A Level computing coursework

Reactive web timetable

Centre number: 64690

Candidate number: 4059

Contents

[Analysis of problem 3](#_Toc507070440)

[Problem identification 3](#_Toc507070441)

[Stakeholders 3](#_Toc507070442)

[Lifeguards 3](#_Toc507070443)

[Pool manager 3](#_Toc507070444)

[Club management team 3](#_Toc507070445)

[Research 3](#_Toc507070446)

[Currently available solutions 3](#_Toc507070447)

[Essential features of solution 3](#_Toc507070448)

[Limitations 4](#_Toc507070449)

[Requirements 4](#_Toc507070450)

[Success Criteria 4](#_Toc507070451)

[Design of Solution 4](#_Toc507070452)

[Description of solution 5](#_Toc507070453)

[Algorithms? 5](#_Toc507070454)

[Usability features 6](#_Toc507070455)

[Testing 6](#_Toc507070456)

# Analysis of problem

## Problem identification

Imber Court is a health club in Thames Ditton, Esher Surrey. On the grounds there is a swimming pool which operates as a discrete division of the club, with workers that report to the pool manager. It offers general swimming, swimming lessons and various swimming activities for members of the club and their guests. At all times that there is a member of the public in the pool must be in compliance with the health and safety at work act (1974), section 3, which states *“It shall be the duty of every employer to conduct his undertaking in such a way as to ensure, so far as is reasonably practicable, that persons not in his employment who may be affected thereby are not thereby exposed to risks to their health or safety”.* This means that they must ensure that the person who is swimming in the pool is reasonably safe or they can be held culpable for any lapses in health and safety that could lead to a member of the public being injured. For this reason, at all times that the pool is open there must be a trained lifeguard watching the pool.

Currently, the lifeguards know who is supposed to be working at a given time each week based on a printout of a spreadsheet that is pinned to a board in the office. This system is not ideal as it means that the pool manager has to print and pin up the spreadsheet each week, and if you have not been in the pool before your shift you do not know what time to be there unless you ask someone to send you a picture of the spreadsheet. This is not ideal as if the shift pattern is changed and the lifeguard is not told it could lead there not being a lifeguard on poolside which could potentially mean the club has to close the pool until a lifeguard is found.

This problem is solvable by computational methods as the ubiquity of the internet means that people can see information in their home or anywhere they can access the internet. As well as this, due to the position of the board and the information displayed on it the current solution is not compliant with the new data protection laws coming into force in 2018 (GDPR).

The problem is amenable to a computational approach because the main problems with the current system are the multiple possible points of failure in the rota being posted on the board and the set location of the rota. In the case of the former, if a database was used then the rota would not have to be printed each week as the website could just display the same database each week, and in the case of the latter the aforementioned ubiquity of the internet means that the a computationally focused solution would be suitable as people could see the rota from anywhere they have access to the internet. Finally, an internet-based solution would be GDPR compliant as it would not require the posting of personal information in the view of the public.

## Stakeholders

### Lifeguards

The lifeguards would be the primary users of the system and so would be stakeholders as they are the ones who would be accessing the information on the database solely for consumption. A web-based solution would be appropriate as they need to be able to see the times that they are working from anywhere with internet access so that they can check what time they need to be at the pool. As well as this, they need their data protected by law and a web-based solution would do this.

### Pool manager

The pool manager would be the person who would be updating the database and so would be a stakeholder. A computational solution would be appropriate to his needs as it would remove the need to print off the rota each week and would so ensure that the lifeguards have access to up to date information concerning their shift times. This would ensure that he doesn’t have to close the pool and would reduce the amount of time spent by him on uselessly printing off paper and arranging last minute cover. As well as this, a computational solution compliant with GDPR would ensure that he has protected the personal data of employees.

### Club management team

As the employees of the pool are employees of the club and the protection of personal information of employees is in the legal interest of the club under both GDPR and the Data Protection Act, a computational approach would also benefit the club management team.

