ezConnect

NUS Orbital 2023
Milestone 3
Artemis

Proof of Concept

Webapp Link: https://ezconnect.ruibin.me/
Instructions: Create an account using your NUS email. You can then log in with your NUS email.

Aim

ezConnect aims to be a consolidated web app dedicated to helping NUS students improve their university experience. ezConnect allows students to seek help from others through mentor-mentee matching and sharing of study plans. Students no longer have to search through numerous Reddit posts or join multiple telegram channels for those purposes. We hope that students will take a shorter time and be successful in finding students for different purposes.

Motivation

Have you ever found it difficult to find someone to seek university advice from due to a small social circle? Are you too shy to ask on Telegram or Reddit but want to ask questions or share resources? Are you having difficulties planning the courses to take over your university life?

University is not easy to navigate and many have found it difficult to seek help from other students. Students want to be able to exchange help and resources with one another. Although there are telegram group chats to facilitate communications between students, the high number of telegram group chats meant for different purposes leads to overflooded telegram chats, making it difficult to communicate effectively. Reddit threads are often repeated and lack responses. Therefore, we wanted to create ezConnect to make it easier for students to connect and share study plans and advice more easily.

User Stories

1. As an academically weak student who wants help with course content, I want to be able to find a mentor.

2. As an academically strong student and as a student who is passionate about helping other students, I want to mentor other students.
3. As a mentee, I want to share my situation so that I can find a compatible mentor.
4. As a mentee/mentor, I want to be able to get the contact information of my mentor/mentee to start mentoring.
5. As a freshman and a confused student who is unsure of how to plan my modules, I want to be able to look at other students' study plans and their rationale, experiences and advice about their study plan.
6. As a user, I want to be able to find study plans for programmes that I am considering.
7. As a user who wants to reference study plans, I want to be able to store study plans that I would like to reference.
8. As a student uncertain about my study plan, I would like to check that I am fulfilling the prerequisites for the modules that I intend to take.
9. As a user of NUSMods, I want to be able to export and import semesters into the study plan from NUSMods.
10. As a user, I want to feel safe and know that other users are real NUS students.

Key Terms

There are some key terms that we are using throughout our features. As different people may have different interpretations of these terms, we will be defining them here for greater clarity.

- **Academic Plan**: The list of degree(s) and programme(s) that a student intends to take.
- **Course**: A course provided by NUS (previously known as a module). Examples include CS1101S and GEA1000.
- **Degree**: The degree that students graduate with.
- **Study Plan**: A collection of courses that are taken by a student over his/her time in university. These courses are split into different semesters that the student will take them in.
- **Programme**: Second majors or minors that students have or special programmes that students are involved in, such as University Town College Programme and Special Programme in Science.
- **Unit**: Units (previously known as modular credit (MC)) are obtained from completing courses. Each course awards a fixed number of units.

Proof of Concept **1**
Aim **3**
Motivation **3**
User Stories **3**
Key Terms **4**
Features **5**
User Account Authentication; sign up and sign in with NUS 5
Introduction and Milestone 1 implementation: 5
Switch to Azure Active Directory B2C from Milestone 2 onwards: 6
Blocker / Problem faced: Permission denied in Azure 6
Enabling NUS Login without any special request to the NUS side administratively 7
Usage of Azure AD B2C 10
Problems faced: Wrong redirect URL set and lack of documentation 14
Sign up / Sign in with Microsoft 15
User creation 18
Restricted pages 19
About Us 20
Homepage 22
Mentor-Mentee Matcher 23
Mentoring Main Page 23
Mainpage screenshots 24
Creating posts / requests and matching 26
Potential Expansions 31
Study Plans 32
Personal Study Plan Gallery 32
Study Plan Gallery 34
Study Plan Post 36
Search, Order by and Filter 38
Study Plan Editor 41
Study Plan Validator 49
Implementation and challenges 49
User interface 52
Potential improvements 53
Navigation Flow **54**
Tech stack **55**
Architecture Diagram 56
Entity Relationship Diagram **57**
API Documentation **59**
Software Engineering Practices **62**
Don’t Repeat Yourself Principle 62
Reuse 62

Code Organisation & Monorepo design & Separation of concern 63
Project documentation 64
Version Control: Branching, pull requests and code reviews 66
Pull requests and code reviews 67
Git Issues, Projects board and Sprint Planning 70
DevOps: Continuous Integration and Continuous Delivery/Deployment 74
Containerisation / virtual environments 74
Integration and Unit Testing with pytest, ruff, GitHub Actions and playwright 75
User Testing 83
Dogfooding 83
Limited Usability Testing from milestone 2 to milestone 3 84
Usability survey for milestone 3 85
Design of survey 85
Findings and subsequent updates 89
Limitations 93
Lack of availability of programme data 93
Reliance on NUSMods Data 93
Timeline and Development Plan 94
Wireframe 97
Posters 99
Video Links 104

Features

User Account Authentication; sign up and sign in with NUS

User stories fulfilled: 10

Introduction and Milestone 1 implementation:

One of our key requirements for this application is limiting users to NUS students. Initially, for milestone 1 we kept things simple by just implementing our own account feature in the app. We used JSON Web Tokens (JWT) to secure our backend APIs. We did not go with cookies as it will introduce additional complexity by necessitating infrastructure on the server side to manage the login state. Additionally, our backend library (Connexion) did not have on its documentation an obvious way to support cookie authentication. Lastly, because of our app architecture with a separate frontend react app and an API backend, we felt that JWT are more appropriate in a single-page application context[1].

Our initial proof of concept required keeping a user's password_hash and salt

We also kept in mind that JWT may allow us to enable login using the OAuth protocol in the future. Initially, we were deterred by the information that access to NUS APIs for login requires an application form and there might be some delays in getting this access.[2] This was another push factor for implementing users on our own. However, we have another idea in mind that we want to experiment with that would enable us to log in with NUS but we faced a blocker initially, more on this later.

Hence for milestone 1, our server generates an email notification and a sign-up request when a user first signs up for an account. We verify that the email is an NUS email (ending with 'u.nus.edu') and only carry on if it meets this requirement. Once the user is verified we create the account, storing a salted hash of the user password as well. When they log in, they are given a JWT that is stored for the remainder of the session. JWTs are generated with a very short lifespan so they expire after 1 hour, making even a leak of the token useless for attackers.

[1] https://developer.okta.com/blog/2022/02/08/cookies-vs-tokens#when-to-use-cookies-or-tokens
https://stackoverflow.com/questions/37582444/jwt-vs-cookies-for-token-based-authentication

[2] https://teams.microsoft.com/l/message/19:3aad7997241d4a879983b256cbafb9e9@thread.tacv2/1684735300534?groupId=42f4de8b-c101-4d45-9937-560b64bdf81b&parentMessageId=1684735300534&tenantId=5ba5ef5e-3109-4e77-85bd-cfeb0d347e82

The use of this meant the frontend needed application code and logic to keep track of the state of user sign-in and their current token. Login is done by sending an email address and password to the server which returns a token which is then tracked by the frontend logic for use in future requests to protected endpoints. Still, we felt that we have met several best practices for storing passwords and the email verification step is good enough for our application.

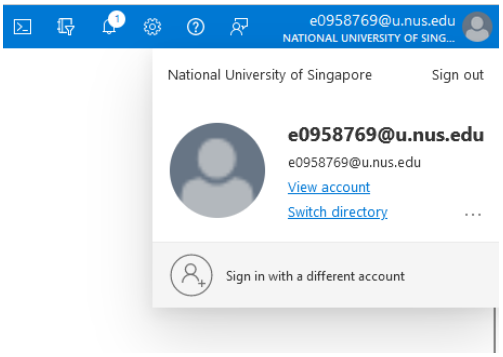
Switch to Azure Active Directory B2C from Milestone 2 onwards:

From our past experience working on applications and authentication, we are aware of the industry standard OAuth 2.0 and OpenID Connect. OAuth 2.0 and OpenID Connect is the protocol various websites and applications use to implement social login, i.e. sign-in with Google, etc.

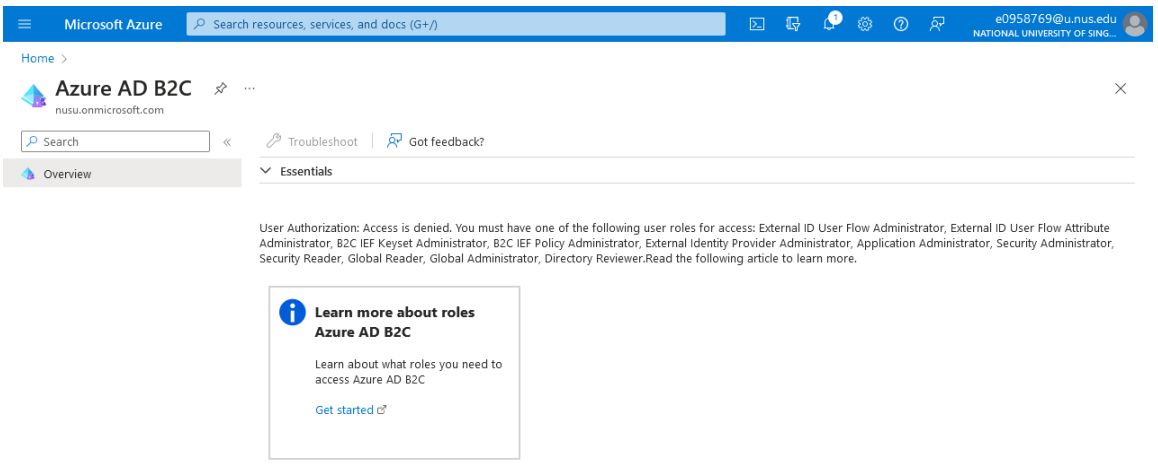
From the start of the project, what we had in mind was to implement a login with the Microsoft option and check that the organisation the user came from was NUS. However we faced a blocker signing up for this service:

Blocker / Problem faced: Permission denied in Azure

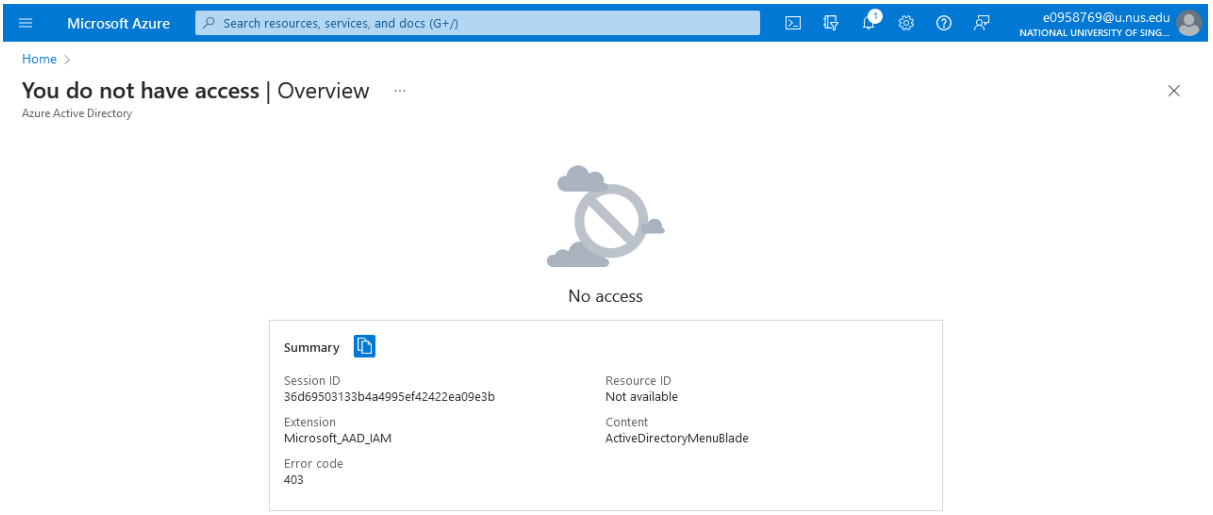
As our Azure for student subscriptions are linked to our school microsoft account, our default directory in Azure is NUS's Azure Active Directory. This led us to believe we are not authorised to work with Azure Active Directory or B2C.



Directory of our Azure User



Error message in Azure AD B2C

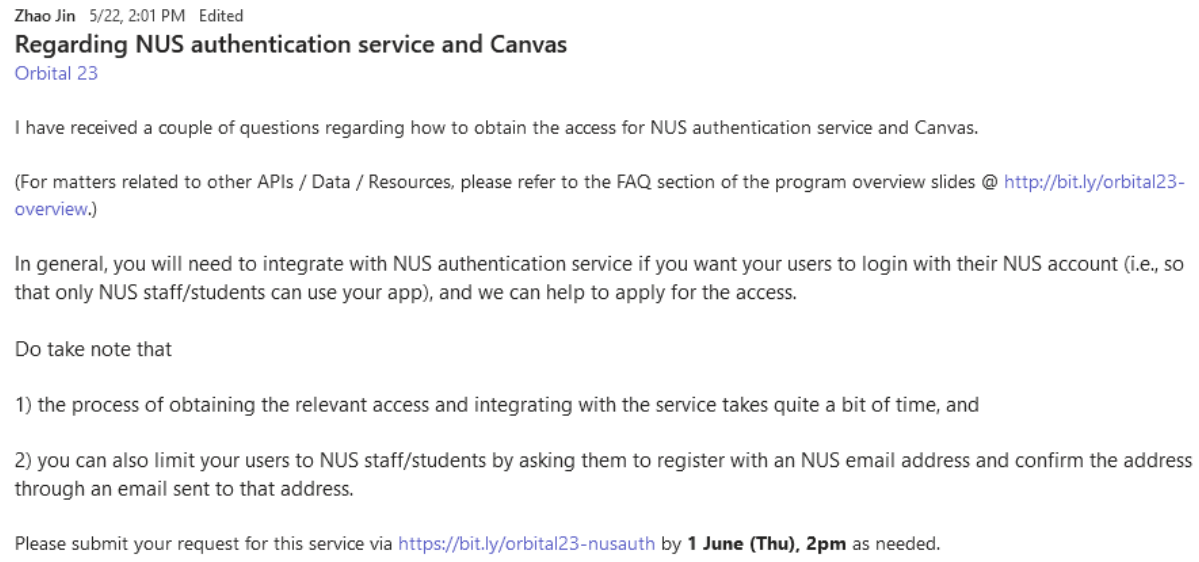


Error message in accessing Active Directory

Eventually our mentor mentioned he has faced similar issues in the past and recommended a different approach by using a service for creating Azure AD B2C tenants from the marketplace. However, we are still facing the same issue and were about to give up on this pathway when I realised that we can transfer the subscription to an existing Azure AD tenant one of the developers had; *ruibin.me*. The Azure AD B2C tenant is then created there, outside of NUS's directory.

Enabling NUS Login without any special request to the NUS side administratively

But what about creating a request for the NUS Authentication service?



Zhao Jin 5/22, 2:01 PM Edited

Regarding NUS authentication service and Canvas

Orbital 23

I have received a couple of questions regarding how to obtain the access for NUS authentication service and Canvas.

(For matters related to other APIs / Data / Resources, please refer to the FAQ section of the program overview slides @ http://bit.ly/orbital23-overview.)

In general, you will need to integrate with NUS authentication service if you want your users to login with their NUS account (i.e., so that only NUS staff/students can use your app), and we can help to apply for the access.

Do take note that

1) the process of obtaining the relevant access and integrating with the service takes quite a bit of time, and

2) you can also limit your users to NUS staff/students by asking them to register with an NUS email address and confirm the address through an email sent to that address.

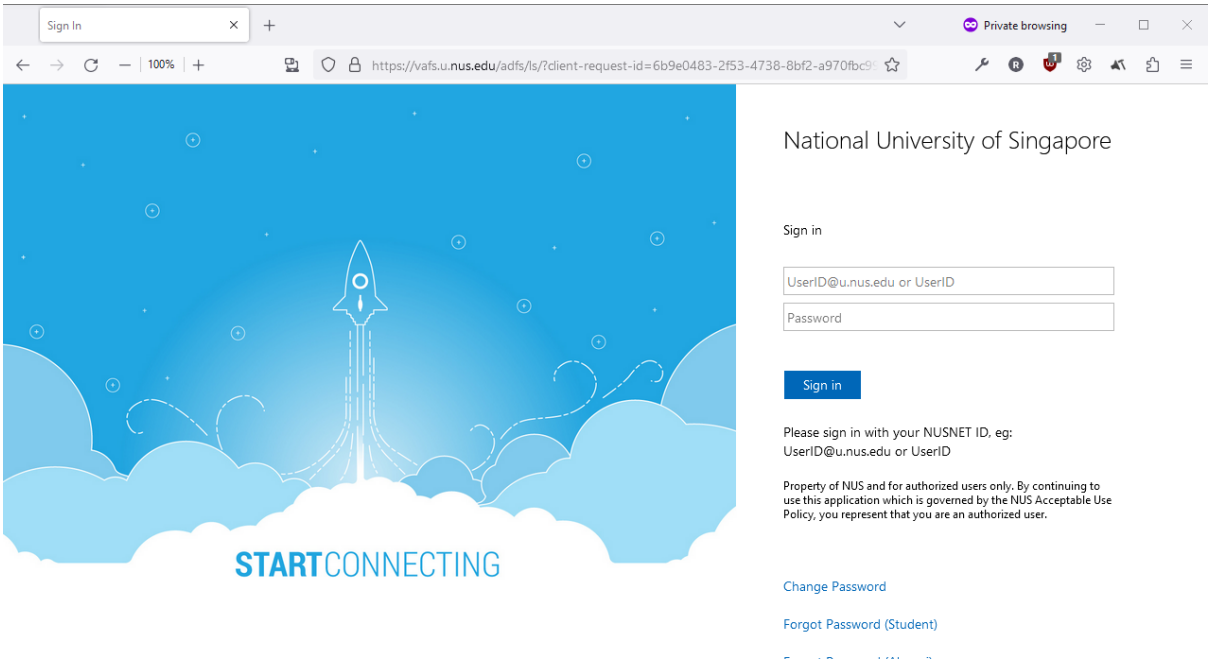
Please submit your request for this service via https://bit.ly/orbital23-nusauth by **1 June (Thu), 2pm** as needed.

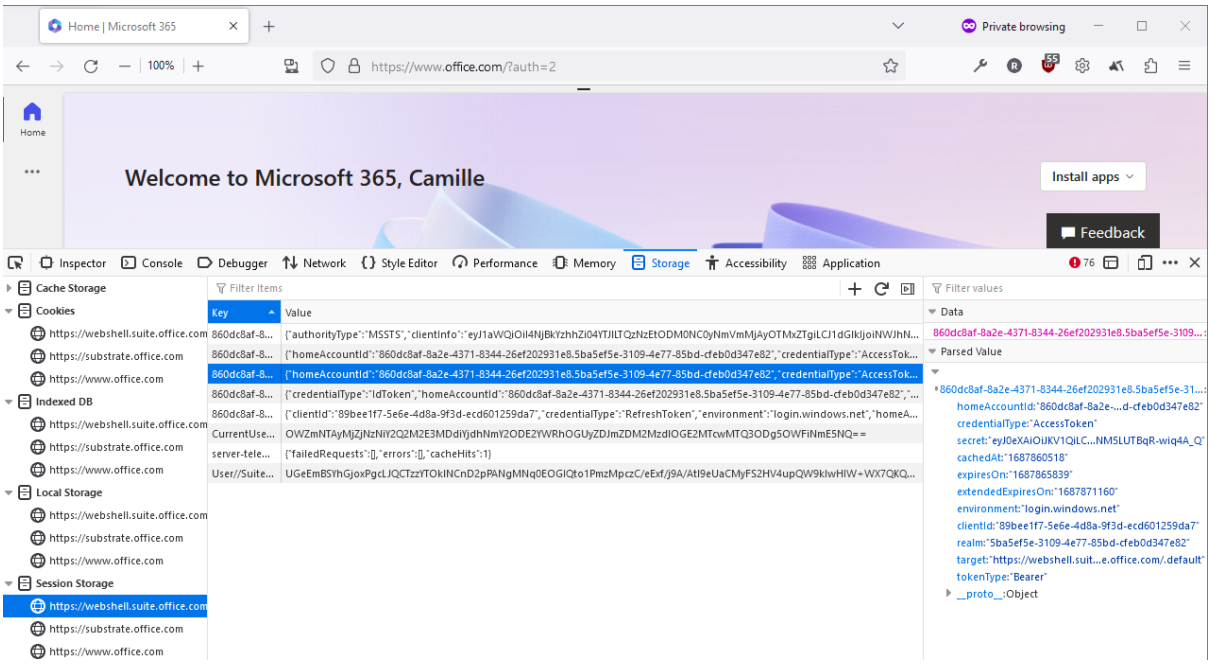
Message in Orbital Teams Channel regarding NUS Authentication

From the information, it seemed as though the team needed to apply for this service. However, the team did not apply for this service as we wanted to carry on with our project in case it takes too long since it "takes quite a bit of time" and we're worried how long it might take to integrate it. Additionally, we also do not want to rely on something that may be rejected.

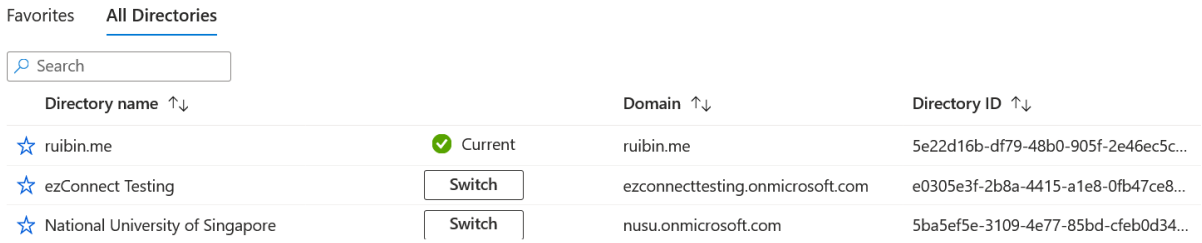
Astute students may realise that NUS uses a directory service, hence some login forms need a 'nusstu\' and 'nusstf\' in front of your account name. The 'nusstu\' is actually a form of indicating your organisation unit in the directory. You can search 'Active Directory' for more information. We know that NUS is an existing tenant on Azure Active Directory[3] since we have access to various Microsoft services like Office, teams and the obvious fact that **our default directory on Azure Portal is NUS**. If you have ever paid attention to NUS login you might be curious about all the redirects and parameters in the URL.

[3] https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis





Realm here is the UUID of NUS's Directory. A more obvious way to find this UUID and the tenant name/domain is simply in Azure Portal itself, where it shows your directories.



Usage of Azure AD B2C

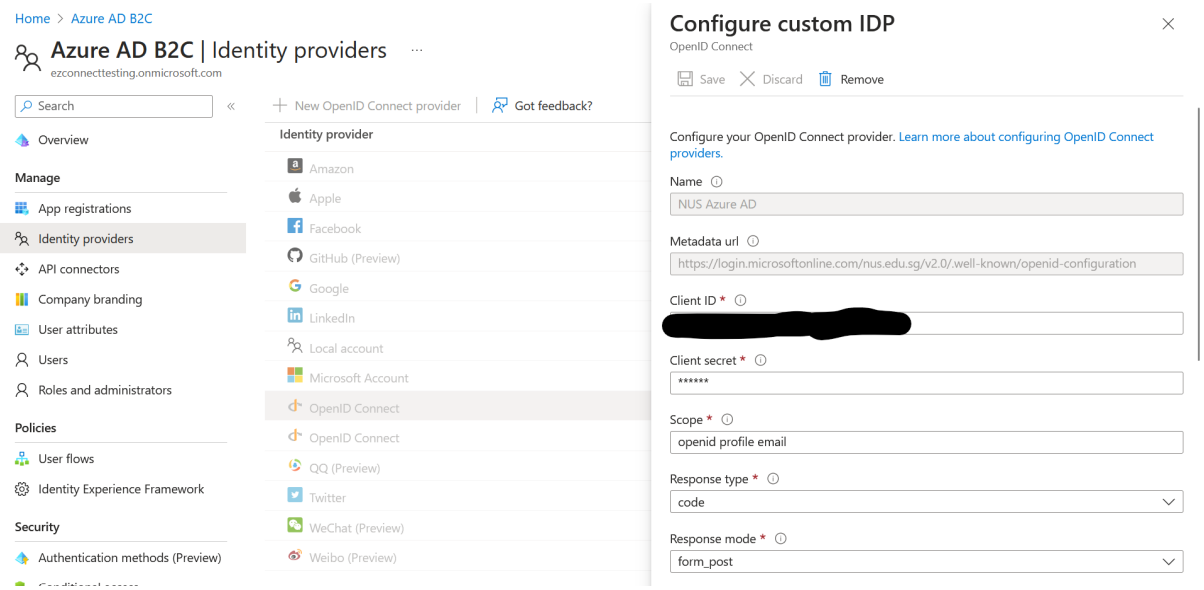
Azure AD B2C allows applications/organisations to create identities for end users without managing the nitty gritty part of authentication and authorisation. Put simply, it is a middleware for managing identity in your application.

The service allows for integration with OpenID Connect, which is supported by Google and other social media services. In fact, OpenID Connect is what powers the “Sign in with” function seen on many websites and services. Basically, the server as the identity provider, is user X so and so? And the identity provider, which can be Google or any other provider, will reply with an id token once User X verified with the identity provider themselves, how User X authenticates themselves to their identity provider is not relevant to us.

With this in mind, our group simply decided to configure NUS’s Azure Active Directory *as* the identity provider. We originally tried to use the built in Microsoft account identity provided but that is restricted to purely personal / social accounts, example being outlook emails or Xbox accounts. The key difference is a bit technical but it is because they are technically under a separate tenant. We managed to find the documentation for setting up a particular tenant / organisation as the identity provider.

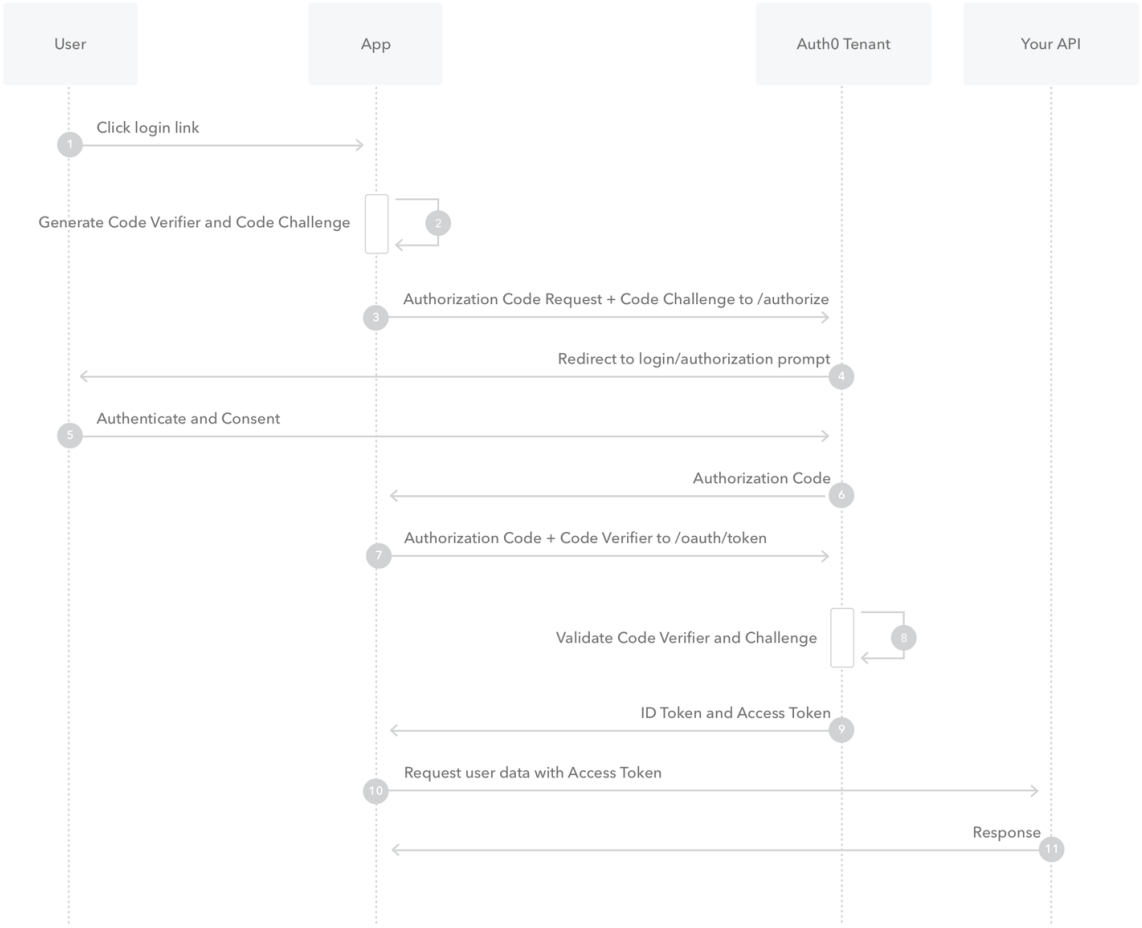
To do this, we need the OpenID configuration file for NUS. Luckily all Azure tenants follow a standard format and we identify the pattern and located NUS's at: https://login.microsoftonline.com/nus.edu.sg/v2.0/.well-known/openid-configuration or https://login.microsoftonline.com/nusu.onmicrosoft.com/v2.0/.well-known/openid-configuration

The endpoints are the same, you may realise that the domain names are actually aliases for the tenant UUID.



Identity provider configuration for NUS on Azure AD B2C

We use authorisation code flow with Proof Key for Code Exchange (PKCE) for our application. Authorisation code flow with PKCE is the recommended way for adding authentication to a single page web application, or public client. Since our frontend is in react and it is basically a public client as the source code can be viewed, and hence we should not store any client secrets in the frontend as this can be easily extracted. Authorisation code flow with PKCE is the recommended flow to use as it prevents man in the middle attack or code interception by making sure the token is returned to the client that is calling it.

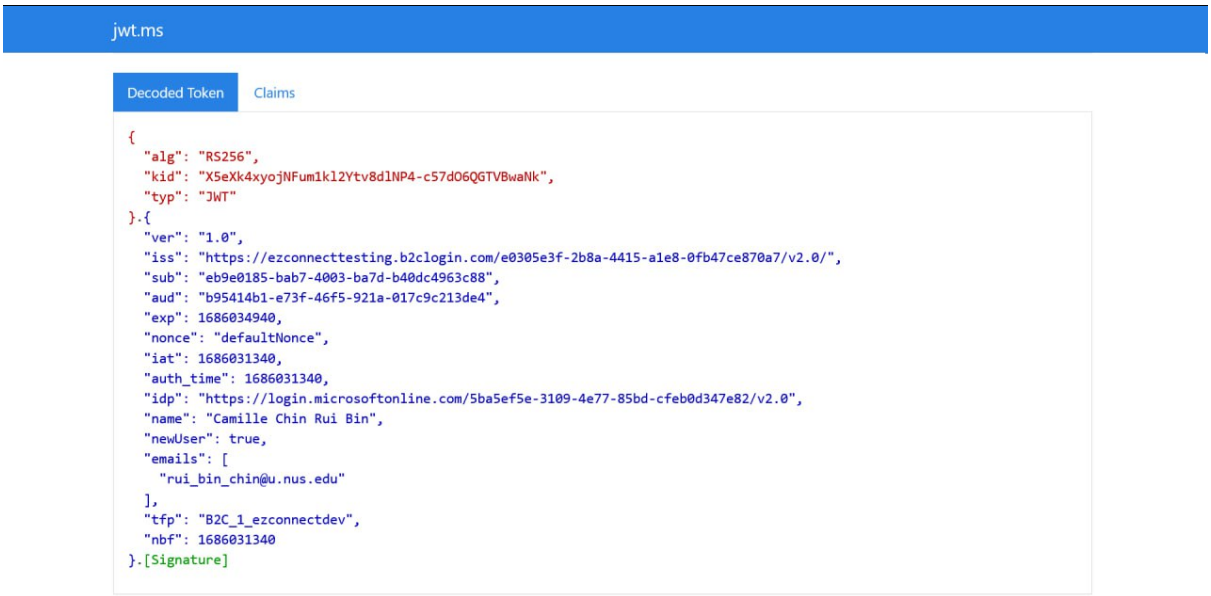


Auth0 explanation of Auth Code flow with PKCE
https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-proof-key-for-code-exchange-pkce

In the frontend this is managed using the MSAL react and MSAL browser libraries. These libraries will handle storing the login state and other user data that is related to interfacing with Microsoft Authentication. When the user logs in, an ID token and Access Token is stored and managed by MSAL. When fetching from a secure API, we append the access token to the Authorisation header, similar to what we do with JWT earlier. The ID token is used to exchange for access tokens when access tokens expire.

In the backend we have to verify the JWT access tokens given to us by the frontend is a valid cryptographically signed token. To do this we have to retrieve the public key which is stored here, https://ezconnecttesting.b2clogin.com/ezconnecttesting.onmicrosoft.com/b2c_1_ezconnectdev_susi/discovery/v2.0/keys. If we can decrypt the token with the public key, it means the token is signed by the private key held by Microsoft, meaning the token is legitimate. The next step is to verify the fields within the token like “exp’, ‘iat’, ‘scp’, ‘nbf’ which contains time for expiry, issued at, the user allowed scope, and lastly not before time.

As we have already committed to using JWT from the start, the infrastructure was already in place to allow us to use Azure AD B2C to allow users to sign in with their NUS microsoft account. This foresight allowed us to greatly save time refactoring the rest of the application as we only need to update the verifying token logic. We can also strip out the moving parts from sending verification emails, password hashing, and etc.

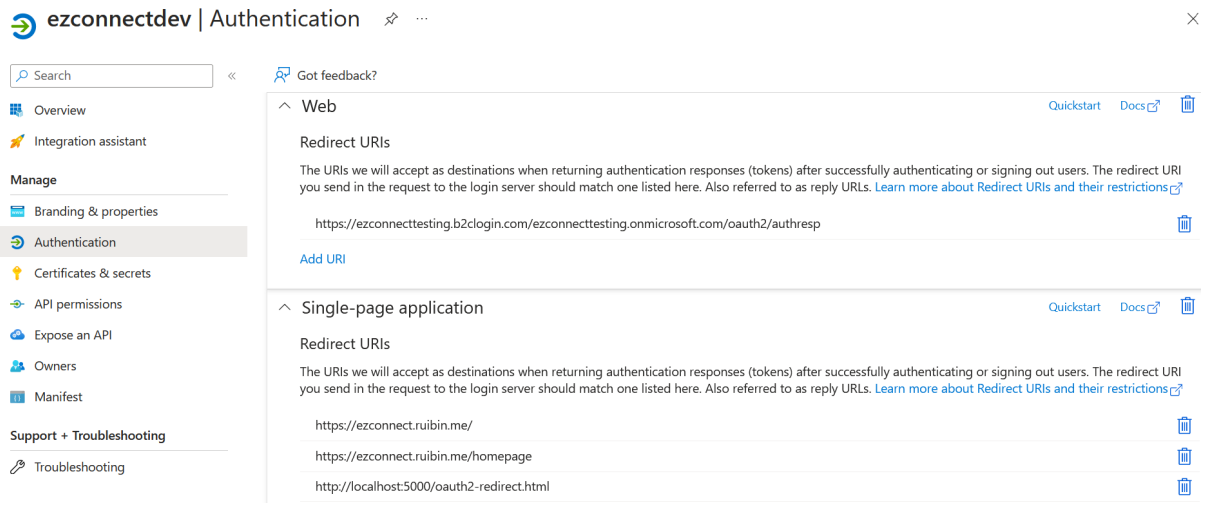


A decoded id_token

Problems faced: Wrong redirect URL set and lack of documentation

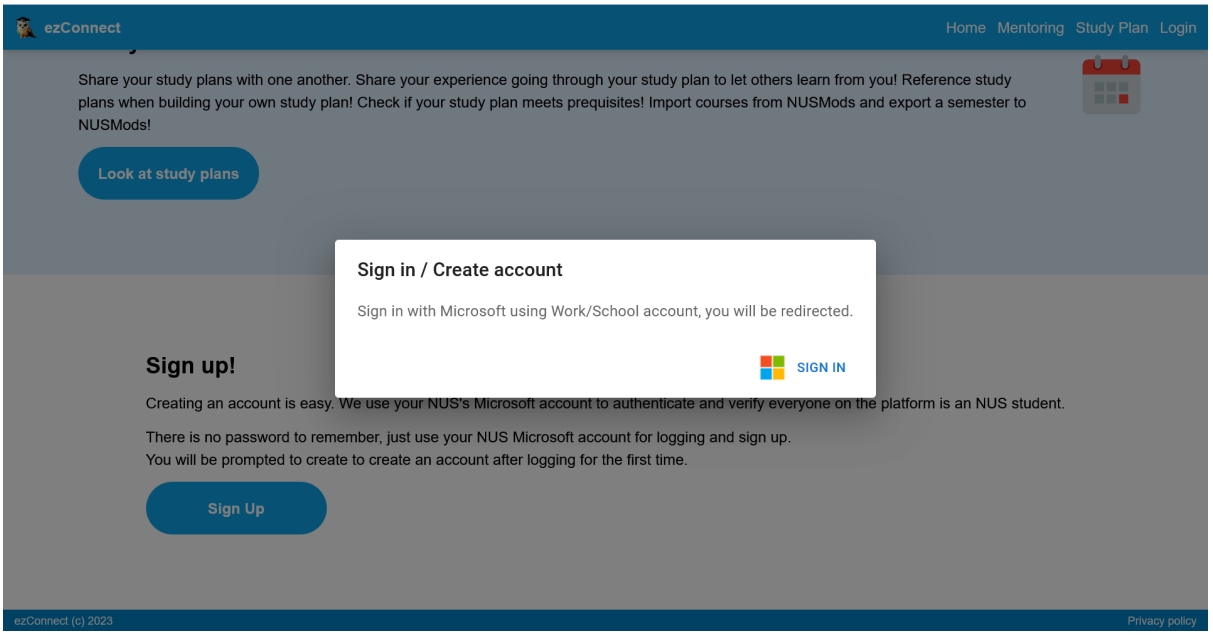
Microsoft Azure is a very big ecosystem and without knowing what to look for it can be confusing and easy to get lost in the labyrinth of documentations. An example of this was when we first configure NUS's IDP, we set the redirect URL that is needed as "Web", however when setting up our frontend, we were told to use "Single-page application". During the debugging we came across various solutions which say to make sure that *all* redirect URLs are Single-page applications, hence the web url was also changed. This broke signing in with NUS Microsoft. We then swapped everything to use Web plus a myriad of other configuration permutations to no avail. Eventually we decided to retrace our steps and read through every instruction one by one and settled that the original url for the IDP needs to stay as web. We were not the only ones to find this problem but when we solved it the team contributed a reply on Microsoft's forum. The other stackoverflow answers said to change all urls as their configuration did not need to have a web url.

