



THE UNIVERSITY OF
WESTERN
AUSTRALIA

CITS3200 Project Team 18

**UWA Academic Skills Drop-in - Client queue flow and usage reporting
software**

Alex Mai, Jordan Hartley, Frinze Lapuz, Jake Yendell, Alex Hoffman, Liam Hovell

Copyright Alex Mai, Jordan Hartley, Frinze Lapuz, Jake Yendell, Alex Hoffman, Liam Hovell

Table of contents

1. Project Description	3
1.1 Authors	3
2. User	4
2.1 User Documentation	4
2.2 Queue	5
2.3 Data analytics	9
2.4 Export data	10
3. Developer	11
3.1 Technical Documentation for Developer	11
3.2 Requirements	13
3.3 Coding Patterns	16
3.4 Frontend and Backend Technical Documentation	17
3.5 Automated Testing	19
3.6 Continuous Integration Pipeline	22
3.7 Deployment	23

1. Project Description

ResQ is an online live queue tool for use by the staff of the UWA STUDYSmarter program. The purpose of it is to provide a robust and convenient way of managing a queue, as well as saving the data for each entry into the queue to allow for some simple data analytics.

1.1 Authors

- Jordan Hartley
- Liam Hovell
- Frinze Lapuz
- Alex Hoffman
- Alex Mai
- Jake Yendell

2. User

2.1 User Documentation

2.1.1 How to access the website

Click [here](#) to redirect to the website. User will need username and password to login.

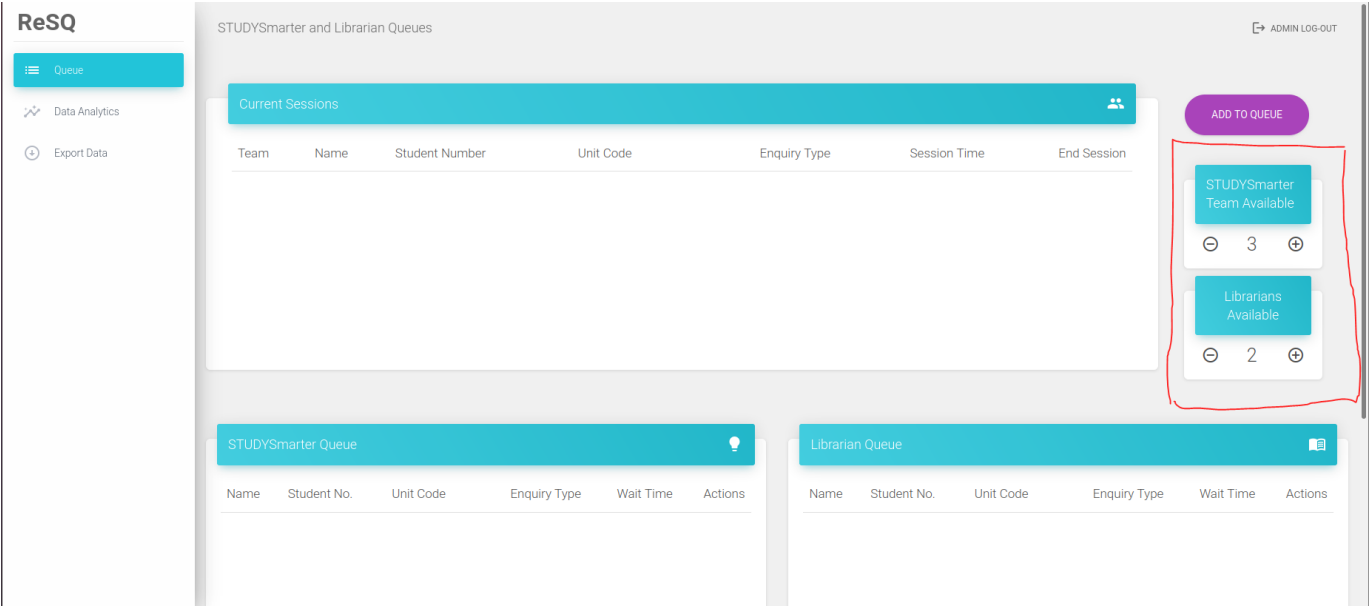
There will be 3 main tabs in the website, which are Queue, Data Analytics and Export Data

