Research Practicum Project Report

Dublin Bus Project Individual Report

Tianyu Huang

A thesis submitted in part fulfilment of the degree of

MSc. in Computer Science (Conversion)

Group Number: 5

COMP 47360

UCD School of Computer Science

University College Dublin

August 17, 2022

Table of Contents

Chapter 1: **Major Contributions**

Key words: Static data, New function, Code Integration, Bug fix

My github: https://github.com/alexhuang19940623/dublin-bus

This section details my main contributions to the project, focusing on 1. code integration 2. bug fixing and 3. static data, and I was solely responsible for these three areas throughout the entirety of the project. The primary focuses of me were the addition of new features and the correction of faults on the backend, the construction of a database to manage static data, and providing assistance to Eoin in his development work.

The initial contribution was to handle static data, coming up with a SQL query that required a route identifier and directions and getting all stops on that route. During the initial sprint of the project, I created the database and started working with the static data.

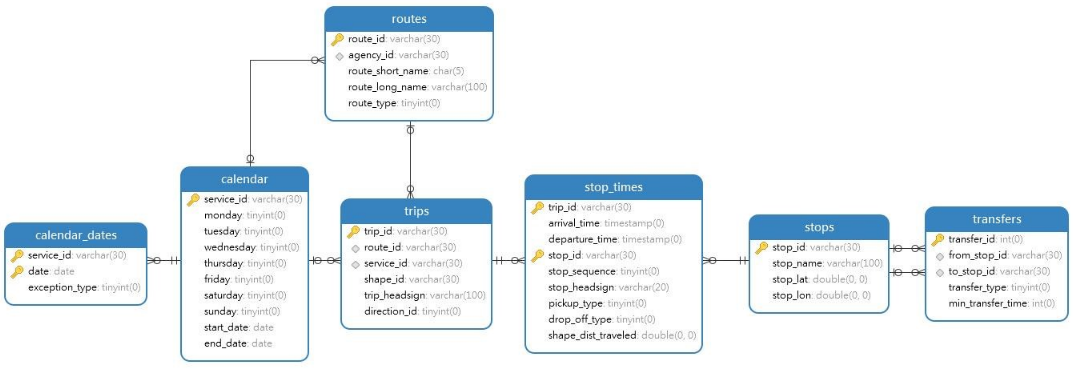


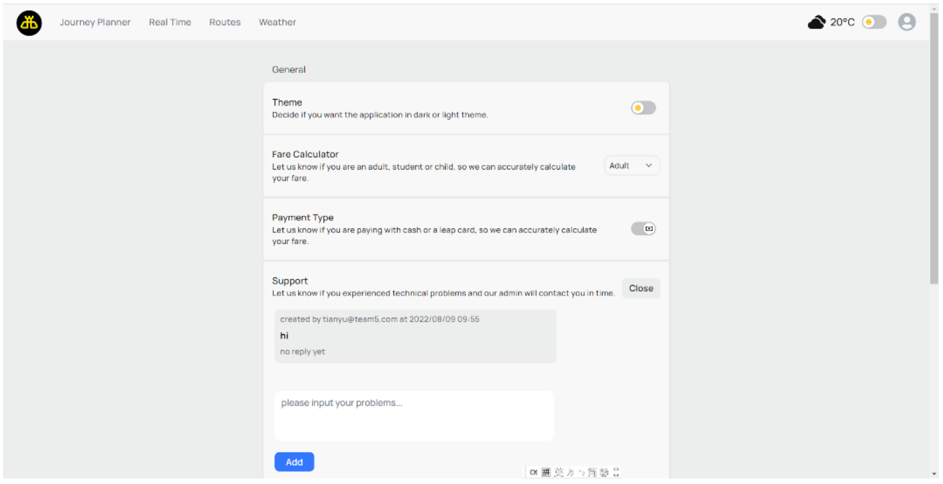
Figure 1.1: My Relational Database

```
The Query is 'select s.stop_id from trips t join stop_times s on t.trip_id=s.trip_id
and t.route_id='10-101-e20-1' or direction_id=0 ORDER BY stop_sequence;'
```

Then I need to find a command that finds number of stops on a particlar route direction when it is given a route and a headsign.

```
After some research, I wrote the query statement as ‘select trip_headsign from routes r
left join (select  route_id,GROUP_CONCAT(trip_headsign) trip_headsign
from trips group by route_id) t on r.route_id = t.route_id order by trip_headsign desc;’
```