## Research

### Currently available solutions

#### When I Work

When I Work is a currently available web-based timetabling app that allows managers to create new timetables each week with easy to use templates and drag and drop components.

This allows even a boss who isn’t technologically savvy to use the software with ease. As well as this, the app will tell you is the person is qualified for the job that you want to assign them to which could be useful as at the pool different jobs require different qualifications. Thirdly, when a table is updated the app will notify its users, so they know that there has been a change. This could be useful as it would allow everyone to know what is going on at the pool. Also, its free.

However, the freedom of access to information on other people means that I doubt the app is GDPR compliant, which could lead to criminal charges being levelled against the pool.

<https://wheniwork.com/l/cap?ref=capterra&utm_campaign=Employee+Scheduling&utm_medium=Directory&utm_source=capterra>

#### Deputy Rota

Deputy Rota is a template-based rota handling software like ‘When I Work’, but the addition of a portal allows employees to communicate with each other which could be useful in the case of arranging cover work. It also allows the manager of the enterprise to publish the rota and send it to employees in a multitude of formats, however it adds the ability for members of staff to arrange their own leave

However, it also gives the information of employees to each other freely and so I don’t think it would comply with GDPR. To add to this it costs money, which the pool at Imber court is unable to spend.

<https://www.deputy.com/online-staff-rota-software>

#### Google docs spreadsheet

Another possible solution would be the creation of a Google docs spreadsheet that can be shared to all of the employees of the pool who have a Google account. This solution allows the dynamic editing of a communally accessible spreadsheet which would allow users to easily view their shifts.

However, the communally editable property of this solution and lack of ability to set up alerts for if the sheet is changed means that if someone changed their shift the pool manager would not know, possibly leaving the pool unstaffed.

To compound this, due to the fact that the solution requires the use of a Google account and that some people may not have said account, the solution is not GDPR compliant.

### Essential features of solution

#### Web based

The shift rota should able to be seen by anyone with internet access, so It will have to be hosted on some form of web server.

#### Intuitive UI

The user interface of solution should be intuitive so that people will know how to use it without any instruction. To do this I will use a CSS style sheet to make all information easy to read and to make sure that the UI easy to use

#### Relevant information

The information displayed by the solution should be relevant to the user so that the app is as useful as possible to the user. To do this I will probably use some type of global variable to record the username of the user, so that it can be used as a search parameter in the selection of data from the database.

#### Secure

The Solution must keep user’s personal data safe and secure. To do this I will most probably use some form of log in screen and use asymmetric key encryption and a password hash to verify that the inputted password is correct without having to store the user’s password on the database.

### Limitations

Due to my lack of time and skill as a programmer there will be several limitations to the final product. Firstly, as it will not be able to display the information from the database graphically as I do not know how to use JavaScript or PHP. Secondly, there will be no way for the lifeguards to update the database. Thirdly, I probably won’t be able to integrate any form of direct messaging into the web app. Finally, due to my lack of skills as a designer the end product will be useable (hopefully) but I doubt it will look in any way nice.

## Specification of proposed solution

### Solution requirements

The designing and building of the solution will require that I improve my skills at python as well as learning how to write HTML, CSS, SQL and to use Flask (a python-based web app framework) and jinja2 (a templating language that allows you to integrate python into HTML). I will also need to find assorted python libraries to allow me to do things like hashing, password verification and form validation.

On the hardware side, I will need to find a cloud server service that is free as well as creating an instance and uploading my applet to it.

In terms of building, I will need to build a flask web app and an SQL database in order to get the solution to work.

### Success criteria

To be considered successful it must satisfy the following criteria:

* It must be hosted on some form of web server so that it can be accessed from the internet
* It must incorporate some form of GUI to make reasonably it easy to use
* It must allow the use of a database to store employee shift information but not implicitly require the workers actual name be stored in the database so as to comply with GDPR.
* It must use some form of user session system to keep track of who is accessing the database so that it can show them only information that is relevant to them so that it is easy to use
* It must have some mechanism by which the pool manager can keep track of all shifts

# Design of Solution

## Decomposition of Problem

To build my solution, I will need to:

* Build an SQL database to store employee usernames, password hash’s and shift information
* Build a login page with form that includes username and password fields
* write function to compare inputted password’s hash to user’s hash stored in database. If it is, the function should load the profile page. If not, it should return the reload the login page.
* write a profile page in HTML
* write python function that will output users shifts In a list
* write jinja2 code to integrate the values from the database into the web page

## Description of solution

The solution would be made up of three layers working together, these would be a UI written in HTML and CSS, a data handling program written in python and a database that the python program would access.