2.2 Queue

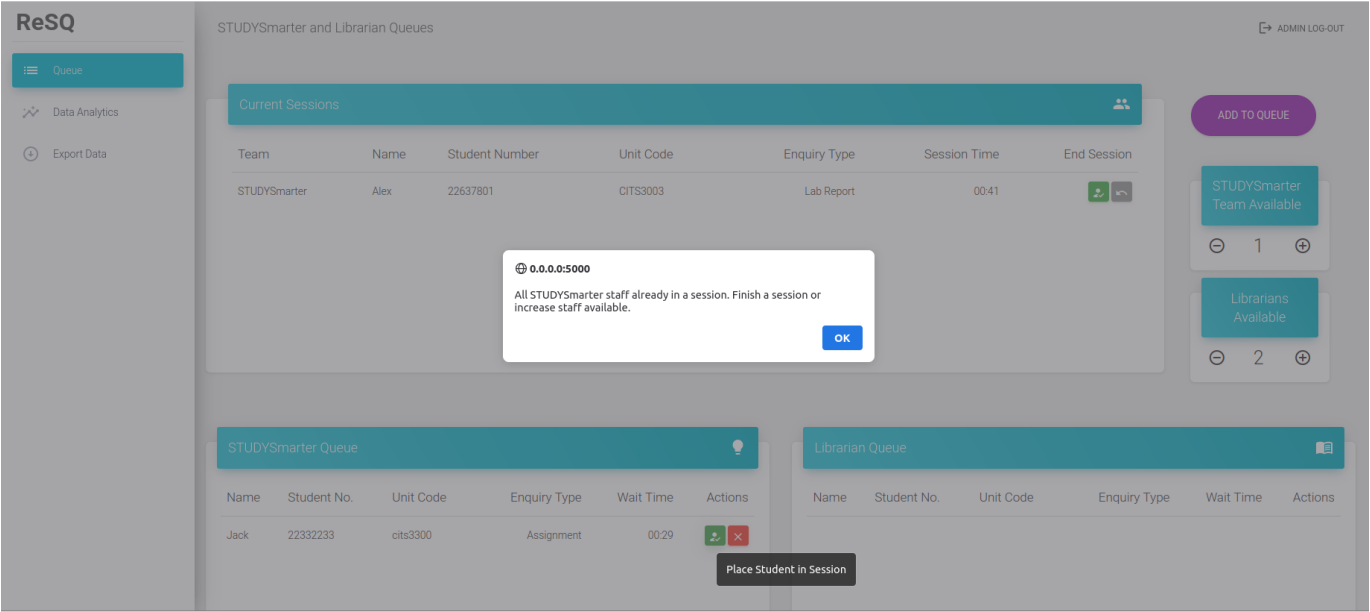
User can manage student sessions in the Queue tab. There are 3 tables, 2 of which are STUDYSmarter queue and Librarian queue. Students will be put into either queue depending on user's choice.

2.2.1 Staff management

On the right side of the queue home page, you can see 2 small boxes saying "STUDYSmarter Team Available" and "Librarians Available". You can click the "+" and "-" icon to modify the number base on your current available staff.

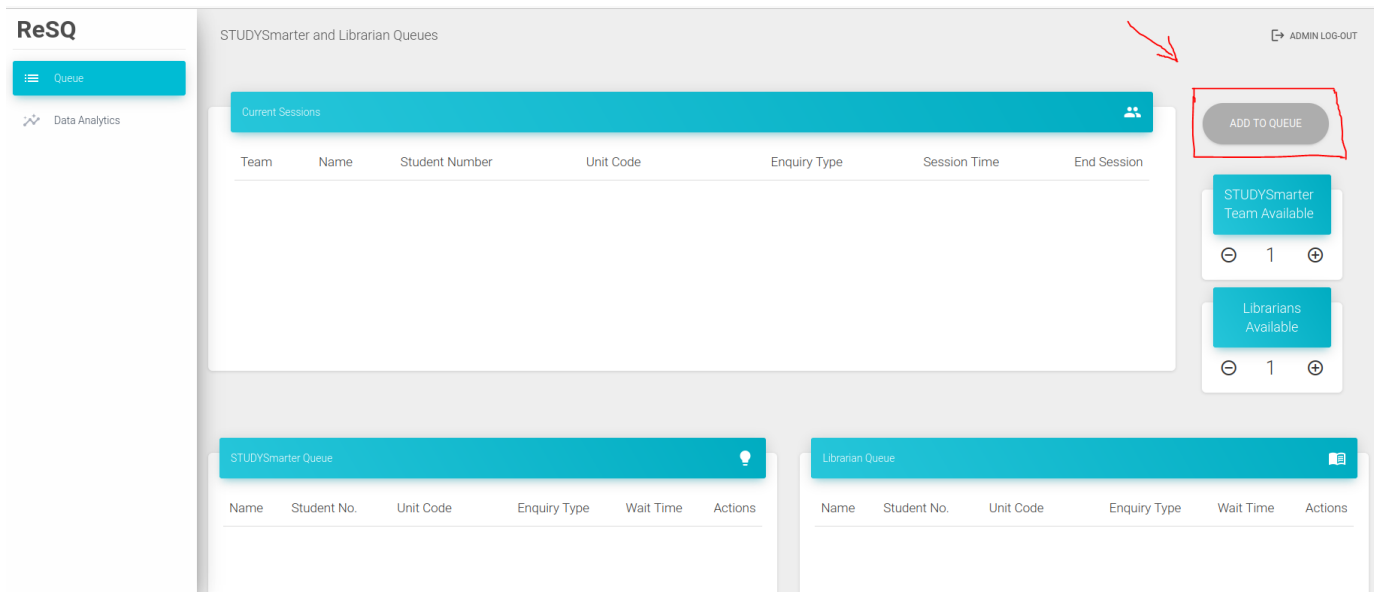


If you try to add student to the inSession queue which exceeds your current setup for each staff type, the website will give a warning alert and require you to modify the staff number.

