



SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

50.003 Final Group Report

Cohort 4 Group 6

Ang Sok Teng, Cassie (1004542)

Chang Min Xuan (1004337)

Pan Feng (1003689)

Peh Jing Xiang (1004276)

Table of Contents

Table of Contents	2
Deliverables	3
Github Link to our project:	3
Notes to Navigating Github Repository	3
Notes to Running our Application	3
Notes to Accessing Staff and Tenant App Accounts	4
Notes to Accessing Gmail Accounts	4
Requirement	5
Use Case Diagram	5
Design	6
Class Diagram	6
Sequence Diagram	8
Implementation Challenges	11
Engineering challenges	11
Frontend	11
Backend	12
Testing challenges	14
Frontend	14
Testing	15
Unit Testing	15
Backend	15
System Testing	16
Frontend	16
Robustness Testing	18
Lessons Learnt	20
Appendix	21
Appendix A:	21
Frontend Screenshots	21

Deliverables

[Github Link](#) to our project:

<https://github.com/minxuan77/TecWatch.git>

Notes to Navigating Github Repository

In the interest of keeping our past changes for tracking progress, we have left a number of deprecated files and unused materials in the repository across different branches. The updated, cleaned and functioning version of our application lies in the **master** branch. Other branches are previous iterations and testing branches for various components, such as earlier React UI exploration on **cass-testing-branch**, and **checklist-branch** where major changes to the Django application functionalities first arrived.

For reference, the bulk of our critical codes in the **master** branch can be found in the **TecWatch**, **Singhealth** and **Checklist** folders to run the application. Other important files at the top level folder includes the upload of the **use case diagram draw.io** .xml file contents, **Class_diagram** (PDF file), and **sequence diagram** (both png and .puml files).

We understand that due to the large size of the diagrams, many have become difficult to view in a document format. Hence, we have accompanied the diagrams in each section with the working file name on our Github repository, embedded with links to their respective directory.

Notes to Running our Application

Due to the nature of Django, we have left some directions to running the code in our repository at our Github [README.md](#) page for reference.

Notes to Accessing Staff and Tenant App Accounts

We implemented a User Group log in function, where Staff and Tenant users are only able to view pages they are responsible for, and will be directed to their respective pages. These are accounts that can be used to access each user path only, and not the other. We have also included the Administrative User Group level account that is the highest privilege group, used to create and manage all Staff and Tenant accounts.

User	Email	Password
tenant_1	spamditenant1@gmail.com	~1qaz2wsx
tenant_2	spamditenant2@gmail.com	
staff_1	spamdisstaff1@gmail.com	
staff_2	spamdisstaff2@gmail.com	

For access to our database at **localhost:8000/admin**:

Administrative User	Password
tecwatch	tecwatch

Notes to Accessing Gmail Accounts

We implemented features that involved sending emails to relevant users when some conditions are fulfilled. Hence, we have sent up Gmail Accounts, 2 for Staff accounts and 2 for Tenant accounts. We have included the account credentials below so that you could access them to view the emails sent.

User	Email	Password
tenant_1	spamditenant1@gmail.com	~1qaz2wsx
tenant_2	spamditenant2@gmail.com	
staff_1	spamdisstaff1@gmail.com	
staff_2	spamdisstaff2@gmail.com	

Requirement

Use Case Diagram

Due to the secure nature of the Django framework, most misuse cases stem from brute-force attacks on the system directly at login. Hence, we only identified one misuse case by attackers here. Other potential cases like invalid form input have already been handled explicitly and can no longer be carried out. We have more detailed explanations in our [Testing](#) section.

This can be found in the file: [use case diagram.drawio](#). To view, first export the contents to .xml file format, then open the file in [draw.io](#).

