Hacking Materials User Interface (HA) Project

Client: Dr. Christian Brandl

Supervisor: Mauro Mello Jr

Authored by Team

BlueRing

BoxJelly

RedBack

Table of Contents

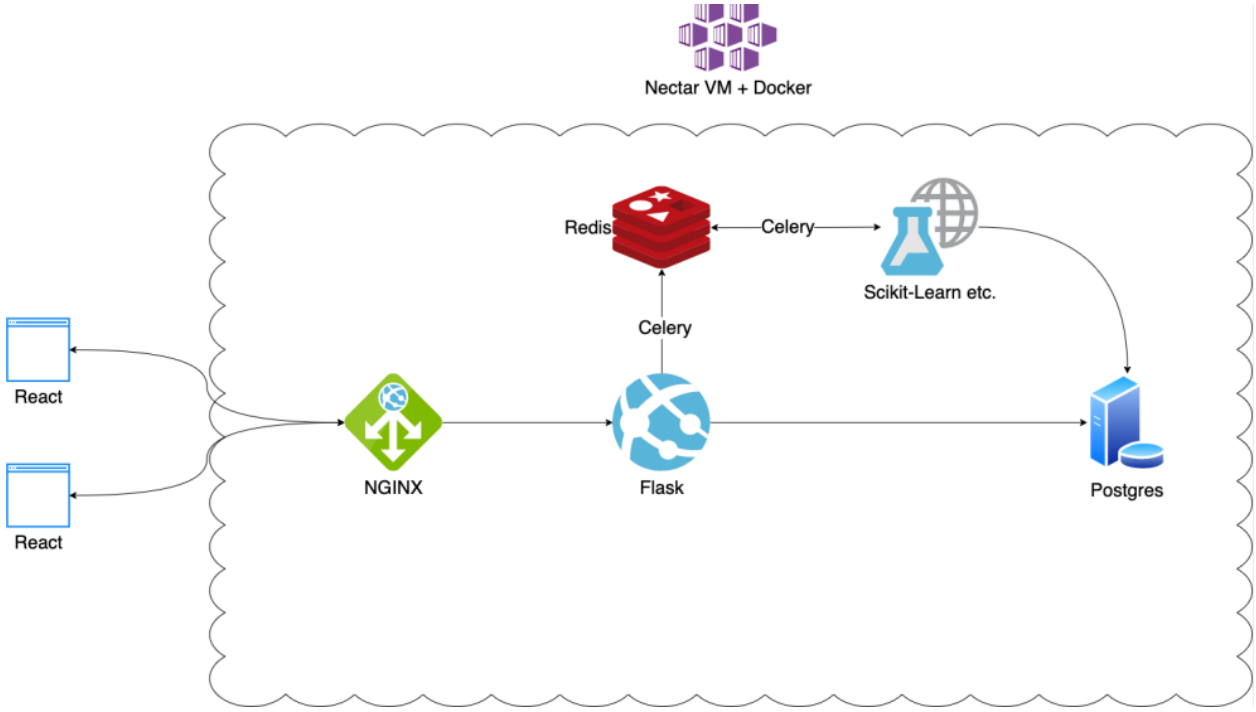
Introduction

The Hacking Materials User Interface (HA) project was conducted by three student teams (BlueRing, BoxJelly and RedBack) in the subject COMP90082 Software Project, 2022 semester 2, at the University of Melbourne.

The project is split into three main parts: frontend, backend and machine learning.

The front end uses React and TypeScript with a storybook for UI prototyping and presentation.
The backend uses Python and Flask, with the key packages being `sqlalchemy`, `matminer`, and `pytest`.
The database is Postgres.
The machine learning is in Python, with the key packages being `sklearn` and `matminer`.

The architecture diagram is as follows:



The workspace follows this storage structure:

Folder	Description
data-samples	Sample data to use in developing and testing the application

docs	Documentation of the project requirements and design
prototypes	Low-fidelity prototypes of the user interface design
src	Source code. Subdirectories are included for backend and frontend development, with machine learning code being considered part of the backend
tests	Documentation on testing the product. In practice, tests are included in the testing folders under `src/backend` and `src/frontend`
ui	Resources required by the user interface and prototypes

How to continue developing the product

- First, you'll need to get access to the project's GitHub repository as described above. If you don't have access to that repository, ask The University of Melbourne COMP90082 Software Project staff for access.
- Once you have a copy of the repository, we strongly suggest that you start by reading the included `README.md` file.
- Getting Started guides for frontend and backend developers are included in the project's GitHub repository in the `DEVELOPERS.md` files under the `src/frontend` and `src/backend` directories. These files provide information about setting up your development environment, installing required tools and libraries, and building the project components.
- Additional documentation and guides are also available within the `docs/` directory under each of the `backend/` and `frontend/` directories.

At minimum, to develop this product you will need:

- A text editor (we recommend Visual Studio Code)
- Docker
- Python
- node.js

How to deploy the product

Prerequisites

1. **Google** API key for SSO login - Authenticate using API keys | Authentication | Google Cloud
2. **Microsoft** API key for SSO login - Enable authentication in a web API by using Azure Active Directory B2C | Microsoft Learn
3. **Email Address** of assigned database administrator

Worth a read:

- Connect over SSH with Visual Studio Code
- Docker - Visual Studio Marketplace

Deployment process

There are three steps to deploying this product.

We have included screenshots of this deployment process in Appendix 9. Deployment process screenshots.

1. Spin up a VM on Nectar ARDC, Google Cloud, AWS or Azure. Ensure you choose an **Ubuntu 22.04 image.**
    a. Use your cloud provider's web interface to set up a virtual machine. For ARDC Nectar, this means:
        i. Create a key pair
        ii. Create a virtual machine, using that key pair
        iii. Enable SSH and HTTPS access to the machine
    b. Once the virtual machine has been deployed, you can access it via SSH using the IP address provided by your cloud provider.
2. Use SSH to connect to the machine.
3. Run `sudo apt update && sudo apt upgrade` to update the existing software on the machine. You may need to restart the machine if the kernel or other important software has been updated.
4. Clone the GitHub repository and run the deployment script using the commands below.

    ```
    sudo apt install git
    git clone https://github.com/COMP90082-2022-SM2/HA-2022-SM2.git
    cd HA-2022-SM2/src/backend/
    git checkout deployment
    ```
5. Ensure the deployment script is runnable, then run it:

```
sudo chmod +x ./deploy_on_ubuntu22_04.sh
./deploy_on_ubuntu22_04.sh
```

The deployment script will:

a. Initialize the directory structure
b. Install and update Docker
c. Start the deployment containers

At three stages, the script will ask for user input.

1. When creating a **self-signed certificate** for HTTPS support it will need information about. If you're deploying to the University of Melbourne's infrastructure, their IT department will likely be able to provide the correct information for you.
2. When initializing the database you will need to provide a **database administrator username and password**.
3. API keys for **Google and Microsoft Single Sign-on APIs**. This is discussed further in Appendix 7. Set-up SSO authorisation guide, but it's also possible that The University of Melbourne IT department can assist with this.

With this done, the script will begin building the project and creating the required containers. At this point you may be able to access the website using your virtual machine's IP address, depending on your browser's security settings. However it's likely that this will not work because the domain of the HTTPS certificate will not match the domain name of the server.

6. Point a domain name to the IP address of the VM. We have used Cloudflare to do this, but again it is likely that The University of Melbourne IT department or your cloud provider can do this for you.

How to test the product

How to use PyTest

The guide can be found in `src/backend/src/tests/testing.md`. Additional resources are available in Appendix 4. Learning Resources.

Where to find our manual test cases

- Refer to Appendix 6. Test Cases.
- High-level setup and preconditions for each test case are included, and the more detailed description can be found in Appendix 6. Test Cases.

Recommendations to future developers

Development practices

How to use common Trello boards between the 3 teams:
Since three teams were working on the same product, a shared epic board and separate user story board for each team were used. This made cross checking the progress of each team a bit difficult. Using one trello board for all teams would be recommended so that progress of tasks is visible for everyone.

How to do code reviews
Quality assurance guidelines were used for code reviews and they can be found under Appendix 2: Development Practices.

Test suggestions for both frontend and backend
For frontend testing, we would recommend testing using Jest with React Testing Library and testing each component separately as well as each of the utils. We recommend these tools as they are common and effective, especially with the frontend technology stack used here. Our technical guides list also includes tutorials for these libraries.

Report bugs in GitHub
Initially there was no way to report bugs for the application. Later GitHub Issues was adopted for bug reporting across the teams. This allowed team members to report problems when they encountered issues while using the application.

Submit early
While working in a combined group of three teams, there can be varying versions of the submissions. Finishing the work earlier gives the teams a chance to review the consistency among the documents. Also, submitting well before the deadline can prevent the end-moment stress and the blunders that come with it.

Branching
The deployment branch is separate to the main branch as the branch's configuration, found in `./src/frontend/nginx`, is designed for deployment. All other files should be kept synced with the main branch whose `./src/frontend/nginx` is designed for local development.

Future work

Data preparation component

Add support for the application of multiple featurizers.
A user story has been completed that allows users to apply a single featurizer to the data before proceeding to the next stage. However, this functionality can be extended by allowing the user to apply multiple featurizers simultaneously.

Add support for multiple column selection for featurization instead of just one.
The current application only allows users to select one column for featurization while the featurizers in matminer can support a list of columns in the dataset. In the future, it would be nice to have the option for users to select multiple columns from the application.

Machine learning component

Plot data type consistency
Currently the plot data is a list of dictionaries in json format in the frontend. No type definition is given for the dictionary items. It may cause problems in further data manipulation and may not be easily extendable. Type definition can be provided in the future.

The identifier field in the plot data is not universal across different datasets. User selection can be added to let users decide what to include as the identifier.

Add support for more machine learning models
The decision tree model and random forest model are obviously not optimal under the current trend of machine learning. Neural network models have gradually become the best models.

In this project, the training of the machine learning model starts after the user selects the dataset and predicts the label. Then, in order to let the user get the reply as soon as possible, the neural network training time is very important.

For future work, more machine learning models can be added to the application to provide more training choices. Besides, the function for the users to select the split ratio between the training data and test data can also be provided to avoid overfitted results due to the same training data and test data. Finally, if the model's comparison can also be made into a visualization among different models, the application's performance can also be further improved.

Use a database cluster

In contrast to decision trees and random forest models, storing a neural network model consumes a lot of space. This can cause database or server crashes. Optimizing existing databases and servers to handle higher loads will help future versions of this product handle this scenario, for example by adding a load balancer that allocates tasks to multiple sub-servers to complete tasks together.

General improvements

Extendability:
The current design assumes that only 30 graduate students will use our product. As the number of users grows, new issues will emerge. Here are a few bottlenecks which could be helpful to future developers.

- There is only one container to handle requests. More servers and containers will be required if the number of users outgrows the server's capability that runs the container.
- There is only one container to store the database. Again, it's possible to run out of storage, or the responses become slow because of hardware limitations.
- Database design can use some optimization. We dump almost everything into one column as a JSON object for flexibility. This approach can slow down some DB operations if new functionalities are added to the product. Another point is that we use user emails as primary keys, which slows down data retrieval operations and could lead to personal information leaks. One solution is to use generated unique tokens as the primary keys.
- The task management component also needs to rework to support multi-server scenarios.

Security:
We didn't consider cyber security, as none of our team members has studied this topic. All messages (some of them have user emails and the configurations of their work) are sent in plain text, which leads to information leaks. Any hacker could easily steal our user's work. Future developers should at least encrypt these messages or find better solutions to minimize the security risks.

Frontend code improvements

Automatically mapping step components to stages
Add the step components in a new field for each step in the `STEPS_DATA` object in the `src/frontend/src/steps/data.tsx` file. Then, the step components can be removed from the `src/frontend/src/sections/appBody/inputPanel/workflowStages/` files. Instead, the stages can be adjusted to automatically render all the steps in `STEPS_DATA` that have the stage number as the first digit in their number. Steps 4 & 5 from the “Adding a new workflow step” process under the “Managing Workflow Steps & Stages” guide in `src/frontend/docs/manage-workflow.md` can be removed.

Execution button consistency
Currently, the execution button does not accomplish anything in Stage 3 as the plots are generated when the Stage 3 step buttons are clicked.
It is suggested that either the Execution button is removed for Stage 3, or the Stage 3 steps be modified to only collect the user’s preferences (e.g. Should a scatter plot be shown? If yes, which columns to use for each axis?) and then the execution button is what triggers the rendering for the plots.

Completing the remaining user stories

There are a lot more features that had been initially identified but marked as out of scope due to time constraints. A list of all user stories that have not been completed are included in Appendix 1 for reference.

Appendices

Appendix 1. User stories that were not completed

Completed user stories are not included here, but note that some of them may still be improved as suggested above.

ID	User Story	Size (days)	Priority
1	As a student, I want to clean and tune data input Input Data, so that I have less noise on visualizations.	5	3 - Could have
2	As a general user, I want to attach comments to workflow objects Jupyter Notebook, so that I can document my work	5	3 - Could have
3	As a Pro user, I want to be able to add new datasets in the future External Data, so that if there's a new dataset that can be used on a new project, it can be added instantly	5	3 - Could have
4	As a Pro user, I want to edit python code on the interface Jupyter Notebook, so that I can have control how the ML algorithms works	5	3 - Could have
5	As a Pro user, I want to upload my own script (in python) if possible Jupyter Notebook, so that I can extend the tool to support custom models and featurizers	5	3 - Could have
6	As a pro user, I want to have access to more processing power Administration, so that I can run more complex operations or use more data	5	3 - Could have
7	As a pro user, I want to combine multiple models together Machine Learning, so that I can model more complex data manipulations	5	3 - Could have
11	As a student, I want to analyze the relationship between different features Machine Learning, so that I can identify which features I need to select for my analysis	5	2 - Should have
12	As a student, I want to export my work to a Jupyter Notebook Jupyter Notebook, so that I can extend my work beyond the capability of the application	5	2 - Should have
13	As a Pro user, I want to add new features Input Data, so that they can be reused in the future	5	2 - Should have
14	As a pro user, I want to easily find and read documentation on the pro features Administration, so that I can use them with ease	5	2 - Should have
15	As a pro user, I want to be able use additional ML models Machine Learning, so that I can improve accuracy	5	3 - Could have
16	As a general user, I want to add specific materials to the workflow for	3	3 - Could have

ID	User Story	Size (days)	Priority
	analysis Machine Learning, so that compare the performance of the specific material my client or I choose with other material		
17	As a pro user, I want to be kept informed about job status Administration, so that I can avoid polling my workspace to check for results	3	3 - Could have
18	As a pro user, I want to be able to apply new featurizers Input Data, so that I can create new features	3	3 - Could have
19	As a student, I want to quickly browse the Materials available in the database for retrieval and simulations Input Data, so that I can quickly perform queries.	3	2 - Should have
24	As a student, I want to receive provided hints and guidance for new users Administration, so that I can quickly learn how to use software	3	2 - Should have
25	As a general user, I want to select specific features from a dataset Input Data, so that I can improve the precision of my model	3	2 - Should have
26	As a pro user, I want to have the option to change the hyperparameters used in the machine learning model Machine Learning, so that I can fine tune my test results.	3	2 - Should have
27	As a pro user, I want to be able to access new databases External Data, so that I can access additional data	3	2 - Should have
28	As a general user, I want to be able to reference / view citations for original data sources Input Data, so that I can retrieve data.	1	3 - Could have
37	As a general user, I want to be able to preview the output of each execution Input Data, so that I could explore the data	5	2 - Should have
38	As a pro user, I want to have my pro user settings persist on each visit Administration, so that I don't have to reconfigure settings to use the features I need	1	2 - Should have
39	As a user, I want to be able to select split ratio of data Machine Learning, so that to train and test the model	1	2 - Should have

Appendix 2. Development guidelines

Quality assurance

Work quality guidelines

1. Code should be able to run. No syntax or compile-time errors.
2. If unit or integration tests are included in a pull request, they should pass
3. All preexisting tests should still pass with the new changes in the pull request
4. A branch should not be put up for a pull request if it has merge conflicts with main. All conflicts should be resolved **before** requesting review of a pull request.
5. Code should be understandable and contain documentation

Code review guidelines

1. Each PR must be reviewed by one member from each of the other two student teams.
2. Pull requests should:
    1. Be assigned reviewers within **24 hours** of being submitted
    2. Be given initial feedback by reviewers within **48 hours** of being submitted
3. Commit messages should describe the work you've done and the steps you took to verify the correctness of your work
4. Pull request descriptions should summarize the commit messages
5. Pull request changes should be reviewed by using the Review Changes button under the Files Changed tab, to encourage the use of **Accept** or **Review Changes** messages.
6. Pull requests should only be merged by the creator of the pull request
7. Branches that have been successfully merged to main should be deleted by the creator of the pull request
8. Reviewers should check that pull requests follow the work quality guideline above

Acceptance criteria definition guidelines

1. Acceptance criteria should be defined from the user's point of view
2. Acceptance criteria should contain a list of steps to test the desired functionality

Definition of "done" for a user story

1. Acceptance criteria should be defined for the user story, and should pass
2. All related code has **passed code review and merged to main**

Development processes

Sprint lifecycle

Sprint kickoff

Attendees:

Everyone

Tasks:

- Pull user stories into the sprint from the product backlog
- Ensure user stories have complete definitions:
    - Do they have acceptance criteria?
    - Do they have test cases?
    - Are they ready for development? What are their dependencies?
- Re-estimate user stories with T-shirt size and priority
- Schedule kickoff meetings for user stories that are ready for development
- Review action items from the previous sprint's retrospective meeting

The aim of the sprint kickoff meeting is to define the high-level objectives for the sprint and consolidate learnings from previous sprints.

Feature kickoff

Attendees:

The product owner, plus a subject matter expert for each domain the user story will involve. As an example, this might include:

- The product owner
- A frontend developer
- A backend developer
- A user experience (UX) designer
- Someone who worked on a related feature who may be able to give useful information

Tasks:

- Conduct a design discussion for the user story
- Break the user story down into multiple smaller tasks
- For each task, define:
    - Dependencies
    - Size (using **magic estimation**)
    - Relevant test cases and acceptance criteria

    - Assigned developer
- Create tasks in Trello

Development

Branching

- Use the format feature/t-<ticket> as a feature branch template, where <ticket> is the Trello card number
- Where branches have other purposes, prefixes other than "feature" may be used. For example:
    - "spike", for experimental work that shouldn't be merged
    - "fix", for fixes to existing features
- In general, follow Git Flow as a guide to using branches for development (we probably only need main, release, scratch, and feature branches)

Style

- Use auto-formatters and linters to maintain consistent code style.
    - For Python, we use black and pylint
    - For TypeScript, we use Prettier

Tests

Where tests are available, **remember to run them before submitting a pull request(PR)**. Reviewers should check that tests pass before approving PRs for merge.

Code reviews

Reviewers should:

- Verify that the pull request description includes:
    - A description of **what work was done**, and **why it was needed**. Usually this just means a link or reference to the relevant user story or Trello card.
    - A description of **how the developer knows their work is correct**. The reviewer can then use this to cross-check the code.
- Verify that the tests pass. At minimum, this means checking that the automated tests run by GitHub Actions passed. For integration and manual tests, see How to test the product, Appendix 5. Acceptance criteria, and Appendix 6. Test cases.
- Verify that all test cases and acceptance criteria identified in the relevant feature kickoff have been satisfied.

At least one developer from each of the other two teams **must** approve the pull request before it can be merged.

Deployment

See How to deploy the product

Sprint retrospectives

Each sprint should end with a sprint retrospective.

If multiple teams are involved in the project, the team representatives should also conduct separate retrospectives to discuss team interactions.

Releases

Versioning

Most of this is defined by the university. In general though, we use semantic versioning, and release by tagging releases on GitHub.

