Add tasks to the users’ To-Do list, and marks it as done
- Goal in Context: Create a categorized to-do list
- Scope: To-Do list function, channels functions
- Level: Primary Function
- Preconditions: The user creating the channel already has an account in the Flockr app.
- Success End Condition: A categorized to-do list is created and displayed
- Failed End Condition: No to-do list is displayed
- Primary Actor: User
- Trigger: A user clicks on ‘To-Do'

MAIN USE CASE SUCCESS SCENARIO

Step 1: A user clicks on the ‘To-Do’ tab
Step 2: Frontend of the app asks backend to validate the user, and then send back the users’ to-do list details
Step 3: User clicks ‘+ New Task’, and gives the description of the task, then clicks the ‘assign’ button to assign the task to a channel
Step 4: Frontend asks the backend to validate the user and then get the possible channels to which the task may be assigned to
Step 5: User picks the channel to assign the task to
Step 4: Frontend asks the backend to validate the user, and then create the task.
Step 5: Frontend asks the backend to get the new to-do list details
Step 6: Backend finds the category to which the channel is assigned to, and returns this with the to-do list details
Step 7: Frontend displays the to-do list with the new task categorised by both category (subject) and channel
Step 8: When the user finishes the task, they click the tick which marks the task as done

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

[Requirements] Validation

Users' comments on the extent to which these use cases would adequately describe the problem they're trying to solve	
Target User 1 (Nicole)	The use case sufficiently addresses the issue of unorganised chats. I like that you went a step further and only displayed the channels once you click a category, this keeps the chats bar really concise.
Target User 2 (Amaan)	The building of the to-do list is really well done, the simplicity is great and the fact that you can mark a task as done without necessarily removing it is nice because recurring weekly tasks don't need to be typed again and again.

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

[Requirements] Interface Design

NB:
Any text in yellow means we are adjusting the implementation of end points already in the spec to allow implementation of new features.

Any text in light blue highlights completely new endpoints designed.

Any text in white is already specified in the specification.

Data Types:

Variable Name	Type
Has suffix _id	Integer
contains substring **name**	string
task_status	boolean
outputs only) named exactly **categories**	List of dictionaries, where each dictionary contains types {category_id, name}
outputs only) named exactly **tasks**	List of dictionaries, where each dictionary contains types {task_id, name, channel_id, task_status}
(outputs only) named exactly **channels**	List of dictionaries, where each dictionary contains types { channel_id, name, category}

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

Interface:

Function Name	HTTP Method	Parameters	Return Type	Exceptions	Description
Channel Categorising					
channels/create category	POST	(token, name)	{category_id}	**InputError**: - Name is not between 1 and 20 characters - Name already used **AccessError:** - When the token passed is not a valid token.	Creates a new category with that name.
channels/create	POST	(token, name, category_name, is_public)	{channel_id}	**InputError**: - Name is more than 20 characters long **AccessError** - When the token passed is not a valid token.	Creates a new channel with that name that is either a public or private channel. Assigns the channel to the category given; if the category is not specified, then the channel is assigned to a category named 'General'
channels/listcategories	GET	(token)	{categories}	**AccessError:** - When the token passed is not a valid token.	Provide a list of all categories, including the pre-made 'General' Category which is already created when the user creates their account.
channels/list	GET	(token)	{channels}	**AccessError:** - When the token passed is not a valid token.	Provide a list of all channels (and their associated details) that the authorised user is part of, organised in the categories in which the channels must be listed in.

NB:
We did not have to adjust the channel_details endpoint because that is used to display the channel details on the main screen and the frontend doesn't use channel details for the sidebar. What is used for the sidebar is channels list and channels list_all. List_**all** lists the functions that the user is not a part of, and so can not be categorized for them - hence we did not have to adjust this either. The only two functions that needed adjusting were channels create and channels list. The listcategories endpoint is used to list the categories ONLY, the list endpoint is used to list the channels organised by categories relating to the user.