## 

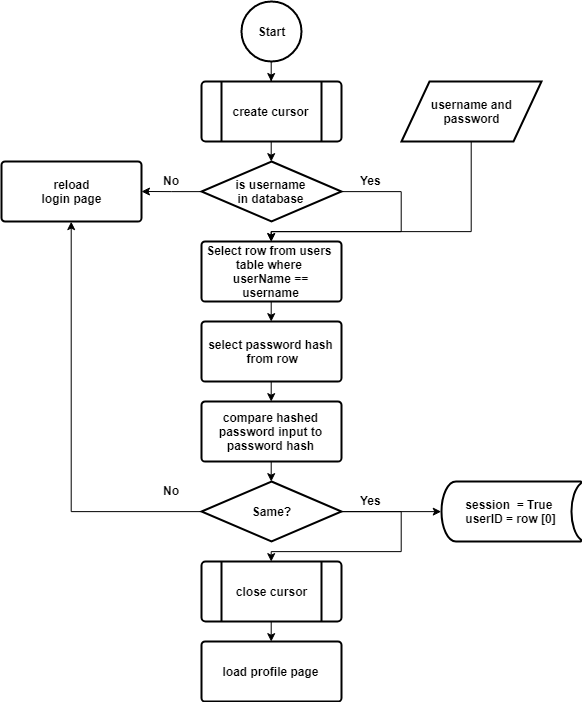
### Algorithms

#### Login algorithm (front end):

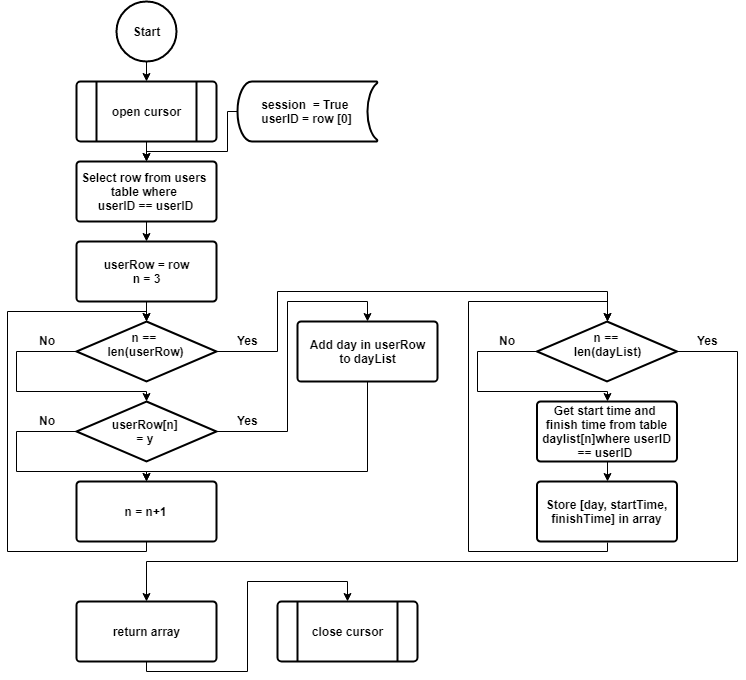
### 

## 

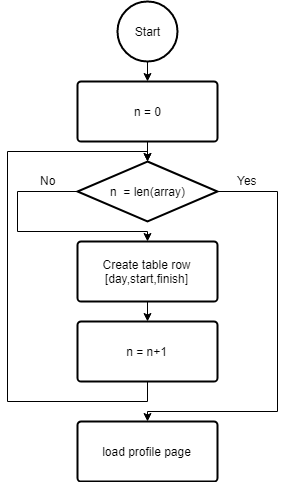
#### Login algorithm (back end):



#### User shift data (back end):



#### User shift data (front end)



## Usability features

The website will be clearly structured to make it easy to use, all form input fields will have placeholder annotations saying what they are. As well as this I will use a colour palette that will allow the text to be read easily.