Appendix 3. Core technologies

Front-end

This project is primarily accessed via a website, which we're building with ReactJS and Styled Components. The development language is Typescript.

We are using Storybooks to automate documentation and previewing for developed UI components.

Back-end

The front-end is supported by a Flask application, which interacts with Matminer on the user's behalf and surfaces required information in the UI.

The development language is Python. We're also using Pytest and Sphinx for running unit tests and generating documentation.

Key libraries we're using include Matminer and SQLAlchemy.

Our database is Postgres, and all back-end services are built as Docker containers. We have not yet selected a deployment technology, and are discussing options with the Melbourne University IT department.

Appendix 4. Learning resources

Backend

- Pytest: https://docs.pytest.org/en/7.1.x/getting-started.html
- SQLAlchemy: https://docs.sqlalchemy.org/en/14/dialects/postgresql.html
- Flask :
    - https://flask.palletsprojects.com/en/2.2.x/
    - https://www.tutorialspoint.com/flask/index.htm
    - https://youtu.be/Z1RJmh_OqeA(Quick guide from setup, creating virtual env similar to the Developer.md , connecting to SQLAlchemy and the frontend)
    - https://github.com/realpython/discover-flask

Accessing workflow related data

Workflow configurations data

WorkflowConfigurations class in workflow_configurations.py is a data class in the backend containing all the configurations needed for executing each stage in the whole workflow. This class is created and saved to the database when a new user is created.

The choices made by users from the frontend are captured in this class and then saved in the Users table in the database.

When new choices are added in the frontend, the WorkflowConfigurations class should be extended as well to accommodate the changes, so choices made by the user can be consistent throughout the whole application.

save_workflow and load_workflow functions in workflow_data_handler.py can be used in the backend to save and load the WorkflowConfigurations object to and from the database.

Dataset for machine learning and plotting

In the first stage of the whole workflow, featurized data will be created after the stage execution. This data is then also saved in the Users table in the database. Using save_featurized_data in workflow_data_handler.py will allow you to add the dataframe to the database if there's any change to it.

To access the data for machine learning, or plotting, load_featurized_data in workflow_data_handler.py should be used to return the data.

Frontend

- React: https://reactjs.org/tutorial/tutorial.html
- Style components: https://www.robinwieruch.de/react-styled-components/
- Testing:
    - RTL Cheat sheet
    - RTL + Jest tutorial
    - Jest cheat sheet
- TypeScript:
    - TS Cheat sheet
    - TS tutorial
    - TS + React tutorial
- Storybooks:
    - Tutorial (start from step 2)
    - Official “how to write stories” guide

Docker

Docker is a tool for managing isolated runtime environments (called "containers"). You can install Docker by following the Docker guide (linked below).

You can think of it as a combination of:

- A way to run virtual machines
    - But it's much more efficient
    - And makes it easier to access host machine resources, for example network ports and filesystems
- A way to build custom virtual machines, just like make can be used to build binaries or applications
    - The build instructions are described by the Dockerfile file

Bear in mind though that **containers are not virtual machines** - that analogy is useful for describing an overview of what Docker does, but it is not accurate.

The two most important things you can do with Docker are:

- Build container **images**, for example by running: `docker image build -t flask-docker` .
    - A container *image* is like a compiled program: it does nothing unless it is run
    - Unlike a program though, it doesn't exist in an obvious way on the filesystem. The Docker runtime is responsible for managing images.

    - `-t flask-docker` tells Docker to "tag" the image with the string "flask-docker". You can refer to Docker objects (including images) by their hash, but naming them makes it easier.
    - The `.` tells Docker to use the Docker file in the current directory. You could provide a path to another directory instead.
- Run a container image, for example by running `docker run -d -p 80:5000 flask-docker`
    - Running an image produces an object called a "container" - this is like a process, just as an image is like a compiled program.
    - The `-d` argument tells Docker to "detach" the container - to run it in the background and not show its output.
    - The `-p 80:5000` argument tells Docker to map port 80 on the host machine to port 5000 inside the container. If you're building a Flask application (which by default listens for requests on port 5000), this means that you would access the application by navigating to port 80 on the host machine.
    - `flask-docker` is the tag of the image we want to run.

Once a container is running, it is given a tag by the Docker runtime. This tag is separate from the tag of the image.

You can view running containers by running `docker ps`.

When you're finished with a container, you can kill it by running `docker kill <container-tag>`

For more information, you can go through the Docker getting started guide, or refer to their reference documentation.

Docker-compose

An alternative to Docker approach detailed above.

Docker-compose can automate the process of setting up & managing multiple containers. See Install Docker Compose

Before running the command below create a .env file in this directory and add the following lines

```
POSTGRES_USER="db_user"
POSTGRES_PASS="somehardpassword"
PGA_USER="user@gmail.com"
PGA_PASS="anotherhardpassword"
```

Replace the text inside the inverted commas with appropriate email/password/usernames

Then create a `.pgpassfile` in the postgres-init folder and add the following line

`pg_db:5432:ha_db:username:password`

Replacing username and password with the postgres username and password chosen above

For security, ensure `.env` and `.pgpassfile` are added to .gitignore

Running docker-compose up -d in this folder will set up 3 containers with persistent storage

- Flask app as defined in **Dockerfile**, available at 127.0.0.1:5000.
- postgres database available at 127.0.0.1:5432.
- PGAdmin, a UI to easily manage the database 127.0.0.1:5050. pgAdmin 4 Docs

Using a docker development container

While it's a little unusual, you can also use Docker to set up a **development container**. This is a container that you would use to create an isolated environment for software development - for example, if your main machine is a Windows machine, but you want to use a Linux development environment.

Visual Studio Code includes some tools for this:
https://code.visualstudio.com/docs/remote/containers

In this kind of setup, you would create a **separate container** for your development environment, and connect to it from Visual Studio Code.

From the VSCode terminal, you would be able to access the container via a shell just as you could on a normal Linux machine.

Inside the container you could install Docker and develop as normal. However, there could be some issues with accessing web services hosted inside the development container. It may be simpler to use the **Windows Subsystem for Linux** instead:
https://code.visualstudio.com/docs/remote/wsl

Appendix 5. Acceptance criteria

STORY ID	USER STORY	GIVEN	WHEN	THEN
21	Select datasets from existing databases	I'm at the dataset selection step and I can see a list of datasets	I select one of the datasets and click on save	I should receive some feedback on whether the action is successful or not
23	Control job execution	I have a task in progress	I click in the start, view and cancel buttons	The running tasks can be started, viewed and canceled
32	Browse and select built-in featurizers	I'm at the featurizer selection step	I click on the dropdown menu	I see a list of all built-in featurizers with readable names
		I'm at the featurizer selection step and I can see a list of featurizers	I select one of the featurizers and click on save	I should receive some feedback on whether the action is successful or not
34	Browse built-in datasets	I'm at the dataset selection step	I click on the dropdown menu	I see a list of all built-in datasets with readable names
35	Export model selections, parameters, and data flows	I am logged into my account	I click on the export button	I can download a file that has all the data needed to recreate my current workflow state
41	View the columns of the saved dataset and select the column I want to featurize	I've selected a dataset and saved my selection	I click on the dropdown menu	I see a list of column names in the dropdown menu
		I'm at the data columns selection step and I see a list of column names	I select one of the data columns and clicked on save	I should receive some feedback on whether the action is successful or not
20	Create an account using single-sign on, restricted to the *.unimelb.edu.au domain	I have a registered account for use of this product	I update any relevant personal information	That updated information persists in the database and is linked to my account
			I make changes to my workflow	That workflow data can be saved in the database and linked to my account. Only I can access it

STORY ID	USER STORY	GIVEN	WHEN	THEN
			I delete my account	All linked data is deleted from the database. Others' data won't be affected
		I own a Unimelb account and would like to access the web application features	I click on the login button	I can sign in using my unimelb.edu.au email as a form of authentication
		I am a first-time user and don't own a Unimelb account	I click on the login button	If I try to sign in using a non-Unimelb email I receive a notification that only unimelb.edu.au emails are allowed
29	Save project specific data/checkpoints	I am logged into my account	I make changes to my workflow	That data can be saved to the database under my user account
35	Export model selections, parameters, and data flows	I am logged into my account	I click on the export button	I can download a file that has all the data needed to recreate my current workflow state
36	Import exported model selections, parameters, and data flows	I am logged into my account	I click on the import button	I can upload a previously exported workflow state
		I have uploaded a previously exported workflow state	The file passes all data integrity checks	My UI will match the state represented in the imported file
10	opt in to pro-user features	I am a pro user and would like to access my pro-user features	I click on the opt button	I jump into a new window with all pro-user features accessible.
23	Control job execution	I have a task in progress	I click on the start, view and cancel buttons	The running tasks can be started, viewed and canceled
33	Browse Machine Learning models	I have selected the target feature and I'm at the model selection step	I click on the dropdown menu	I should see a list of Machine Learning models with readable names
31	Select a Machine Learning model	I'm at the model selection step and I can see a list of Machine Learning models	I select one of the Machine Learning models and click on save	I should receive some feedback on whether the action is successful or not

STORY ID	USER STORY	GIVEN	WHEN	THEN
39	be able to select split ratio of data	I've selected a dataset and saved my selection	I type in a ratio of train and test data and click the continue button	I should receive some feedback on whether the action is successful or not
26	Browse and be able to change the hyperparameters used in the machine learning model	I successfully select one of the Machine Learning models	I receive the feedback that my selection is succeed	I should see some adjustable functionality for the hyperparameters of the model I selected
		I see some adjustable functionality for the hyperparameters of the model I selected	I adjust the values of these hyperparameters and click the continue button	I should receive some feedback on whether the action is successful or not
15	be able use additional ML models	I have selected the target feature and I'm at the model selection step	I click on the dropdown menu	I see enough Machine Learning models with readable names
7	Combine multiple models together	I have selected the target feature and I'm at the model selection step	I receive the successful feedback from feature selection	I should see multiple dropdown menus that allow me to select multiple Machine Learning models to combine
		I see multiple dropdown menus that allow me to select multiple Machine Learning models to combine	I select several different models	I should receive some feedback on whether the action is successful or not
40	Select features used for x and y axis	I've selected a dataset and saved my selection	I click on the dropdown menu	I see a list of column names in the dropdown menu
		I am at the x and y axis selection step and I see a list of column names	I select two of the data columns as x and y axis and clicked on execution	I should receive some feedback on whether the action is successful or not
9	View and plot the results of the model	I successfully select x and y axis used for plotting	I click the plot button	I can see plots about the selected columns

STORY ID	USER STORY	GIVEN	WHEN	THEN
8	Browse different regression models	I successfully select x and y axis used for plotting	I click on the dropdown menu of selecting regression models	I see several types of regression models
8	See different results of plotting graphs	I see several types of regression models	I choose one of the regression models and click on the plot button	I see a correct regression plot of my selection

Appendix 6. Test cases

Definitions

Standard setup

This is the minimum set of requirements for a user to log in and run ML jobs.

- Backend, Frontend, Databases, Job Queue, and ML Service are available
- Google API key has been registered
- Databases have been created

Test cases

US30

Test Type	Functional
Execution Type	Manual
Objective	Users are able to view citations, authors, and help text for featurizers
Setup	Standard setup

Execution Steps	1. Log in using a Melbourne Uni student account
	2. Select a featurizer from the featurizers dropdown menu
	3. Click the "Featurizer Details" button
	4. **Verify:**
	    1. **Citations** are shown for the selected featurizer
	    2. **Implementors** are shown for the selected featurizer
	    3. **Help text** is shown for the selected featurizer
	    4. A list of **generated labels** is shown for the featurizer

Notes	A good featurizer for testing this is "Miedema" There's no guarantee that every featurizer has all four of the items to be verified, but that one has all of them. Remember to check the browser and backend logs for errors.
Time constraint	1m per tested featurizer

US32

Test Type	Functional
Execution Type	Manual
Objective	Users are able to select a featurizer from a dropdown list
Setup	Standard setup; Use either PGAdmin or a psql connection to the database to verify changes

Execution Steps	1. Log in using a Melbourne Uni student account
	2. **Verify:** The featurizers dropdown is displayed in step 1.3, and has content
	3. Select a featurizer, click "Save"
	4. **Verify:**
	    1. A "Your selection was successfully saved" message appears
	    2. The featurizer selection is communicated to the backend and the database is updated

Notes	
Time constraint	2m per tested featurizer

US34 and US21

Test Type	Functional
Execution Type	Manual
Objective	The user can browse and select built-in datasets
Setup	Standard setup; Use either PGAdmin or a psql connection to the database to verify changes

Execution Steps	1. Log in using a Melbourne Uni student account
	2. **Verify:** The datasets dropdown is displayed in step 1.1, and has content
	3. Select a dataset, click "Save
	4. **Verify:**
	  1. A "Your selection was successfully saved" message appears
	  2. The dataset selection is communicated to the backend and the database is updated

Notes	
Time constraint	2m per tested dataset

US41

Test Type	Functional
Execution Type	Manual
Objective	Users can select a column to featurize from the columns available in the dataset they have already selected

Setup	Standard setup, plus: ● Use either PGAdmin or a psql connection to the database to verify changes ● Use a Python notebook to verify the list of columns that should be listed for a given dataset

Execution Steps	1. Log in using a Melbourne Uni student account
	2. Select a dataset from the step 1.1 dropdown, and click "Save"
	3. **Verify:**
	  1. A "Your selection was successfully saved" message appears
	  2. The dataset columns dropdown is displayed in step 1.2, and has content
	  3. The list of columns matches those listed by Matminer
	4. Select a dataset column
	5. **Verify:** The dataset column selection is communicated to the backend and the database is updated

Notes	
Time constraint	5m per tested dataset (to verify the columns match)

Foundation

Foundation TC01

Test Type	Functional
Execution Type	Manual
Objective	Users can step through the different stages of the workflow, and saved inputs are preserved.
Setup	Standard setup

Execution Steps	1. Log in using a Melbourne Uni student account 2. In step 1.1, select a dataset and click "Save" 3. **Verify:** a "Your selection was successfully saved" message appears 4. Step through to stage 2 5. **Verify:** the output from step 3 is still visible 6. Step through to stage 3 7. **Verify:** the output from step 3 is still visible 8. Step back to stage 1 9. **Verify:** the dataset previously selected is still selected
Notes	The dataset selection should only be preserved if the user clicked "Save"; if the selection was not saved it does not need to be preserved.
Time constraint	2m

Foundation TC02

Test Type	Functional
Execution Type	Manual
Objective	Output messages are not overwritten by new messages
Setup	Standard setup

Execution Steps	1. Log in using a Melbourne Uni student account
	2. In step 1.1, select a dataset and click "Save"
	3. **Verify:** a "Your selection was successfully saved" message appears
	4. Bring down the back-end systems
	5. Select a dataset column from the step 1.2 dropdown menu
	6. **Verify:**
	  1. An error message appears in the output pane
	  2. The previous "Your selection was successfully saved" message is still shown and is not replaced.

Notes	The dataset selection should only be preserved if the user clicked "Save"; if the selection was not saved it does not need to be preserved.
Time constraint	2m

US23

Test Type	Functional

Execution Type	Manual
Objective	Users can execute a featurizer
Setup	Standard setup

Execution Steps	1. Log in using a Melbourne Uni student account
	2. Select a dataset from the step 1.1 dropdown, and click "Save"
	3. **Verify:**
	  1. A "Your selection was successfully saved" message appears
	  2. The dataset columns dropdown is displayed in step 1.2, and has content
	  3. The list of columns matches those listed by Matminer
	4. Select a dataset column from the step 1.2 dropdown, and click "Save"
	5. **Verify:** The dataset column selection is communicated to the backend and the database is updated
	6. Select a featurizer column from the step 1.3 dropdown, and click "Save"
	7. **Verify:** The featurizer selection is communicated to the backend and the database is updated
	8. Click "Run Stage 1"
	9. **Verify:**
	  1. The "Run Stage 1" button becomes disabled
	  2. Job execution begins (Use Docker logs or Flower to verify this)
	  3. Job execution ends successfully
	  4. The "Run Stage 1" button becomes enabled again
	  5. A success message appears in the output pane

Notes	Depending on the combination of dataset, formula, and featurizer, this might not work because they're not all compatible. A combination that should work is: ● Dataset: Elastic Tensor 2015 ● Column: Formula ● Featurizer: StrToComposition
Time constraint	12m

US10: Login with the pro user(successful)

Test Type: Functional	**Execution type:** Manual
Objective: Pro users are able to log in with a pro-user feature	
Setup The databases have been created, and a google API key has been registered for the project	
Pre-condition 1. A registered user has a pro-user access	
Notes: [1]: Click on Start exploring on the home page *application pop-up a google SSO [2]: When a google pop-up reveals, tick the "i am pro user". *application shows ticked marked on "i am pro user" [3]: Click sign in with google	
Time constraint: Minimum: 10 seconds Maximum: 1 minute	

US 20

US 20, TC01: create a new account (successful)

Test Type: Functional	**Execution type:** Manual
Objective: Users are able to create an account using an SSO with his/her student account.	
Setup The databases have been created, and a google API key has been registered for the project	
Pre-condition 1. The user does not have an account on the databases.	
Notes: [1]: Click on Start exploring on the home page *application pop-up a google SSO [2]: When a google pop-up reveals, click on login using google, then it will create an account using the student email account. *application redirects to the /workflow	
Time constraint: Minimum: 10 seconds Maximum: 1 minute	

US 20, TC02: existing user login (successful)

Test Type: Functional	**Execution type:** Manual
Objective: Users are able to log on to an account using SSO with his/her student account.	
Setup The databases have been created, and a google API key has been connected to the system	
Pre-condition 1. The user has an account on the databases.	
Notes [1]: Click on Start exploring on the home page *application pop-up a google SSO [2]: When a google pop-up reveals, click on login using google, then log in using the student email account. *application redirects to the /workflow	
Time constraint: Minimum: 10 seconds Maximum: 1 minute	

US 29

US 29, TC01: Save selected dataset (successful)

Test Type: Functional	**Execution type:** Manual
Objective: The dataset user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user has logged in. 2. The user selected a dataset at step 1.1	
Notes: [1]: Click on the save button at step 1.1 * Application shows “Step 1.1 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”	
Time constraint: Minimum: 1 second Maximum: 10 seconds	

US 29, TC02: Save selected dataset column (successful)

Test Type: Functional	**Execution type:** Manual
Objective: The dataset column user selected is saved successfully.	

Setup The database has been established in the back end.
Pre-condition 1. The user saved the dataset successfully. 2. The user selected a dataset column at step 1.2
Notes [1]: Click on the save button at step 1.2 * Application shows “Step 1.2 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”
Time constraint: Minimum: 1 second Maximum: 10 seconds

US 29, TC03: Save selected featurizer (successful)

Test Type:	**Execution type:**
Functional	Manual
Objective: The featurizer user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user saved the dataset column successfully. 2. The user selected a featurizer at step 1.3	

Notes [1]: Click on the save button at step 1.3 * Application shows “Step 1.3 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”
Time constraint: Minimum: 1 second Maximum: 10 seconds

US 29, TC04: Save selected machine learning model (successful)

Test Type: Functional	**Execution type:** Manual
Objective: The machine learning model user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user saved the featurizer successfully. 2. The user selected a machine learning model at step 2.1	
Notes [1]: Click on the save button at step 2.1 * Application shows “Step 2.1 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”	

Time constraint: Minimum: 1 second Maximum: 10 seconds

US 29, TC05: Save selected property (successful)

Test Type: Functional	**Execution type:** Manual
Objective: The property user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user saved the machine learning model successfully. 2. The user selected property at step 2.2	
Notes [1]: Click on the save button at step 2.2 * Application shows “Step 2.2 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”	
Time constraint: Minimum: 1 second Maximum: 10 seconds	

US 29, TC06: Save selected x-axis feature (successful)

Test Type:	**Execution type:**
Functional	Manual
Objective: The x-axis feature user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user saved the property successfully. 2. The user selected an x-axis feature at step 3.1	
Notes [1]: Click on the save button at step 3.1 * Application shows “Step 3.1 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”	
Time constraint: Minimum: 1 second Maximum: 10 seconds	

US 29, TC07: Save selected y-axis feature (successful)

Test Type:	**Execution type:**
Functional	Manual
Objective: The y-axis feature user selected is saved successfully.	