[How to fix AADSTS9002325: Proof Key for Code Exchange is required for cross-origin authorization code redemption error in msal react application. - Microsoft Q&A]



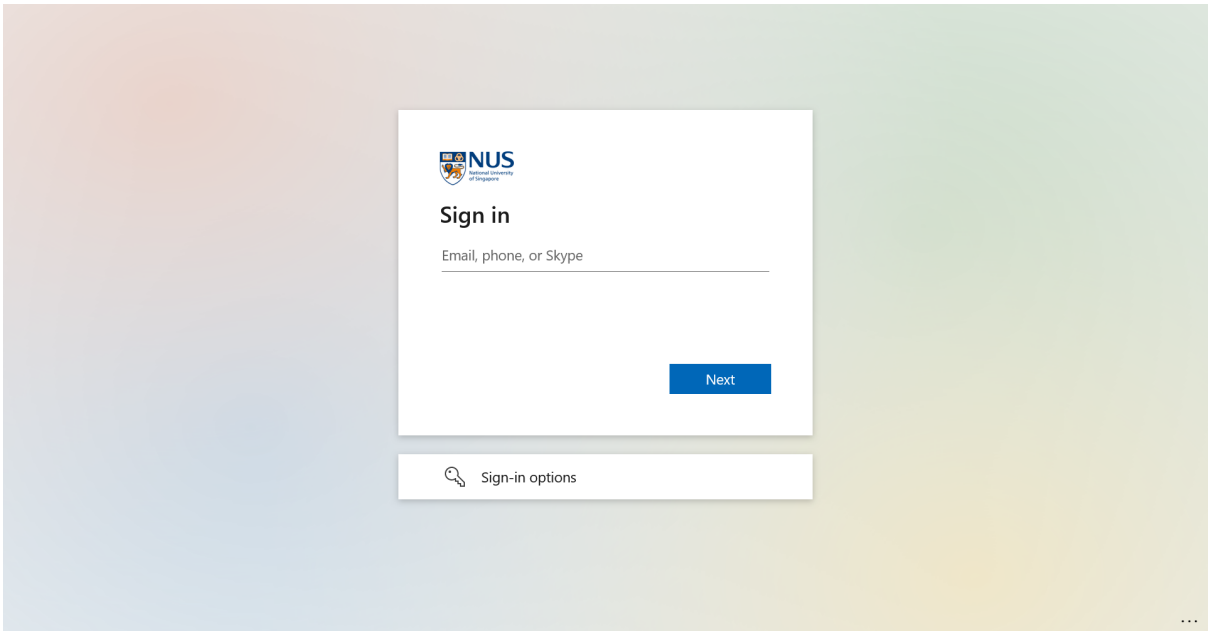
Another issue we have is finding documentation on how to validate the access token on a python backend. There are libraries provided for other languages on validating the tokens but none were readily available in Python. We had to read one of the provided code samples to understand what they did and how they verified the token. Hence setting up this feature took much longer than initially estimated due to confusing documentation and changes.

Sign up / Sign in with Microsoft

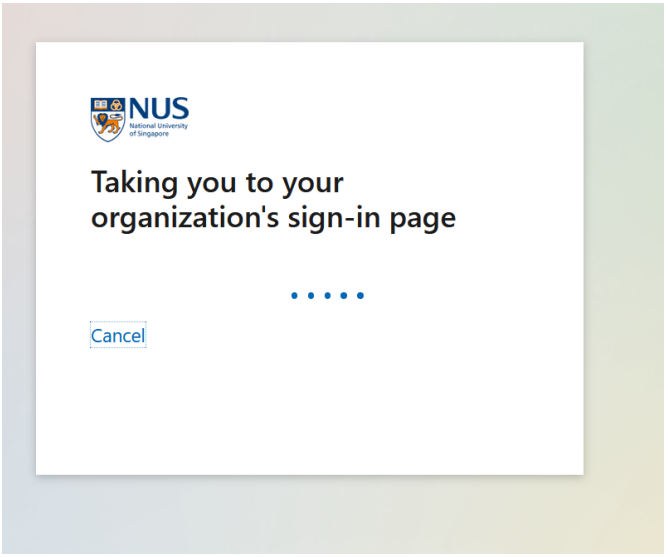


Sign in / create account pop up

When a user presses login on the navbar or “Sign Up” in the about us page, they activate the login modal which shows as a sign in / create account dialogue box. There when they click “Sign In” they will be redirected to Microsoft’s page for signing in.

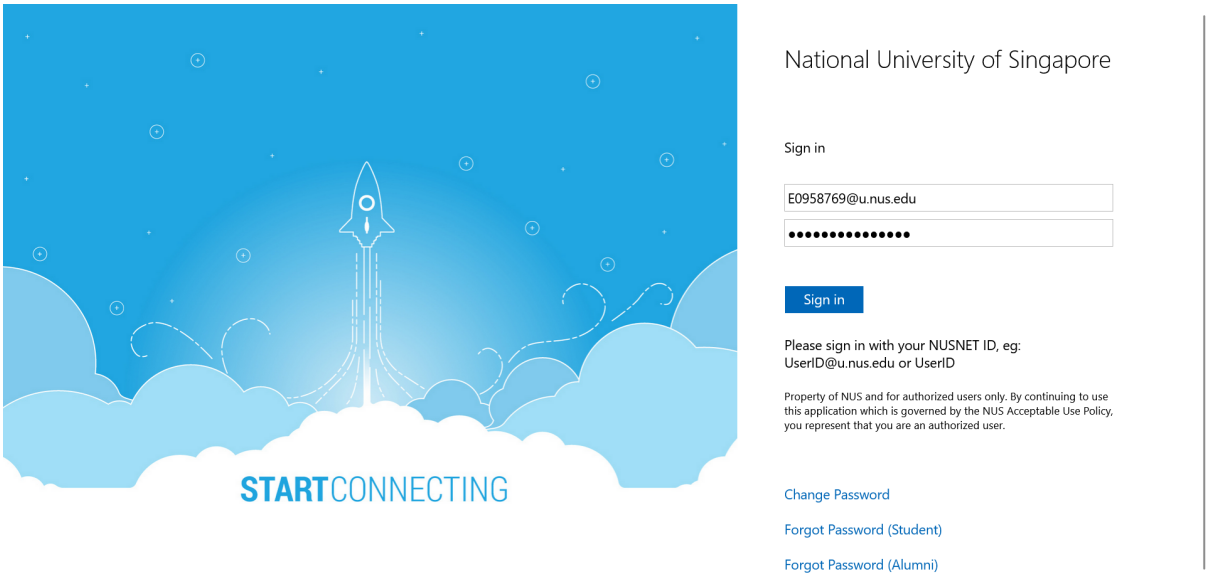


NUS Microsoft Sign In Page



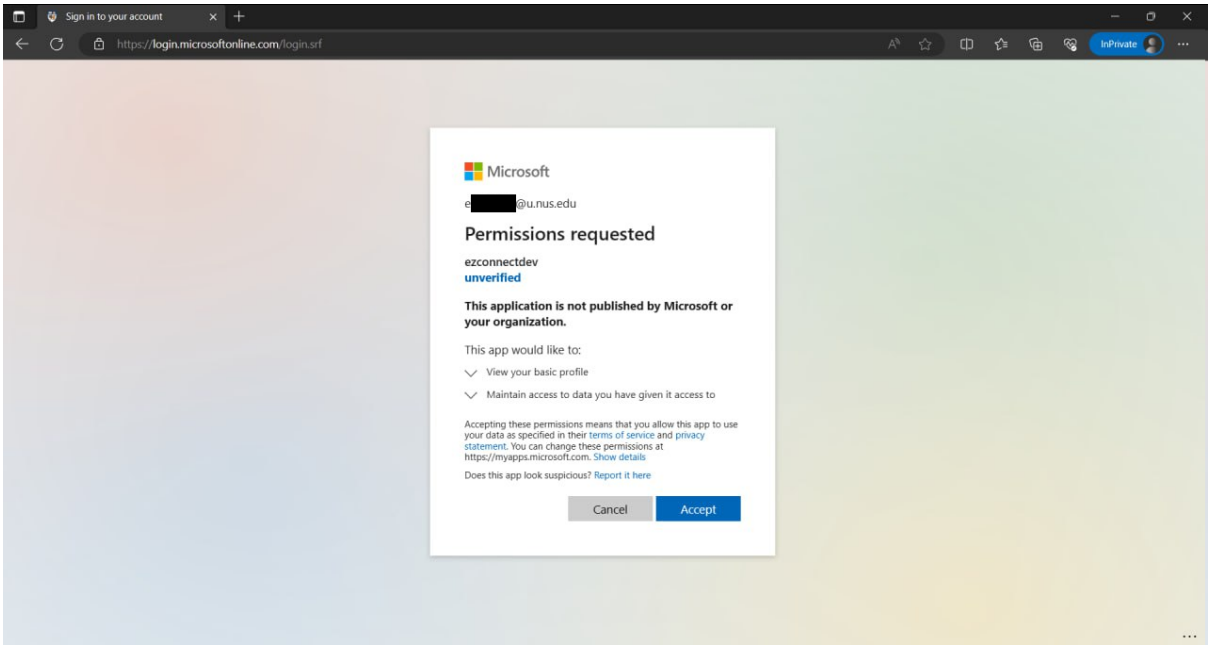
Redirect prompt

The student is then brought to NUS's sign in page where they are prompted to enter their password.



NUS Sign In Page

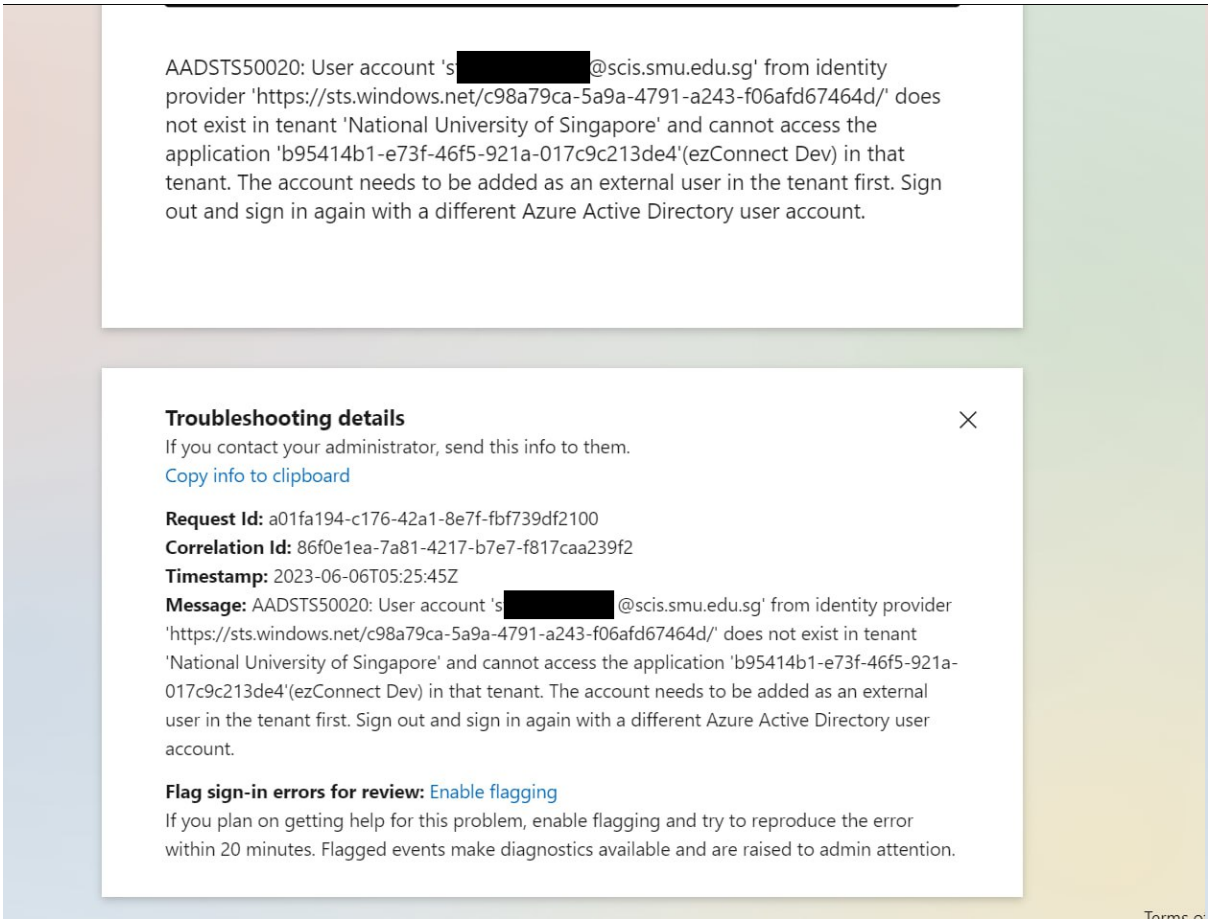
On successful Sign in, if this is the user's first time signing in to this app, Microsoft will prompt the user to give permission to ezConnect to access their personal data. If they have already given permission to the app they are directed to the homepage.



ezConnect asking for permission from a user

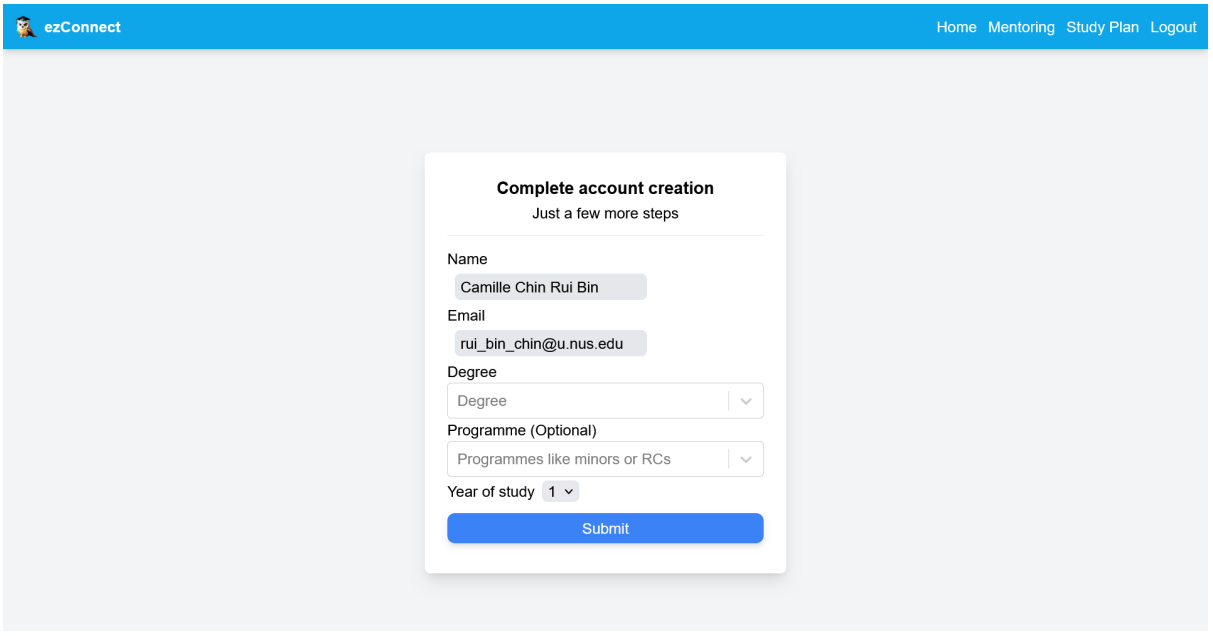
Once the user consents and is redirected, the user will go to the homepage where they will then be redirected to the user creation page.

If the user is not in NUS, they will get an error similar to this:



When a user not from NUS tries to log in

User creation



Prefilled sign up form

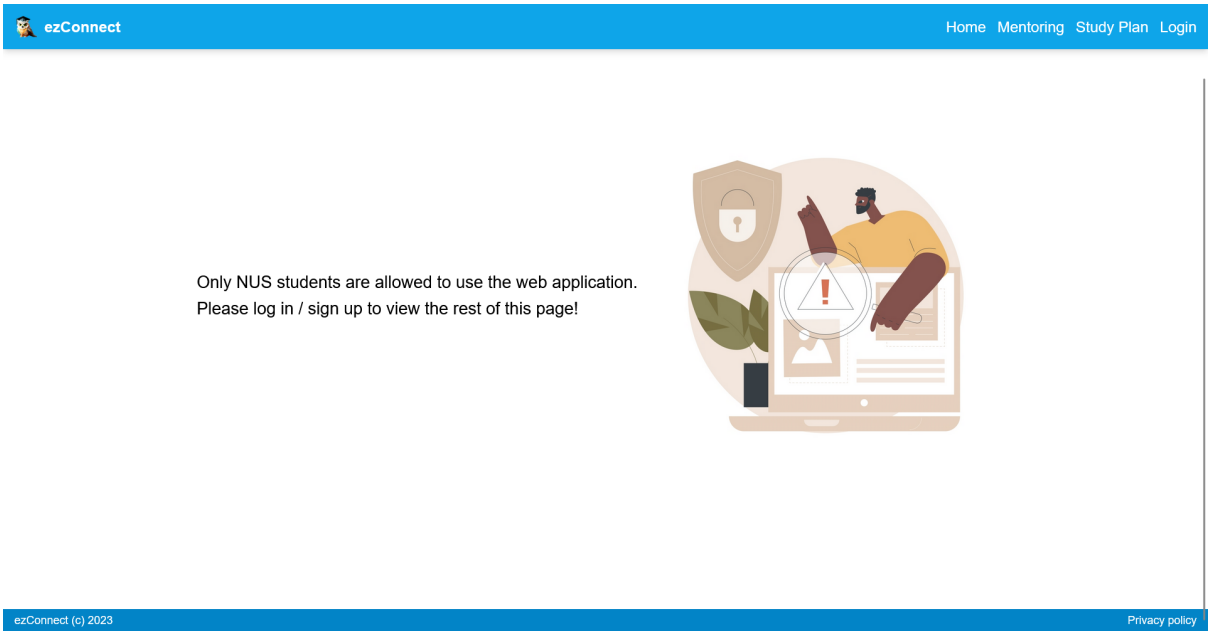
On the user creation page, the name and email of the user is already prefilled with information from their ID token, which is mapped over from details of their original id token given by the NUS Identity Provider.

The user only has to fill in their degrees, programmes, and year of study. Unfortunately, we do not get this information from NUS as that may require a custom API to fetch from. For the purpose of restricting users to NUS students, we feel this feature is good enough.

Internally, we submit the UUID provided by Azure AD B2C, this is because a user email might change. In the future, we plan to add a mechanism for changing or updating the email if NUS email changes.

Restricted pages

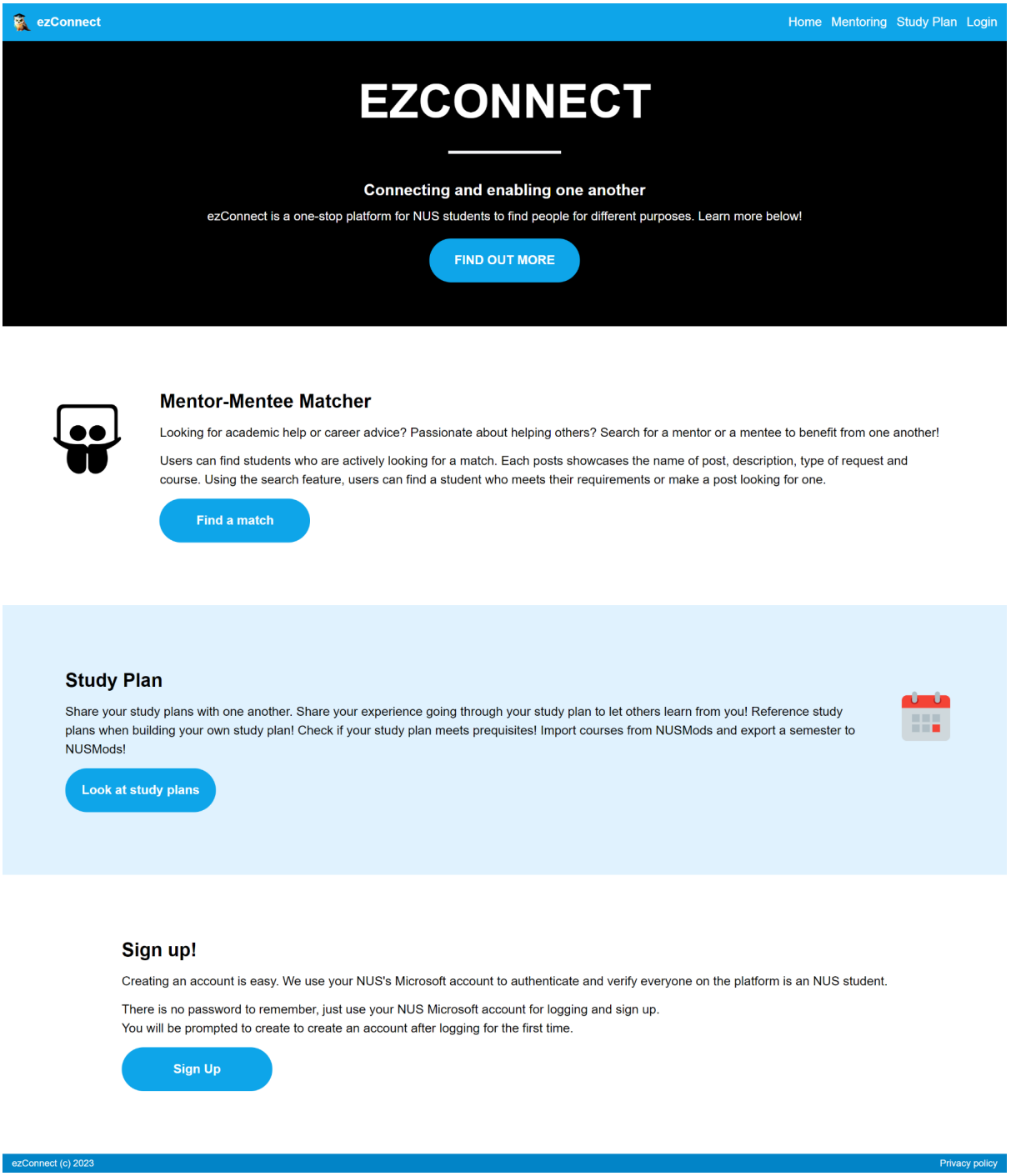
We have several pages that are restricted to logged in users only. Using the MSAL libraries we can easily implement a non logged in view.



None logged in view of homepage

About Us

Upon entering the web app, users will be brought to the about us page. The about us page provides information on the webpage.



About Us Page

If users do not log in, they will only be allowed to view the about us page of the web app. Therefore, making the about us page important in informing potential users about the features

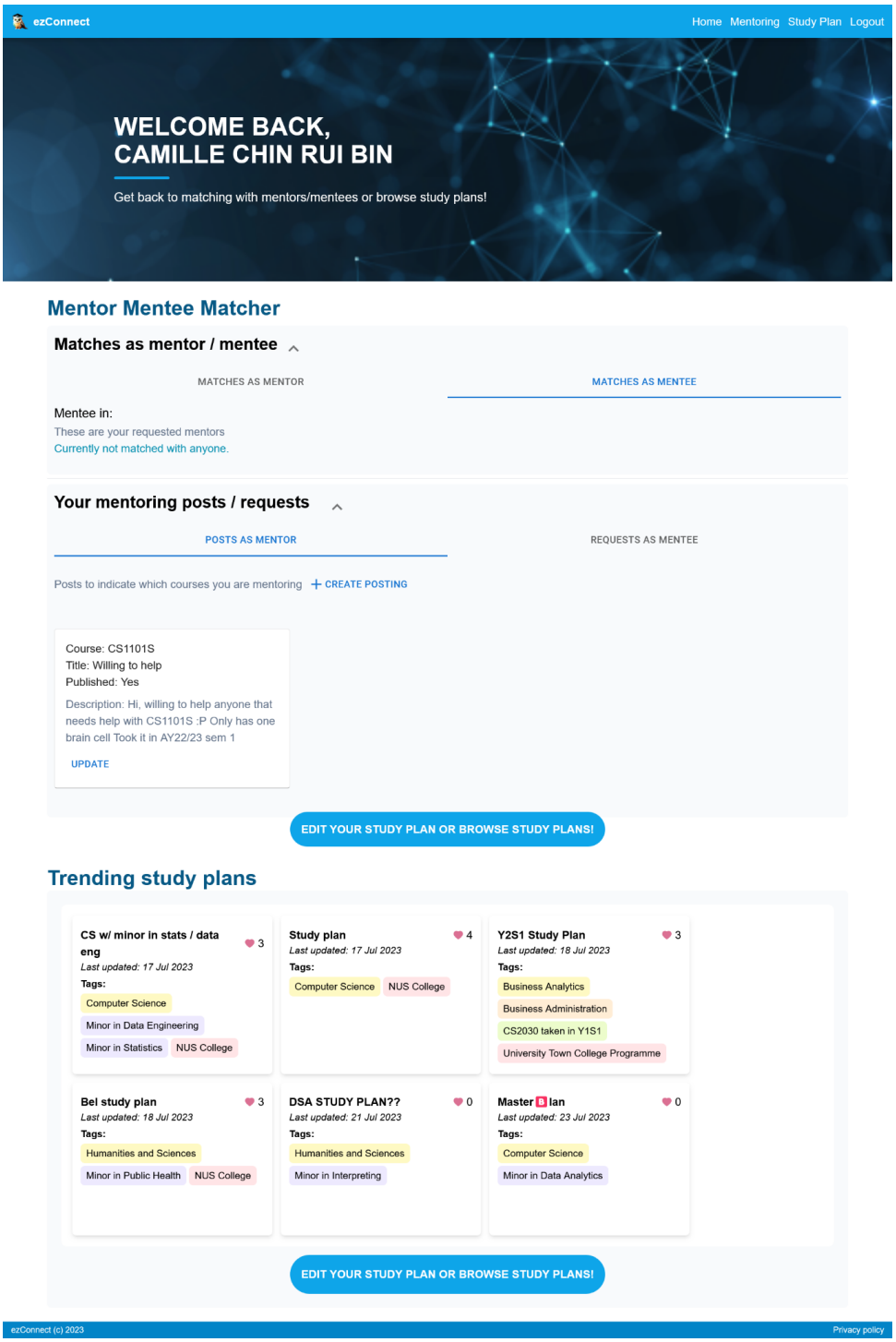
of the web app. Clicking on any features of the web app will show users a limited view of the page and prompt users to log in.

We restricted access to other pages as this web app is intended for NUS students to use and we want to ensure that only authorised users are using the web app.

Homepage

Description

Upon successful login, users can view the homepage that is personalised to them. The homepage will contain essential information about mentoring and study plans. The homepage's mentoring section will contain the user's mentors/mentees and the user's postings. The study plan section will feature trending study plans.



Current Homepage

Mentor-Mentee Matcher

Fulfil user stories: 1, 2, 3, 4

Navigation flow

Our Mentor Mentee Matcher feature is a comprehensive tool designed to facilitate connections between mentors and mentees within our platform. The feature revolves around posts and matches, enabling users to advertise themselves as mentors or seek mentorship opportunities.

There are two types of posts: postings and requests. Postings allow users to create a profile advertising themselves as mentors for a specific course or subject area. These posts act as announcements stating, "I want to mentor" or "I am a mentor." On the other hand, requests are posts made by users who are seeking a mentor, expressing their interest in finding guidance or mentorship. These posts convey messages such as, "I want a mentor" or "I am a mentee."

When a user wants a mentor / mentee, they first create either a mentor request or mentor posting respectively. Then they can either proactively seek out prospective mentors / mentees or wait for interested mentees and mentors to reach out to them. Once a user finds a counterpart they are satisfied with, they request a match and an email will be sent to the counterparty informing them of their request.

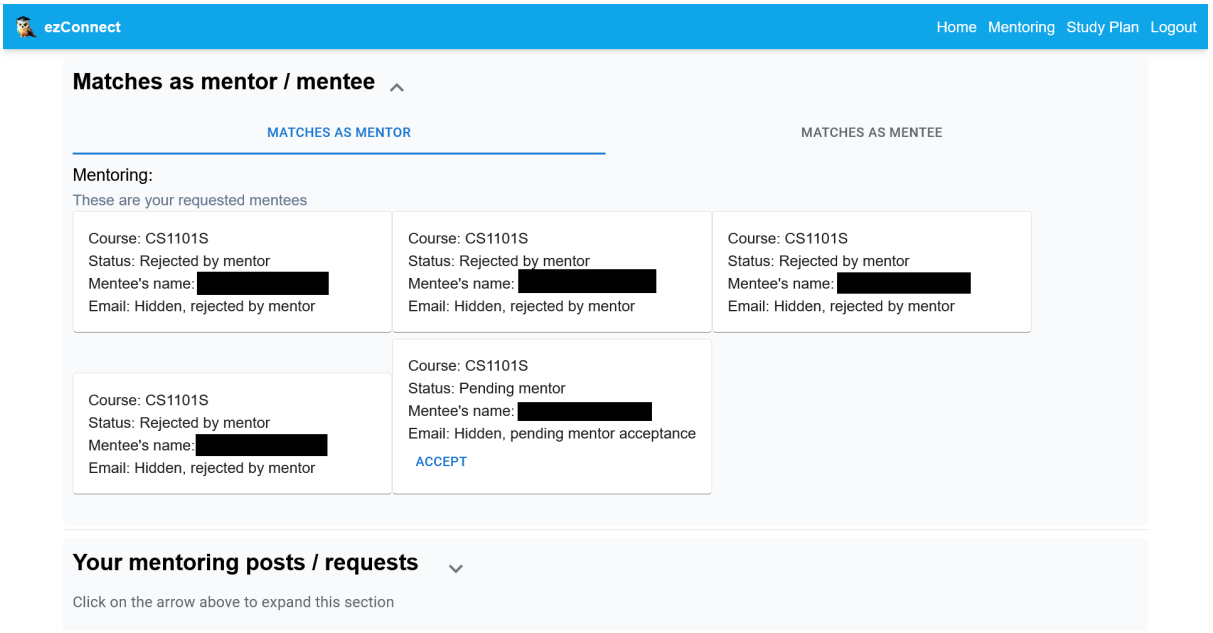
The request can either be rejected or accepted, once accepted both the mentor request (mentee's posting) will be made private by default. When accepting or rejecting, the requested party has to write a message along with their acceptance or rejection. Once accepted, the email of the other party will be visible. If however it is rejected, the email stays hidden. This can be adapted to include other contact details in the future.

Mentoring Main Page

The main page of mentoring serves as the central hub for the Mentor Mentee Matcher feature. The first two tabs are collapsible sections meant for showing a user's matches and posts. These are the same as those shown on the homepage for quick access.

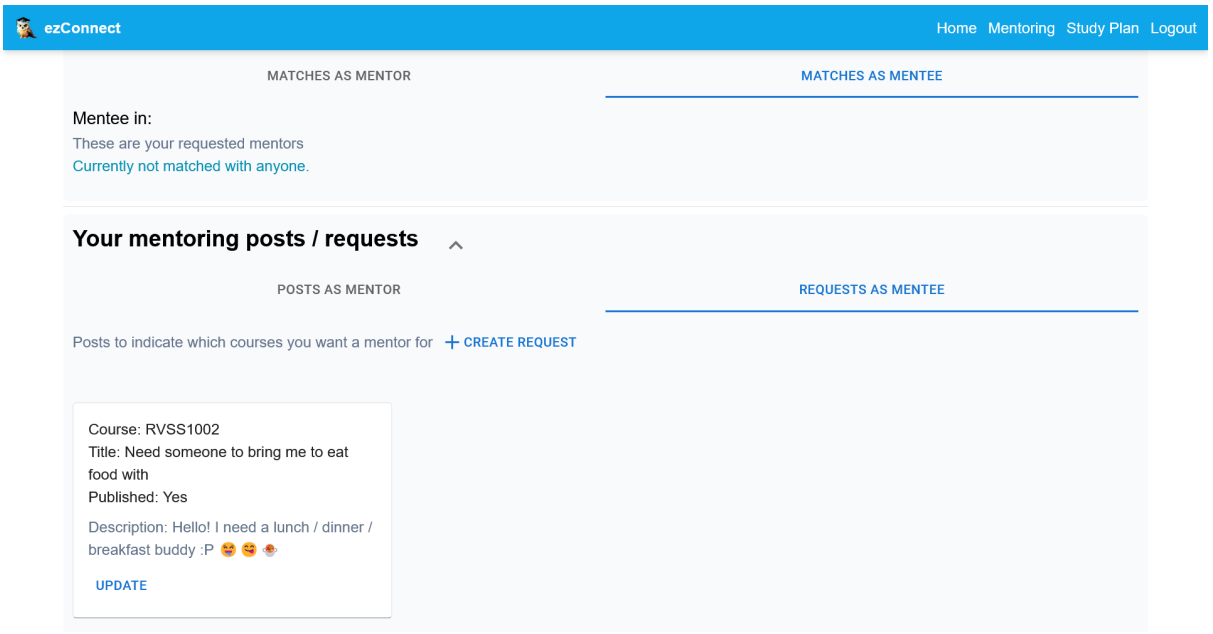
At the bottom of the main page is the community section which lists all the posts or requests in the community sorted by recency. There is also a course selector to search for specific courses.

Mainpage screenshots



Tab showing matches and the collapsed posts tab

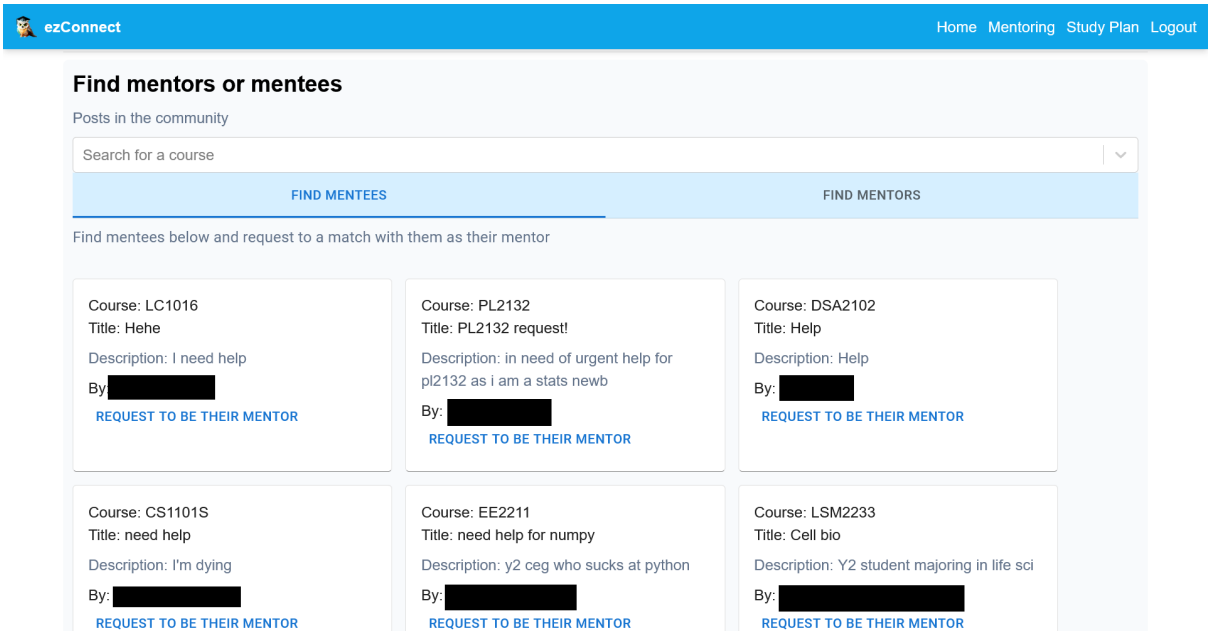
If a match is pending acceptance by the user, an accept button will appear. Otherwise it just displays the status and information of the match.



Example of an empty match section and mentor requests tab

Users can also press “UPDATE” to unpublish or change the details except for the course of their postings.

Users can view their matches and posts, which are also already displayed on the homepage, in collapsible tabs at the top. The mentoring main page expands upon this by providing a community area where users can explore all published mentor postings and requests.



Bottom of mentoring feature mainpage with the community section

Posts that are not made by the same user will have a “REQUEST TO BE THEIR MENTOR” or “REQUEST TO BE THEIR MENTEE” button. Pressing this will bring the user to a request match page with a button to initiate a match. This will be elaborated on in the later section.

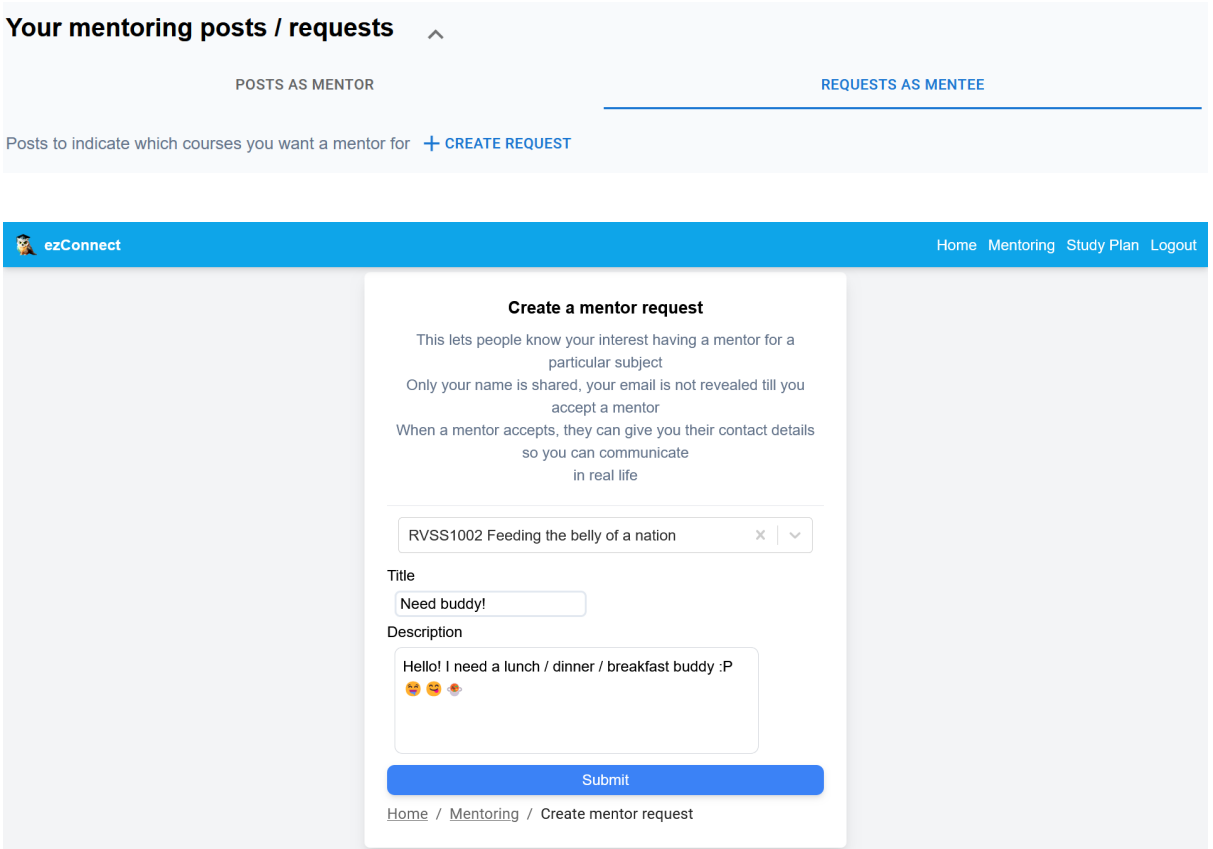
Creating posts / requests and matching

We decided that each mentor posting should correspond with a mentee request. We require that a user first create a request for that course when applying to a mentor posting since we thought that it would be logical to still have the post be up in case that mentor rejects the match, that way other students in the community would still be able to have the opportunity to mentor or mentee for that student.

Additionally for our database, even though post and request seems very similar to one another, we decide to keep them as separate classes or models, and hence tables as possible expansion in the future can make each other diverge from a common schema, beyond the current column, for example, it would not make sense for a mentor request to keep a column for amount of mentees currently being mentored by a mentor.

The initial reasoning behind having a published column is so mentors can choose to pause taking in mentors or mentees without deleting their entire posting. It was also thought of as a way to have a draft posting as a user can write something and save it without publishing so they can publish it again when they are ready.