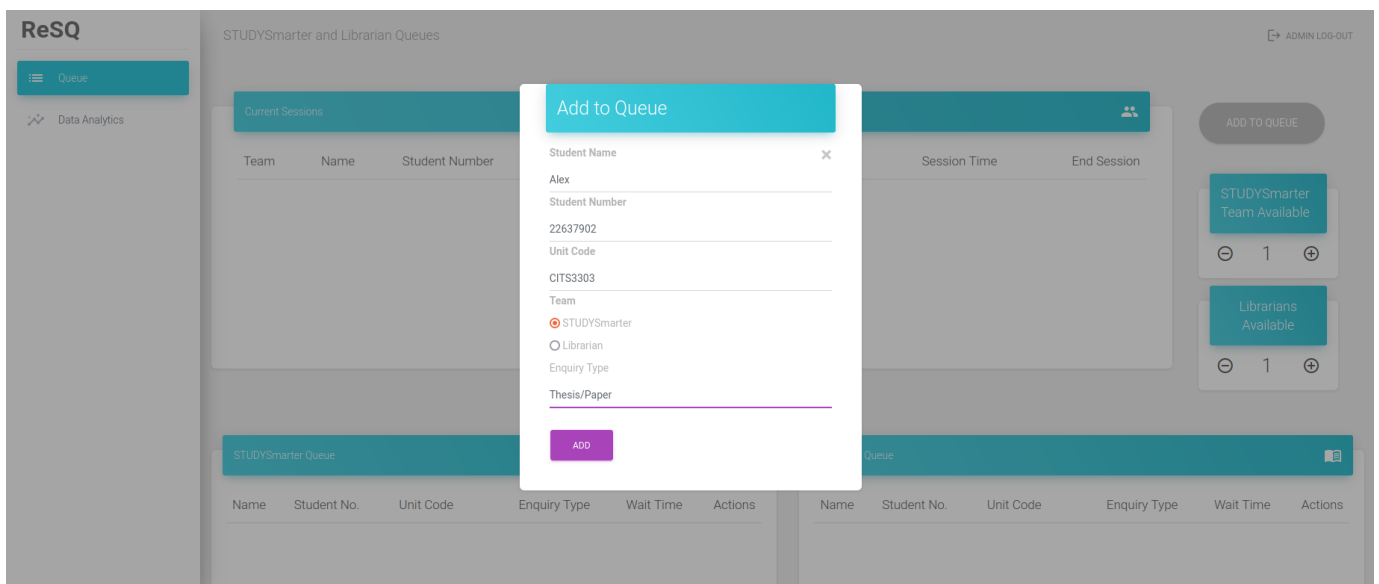


2.2.2 Add to queue

In order to add a student to a waiting queue, look for the "ADD TO QUEUE" button in the top right corner



A form will show up as below. In this form, you can enter student information and choose the queue type between "STUDYSmarter" and "Librarian"



When successfully added, the student information will show up in the corresponding queue. Below is an example

STUDYSmarter and Librarian Queues ADMIN LOG-OUT

Current Sessions

Team	Name	Student Number	Unit Code	Enquiry Type	Session Time	End Session
------	------	----------------	-----------	--------------	--------------	-------------

ADD TO QUEUE



STUDYSmarter Team Available

⊖ 1 ⊕

Librarians Available

⊖ 1 ⊕

STUDYSmarter Queue

Name	Student No.	Unit Code	Enquiry Type	Wait Time	Actions
Alex	22637902	CITS3303	Thesis/Paper	00:09	 

Librarian Queue

Name	Student No.	Unit Code	Enquiry Type	Wait Time	Actions
------	-------------	-----------	--------------	-----------	---------

2.2.3 Managing a session

If you want to move students from either waiting queue to "Current Session" table, click the green button in the "Actions" column

STUDYSmarter and Librarian Queues ADMIN LOG-OUT

Current Sessions

Team	Name	Student Number	Unit Code	Enquiry Type	Session Time	End Session
------	------	----------------	-----------	--------------	--------------	-------------

ADD TO QUEUE



STUDYSmarter Team Available

⊖ 1 ⊕

Librarians Available

⊖ 1 ⊕

STUDYSmarter Queue

Name	Student No.	Unit Code	Enquiry Type	Wait Time	Actions
Alex	22537809	CITS2200	Assignment	00:05	 

Librarian Queue

Name	Student No.	Unit Code	Enquiry Type	Wait Time	Actions
------	-------------	-----------	--------------	-----------	---------

[Show all](#) >



If a student cancel the appointment, you can remove her/him from the waiting queue by clicking the red button in the "Actions" column

A message box will show up and ask to confirm your action, click "Yes".

2.2.4 Finish/Undo a session

When a session is done, user can click the green button to finish the session.