Setup The database has been established in the back end.
Pre-condition 1. The user saved the x-axis feature successfully. 2. The user selected a y-axis feature at step 3.2
Notes [1]: Click on the save button at step 3.2 * Application shows “Step 3.2 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”
Time constraint: Minimum: 1 second Maximum: 10 seconds

US 29, TC08: Save selected regression function (successful)

Test Type:	**Execution type:**
Functional	Manual
Objective: The regression function user selected is saved successfully.	
Setup The database has been established in the back end.	
Pre-condition 1. The user saved the y-axis feature successfully. 2. The user selected a regression function at step 3.4	

Notes [1]: Click on the save button at step 3.4 * Application shows “Step 3.4 output:” * Application shows a green box where a notice saying, “Your selection was successfully saved”
Time constraint: Minimum: 1 second Maximum: 10 seconds

US35, TC01: Export workflow (Successful)

Test Type: Functional	**Execution type:** Manual
Objective: Exporting current workflow into .json state	
Setup The databases have been created, a google API key has been registered for the project, and the user has logged on.	
Pre-condition 1. A registered and logged-in user 2. The user has selected workflow step 1. Prepare Data and 2. Machine learning step that he want to save as a json file	
Notes: [1]: Click the export workflow *Application downloads the JSON step of the workflow into the download folder user desired	

Time constraint: Minimum: 10 seconds Maximum: 1 minute

US36, TC01: Import workflow (successful)

Test Type: Functional	**Execution type:** Manual
Objective: Importing a .json file into the workflow	
Setup The databases have been created, a google API key has been registered for the project, and the user has logged on.	
Pre-condition 1. A registered and logged-in user	
Notes: [1]: Click import workflow *Application open a windows folder for user to upload their saved workflow in .json format [2]: Click the desired .json files *Application will have all the .json	
Time constraint: Minimum: 10 seconds Maximum: 1 minute	

US 31: As a general user, I want to be able to select a Machine Learning model.

US31, TC 01: Select a model (successful)

Test Type: Functional	**Execution Type:** Manual
Objective: Verify if a machine learning model is correctly selected.	
Setup: The IO stream has been established between the front end and the back end.	
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has selected the target feature.	
Notes: [1] Select the machine learning model in the interface. Must select a choice in the given place. Do not select the models that are out of scope in the given choices (Linear regression model and Random forest model). [2] Submit the information. * Application starts machine learning using the given input dataset and selected feature.	
Time constraint: Minimum: 10 min Maximum: 30 min	

US31, TC 02: Select a model (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual
Objective: Verify if a machine learning model is correctly selected.	
Setup: The IO stream has been established between the front end and the back end.	
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has selected the target feature.	
Notes: [1]Try not to select the machine learning model in the interface. [2] Submit the information. * Application cannot learn without a selected machine learning model.	
Time constraint: Minimum: 10 min Maximum: 30 min	

US 40: As a general user, I want to select features used for the x-axis and y-axis.

US40, TC 01: Select an x-axis feature (successful)

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if a feature for the x-axis is correctly selected.

Setup:

The IO stream has been established between the front end and the back end.

Pre-Condition:

1. The user has inputted the data or selected the data.

2. The user has executed the machine learning model.

Notes:

[1] Select the feature for the x-axis in the interface.

Must select a choice in the given place.

Do not select the features that are out of scope in the given choices (The features are displayed in the interface).

[2] Click the Execute button.

* Application receives the x-axis feature value.

Time constraint:

Minimum: 1 min

Maximum: 5 min

US40, TC 02: Select an x-axis feature (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if a feature for the x-axis is correctly selected.

Setup:

The IO stream has been established between the front end and the back end.

Pre-Condition:

1. The user has inputted the data or selected the data.

2. The user has executed the machine learning model.

Notes:

[1] Try not to select a feature for the x-axis in the interface.

[2] Click the Execute button.

* Application cannot learn without a selected x-axis feature value.

Time constraint:

Minimum: 1 min

Maximum: 5 min

US40, TC 03: Select a y-axis feature (successful)

Test Type:	**Execution Type:**
Functional	Manual

Objective: Verify if a feature for the y-axis is correctly selected.
Setup: The IO stream has been established between the front end and the back end.
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has executed the machine learning model.
Notes: [1] Select the feature for the y-axis in the interface. Must select a choice in the given place. Do not select the features that are out of scope in the given choices (The features are displayed in the interface). [2] Click the Execute button. * Application receives the y-axis feature value.
Time constraint: Minimum: 1 min Maximum: 5 min

US40, TC 04: Select a y-axis feature (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual

Objective: Verify if a feature for the y-axis is correctly selected.
Setup: The IO stream has been established between the front end and the back end.
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has executed the machine learning model.
Notes: [1] Try not to select a feature for the y-axis in the interface. [2] Click the Execute button. * Application cannot learn without a selected y-axis feature value.
Time constraint: Minimum: 1 min Maximum: 5 min

US 09: As a general user, I want to be able to view and plot the results of the model.

US09, TC 01: Plot the diagram (successful)

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if a diagram can be displayed correctly.

Setup:

The IO stream has been established between the front end and the back end.

Pre-Condition:

1. The user has inputted the data or selected the data.

2. The user has executed the machine learning model.

3. The user has selected features for the x-axis and the y-axis.

Notes:

[1] Click the Plot button.

Must select an x-axis and the y-axis feature value before plotting the diagram.

* Application receives the requests and plots the diagram.

Time constraint:

Minimum: 1 min

Maximum: 10 min

US09, TC 02: Plot the diagram (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if a diagram can be displayed correctly.

Setup:

The IO stream has been established between the front end and the back end.

Pre-Condition:

1. The user has inputted the data or selected the data.

2. The user has executed the machine learning model.

Notes:

[1] The required choices are not selected:

  [1.1] Not select the x-axis feature.

  [1.2] Not select the y-axis feature.

[2] Click the Plot button.

* Application cannot plot the diagram without the x-axis and y-axis feature values.

Time constraint:

Minimum: 1 min

Maximum: 10 min

US 08: As a student, I want to use different types of plotting graphs

US08, TC 01: Select a type of regression model (successful)

Test Type:	**Execution Type:**
Functional	Manual

Objective:

Verify if a regression model is correctly selected.

Setup:

The IO stream has been established between the front end and the back end.

Pre-Condition:

1. The user has inputted the data or selected the data.

2. The user has executed the machine learning model.

Notes:

[1] Select the regression model in the interface.

Must select a choice in the given place.

Do not select the regression models that are out of scope in the given choices (The regression model choices are displayed in the interface).

[2] Click the Execute button.

* Application receives the regression model.

Time constraint:

Minimum: 1 min

Maximum: 5 min

US08, TC 02: Select a type of regression model (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual
Objective: Verify if a regression model is correctly selected.	
Setup: The IO stream has been established between the front end and the back end.	
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has executed the machine learning model.	
Notes: [1] Try not to select a regression model in the interface. [2] Click the Execute button. * Application cannot execute without a selected regression model value.	
Time constraint: Minimum: 1 min Maximum: 5 min	

US08, TC 03: Create the regression plot (successful)

Test Type:	**Execution Type:**
Functional	Manual

Objective: Verify if a regression plot can be displayed correctly.
Setup: The IO stream has been established between the front end and the back end.
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has executed the machine learning model. 3. The user has selected features for the x-axis and the y-axis. 4. The user has selected the regression model.
Notes: [1] Click the Plot button. Must select an x-axis, the y-axis feature value, and a regression model before plotting the diagram. * Application receives the requests and plots the diagram.
Time constraint: Minimum: 1 min Maximum: 10 min

US08, TC 04: Create the regression plot (unsuccessful)

Test Type:	**Execution Type:**
Functional	Manual

Objective: Verify if a regression plot can be displayed correctly.
Setup: The IO stream has been established between the front end and the back end.
Pre-Condition: 1. The user has inputted the data or selected the data. 2. The user has executed the machine learning model.
Notes: [1] The required choices are not selected: [1.1] Not select the x-axis feature. [1.2] Not select the y-axis feature. [1.3] Not select the regression model. [2] Click the Plot button. * Application cannot plot the diagram without filling in the required choices.
Time constraint: Minimum: 1 min Maximum: 10 min

Appendix 7. Set-up SSO authorisation guide

1. Background information

The first version of this application is set-up to be only accessible via SSO (Single SignOn) login restricted unimelb.edu.au email accounts.

Emails with unimelb.edu.au domain are either provided by:

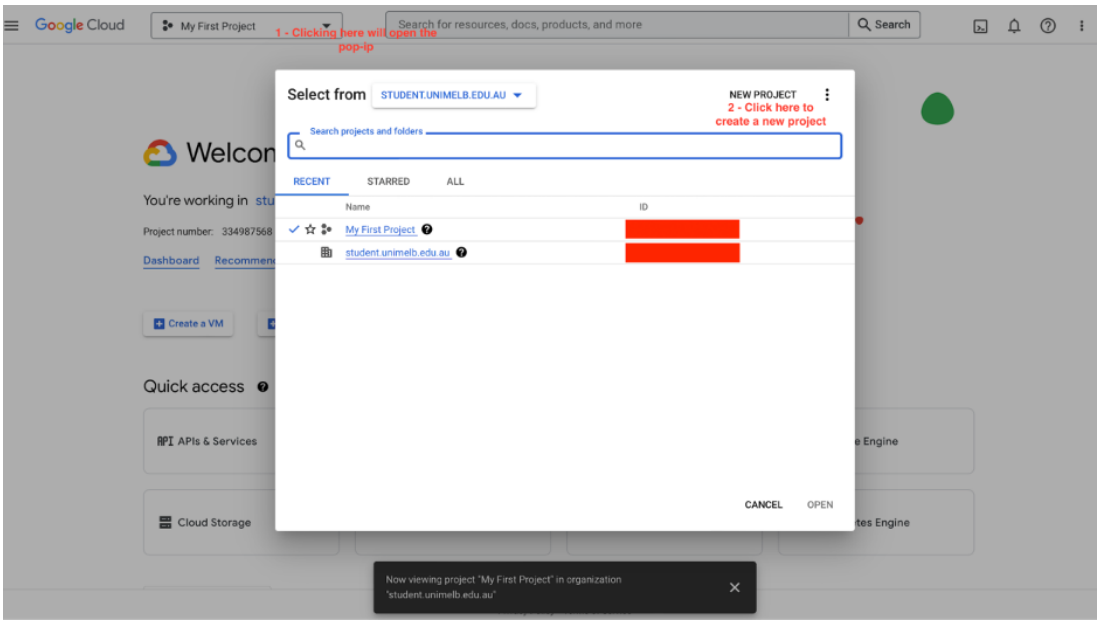
- Students accounts: Gmail
- Staff accounts: Microsoft

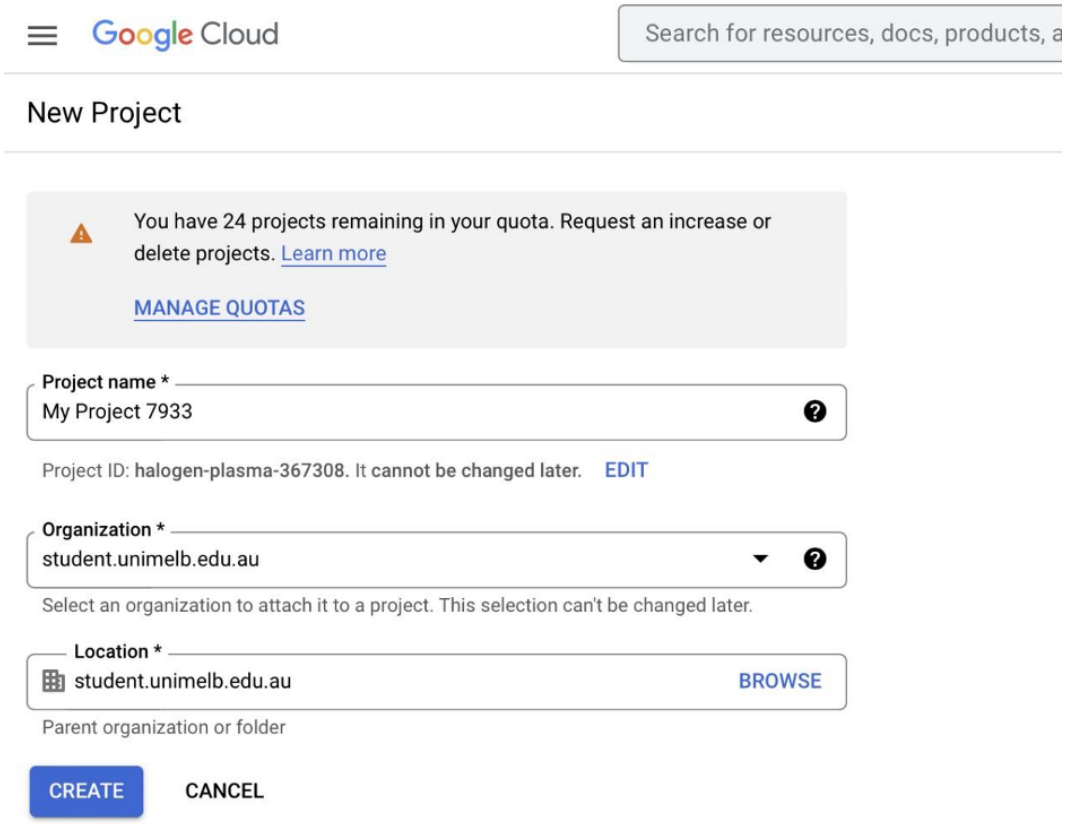
In order to be able to use Gmail and Microsoft SSO authentication the following steps must be done:

1. Create a client ID in Google Cloud Console and Microsoft Portal Azure and associate it with Hacking Materials Applications.
2. Client ID is a unique identifier associated with an application that assists with client and server OAuth 2.0 authentication.
3. Once the client ID is created plug it in specific places in the source code **before deploying the application.**

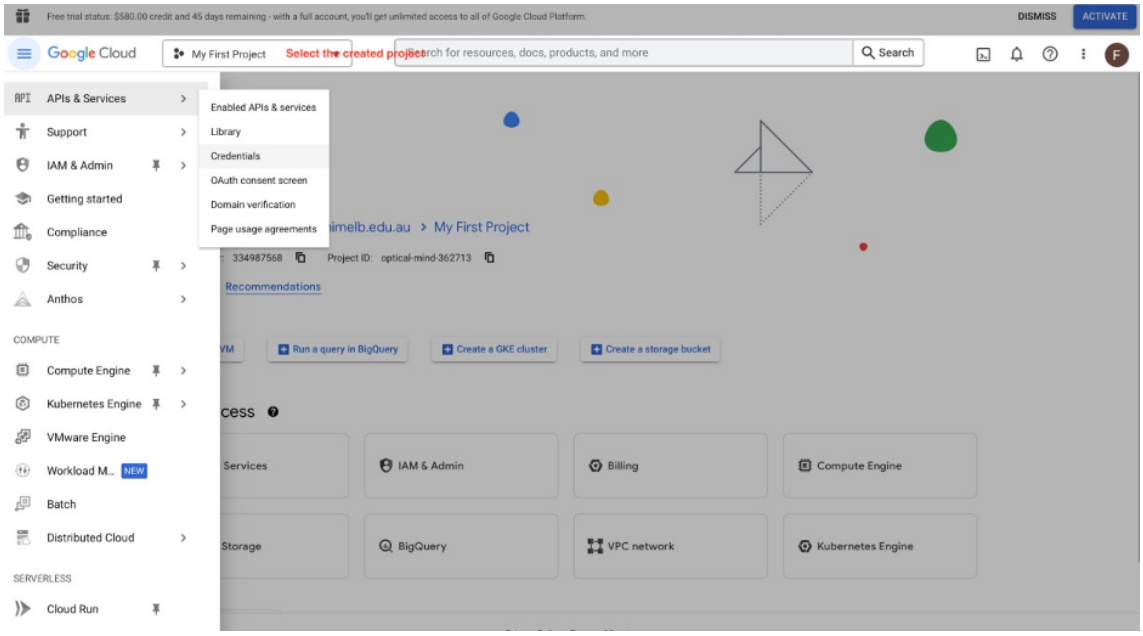
2. Create a client id in the Google Cloud Console

To get a client ID from Google, create a google account, go over to your Google Cloud Console and create a new project.

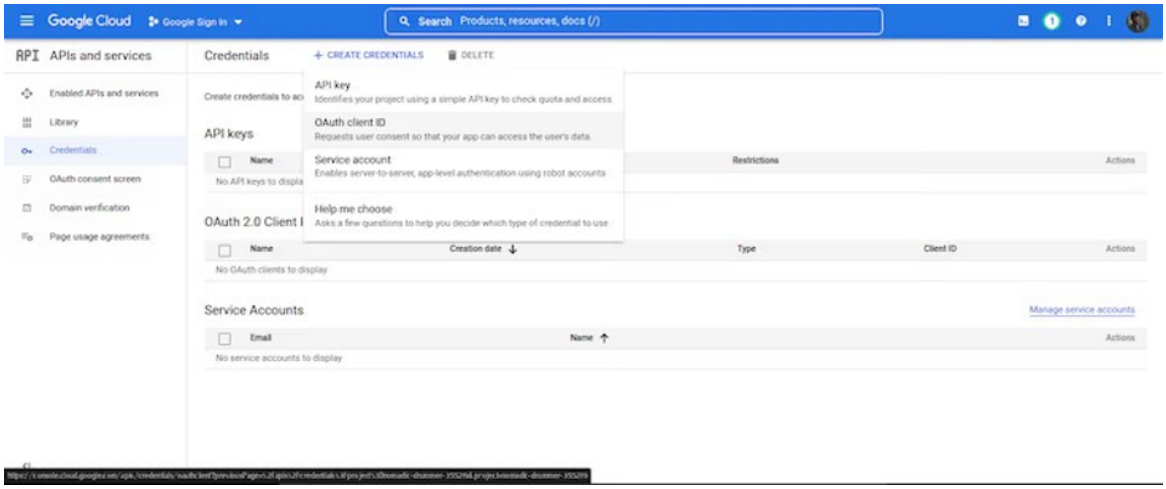




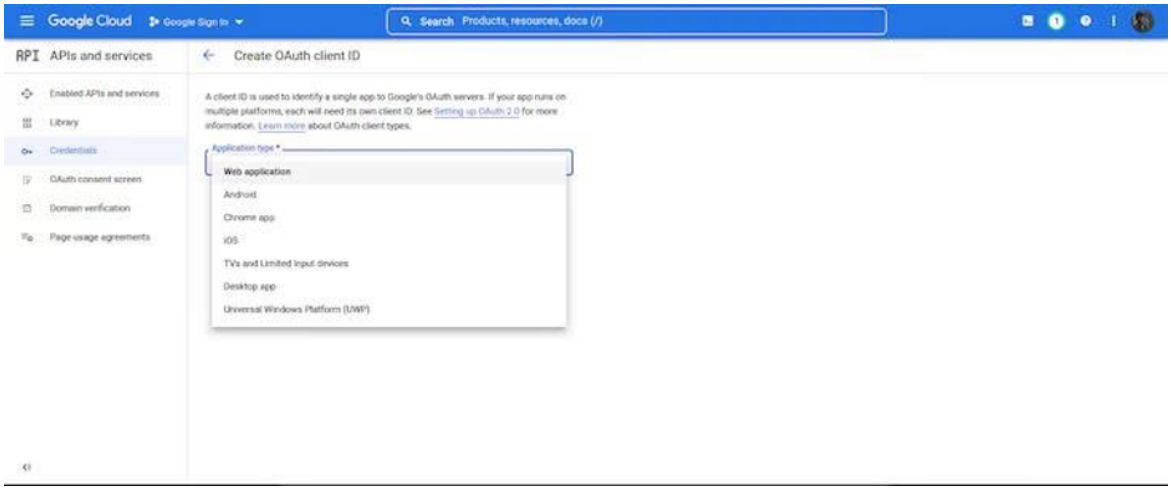
After creating the project. Go back to the console, select the project in the dropdown list, from the left side menu select APIs & Services > Credentials



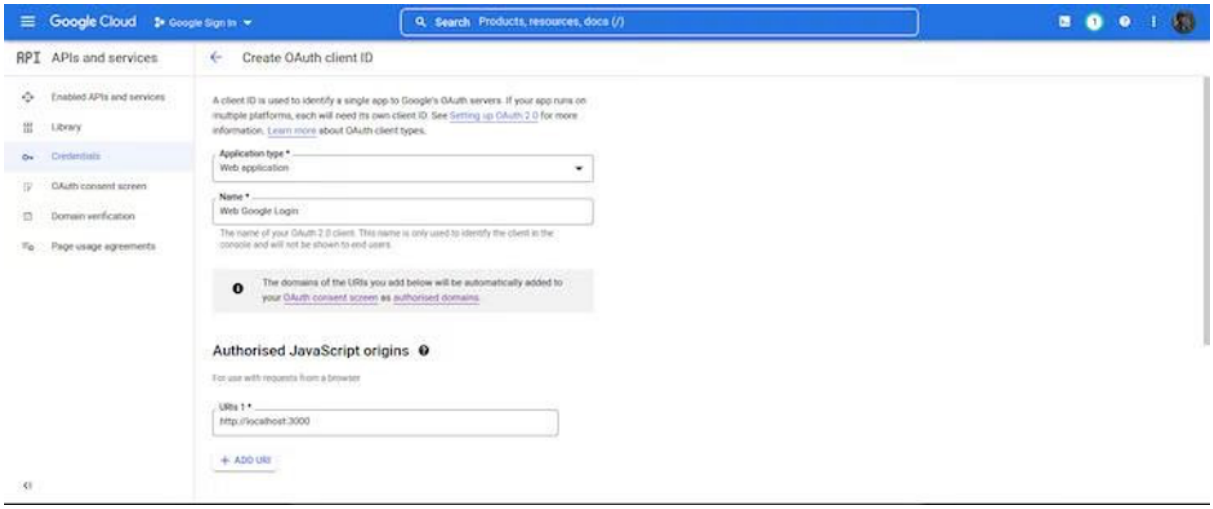
Click + create credentials > Oauth client ID



For Application Type select Web application.