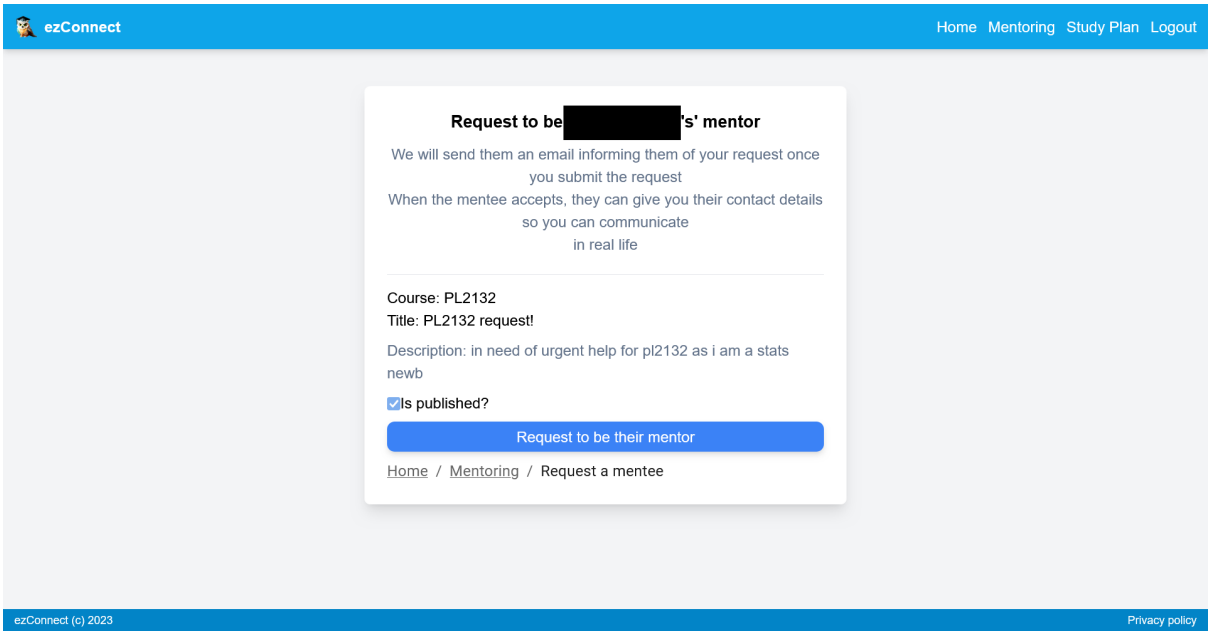
To create a posting or request, they just have to press the button under their respective tabs.



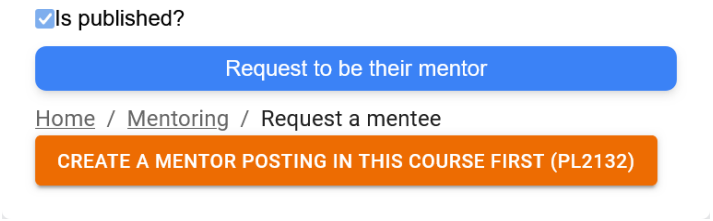
Mentor request creation page

If the post is created successfully, they will be redirected to the main page of mentoring.

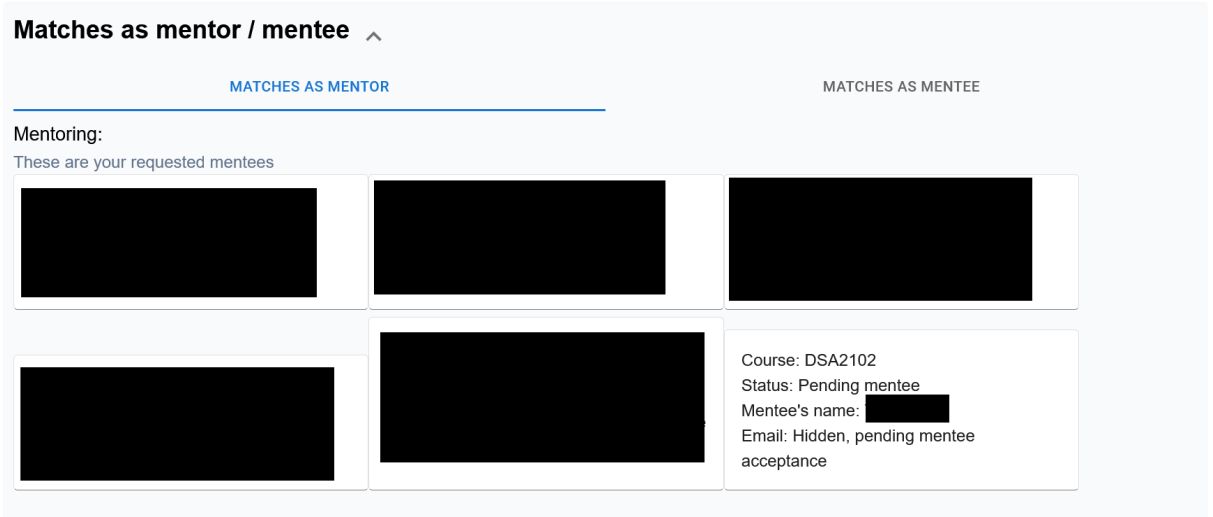
When a user finds a person they want to mentor or be a mentee for, they can press the “Request to be …” button to be brought to the request match page.



Request to be their mentor page

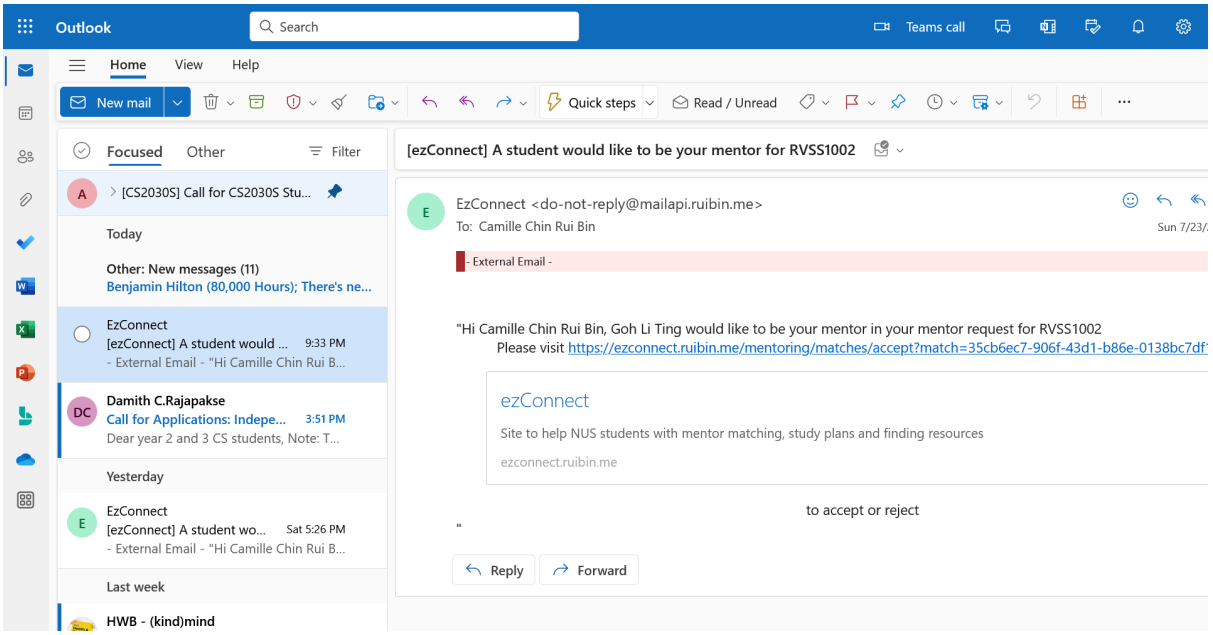


Button that appears when a user do not have a post / request for that course yet

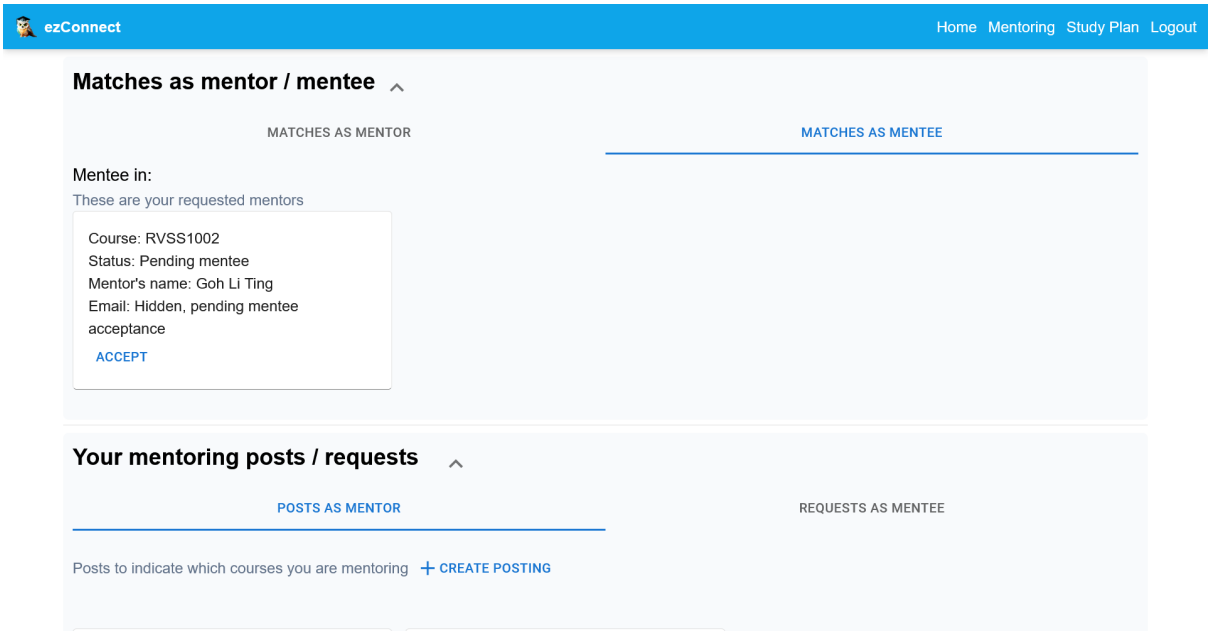


The new match in the match tab

A new match would then be added to the correct tab, as mentors or as mentees. If the user is the accepting party a button would appear to accept. The accepting / requested party will also receive an email.



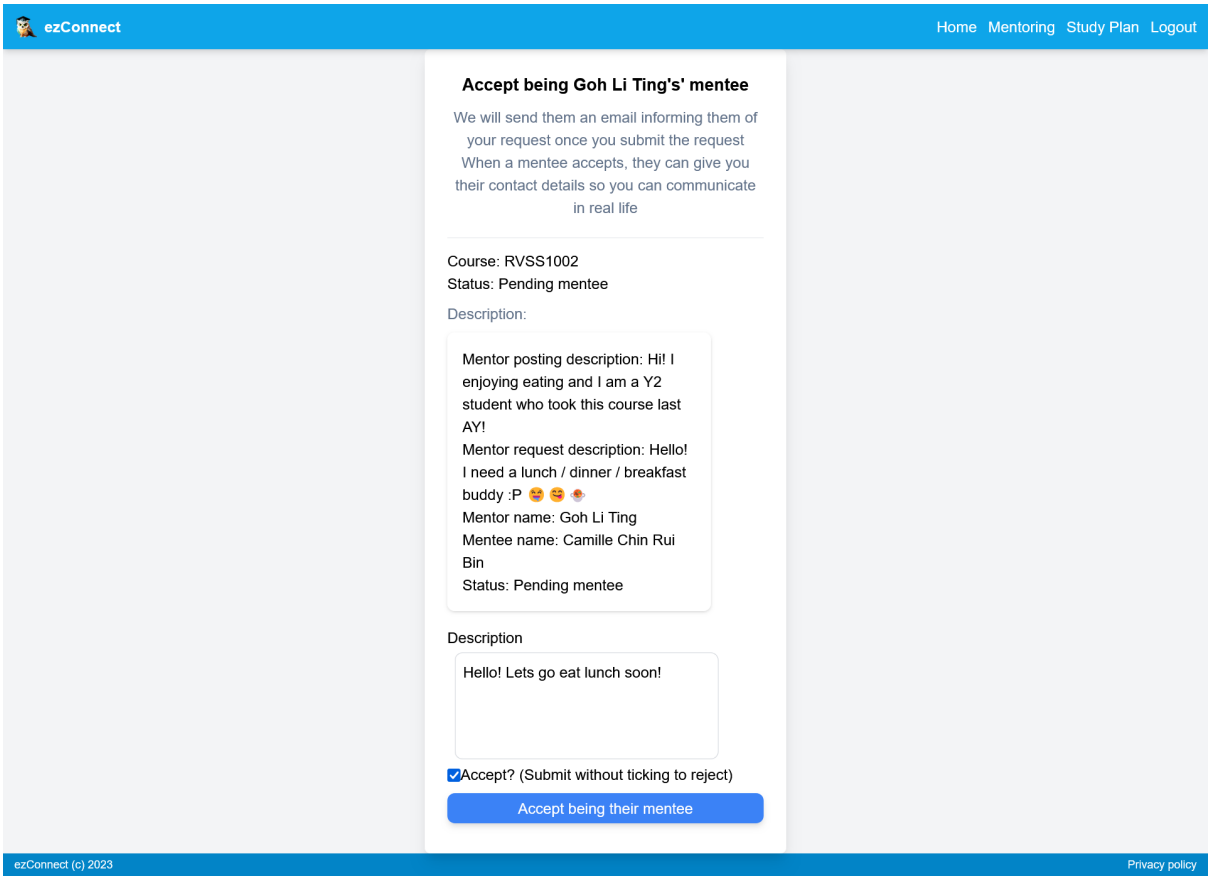
Request email notification



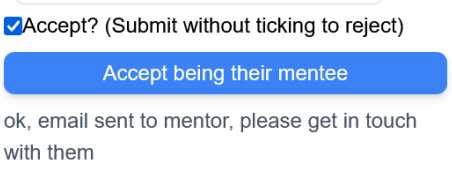
Match request (for a mentee) as the mentee in user's matches tab

The link in the email points to the same page as the accept button in the match tab. This way, if the user accidentally deletes the email, they can still find the match in their homepage or mentoring page.

Pressing the accept button will open up the accept request page.

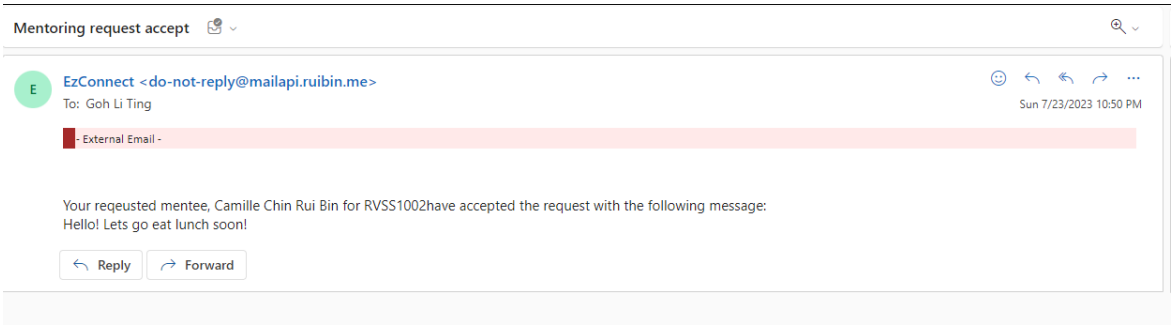


Accept page

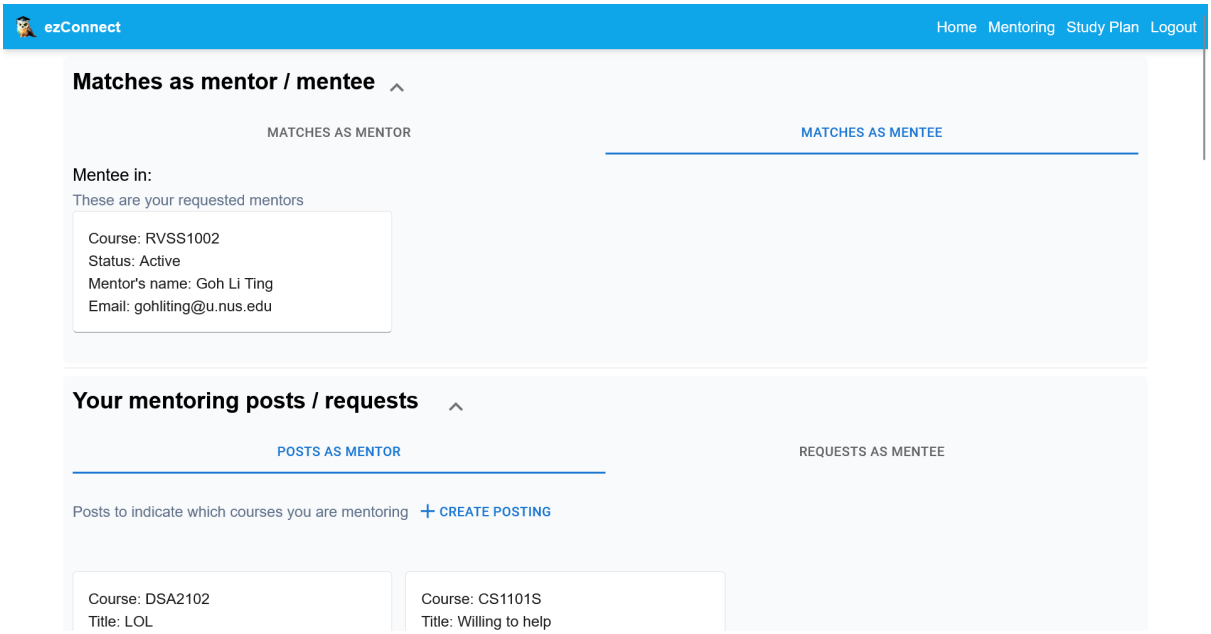


Info message indicating mentee have accepted

After accepting the request, an email with the description written by the user will be sent to the mentor who requested the match. The reverse will be true if a mentee requested a match with a mentor.



Message from mentee accepting the match from mentor



The match will be reflected in the matches tab and status set to active

After the opposing party accepts, the match is now set to active and the email will be revealed. This functionality can be useful in the future for controlling other features we can add to active matches.

Potential Expansions

The mentoring feature is easily modularised due to separation of concern and extensibility. We plan to introduce features after milestone 3 to improve and flesh out the mentoring platform, bringing its community section to parity with the study plan gallery.

We aim to allow deleting of postings / requests so the view will not be as cluttered as gathered from feedback during our beta testing. Users would also be allowed to directly create a request or posting when requesting a match if they do not already have one for the course to reduce friction.

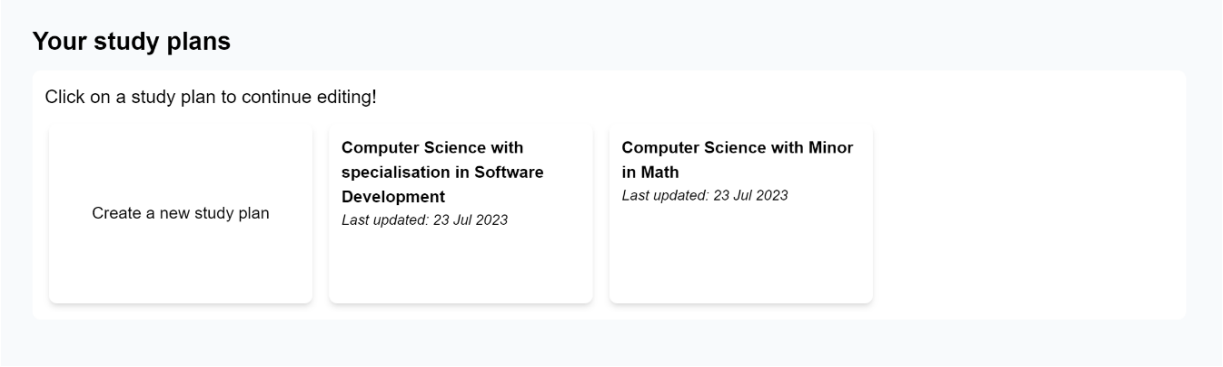
[User stories fulfilled: 1, 2, 3, 4]

Study Plans

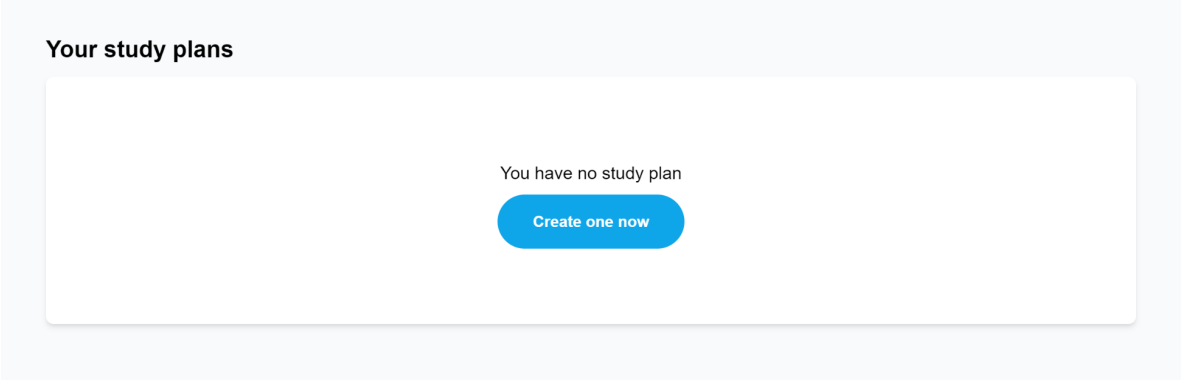
User stories fulfilled: 6, 8

Personal Study Plan Gallery

Users can view their created study plans in the *Personal Study Plan Gallery*. This allows them to quickly view the study plans that they have and access the study plan that they want to continue editing. The personal study plans are always ordered by date such that the most recently updated study plan will be shown first as users are more likely to edit and view a study plan that they have recently updated.

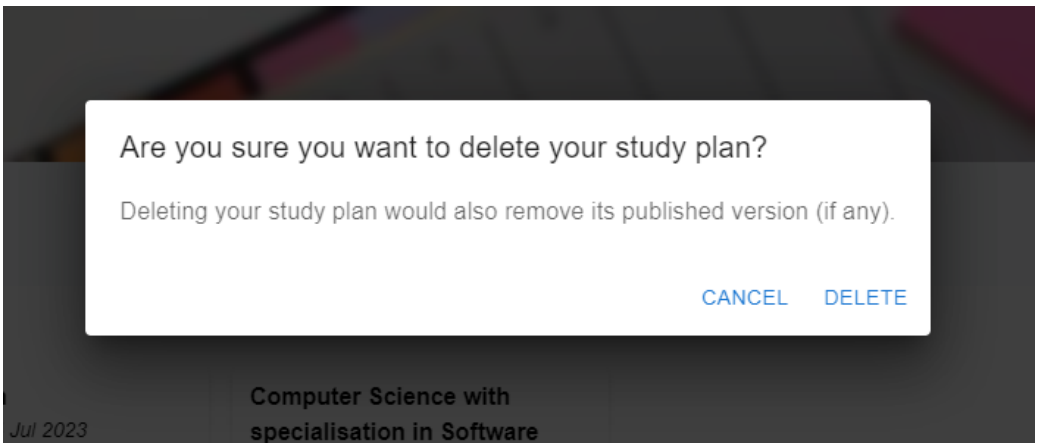


Personal Study Plan Gallery



Personal Study Plan Gallery if no study plans have been created

Users can directly create and delete their personal study plans from this page, increasing convenience for users. These are implemented by making API calls to the backend to GET, POST, and DELETE. When a user creates a study plan, the study plan is automatically set as “is_published = False” to keep the study plan private. When a user deletes a study plan, an alert will appear on the screen, just in case the user accidentally clicked on the delete button.



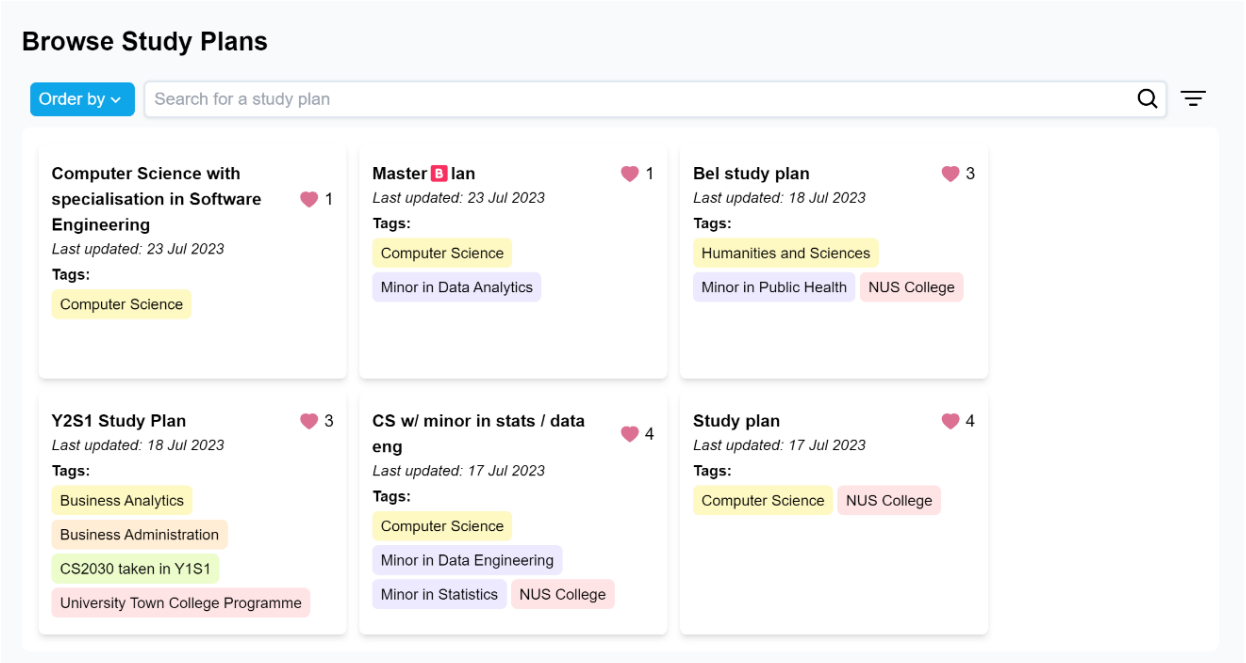
Delete Prompt

Students often do not have a clear idea of the minors or specialisations that they want to pursue in the future (especially for freshmen). Hence, users are allowed to create multiple study plans intended for users to explore different academic plans. The ability to set different titles for the study plans allows users to differentiate the study plans easily.

Study Plan Gallery

User stories fulfilled: 5

Users can view study plans that are created and published by other users. Students often find it confusing to plan their own study plans as they do not know which courses are better to be taken together or separately or which order they should take them in. By viewing study plans made by current students, students can consider these study plans when making their own study plan.



Study Plan Gallery

Each study plan in the Study Plan Gallery will display the study plan's title (which briefly informs users about the study plan), number of likes, date updated and tags. This allows users to quickly view multiple study plans at once. Clicking on a study plan can allow users to view more information about the study plan (elaborating in *Study Plan Post*).

The number of likes indicates the receptiveness of the study plan. If there are a large number of users using the web app, the number of likes can differ significantly between study plans, allowing users to use the number of likes for their consideration when referencing study plans.

The last updated date can provide an indication of the relevancy of the study plan to the user. A study plan that has not been updated for years is likely to be outdated and irrelevant to the user as programme requirements change over the years. While outdated study plans will not be an issue now, if the web app runs for a long time, it could become a potential issue.

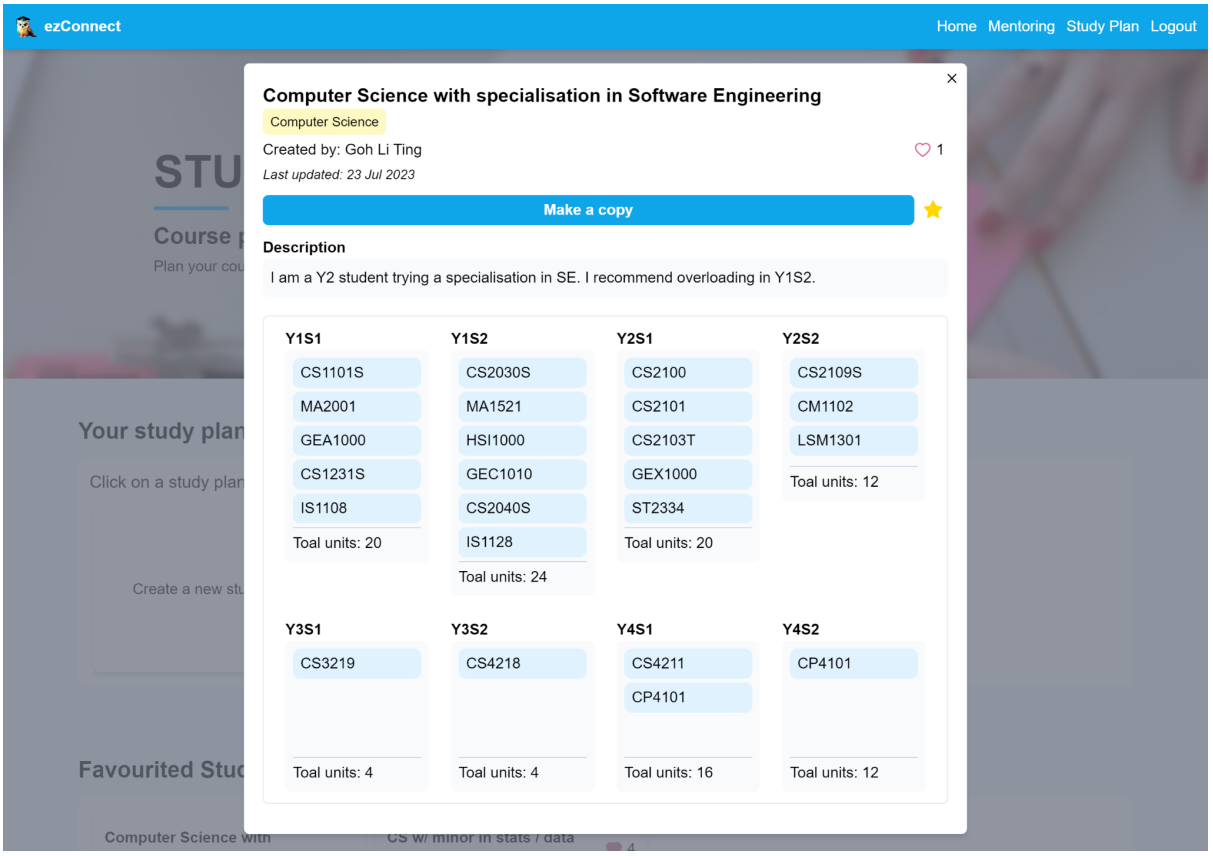
Tags highlight essential information about the study plan. Using the tags, users can quickly identify what these study plans are about without looking at the courses in the study plan. The

study plans are also colour-coded by “first degree”, “second degree”, “second major”, “minors” and “specialisations”. The colour coding helps users to differentiate the different degrees and programmes and makes the gallery look more aesthetically pleasing.

Study Plan Post

User stories fulfilled: 5, 7

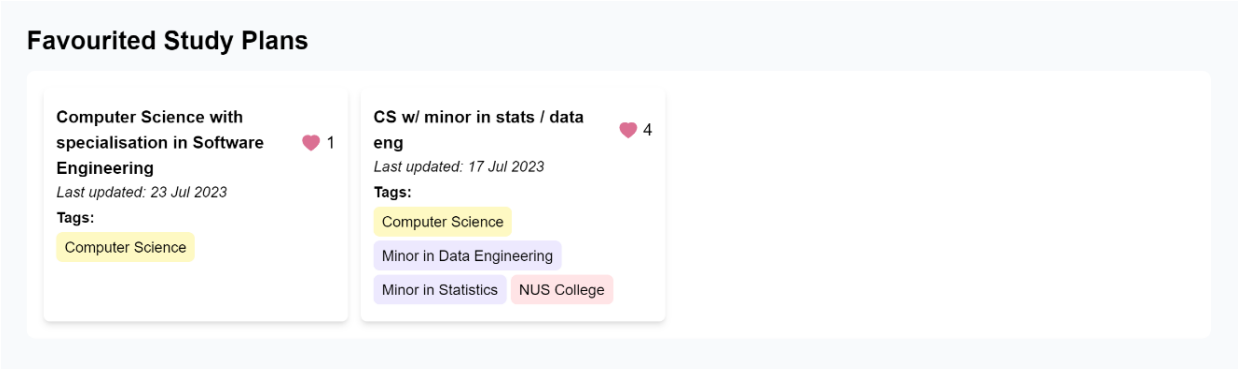
Clicking on a published study plan opens a pop-up window which displays information about the study plan: a preview of the study plan, title, tags, creator, last updated, likes, and description. Users can view the courses in the study plan and read the description of the study plan which can contain an explanation behind the courses that are taken, the background of the creator, advice from the creator or the creator's experience with the study plan. This allows users to understand more about the study plan.



Study Plan Post

Users can directly make a copy and work on a copy of that study plan. This makes it more convenient for the user when creating a study plan as they do not need to search for the courses themselves if there are a lot of overlapping courses. After making a copy, users will be redirected to their new study plan so that they can start editing it.

Users can choose to favourite the study plan by clicking on the star button. The favourited study plans will appear in the favourited study plan section on the study plan main page. This allows users to have easier access to these study plans for easier reference and comparison. The study plans in the favourited study plan page can be interacted with, like in the study plan gallery.

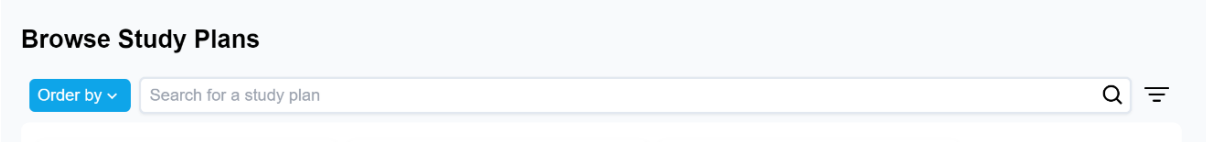


Favourited Study Plan Section

An implementation challenge we faced was that we wanted to implement the preview of the study plan using an image that was also supposed to appear in the study plan gallery. However, we could not come up with a suitable way to create an image. We considered using a puppet page to render the study plan and create an image. However, after further consideration, we realised that putting a picture would make the courses too small to be seen and make the preview useless. We decided that rendering a preview of the study plan directly in the post would be more ideal and easier to view.

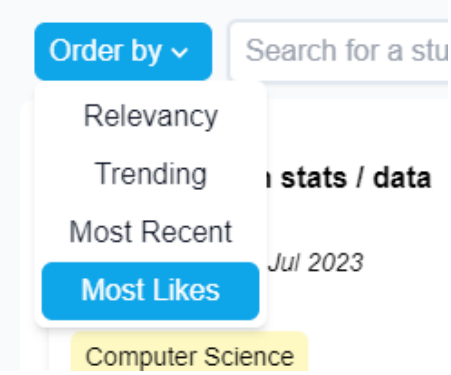
Search, Order by and Filter

User stories fulfilled: 6



Search bar

Users can search for published study plans based on the title. The gallery will show all the study plans whose title includes the search input. This is done by filtering the study plans whenever the search input changes. The title and search input will be converted to lowercase and then we will check if the search input is in the title.



Options to order by

Users can order the published study based on relevancy, trending, most likes or most recent. This helps users to find study plans based on the order that they deem important to them. The ordering choice is passed into the API call and the ordering is done in the backend.

The *relevancy* of the study plans is determined based on the tags of the study plans and the profile of the user. The more tags that match the user's profile, the more relevant the study plan will be. In the implementation of the relevancy ordering, we faced the challenge of coming up with a suitable way to determine the relevancy of the study plan to the user. We first determined the definition of relevancy, which was the extent to which the study plan fits the degrees and programmes undertaken by the user. We implemented the relevancy ordering by calculating a "relevancy score" for each study plan. A matching first degree, second degree, second major, minor and special programme gives a score of 10, 5, 3, 1 and 1 respectively. Summing these scores up will give the relevancy score of the study plan. We decided to give the first degree a much higher score than other components as the first degree had to match the user's degree to even be relevant to the user. A student can only take up to 3 minors and one second major on top of their degree. A student will usually take 0 to 2 other special programmes. Hence, if a study plan has 3 matching minors, 1 matching second major and 2 matching special programmes but has a different first degree, this study plan will still be ranked less relevant as compared to a

study plan that has the same first degree. After calculating the relevancy score, the study plans will be sorted based on the relevancy score such that the most relevant study plan will appear first.

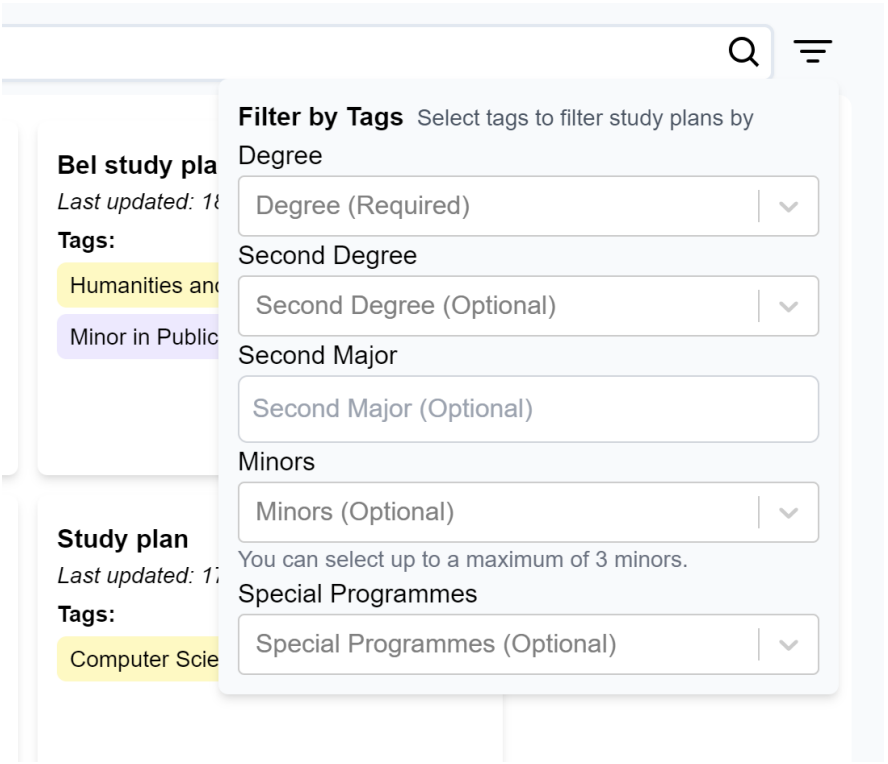
Trending study plans are determined based on the number of views, likes and favourites. We implemented this similarly to the relevancy ordering. We calculated a “trend score” for each study plan. A view, like, and favourite gives a score of 1, 2 and 3 respectively. Summing these scores up will give the trend score of the study plan. When the trend score is calculated, only views, likes and favourites within the last 30 days are taken into account. This is to ensure that study plans that used to be trending in the past will not be shown when trending is chosen, or else study plans with the highest number of views, likes and favourites will be shown instead of the trending study plans.

Views are recorded in the database whenever a user clicks on a published study plan. However, repeated views of the same study plan on the same day by the same user will not be recorded as users may click the same study plan multiple times in a short period unnecessarily and cause the database to be overloaded. Hence, each view is associated with a date, a user and a published study plan. Whenever the study plans need to be ordered by trending, views data beyond 30 days will be removed as such old data is no longer needed in any part of the web application.

When a user likes/favourites a study plan, it is recorded in the database. There are many-to-many relationships between the PublishedStudyPlan and the User model to support likes and favourites. The date that the user likes/favourites a study plan is also recorded to support the trend score calculation.

Most likes study plans are obtained by ordering the study plans by the number of likes when querying the data in Flask from the database.

Similarly, the *most recent* study plans are obtained by ordering the study plans by the date updated when querying the data in Flask from the database. If no ordering is given, the study plans will be ordered by most recent by default.