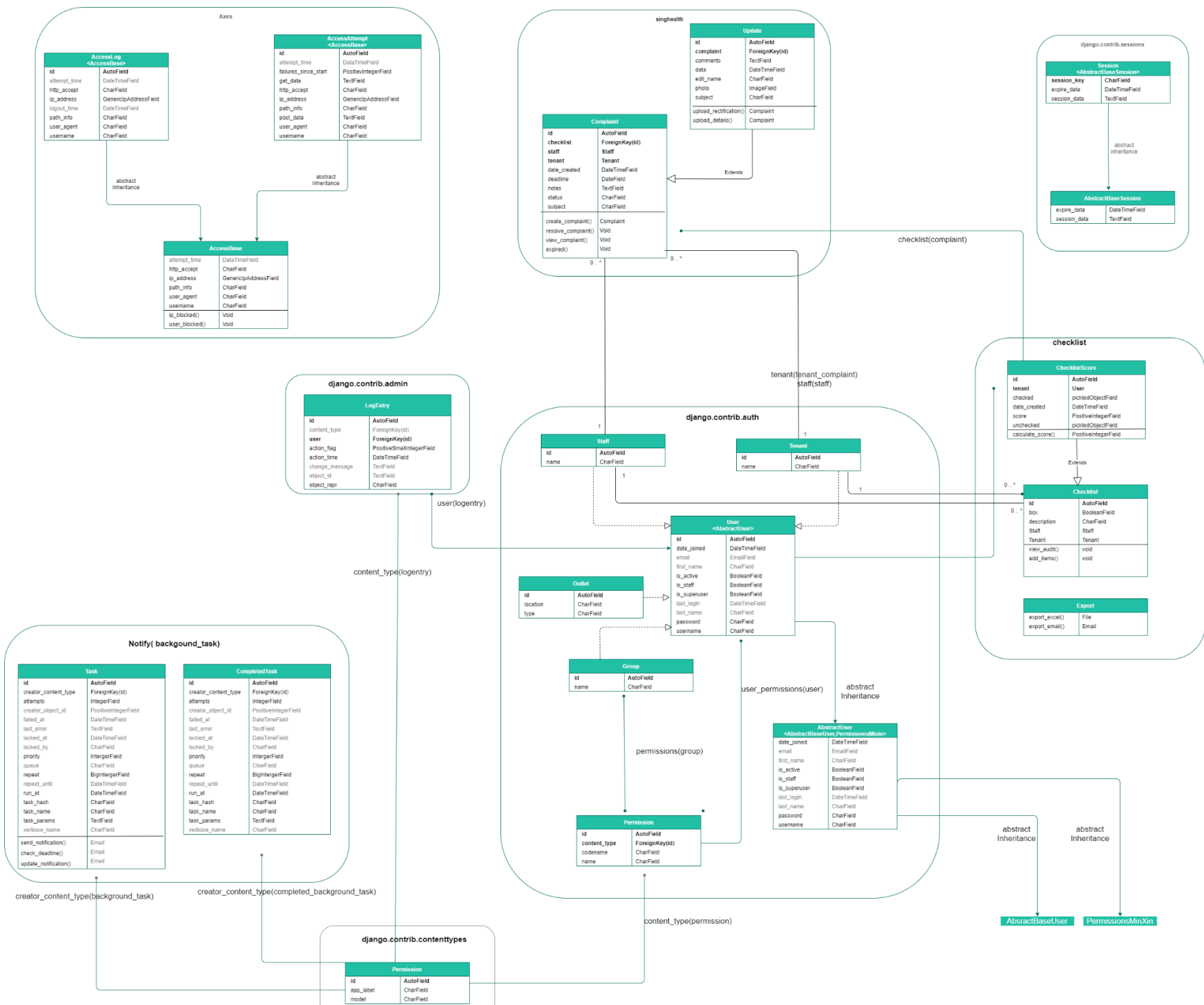


Design

Class Diagram

Due to the size of the image, the resolution has been lowered here, hence we have included this diagram in our Github repository for convenience as well, in the file: [Class diagram.pdf](#)

