Iteration 3 M13A_AERO Planning Report

Requirements: Elicitation

Two target users were interviewed, Gabriel (z5363966@unsw.edu.au) and Raman (z5363840@unsw.edu.au).

The following are the set of questions we designed, and their responses. Note that "teamwork-driven communication tools" will be abbreviated to "TCTs" throughout the report for readability.

How do you typically use TCTs, and if infrequently, why so?

G: very often, almost a part of the necessary workflow
R: very often while working in a group, otherwise has no need for TCTs as there are better options available

What features do you use the least on TCTs?

G: events notifications tool, event planning tool
R: live transcription, white board functions

What features are limited on TCTs that should be unlimited?

G: ability to use the same account across many different device, unlimited voice call room creating ability
R: capped meeting times, limits on the size of data to be sent

What kind of issues, if you had any, did you have in joining discussion?

G: UI not great, too many clicks/taps too navigate
R: difficulty joining relevant meeting channels for discussion due to visual clutter

What can make communication regarding issues via TCTs more efficient?

G: direct gathering of analytics from the frontend, automatic sending of issues to address sent to the backend team i.e. feedback hotline
R: reduce need to message and/or wait for meetings to bring up the issue, assign an issue between chosen people so it's easier for them to communicate

What has your experience been with issue resolution through the TCTs?

G: regular report of bug fixes in every software update
R: would like something that keeps track of resolved issues

In which aspects do you think would improve the sense of connection between team members?

G: see if message has been sent, is sitting in the recipient's notification centre, read or ghosted
R: online status indicator, ability to bring attention to urgent matters eg. urgent questions tab

From the responses, the main problems the target users had with TCTs were:

- Difficulty navigating and finding features on the TCT: visual clutter, too many clicks/taps
- Mediocre issue communication: issues tab for easier issue discussion + contacts
- Difficulty tracking of issues: Issues board - stores history and result of an issue
- Improved raising of issues desired: feedback hotline - streamline issue raising, urgent questions tab
- Lack of sense of connectedness, knowing if communication was worth the effort: read receipts, online indicator

Responses relating to software limitations and more frontend-UI design specific issues have less priority in the problem resolution stack as they are more data management or even business related. We primarily want to focus on improving communication, issue resolution, and encouraging collaboration as these are the main purposes of TCTs.

To resolve the problems raised by the target users, we propose an issue tracking system that includes features such as facilitating more targeted discussion, stores the history of the issue and/or similar issues, and enables users to have the ability to directly reach a team member about an issue. The sense of teamwork and collaboration that is often lost through digital communication can be boosted by implementing features that improve the online presence of teammates, such as online status indicators and read receipts that describe if communication has reached the teammate.

Requirements: Analysis and Specification - User Stories & Use Cases

User Story 1

As a team member, I want to keep track of resolved issues so that I can refer back to potential solutions of recurring problems.

User Acceptance Criteria:
- There is a feature page titled “Issue History”
- There is a button on the side bar labelled with “Issue History”
- Clicking on the button navigates the user to the Issue History page
- Conversations of resolved issues can be flagged to go onto the Issue History page
- Conversations of unresolved issues cannot be flagged to go onto the Issue History page
- The Issue History page contains flagged conversations

User Story 2

As a team member, I want to see if my fellow teammates are online so that I know when I’ll be more likely to get a response from them.

User Acceptance Criteria:
- There is a bar that goes along the top of the page containing circles to represent team members
- The user can choose which team members the circles should represent
- Hovering over a circle will display if the team member is online or not
- Moving the cursor away from the circle causes the display to disappear

User Story 3

As a team member, I want to see if my messages have been read so that I have an idea of the progress of communication.

User Acceptance Criteria:

- The user can send messages to other users via channels and DMs
- The recipient can open the channel/DM the message was sent to
- The system notes that the recipient successfully received the message
- The confirmation will be recorded
- The record of the confirmation will be displayed as a read receipt
- The sender will be able to view the read receipt
- If the recipient doesn’t open the channel/dm, a receipt doesn’t get sent

User Story 4

As a programmer, I want to be able to find current problems easily so that I can solve them quickly.

User Acceptance Criteria:
- There is a feature page titled “Issues”
- There is a button in the bottom left corner of the TCT

- Clicking on the button leads the person to the Issues page
- The programmer can view ongoing issues on the Issues page
- New issues can be flagged to go onto the Issues page
- Issues on the Issues page can be ordered in terms of how recently they were flagged

User Story 5

As a team leader, I want my team to see how allocated tasks are progressing so that everyone can see how the team is progressing.