Filter options

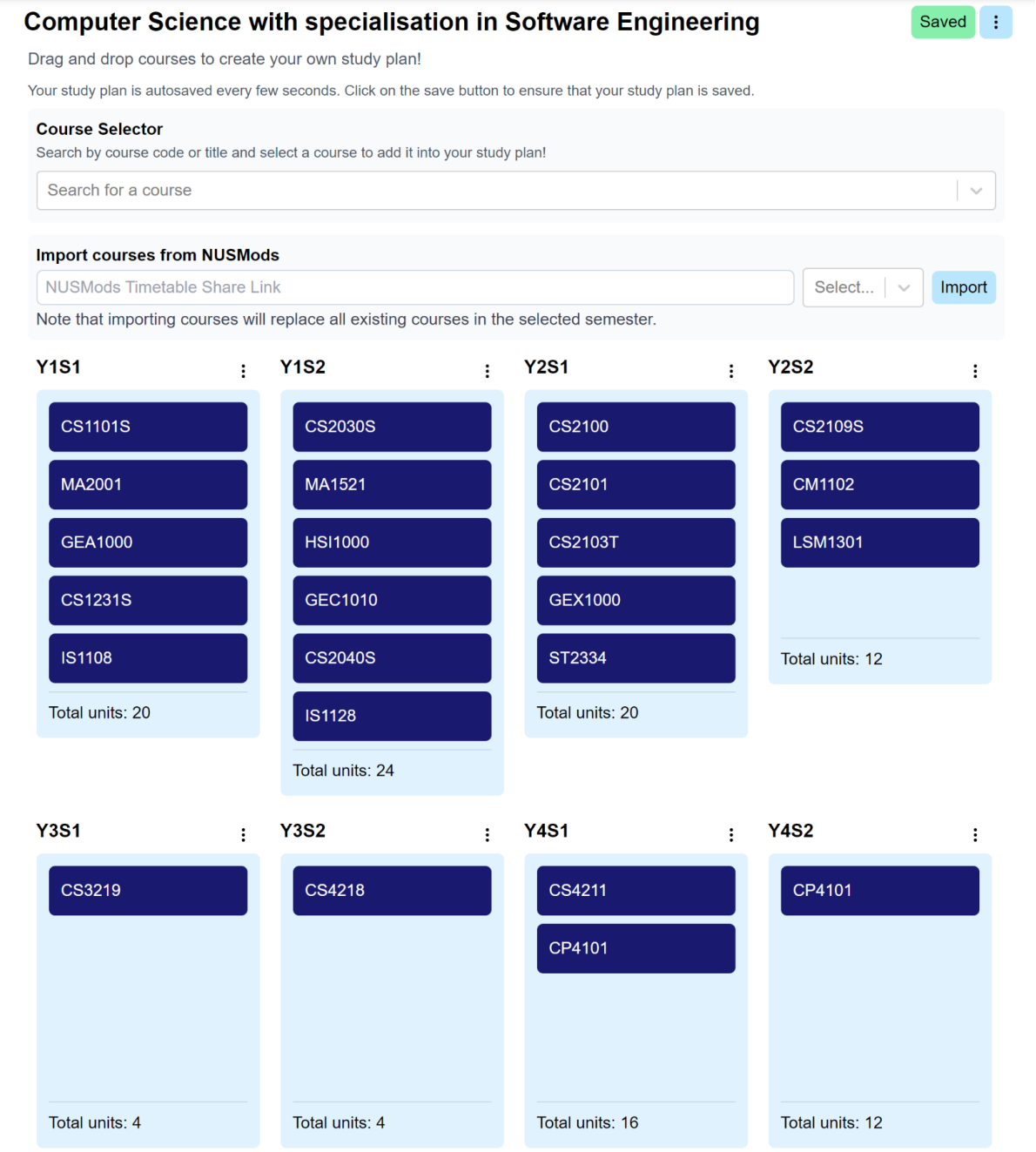
Users can filter the study plans based on tags of the study plans: first degree, second degree, second major, minors and special programmes. For each study plan, we will check if the study plan contains the selected tags. Study plans which do not contain all the selected tags will be filtered away. The filtering is done in the frontend to reduce lag from using API calls. The gallery will automatically be updated as options are chosen in the filter tab.

Using a combination of search inputs, ordering and filtering of study plans, users can search for their desired study plans more efficiently and find relevant study plans.

Study Plan Editor

Drag and Drop Editor

Users can build their own study plan in the *study plan editor*.



Study Plan Editor

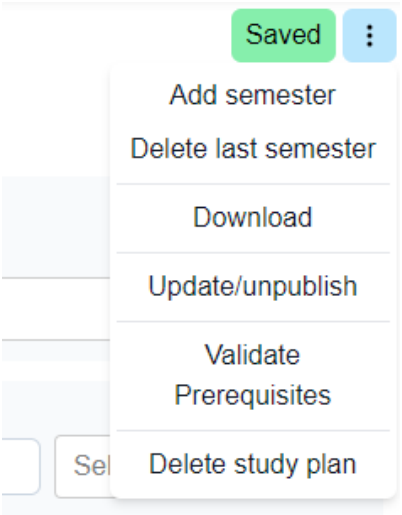
For better user experience, users can drag and drop courses around the editor into different semesters. The number of units obtained per semester will also be calculated automatically for users to ensure that they are taking a suitable number of units each semester.

We faced an implementation challenge where the drag-and-drop editor was extremely buggy. The user experience was poor as consecutively dragging courses around could cause the course to return to its previous semester after dragging. This was caused by the initial autosave logic. Initially, every action that a user took to change the study plan would fire an API call to the backend to save the study plan. After the API fetch call is completed, the editor will be re-rendered. However, if a course was dragged while the API call was being made, that new change would not be reflected when the study plan is re-rendered, causing the new change to be reverted.

To resolve this issue, we tried to remove the autosave feature and implemented a button for users to save their changes. To ensure that the users saved their changes, we tried to implement a prompt to prompt users to save their changes when navigating away from the page by adding a “beforeunload” event listener. However, this only gave the prompt when the webpage was reloaded or when the user was trying to go to a different webpage by typing the link. The prompt will not be shown when the user clicks on the navigation bar or presses the back button. We could not find a suitable solution online.

Therefore, we tried to think of a new way to implement autosave and implemented a new autosave logic. The study plan will be autosaved every 5s. If the study plan is modified, an API call will be made to the backend to save the study plan. However, the editor will not be re-rendered after the API call is completed so that the editor will not feel buggy. We kept the button so that the users can manually save their changes if the user wants to leave the webpage before the 5s. If the changes have not been saved, the button will show “save”, else the button will show “saved”.

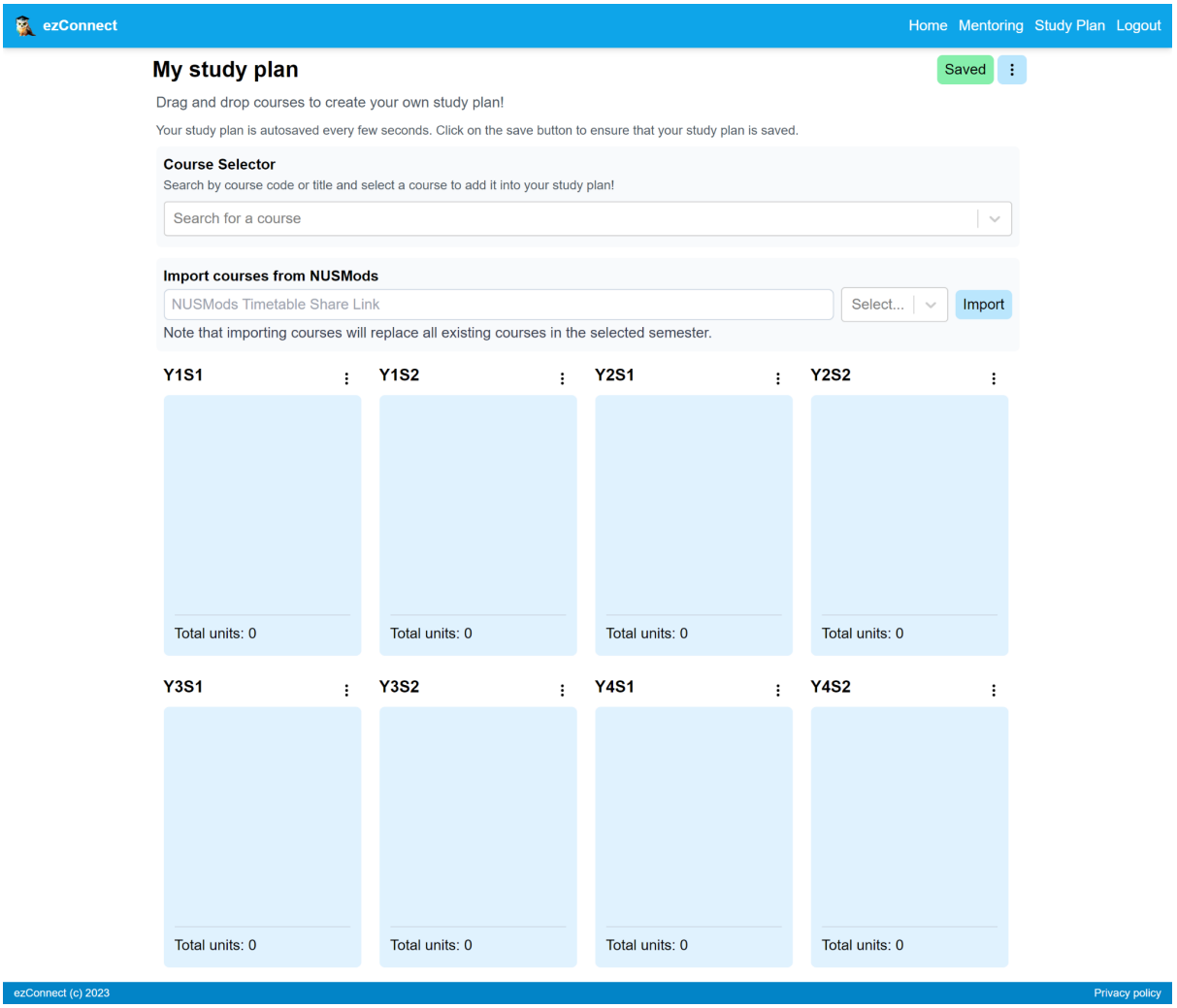
Users can customise their study plans by adding or deleting semesters from the study plan through the editor menu.



Editor Menu

This is to accommodate users who are taking a different number of semesters, such as students who are exempted from certain courses or students taking a double degree. The new semester will be added to the back of the study plan and the semester to be deleted will be the last semester.

By default, the study plan will have 8 semesters as this is the most common number of semesters that a student undergoes. This helps users to prepopulate their study plans with semesters so they do not have to manually add many semesters into their study plans.



Default Study Plan Editor

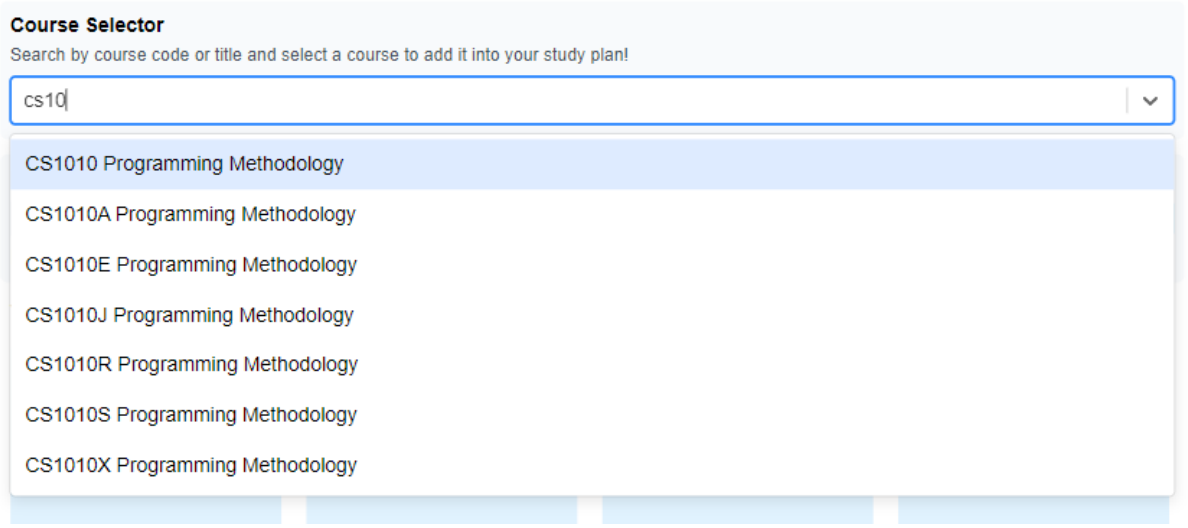
A user can also choose to clear all the courses from one of the semesters through the semester menu if they dislike their selection of courses in a particular semester.

Semester Menu

After completing the study plan, users can download their study plan from the editor menu. We implemented the download functionality by getting the div that the study plan was in and then converting the div into a canvas before creating an image whose file name is the title of the study plan. However, this is similar to screenshotting the study plan as the download button will not give a fixed layout of the study plan. We considered using a puppet website to render the study plan. However, due to a lack of time, we could not explore this option further.

Course Selector

Users can add courses to the study plan by searching for the course code or name. The course selector will only show up to 10 courses.



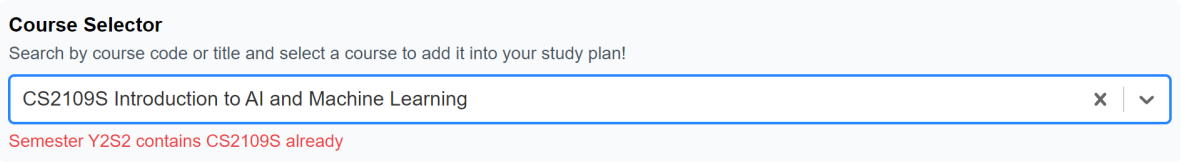
Course Search Bar

When implementing the course selector, we faced an implementation challenge where the page lags due to the course selector. There were a lot of courses in the database and the course

selector could not display all of these options. Hence, we decided to limit the course options to 10 options by filtering away the remaining options. When the search input is changed, the course options will be filtered again based on the search input and limited to 10 options.

The selected course will be added to the last interacted semester as the last interacted semester is more likely to be the semester that the user intends to add the course into. This will reduce the need for users to continuously drag the courses around the editor. A semester is considered to be interacted with when a user drops a course, imports courses, clears courses from the semester or adds a new semester. Clicking on a semester is also considered an interaction. When a semester is deleted, the semester preceding the deleted semester will be considered as the last interacted semester. If no semesters were interacted with, the first semester will be considered the last interacted semester.

Users can add duplicate courses into the study plan in different semesters. This is to allow year-long courses to be added to the study plan. Users are not allowed to add duplicate courses into the same semester as a student cannot take two same courses in the same semester. Trying to add a duplicate course into the same semester will flag an error to the user.



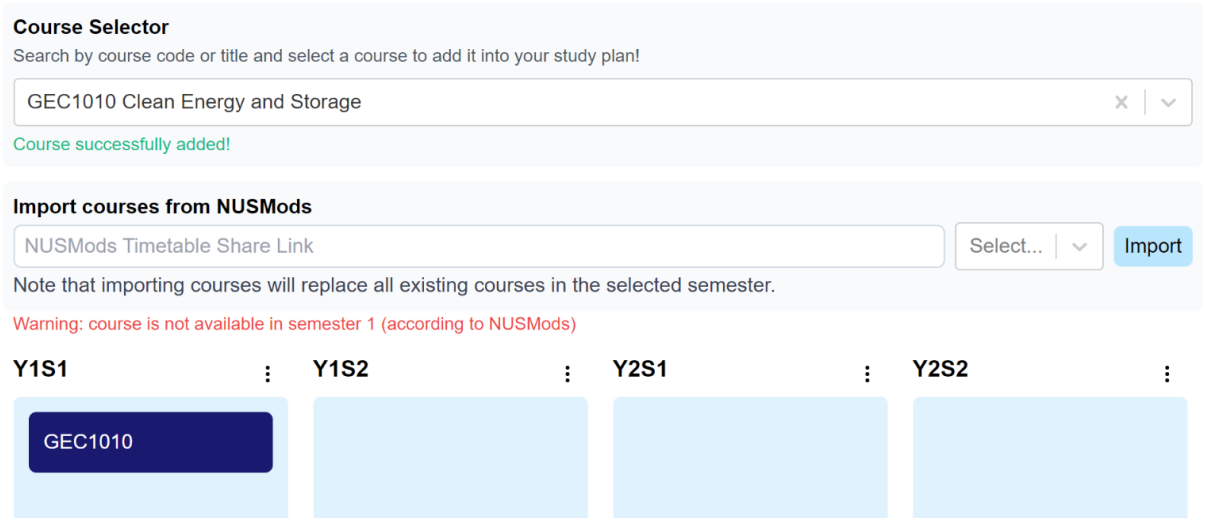
Error Message

When implementing this, we faced an implementation issue where adding a duplicate course breaks the dragging functionality of that course. This is because “course codes” are used as the “draggable id” which needs to be unique. This causes all courses with the same course code to be dragged together before running into an error. We fixed this by changing the draggable id to be a combination of the course code, semester number and index of the course in the list, which ensures that all courses have a unique draggable id.

We later realised that the unit calculation does not support year-long courses well. Year-long courses should contribute only half its units to each semester. However, all the units will be used in the calculation of the total number of units in the editor. While we can know whether a course is a year-long course from the NUSMods API, we did not include this attribute in our database.

When adding courses or after dragging or dropping courses, the availability of the course added in the selected semester will be checked. If the course is not available in semester 1 or 2, based on the semester the course is being added to, a warning will be sounded out to the users. However, users will still be allowed to add the course to the semester as the course may be available in future semesters but our data is only from the current academic year. Furthermore, some courses may be taken outside of the academic year, causing them to be unavailable in

both semesters, such as internship attachments, but users may still be keen on adding them to their study plan, such as for unit calculations.

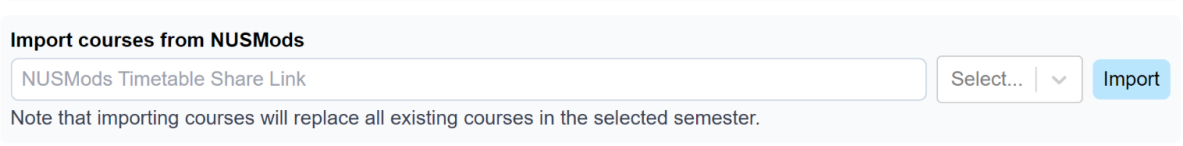


Warning Message

Integration with NUSMods

[User stories fulfilled: 9]

Other than adding courses through the course search bar, users can import their timetable from NUSMods. The courses in that timetable will be added to a semester chosen by the user and overwrite any existing course in the selected semester. This makes it easier for users who have already created timetables on NUSMods to add their courses to the study plan.



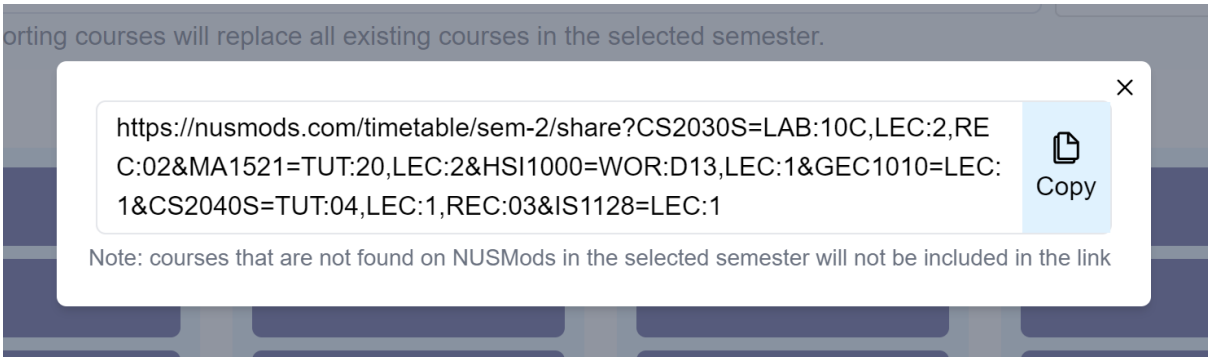
Import Bar

We implemented the import functionality by parsing the import link using regular expression. We will find the course codes using the pattern `/(?![?&])([A-Z]+[0-9]+[A-Z]*)(?=\=)/g` as all course codes will appear after ? or & and appear before "=". This will give us an array of course codes. We will then filter the output list by checking if the course codes appear in the hashmap of courses to remove courses that are not in our database so that we do not run into errors later on.

Users can export the courses in a semester through the semester menu. This allows users to not have to key in those courses manually into NUSMods. As certain courses are only available in semester 1 or semester 2, if the course is not found in the selected semester, then the course

will not be included in the link. Users can also copy the link into their clipboard using the copy button.

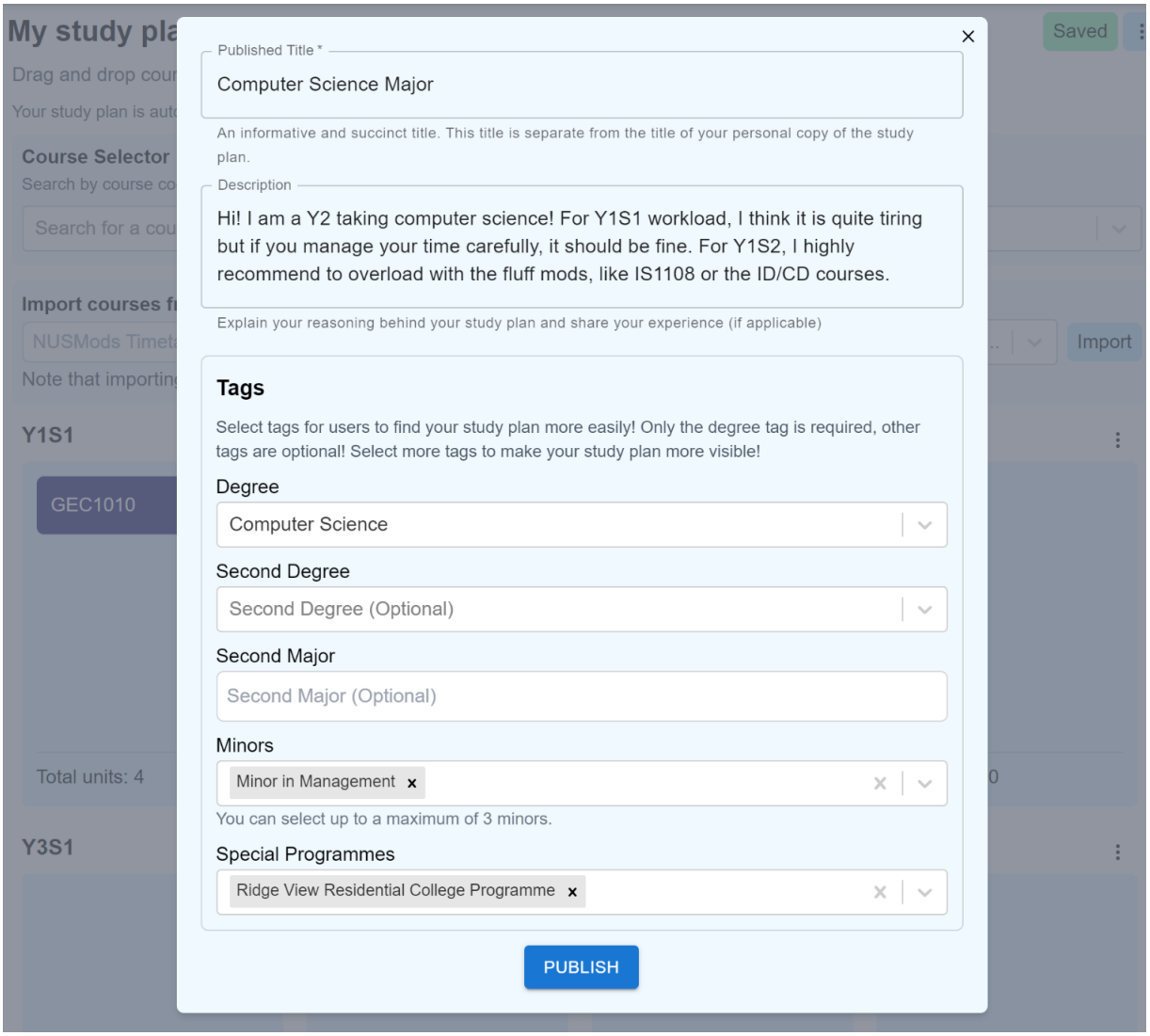
We implemented the export functionality by iterating through the course list in the semester. For each course, we fetch the course information from NUSMods API. Then we get the semester data from the API response. We then find a semester that matches the semester number of the semester that is being exported. After that, we will iterate through the timetable information obtained from the semester data, which gives an array of information on all classes. We will iterate through this array and if we find a new lesson type, we will add the lesson type and class number to the link. The resultant export link will be dependent on the order of the classes that the NUSMods API returns.



Export Semester

Publisher

Users can publish their study plans through the semester menu. Users will be required to fill up information about the study plan. Users are encouraged to provide an informative title and optional description to provide their thoughts on their study plan to help other users. Users will also be encouraged to select tags to add to their study plans so that other users will know what the study plan is for. Only the degree is mandatory as all students will have a degree.



Study Plan Publisher

After publishing a study plan, changes made to the study plan will not be reflected in the published version. Users will have to click on “update” to update the study plan. This is so that the user can continue to edit their study plan and update the published version only when they are ready to do so.

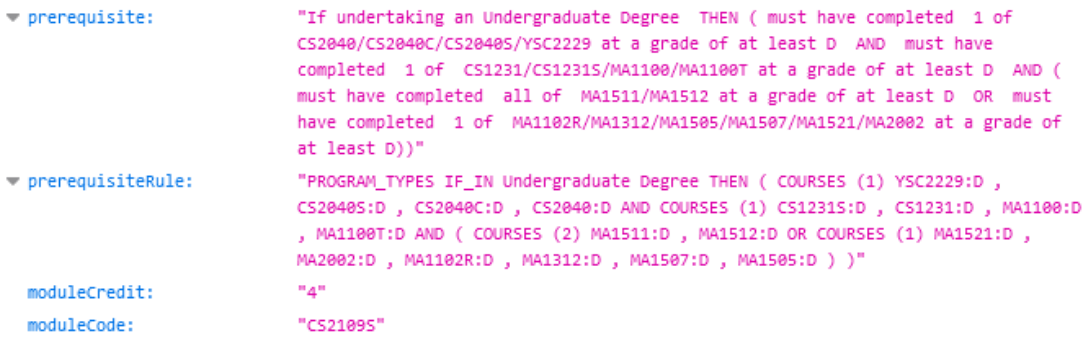
Study Plan Validator

User stories fulfilled: 8

The validator is a complex feature for validating that the user's study plan fulfils the course requirements for the courses in their study plan. This feature aims to help the user have better confidence in the validity of their study plans and the order of the courses they intend to take. Potentially the feature can be expanded to validate their academic plan, making sure university-level requirements and degree graduation requirements are met as well. However, due to the lack of time, this is not implemented.

Implementation and challenges

Implementing the validator was an extremely complex process. We were aware that NUSMods used to have a prerequisite tree feature and we intend to use the data backing that for our feature.



Sample module information json from nus mods api at https://api.nusmods.com/v2/2023-2024/modules/CS2109S.json

The first challenge is to parse and convert the prerequisiteRule into something a program can work with. We noted the similarity of this with SQL and that the presence of keywords like 'AND' and 'OR' indicates boolean operations.

We immediately recall what we learnt in modules like CS1101S. In essence, we are building a calculator that works with this custom language. We also have to find a way to handle the logic of the 'COURSES' keyword.

We noted that the vast majority of prerequisites follow the format '`PROGRAM_TYPES IF_IN Undergraduate Degree THEN ( …`'. We initially wanted to use a library to pyparser to extract the tokens and form an abstract syntax tree but due to time constraints we settled on building a simple calculator-like function to resolve the rule, similar to how a program would solve a maths equation.

Eventually, we decided to use regex to tokenize the rule into a list of tokens (extremely primitive lexical analysis). The regex
$((COURSES(\([1 - 9]\))?)|([A - Z] + [0 - 9] + [A - Z] *)(? = (\: [A - F])?)|\(|\)|OR|AND)$ will

be able to be able to extract all the keywords from a rule. For example, the rule above for CS2109 will be tokenized into a list containing the tokens:

```
['(', 'COURSES (1)', 'YSC2229', 'CS2040S', 'CS2040C', 'CS2040', 'AND', 'COURSES (1)', 'CS1231S', 'CS1231', 'MA1100', 'MA1100T', 'AND', '(', 'COURSES (2)', 'MA1511', 'MA1512', 'OR', 'COURSES (1)', 'MA1521', 'MA2002', 'MA1102R', 'MA1312', 'MA1507', 'MA1505', ')', ')']
```

To save time, the skeleton was generated by telling Bing AI with the prompt; "*make a iterative function in python that solves boolean algebra given in a list of tokens that supports nested equations and brackets*" to generate a Boolean evaluator that will take a list of tokens.[4] We then corrected the program which was wrong to work with simple boolean expressions, supporting only the keywords; `'TRUE'`, `'FALSE'`, `'AND'`, `'OR'` and parentheses. Once we are satisfied the program is working, we begin implementing the custom keywords `` `COURSES` ``.

The function currently works by going backwards and evaluating an expression whenever it encounters a closing bracket; this is incompatible with the `` `COURSES` `` keyword, which contains course codes that are after the keyword. Backward seeking is simply not ideal for this.

To solve this, the function collects the trailing course codes until it hits another keyword and then evaluates it and adds it to the stack of evaluated tokens before.

After adding the function to support the courses keywords and making sure it supports matching *n* amount of courses, we integrate the code to lookup course codes and recursively validate a course's prerequisites.

Due to limitations and the complexity of NUSMods data, not all courses exist and the source of truth sometimes conflicts. We decided to limit our modules to our modules offered this semester to facilitate testing and reducing the possible amount of edge cases. It is possible to change this in the future.

As our list of courses was originally gathered from moduleInfo and the prerequisites are gathered from courses in moduleList, there is a problem when prepopulating the database where the courses of the prerequisite are not created in the database as their original source does not contain the information. At that point in time we did not fully understand the difference between moduleInfo.json and moduleInformation.json on nusmods's api due to the dearth of information available. Hence we made the decision to limit it to only courses that appear in moduleList.json. This led to courses that are no longer offered in this semester to not be in our course selector. However we think this is one of the ways we can reduce breaking bugs in our validator.

A very major limitation of our validator is its inability to point out which part of a rule it did not validate, this is due to limits in how our validator is designed, where it attempts to reduce the

[4] https://sl.bing.net/eO1sWXXBlOm

expression to a single true or false. Hence it is unable to keep track or act like a stepper in pointing out which part of the conditions failed. To the validator, the rules are flat and linear with no awareness of groupings. It may be possible with more knowledge and learning[5] by the developers but at this stage of the project this was the fastest way to make a validator that can validate prerequisites.

Another major drawback of this validator is that due to the lack of documentation around NUS's prerequisite language, we do not know what keywords exist. Hence we have to rely on random sampling of the courses and trying to identify keywords. We eventually identify keywords which we are unable to process and filter them out. Courses with such prerequisites will automatically pass the validator.

```
# Remove any rules containing information we cannot process
if 'UNITS' in prereq_str or 'MUST_BE_IN' in prereq_str or 'COHORT_YEARS' in prereq_str \
    or 'SUBJECTS' in prereq_str or 'Certificate' in prereq_str or 'Graduate' in prereq_str:
    done += 1
    continue
```

Filtering of prerequisites with unsupported keywords

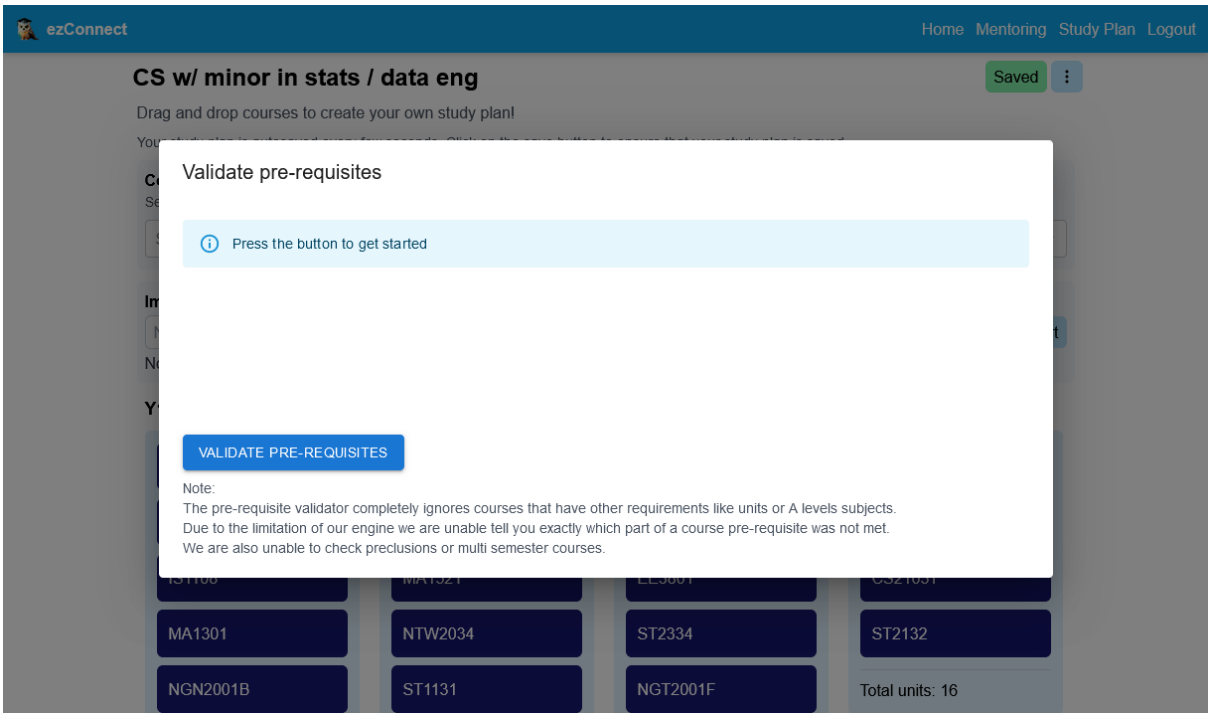
Unfortunately while this led to less errors, we estimated that we filtered out about 50% of courses with prerequisites. These prerequisites will automatically pass to ensure the evaluator can carry on with its operation. The onus is on the user to ensure their study plan is correct manually. Similarly we are unable to check for courses with preclusions.

There were other bugs that were not caught until user testing by sheer luck of the user entering that course into the study plan, for example some courses did not come in enclosing brackets around their rules and we overlooked that some courses can come without the 'PROG … THEN (…)' format. This was not found during our initial testing as there was over 1000 courses after filtering.

The validation can be an extremely expensive process, each request can take upwards of 300 ms to run. To achieve this speed, memoization was used extensively. As the validator checks through the prerequisites' prerequisite (a kind of depth first search), we cache evaluated courses for retrieval later, shortening time for later courses. This is also useful as some courses share prerequisites. The validator can also skip validating courses in the earlier semester as it starts from the latest semester (e.g. Year 2 Sem 2).

[5] https://nusmods.com/courses/CS4212/compiler-design

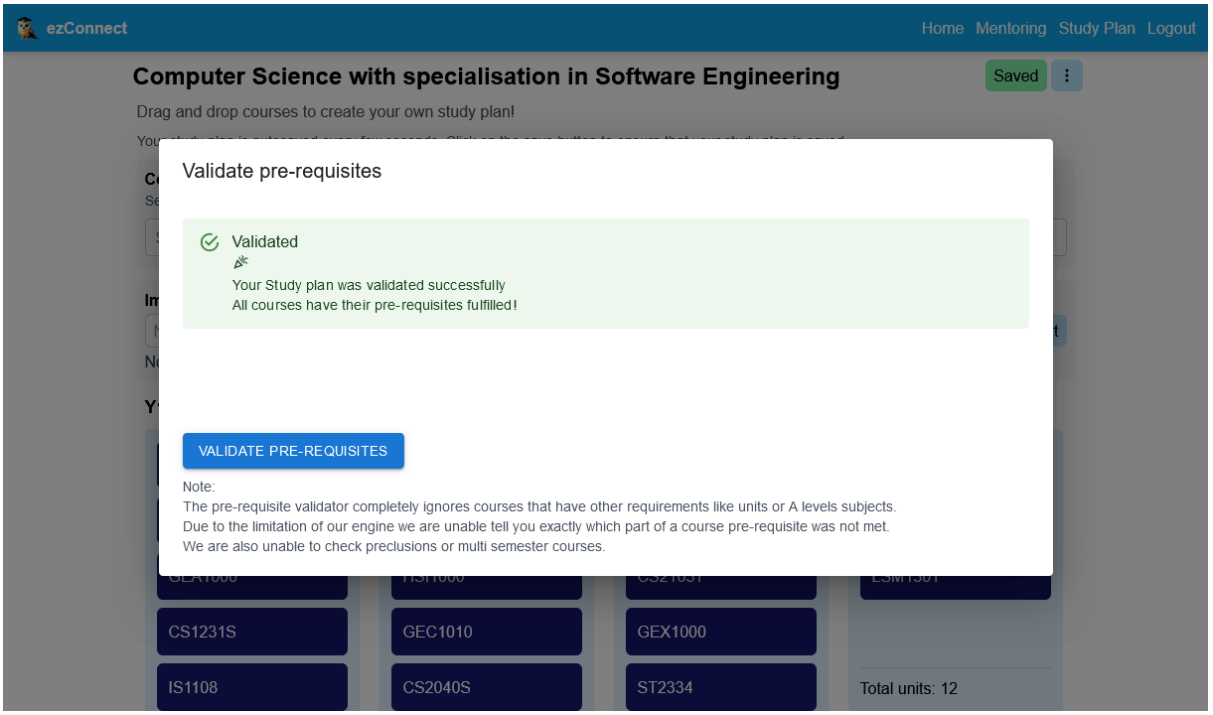
User interface



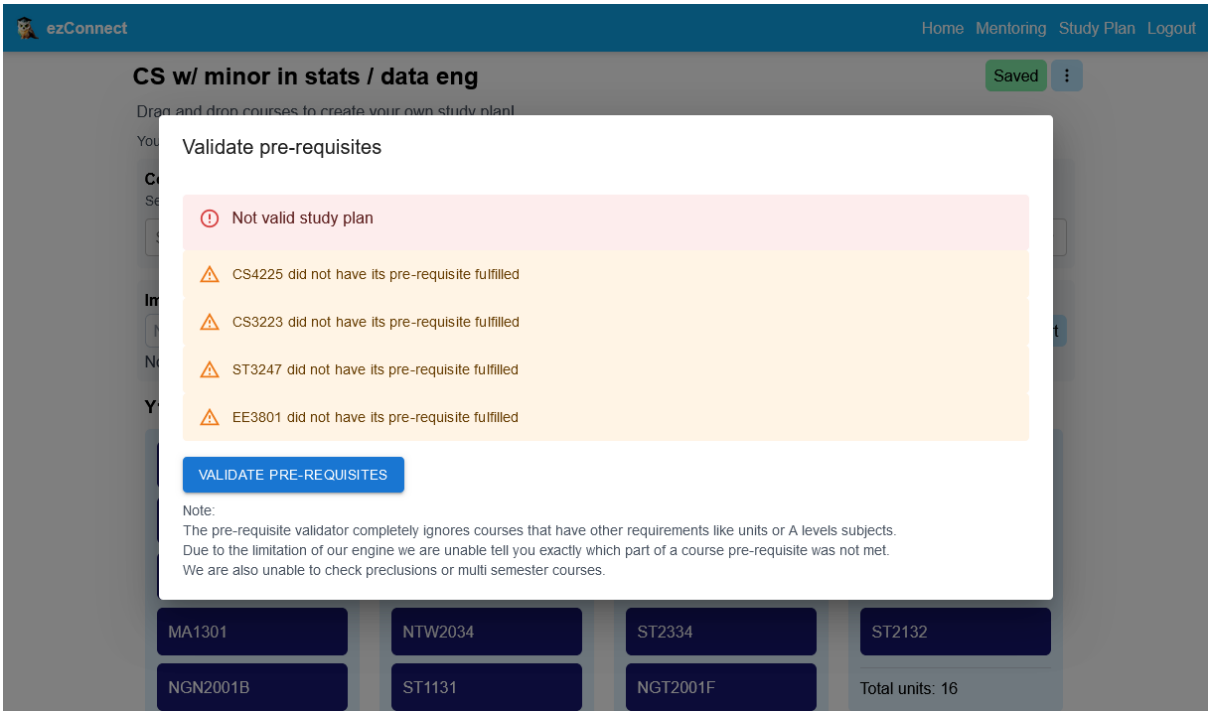
The validator modal

The user presses the '. . .' button on the top right to be presented with a drop down menu and they can select "Validate Prerequisites". When they click validate pre-requisites, the study plan is automatically updated with the backend and then validated.