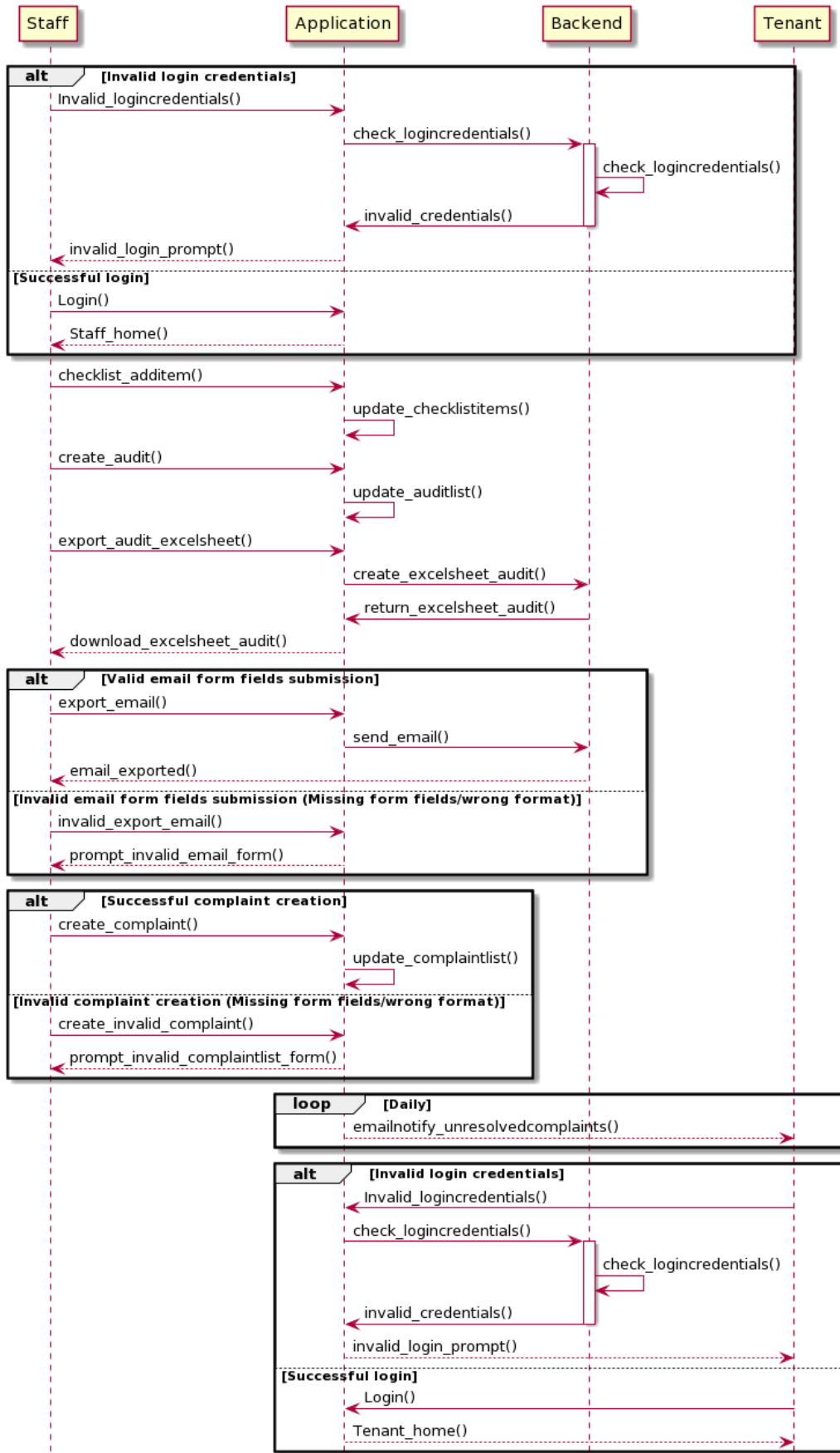
The class diagram represents the various models and their functions that we have in our Django web app and how they interact with one another.