When user accidentally added a student to this table from waiting queue, undo it by choosing the undo button

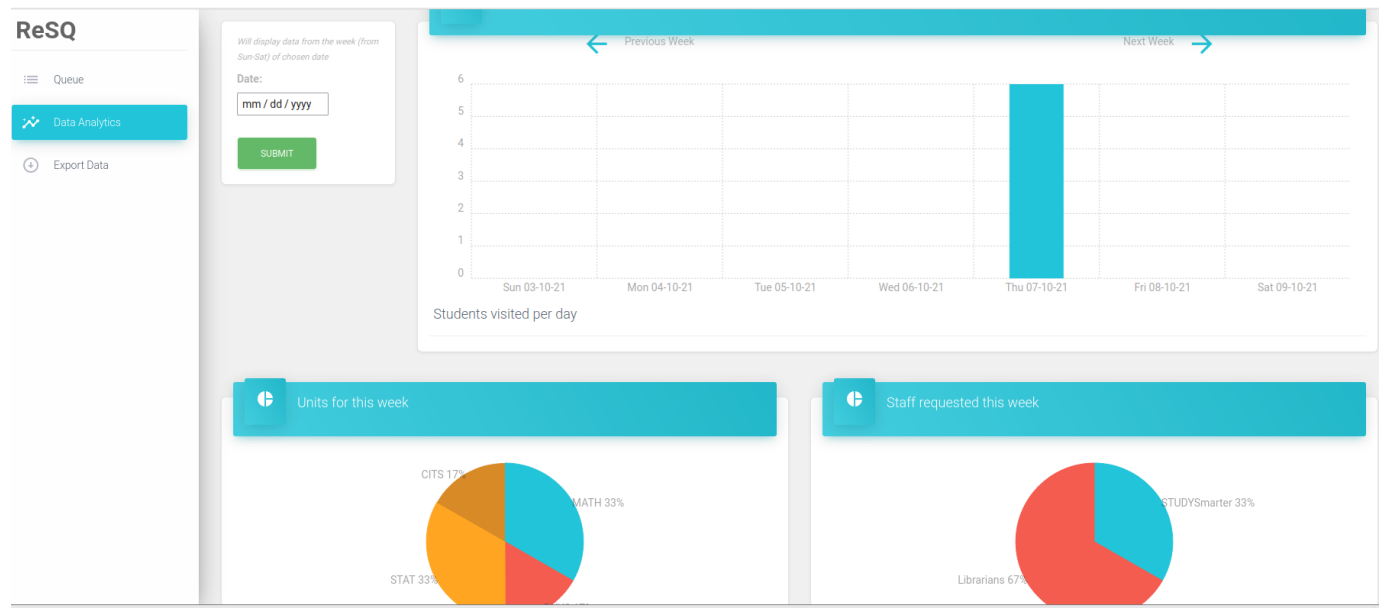
Current Sessions							
Team	Name	Student Number	Unit Code	Enquiry Type	Session Time	End Session	
STUDYSmarter	Alex	22537809	CITS2200	Assignment	00:30	 	

2.3 Data analytics

This website provide some basic data analytics for "at a glance" view.

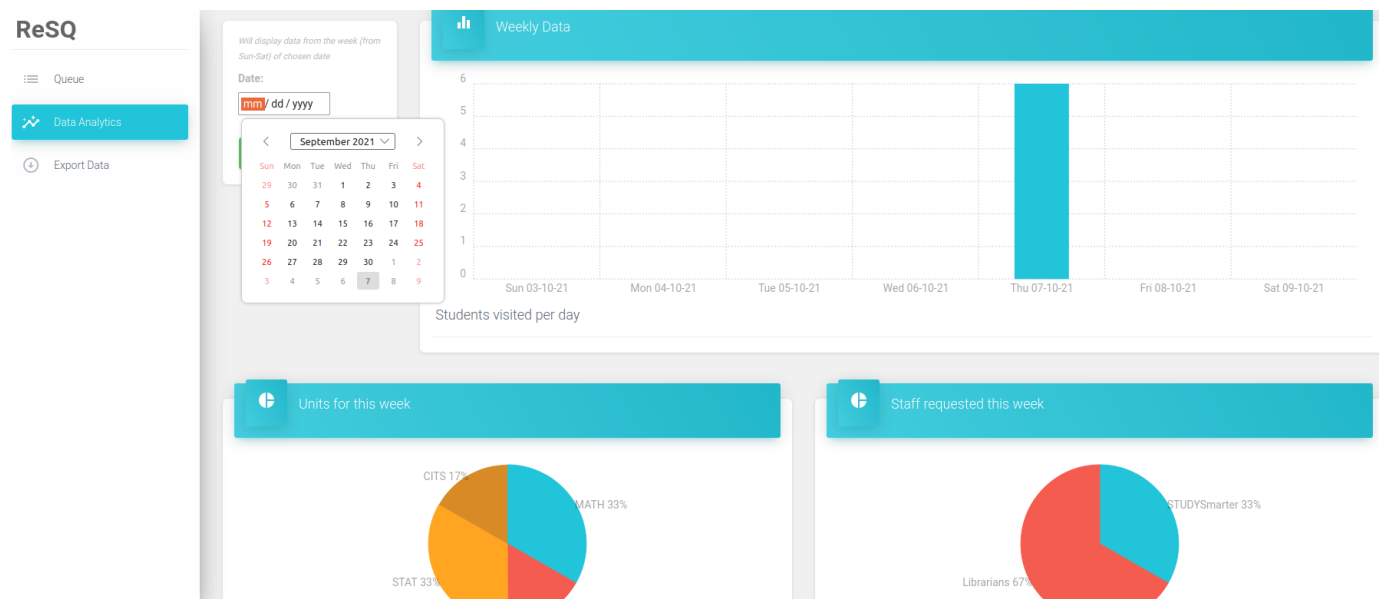
There will be 3 charts generated in the Data analytics tab:

- Students visited per day in the chosen week
- The percentages of popular units in that week
- Comparison between students enter STUDYSmarter and Librarian queue in that week



In order to generate data analytics for a chosen week:

- Choose an arbitrary day in the week you want to generate data



- Click submit

2.4 Export data

If user wants to export data to a csv file for further data analysis:

- Step 1: Navigate to the 'Export Data' tab
- Step 2: There are premade options such as "Last week" or "Last Day". You can also choose "Custom" option to export data between a period of your choice

Overview - CITS3200 Pro x ReSQ x +

0.0.0.0:5000/export

ReSQ

Queue

Data Analytics

Export Data

Export Data

ADMIN LOG-OUT

Export to CSV

☐ Last Day

☐ Last Week

☐ Last Month

☐ Last Year

☐ All Time

☒ Custom

Start Date

End Date

SUBMIT

Realtime STUDYSmarter Queue

- Step 3: Click 'Submit', then download the file

3. Developer

3.1 Technical Documentation for Developer

3.1.1 Application

The website is run using a flask server. Flask is a micro framework for the backend of the website. Jinja is used inside the HTML so the display can adapt to server data as well as for running loops. Users and test attempts are saved inside a SQLite database. The username, password, and scores of the user are saved so progress can be encouraged.

3.1.2 Development Workflow

This project uses docker to orchestrate multiple services. Make sure to install docker, see [here](#) for documentation.

Once you have it installed do the following:

Environment Variables: Create the `.env` file

There is a file called `template.env`. This contains all the configurations for the entire application for database, flask app, and pgadmin. Copy this to `.env` (you have to create this file).

Run the Docker-Compose

Run the following command

```
1 docker-compose up
```

This one command will build all the containers. Most notably this will create the Flask Application with pip installation, and database migration to PostgreSQL.

building containers

If you do need to build containers, run the following:

```
1 docker-compose up --build
```

Services that are running

There are a couple of services that are running

Services	URL
PostgreSQL Database	http://localhost:5432
PgAdmin (PostgreSQL GUI)	http://localhost:8002
ReSQ Flask App	http://localhost:5000
MkDocs Documentation	http://localhost:8001

Going inside the container / Remote Code Execution

Most likely you will be developing inside this container such as doing pip installation and other commands. You can do remote code execution to the container using the following command

```
1 docker exec -it resq_app bash
```

This will allow you to connect to the container and do whatever command that pleases you.

3.2 Requirements

3.2.1 Acronyms, Abbreviations and Definitions

- UWA: The University of Western Australia
- Tutor – The person that is providing service/help on a particular topic (eg. References, math and statistics, English)
- Student leader (SL) – the main point of contact of the student in need (SNS)
- SNS – Student in Need of Service – the person that requires help from a tutor
- STUDYSmarter – a service in UWA that is staffed with tutors and SNS come for help
- API – Application programming interface – an application that handles data requests for server-side processes
- Database – a place where data is stored
- SQL – Structured Query Language – used for building and interacting with databases
- HTML, CSS, JS – core web technologies that are used to build web interfaces
- Flask – a Python (an easy-to-use programming language) web framework that makes it easy to build APIs
- VPS – Virtual Private Server is a virtual machine that runs as a server within a bigger set of services

3.2.2 Aim and Scope

UWAs STUDYSmarter and Library teams run a daily drop-in service in the Reid Library, 10am-12 noon, where students can drop in to receive one-on-one advice on writing, research, referencing, and study skills. The service is staffed each day by two STUDYSmarter learning advisers (writing and study skills enquiries), one librarian (research and referencing) and one student leader (reception and queue management). The student leader acts as the first point of contact for the service, triaging enquiries, inputting student data, and managing the queue flow. During peak periods, demand for the service is high, and long queues can form, creating a bottleneck for the service as the student leader must manage both new and waiting clients.

The ideal system would be a software program to streamline the data input/queue flow process to allow the service to operate more efficiently. Additionally, it would generate usage reports and infographics to allow STUDYSmarter to anticipate trends in usage and ensure that the service is staffed appropriately to meet demand.

3.2.3 Requirements

Stage 1 Functional Requirements

The following are the core functional requirements for the first stage application

Identifier	Name	Description
S1_FR1	Login System	The system should only allow SL to use the system.
S1_FR2	Adding student to the queue and showing the queue	The system should allow SL to add students in the queue as well as show who is currently in the queue.
S1_FR3	Consuming the queue	The SL should be able to assign SNS from the queue to a tutor and show the current SNS in session with a tutor.
S1_FR4	Time display data	For S1_FR2 and S1_FR3, the system should show how long an SNS is waiting or in session with a tutor.
S1_FR5	Remove an SNS in the queue	The system should allow removal of SNS in queue as per S1_FR3 or the drop-in session run out of time, and SNS needs to be removed from the queue.