My biggest contribution was adding a chat function to our app, re-writing all the jwt files like views.py/serializers.py/models.py/urls.py, setting admin privileges, and backend operations for the database, and making sure the app ran bug free. My aim is to make everything as easy to operate as possible. I added a support function to the already completed code of Eoin. The main purpose of this function is to allow users to communicate with the backend and chat. As shown in the figure below, this feature allows users to chat with the backend in real time. At the beginning, when I conceived the feature, because of my previous experience with databases, I envisioned text storage through the database, as well as calling the database for text deletion and redaction display, and designed the administrator permissions so that the administrator has the right to delete and redact the information submitted by the user, and all back-end interactions are through the database.



General

Theme
Decide if you want the application in dark or light theme.

Fare Calculator
Let us know if you are an adult, student or child, so we can accurately calculate your fare.
Adult

Payment Type
Let us know if you are paying with cash or a leap card, so we can accurately calculate your fare.

Support
Let us know if you experienced technical problems and our admin will contact you in time.
Close

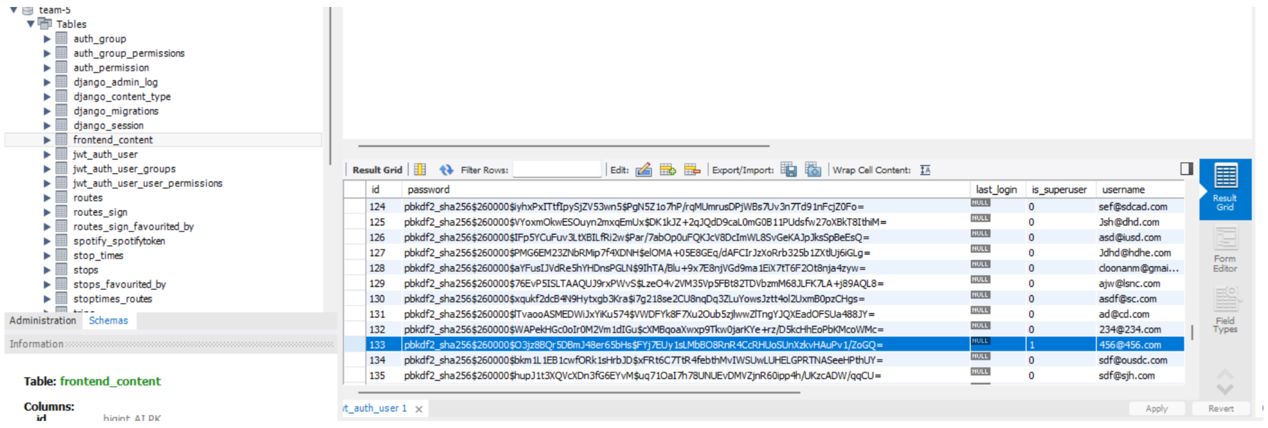
created by tianyu@team5.com at 2022/08/09 09:55
hi
no reply yet

please input your problems...

Add

Figure 1.2: The administrator page can display all content for multiple user accounts

The advantage of using data base directly to set permissions is that it can be easily changed. If I need to set a user as an administrator, I just need to change the database, change superuser from 0 to 1. Also, since the front-end only displays the database content and does not perform any delete or change operations, if I delete the chat content stored in the database, then the front-end page will also be deleted in real time. I designed a special schema for the front and back end connections, and this database is used to receive all the content sent by the user and all the content replied by the administrator.



team-5
Tables
auth_group
auth_group_permissions
auth_permission
django_admin_log
django_content_type
django_migrations
django_session
frontend_content
jwt_auth_user
jwt_auth_user_groups
jwt_auth_user_user_permissions
routes
routes_sign
routes_sign_favourited_by
spotify_spotifytoken
stop_times
stops
stops_favourited_by
stoptimes_routes