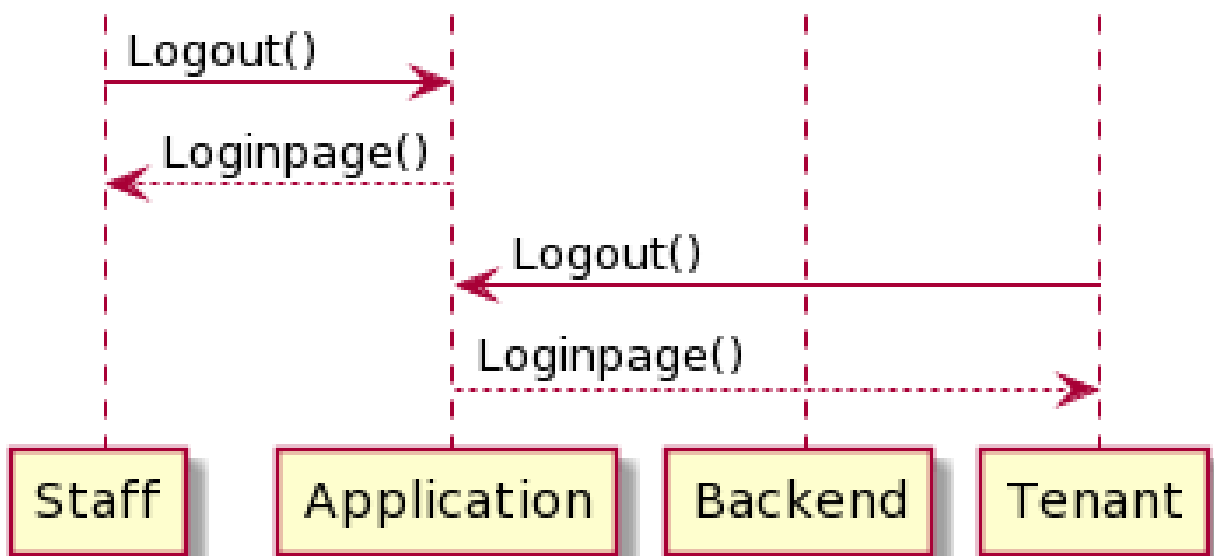
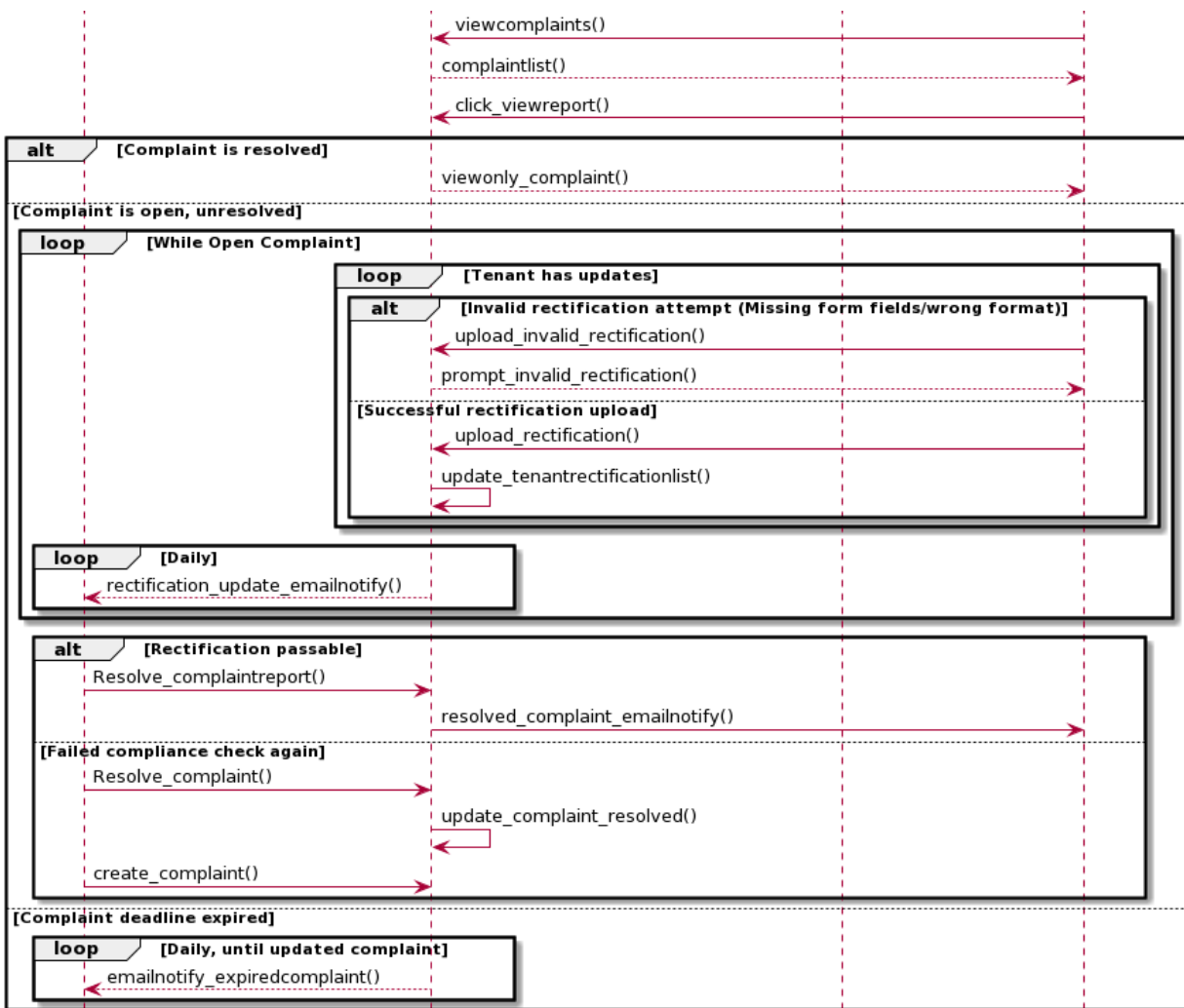


Sequence Diagram

We created a combined sequence diagram to illustrate the main flow of usage by both Staff and Tenants on the application. Due to the diagram being too long, some diagram generation issues arose when using Plantuml to create a complete image (view image file in Github: [seq diagram withinvalid2.png](#)), and we generated the log out steps separately for illustration purposes. The complete code for the diagram, including the ungenerated steps in the image can be found in the source [finalreport.puml](#) file on Github.