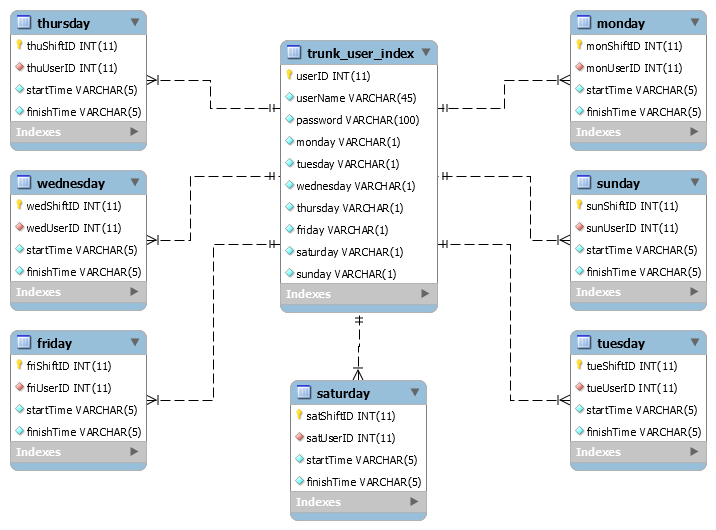
## Key variables, data structures and classes

### Key Variables:

userID will act as the primary key in the trunk\_user\_index table, in the shift tables a foreign key linked to this be made up of the previous key concatenated to the first three letters of each day altered to be half capitalized (eg. monUserID).

### Data structures:

This is the EER diagram of the database that I am going to use:



### Classes

If you mean classes as in for databases, I will be using a user information class and a shift information class

## Test info

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| userID | userName | password | mon | Tue | wed | thu | Fri | sat | sun |
| 0 | Admin | \* | N | N | N | N | N | N | N |
| 1 | Test1 | \* | N | N | Y | Y | Y | N | N |
| 2 | Test2 | \* | N | N | N | Y | N | Y | Y |
| 3 | Test3 | \* | Y | N | Y | N | N | N | Y |
| 4 | Test3 | \* | N | Y | N | N | Y | Y | N |
| 5 | Test5 | \* | Y | Y | N | N | N | N | Y |
| 6 | Test6 | \* | N | N | N | N | N | N | N |

\* $5$rounds=535000$SMQ30riopJ5G57Zf$XdBxYJ.QApfz/ddnEY8KWt69z4WQO2HObpMJJ3pW6cB is a SHA265 hash generated for the phrase “password”.

|  |  |  |  |
| --- | --- | --- | --- |
| monShiftID | monUserID | startTime | finishTime |
| 0 | 3 | 08:00 | 13:00 |
| 1 | 5 | 12:00 | 21:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| tueShiftID | tueUserID | startTime | finishTime |
| 0 | 4 | 08:00 | 13:00 |
| 1 | 5 | 12:00 | 21:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| wedShiftID | wedUserID | startTime | finishTime |
| 0 | 1 | 08:00 | 13:00 |
| 1 | 3 | 12:00 | 21:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| thuShiftID | thuUserID | startTime | finishTime |
| 0 | 4 | 08:00 | 13:00 |
| 1 | 5 | 12:00 | 21:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| friShiftID | friUserID | startTime | finishTime |
| 0 | 1 | 08:00 | 13:00 |
| 1 | 4 | 12:00 | 21:00 |