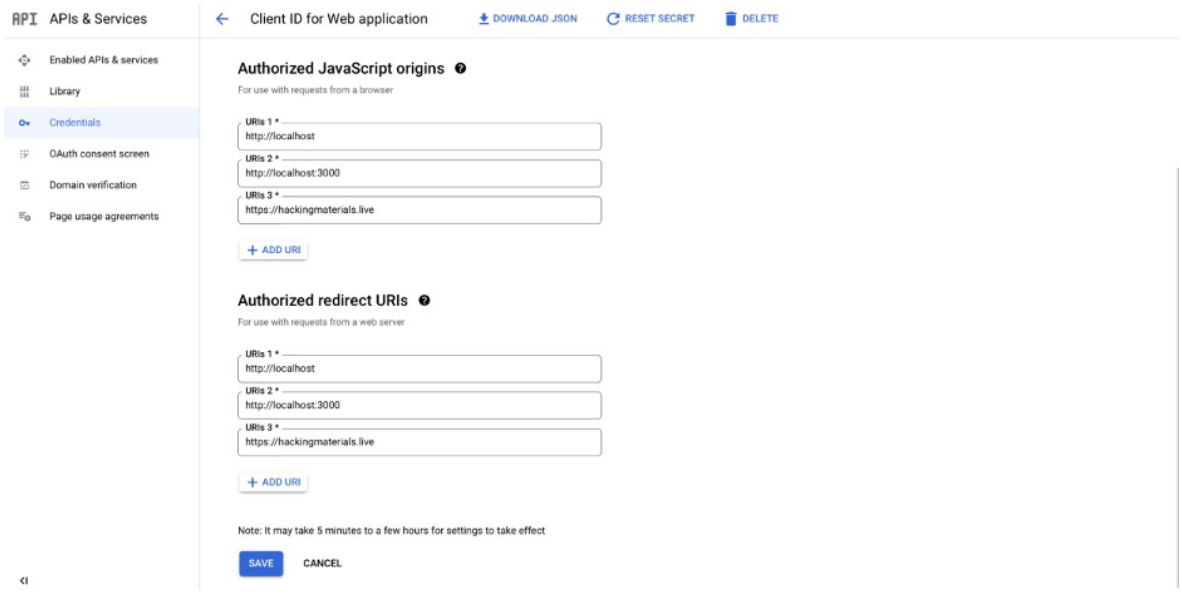
Next, we will choose a name for our client ID. This name is simply used to note or define the particular ID we are creating. So for example, if we are creating web, iOS, and Android IDs, we can include “Web ID,” “Android ID,” “iOS ID,” and so on in their names to differentiate them.



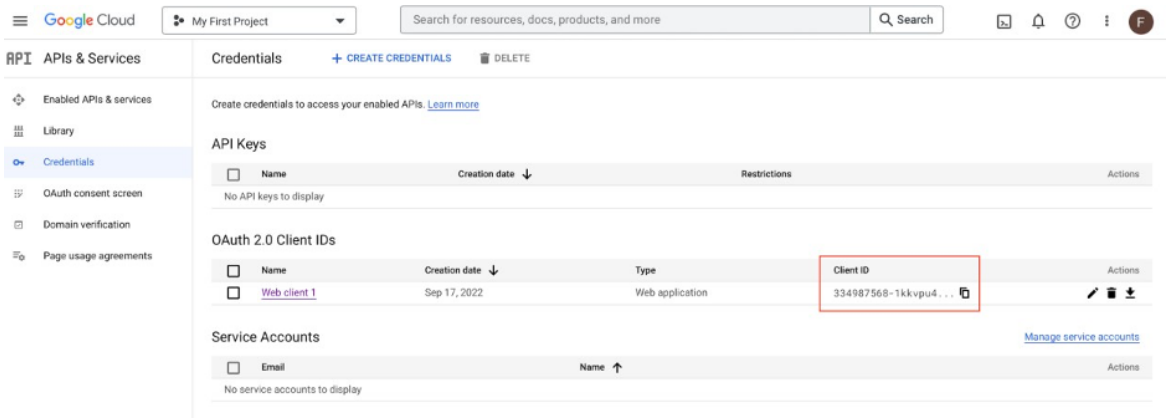
Next, we will also be adding two types of URLs: authorized JavaScript origins and authorized redirect URLs.

The authorized JavaScript origins URL is the URL from which your application is originating the login. In this case we originate the login from the home page. Therefore if the hosting address is http://hackingmaterials.live , this should be the value to be filled here.

The authorized redirect URL is the link that Google will redirect the user to after the login is successful. In this case we configured it to take to the original link.

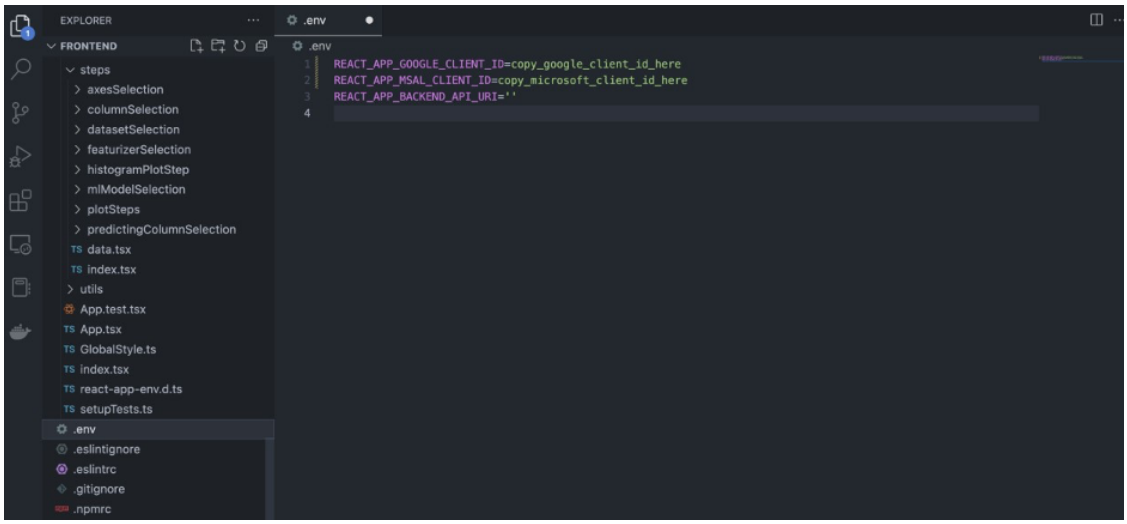


Finally, save, click on the “Create” button to create your web client ID. You will be taken back to the homepage, where you will see your newly created credential. Click on the copy icon to copy your new web client ID.



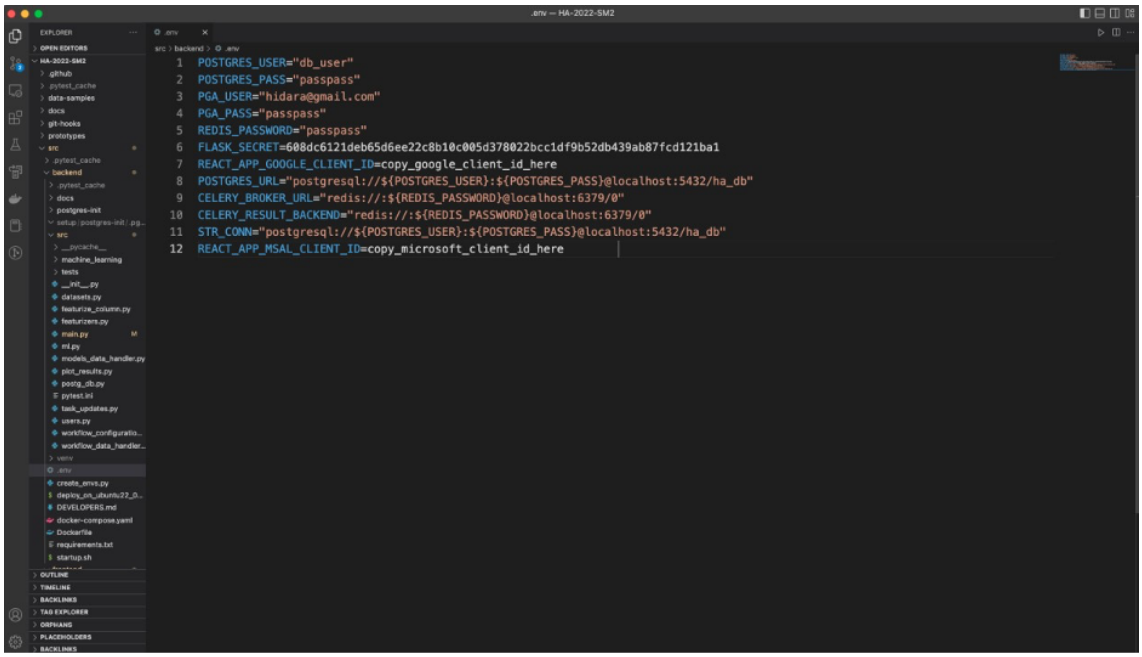
3. “PLUG” the Google client ID in the frontend source code.

In the FRONTEND folder go to.env file and replace **copy_google_client_id_here** with the client ID created in the previous step.



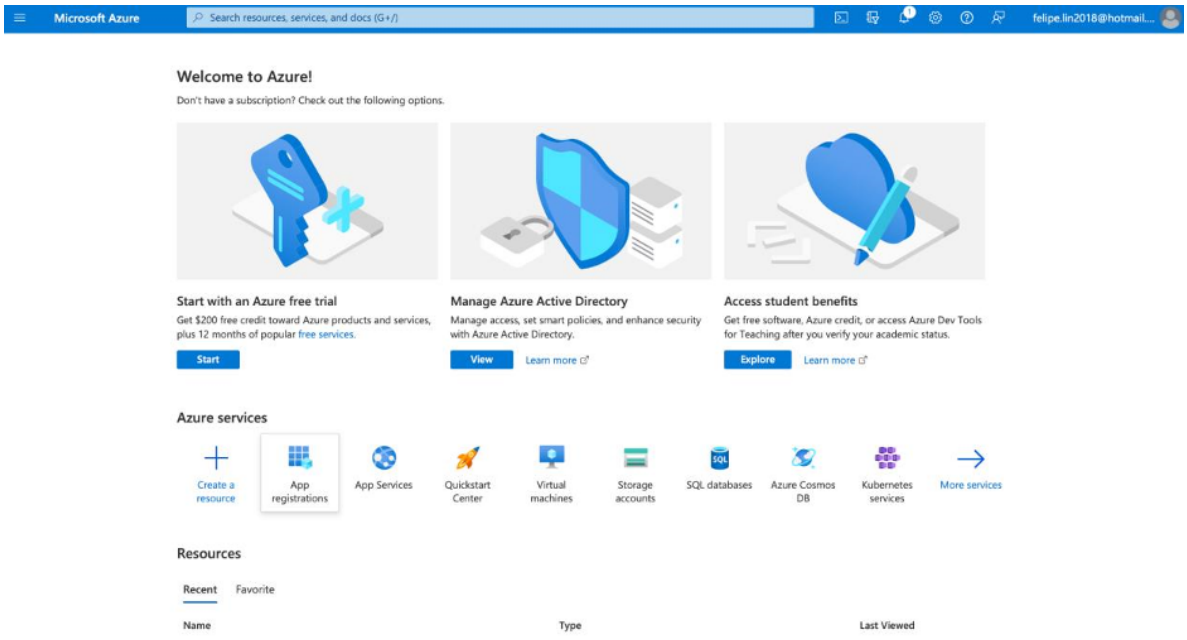
4. “PLUG” the Google client ID in the backend source code

In the BACKEND folder go to.env file and replace **copy_google_client_id_here** with the same client ID.

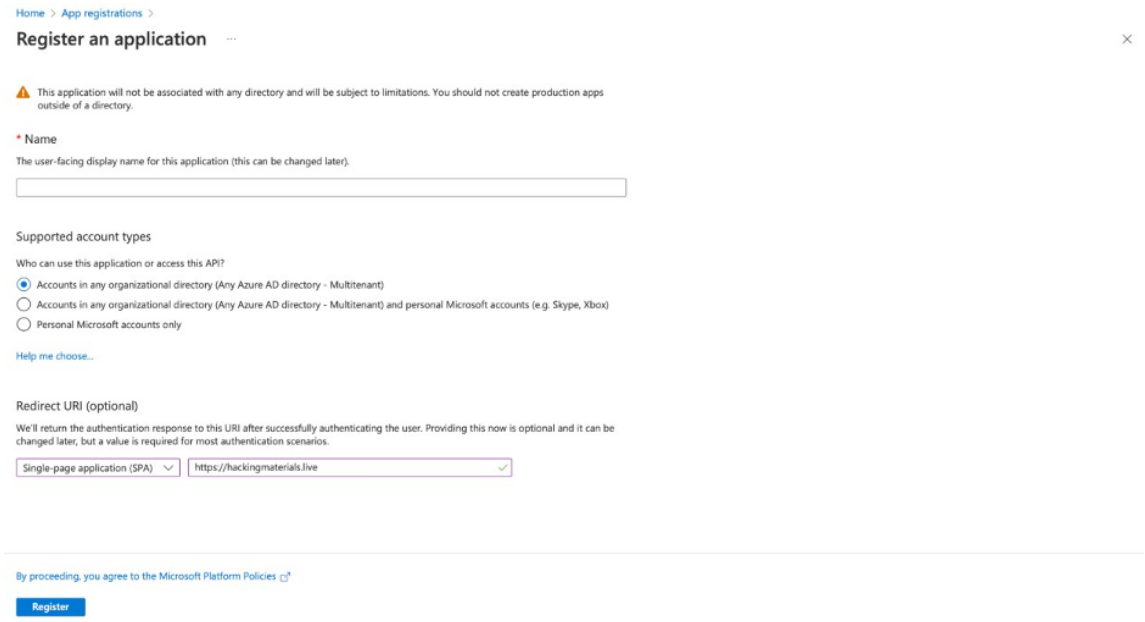


5. Create a client id from Microsoft Azure’s Cloud Portal

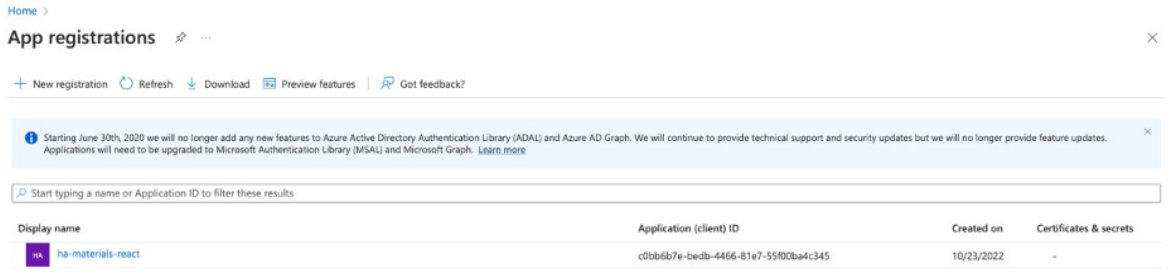
After creating a microsoft email account, after signed-in got to https://portal.azure.com/#home, select App registrations.



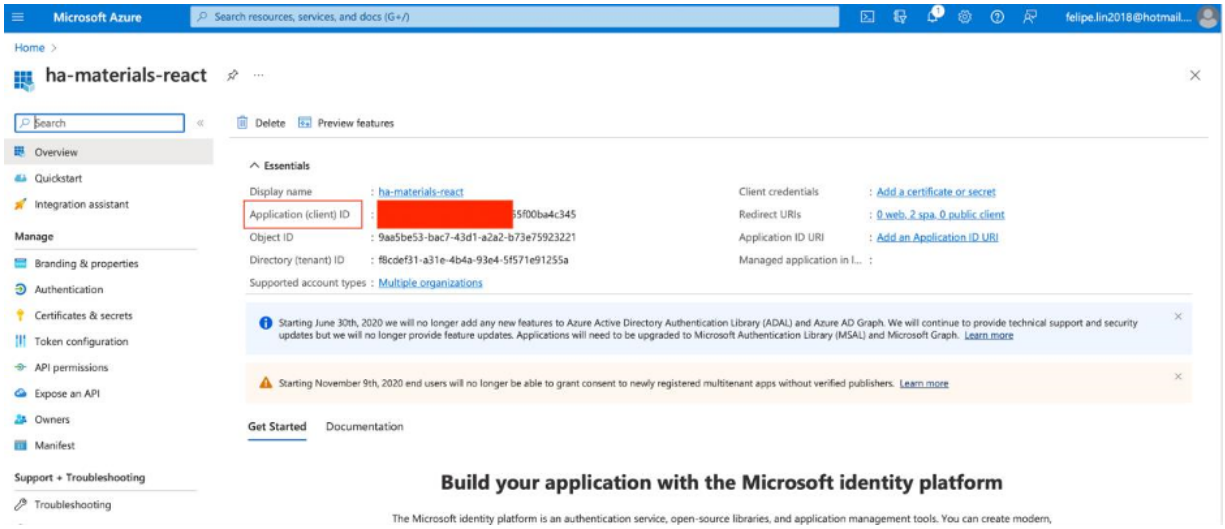
1. Enter a **Name** for your application. Users of your app might see this name, and you can change it later.
2. Choose the **Supported account types (Accounts in any organizational directory to restrict to org accounts such as unimelb.edu.au)** for the application. Choose Single-page application (SPA) and register a Redirect URI.
3. Select **Register** to create the app registration.