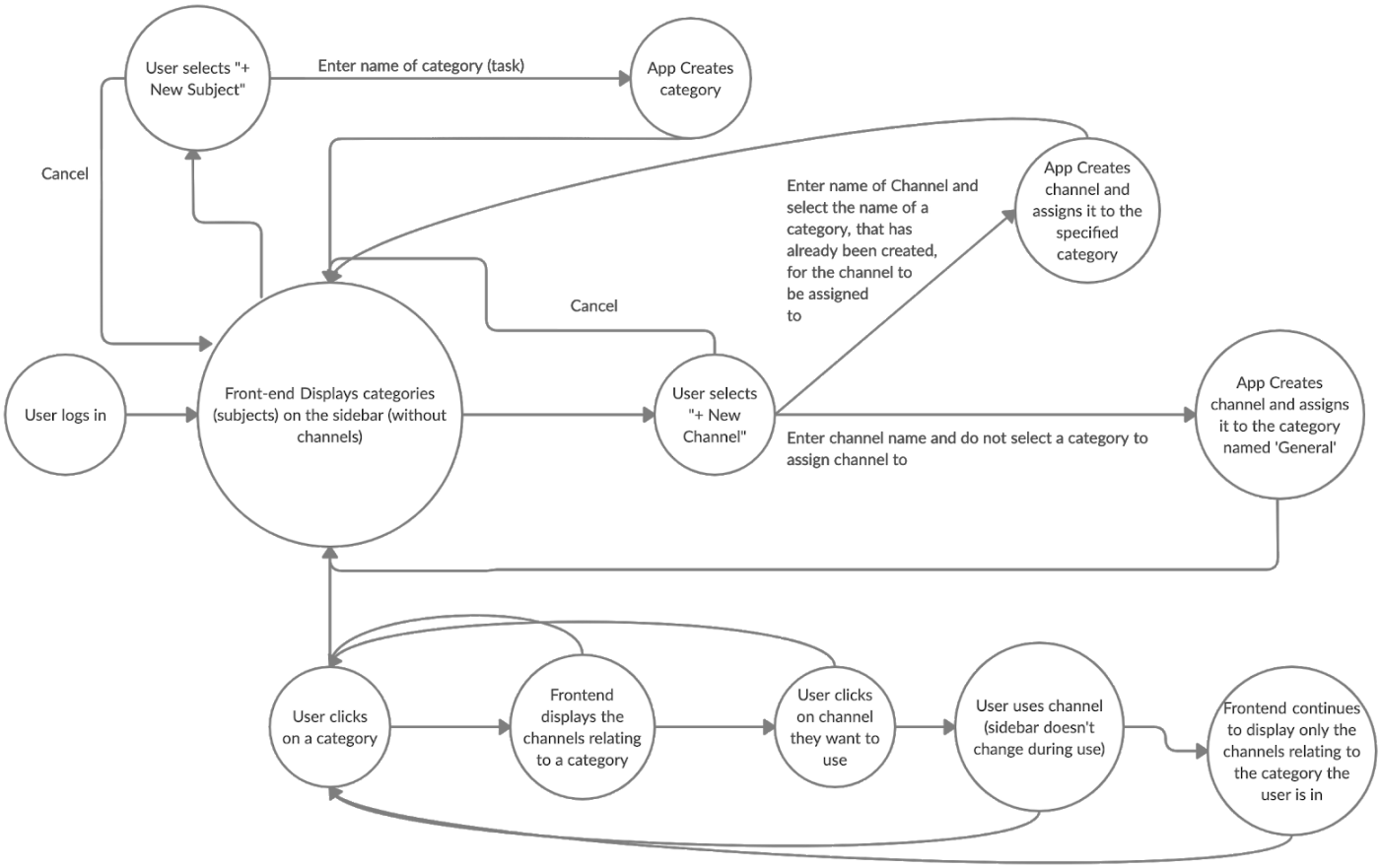
SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

To-Do list					
todo/details	GET	(token)	{tasks}	**AccessError:** - When the token passed is not a valid token.	Displays the users' to-do list organised by channel.
todo/task/create	POST	(token, name, channel_id)	{task_id}	**InputError**: - Name is not between 1 and 20 characters long - Channel ID is not a valid channel **AccessError** - When the token passed is not a valid token.	Creates a new task with that name and assigns task to channel with channel ID channel_id (task must store the id of the channel it is assigned to, not vice versa).
todo/remove	DELETE	(token, task_id)	{}	**InputError:** - task (based on ID) no longer exists - task_id is not a valid id **AccessError** when none of the following are true: - token is not a valid token	Given a task_id for a task, this task is removed from the to-do board
todo/tick	POST	(token, task_id)	{task_status}	**InputError:** - task_id doesn't refer to a valid task **AccessError**: - token is not a valid token	Marks the task with task_id as not done. This means the task status is set to done.
todo/untick	POST	(token, task_id)	{}	**InputError:** - task_id doesn't refer to a valid task **AccessError**: - token is not a valid token	Marks the task with task_id as not done. This means the task status is set to not done.
NB: To get the possible channels in which a task can be assigned to upon creating a task, the frontend will use channels_list; hence we do not have to define a new function to do this.					