On successful validation, bearing in mind the limitation mentioned above, a green info box is placed.



Successful validation



Unsuccessful validation

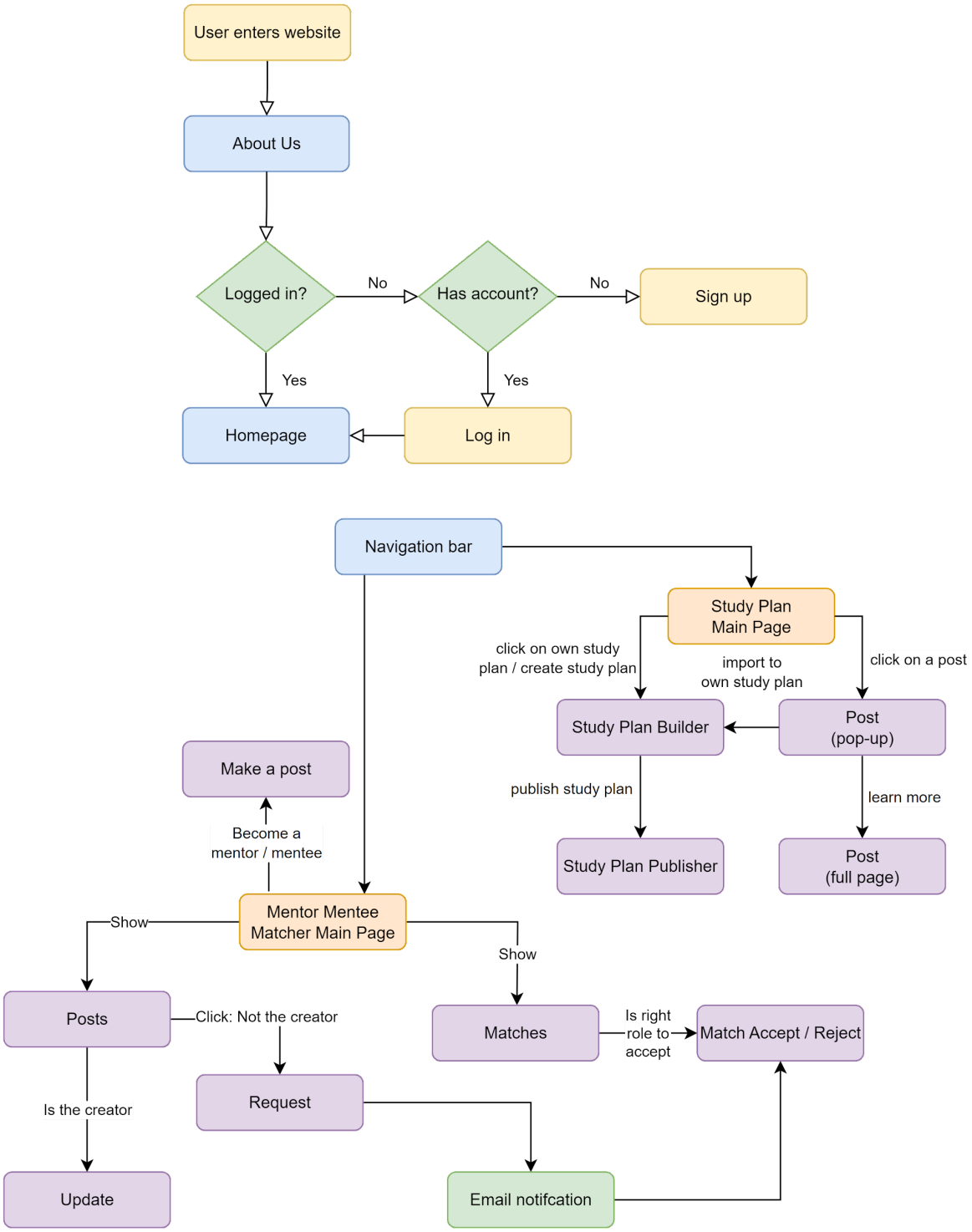
Potential improvements

The system can be extended to support Advanced Placement credits / exemptions and academic plans, by the creation of special graduation “courses” which contain programmed rules representing the requirements for graduation, e.g. GE pillars, common courses.

Navigation Flow

Users can enter the *About Us* page and the *Login* Page when they enter the webpage. They will be prompted to login if they go to any other page as the contents of the web application are restricted to NUS students.

From any page, users will be able to go to other main pages through the navigation bar.



Navigation Flow Diagram

Tech stack

Frontend	**Backend**
React (Javascript) Tailwind CSS Material UI	Python - Flask - Connexion - SQLAlchemy ORM
CI/CD	**Cloud infrastructure, databases**
GitHub Actions pyTest Eslint Playwright	Azure virtual machines, Cloudflare tunnel PostgreSQL Docker
Productivity	
Data cleaning: - Bing AI - ChatGPT Developer Productivity: - Built in swagger UI of Connexion for API testing	

Our tech stack uses Caddy, Flask + Connexion, Cloudflared, and Postgres, which are hosted in a Docker container on an Azure virtual machine (VM). The application is automatically deployed through an Ansible playbook when a push is made to a branch whose name contains the string ‘release’ or ‘ms’. The GitHub Actions workflow is explained in the DevOps section below.

Caddy serves as the web server for our static files and as a reverse proxy to the Python Flask Web API. The react app is built and then served statically.

Flask and Connexion enable us to develop RESTful APIs with built in documentation. We make use of Docker to package and deploy our application for consistency and scalability.

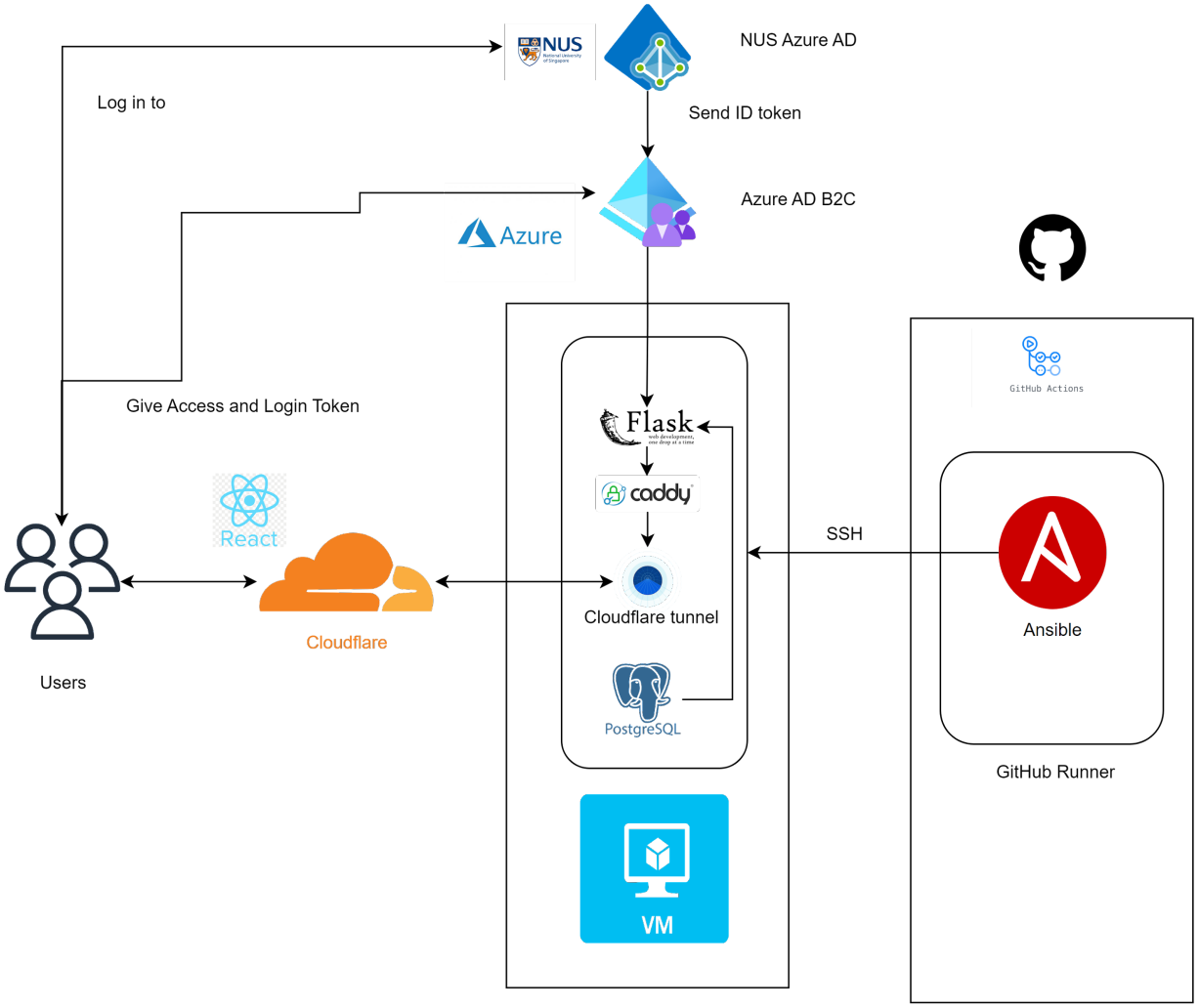
Cloudflared is a tunnelling daemon that adds an additional layer of security to our infrastructure. Traffic is sent through Cloudflare, a content delivery network which offers services to improve security and reduce latency. The only port exposed on the VM is 22 for secure shell connection from Ansible when the deployment is triggered by GitHub Actions.

User management is handled using Azure B2C, with NUS’s Azure Active Directory configured as an identity provider (IdP) for OpenID Connect. This allows users to authenticate and access our

application using their NUS credentials, providing a seamless experience while fulfilling user story 11 that other users are verified NUS students.

We had an issue initially where the server did not have enough RAM to compile the production build for the react frontend. We resolved this by setting up swap space on the virtual machine's SSD to increase available memory and passing it as a node option to tell Node to use more RAM, which allowed the build to succeed.

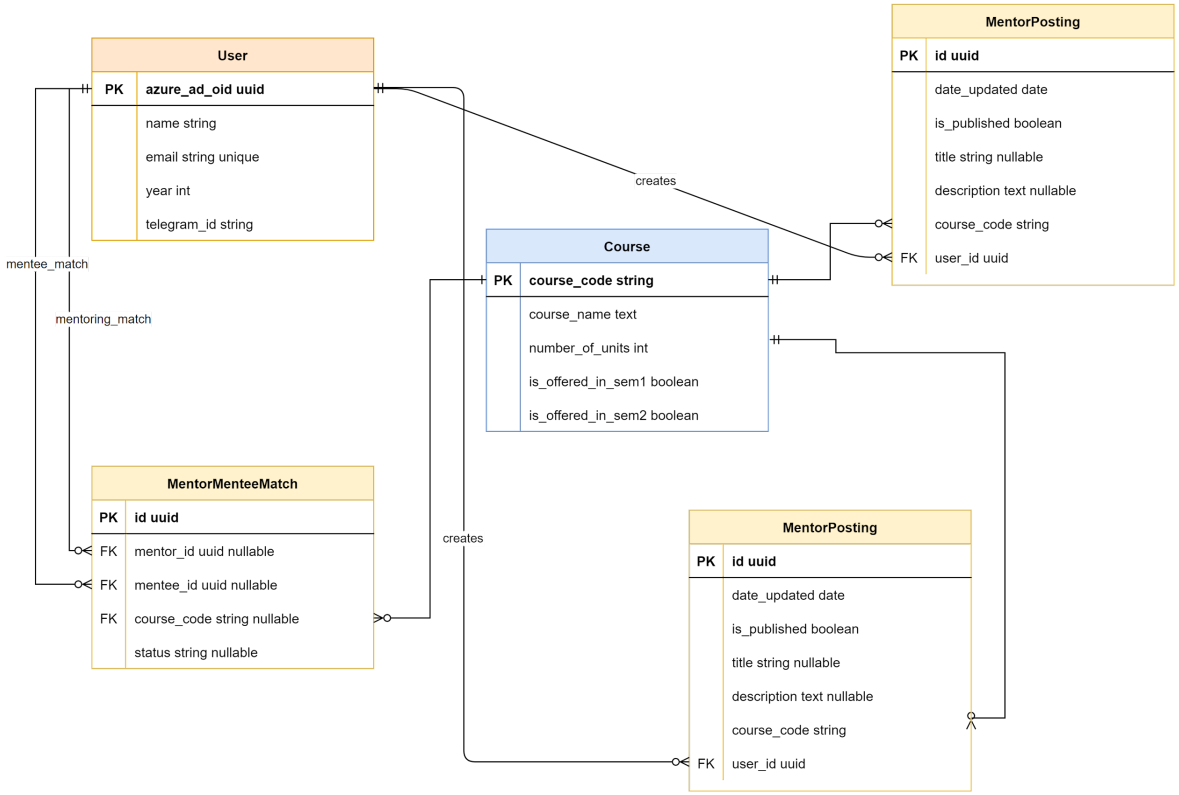
Architecture Diagram



Entity Relationship Diagram

We made an entity relationship diagram for our database to depict the relationship between the models in the database. This helped us to plan our models before implementing the models in the database. We included bridge tables which are meant for creating many-to-many relationships in the database. The entity relationship diagram gave us visual aid as we made the database. The information in the diagram also helped us to ensure that we implemented the necessary constraints in the database models.

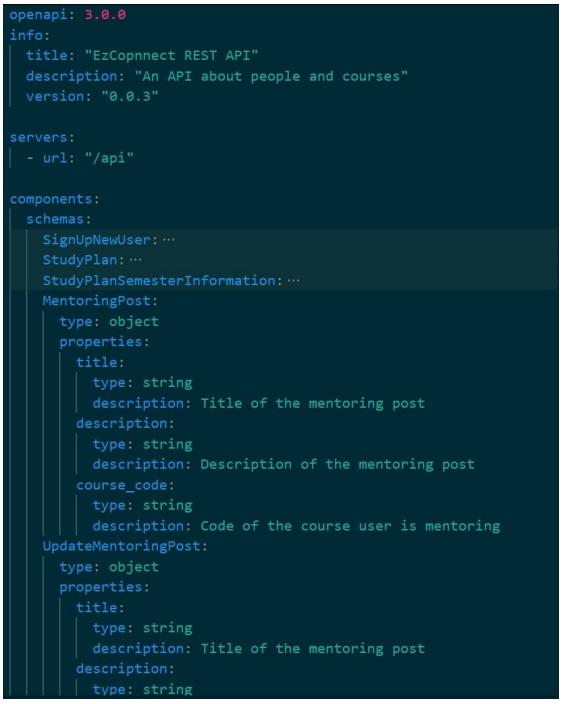
Study Plan Entity Relationship Diagram



Mentor Mentee Matcher Entity Relationship Diagram

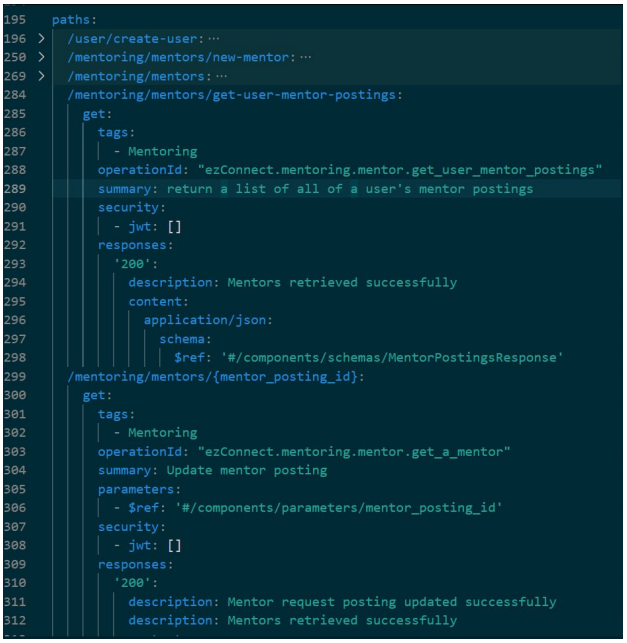
API Documentation

We used Swagger / OpenAPI to document our API. Swagger allowed us to document the API paths, the data to be passed into the API request, and the expected responses status codes, description and data returned.



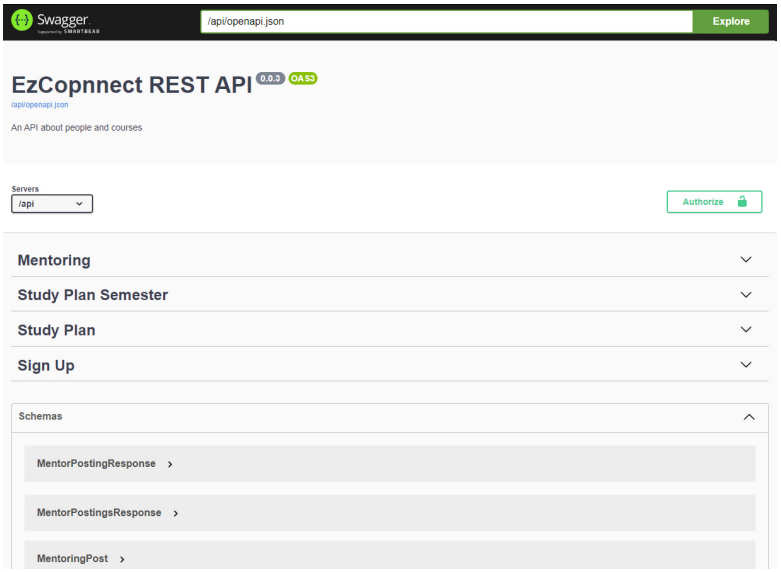
```yaml
openapi: 3.0.0
info:
  title: "EzCopnnect REST API"
  description: "An API about people and courses"
  version: "0.0.3"

servers:
  - url: "/api"

components:
  schemas:
    SignUpNewUser: ···
    StudyPlan: ···
    StudyPlanSemesterInformation: ···
    MentoringPost:
      type: object
      properties:
        title:
          type: string
          description: Title of the mentoring post
        description:
          type: string
          description: Description of the mentoring post
        course_code:
          type: string
          description: Code of the course user is mentoring
    UpdateMentoringPost:
      type: object
      properties:
        title:
          type: string
          description: Title of the mentoring post
        description:
          type: string
```

Swagger API Documentation

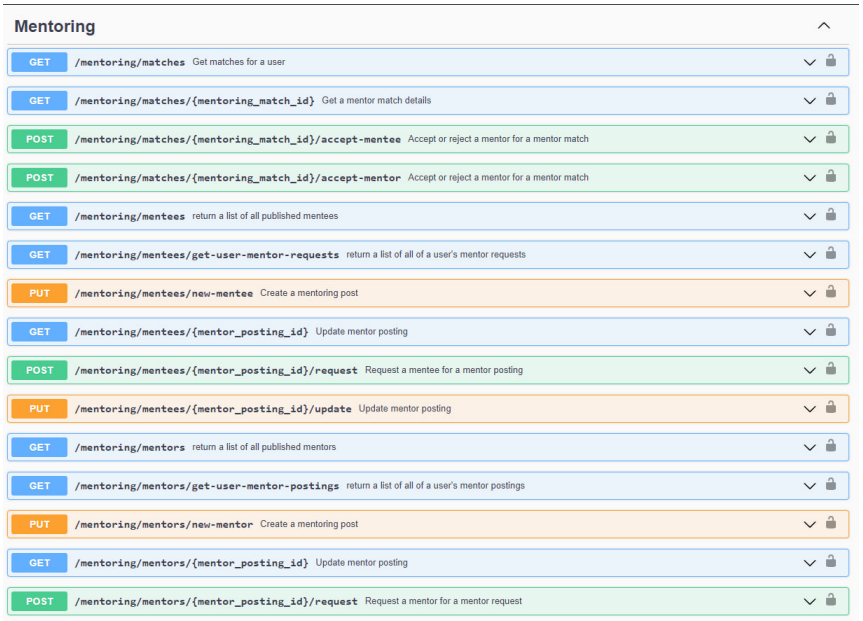


```yaml
paths:
  /user/create-user: ···
  /mentoring/mentors/new-mentor: ···
  /mentoring/mentors: ···
  /mentoring/mentors/get-user-mentor-postings:
    get:
      tags:
        - Mentoring
      operationId: "ezConnect.mentoring.mentor.get_user_mentor_postings"
      summary: return a list of all of a user's mentor postings
      security:
        - jwt: []
      responses:
        '200':
          description: Mentors retrieved successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/MentorPostingsResponse'
  /mentoring/mentors/{mentor_posting_id}:
    get:
      tags:
        - Mentoring
      operationId: "ezConnect.mentoring.mentor.get_a_mentor"
      summary: Update mentor posting
      parameters:
        - $ref: '#/components/parameters/mentor_posting_id'
      security:
        - jwt: []
      responses:
        '200':
          description: Mentor request posting updated successfully
          description: Mentors retrieved successfully
```

Swagger Specification API Paths

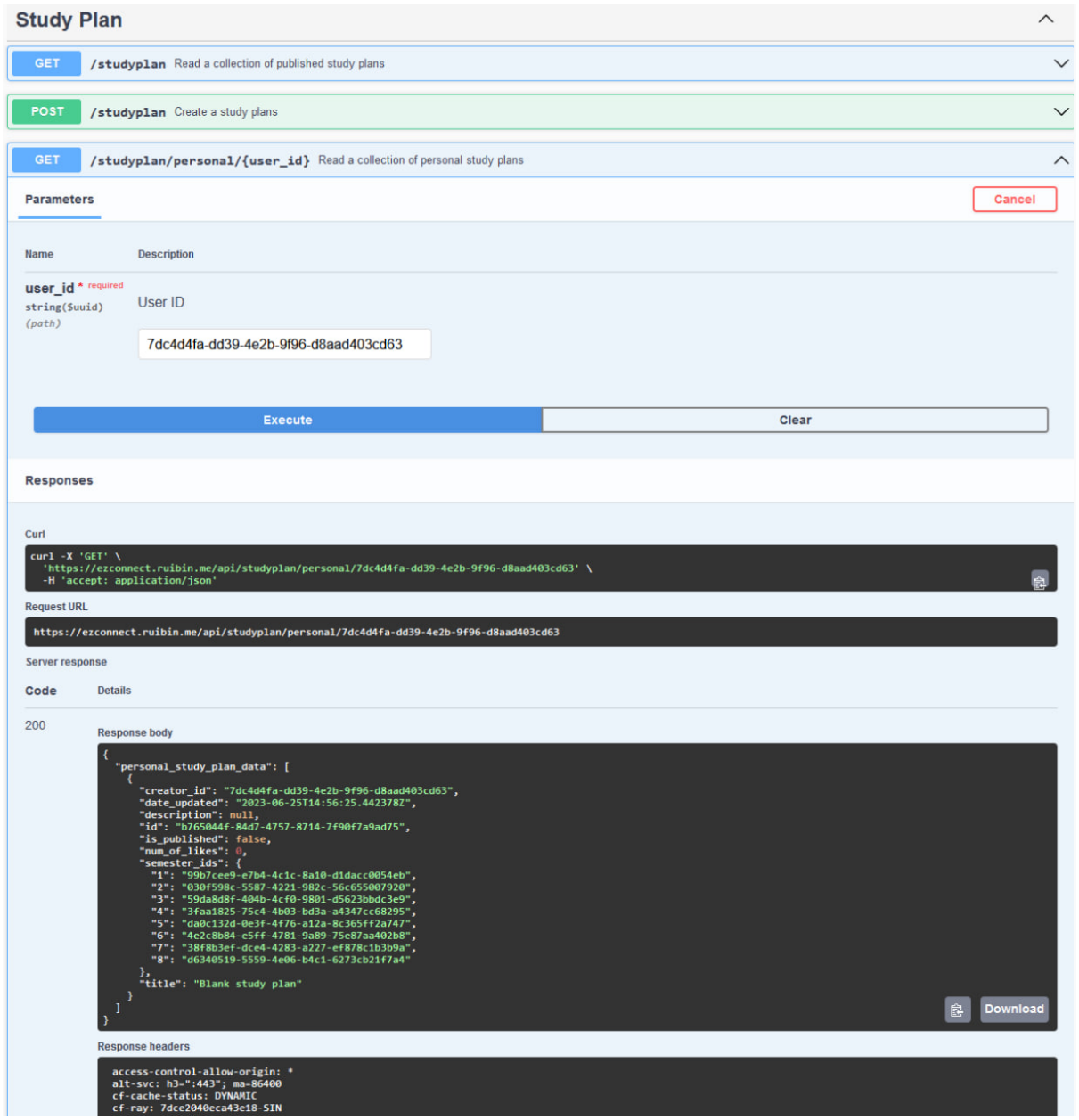


Swagger UI



Swagger UI Paths - Mentoring

Such documentation allows for easy reference to the API paths to be used in the frontend. Swagger also allows us to test the API calls directly in Swagger UI, reducing the need for an external platform. This allows us to debug issues with the functions used to implement the API paths before they are used in the frontend.



Swagger Testing

Software Engineering Practices

Don’t Repeat Yourself Principle

https://nus-cs2103-ay2223s1.github.io/website/se-book-adapted/chapters/principles.html#dry-principle

One key principle of software engineering is Don’t Repeat Yourself. It means implementation and values that are repeated should only be defined in one place. Defining the same thing multiple times in multiple places can lead to unintended bugs and inconsistency in the data.

Our project accomplishes this by ensuring commonly repeated or used functions are extracted into their own functions or components. For example, the component for the matches tab which was defined in an external component which was then added into the pages they are required. Changes made to the component can now be centralised.

Another example are values or constants used throughout the application. They are declared in environment files so they only need to be changed there for it to affect the entire application.

Last but not least we have the function for secureApi which is in charge of getting the JWT access token from the MSAL instance object and adding it to the authorization header when requests are made to our backend API. This reduces the need to write a custom fetch function and wrapper everywhere that we need to make an API call.

Reuse

https://nus-cs2103-ay2223s1.github.io/website/se-book-adapted/chapters/reuse.html

Similar to DRY, another principle is to adopt the practice of reusing, instead of re-inventing with authentication for example, we abstract certain components out to widely used platforms and libraries, in this case Azure AD B2C and MSAL.js libraries.

Similarly we made use of Material UI for our styling so we did not have to spend so much time fixing and correcting style errors when we could reference samples from the library documentation.

Code Organisation & Monorepo design & Separation of concern

We decided to adopt a monorepo approach to organising our source code as it allows a single repo to store the entire project and enables several benefits. Being able to store the entire project in one repository does not necessarily result in an unorganised or messy repository. We have a clear separation of concerns and adopted modularity in our code that enables our code to be easily segmented.

Our code mainly resides in three main folders, backend, frontend and infra. The names are pretty self-explanatory, the backend folder contains everything pertaining to the Flask backend, tests, as well as the scripts for pre-generating data for loading into the database. It also contains its own Docker Compose file for spinning up a local instance of the backend services for development work. Finally, it contains the environment file which specifies the version of Python modules that are needed for the project, which ensures the production environment and the developers are using the same versions.

Inside the backend folder, the ezConnect application is also split into separate modules each for different functions. For example authentication related code resides entirely in auth while mentoring and study_plan resides in different folders.

Frontend folder contains the react project along with the relevant tests and node packages. The packages themselves are not added into version control but their lock files are, ensuring a consistent development and production environment. Similar to backend, the frontend also split into application functions as study plans and mentoring are split into StudyPlan and Mentoring folders.

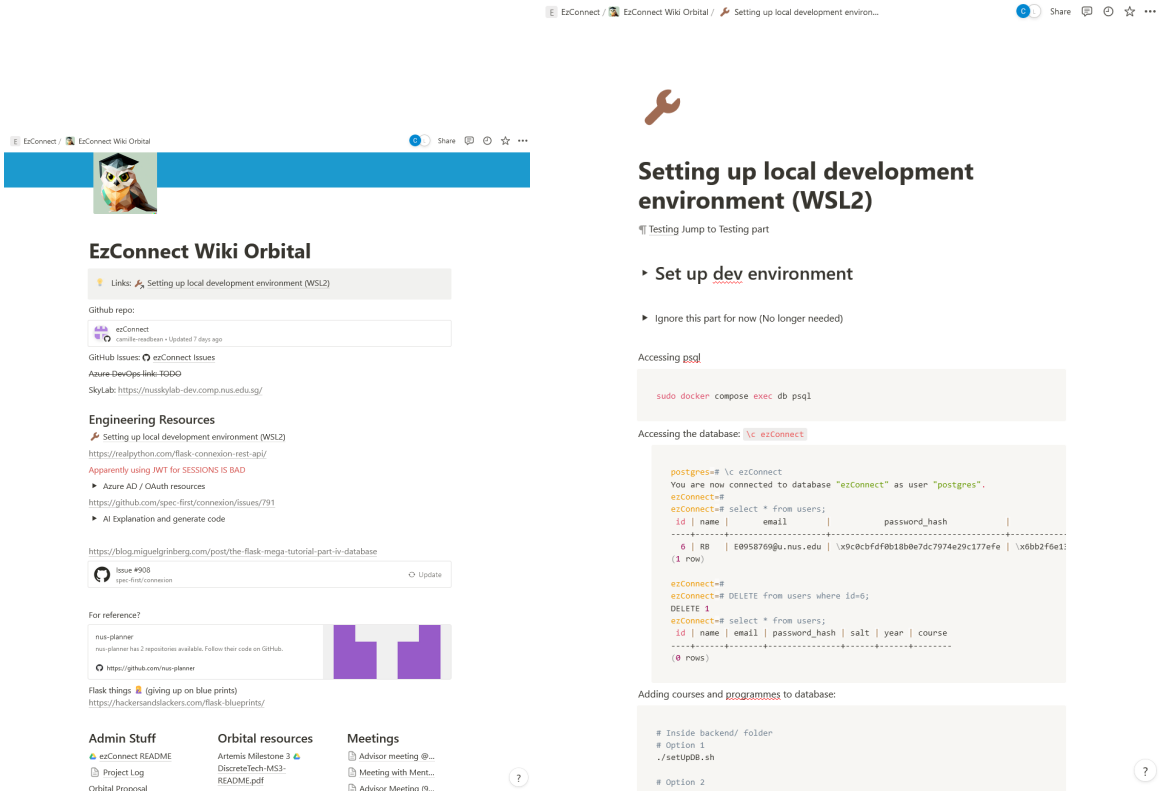
Finally infra contains the Ansible playbook along with other miscellaneous files for production like the caddy-config directory containing the config files for the production Caddy web server used in production.

Additionally there's a ‘.github’ folders which is needed to contain the actions workflow files. The monorepo also helps with CI/CD as the Action runner has everything it needs with regards to the project in one repository.

Having all these in my repo allows the developers to stay up to date with changes happening in other parts of the project. Adoption of separation of concern allows developers to work in the same git repo on different parts of the project at the same time and easily merge them together when ready. We will discuss more on how the team handles branching and merging in a separate section.

Project documentation

https://nus-cs2103-ay2223s1.github.io/website/se-book-adapted/chapters/documentation.html

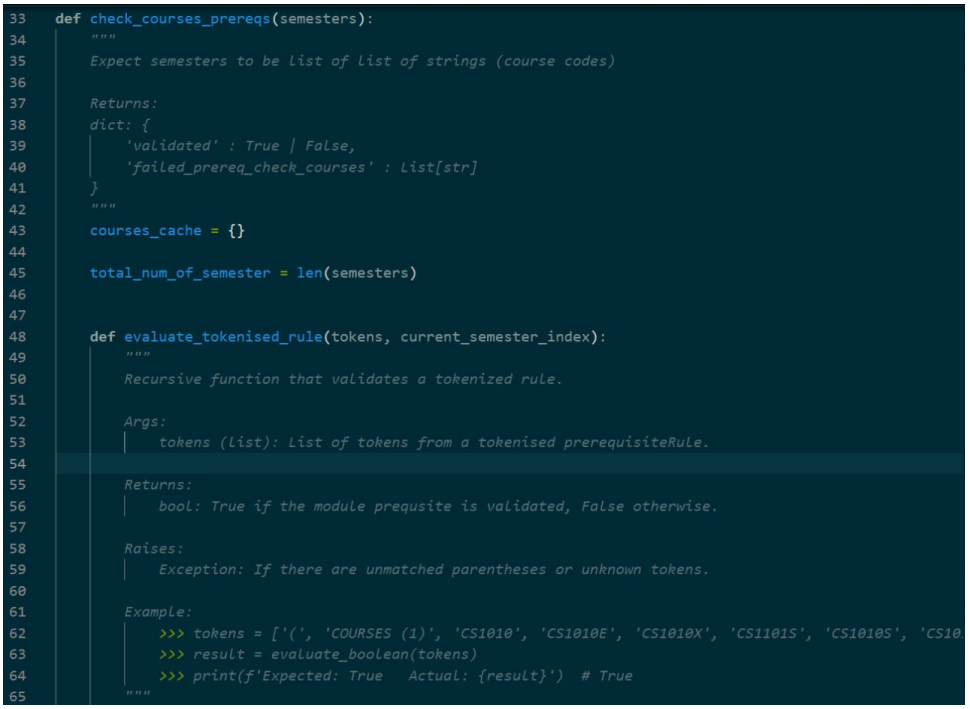


Left: Project Wiki on notion
Right: a page on the engineering wiki on notion

The team utilised a wiki on notion for the centralising resources and brainstorming notes regarding the project. Engineering notes were also kept there like How-Tos for setting up the developer environment or recreating the database for manual deployment if the automated build fails.

Links to API references were also kept for quick look up. The documenting of thought processes and tutorials behind parts of the program help developers follow along on what one developer has done. It ensures that the developers are at least on the same level of understanding especially when it concerns learning something new. It also allows for easier backtracking to check steps and debugging especially when debugging something. This was very helpful when debugging our Azure AD B2C due to the vast amount of information and steps at attempted fixes.

On top of the generated API documentation, we also documented particularly complex code with documentation that can be displayed in the IDE like Intellisense. This helps in development when one developer has to use functions or components created by another.



```python
def check_courses_prereqs(semesters):
    """
    Expect semesters to be list of list of strings (course codes)

    Returns:
    dict: {
        'validated' : True | False,
        'failed_prereq_check_courses' : List[str]
    }
    """
    courses_cache = {}

    total_num_of_semester = len(semesters)


    def evaluate_tokenised_rule(tokens, current_semester_index):
        """
        Recursive function that validates a tokenized rule.

        Args:
            tokens (list): List of tokens from a tokenised prerequisiteRule.

        Returns:
            bool: True if the module prequsite is validated, False otherwise.

        Raises:
            Exception: If there are unmatched parentheses or unknown tokens.

        Example:
            >>> tokens = ['(', 'COURSES (1)', 'CS1010', 'CS1010E', 'CS1010X', 'CS1101S', 'CS1010S', 'CS10
            >>> result = evaluate_boolean(tokens)
            >>> print(f'Expected: True   Actual: {result}')  # True
        """
```

Example pydoc for course validator

```javascript
/**
 * Sends secure request to backend by getting access token and adding to header.
 * @param {import("@azure/msal-browser").IPublicClientApplication} instance
 * @param {string} method The HTTP method for the request (e.g., "GET", "POST", "PUT", etc.).
 * @param {string} url The path of the backend API endpoint.
 * @param {Object} req_body The request body to be sent with the request.
 * @returns {Promise<JSON>} A promise that resolves with the JSON response from the backend.
 */
export function secureApiRequest(instance, method, url, req_body) {
```

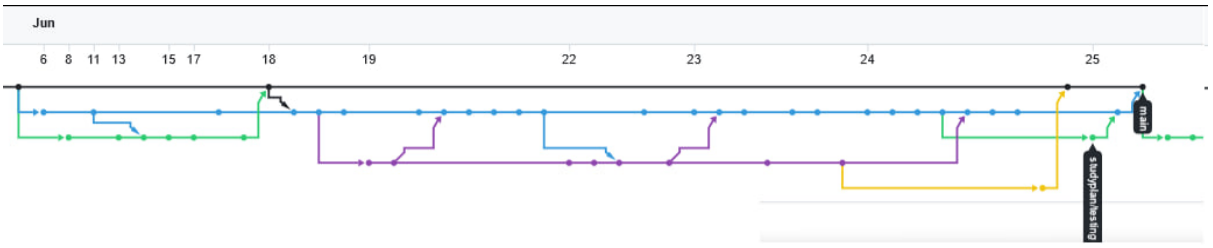
Example javascript docs for secureApiRequest that is frequently reused

Version Control: Branching, pull requests and code reviews

Our project made extensive use of version control for collaboration between developers. We use Git with GitHub as our remote Git repository and project management tool. Our usage of these tools will be elaborated more in the later section.

Firstly we made extensive use of branching as required by the course[6]. Our branching workflow is best described as feature branching. Each branch created is dedicated to a specific feature or set of features. The team frequently merge from another branch to reduce the chances of merge conflicts down the line though this is already unlikely to happen due to consistent separation of concerns.

The main branch is write protected and requires a pull request to be created with at least one review before it can be written to.

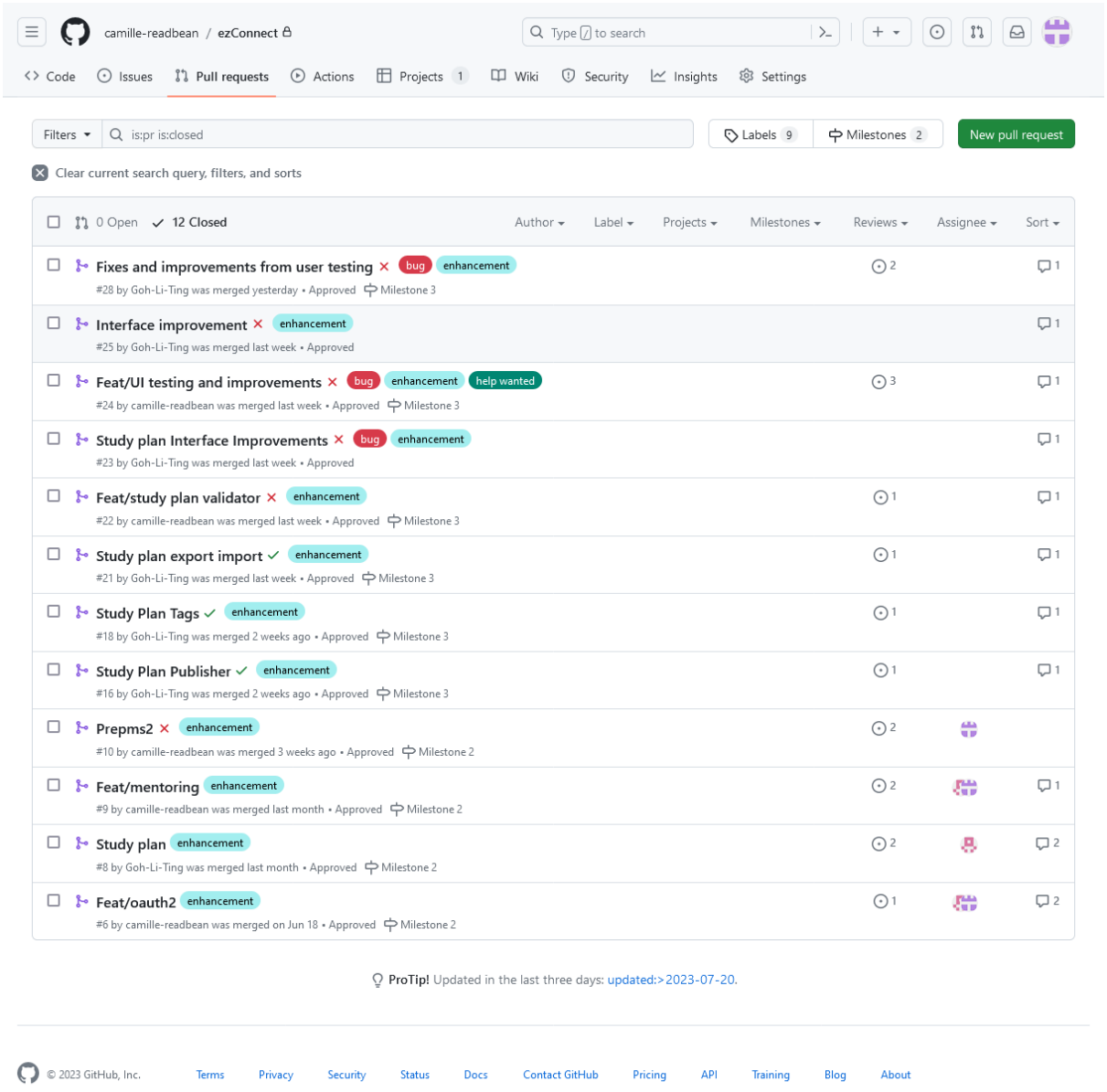


Project's Git Branches in June

To prepare for release, we conduct a feature freeze before release and create a release branch. Any new updates or critical patches will be tested and then pushed to a release branch. The release branch is the only branch configured for deployment so patches will be automatically pushed to live. This is quite important if there are critical bugs or minor fixes discovered.