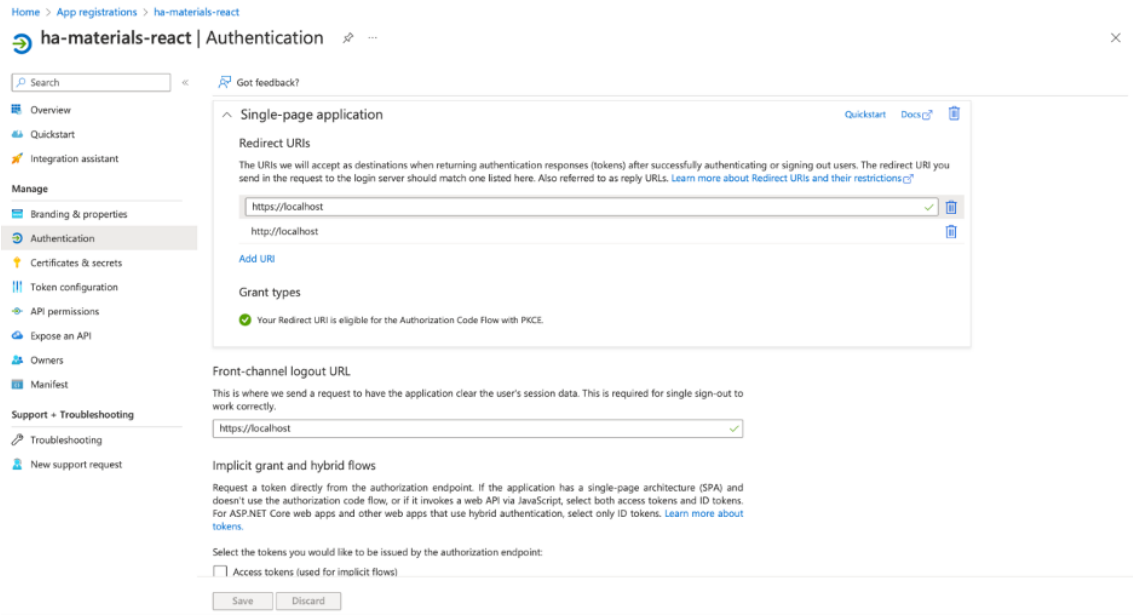
Once registered, select App Registrations and select the application created.



On the application dashboard you can find the Application (client) ID.

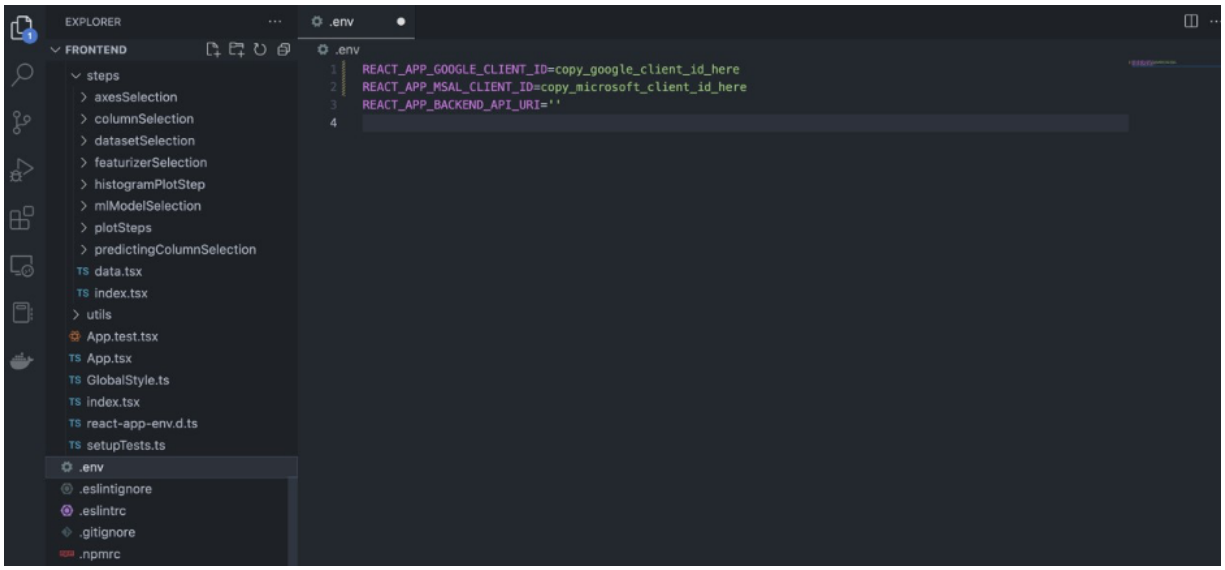


(optional) In case you need to change the redirect URI, it can be done under the **Authentication** tab.



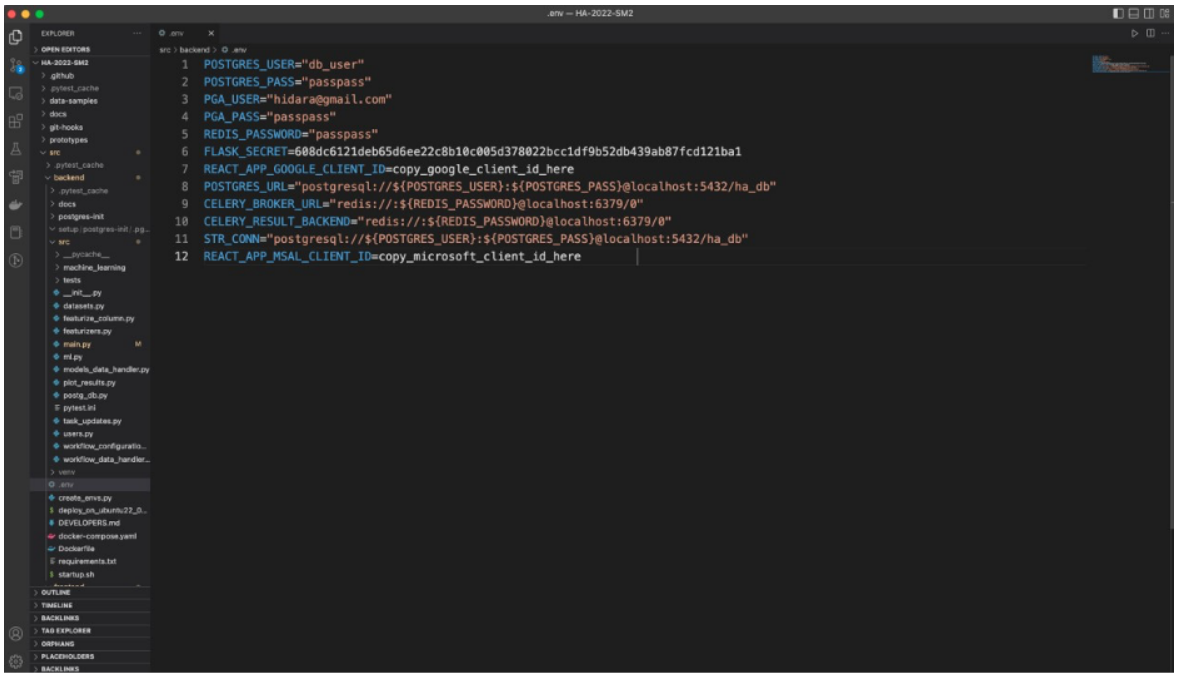
6. “PLUG” the Microsoft client ID in the frontend source code.

In the FRONTEND folder go to the `.env` file and replace **copy_microsoft_client_id_here** with the client ID created in the previous step.



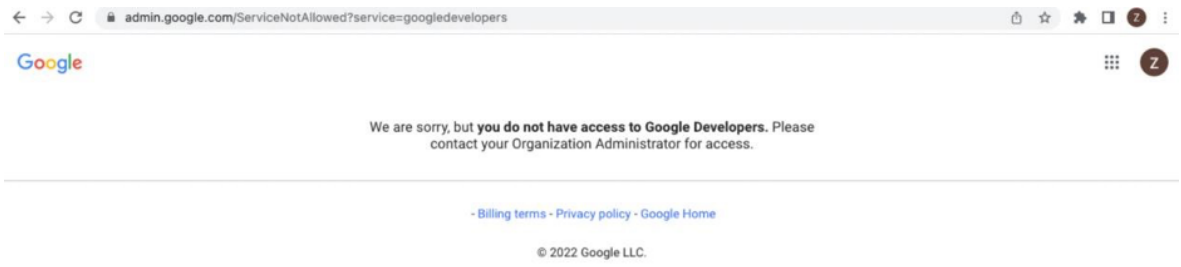
7. “PLUG” the Microsoft client ID in the backend source code

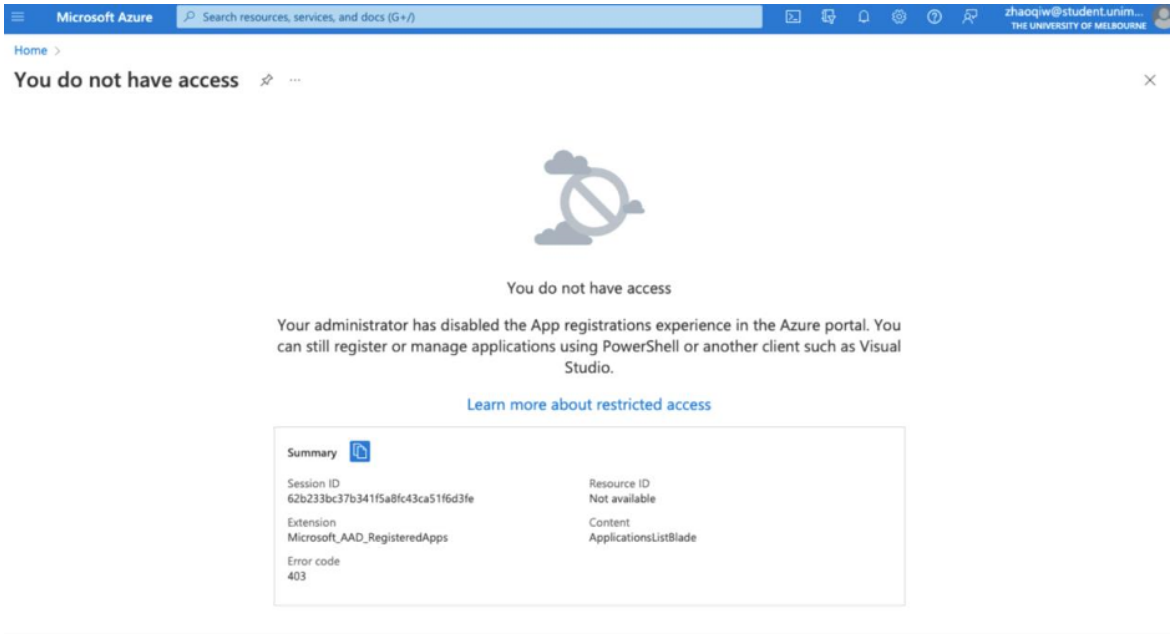
In the BACKEND folder go to.env file and replace **copy_microsoft_client_id_here** with the same client ID.



8. If you get the following error pages when creating Google Client Id or Microsoft Id

If you get one of the below pages while creating either the Google client ID or Microsoft client ID while using an organizational account, you might need to contact the administrator to set the permissions accordingly.




Microsoft Azure
Search resources, services, and docs (G+/)
zhaoqiw@student.unim...
THE UNIVERSITY OF MELBOURNE
Home
You do not have access
You do not have access
Your administrator has disabled the App registrations experience in the Azure portal. You can still register or manage applications using PowerShell or another client such as Visual Studio.
Learn more about restricted access
Summary
Session ID
62b233bc37b341f5a8fc43ca51f6d3fe
Resource ID
Not available
Extension
Microsoft_AAD_RegisteredApps
Content
ApplicationsListBlade
Error code
403

Appendix 8. Frontend workspace structure

PROPOSAL - 1st September 2022

The proposed structure would follow this rough directory tree:

```
|_assets
|_src
        |_components
                |_exampleComponent
                        |_examples.tsx
                        |_index.tsx
                        |_test.tsx
                        |_styled.tsx
                |_ dropdownSelectStepType
                        |_examples.tsx
                        |_index.tsx
                        |_test.tsx
                        |_styled.tsx
                ...
        |_steps
                |_datasetSelection // (e.g.)
                        |_index.tsx
                        |_test.tsx
                        |_HelpModal
                                |_index.tsx
                                |_styled.tsx
                                ...
                        ...
                ...
        |_sections
                |_appHeader
                        |_index.tsx
                        ...
                |_appBody
                        |_InputPanel
                                |_index.tsx
                                ...
                        |_ViewingWinow
                                |_index.tsx
                                ...
                        |_index.tsx
                |_appFooter // amendment suggested by Felipe
                        ...
        |_App.tsx
        ...
|_package.json
```

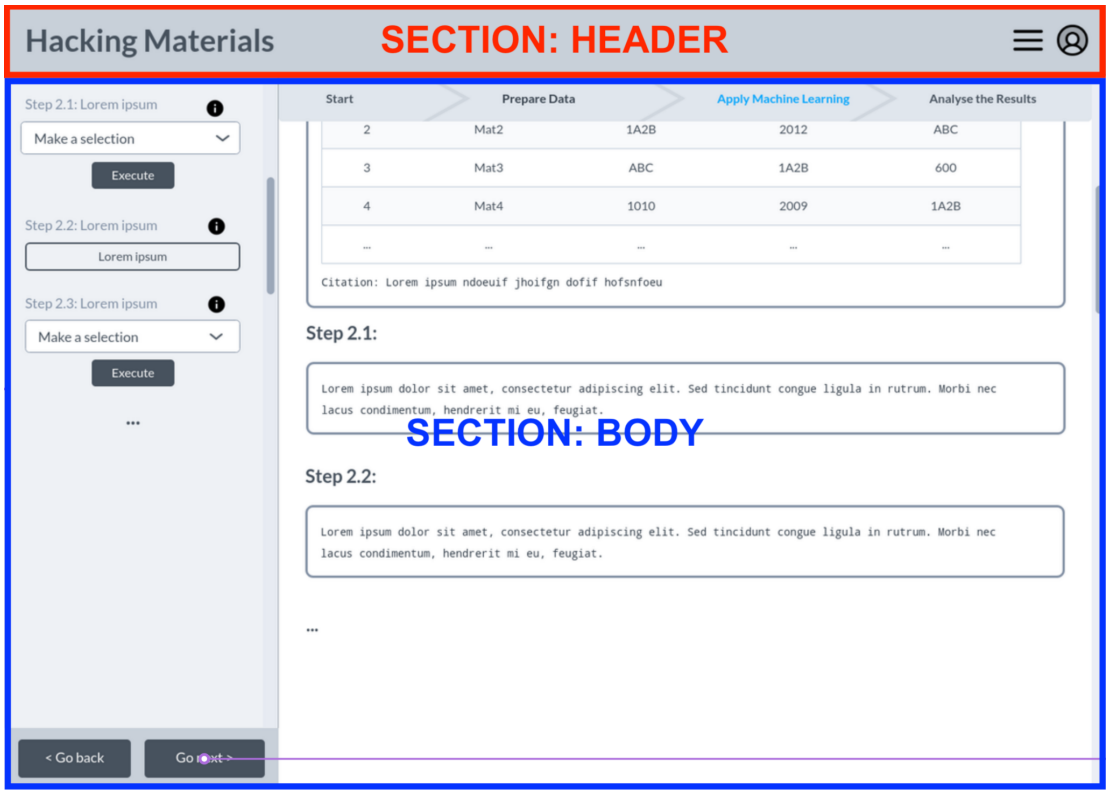
```
|_README.md
...
```

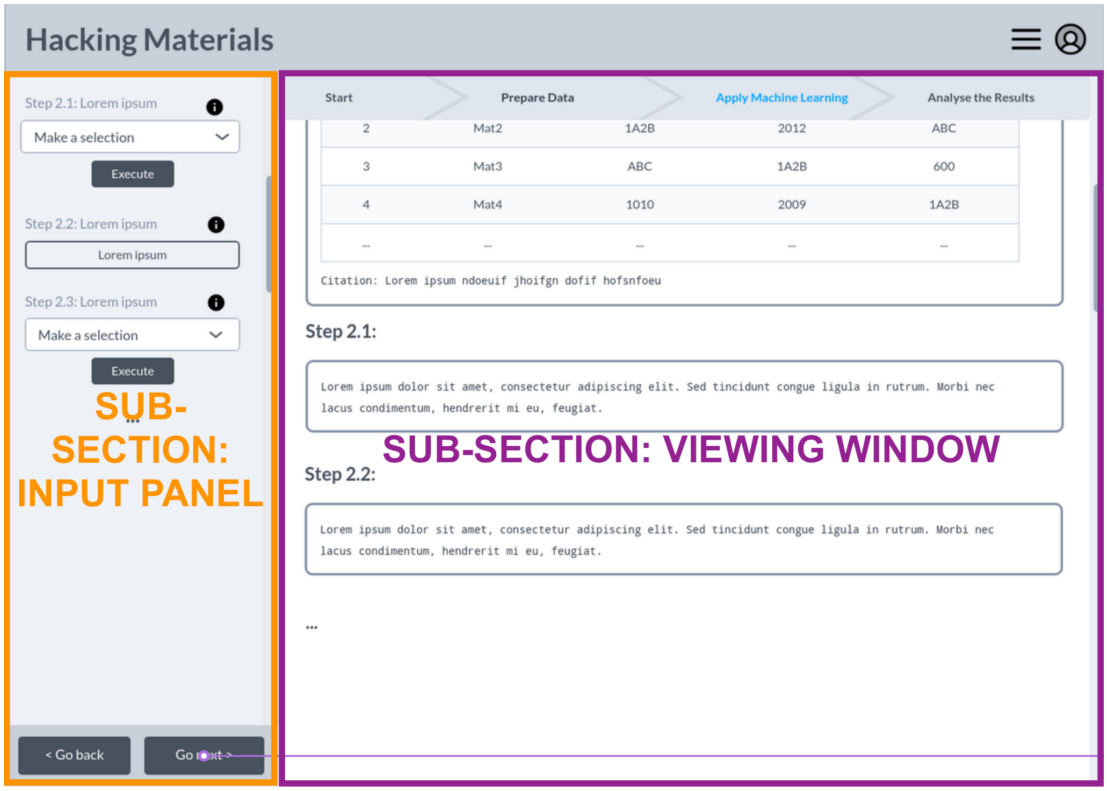
The main ideas of this are as follows:

Sections

By referring to the low-fidelity prototype created earlier in the project, we divide the main application page into 2 main sections:

- Header: the top bar, which does not need to have context of what stage the user is up to and what's happening at any given point.
- Body: Includes 2 subsections that both need to know which stage the user is at (e.g. "Pre-process data" or "Apply machine learning"):
  - Left-side panel: named in the structure as InputPanel. Example code for this panel and how it shows the workflow steps is included in the sample code section below.
  - Main window on the right: named in the structure as `ViewingWindow`

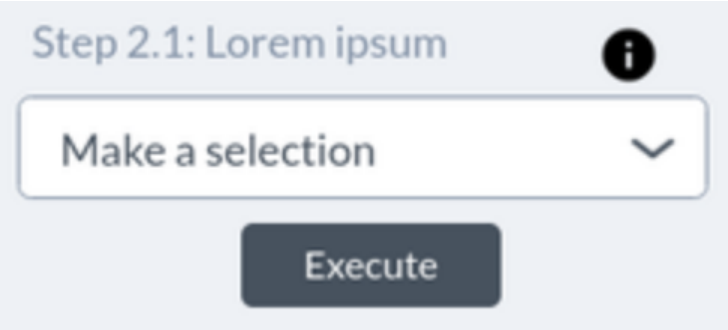




Components

This folder will mainly contain all reusable components, e.g. Button, Tooltip, Modal, etc.