Included in the diagram are normal usage flow, as well as return prompts by our application in the event of an invalid input in different areas of the application. These can be viewed across the next 2 pages.





Implementation Challenges

Engineering challenges

The initial challenge that we had when starting out this web application project was that none of us in our group had any prior experience with any frontend or backend frameworks/tools.

After some brief research, we decided to use React Native as our frontend and Django python as our backend. The first week was mostly dedicated to learning and understanding the frameworks and their implementations.

As we progressed through the second and third week, the second challenge that we faced was the integration of React frontend and Django backend using the REST API. However, despite countless hours poured into learning REST API, we were unable to understand its usage and implementation. Since Django also supports frontend development in HTML, we decided to switch to Django fully and use it as both our frontend and backend.

Frontend

Our initial challenge outside of integration issues was the need to learn Javascript (for React Native, later discarded), HTML and CSS from scratch since we had no prior experience before, which was fairly time consuming at the beginning. The decision to not implement React Native also helped make our application more lightweight as there was no need to include the installation of Node.js packages and modules.

The initial process to draw up a working UI was challenging as well, as we had little idea of how the practical coding process and frameworks were like. Despite this, in order to develop our UI design, we first looked into the 7 basic principles of graphic design, considering many aspects such as balance, colour, space and alignment, with multiple low-to-medium-fidelity mockups on Figma to visualize the user flow in various layouts. This helped us decide on this final layout out of the many possible choices to choose from.

However we were met with a unique challenge of designing the UI of Django forms to match our application, since the forms were preset from the backend creation level, and were already delivered in a certain format on the frontend. It took some experimenting, and learning about Developer Tools on various browser platforms to learn how to work and wrap around the components of different form formats each, aligning them properly to reduce display issues. This also led us to explore options like Bootstrap that helped us to speed up the front end development and enable us to rapidly prototype layouts and experiment with each page individually without affecting other pages and Django applications. Due to the nature of the application being a web application, we also had to learn to handle a reactive UI that is focused on tablet usage, which Bootstrap helped us to handle a good portion of after implementation.

Lastly, we met with issues like understanding the backend application methods and unique errors upon interacting with different features of our application. Many of these were also uncovered by using Developer Tools extensively. To overcome this, as a group on both development ends we learned to communicate clearly so that we could debug together, especially after we learnt about Unit Testing, particularly Selenium testing in class, improving some of the robustness of the application on the frontend such as by prompting users for empty or invalid fields to handle potential misuse cases and errors.

Backend

One challenge that we faced when using Django as a backend was choosing a database. We had 2 options, namely SQLite and Postgresql. SQLite is the default Django database while Postgresql is an external database. Suffice to say that we had no experience in either one. After weighing in the factors such as time required to learn how databases work, integration with Django and the scale of our web app, we decided to go with SQLite. As SQLite is the default Django database, integration with Django would be much more straightforward than using Postgresql. Coupled with the fact that many Django tutorials were also using SQLite as their database, it would be easy to follow the tutorials without having to adapt it to Postgresql, which we were not familiar with. As only the Singhealth staff and their tenants would be using the web app, we should not be expecting high traffic for the app, hence SQLite would be the ideal choice here. Likewise, it is also easy to migrate from SQLite to Postgresql, thus scalability is not an issue.

Another challenge is the authentication system for the users. Since this web app contains sensitive information about the tenants and their non-compliances, security is paramount. The challenge for us is to design a login system so as to prevent brute force attacks.

One measure that we came up with is to disallow the creation of user accounts by external users. Instead, we have 3 possible user groups, respectively labelled Staff, Tenant, and Administrative, with the last user group having the highest privileges. This enables us to improve security by allowing only Administrative level users (in this case, the Singhealth administrative staff) to create and sort new Staff and Tenant accounts linked to their respective authentication information like emails. This enables us to guarantee that all users registered are done so with the knowledge of Singhealth administrative management.

We also created a second layer of security for the Staff and Tenant accounts, done by increasing the complexity of the users' password. Firstly, we require each password to have a minimum length of 8. Secondly, the password should not be a common password. The system will check whether the password occurs in a list of common passwords, if it is, it will be rejected. Lastly, the password should not be entirely numeric. This ensures that our passwords will be sufficiently complex so that they would not be easily brute-forced by attackers.