Table: frontend_content

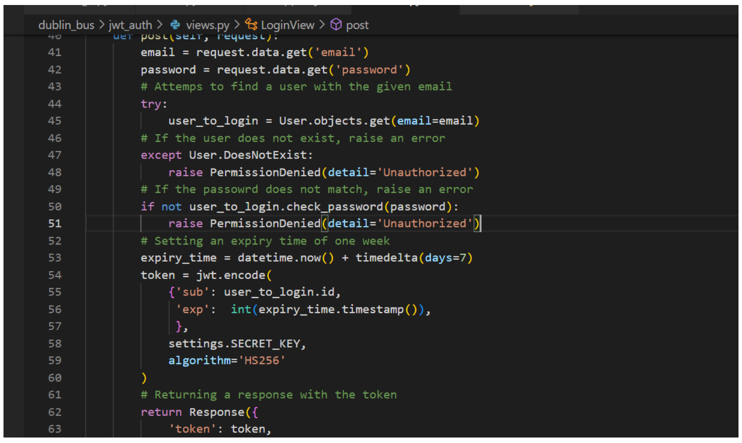
Columns:
id bigint AI PK

id	password	last_login	is_superuser	username
124	pbkdf2_sha256$260000$iyhxPxITtfIpySjZV53wn5$PgN5Z1o7hP/rqMUmrusDPjWBs7Uv3n7Td91nFcjZ0Fo=	NULL	0	sef@sdcad.com
125	pbkdf2_sha256$260000$VYoxmOkwESOuyn2mxqEmUx$DK1kJZ+2qJQdD9caL0mG0B11PUdsfw27oXBkT8IthiM=	NULL	0	Jsh@dhd.com
126	pbkdf2_sha256$260000$IFp5YCuFuv3LtXBILfRi2w$Par/7abOp0uFQKJcV8DcImWL8SvGeKAJpJksSpBeEsQ=	NULL	0	asd@iusd.com
127	pbkdf2_sha256$260000$PMG6EM23ZNbRMip7f4XDNH$elOMA+05E8GEq/dAFCIrJzXoRrb325b1ZXtlUj6iGLg=	NULL	0	Jdhd@hdhe.com
128	pbkdf2_sha256$260000$aYFusIJVdRe5hYHDnsPGLN$9IhTA/Blu+9x7E8njVGd9ma1EiX7tT6F2Ot8nja4zyw=	NULL	0	doonanm@gmai...
129	pbkdf2_sha256$260000$76EvP5ISLTAAQUJ9rxPWvS$LzeO4v2VM35Vp5FBt82TDVbzmM68JLFK7LA+j89AQL8=	NULL	0	ajw@lsnc.com
130	pbkdf2_sha256$260000$xqukf2dcB4N9Hytxgb3Kra$i7g218se2CU8nqDq3ZLuYowsJztt4ol2UxmB0pzCHgs=	NULL	0	asdf@sc.com
131	pbkdf2_sha256$260000$lTvaooASMEDWiJxYiKu574$VWDFYk8F7Xu2Oub5zjlwwZlTngYJQXEadOFSUa488JY=	NULL	0	ad@cd.com
132	pbkdf2_sha256$260000$WAPekHGc0oIr0M2Vm1dIGu$cXMBqoaXwxp9Tkw0jarKYe+rz/D5kcHhEoPbKMcoWMc=	NULL	0	234@234.com
133	pbkdf2_sha256$260000$O3jz8BQr5DBmJ48er65bHs$FYj7EUy1sLMbBO8RnR4CcRHUoSUnXzkvHAuPv1/ZoGQ=	NULL	1	456@456.com
134	pbkdf2_sha256$260000$bkm1L1EB1cwfORk1sHrbJD$xFRt6C7TtR4febthMvIWSUwLUHELGPRTNASeeHPthUY=	NULL	0	sdf@ousdc.com
135	pbkdf2_sha256$260000$hupJ1t3XQVcXDn3fG6EYvM$uq71OaI7h78UNUEvDMVZjnR60ipp4h/UKzcADW/qqCU=	NULL	0	sdf@sjh.com

Figure 1.3: Set the superuser

Also I designed a special schema for the front and back end connections, and this database is used to receive all the content sent by the user and all the content replied by the administrator. Finally, the bug fixing part, because my job is to add new features to the original code, there will inevitably be bugs. The first bug I encountered was the inability to log into my account, a wicked bug for which I rewrote the jwt section. And I found that this bug is determined by the fact that the original code has no support function at all, so if I need to determine the administrator account, then I have to rewrite jwt. Another example is the timestamp problem, at first my login kept expiring, then I rewrote the code to solve this problem, solving the most part of the bug is to rewrite the jwt code.