Notably, some of these reusable components would be "step types", e.g. DropdownSelectStepType. This example step type refers to the entire object shown below, including a step number, title, tooltip, dropdown list, button and whatever else may be needed. We would create this as a reusable component because many steps have similar requirements, e.g. selecting a dataset and selecting a featurizer should both be dropdown list type steps.



Steps

The word "steps" in this section refers specifically to the workflow steps that would be shown in the input panel, e.g. Dataset Selection step, Featurizer Selection step, etc.

A separate folder is created for these so that there would be a clear pattern that is easy to follow whenever more steps need to be added. Each step would use a step type component that is imported from the /components folder. E.g. the DatasetSelectionStep would use the DropdownSelectStepType, as shown in the sample code snippet below.

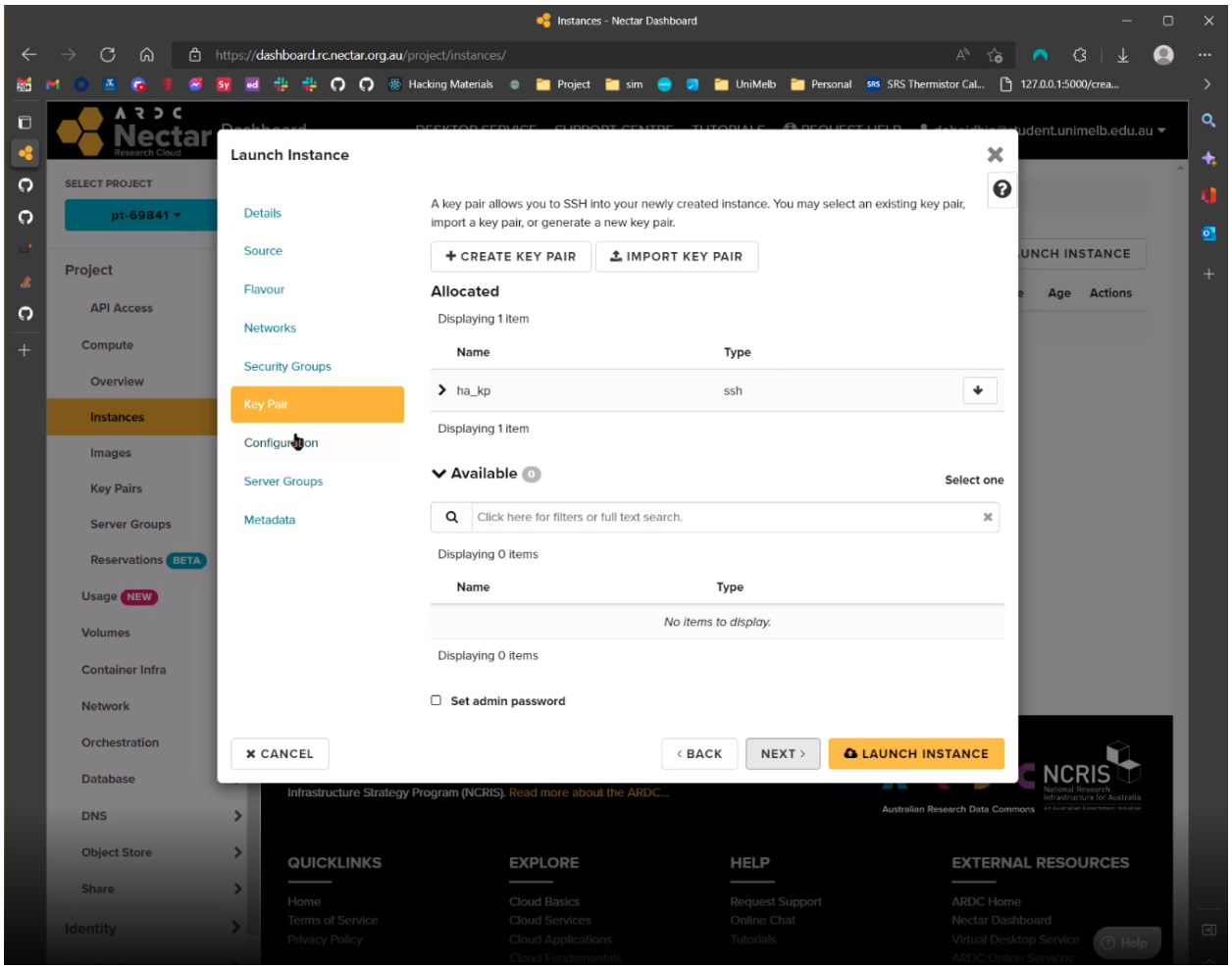
Sample code snippets

```
import DropdownSelectStepType from '../../components/dropdownSelectStepType';
import HelpModal from './HelpModal';
...
const DatasetSelectionStep = (props) => {
	...
	const STEP_KEY = "dataset_selection"

	const options = api_call_here() // calls backend API to get the dataset options

	const onSubmit = selected_value => send_to_backend() // send to backend using api

	return (
		<DropdownSelectStepType
			stepNumber={props.stepNumber}
			title="Select Dataset"
			description="bla bla"
			tooltipContent={HelpModal}
			options={options}
			onSubmit={onSubmit}
		/>
	);
};
```

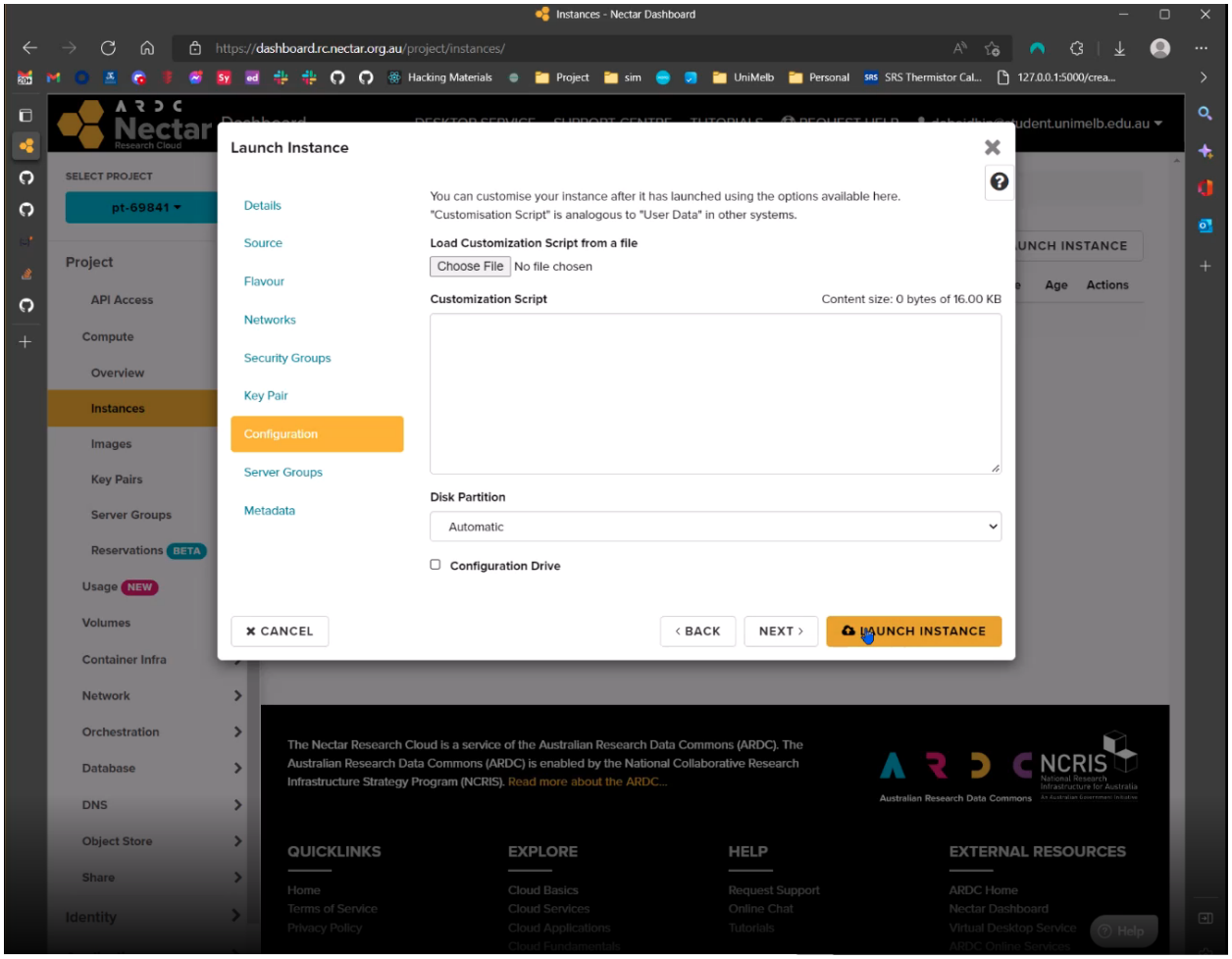
AMENDMENT - 2nd September 2022

It was suggested that an appFooter is potentially added as a section as well. No objections to this so far.

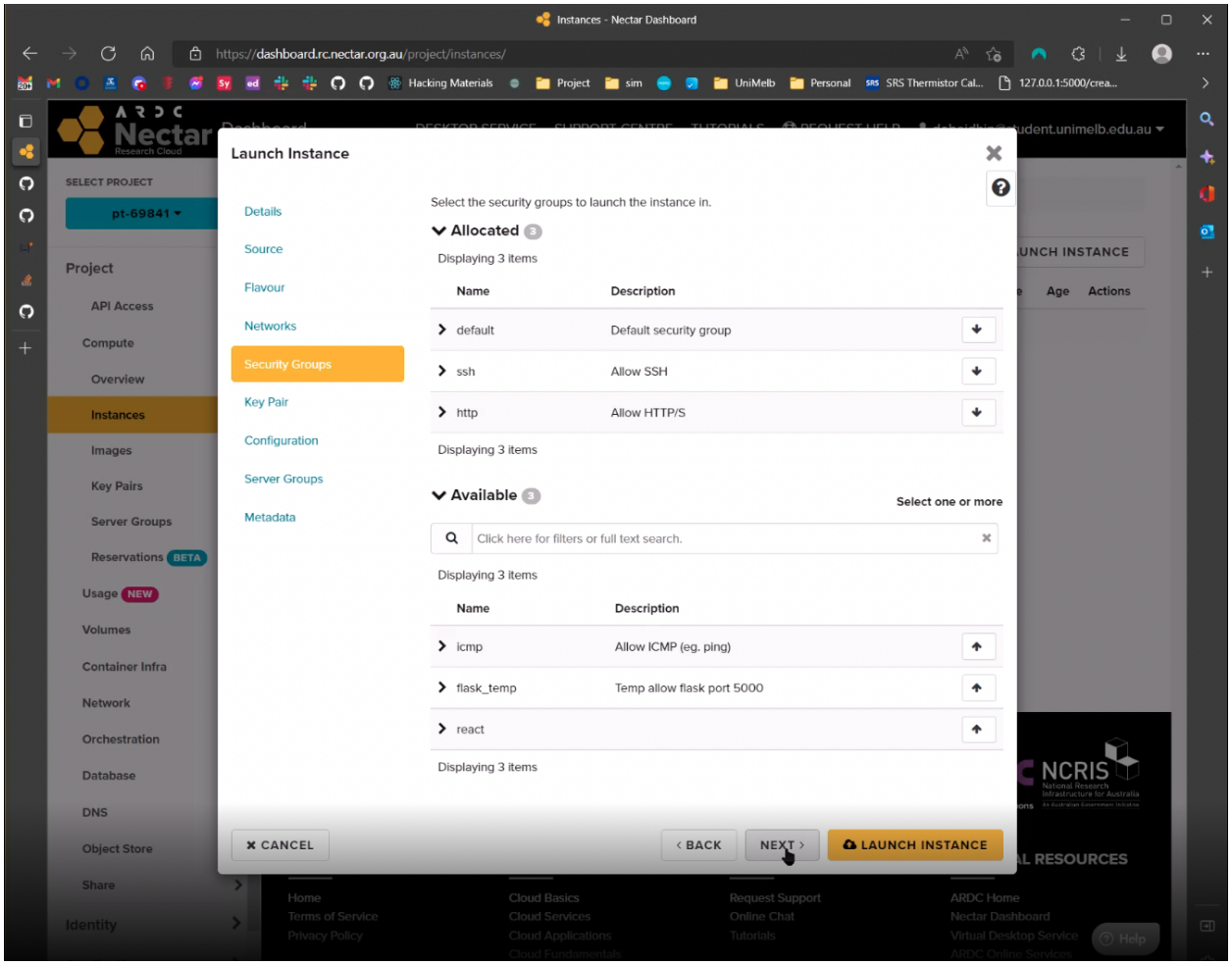
Appendix 9. Deployment process screenshots



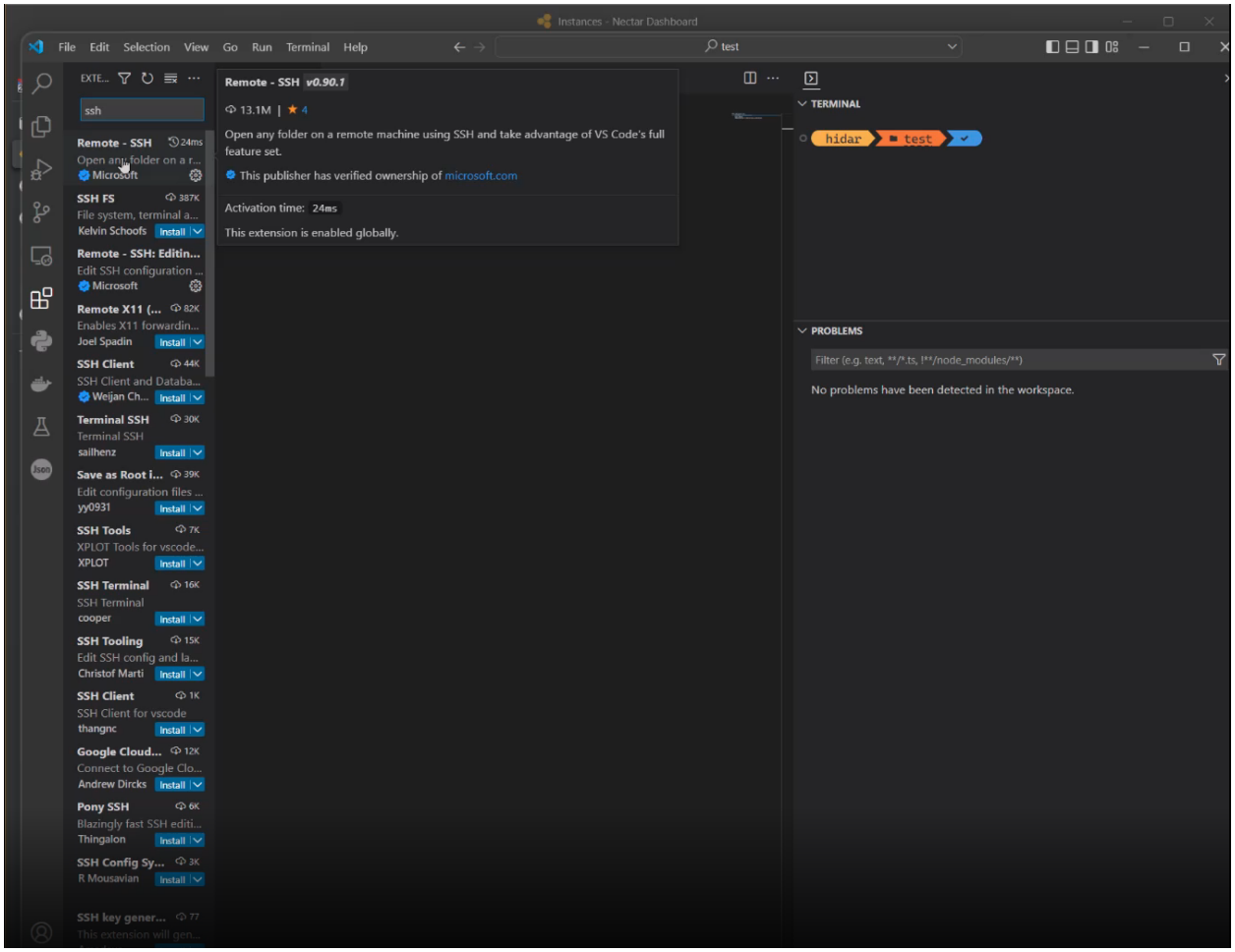
Create a key pair to use when accessing the virtual machine