Another measure is to implement Django Axes. Axes records login attempts to our web app and prevent attackers from attempting further login when they exceed the configured attempt limit. The IP address of the attacker will then be recorded and blocked. If the attacker uses distributed brute force password attacks with different ip addresses, then the username of the account which the attacker is trying to hack into will be recorded and locked. The administrator will then be notified and require the user to change his password before unlocking the account.

Overall, we also faced challenges integrating our code after we have implemented our individual functions. We used Github to share our code. However, one limitation of Github is that it is unable to merge files that have unrelated histories, which happened most of the time as we would create a new branch after implementing a new function in order to keep track of changes made. Hence, we would need to integrate the code manually, which required communication within the group, as well as understand the rationale behind each other's code, in order for the consolidated code to work as intended and that codes were not broken due to another's implementation. Also, generating a

'requirements.txt' file allowed everyone to import necessary packages before running the code, to prevent unexpected errors from surfacing while testing out new codes.

Testing challenges

Frontend

Since our web application depends heavily on user interaction, we devoted a sizable portion of our time towards Selenium testing.

The main challenge when using Selenium with Django was that we could not use our current development database to test our web app. Django will instead use a test database created at the start of every test, which is destroyed when the test finishes. This means that we would have to create new sets of our data such as the staff and tenant users before each test. When we had to test a particular function, say the 'complaint' function, which depends on the existence of an audit checklist, we would have to manually create the audit checklist by running through the steps using Selenium before testing whether the complaint function is working. This creates a lot of redundancy and results in significant time wastage for Selenium testing. One solution that we implemented to reduce time wastage was to test all the functions in one test, for example the `test_user_full_cycle()` test which simulates a full user cycle, from login function to creating audit to complaint and lastly resolving the complaint. This also helps to ensure that we do not miss any critical steps in user interaction along the way.

Another challenge that we had was the error `ElementNotFoundException` when designing the Selenium tests. We realised that we did not create a name for most of our elements when coding the html template. One easy fix was to add in the 'name=' to every button and form element that we had while ensuring that the name is unique and intuitive. It would hence be easy for the tester to reference the object, and as a bonus, further aided our design of the UI by making each element accessible individually on the CSS level, since we were working on both ends at the same time. Another factor that resulted in the error was that our testing code attempted to find the element before the page loaded fully. One solution was to add `time.sleep(2)` everytime we had to search for an element. This allows all elements in the page to load fully before we start to find them.

Testing

Our Github repository test cases can be found under the **master** branch, in the [Singhealth/tests](#) folder.

Unit Testing

Backend

We use Django *TestCase* which uses the Python standard library module, *unittest*.

We implemented testing for our login system, database system and email notification system.

For the login system test located in *test_views.py*, we implemented testing for:

- Login with invalid username and invalid password
- Login with empty password
- Login with empty username
- Login with valid username and password

For the email notification system test located in *test_tasks.py*, we implemented testing for:

- Tenant to receive email notification for unresolved complaints
- Staff to receive email notification on expired complaints made against tenant
- Update of complaint status when deadline is passed
- Tenant to receive email notification when complaint has been resolved by staff
- Staff to receive email notification when tenant has uploaded their rectification for the complaint

For our database system test located in *test_models.py*, we implemented testing for:

- Correct labels for model attributes
- Correct maximum length of model attributes
- Auto generated date created field of complaint is accurate
- Correct model methods implementation eg. `__str__` method

System Testing

Frontend

Blackbox testing done using Selenium, located in test_selenium.py:

Feature Tested	Test Case Sequence
Staff login	<ol style="list-style-type: none">1. Invalid username, valid password2. Invalid password, valid username3. Empty username, empty password4. Valid username, valid password <p>Findings: Login page is working as intended</p>

<p>Full user cycle:</p> <ul style="list-style-type: none"> • Test audit system • Test complaint system • Test rectification system 	<p>Sequence of events:</p> <ol style="list-style-type: none"> 1. Staff login. 2. Staff creates an audit by adding necessary items to the dynamic checklist. 3. Staff checks compliance practices and unchecks non-compliance practices. 4. Staff creates a complaint form based on the non-compliance practices. 5. Tenant login 6. Tenant sees complaint and uploads rectification. 7. Staff login 8. Staff accepts the rectification and resolves the complaint. <p>Findings: If successful, it means that the general workflow is done correctly, no critical issues or bugs have been detected when the features interact with one another.</p>
<p>Complaint form - Test for mandatory fields</p>	<ol style="list-style-type: none"> 1. Empty tenant field 2. Empty checklist field 3. Empty deadline field 4. Empty subject field 5. Empty file 6. Empty Comments