User Acceptance Criteria:
- Team members can create a task, title it, and assign it to a team member
- Tasks can be labelled in terms of progress
- Tasks can be updated by the corresponding team member
- All tasks can be viewed by all team members

Use Case 1 - corresponds to User Story 3

Use case: See read receipt of sent message

Goal in context: A user can see the read receipt of a message they sent to another user

Scope: UNSW Treats

Preconditions: The user has registered an account, is logged in, and is in a channel/DM with another user

Success End Condition: The user can see the read receipt of the message they sent

Failed End Condition: The user cannot see the read receipt of the message they sent or can see a read receipt indicating the wrong flag

Primary Actor: User

Trigger: A user sends a message to another user

Success scenario:

1. The user sends a message to a team member
2. The message is sent to the channel/DM of the recipient
3. The recipient receives the message
4. The message is flagged as “Not Read”
5. The recipient opens the channel/DM with the message
6. The message is flagged as “Read”
7. The sender sees the read receipt of the message indicating “Read”

Use Case 2 - corresponds to User Story 4

Use case: Flag messages as issue

Goal in context: A user can flag a selection of messages in a channel as an issue

Scope: UNSW Treats

Preconditions: The user has registered an account and is logged in. They are also in a channel with other users and have been discussing an issue.

Success End Condition: The user can flag a selection of messages in a channel as an issue and can see the issue displayed on the “Issues” page of that channel

Failed End Condition: The user cannot flag a selection of messages in a channel as an issue and cannot see the flagged issue on the “Issues” page of that channel

Primary Actor: User

Trigger: A user selects messages in the channel and selects the option to flag them as an ongoing issue

Success scenario:

1. The user selects messages in the channel to be bundled into an issue
2. The user flags the bundle of messages as an “Ongoing Issue”
3. The bundle of messages is added to an “Issues” page in the channel
4. The user navigates to the “Issues” page
5. UNSW Treats displays the new issue along with the rest of the ongoing issues, ordered by how recently they were added
6. When the issue is resolved, the user flags the issue as “Finished”
7. The finished issue is hidden from view on the “Issues” page

Requirements: Validation

Use Case 1

G: That seems about right, means that I can see where the message and the recipient are at. Should be good for when I have an urgent question to ask a buddy or just want to see if someone’s online so we can work on the project together.

Use Case 2

R: That’s good, having a separate section where ongoing issues can be seen in one go would help me to gauge progress on a project we’re working on. Though, I think it would be good to be able to see finished issues in case I want to refer back to them.

Design: Interface Design

Use Case 1

Name & Description	**HTTP Method**	**Data Types**	**Exceptions**

/status/changereadreceipt
 	
Given a uId a channelId | dmId, changes the read receipts of all messages from false to true when the user opens a channel/dm.
	PUT	Parameters:
		
		{uId, channelId | dmId}
		
		Return type if no error:
		
		{}
			400 Error when any of:
			
			- token is invalid
			- uId does not refer to a valid uId

/status/viewreadreceipt
 	
Give a messageId and uId, returns whether the message has been read by the user.
 	
The message object will have another key-value, hasBeenRead: boolean, which is true if the message has been read, and false if the message has not been read. This can be updated from false to true when the user opens the channel/DM of the new message, and cannot be changed back to false.
	GET	Parameters:
		
		{uId, messageId}
		
		Returns:
		
		{boolean}
			400 Error when any of:
			- token is invalid
			- messageId does not refer to a valid messageId
			- uId does not refer to a valid uId
			
			403 Error when any of:
			- messageId is valid but token user is not part of the DM or channel
			- uId is valid but token user is not part of the DM or channel

Use Case 2

code(1):
```
let data: dataBase = {
        Users: []
        Channels: []
        DM: []
        Tokens: []
        Issues: []
}
```

code(2):
```
interface conversation {
        conversation Id: number
        messages: message[],
        timeBundled: number,
        ongoing: boolean,
}
```

Name & Description	**HTTP Method**	**Data Types**	**Exceptions**

/issue/bundle
 	
Give a messageId(s) returns the time that the messages have been bundled into a conversation.
 	
The selected messages with the valid messageId's will be pushed to the messages array in a new object conversation, refer to code(2), timeBundled will be updated with the current time, and conversationId will be updated - it is unique.
 	