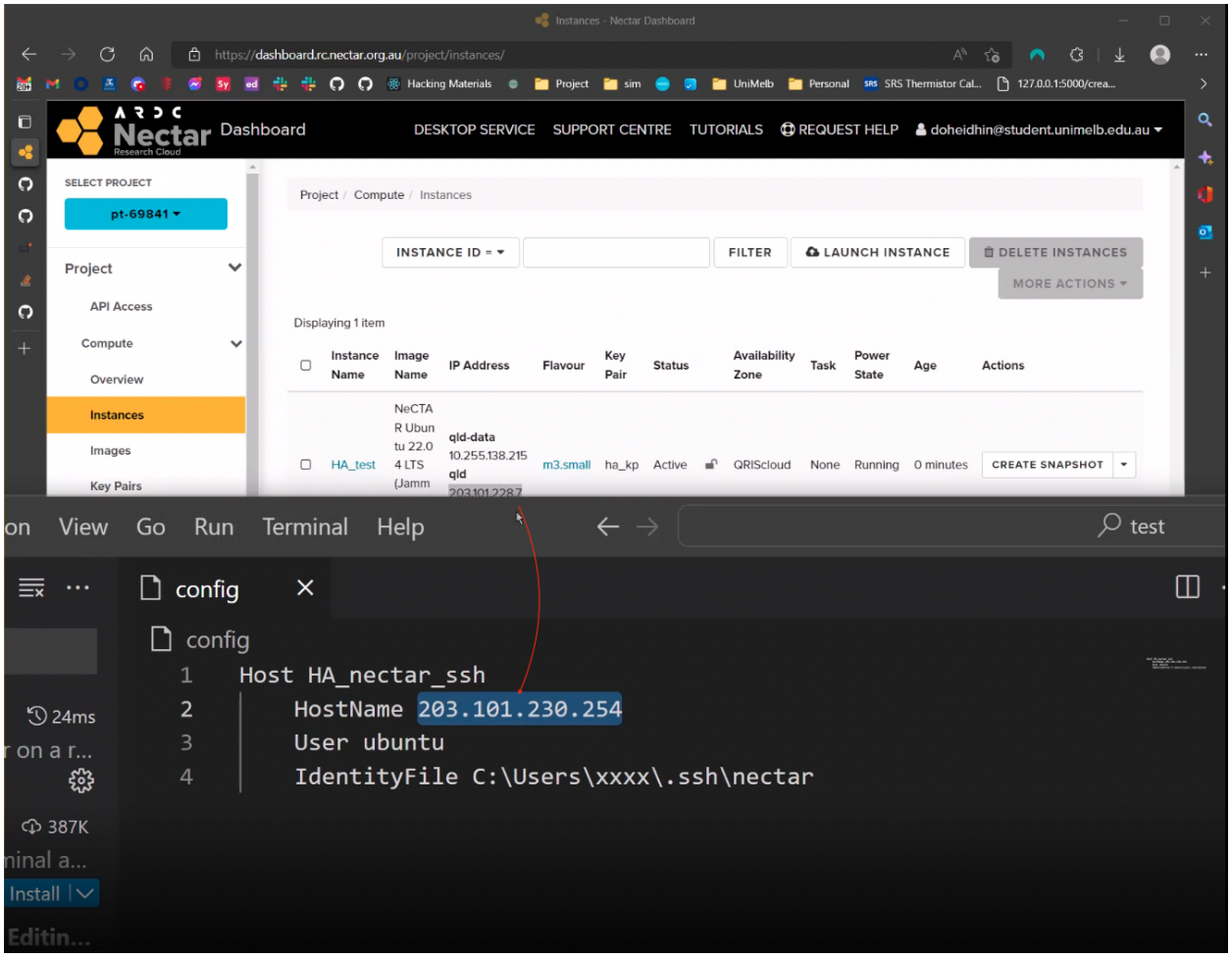
Launch the virtual machine



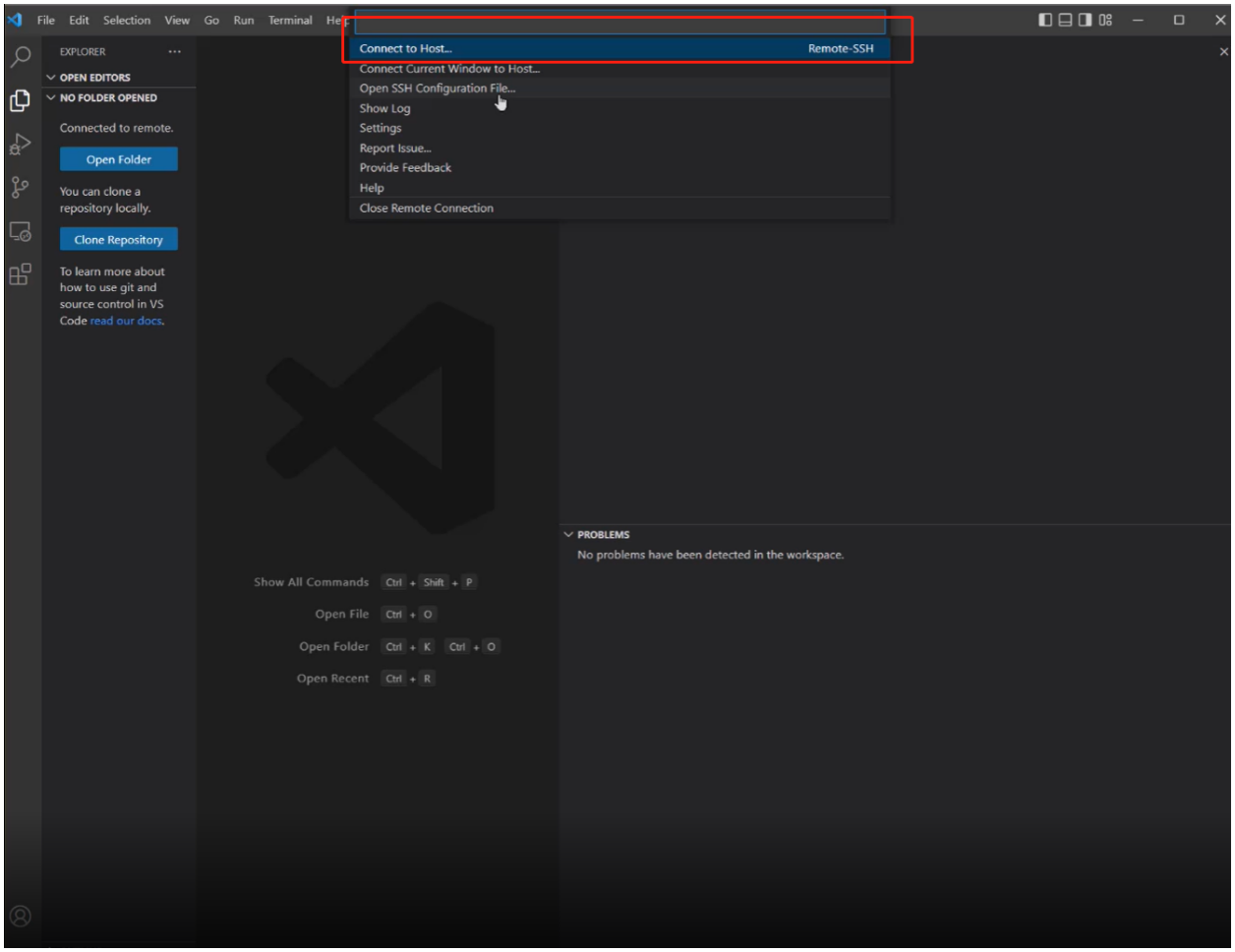
Create a security group in Nectar to enable HTTPS and SSH access



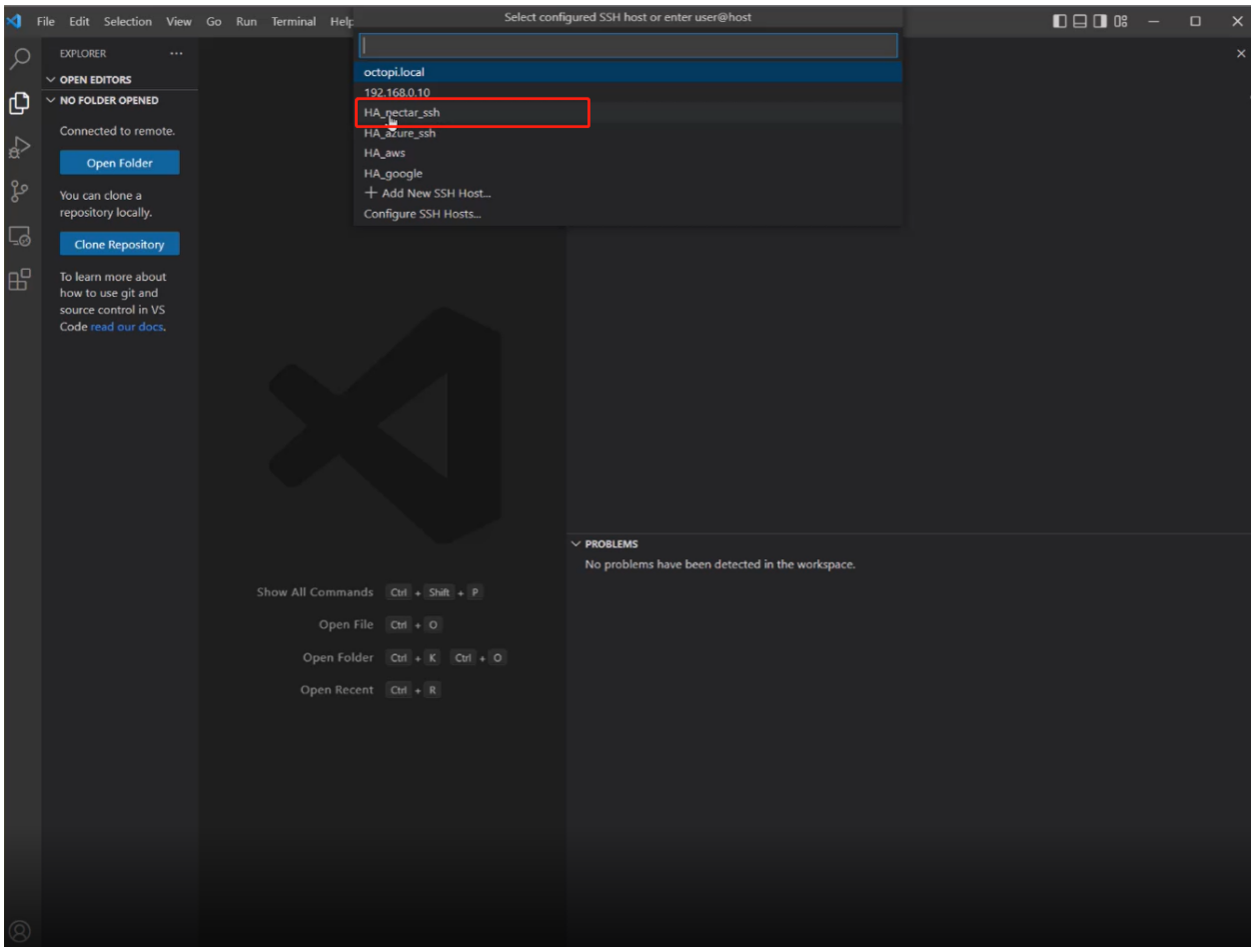
Connect to the machine using SSH. In this example, we have used a Visual Studio Code remote connection



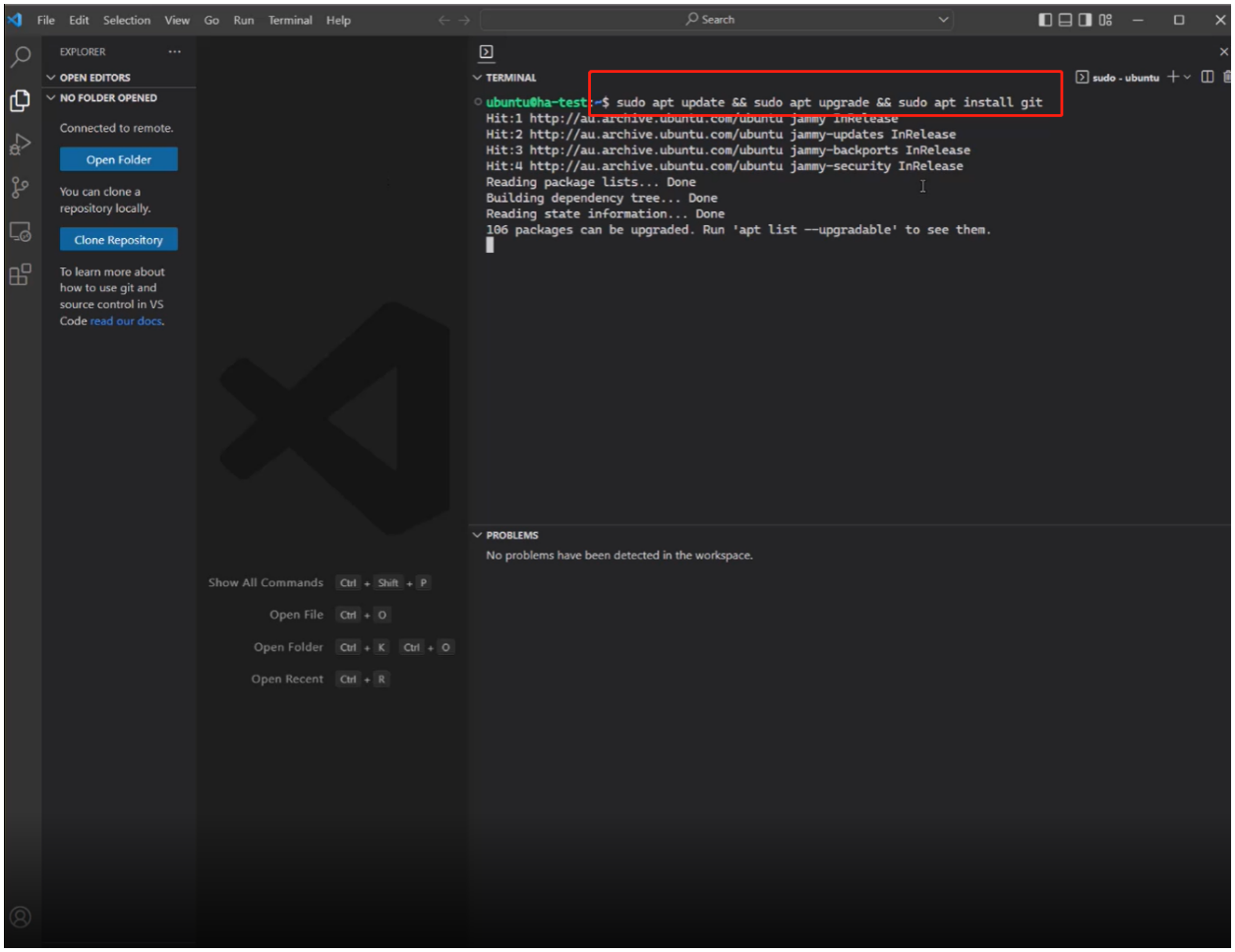
Set up an SSH configuration file entry for the virtual machine

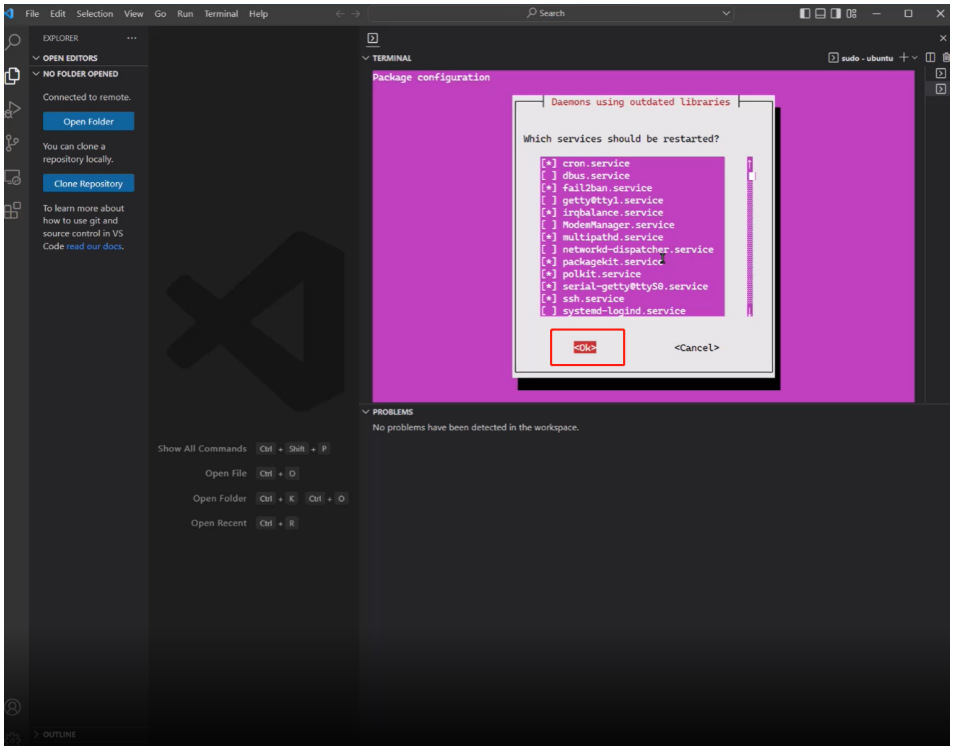


Connect to the virtual machine using SSH

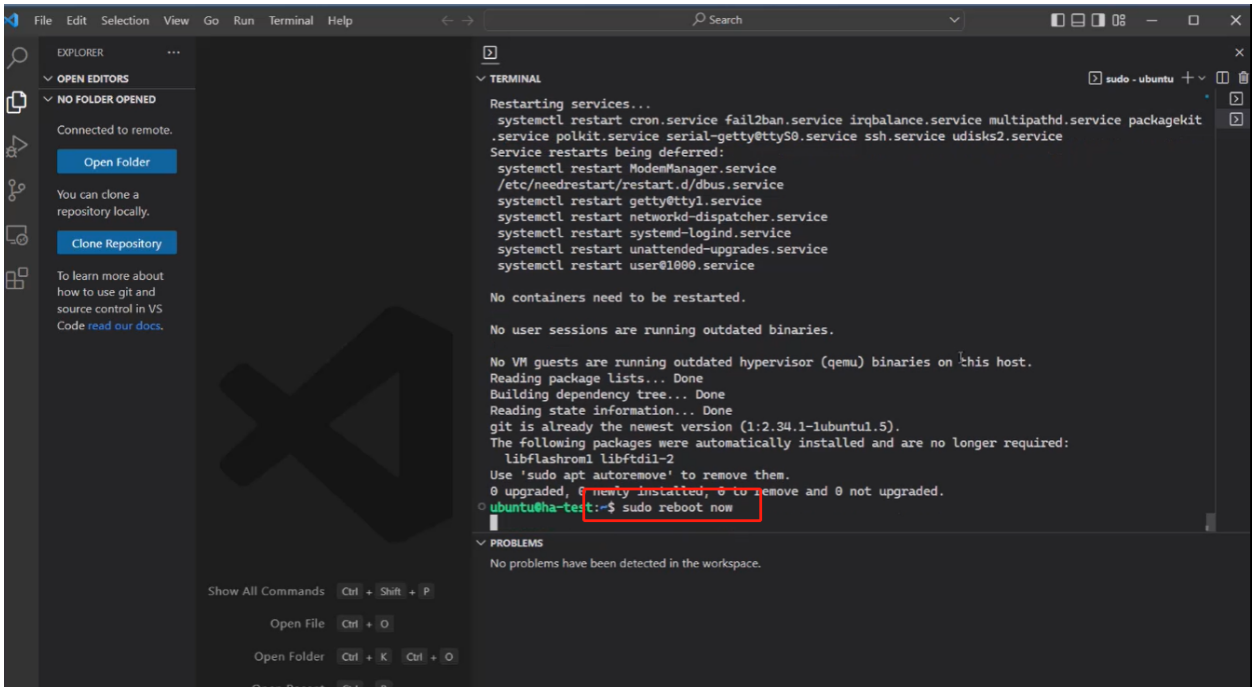


Choose the right machine to connect to

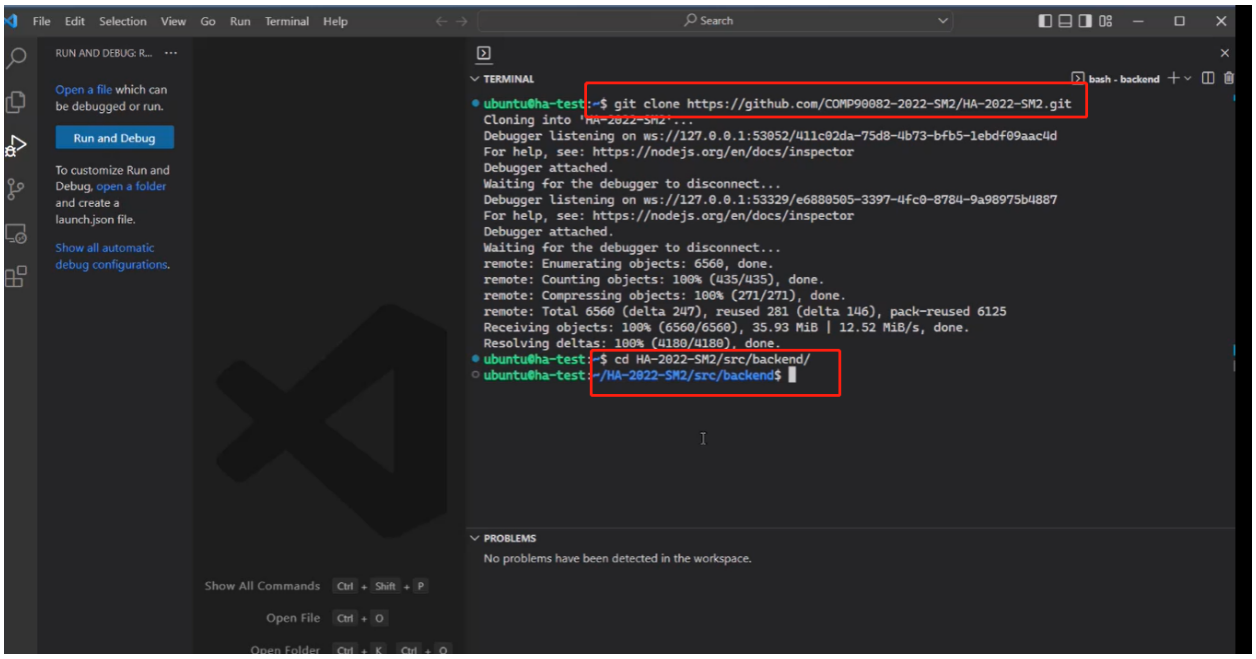

File Edit Selection View Go Run Terminal Help
Search
EXPLORER
OPEN EDITORS
NO FOLDER OPENED
Connected to remote.
Open Folder
You can clone a repository locally.
Clone Repository
To learn more about how to use git and source control in VS Code read our docs.
TERMINAL
sudo - ubuntu
ubuntu@ha-test:~$ sudo apt update && sudo apt upgrade && sudo apt install git
Hit:1 http://au.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://au.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://au.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://au.archive.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
106 packages can be upgraded. Run 'apt list --upgradable' to see them.
PROBLEMS
No problems have been detected in the workspace.
Show All Commands Ctrl + Shift + P
Open File Ctrl + O
Open Folder Ctrl + K Ctrl + O
Open Recent Ctrl + R