Stage 2 Functional Requirements

These are the main requirements for the data usage report. It is worth noting that the stage 2 requirement is a feature block of a "nice-to-have" and will only be implemented when there is extra time in the project.

Identifier	Name	Description
S2_FR1	Collection of data usage	After a session, details such as the start time, the end time are recorded for data analytics (such as aggregation, computation can be done)
S2_FR2	Data usage analytics	With collected data from S2_FR1, the data that is most interest is the visualisation that shows how many students use the service for different periods in the semester.
S2_FR3	Daily usage report	A way to send a summarised usage report for a day.
S2_FR4	At-a-glance usage infographic	End of each week to summarise usage/waiting times/peak usage times/ comparisons to previous weeks and/or semesters

"Nice to have"

Some of the "nice to have" of this project will be covered in this requirements documents. However, "nice to have" usually will come along as the users of the system see fit. This will be documented in the [Issue Tracking Management system](#) of the code repository.

Non-functional Requirements

Identifier	Name	Description
NFR1	Security	Only authenticated and authorised users should be able to perform actions such as adding equipment, updating equipment location and information, or searching for specific equipment.
NFR2	Performance	The loading time should not hinder the user experience and productivity of the user in the website. The page/actions should have a loading time < 5 seconds on most computing environments on standard internet connections**
NFR3	Maintainable and extensible	The website should be relatively easy to update and extended to accomodate for new contexts.
NFR4	Recoverable	In the event of the web server or database server crashing, all stored data should be fully recoverable.
NFR5	Intuitive user interface	The website should have an intuitive / easy-to-use user interface, so that users will be able to easily use the website and update the equipment database
NFR6	Compatibility	The application should be compatible with recent versions of the major browsers (Safari, Chrome, Firefox and Edge) on laptop and desktop computers
NFR7	Deployability	The application should be compatible with deployment in the SHL VPS

3.2.4 Proposed Solution

With the requirements and analysis of the current system, the decision of the team is to create a custom web application. The web application will be built with a MVC (Model-View-Controller) pattern using Flask (for API requests), SQL (for database solution), and Jinja Templating with core web technologies - HTML, CSS, and JavaScript (for user interface). The system will be deployed the in the UWA infrastructure. See below for the high-level solution architecture of the system.

Core Technologies

The custom web application will aim to satisfy all the requirements in here along with the "Nice to have" as they come up. The application will be built using the ...

API – FLASK

Flask is an easy-to-use Python web framework. This is chosen as the development team has a Python background and is the web framework used in CITS3403 – Agile Web Development.

DATABASE SOLUTION – POSTGRESQL

SQL is a language that can be used to build and interact with database. The exact database management solution will be with PostgreSQL – a widely used and reliable database in the industry.

USER INTERFACE – HTML, CSS, JS

The core web technologies will be used for building the interfaces. No advanced framework will be used with concern to higher learning curve on top of the existing technologies being used. To aid with the design, the team will be using Bootstrap as a CSS framework to reduce the time needed for creating “good-looking” and intuitive interfaces.

DEPLOYMENT - DOCKER

Docker is a deployment technology that allows virtualization in a server to allow the packaging of software into containers for deployment. To satisfy NFR7 - Deployability, the web platform will use docker to allow the deployment through the SHL VPS Server.

Furthermore, Docker will be used for orchestration of different services in development to increase speed of development, and reduce inconsistency between developers devices (NFR3 - Maintainable and extensible).

Deployment will be done with docker containers in the UWA System Health Lab VPS. As part of NFR1, this deployment methodology will be within the UWA infrastructure (VPS and Cloudflare Proxy).

VERSION AND QUALITY CONTROL

The version and quality control will be facilitated using Git (a version control tool), and GitHub to facilitate peer reviewed code.

DOCUMENTATION - GITHUB MKDOCS

The documentation will be informally made in Microsoft Word documents hosted in OneDrive. However, for future maintainability purposes, these will be uploaded to GitHub along with the source code.

CODE QUALITY

The code quality will be ensured by peer reviews between the developers in the team.

CODE STORAGE AND DEVELOPMENT CONTROL

Git source control will be used, using the remote UWA System Health Lab organisational GitHub (NFR3 - Maintenance and Extensibility).

Prototype

See the Prototype mentioned in [Figma Interface Prototype](#)

3.3 Coding Patterns

3.3.1 Casing

This codebase will be using camel casing.

3.3.2 Linters / Formatters

This will automatically format your code if you install [ESLint](#) in VS Code or type `yarn lint` in the specific folders.

Make sure you have installed the devDependencies so additional linters can be used.

3.3.3 Github Issues and Pull Requests

Most changes in the codebase can be matched to a github issue that contains description of the work that needs to be done. Each of the pull request are matched to this github issue with the branch name that has a standard `c{Issue Number}-{branch name}`. The issue number allows referencing especially when resolving reason for change.