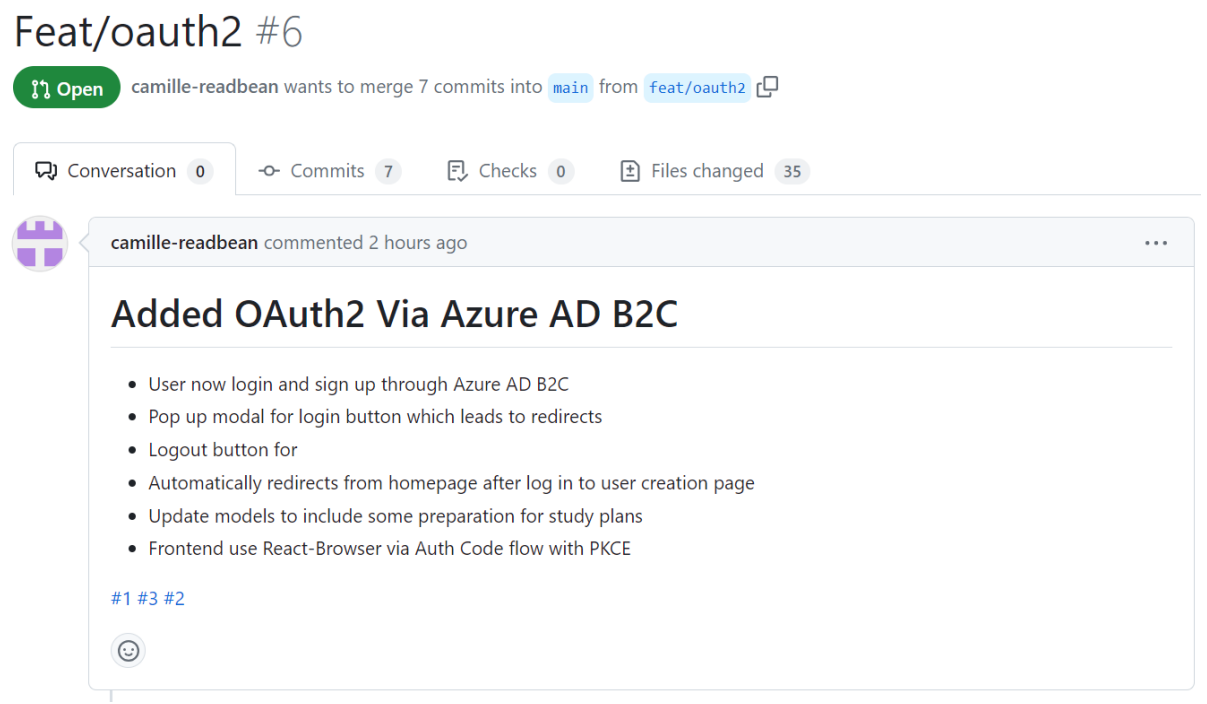
6 https://docs.google.com/presentation/d/1r1MwkuUeSY-294KVZThXmuQhcyxi2iSHxqc-LPEp7Qc/edit#slide=id.g12755c2ac1a_1_55

Pull requests and code reviews

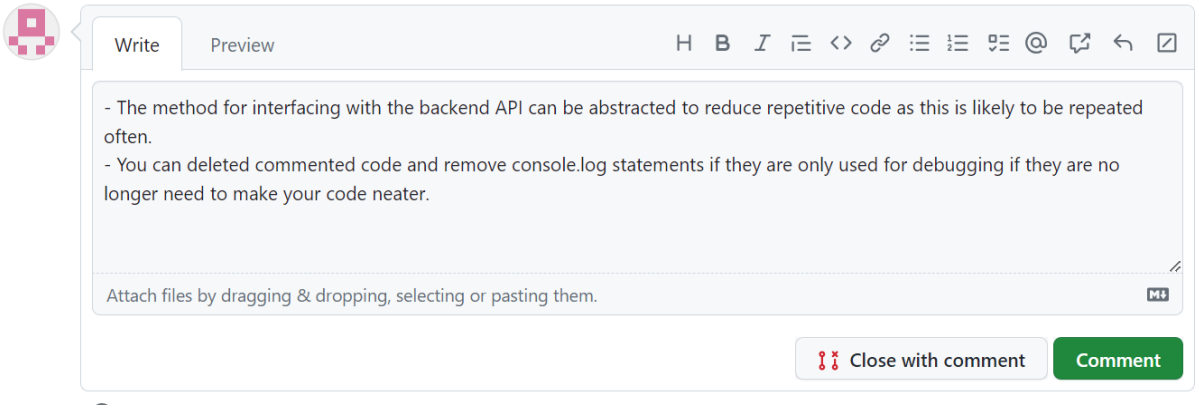


Pull requests in the project

Due to the numerous branches created as a result of flows, we inadvertently also have many pull requests. Each pull request is approved by the other development that did not create the pull request. This ensures that each other is kept in the loop regarding the project which is feasible as the project only has 2 developers.

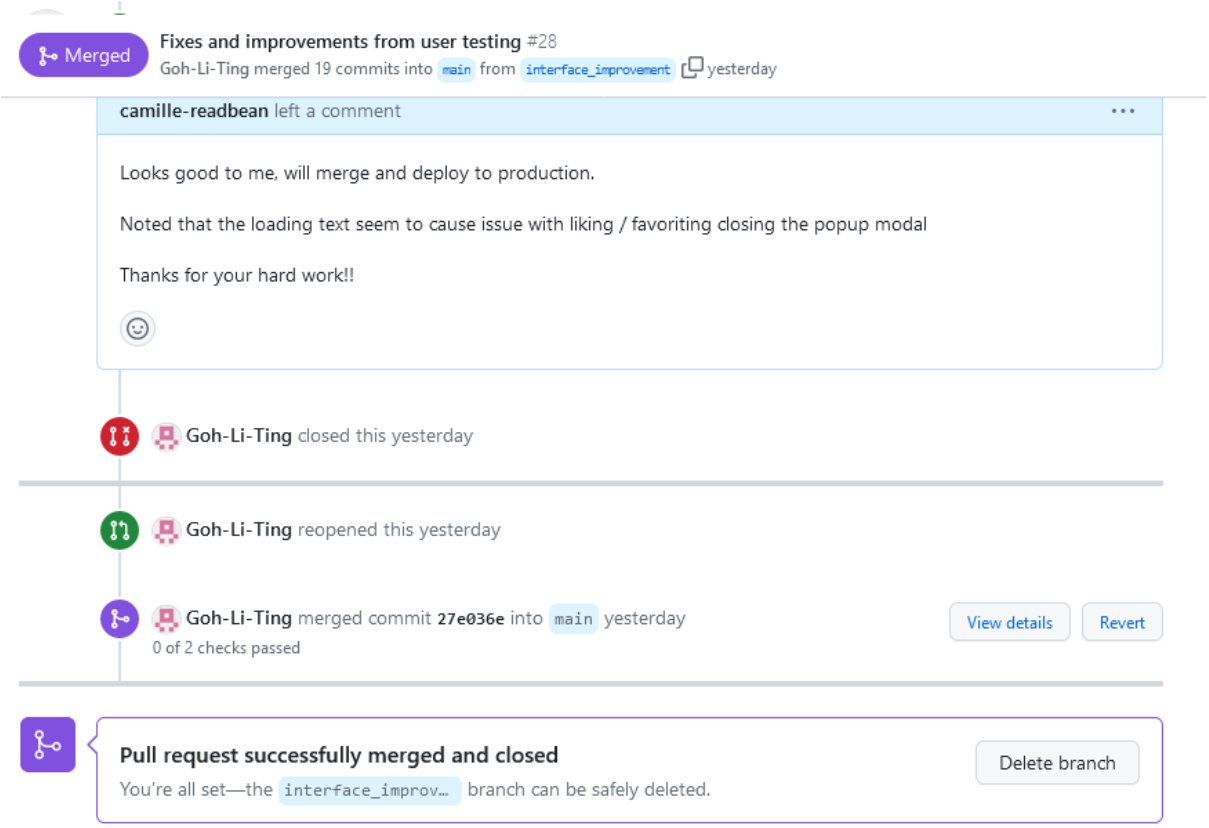


Pull Request Example



Review Example

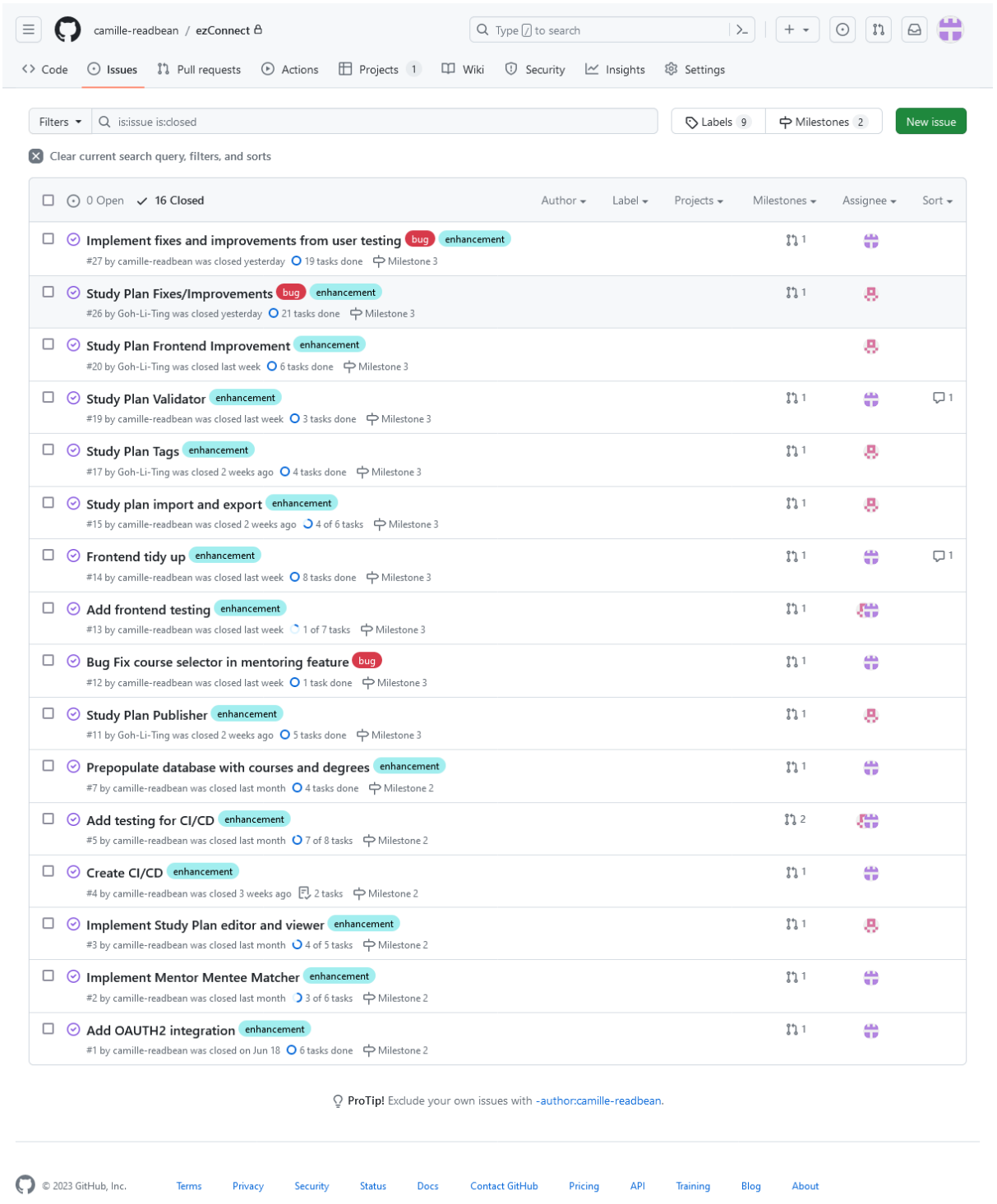
Code reviews also enforced a certain degree of coding standard, like modularity where repeated components are broken out and removing unused code. The project also attempts to stick to PEP8 for Python code and for Javascript Google's recommended Javascript styling. These are enforced using ruff and eslint in Github Actions.



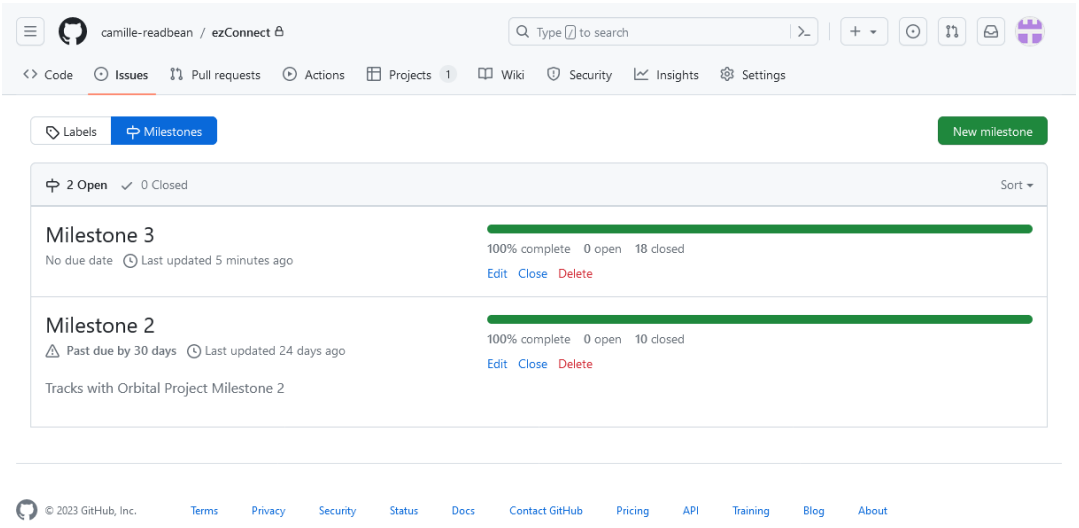
Code review on a pull request and subsequent merge commit

Git Issues, Projects board and Sprint Planning

We use GitHub issues to track features and bugs in our project. Each issue often corresponds to a set of features to be implemented. This allows for some advanced project management features. This helps us to be clear about the tasks for each sprint.

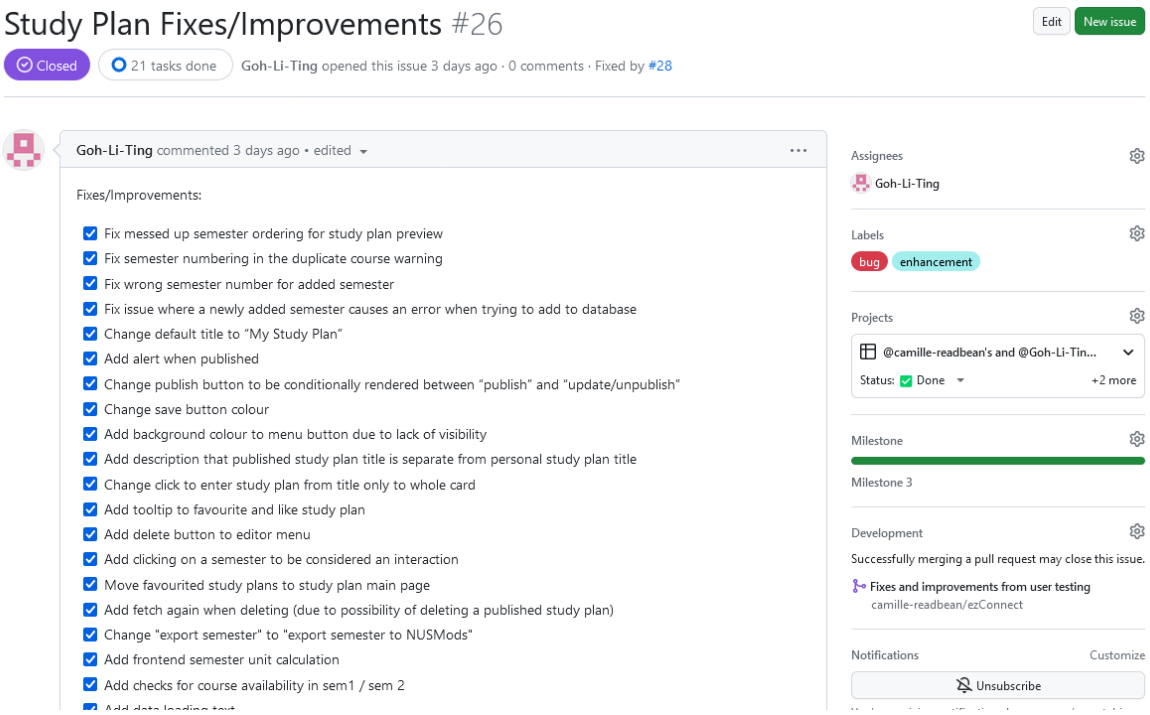


GitHub Issues



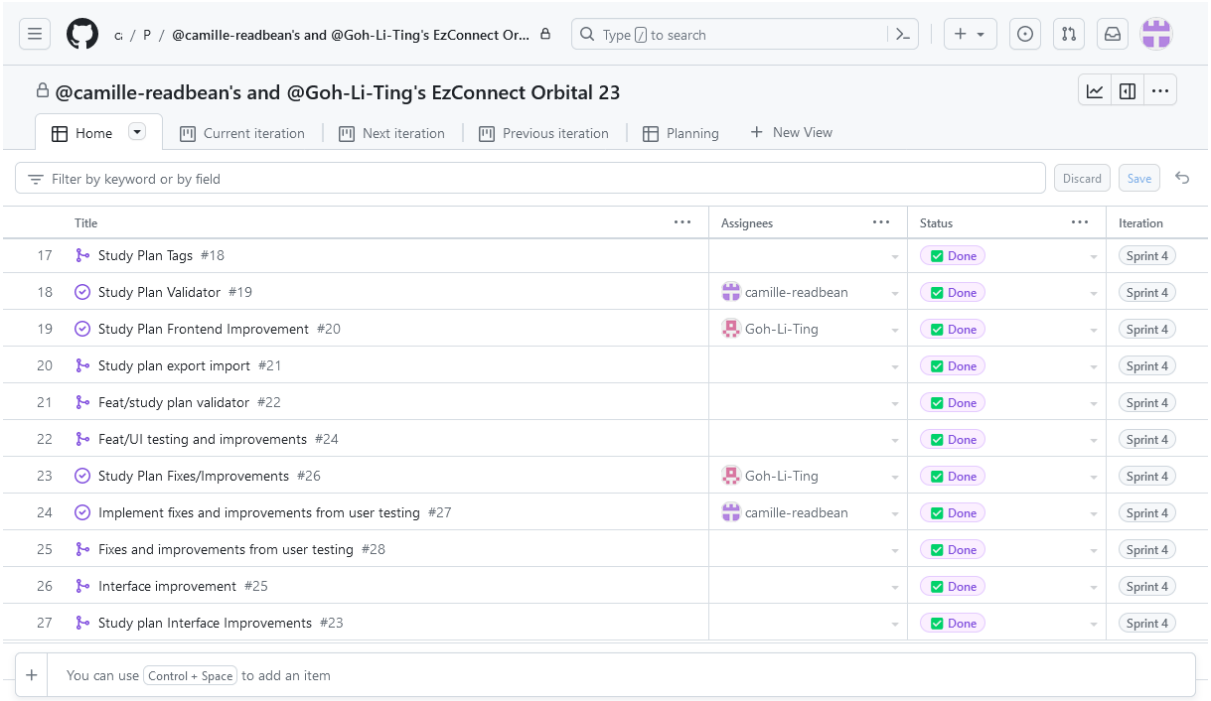
Milestones for tracking issues

These issues are mapped to milestones that track with the milestone of the Orbital Project. This helps us in organising issues and tasks, and assigning tasks. In each issue we can create checkboxes to indicate tasks that need to be done for that issue. We can also specify which branch is assigned to this issue which helps in understanding what different branches are for on top of their names.

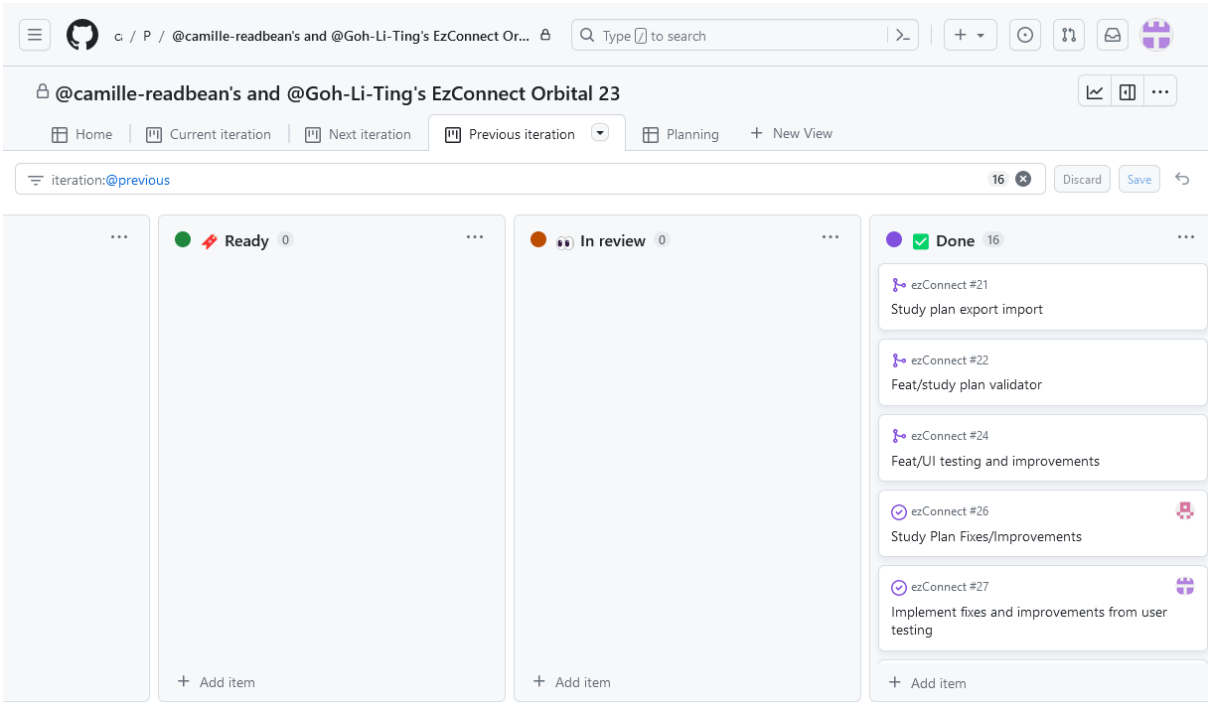


Issue example

Integrating this feature nicely is GitHub's projects which allow us to organise issues and pull requests onto a board. The board can be divided into different sections such as in progress, backlog, in review.



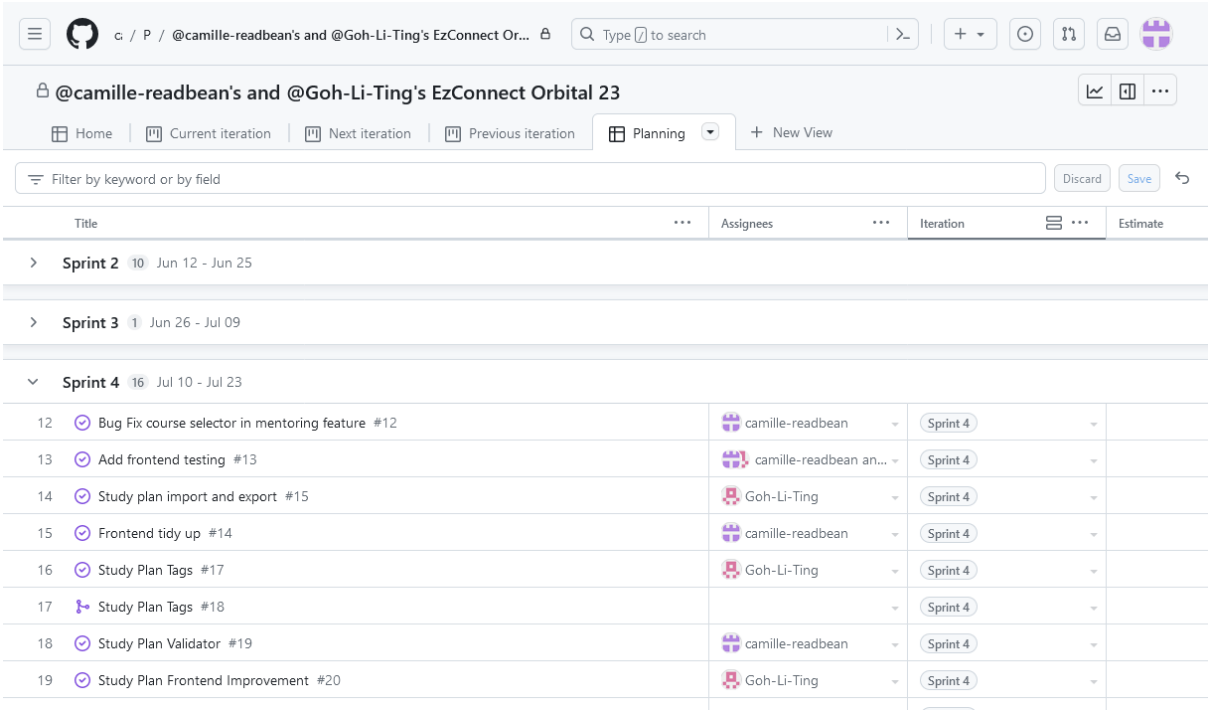
GitHub project list view



Just before Milestone 3

GitHub Project has the ability to create iterations as well. This greatly supports our ability to plan our project in 2 weeks sprints which is a requirement for Artemis level of achievement.

If an issue is backlogged we can shift it to the backlog section of the next iteration.



Sprint planning view

GitHub issues and pull requests can be added to projects, which are then shown on the project view as well. It helps the team track which feature was completed in which sprint which can aid us when we update the project log.

Most importantly it allows a visual representation of a task so we can tell if a task have been stuck or not updated for a period of time.

DevOps: Continuous Integration and Continuous Delivery/Deployment

Continuous Integration also called CI is a critical part of the modern developer's toolkit. It ensures that bugs are caught early on in development, reducing the complexity needed to deal with them or remove them in a later stage where it might be too late due to their entanglement in more and more moving parts. Automated testing is a key part of this process in ensuring the quality of our code.

Just as Continuous integration helps developers be better programmes, continuous delivery / deployment reduces the complexity and streamline the process of getting the higher quality code to the clients. This involves the automated testing of code, building of artefacts and the deployment of them to production systems.

Our project leverages multiple technologies to provide us with the best experience for developer operations to produce quality code and reduces operational headaches. Although there are significant upfront costs in setting up these systems they greatly reduce the time needed to debug deployment or worrying about whether new codes will unwittingly affect existing user flows.

Containerisation / virtual environments

Our project makes extensive use of *Conda* and *Docker* to standardise developer environments. Dependencies are installed from and stored in lock files like *environment.yml, requirements.txt*, for Python modules or *package-lock.json* for NPM packages.

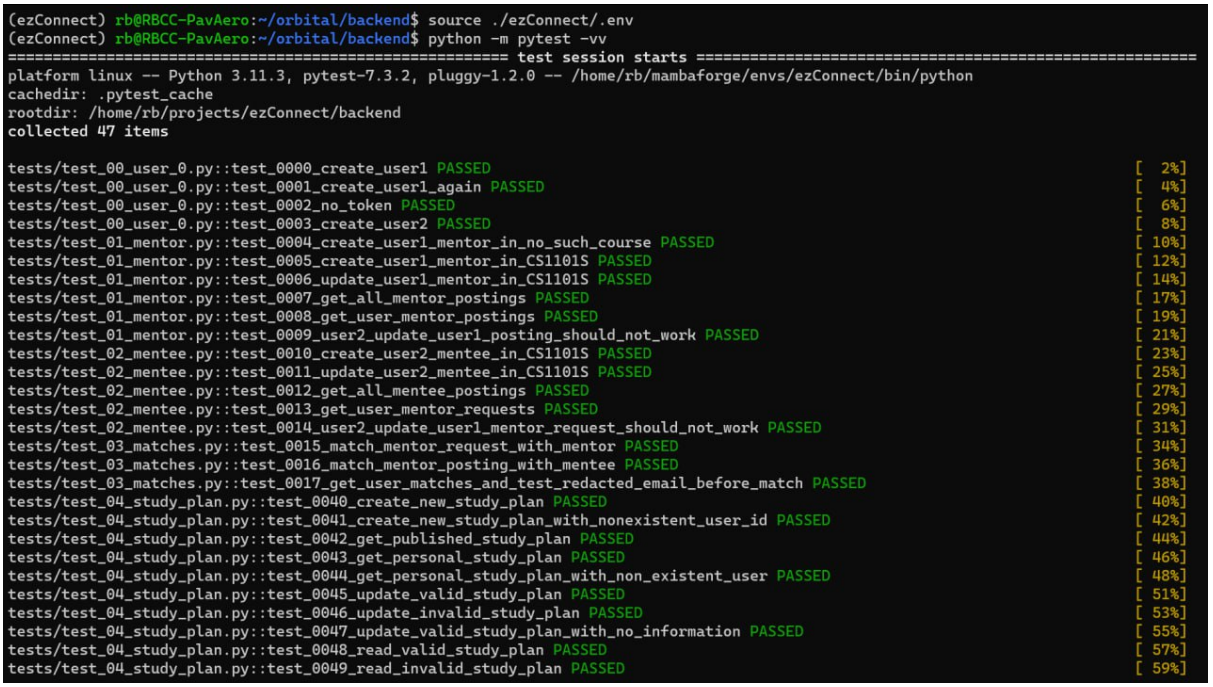
This goes a long way in making issues reproducible when every developer has the same packages and libraries in their systems, isolating errors from developer misconfiguration. CI/CD will use these lockfiles to install the exact same packages and libraries in production when they build the container for production as these files are stored in version control as well.

Storing these files in version control also allows software engineering tools like dependency checkers to check our project dependency for vulnerabilities. It also allows easy roll back and tracking of changes to our dependencies as the project evolves.

Integration and Unit Testing with pytest, ruff, GitHub Actions and playwright

To increase confidence in our code, we utilise integration and unit testing along with code linting in our project. Tests were written for the backend with pytest which test the functionality and correctness of the backend API. These tests also acted as a regression test, making sure future changes do not inadvertently cause the existing features to break.

We manage to achieve 84% code coverage as of milestone 2, giving us confidence in the backend api. Testing also revealed and fixed some bugs that slipped through developer testing. Sadly due to time constraints we were not able to update the tests for milestone 3.



Example of tests running on developer machine - non automated

```
(ezConnect) rb@RBCC-PavAero:~/orbital/backend$ coverage report --omit="*/test*,conftest.py"
Name                                             Stmts   Miss  Cover
--------------------------------------------------------------------
ezConnect/__init__.py                                0      0   100%
ezConnect/app.py                                    34      4    88%
ezConnect/auth/__init__.py                           0      0   100%
ezConnect/auth/login.py                             39     15    62%
ezConnect/auth/signup.py                            32      5    84%
ezConnect/config.py                                 16      0   100%
ezConnect/mentoring/matches.py                      89     16    82%
ezConnect/mentoring/mentee.py                      103     28    73%
ezConnect/mentoring/mentor.py                      103     29    72%
ezConnect/models.py                                122      5    96%
ezConnect/study_plan/study_plan_semester.py         75      2    97%
ezConnect/study_plan/studyplan.py                   57      1    98%
ezConnect/utils/__init__.py                          0      0   100%
ezConnect/utils/emailer.py                           8      1    88%
ezConnect/utils/exceptions.py                        4      0   100%
--------------------------------------------------------------------
TOTAL                                              682    106    84%
(ezConnect) rb@RBCC-PavAero:~/orbital/backend$
```

84% coverage of backend code

For test automation, we utilised GitHub actions where we set up a workflow which triggered every push to a branch. This helps developers develop safely and robustly as they can quickly catch errors in the system with every commit and push.

Our test cases are designed to mimic the flow of users interacting with our systems. For example test 0000 to test 0003, creates 2 users in the system, and ensures a user cannot accidentally sign up and create an account twice. Test 0002 ensures a token is checked and required before creating an account.

Tests like 0011 to 0017 ensure the correctness of the study plan feature, that mentees are able to see their requests and matches do not reveal emails before they are accepted.

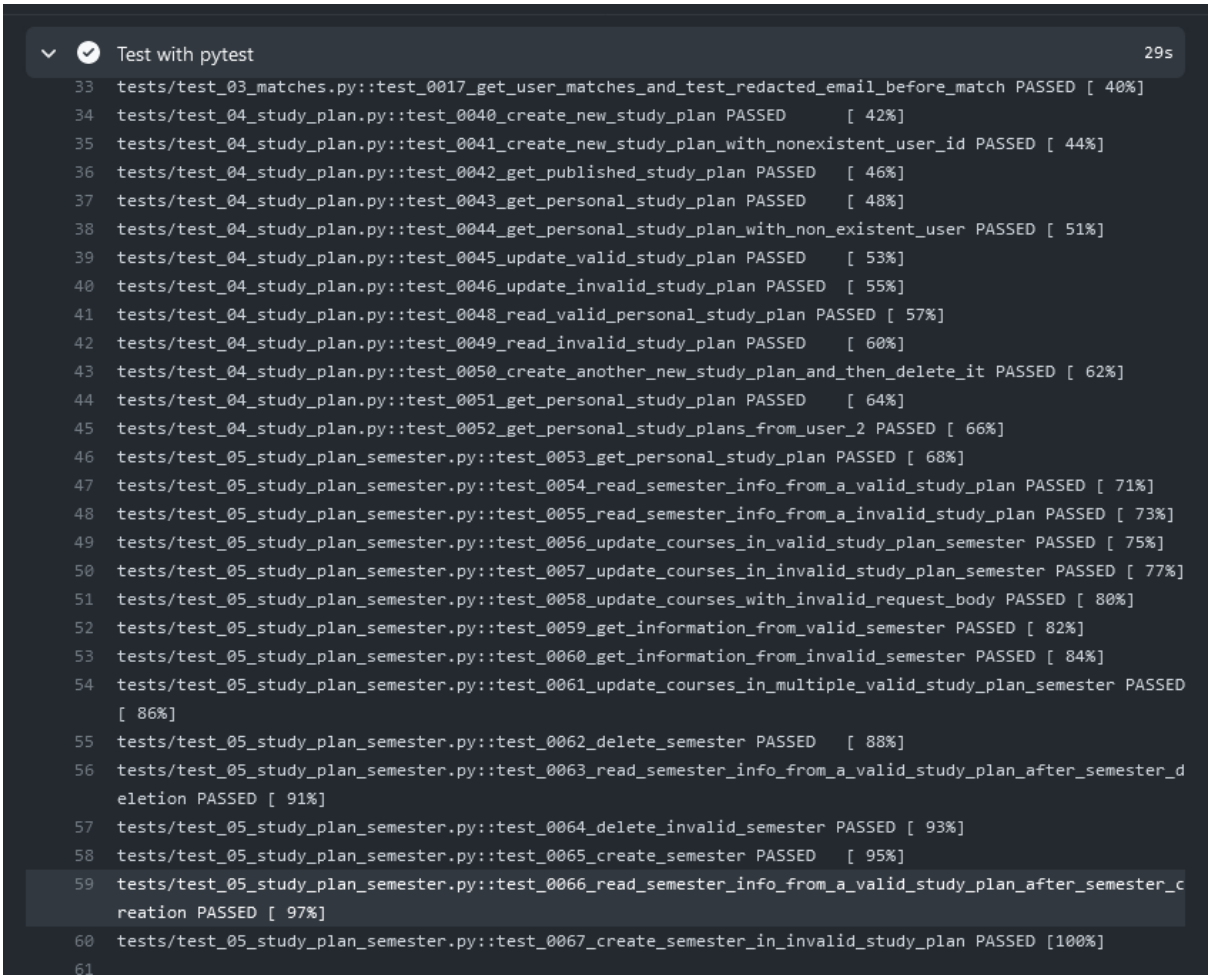
Our test cases are comprehensive and attempt to check for edge cases.

Our test cases started failing on July 22 2023 due to the built-in test tokens expiring and the addition of the broken playwright test, unfortunately, we did not have time to fix these errors as we were working on updating the interface from the user testing feedback.