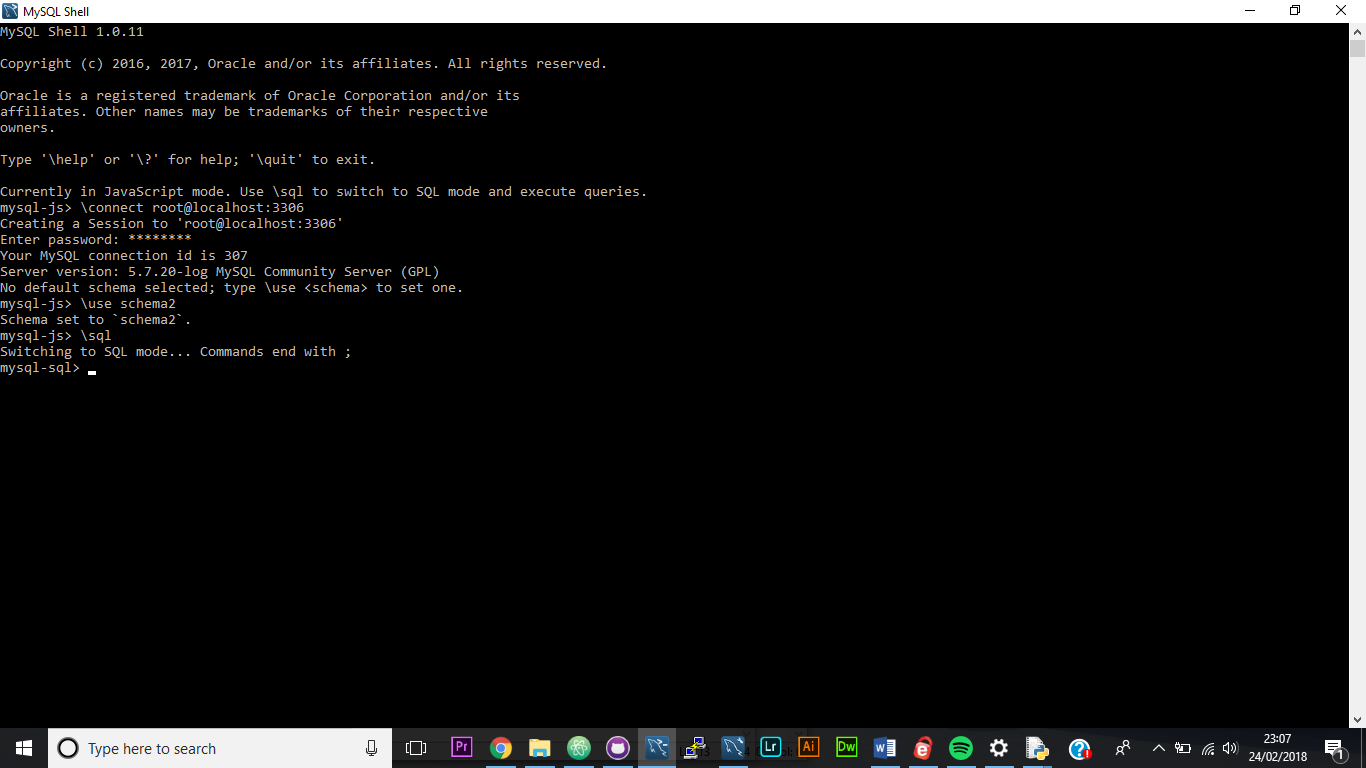
|  |  |  |  |
| --- | --- | --- | --- |
| satShiftID | satUserID | startTime | finishTime |
| 0 | 2 | 08:00 | 13:00 |
| 1 | 4 | 12:00 | 21:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| sunShiftID | sunUserID | startTime | finishTime |
| 0 | 2 | 08:00 | 12:00 |
| 1 | 3 | 10:00 | 16:00 |
| 2 | 5 | 12:00 | 18:00 |