3.3.4 Development with Docker

The development is done with Docker to orchestrate multiple services as defined in the `docker-compose.yml` file:

- Documentation at localhost:8001

3.3.5 Inconsistencies

During the project, different developers have different terminology. Some of the inconsistencies are documented below

3.3.6 `queue` means Team Name as well

`queue` refers to where the student belongs to in the queue. This can either be `StudySmarter` or `Librarian` team.

3.4 Frontend and Backend Technical Documentation

The web application can be mainly be divided into two sides: frontend or also known as client-side application, an application that runs inside the browser of the user, backend or also known as server-side application, an application that is connected with a database to facilitate data validation, transformation, and storage.

The choice of a hybrid approach is deemed to be appropriate to get the best of both worlds.

- Client-side application with AJAX allows the user of the website to have a better user experience with minimal latency when using the application. In other words, this is also known as client-side rendering as it allows the generation of the frontend dynamically without reloading.
- Server-side application allows processing of the HTTP requests, but also generates a simple starting user interface before the user interacts with the application.

3.4.1 Frontend

The relevant files for these section can be found at `app/app/static` for the assets (CSS + JS) and `app/app/templates` for the Jinja-HTML files.

Frontend Rendering

The frontend is rendered using [Jinja2](#), the usual template engine for Flask.

Frontend Design

The design of the website is from Material Dashboard, a bootstrap template. The documentation for it can be seen [here](#) with live demo [here](#).

This was chosen to accelerate the development of the project.

Client-Side Interaction with AJAX

AJAX allows browsers to fetch data without reloading. Essentially, makes it a better user experience.

Notice the Javascript files in `app/app/static/custom`, these are the custom JS, and notice that each page in `app/app/templates` loads a particular JavaScript file.

Frontend Routes

The frontend routes are defined in `routes.py`.

3.4.2 Backend

Database

The database is a PostgreSQL connected with the classic ORM for [Flask - SQLAlchemy](#). With [Flask-Migrate](#) for database migration (combo of SQL Alchemy and Alembic).

All the database models can be found in `models.py`, and database migration scripts at `migrations` folder.

Docker Auto Migration

The docker `runtime.sh` is configured to do auto migration. If you need to do a reverse migration, or anything very custom, Please be prepared to read the [command reference](#).

Application Programming Interface (API)

API is where the AJAX interaction points to. The api endpoints are defined in `app/app/__init__.py`.

At the time of this writing, there are three main API endpoints: - `queue` for the main operation of digital queue - `export` for the exporting functionality - `data` for the analytics page

See the individual files for more information about it.

3.4.3 Application Configuration

The application configuration containing database information and encryption key setups can be found in `config.py`.

This is just standard, nothing special or out of the ordinary.

3.5 Automated Testing

3.5.1 Unit Testing

Unit testing is created using [Pytest](#). Refer documentation closely to [Pytest-Flask](#) for this project.

conftest

`conftest.py` are a special file for pytest that is automatically loaded by pytest and typically contains fixtures and other setup code.

Docker Container Running

Make sure that you are running the docker container before doing any testing.

How to run unit tests?

Use the docker remote code execution

```
1 docker exec -it resq_app pytest
```

or if you want to generate the coverage data

```
1 docker exec -it resq_app coverage run --source="." --rcfile=.coveragerc -m pytest
```

Coverage File

This will produce a file called `.coverage` that contains the records and can be converted to reports.

Test Results

Whenever you are running these tests, it will produce a folder called `test_results` that will contain results of the test. Refer below for more information about Allure.

HOW TO GET COVERAGE REPORT?

If you've run the tests using the above command and have `.coverage` file, you can generate the reports in multiple ways. More information in [here](#).

```
1 coverage report -m
```

will print the coverage report in the terminal.

```
1 coverage html
```

for the html report.

What is `.coveragerc` file?

This file contains the configurations for the coverage testing.

Unit Testing in Pipelines

This was part of [#12](#), but was cut out of scope. Some artefacts of the code can be seen here.

One particular one is the docker `runtime.sh` that can accomodate a `APP_ENV=UNIT_TESTS` to only run unit tests inside the docker.

3.5.2 Allure Report Generator

Allure Testing report is used as a tool to generate test report. More information here <https://docs.qameta.io/allure/>

This repo has a file called `send_and_generate.py`. It is a simple script that sends a test report to Allure and generates a report. This is currently integrated with the [UWA System Health Lab Allure Setup](#). Documentation can be seen [here](#).

3.5.3 Integration Testing

The integration tests are created with [Cypress](#). This integration testing allows us to create frontend tests and obtain video or screenshots that can improve the quality of the test.

Running Tests Via Cypress IDE

Installation Prerequisite

1. Install [Nodejs](#) and NPM (should be automatically installed with Nodejs).
2. Go to `integration_tests` folder and do `npm install`. That is going to install your `package.json` packages.

1. In one terminal, run `docker-compose up` for running application.
2. Without closing terminal opened in 1), open a new terminal and go to `/integration_tests` and:
 - Run `yarn run cypress:open` or
 - Run `npm run cypress:open`

Note that by using this, the Cypress IDE will be running outside the docker container.

Configuring Cypress

Cypress configuration can be adjusted in the `cypress.json` file. If you want to define global variables, this is the place to do it. Global variables can be used in the code with `Cypress.env('VAR')`.