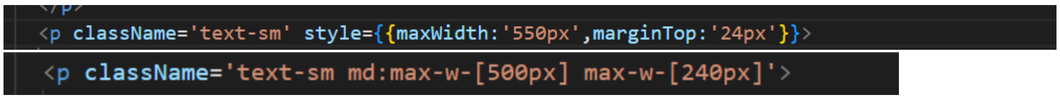
The second bug I found was related to css, as our framework uses tailwind, which can cause some



```python
    def post(self, request):
        email = request.data.get('email')
        password = request.data.get('password')
        # Attemps to find a user with the given email
        try:
            user_to_login = User.objects.get(email=email)
        # If the user does not exist, raise an error
        except User.DoesNotExist:
            raise PermissionDenied(detail='Unauthorized')
        # If the passowrd does not match, raise an error
        if not user_to_login.check_password(password):
            raise PermissionDenied(detail='Unauthorized')
        # Setting an expiry time of one week
        expiry_time = datetime.now() + timedelta(days=7)
        token = jwt.encode(
            {'sub': user_to_login.id,
             'exp':  int(expiry_time.timestamp()),
             },
            settings.SECRET_KEY,
            algorithm='HS256'
        )
        # Returning a response with the token
        return Response({
            'token': token,
```

Figure 1.4: Example of rewriting jwt files

loading problems if I use some code from react. This kind of bug can be fixed by keeping the code uniform. e.g. if you add style to the tag of class at the beginning, change it to tailwind's md kind. At the same time to keep the page ui design consistent, which requires a lot of parameters fine-tuning.



```
<p className='text-sm' style={{maxWidth:'550px',marginTop:'24px'}}>
```

```
<p className='text-sm md:max-w-[500px] max-w-[240px]'>
```

Figure 1.5: Comparison of css changes

Finally, there is the issue of renaming some properties, which can also lead to some infinite loops in loading. For example, I added a new file to the component, I reused the code directly at first, but there was a rename that led to infinite loading, and then I reset all the property names. The bug fixing part was painful, but when I saw that the app worked perfectly, I thought it was all worth it.

In general, the part of rewriting jwt involves the views/model/url files in order to ensure that the app still works properly after modifying jwt, I spent a lot of time on testing, while moving the jwt user authentication from the online server to the local one. A lot of time was also spent on keeping the ui style consistent, and testing for bugs. At the same time, I set the administrator privileges to avoid back-end operation problems and to allow the administrator to easily communicate with users.

As a final note, I also spent a week at first getting to grips with the existing code and making a web version of the chat function, but in the end it wasn't adopted because it was a waste of resources to have a separate web page for the chat feature. The final product is as we showed, the chat feature is integrated into the accounts menu. So my contribution is to integrate features, fix bugs and provide help with static data.

Chapter 2: **Background Research**

Key words: Modularization, Integration, Auth

Django is a free and open-source Web framework developed in Python. It covers almost all aspects of web applications and can be used to quickly build high-performance and elegant websites. Django has built in many modules that are often used in the background development of websites, so that developers can focus on the most personalized parts.

The idea of Django modularization is to extract the most widely used and universal common functional modules according to the common needs of ordinary developers in web application design, which greatly reduces the workload of developers. Taking database operation, the most extensive function in web applications, as an example, Django ORM module provides a unified method for us to operate different types of databases. ORM adapts to many commonly used relational databases, such as PostgreSQL, MySQL, Oracle, SQLite3, etc. Another good example is Django auth module. Any developer knows that when designing and developing a website, he should design the basic user management functions such as registration, login, logout or logout, browsing and subscription from the perspective of "user". However, if the developer has not enough time and experience on the user management system, the system development task will be disaster for the developer. Django is a fully functional web framework, and its auth module can quickly realize the basic functions of the user module. In order to better implement modularization, Django also introduced "templates" to decouple the HTML and view layers.

In all, it's good news for any developer using Django from the beginning of web design. But we know the reality of web design is the mix of existing and outstanding work. How to use Django to integrate code in existing projects? Before Django, the common practice to move applications from other projects to the merged project. And then, the project level files will be merged and reduced to ensure that the parent project will not have duplicate code while the resulting project can still be managed and organized. Under the Django framework, the first step will be to move the existing application to the merged project. The next step is to merge the supporting infrastructure from the parent project into the merged infrastructure. Each Django project ultimately needs only one settings module. When n projects are merged, n modules need to be processed. This part is often the most time-consuming. The result of this step is to end with a module that contains all the settings in the settings module of the parent application.