	<ol style="list-style-type: none"> 7. Empty Notes 8. Unable to submit form 9. Fill in all fields 10. Submit successfully <p>Findings: All fields in complaint form are mandatory to ensure that the complaint is explained and documented well.</p>
<p>Export audit to excel + Export excel to email</p>	<ol style="list-style-type: none"> 1. Create audit 2. Export audit to excel file and store it in specified download directory 3. Attach the downloaded excel file to email function and email it to specified email address <p>Findings: If successful, it means that the export to excel and email is working as intended, no critical bugs found.</p>

Robustness Testing

Test using Selenium

Test Implemented	Test Case Sequence
Brute Force Attack	<ol style="list-style-type: none"> 1. Create inputs using Fuzzer 2. Try to login using the fuzzer inputs 3. Failed for 5 attempts 4. Account / ip address locked <p>Findings: If successful, shows that our web app is sufficiently protected from brute force attacks.</p>
Click Random Links	<ol style="list-style-type: none"> 1. Login in as staff or tenant 2. Click random links 3. Check for server error code <p>Findings: If successful, it shows that each of our</p>

	web pages is working or displaying as intended, no unauthorised sites are visited.
--	---

Lessons Learnt

We followed the Agile methodology most closely out of all the development cycles we learnt early on in the course, with some differences. Namely, we had less interactions with the Singhealth representative than originally intended, as most of our concerns and questions had already been answered early on by the correspondent. Additionally, the correspondent was extremely detailed in their desired objectives for the application, allowing us to be familiar with their original auditing process. Hence, although there were few arrangements to meet with the Singhealth correspondent, we still managed to meet the desired features and functions requested for the application.

Beyond that, we worked to meet software milestones within our agreed upon time frames at each iteration. Starting with the base application's log in screen, we steadily progressed from the initial functions to create compliance reports to the final week's implementation of a working email notification system and export functions. We managed to stick closely to our goals throughout the project. This helped greatly with time management, and prevented us from dealing with excessive backlog as we were able to quickly modify, improve and discard codes as necessary along the way, keeping track of our changes by communicating our progress frequently.

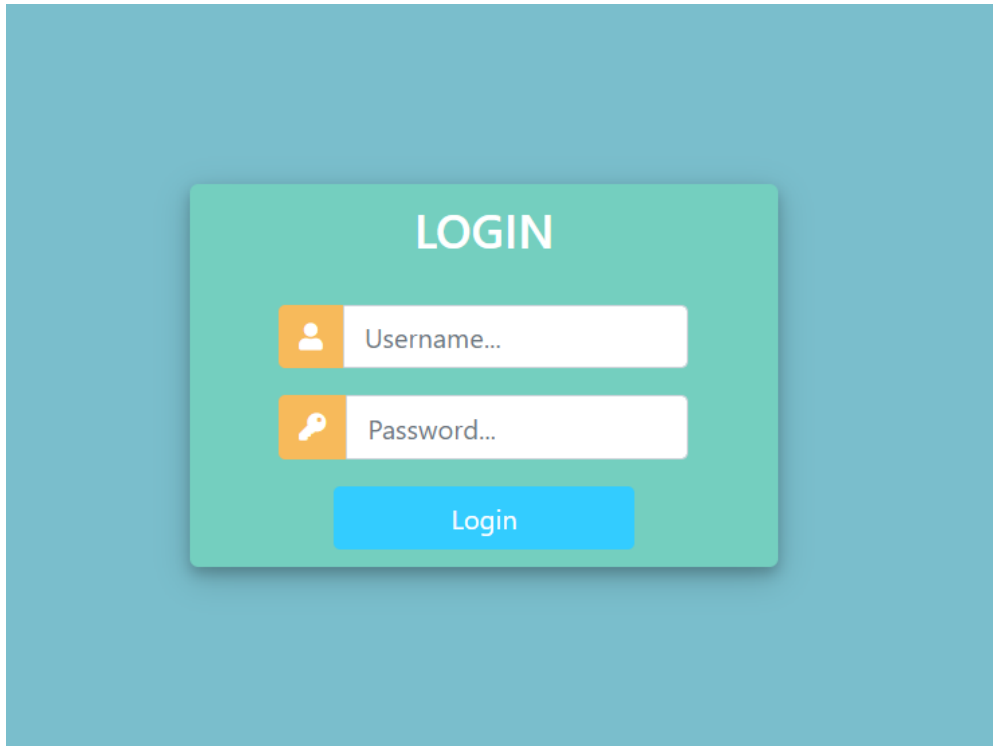
During the development process, we faced difficulties translating the physical auditing process into a digital one. We wanted to develop an application that is easy to use, one that streamlines the process. Developing ways to lighten the paper workload was challenging early on, even during the later stages of the development process. The challenges that we faced include solving the problem of how we wanted the user to interact with the app in contrast to how they might misuse it. However, we learnt that being consistent in updating our user diagrams and clarifying the main application flow with each other before actually tweaking the features went a long way in ensuring that the codes were up to the standard we wanted them to be.