Test case name	Date last passed	Passed?
test_0000_create_user1	Jul 17 2023	✅
test_0001_create_user1_again	Jul 17 2023	✅
test_0002_no_token	Jul 17 2023	✅
test_0003_create_user2	Jul 17 2023	✅
test_0004_create_user1_mentor_in_no_such_course	Jul 17 2023	✅
test_0005_create_user1_mentor_in_CS1101S	Jul 17 2023	✅

test_0006_update_user1_mentor_in_CS1101S	Jul 17 2023	✅
test_0007_get_all_mentor_postings	Jul 17 2023	✅
test_0008_get_user_mentor_postings	Jul 17 2023	✅
test_0009_user2_update_user1_posting_should_not_work	Jul 17 2023	✅
test_0010_create_user2_mentee_in_CS1101S	Jul 17 2023	✅
test_0011_update_user2_mentee_in_CS1101S	Jul 17 2023	✅
test_0012_get_all_mentee_postings	Jul 17 2023	✅
test_0013_get_user_mentor_requests	Jul 17 2023	✅
test_0014_user2_update_user1_mentor_request_should_not_work	Jul 17 2023	✅
test_0015_match_mentor_request_with_mentor	Jul 17 2023	✅
test_0016_match_mentor_posting_with_mentee	Jul 17 2023	✅
test_0017_get_user_matches_and_test_redacted_email_before_match	Jul 17 2023	✅
test_0040_create_new_study_plan	Jul 17 2023	✅
test_0041_create_new_study_plan_with_nonexistent_user_id	Jul 17 2023	✅
test_0042_get_published_study_plan	Jul 17 2023	✅
test_0043_get_personal_study_plan	Jul 17 2023	✅
test_0044_get_personal_study_plan_with_non_existent_user	Jul 17 2023	✅
test_0045_update_valid_study_plan	Jul 17 2023	✅
test_0046_update_invalid_study_plan	Jul 17 2023	✅
test_0048_read_valid_personal_study_plan	Jul 17 2023	✅
test_0049_read_invalid_study_plan	Jul 17 2023	✅
test_0050_create_another_new_study_plan_and_then_delete_it	Jul 17 2023	✅
test_0051_get_personal_study_plan	Jul 17 2023	✅
test_0052_get_personal_study_plans_from_user_2	Jul 17 2023	✅
test_0053_get_personal_study_plan	Jul 17 2023	✅
test_0054_read_semester_info_from_a_valid_study_plan	Jul 17 2023	✅

test_0055_read_semester_info_from_a_invalid_study_plan	Jul 17 2023	✅
test_0056_update_courses_in_valid_study_plan_semester	Jul 17 2023	✅
test_0057_update_courses_in_invalid_study_plan_semester	Jul 17 2023	✅
test_0058_update_courses_with_invalid_request_body	Jul 17 2023	✅
test_0059_get_information_from_valid_semester	Jul 17 2023	✅
test_0060_get_information_from_invalid_semester	Jul 17 2023	✅
test_0061_update_courses_in_multiple_valid_study_plan_semester	Jul 17 2023	✅
test_0062_delete_semester	Jul 17 2023	✅
test_0063_read_semester_info_from_a_valid_study_plan_after_semester_deletion	Jul 17 2023	✅
test_0064_delete_invalid_semester	Jul 17 2023	✅
test_0065_create_semester	Jul 17 2023	✅
test_0066_read_semester_info_from_a_valid_study_plan_after_semester_creation	Jul 17 2023	✅
test_0067_create_semester_in_invalid_study_plan	Jul 17 2023	✅



Test on GitHub Actions

```
32  Error: backend/ezConnect/config.py:2:46: F632 Use `!=` to compare constant literals
33  Error: backend/ezConnect/config.py:2:89: E501 Line too long (119 > 88 characters)
34  Error: backend/ezConnect/config.py:8:36: F632 Use `!=` to compare constant literals
35  Error: backend/ezConnect/config.py:8:89: E501 Line too long (99 > 88 characters)
```

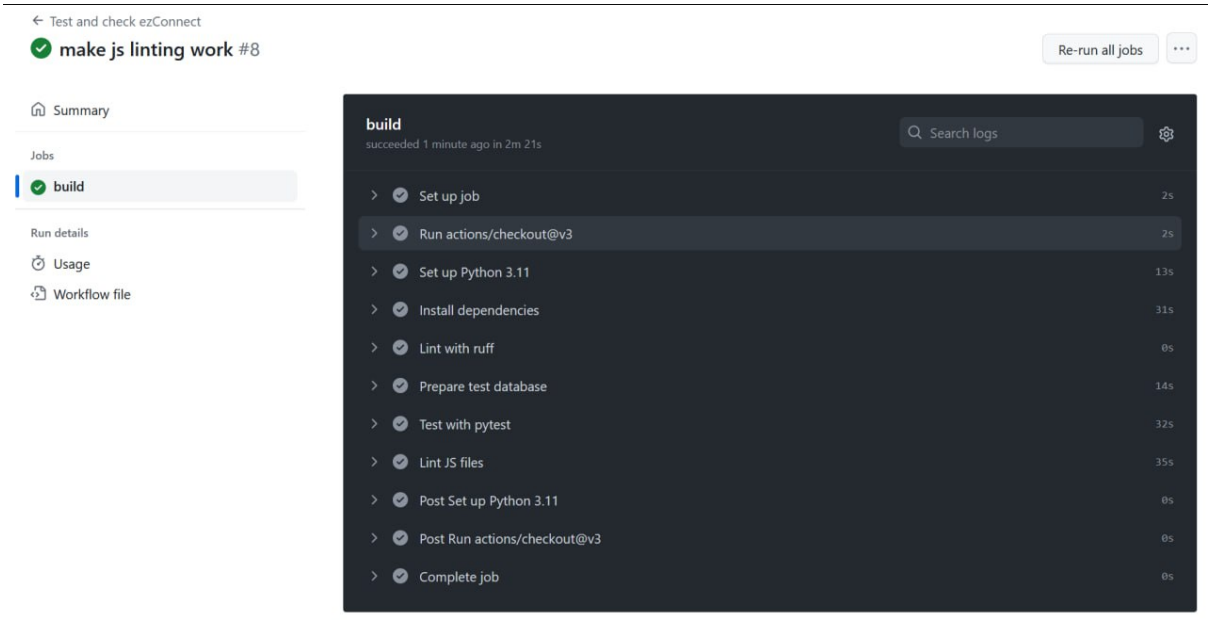
4 backend/ezConnect/config.py

```
@@ -1,11 +1,11 @@
 import os
-FRONTEND_HOSTNAME="http://localhost:3000" if os.environ.get('APP_ENV') is not 'prod' else
 'https://ezconnect.ruibin.me'

 DOMAINS = ["u.nus.edu"]

 DATABASE_USER = "postgres"
 DATABASE_PASSWORD = "test"
-DATABASE_HOSTNAME = 'localhost' if os.environ.get('APP_ENV') is not 'prod' else
 'postgresql-server'
 SQLALCHEMY_DATABASE_URI = f"postgresql://{DATABASE_USER}:
 {DATABASE_PASSWORD}@{DATABASE_HOSTNAME}:5432/ezConnect"

 # FORMAT '{hostname}:{port}'
```

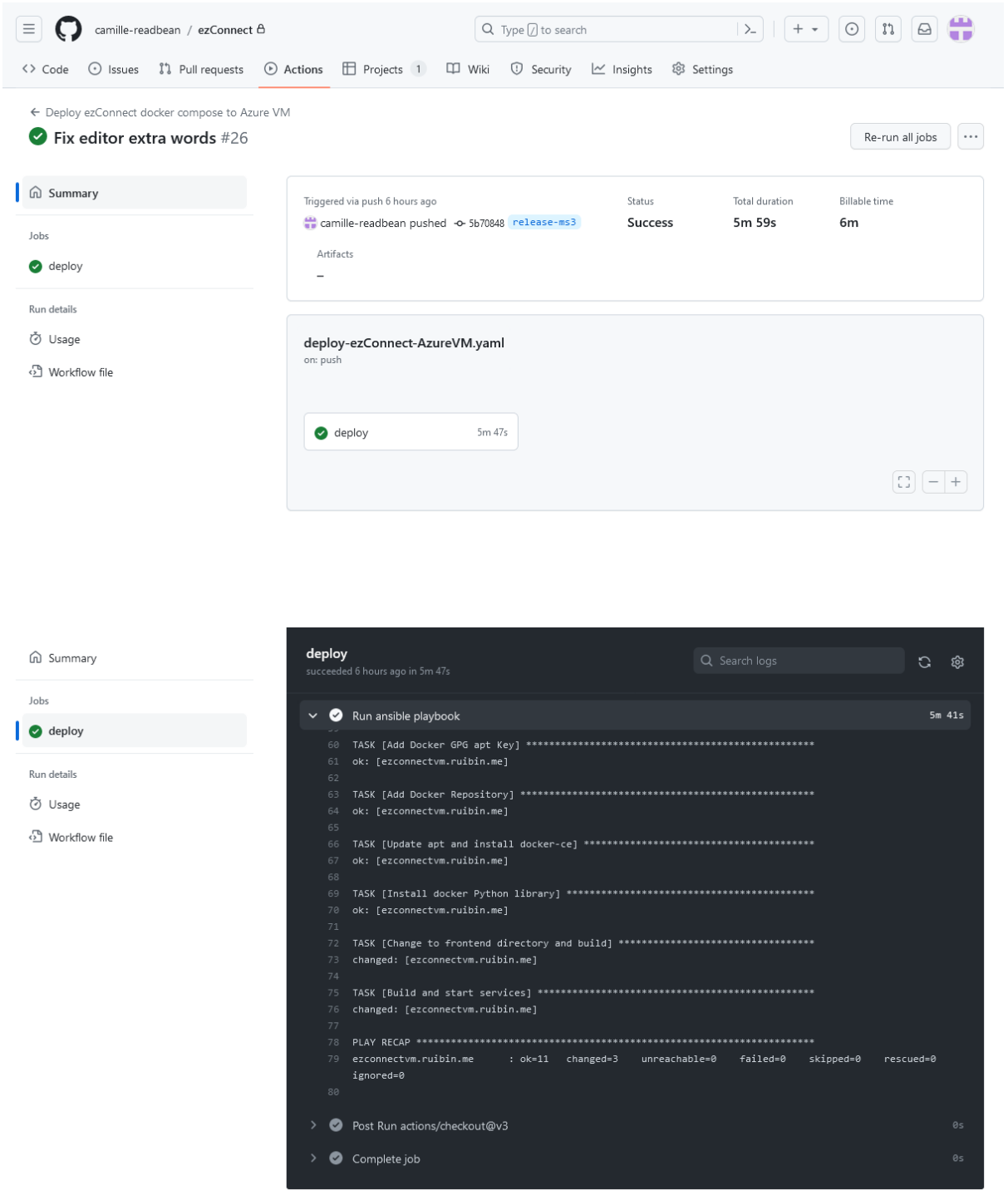
```
 import os
+FRONTEND_HOSTNAME="http://localhost:3000" if os.environ.get('APP_ENV') != 'prod' else
 'https://ezconnect.ruibin.me'

 DOMAINS = ["u.nus.edu"]

 DATABASE_USER = "postgres"
 DATABASE_PASSWORD = "test"
+DATABASE_HOSTNAME = 'localhost' if os.environ.get('APP_ENV') != 'prod' else 'postgresql-
 server'
 SQLALCHEMY_DATABASE_URI = f"postgresql://{DATABASE_USER}:
 {DATABASE_PASSWORD}@{DATABASE_HOSTNAME}:5432/ezConnect"

 # FORMAT '{hostname}:{port}'
```

Errors in config.py caught with ruff

We also used code linting to check and review code style, which in the case caught a bug which could run but behaviour is different from what the developer intended.



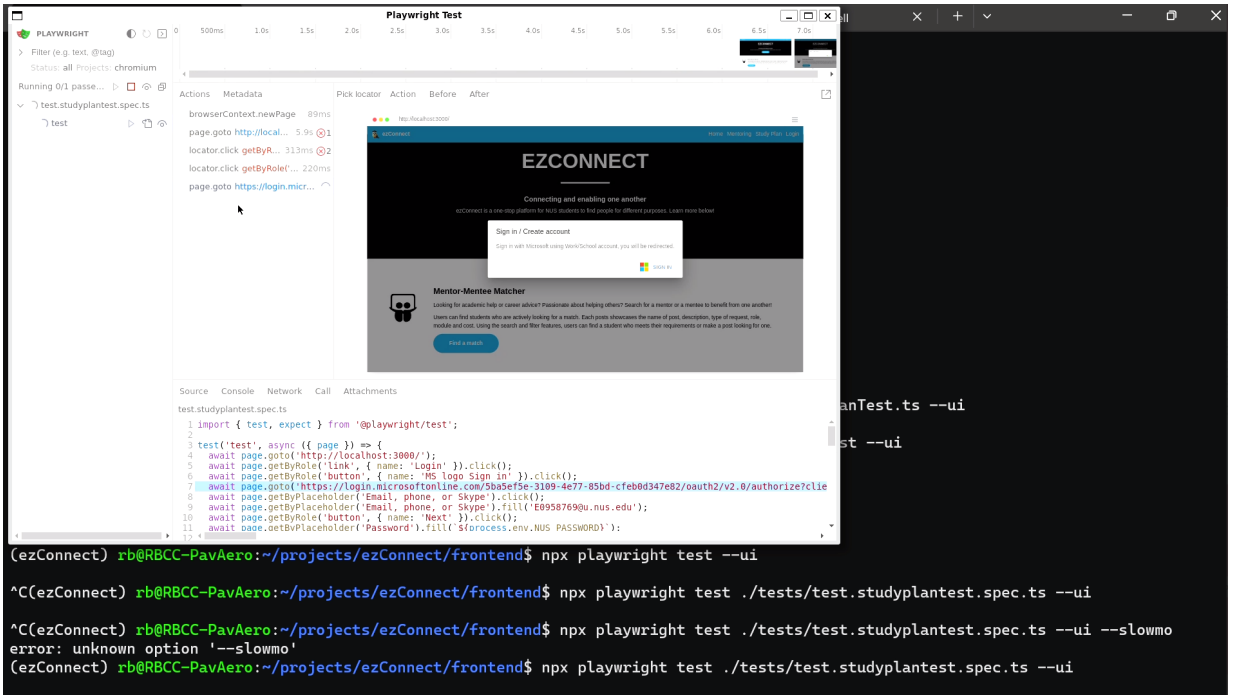
GitHub actions workflow for linting and testing



GitHub Actions for CD

We also made use of GitHub actions for our continuous deployment. On pushes to the branch with names containing the word `release` or `ms`, a deployment workflow is triggered which will run an Ansible playbook with the instructions for deployment.

The deployment attempts to be non destructive which means the database can be persisted and only the code is updated. This allows us to seamlessly push updates which do not change the database to users allowing for critical patches or hotfixes.



Playwright test running

Our team also attempted to set up frontend testing through automated browser testing via playwright. Unfortunately, we did not have enough time to set this up beyond a rudimentary test which checks if the login button can be pressed and the pop up appears.

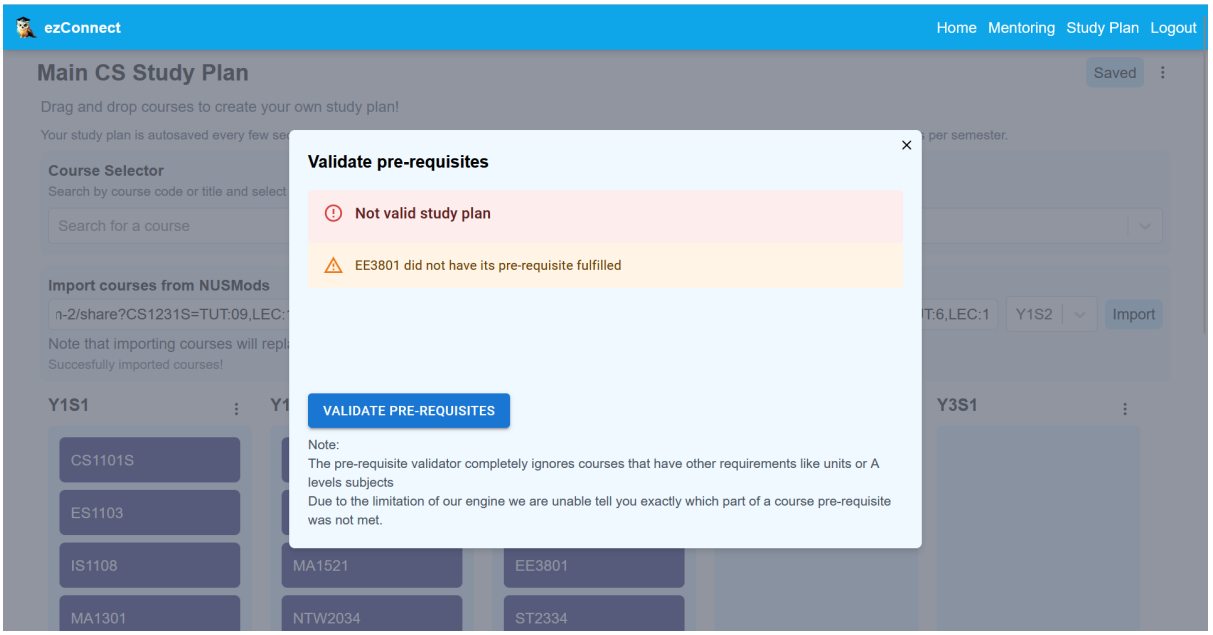
Due to the nature of our application, the test will need to log in, requiring us to give it access to a real NUS ID and Password. A possible step is to do this step interactively where a human manually enters the username and password, however, there seem to be some issues with this. Microsoft's security feature may be blocking the signing attempt as it is being done by a robot. Hence we are unable to fully implement this feature by milestone 3 but it shows great promise in the future.

User Testing

User testing is one of the critical steps undertaken by us in preparation for milestone 3. As stated, testing increases our confidence in our system. Additionally, user testing further allows us to cover more blind spots that we as the application developers may miss out on. Hence, throughout our project, we have conducted numerous forms of user testing to validate the utility of our application.

Dogfooding

One form of user testing that can be done by the developers themselves is dogfooding. Dogfooding refers to developers using the software they are developing themselves as users to evaluate the usefulness of the software. By using the software themselves, we, the developers, put ourselves as the first line of contact between bugs and the users. This allows us to find potential flaws and bugs before the users do so when the software is released.



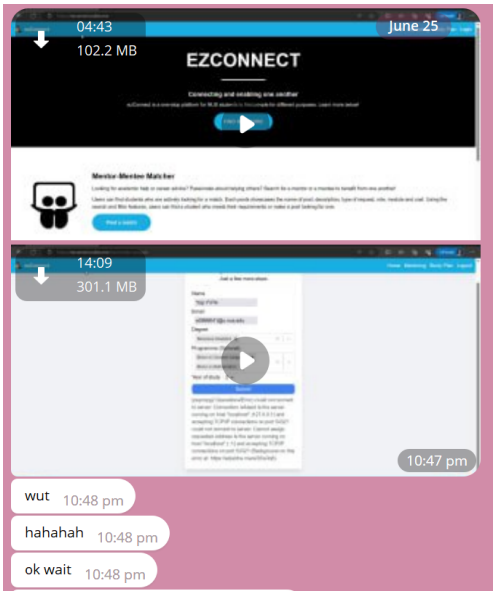
Using the study plan feature to check course pre-reqs for AY2023-2024 Course Registration

The above image shows one of our developers using the study plan feature to plan and check for courses' pre-requisites during Course Registration exercise. The developer managed to use the software successfully and was surprised at the failure in pre-requisite check, which upon further inspection might arise from faulty error from the original source of information from NUSMods. Developers also used this opportunity to test that the features like NUSMods imports are working as intended.

However Dogfooding is not meant to replace user testing, it only strengthens the foundation and increases confidence before proper user testing, which brings us to one of our first user testing sessions.

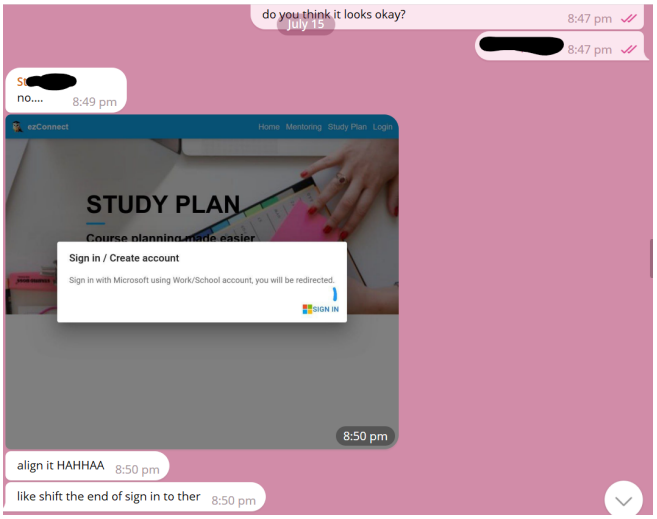
Limited Usability Testing from milestone 2 to milestone 3

In preparation for milestone 2 and at the same time generating content for the project video, a guided usability testing session was held with a school mate who does not work on the project. The test was done over Zoom and the user had to do several tasks on the website. The developer also analysed the screen recordings to identify difficult moments in the user interactions, correlating this with verbal feedback from the session. The resulting footage was incorporated into milestone 2's video which features footage from two user's perspectives.



Screen recordings from usability testing

Unmoderated usability testing was also done by letting other students access the production application and gathering their feedback after they have tested the system at their own pace.



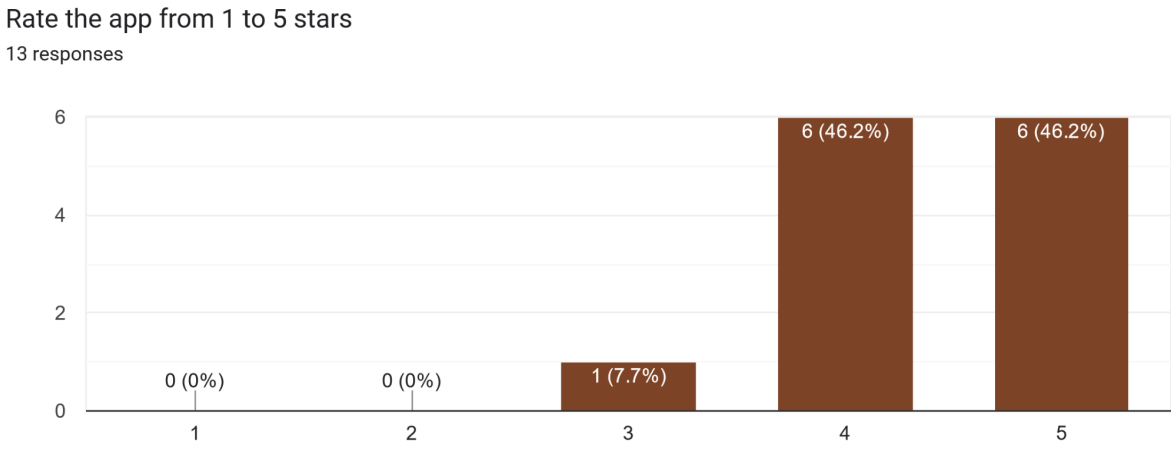
Example of UI feedback from student testers

Usability survey for milestone 3

User testing is meant to increase the confidence we have in the usability of our application for our users. In preparation for milestone 3, we conducted a usability survey of our application with real users. A Google form was created with questions regarding the experience and intuitiveness of completing various tasks on the application. This survey was sent out on 17 July 2023 with the last response on 22 July 2023. In total **13 users** replied to our survey with 17 users using or trying out our application.

We were able to utilise the surveys and discover hidden bugs that were not caught in our initial testing, an example being certain courses that crashed the validator as there were over 1000 courses with prerequisites and we were unable to test them all. Users also gave beneficial feedback regarding the user interface and user experience which was used to make updates to the application interface. Some spelling mistakes which persisted from milestone 1 were also caught and resolved.

Overall, users were satisfied with our application and some expressed interest and wished that it can be integrated into NUSMods. A majority of the users expressed that they would use such an application and 92% of users rated it a minimum of 4 out of 5 stars, the average rating is 4.38 stars. In the following sections, we will be going over the design and results of the survey in greater detail.



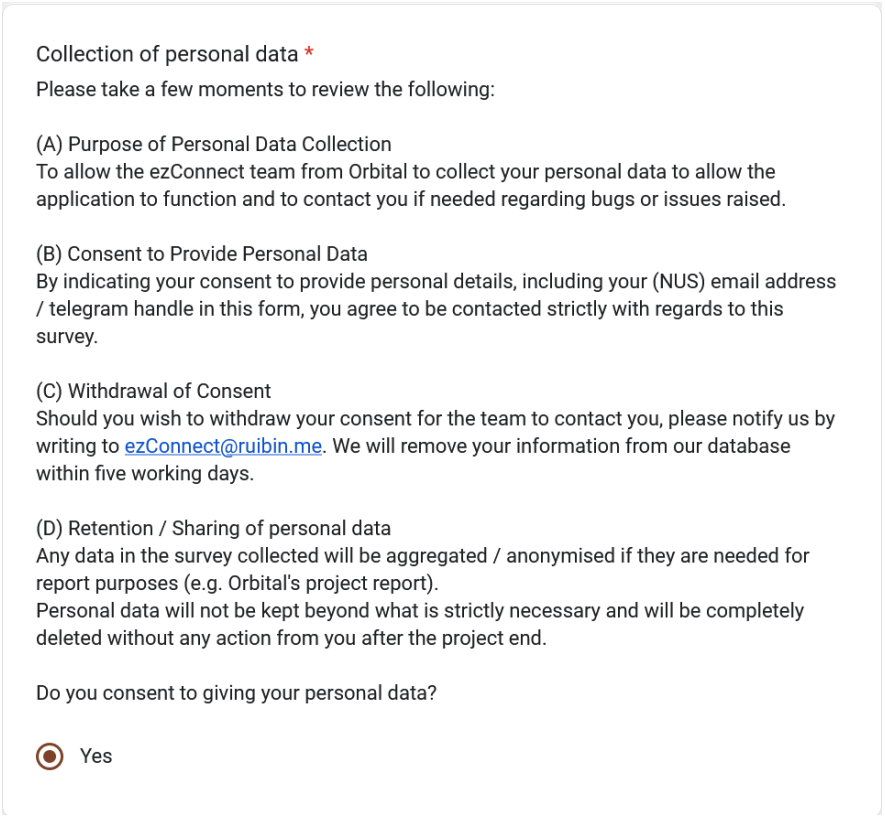
Overall rating of the application from the survey

Design of survey

The goal of this survey is to give us a better understanding of our users and their experience with our application. As such we designed a usability survey to investigate how the application interface and flow are perceived by the users. We originally considered doing a remote user testing session using Loop11 with users but decided against that due to time constraints. Setting up user activity heatmaps may also take too much time so we settled on conducting a survey as

that allows more scale and flexibility in reaching more end users. We decided that the trade-offs are worth it as we do not have that much time to analyse and schedule sessions with users. The survey method will provide potentially more responses and will be good enough for analysis if we can design the questions well.

We designed the question of our usability survey around our proposed user stories. We first requested consent from the user to collect their personal data, especially so that their NUS email can contain their NUSNet ID and for the purpose of contacting them for more information regarding any bugs they discover.



Collection of personal data *

Please take a few moments to review the following:

(A) Purpose of Personal Data Collection
To allow the ezConnect team from Orbital to collect your personal data to allow the application to function and to contact you if needed regarding bugs or issues raised.

(B) Consent to Provide Personal Data
By indicating your consent to provide personal details, including your (NUS) email address / telegram handle in this form, you agree to be contacted strictly with regards to this survey.

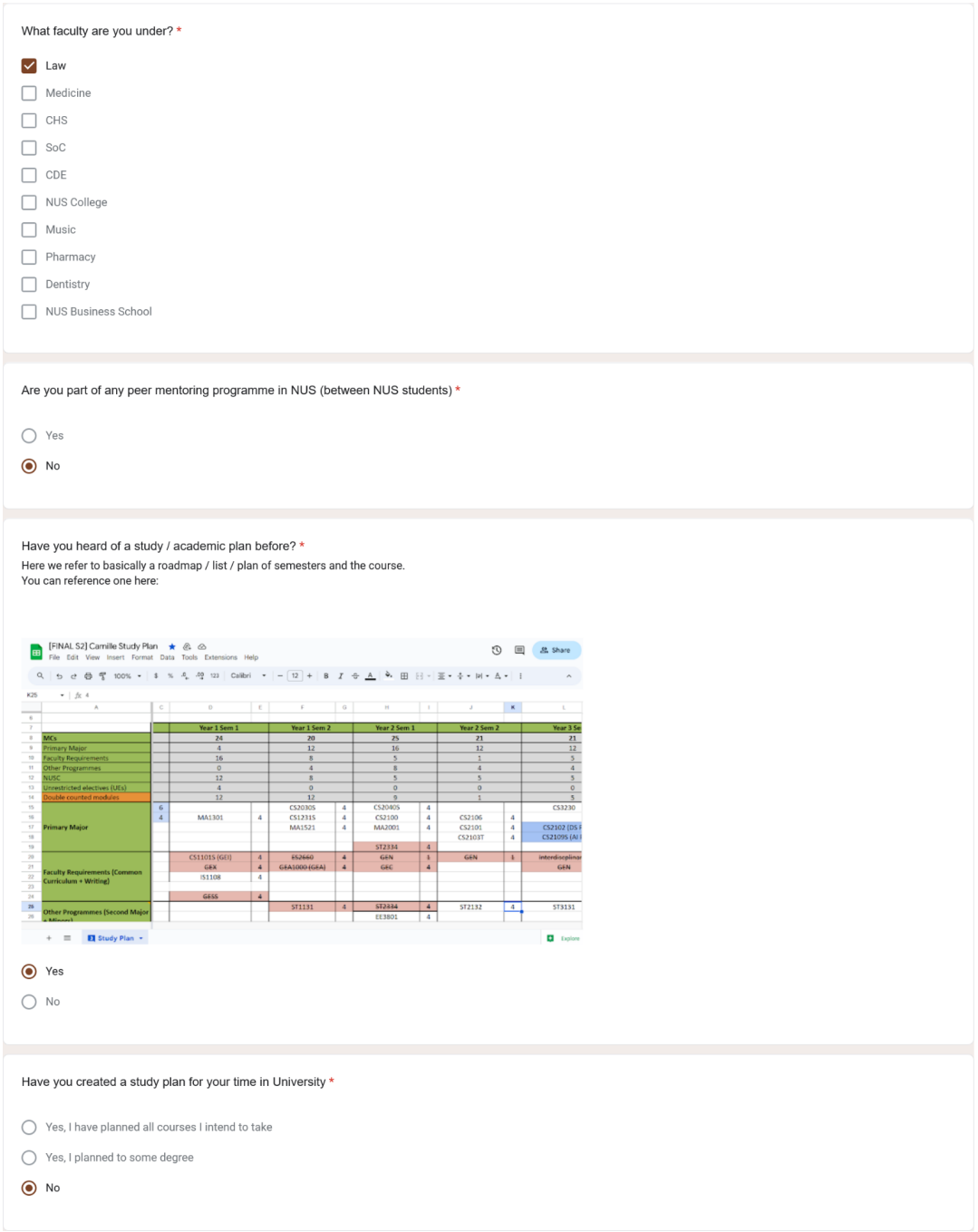
(C) Withdrawal of Consent
Should you wish to withdraw your consent for the team to contact you, please notify us by writing to ezConnect@ruibin.me. We will remove your information from our database within five working days.

(D) Retention / Sharing of personal data
Any data in the survey collected will be aggregated / anonymised if they are needed for report purposes (e.g. Orbital's project report).
Personal data will not be kept beyond what is strictly necessary and will be completely deleted without any action from you after the project end.

Do you consent to giving your personal data?

Yes

Personal Data consent form



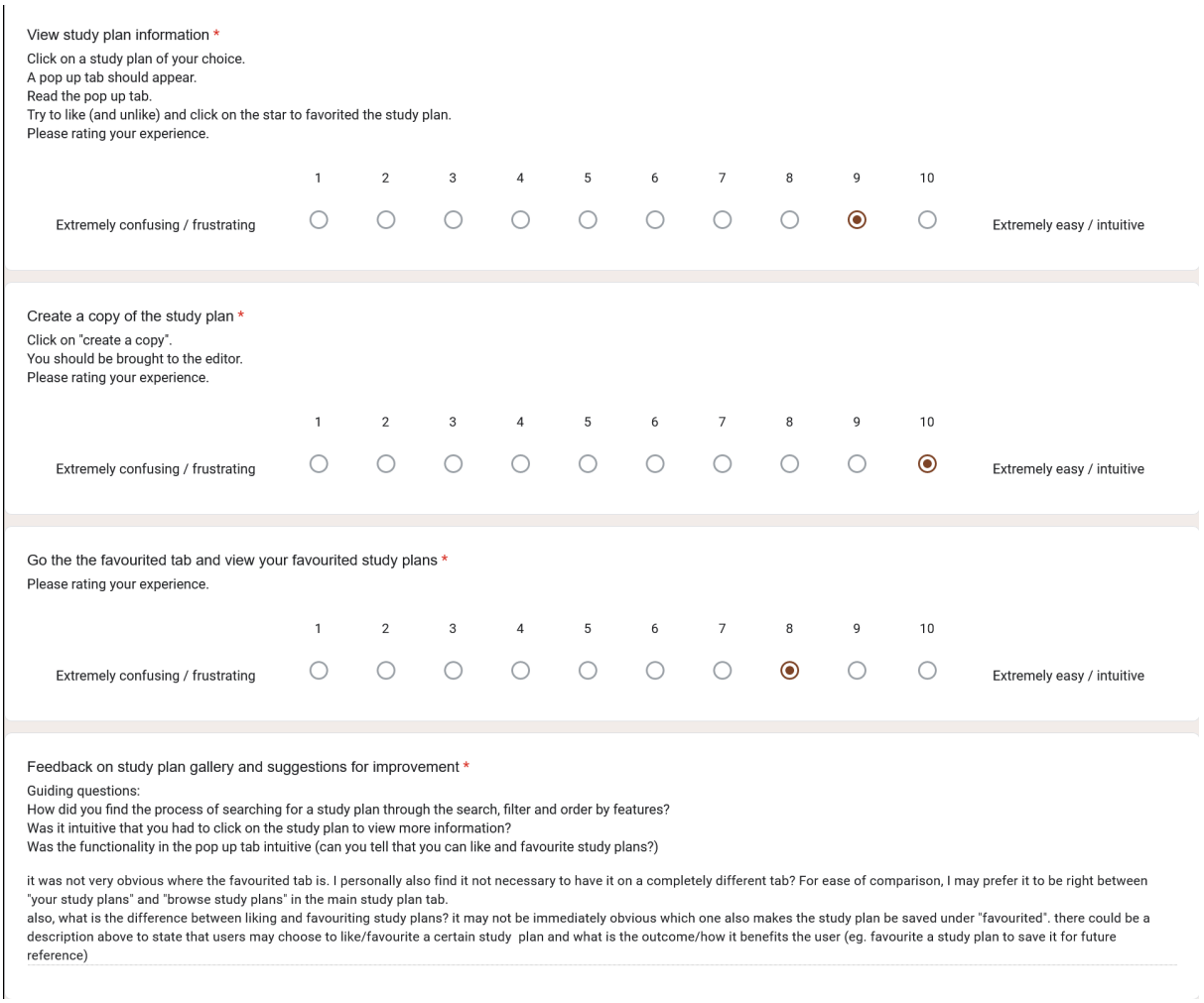
Section on background of user

To achieve one of our additional goals of understanding the user and finding out the usefulness of our application, we included some questions about the background of the user; i.e. whether they are part of any mentoring programme in school and whether they have created a study plan or seen one before. This may be because a “study plan” in the context of this application may be new to some users. We made sure to revisit this topic by asking at the end whether the user would use an application like this.

We constructed the usability survey to mirror a possible flow for a new user. The first section covers the ease of use of signing up and creating an account. A scale from 1 to 10, from

“Extremely confusing / frustrating” to “Extremely easy / intuitive” is used to let the user feedback how easy certain tasks are to be completed as a user. Additionally, we have included long response questions for users to give detailed bug reports and feedback on the user interface and experience.

Questions are based on our planned user stories, each question involves completing a task on the platform that is related to the user story. For example, stories regarding finding a mentor will involve creating and requesting mentors on the mentoring features. We planned the tasks to cover most of our application so that all features will be thoroughly interacted with.

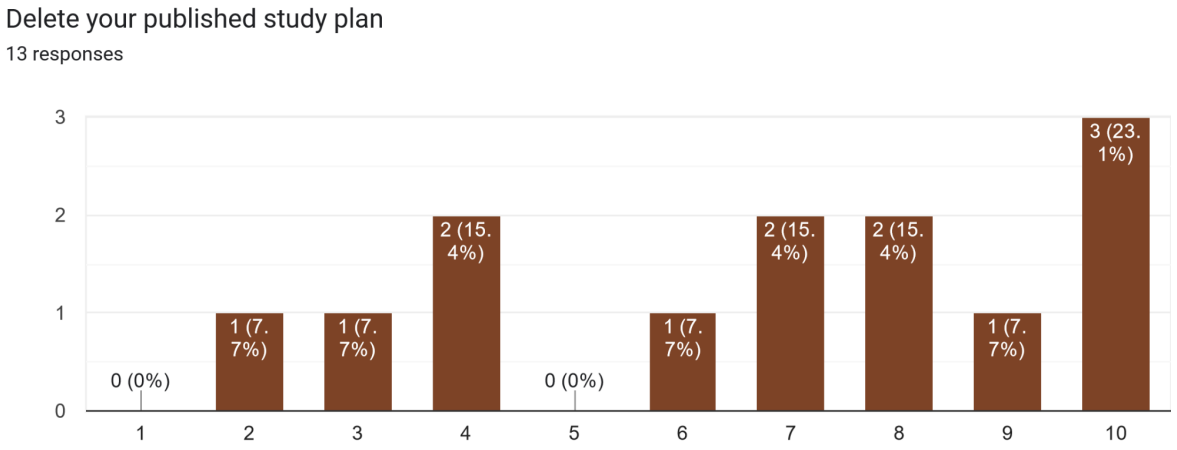


Example of feedback on study plan galley

The tasks are given with minimum instructions to test for the user experience. We want to know if using the application is intuitive and the user experience should not be “contaminated” with prior knowledge or instructions. This way, the user gets to have an experience most similar to what an actual user would feel like trying to accomplish certain tasks. The results we received from our survey are very promising and led to changes in parts of our user interface which we will discuss below.

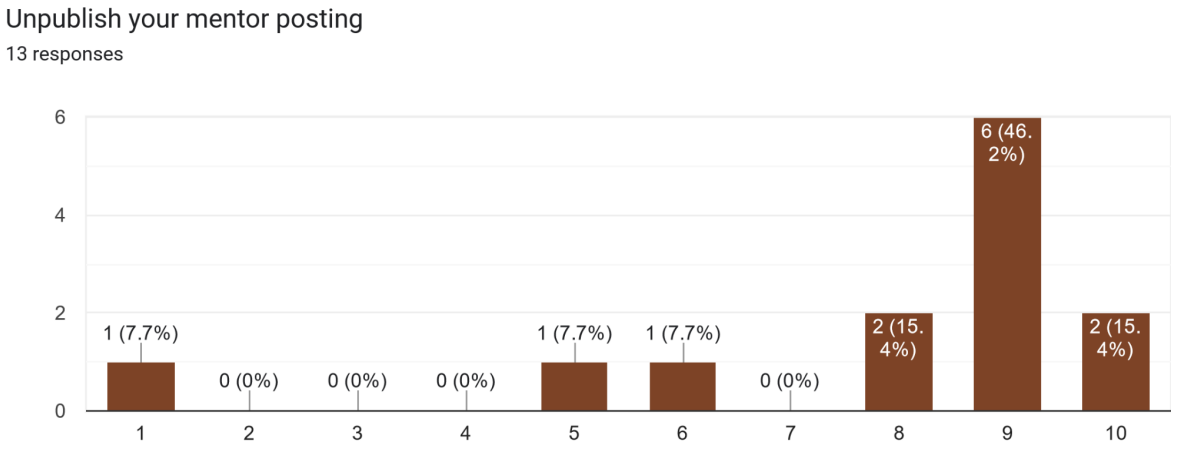
Findings and subsequent updates

Overall we find that most users (**n=13**) are satisfied with the usability of the app, the average rating of the easy / intuitiveness scale is **8.51/10** across all the questions. The task with the lowest rating was “Delete your published study plan” at **6.7/10**.



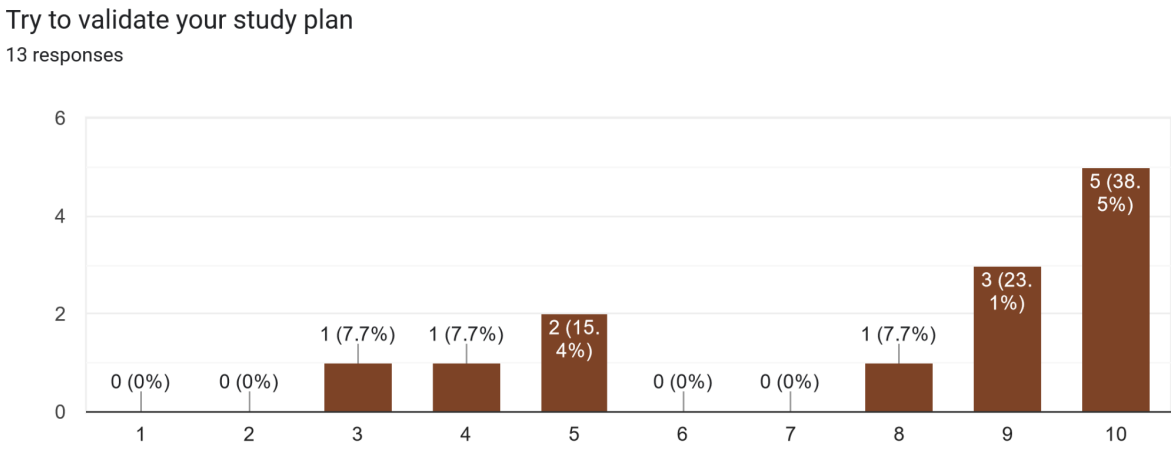
Ratings for the intuitiveness / easiness for unpublishing mentor posting

This was due to the user not being able to delete a published study and a published study plan cannot be deleted. This was missed out during initial testing and overlooked. User testing allowed us to catch this bug. This was later fixed in the final update before our feature freeze for milestone 3.