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

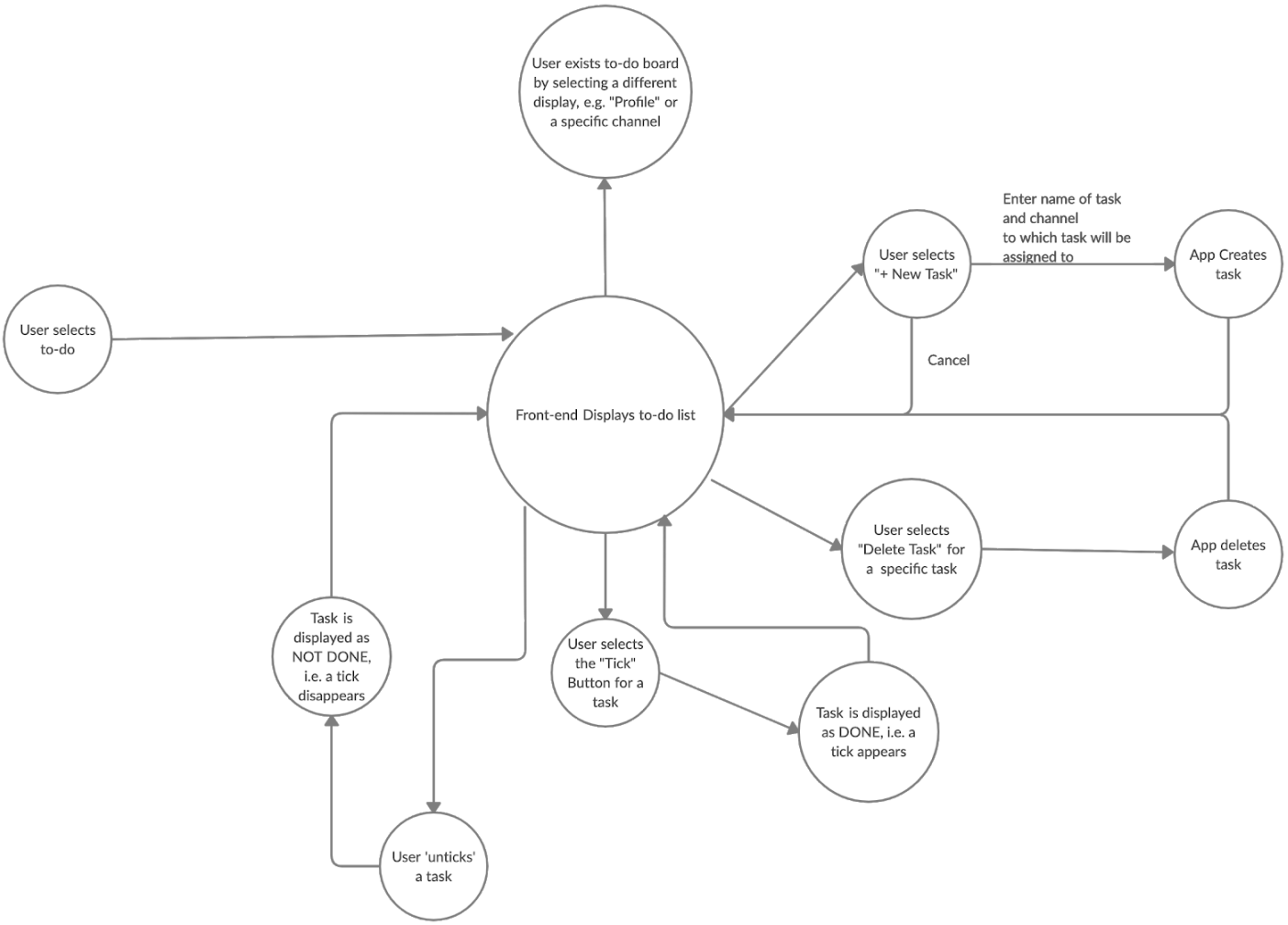
[Requirements] Conceptual Modeling (State)

Categories State Diagram



SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

To-Do List State Diagram



SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE

SECTION 5.3 – PLANNING FOR THE NEXT PROBLEMS TO SOLVE