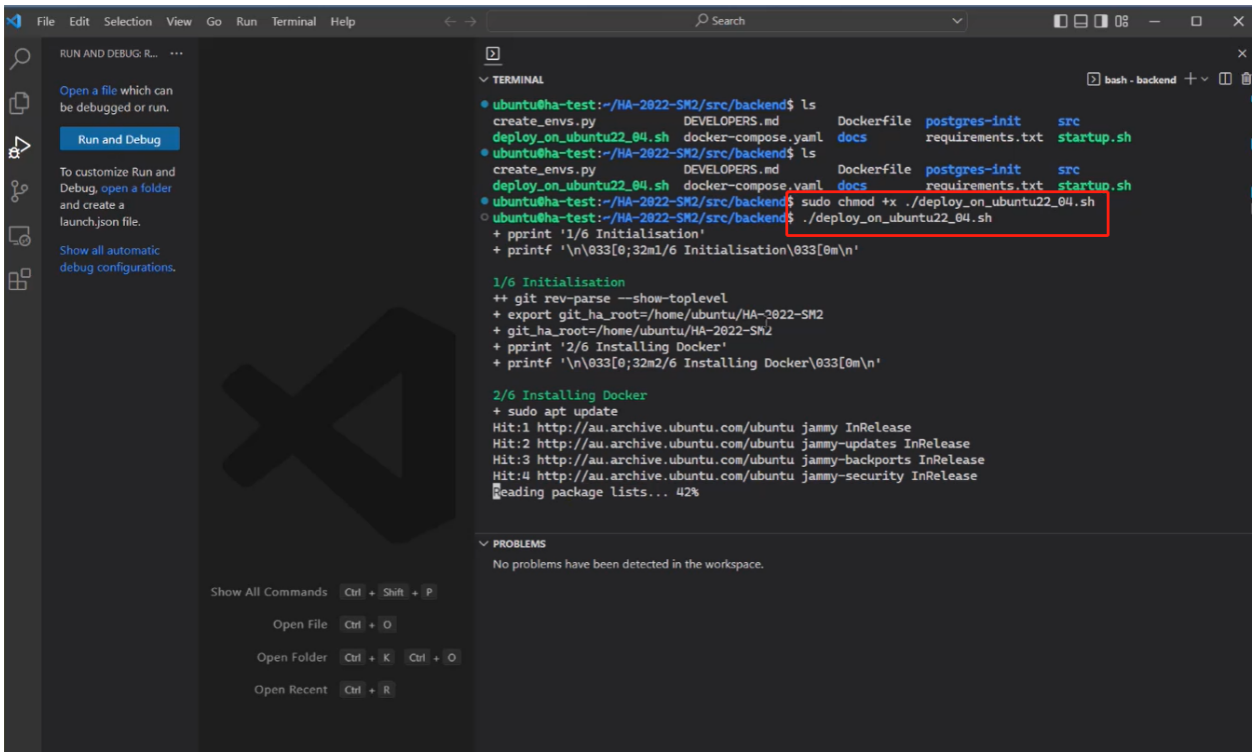
Update the existing software on the machine



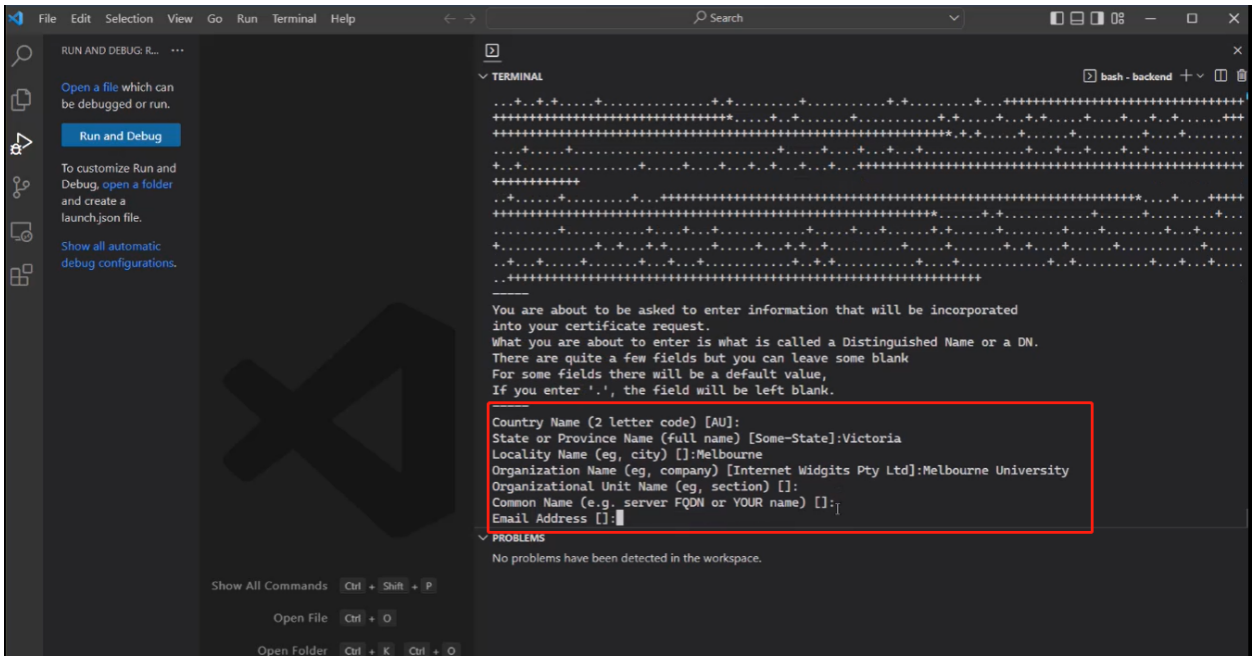
Reboot the machine



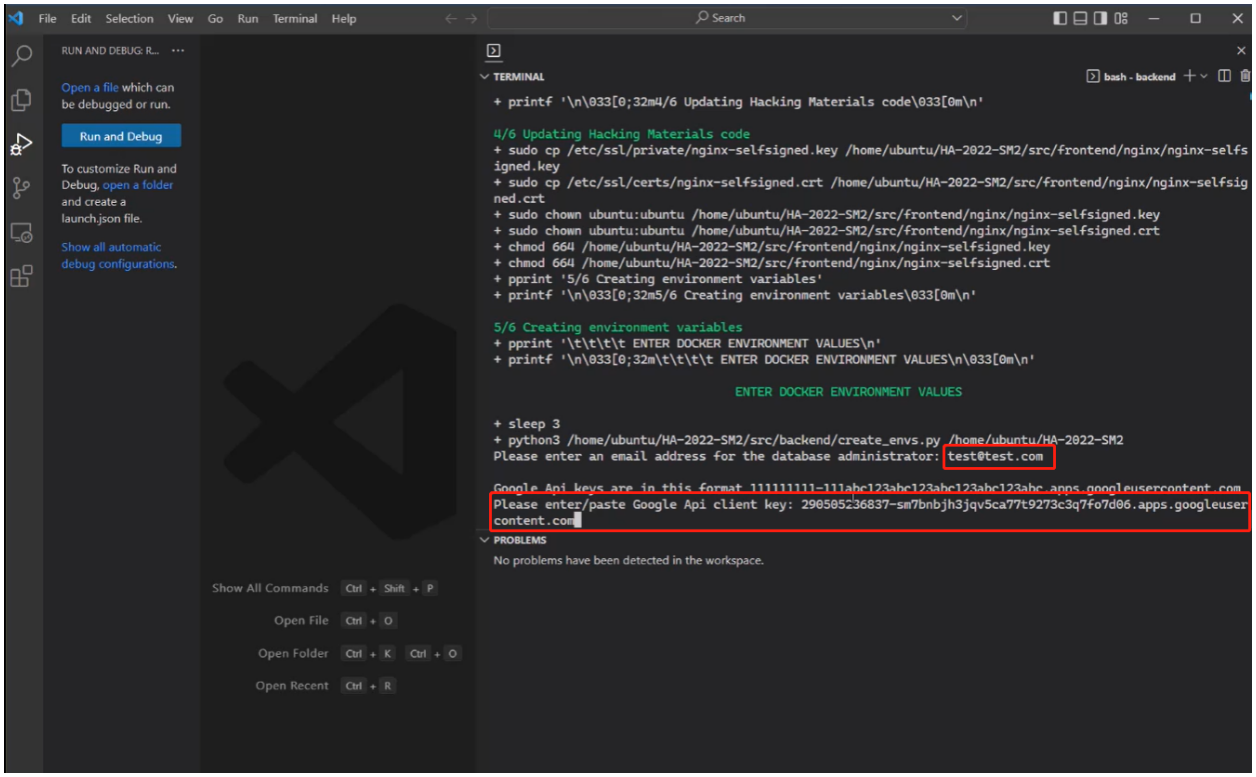
Clone the GitHub repository



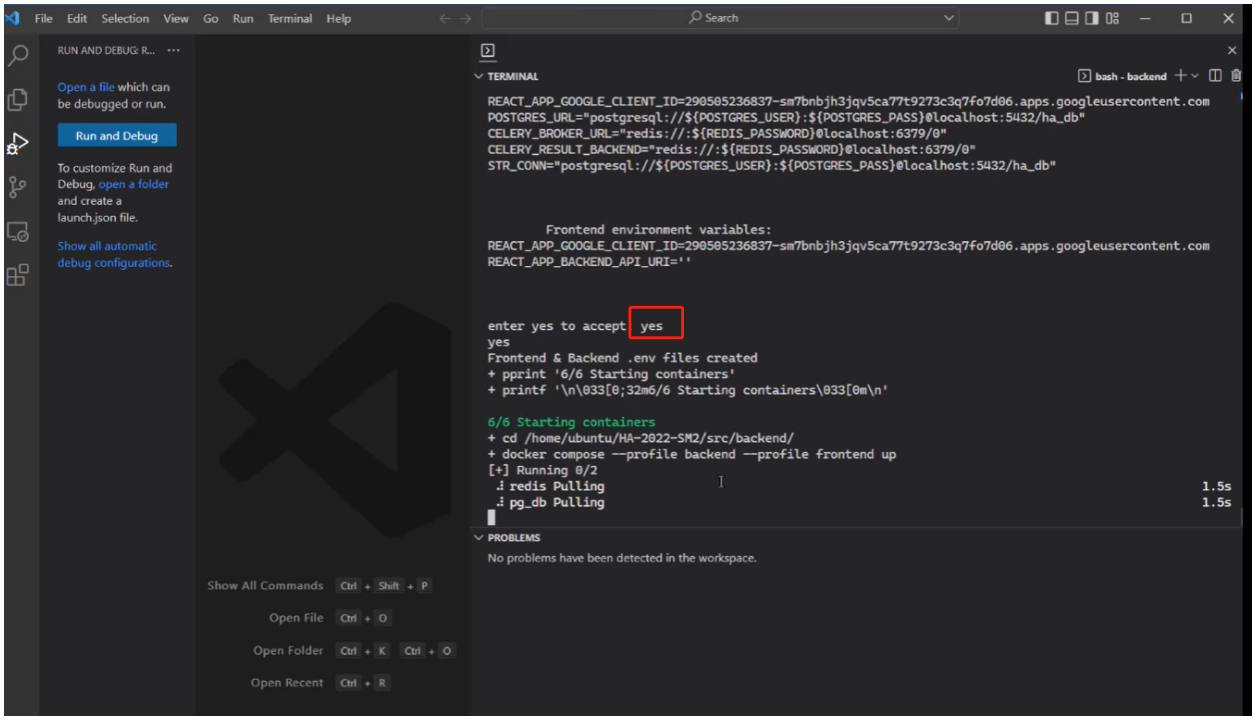
Run the deployment script



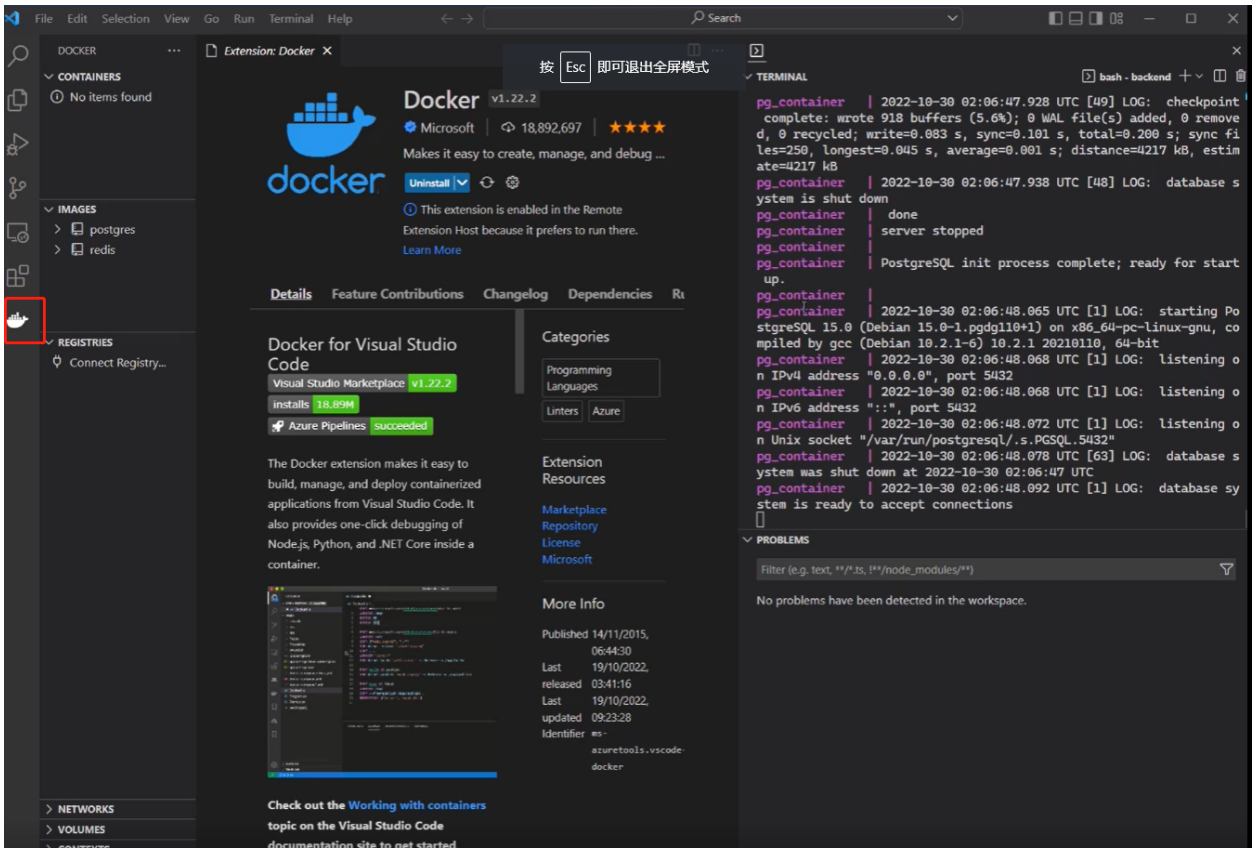
Enter Self-Signed Certificate data



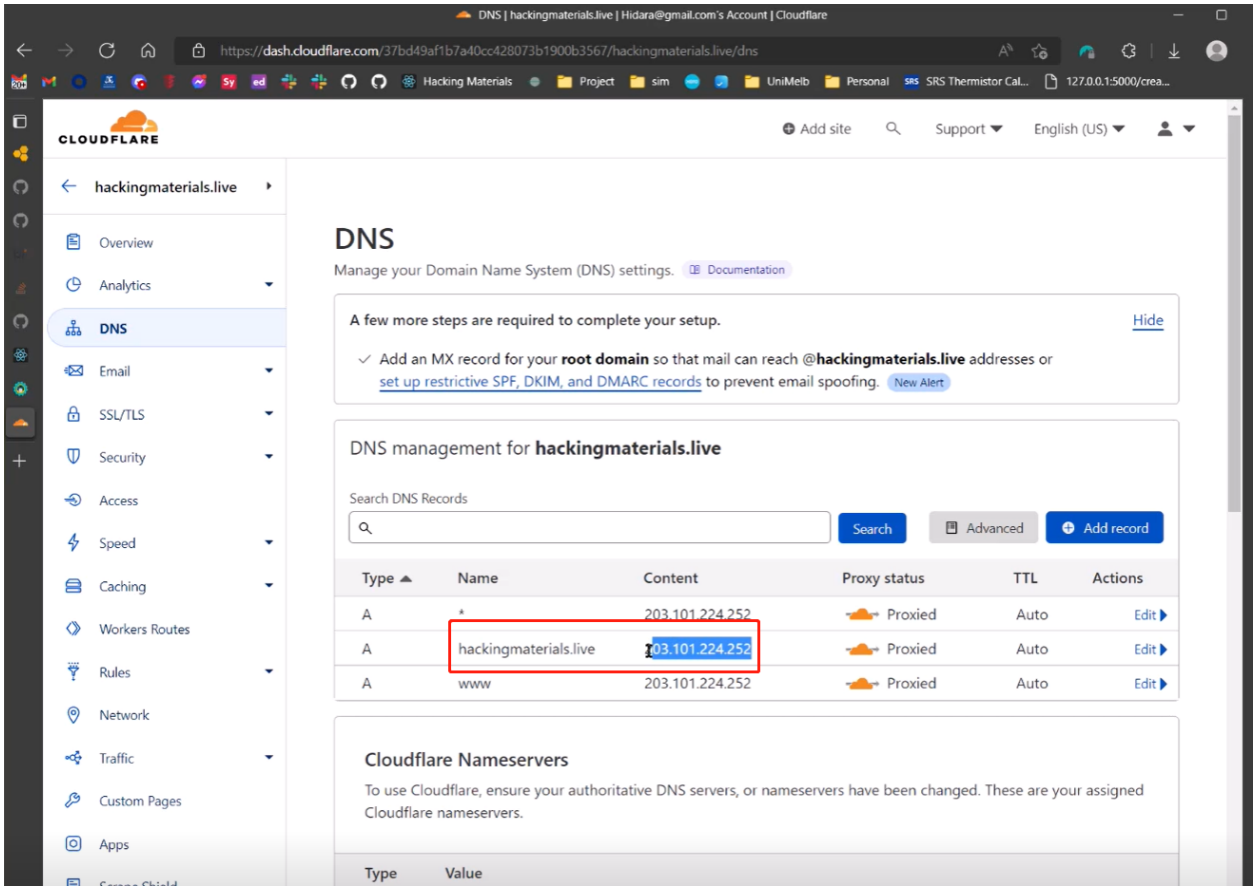
Enter API keys for Google and Microsoft Single-Sign On APIs



Accept the generated environment variables



The application containers are launched



Add a DNS entry for the virtual machine in Cloudflare