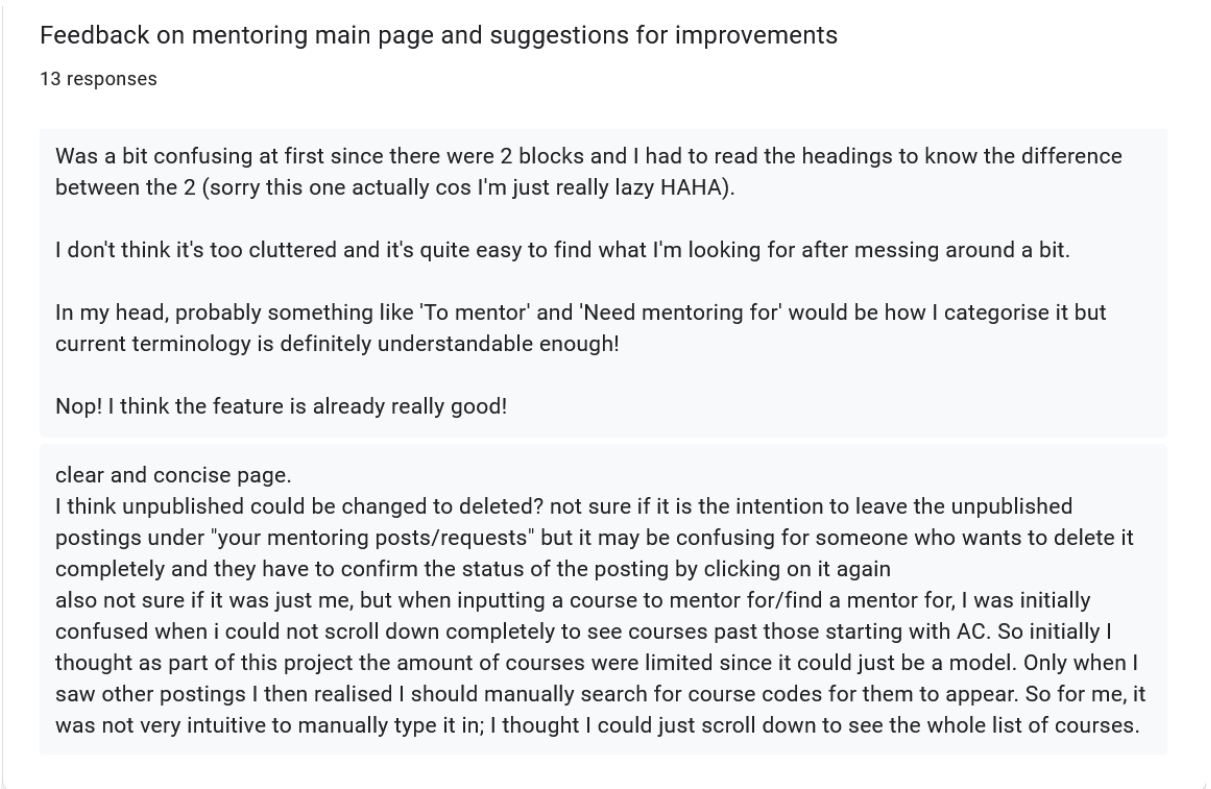
Unpublishing mentor posting

Similarly, unpublishing mentor postings also have quite a low rating of **7.84/10** due to the need to click through multiple steps to the update posting section before it can be unpublished.



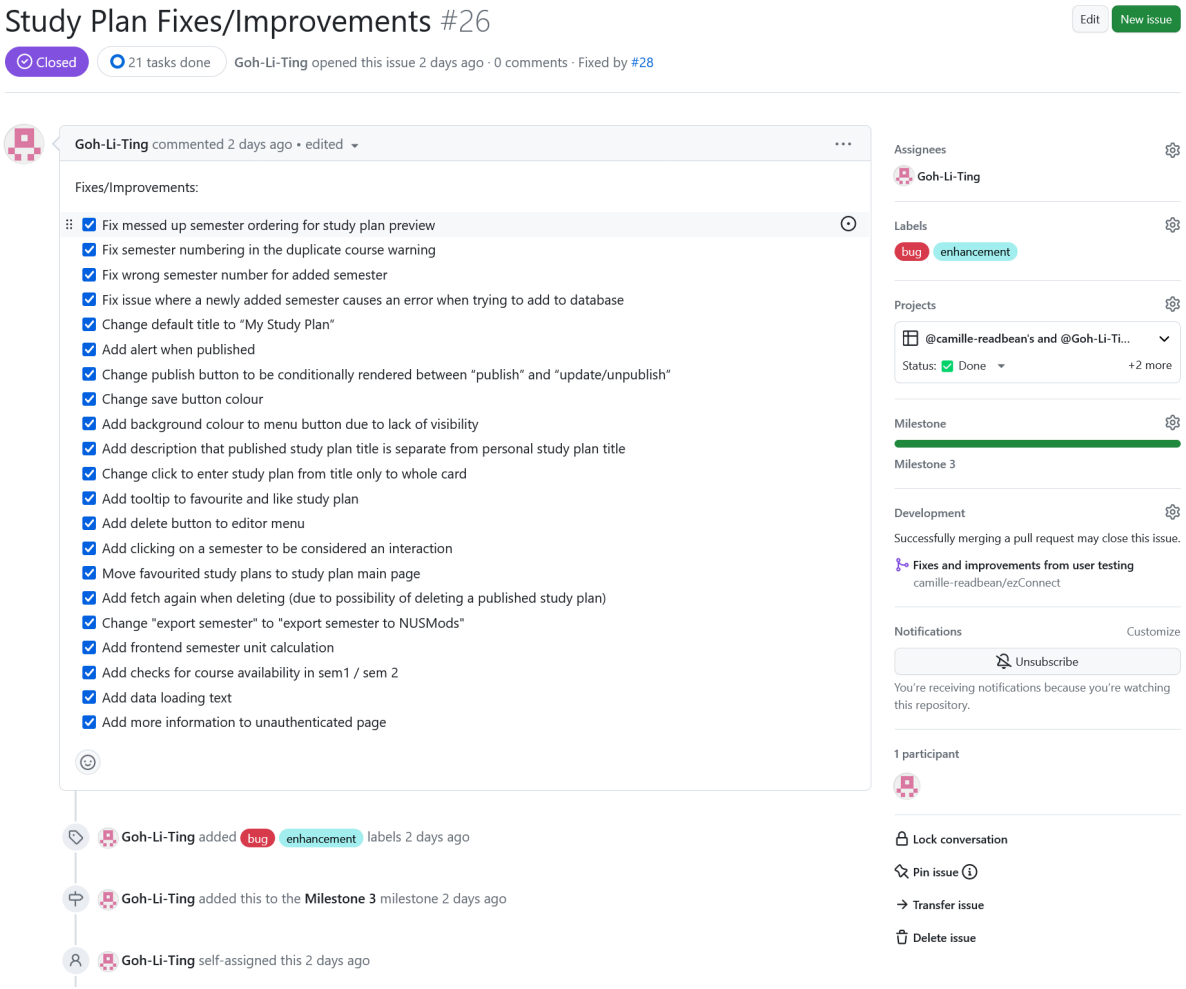
Responses for validating study plans

Ratings for validating study plans also suffered as there were some uncaught bugs that caused the validator to give a blank screen when validating study plans. This was isolated to the difference in how the prerequisites are stored in NUSMods API, with and without enclosing parentheses. Some prerequisites were also not in the expected format even after we have filtered out all potentially breaking prerequisites.

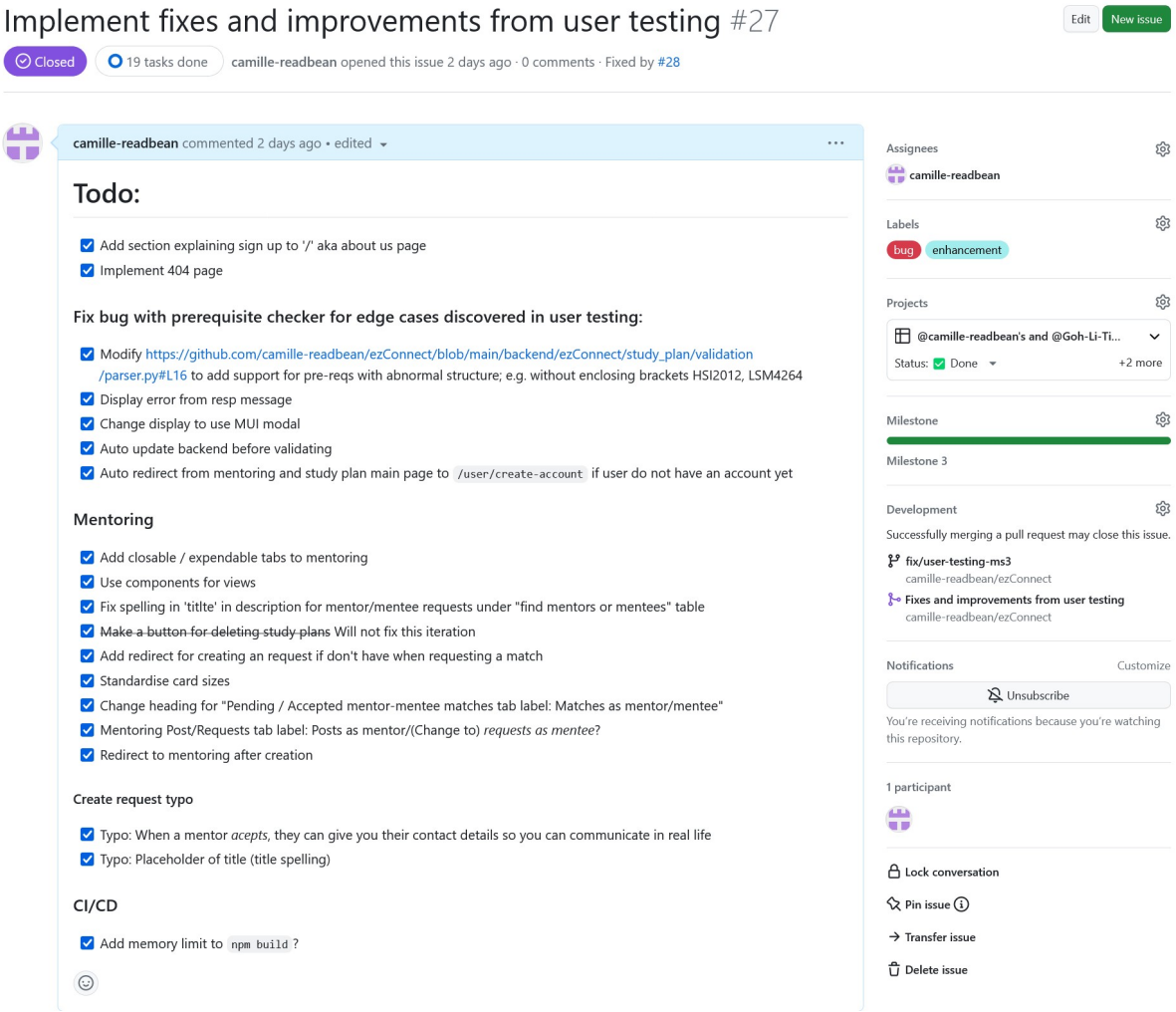


Long response feedback on features

We also received very detailed feedback which helped us get an even better understanding of rough moments in the application experienced by the users as well as suggestions for improvements to the interface. Due to time constraints not all feedback can be implemented, especially with regards to changing unpublishing mentor / mentee postings to delete said postings instead due to how significantly the backend and subsequently the corresponding test will have to be changed and hence we decide to keep it for after milestone 3.



One of the GitHub issues with tasks based on the usability testing feedback



Another issue that was also created as part of usability testing

Overall, 100% of users agreed that the application will be useful for students, and except for one senior who has already completed their course requirements, 12 out of 13 indicated they are likely to use such an app.

Any last thoughts?

8 responses

LOVE the study plans!!! pls share after orbital I wan use!

This website is soo professional compate to my orbital app I think i should spend more time to work on my orbital project haha

Felt that it's a good utility application. Can consider implementing better colour scheme for alert or confirmation messages.

This is a very useful tool, and I think that it can be as big as NUSMODS if the UI is set to be more intuitive and aesthetic (but of course after the bugs have been fixed and changes have been made to make the user experience better)! Great job everyone!!!!

Nil

Limitations

Lack of availability of programme data

There are many degrees in NUS but there is no consolidated list available. This caused some of the degrees to be quite general such as “Engineering” instead of programmes such as “Chemical Engineering” and “Biomedical Engineering”. This issue is similar for second majors. The large number of programme combinations made it difficult for us to account for all the combinations, especially without any available list of programmes except for a list of minors offered in NUS. Furthermore, programme availability changes over the years and since our current list of programmes is obtained manually, we currently do not have a viable option to continuously update our database. Something interesting we come across seems to be NUS’s internal tracking of Degrees / Majors in codes like “0300CSHON” or “0300ISHON” but without access to NUS’s internal source of truth we can only do this manually or with some kind of scraping, which would be one project by itself.

Reliance on NUSMods Data

The courses available in our database are sourced from NUSMods for the upcoming academic year. However, some courses may have changed or been replaced by new courses, which means that existing students may be unable to add courses that they have previously taken to their study plan. This affects the study plan building experience and also causes the validator to raise flags that their study plans are invalid as the courses that were removed could have been prerequisites to other courses. Furthermore, the availability of the courses may change in future academic years.

Timeline and Development Plan

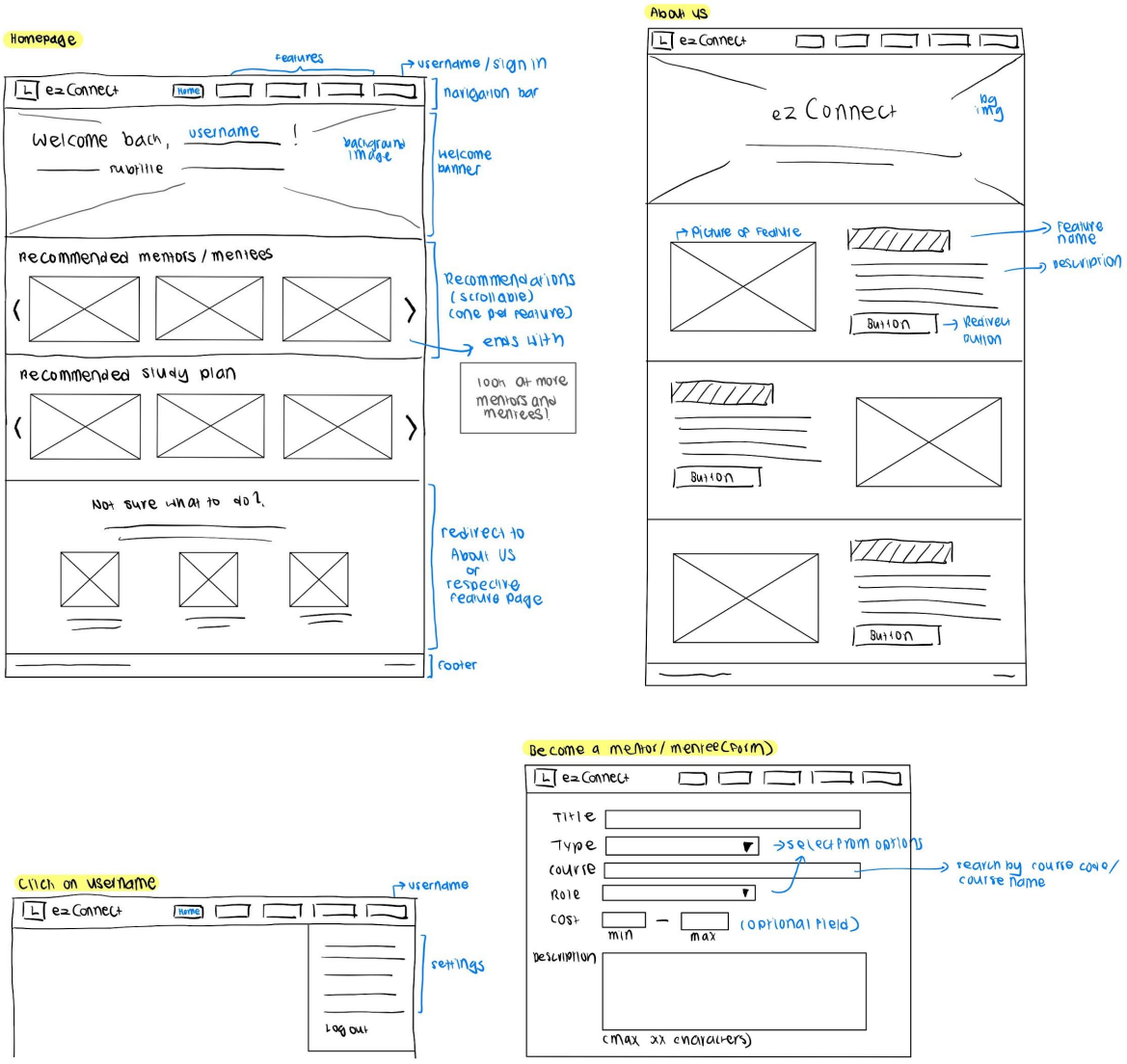
MS	Sprint	Tasks	Description	In-Charge	Date
1	0	Ideation	Develop ideas and description for features	Rui Bin Li Ting	11 - 15 May
		Research	Familiarise with the tech stack	Rui Bin Li Ting	16 - 21 May
		Set up dev environment	Set-up Azure resources and git repo	Rui Bin	19- 29 May
			Set-up dev web server + domain etc	Rui Bin	21 - 29 May
		Postgresql Database	Design schema and docker-compose	Rui Bin	21 - 29 May
		SQLAlchemy setup	Initial Migration and upgrading and integration with Flask	Rui Bin	21 - 25 May
		About us	Set up frontend and create about us page	Li Ting	19 - 24 May
		User account creation and login	User schema and ORM modelling and creating supporting APIs	Rui Bin	19 - 25 May
			Build webpages for sign up and login	Li Ting	24 - 28 May
Evaluation Milestone 1: Ideation - Ideation - Proof-of-concept: Create account and login					29 May
2	1	Azure AD B2C Prototype	Build the models / frontend for Azure B2C	Rui Bin	30 May - 11 June
		Study Plan Frontend	Build the pages for Study Plan	Li Ting	
		Interface improvement	Improve navigation bar interface and homepage to support different screen sizes	Li Ting	
		Study Plan Backend	Models and APIs for Study Plan	Li Ting	

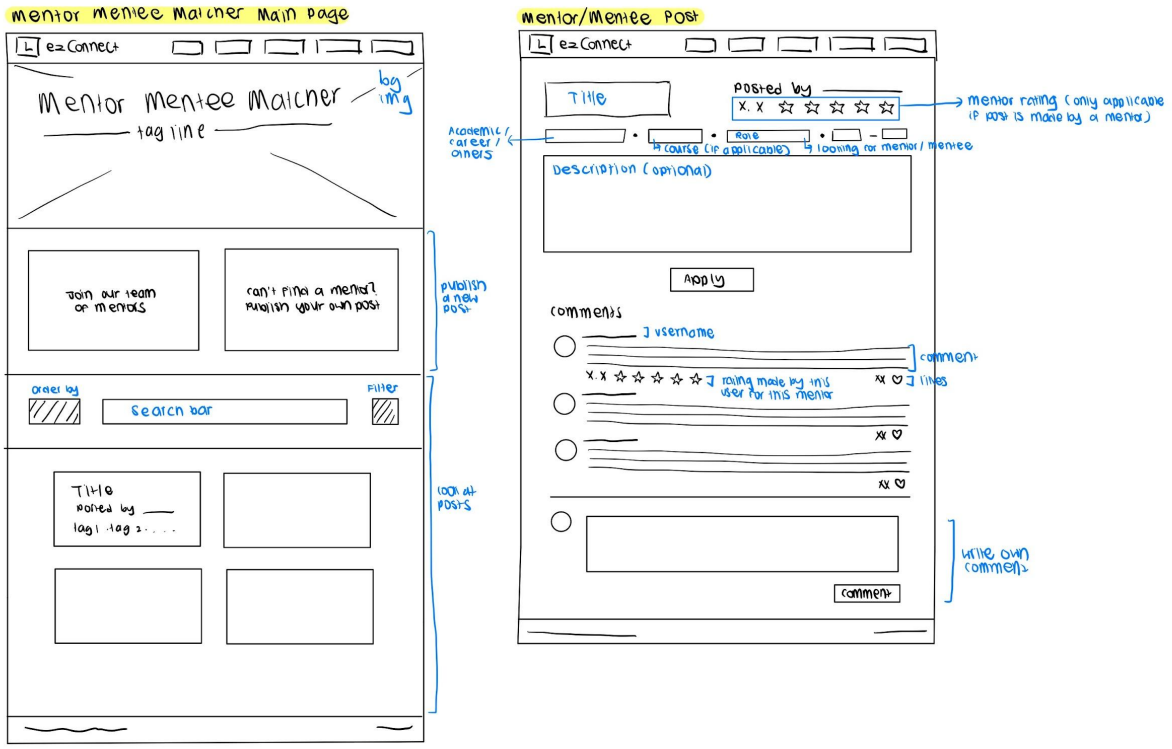
	2	Search and filter by courses	Implement search and filter algorithm, pre loading data from nus mods and etc	Rui Bin Li Ting	12 - 25 June
		Azure AD B2C	Update backend and users to use Azure AD B2C	Rui Bin	
		Mentoring	Create backend apis, frontend views	Rui Bin	
		CI/CD	Research into product to use, implementation of GitHub actions, writing of workflows, docker compose and ansible playbooks	Rui Bin	
		Testing and debugging	Test systems for milestone 2 Write tests	Rui Bin Li Ting	
Evaluation Milestone 2: Prototyping - Mentor Mentee Matcher - Study Plan					26 June
3	3	Feedback work	Responding to feedback	Rui Bin Li Ting	27 June - 9 July
		Study Plan Publisher	Implement frontend and backend to support publishing and view of study plans	Li Ting	
		Study Plan Validator Prototype	System to check that prerequisites are fulfilled	Rui Bin	
	4	NUSMods semester importer and exporter	Implement a way to import a study plan semester from NUS mods	Li Ting	10 - 23 July
		Study Plan validation implementation	System to check that prerequisites are fulfilled	Rui Bin	
		Testing and debugging	Test systems for milestone 3	Rui Bin Li Ting	
		Usability Beta testing	Create form and implement feedback	Rui Bin Li Ting	
Evaluation Milestone 3: Extension - Study Plan					24 July

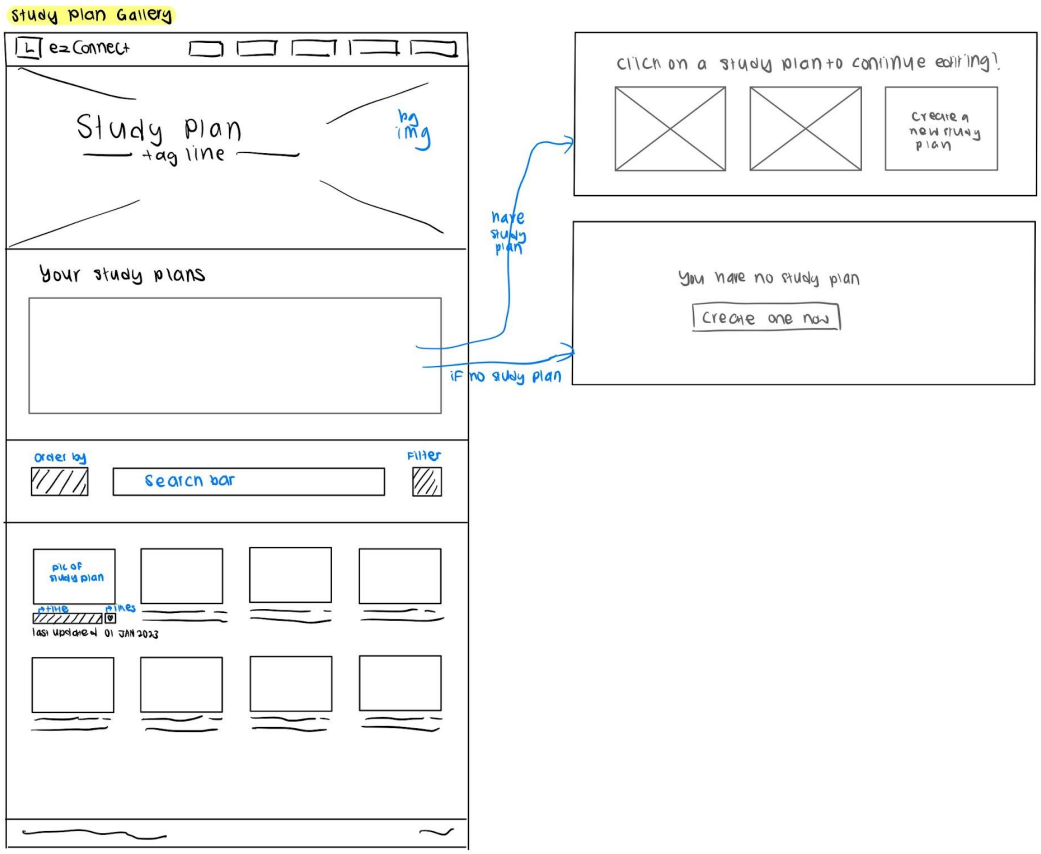
4	5	Refinement of user interface	Improve user interface	Li Ting	25 July - 6 Aug
		Refinement of database	Test and improve database and integration with frontend	Rui Bin	
	6	Test and improve algorithms	Testing and debugging and improving systems	Rui Bin Li Ting	7 - 22 Aug
Splashdown: Refinement					23 Aug

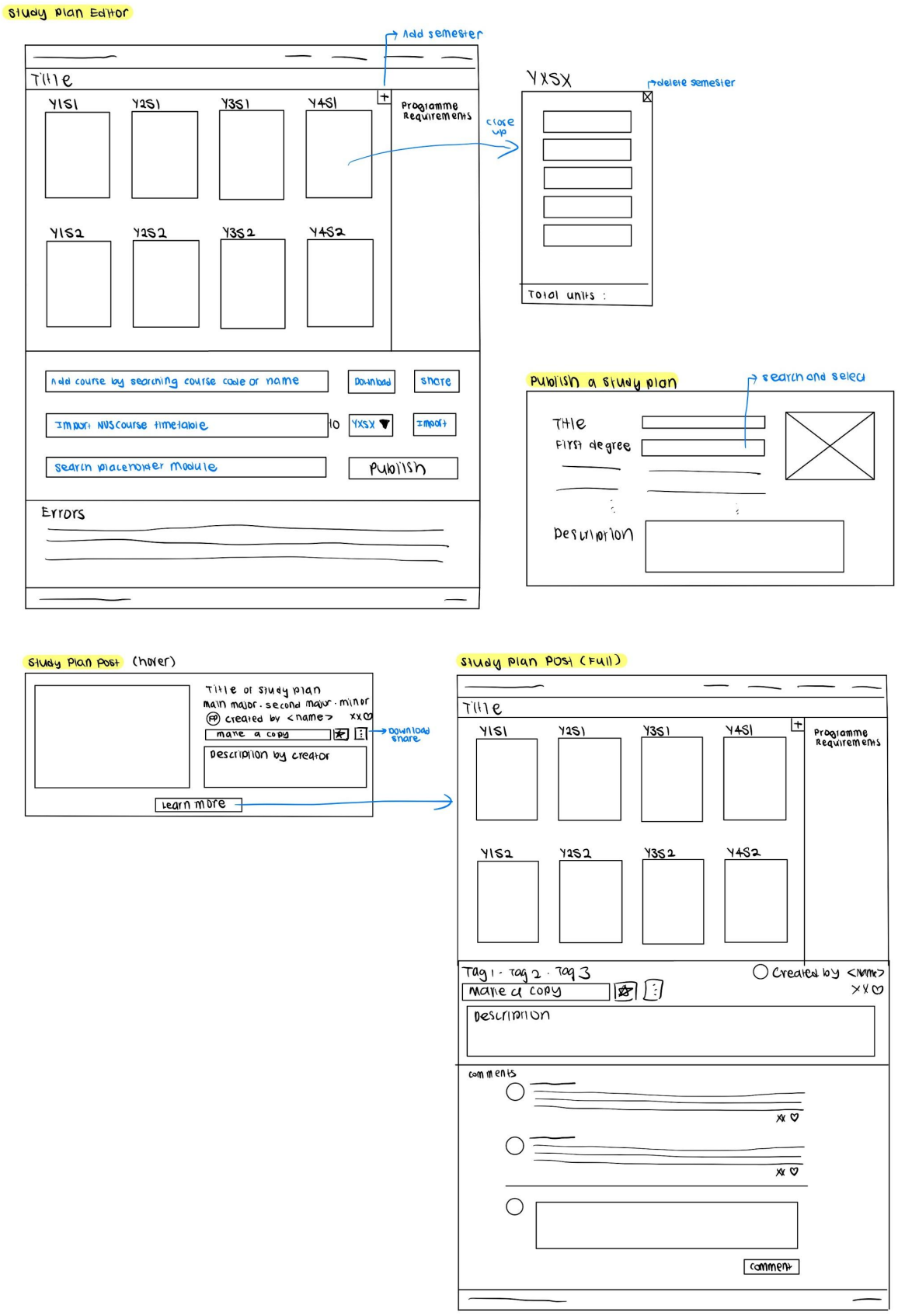
Wireframe

We made a wireframe to help visualise the elements that we needed in the frontend of our webapp. This helps in creating the components for the frontend and gives visual aid when styling the components.




Mentor Mentee Matcher Main Page
ezConnect
Mentor Mentee Matcher
tag line
bg img
Join our team of mentors
can't find a mentor? publish your own post
publish a new post
order by
Search bar
Filter
Title
posted by
tag1 tag 2 . . . .
look at posts
Mentor/Mentee Post
ezConnect
Title
posted by
X.X
mentor rating (only applicable if post is made by a mentor)
Academic / career / others
Role
looking for mentor / mentee
course (if applicable)
Description (optional)
Apply
comments
username
comment
X.X
rating made by this user for this mentor
likes
write own comment
comment


Study plan Gallery
ezConnect
Study Plan
tag line
bg img
your study plans
order by
Search bar
Filter
pic of study plan
title
dates
last updated 01 JAN 2023
have study plan
if no study plan
click on a study plan to continue editing!
create a new study plan
you have no study plan
create one now



Initial wireframe

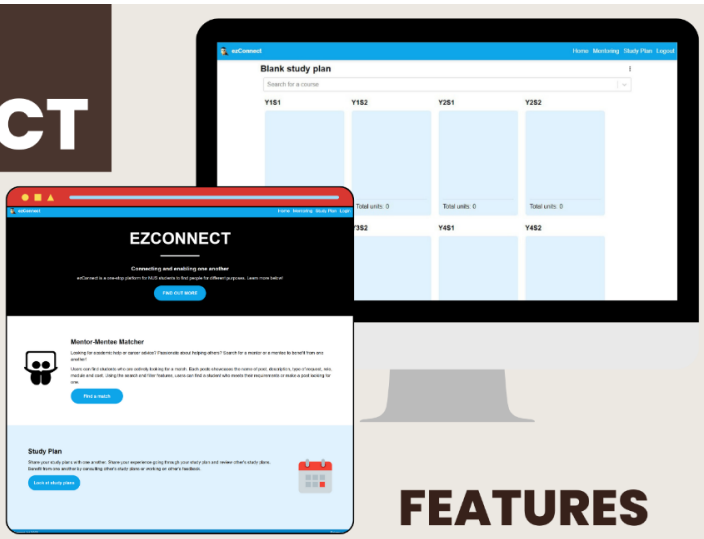
Posters

Liftoff Poster

Milestone 1 Poster

CONNECTING AND ENABLING ONE ANOTHER

EZCONNECT



ABOUT

ezConnect aims to be a consolidated webapp for users to conduct tasks relating to study life and academics and make it easier for students to exchange contact information and help one another. With ezConnect, students no longer have to search for or join multiple telegram channels for such purposes.

SWE PRACTICES

We used git to conduct version control using branching and pull requests. Github Issues and Project was also used to set tasks and deadlines.

TESTING

We conducted automated testing using pytest and Github Actions to test API calls. We made test cases for each API path and included test cases to check for both the expected success and failure of each API call. These tests help us to ensure that the changes we made to the code does not affect the functionality of previous features.

TECH STACK

React

Tailwindcss

Git/Github

SQLAlchemy

Flask

Azure

INTEGRATION WITH NUSMODS

Import your NUSMods timetable into your study plan and export a semester to a NUSMods timetable. Helps you to reduce the need to type the same courses on both sites!

FEATURES

MENTOR MENTEE MATCHER

Find a mentor or mentee for a specific course or for career advice. Only users with a valid NUS account can use ezConnect, allowing for safe matches. Emails are hidden before acceptance to protect private information.

STUDY PLAN EDITOR

Plan your study plan using the drag and drop functionality. Add courses by simply searching using the course code or course name. The total units for each semester is automatically updated. Accommodates users with varied number of semesters. Download and share your study plan easily.

STUDY PLAN GALLERY

View all published study plans easily to use for reference or consideration. Search through the study plans using search and filter. Filter study plans based on their tags (majors and minors). Order study plans by relevancy, trending, most likes or most recent.

RECOMMENDATIONS

Receive recommendation on mentors/mentees or study plans that are more relevant to you based on your profile!

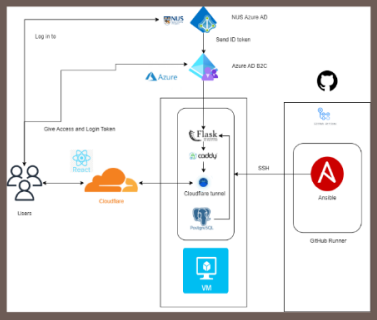
STUDY PLAN VALIDATOR

Check if courses in your study plans are in valid order! The validator checks that prerequisites have been fulfilled in earlier semesters. The validator will also check the validity of the number of units in each semester and course availability in that semester. Check if you have fulfilled your academic plan requirement using the validator.

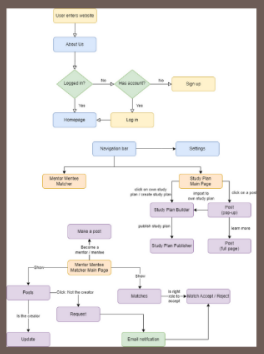
STUDY PLAN POST

Read about the reasoning behind the study plan and the creator's experience with the study plan to use for consideration when making your study plan. Make a copy of a study plan and modify it to suit your needs. Star comments to refer to. Add comments to share your views about the study plan or ask the creator questions.

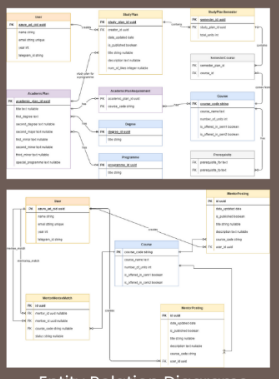
DIAGRAMS



Architecture Diagram



Navigation Flow Diagram

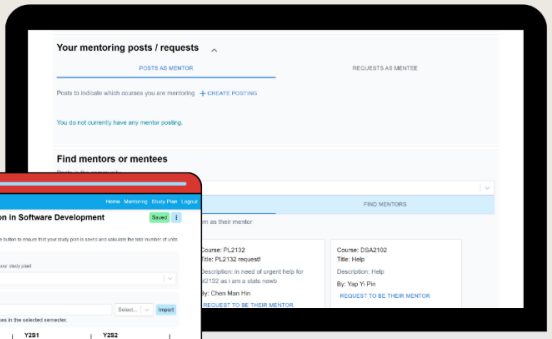


Entity Relation Diagrams

[Milestone 2 Poster]

CONNECTING AND ENABLING ONE ANOTHER

EZCONNECT



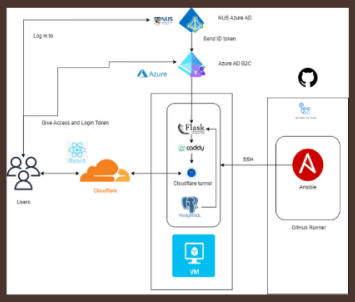
ABOUT

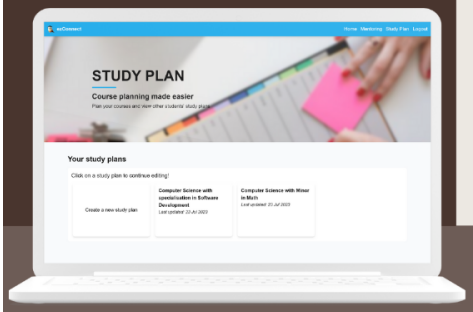
ezConnect aims to be a consolidated webapp for users to conduct tasks relating to study life and academics and make it easier for students to exchange contact information and help one another. With ezConnect, students no longer have to search for or join multiple telegram channels for such purposes.

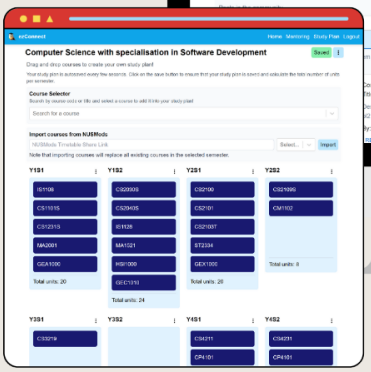
TECH STACK

React
Tailwindcss
Git/Github
SQLAlchemy
Flask
Azure

ARCHITECTURE DIAGRAM







FEATURES

MENTOR MENTEE MATCHER

Find a mentor or mentee for a specific course or for career advice. Emails are hidden before acceptance to protect private information.

STUDY PLAN EDITOR

Plan your study plan using the drag and drop functionality. Add courses by simply searching using the course code or course name. Study plan is automatically saved and total units for each semester is updated. Accommodates users with varied number of semesters.

STUDY PLAN GALLERY

View all published study plans easily to use for reference. Search through the study plans using search and filter. Filter study plans based on their tags (degrees and minors). Order study plans by relevancy, trending, most likes or most recent.

NUS EMAIL SIGN IN

Only users with a valid NUS account can sign in through NUS email to use ezConnect.

STUDY PLAN VALIDATOR

Check if pre-requisites to courses in your study plans are fulfilled in earlier semesters!

INTEGRATION WITH NUSMODS

Import your NUSMods timetable into your study plan and export a semester to a NUSMods timetable. Helps you to reduce the need to type the same courses on both sites!

STUDY PLAN POST

Read about the creator's reasoning and experience to use for consideration. Make a copy of a study plan and modify it to suit your needs. Favourite study plans to reference. Like study plans to make them more visible.

SWE PRACTICES

USER TESTING

Google forms were sent to NUS students to test the application. Testers provided insight, suggestions and found bugs.

AUTOMATED TESTING

We used pytest and Github Actions to test API calls automatically. These tests help us to ensure that the changes we made to the code does not affect the functionality of previous features.

DON'T REPEAT YOURSELF PRINCIPLE

Reuse code by extracting commonly used functions and react components.

CONTAINERISATION

Used Conda and Docker to standardise developer experience. Stored dependencies in lock files like environment.yml, requirements.txt, or package-lock.json.

VERSION CONTROL

We used git to conduct version control using branching and pull requests.

PROJECT MANAGEMENT

We managed the implementation of features and bug fixes through GitHub Issues and Project Board.

Milestone 3 Poster

Video Links

Liftoff: https://drive.google.com/file/d/1I2-btzf9VxkMLeBT-Qd8TSBUll5q3V-o/view?usp=drive_link

Milestone 1:
https://drive.google.com/file/d/1IV7sdR94k7ocj6dPWBZyT74iQO4dsHy4/view?usp=drive_link

Milestone 2:
https://drive.google.com/file/d/1-V66rKcqlB3RMx0gKuZBE1WnX9N5HD3Z/view?usp=drive_link

Milestone 3:
https://drive.google.com/file/d/127dfkwFZk0Ggd46LP9TpH-W1CpL6LTWf/view?usp=drive_link