The object conversation is then pushed to the issues[] array in the dataBase, refer to above code(1) for the updated dataBase, in order of timeBundled.
	POST	Parameters:
		{messageId[]}
		
		Returns:
		{timeBundled}
			400 Error when any of:
			- invalid token
			- a messageId does not refer to a valid messageId
			
			403 Error when:
			- messageId is valid but token user is not part of the DM or channel

/issue/flagongoing
 	
Given a conversationId, changes “ongoing” to true, to mean an ongoing issue. This should mean that it is visible on the “Issues” page
	PUT	Parameters:
		{conversationId}
		
		Returns:
		
		{}
			400 Error when any of:
			- invalid token
			- conversationId does not refer to a valid conversationId
			
			403 Error when any of:
			- messageId is valid but token user is not part of the DM or channel
			- conversationId is valid but token user is not part of the DM or channel

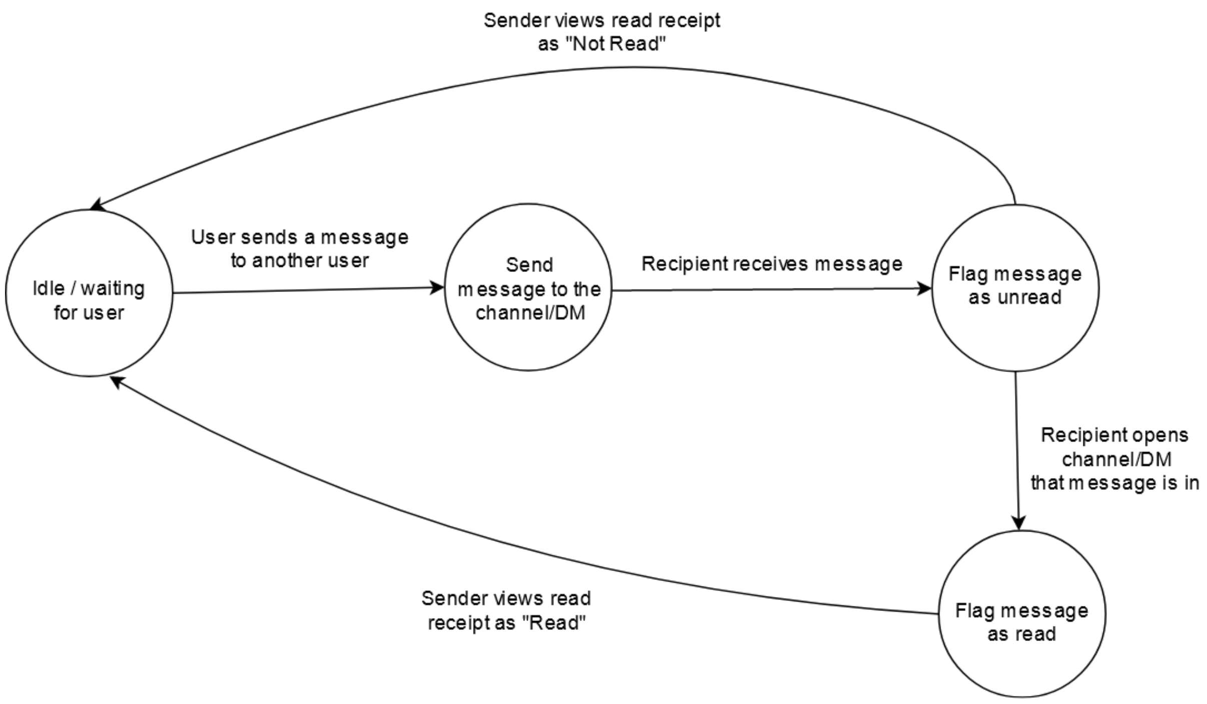
/issue/flagfinished
 	
Given a conversationId, changes “ongoing” to false, to mean a finished issue. This should mean that it is not visible on the “Issues” page
	PUT	Parameters:
		{conversationId}
		
		Returns:
		
		{}
			400 Error when any of:
			- invalid token
			- conversationId does not refer to a valid conversationId
			
			403 Error when:
			- conversationId is valid but token user is not part of the DM or channel

/issue/viewfinished
 	
Returns an array of conversation retrieved from the “Issues” page for which their status is “finished”.
	GET	Parameters:
		