Appendix

Appendix A:

Frontend Screenshots

Login Page



Staff Homepage

Welcome staff_1!

Log Out

You can create a new complaint or view all complaints you have made

Create a new complaint


Checklist Options

Add items to checklist

Update F and B checklist

Update Non F and B checklist

All Tenants




Name: tenant_1

Outlet: North (F & B)

View related complaints

View audits




Name: tenant_2

Outlet: West (F & B)

View related complaints

View audits

Your complaints




Complaint id: 20

Tenant Involved: tenant_2

Subject: Floor is Dirty

Status: Open

View more details



Complaint id: 19

Tenant Involved: tenant_2

Subject: Test 2

Status: Resolved

View more details

Log Out

Welcome tenant_2!

Here is a list of complaints made against you:



Complaint id: 20

Subject: Floor is Dirty

Given Deadline: April 21, 2021

Status: Open

[View more details](#)



Complaint id: 19

Subject: Test 2

Given Deadline: April 21, 2021

Status: Resolved

[View more details](#)



Complaint id: 18

Subject: afsafs

Given Deadline: April 29, 2021

Status: Resolved

[View more details](#)

Create Complaint

Return Home

Tenant:

Relevant Checklist:

Deadline:

mm/dd/yyyy

Subject:

Upload picture:

Choose File

No file chosen

Comments:

Notes:

**Not visible to tenant

Submit Complaint

View Complaint Page

tenant_2

Complaint Records:

Rectification Progress

75.0%

Complaint id: 20
Tenant Involved: tenant_2
Subject: Floor is Dirty
Status: Open

View more details

Complaint id: 19
Tenant Involved: tenant_2
Subject: Test 2
Status: Resolved

View more details

Complaint id: 18
Tenant Involved: tenant_2
Subject: afsafs
Status: Resolved

View more details

Add items to checklist

Add Checklist Items

Description:

Add more items

Select items for checklist

Select checklist items

- ☒ Workplace Safety & Health
- ☒ Healthier Choice
- ☒ Food Hygiene
- ☒ Housekeeping & General Cleanliness
- ☒ Professionalism & Staff Hygiene
- ☒ Tables are dirty
- ☐ Exit blocked

Update checklist

View Audits Page

tenant_1

Audit History:

New Audit

Export to email

Date of Audit: April 13, 2021, 10:36 a.m.

Tenant Involved: tenant_1

Satisfactory Fields:

Housekeeping & General Cleanliness

Professionalism & Staff Hygiene

Tables are dirty

Unsatisfactory Fields:

Healthier Choice

Food Hygiene

Workplace Safety & Health

Total Score: 3

Export Excel

Return Home

Calculate Score for Checklist

Select those that apply

- ☐ Workplace Safety & Health
- ☐ Healthier Choice
- ☐ Food Hygiene
- ☐ Housekeeping & General Cleanliness
- ☐ Professionalism & Staff Hygiene
- ☐ Tables are dirty

Submit

Back

Export to email page

Email :

Subject :

Message :

File :

Choose File

No file chosen

Send Mail With File

Return Home

[View Complaint in Detail Page](#)

[Return Home](#)

Floor is Dirty

Tenant Involved: tenant_2

Current Status: Open

Given Deadline: April 21, 2021

Score: 3

Notes: aaegaeg

Updated by: staff_1

Subject: Floor is Dirty

Date: April 20, 2021, 12:57 a.m.

Uploaded Photo:



Comments: awfawf

[Upload more details](#)

[Resolve Complaint](#)

Upload Rectification: Floor is Dirty

Subject:

Upload picture:

Choose File

No file chosen

Comments:

Update Complaint

Past Details

Current Status: Open

Given Deadline: April 21, 2021

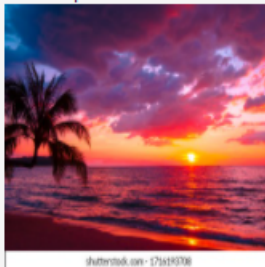
Score:

Updated by: staff_1

Subject: Floor is Dirty

Date: April 20, 2021, 12:57 a.m.

Uploaded Photo:



shutterstock.com - 171619306

Comments: awfawf