In many cases, settings are only repeated between projects and can be simplified. However, if there are multiple database instances, you may need to insert some code. We may need to consider migrating data to a single database to completely avoid this situation. Special attention should also be paid to the settings of the third-party application. Following the same method above, we need to copy and paste all the settings of the third-party application into the project settings, and resolve all duplicate settings in the consolidation. The last step is to ensure that the newly migrated applications are listed in the project settings.

In a word, Django is most appropriate for a project from scratch, but it is still feasible to integrate Django framework with existing databases and applications.

Chapter 3: **Critical Evaluation & Future Work**

Key words: Modular design, Practice of commercial operation

My philosophy for this project is modular code, a structure that anyone can understand, and very easy to manage. Taking my own creation as an example, if I want to add the support function, I need to create a new support folder in the components in our src and add content to it, and rewrite jwt. My github repository has uploaded the project after I modified the structure, which is my understanding of modular design.

It's tedious, right? If in reality, a new programmer needs to add a new feature to the current project, for example, I need to add a comment system to the website, it will take a lot of learning time. If we want to achieve modular design, I think our project needs to adjust the whole structure design, for some folders to set up special placement, which is a very abstract concept. The summer project practice was a practical exercise of a commercialized project, so I found out this biggest problem. In reality, the project is not a job delivered and gone, but needs long-term operation and maintenance, so the modular design is important.

The first thing to do is to adjust the file structure, I think all our current file structure is designed for homework. For example, I think we can adjust the model under frontend instead of where it is now. Because frontend is a submodule of django Since it is a sub-module of django, there is nothing wrong with putting the model, url, views, and backend logic into it. And any new features added to the frontend folder are put under src as a new file, because The src in frontend is the front end in development.

And there is the modularity of the url. In the future, if the project is going to get bigger and bigger, then the url is going to get bigger and bigger as well. If everything is put into the main 'urls.py' file, then the management won't be very good. In most cases, we will create a new "urls.py" file within the app to hold all of the app-related sub-urls. This is done so that you may put each app's unique urls into itself for administration purposes. In addition, there are split settings files, the primary purpose of which is to take into mind the deployment in various environments. When working on real projects, we frequently need to generate many sets of configuration files for various settings, such as development, testing, and production. These configuration files include database account passwords, project keys, and other relevant information. The essence of splitting settings is the rewriting of variables within base.py. Other minor modifications that I believe may be done to the.gitignore file include, as suggested by the file's name, ensuring that files that fit the rules provided in the file are not added to the git repository. We can avoid adding duplicate static files, compiler caches, IDE temporary files, etc. to the repository by utilizing this file. Here is an example of a jetbrain-specific.gitignore file to which we may add our own rules to streamline the github repository. Currently, we do not do this step, but it will be necessary in the future when we confront actual commercial projects.

I could say a lot more about similar modifications based on space limitations, but in the end, everything is designed to be quickly grasped by new beginner programmers.

Bibliography

[1] Mukul Mantosh, 2021. Exploring Project Structure Creating Django App. [Online] Available at: https://www.jetbrains.com/pycharm/guide/tutorials/django-aws/project-explore/ [Accessed 4 July 2022].

[2] Big Nige, Mastering Django: Structure. [Online] Available at: https://djangobook.com/django-tutorials/mastering-django-structure/ [Accessed 5 July 2022].

[3] WebDataRocks, Integration with Django. [Online] Available at: https://www.webdatarocks.com/doc/integration-with-django/ [Accessed 10 July 2022].

[4] TechVidvan, Django Project Structure and File Structure. [Online] Available at: https://techvidvan.com/tutorials/django-project-structure-layout/ [Accessed 10 July 2022].

[5] DelftStack, 2021. Best Practices for a Django Working Directory Structure. [Online] Available at: https://www.delftstack.com/howto/django/django-project-structure/ [Accessed 10 July 2022].

[6] Humphrey, 2021. Integration tests in Django. [Online] Available at: https://www.codeunderscored.com/integration-tests-in-django/ [Accessed 11 July 2022].