		{}
		
		Returns:
		{conversations}
			400 Error when:
			- invalid token

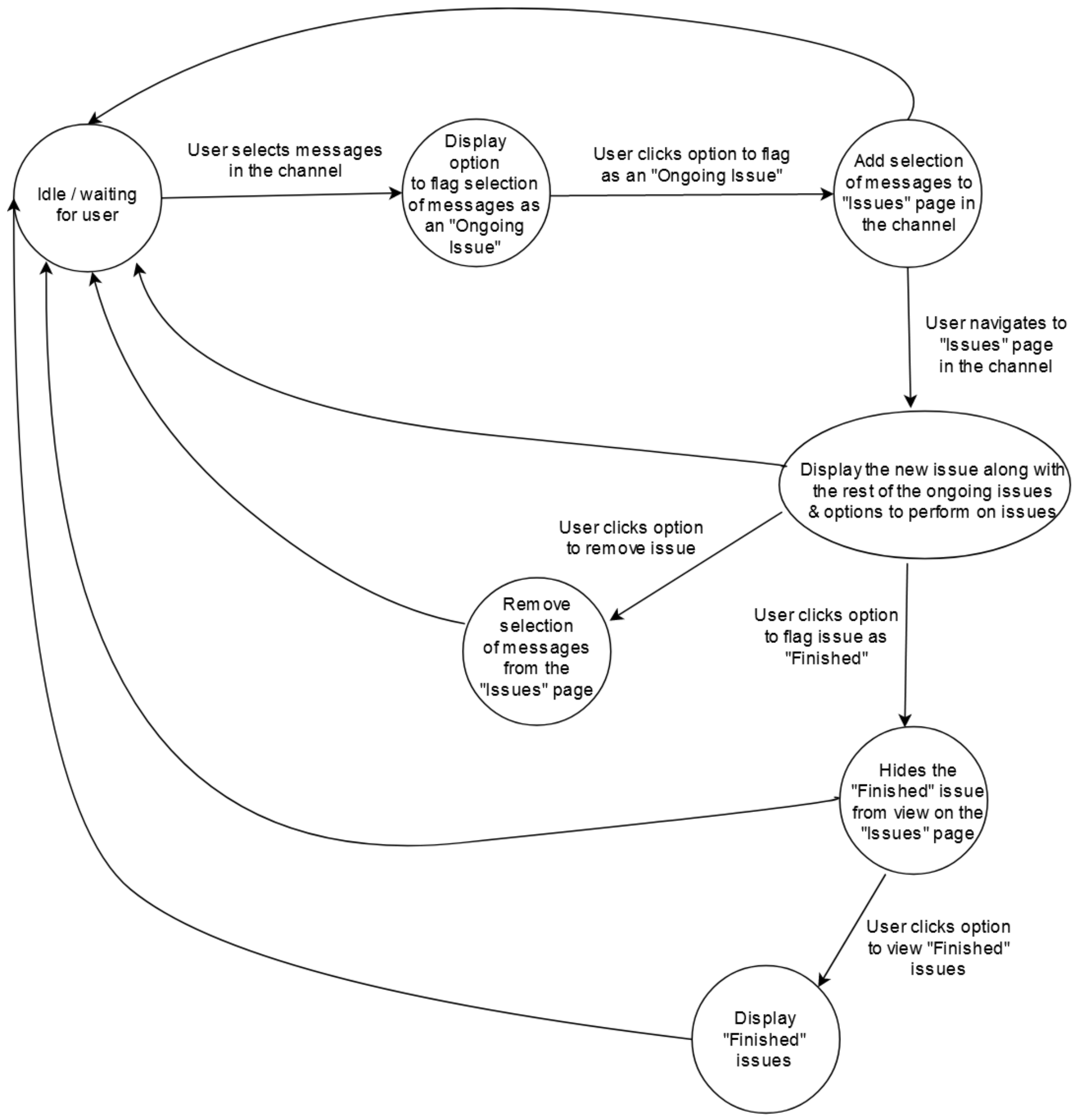
/issue/remove
 	
Given a conversationId, removes the conversation from the issues array.
	DELETE	Parameters:
		{conversationId}
		
		Returns:
		
		{}
			400 Error when any of:
			- invalid token
			- conversationId does not refer to a valid conversationId
			
			403 Error when:
			- conversationId is valid but token user is not a global owner

<u>**Design: Conceptual Modelling - State Diagrams**</u>

<u>Use Case 1</u>


Sender views read receipt as "Not Read"
Idle / waiting for user
User sends a message to another user
Send message to the channel/DM
Recipient receives message
Flag message as unread
Recipient opens channel/DM that message is in
Flag message as read
Sender views read receipt as "Read"

<u>Use Case 2</u>


Idle / waiting for user
User selects messages in the channel
Display option to flag selection of messages as an "Ongoing Issue"
User clicks option to flag as an "Ongoing Issue"
Add selection of messages to "Issues" page in the channel
User navigates to "Issues" page in the channel
Display the new issue along with the rest of the ongoing issues & options to perform on issues
User clicks option to remove issue
Remove selection of messages from the "Issues" page
User clicks option to flag issue as "Finished"
Hides the "Finished" issue from view on the "Issues" page
User clicks option to view "Finished" issues
Display "Finished" issues