More information can be seen in the official docs <https://docs.cypress.io/guides/references/configuration>

Writing Tests

There are 2 ways to create tests:

1. By writing code tests, see [Writing Test](#)
2. Or by using the [Cypress Studio](#)

Cypress studio makes it easy to create test by recording the user action through the Cypress IDE. Upon action of the user, Cypress studio writes codes in the tests which could be adjusted later for flexibility.

Use Cypress Studio Test Creation

Firstly, you need a test suite ends with `.spec.js` in the `integration_tests/cypress/integration` folder (either newly created from a test template or existing test).

Test Template

```
1 describe('Test Suite', () => {
2   beforeEach(() => {
3     // Setup Scripts Here
4   })
5
6   it('Test Name', () => {
7     // Extend test with Cypress Studio
8   })
9 })
```

1. Select a test suite in the Cypress IDE to extend
2. Hover on the test, and click the "magic wand" icon near the test
3. Interact with the Cypress Web interface as if you are the test runner
4. Save the test when you are done
5. Modify the tests created by Cypress IDE as you need

Magic Wand Icon

Magic Wand Icon

Test files and directory organization

All test specs are located in `integration_tests/cypress/integration`. We recomend follow an structure by feature.

In `integration_tests/cypress/support` you can write reusable pieces of code for execute in all your tests with [commands](#).

Test Results

Results after tests suite finished are stored in the following folders:

- `integration_tests/cypress/screenshots`: Screenshots generated by cypress
- `integration_tests/cypress/videos`: Videos generated by cypress
- `integration_tests/cypress/results`: Allure formatted results

Allure Integration with Cypress

Cypress test runner has installed an allure plugin which is used to generate tests results with allure format.

- [Documentation](#)

3.6 Continuous Integration Pipeline

These are just scripts that run whenever you do pull requests and successful merges. There are a couple of scripts that are currently configured see `.github/workflows`:

3.6.1 Automated Documentation Deployment `docs.yml`

This automatically deploys this documentation whenever `main` is updated with new changes.

3.7 Deployment

The deployment of ReSQ is in the UWA Infrastructure (to be precise in the [UWA System Health Lab](#)). The reason being is that permission is granted to Frinze Erin Lapuz (Software Team Lead of the Redbacks Team at the UWA System Health Lab).

There a couple of steps that were involved in doing this:

3.7.1 DNS Configuration

This is a 1 time configuration

The domain name is set to "resq.systemhealthlab.com" in UWA Cloudflare.

3.7.2 NGINX Configuration

Using [Binchicken](#), I have created the NGINX configuration that will handle all requests going to the application (reverse-proxy).

Nginx Configuration

```

1  server {
2      server_name resq.systemhealthlab.com;
3      location / {
4          proxy_set_header Host $host;
5          proxy_set_header X-Real-IP $remote_addr;
6          proxy_pass http://localhost:10023;
7          proxy_set_header X-Forwarded-Proto $scheme;
8          proxy_http_version 1.1;
9          proxy_set_header Upgrade $http_upgrade;
10         proxy_set_header Connection "upgrade";
11         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
12         proxy_read_timeout 3m;
13         proxy_send_timeout 3m;
14     }
15
16     listen [::]:443 ssl;
17     listen 443 ssl;
18     ssl_certificate /etc/letsencrypt/live/systemhealthlab.com/fullchain.pem;
19     ssl_certificate_key /etc/letsencrypt/live/systemhealthlab.com/privkey.pem;
20     include /etc/letsencrypt/options-ssl-nginx.conf;
21     ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
22 }
23
24 server {
25     listen 80;
26     listen [::]:80;
27
28     server_name
29         resq.systemhealthlab.com
30         www.resq.systemhealthlab.com;
31     return 301 https://resq.systemhealthlab.com$request_uri;
32 }
```

Diagrammatic Explanation

graph TD
 User -->|User goes to resq.systemhealthlab.com| Cloudflare
 Cloudflare -->|Sees that it is under the DNS registered on VPS| VPS
 subgraph VPS
 UWA_Infra[UWA Infrastructure]
 VPS -->|Configuration on Locations| NGINX
 NGINX -->|Host Port Location| VPS
 end
 VPS -->|VPS subgraph| Docker
 Docker -->|ReSQ subgraph| ReSQ
 ReSQ -->|Other_UWA_SHL_Services end end

3.7.3 Deployment with Docker Image

This requires access towards the application inside the VPS. The easiest way to do this is to have access with the VPS through SSH (you may need permission for this).

Once you are in there, do

```
1  git pull
```

to pull in the new changes from the `main` branch.

```
git pull
```

This assumes that you already have the repository in the VPS. If it does not exist, just do `git clone`.

Then run

```
1 sh deploy.sh
```

This will rebuild all the containers (for production) as well as the new code.

3.7.4 Gunicorn Process in Production

The reason as to why we gunicorn process instead of `flask run` in production is for the main following reason:

- gunicorn allows parallelising of HTTP request
- automatic disconnect towards the database after the short-lived process (of responding to HTTP request)

More information about its setup [here](#).