# Development of solution

## Step 1) MySQL database:

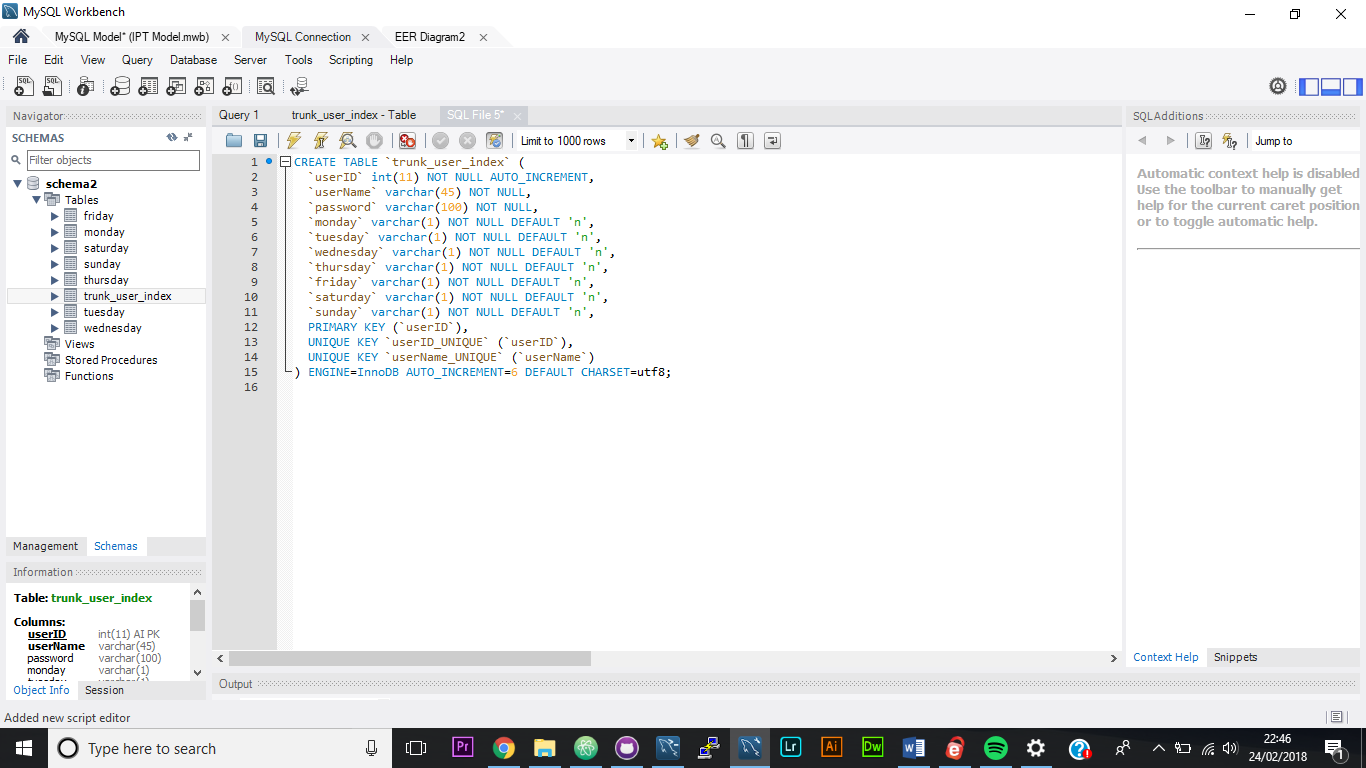
In order to speed up the creation of the SQL database I first downloaded ‘MySQL Workbench’, a free piece of software provided by Oracle in order to aid the creation and upkeep of databases. In this section I will be showing the scripts needed to create the two classes of table that I will use in the database, the ‘trunk-user-index’ and the shift tables. The former stores user information including the ‘userID’, the ‘userName’ and the hash of the password, as well as whether the user is working on a given day. The latter stores the shift information such as the dayShiftID, the dayUserID (which acts as the foreign key), and the start and finish times (in the 24-hour format ‘hh:mm’). I have also downloaded ‘MySQL Shell’, another free piece of software provided by Oracle, which in this case allows you to connect to a database in order to run queries and tables.



In the above picture I am creating a session in which I can connect to a database called ‘schema2’, and then setting up the shell so that I can use SQL commands.

### 

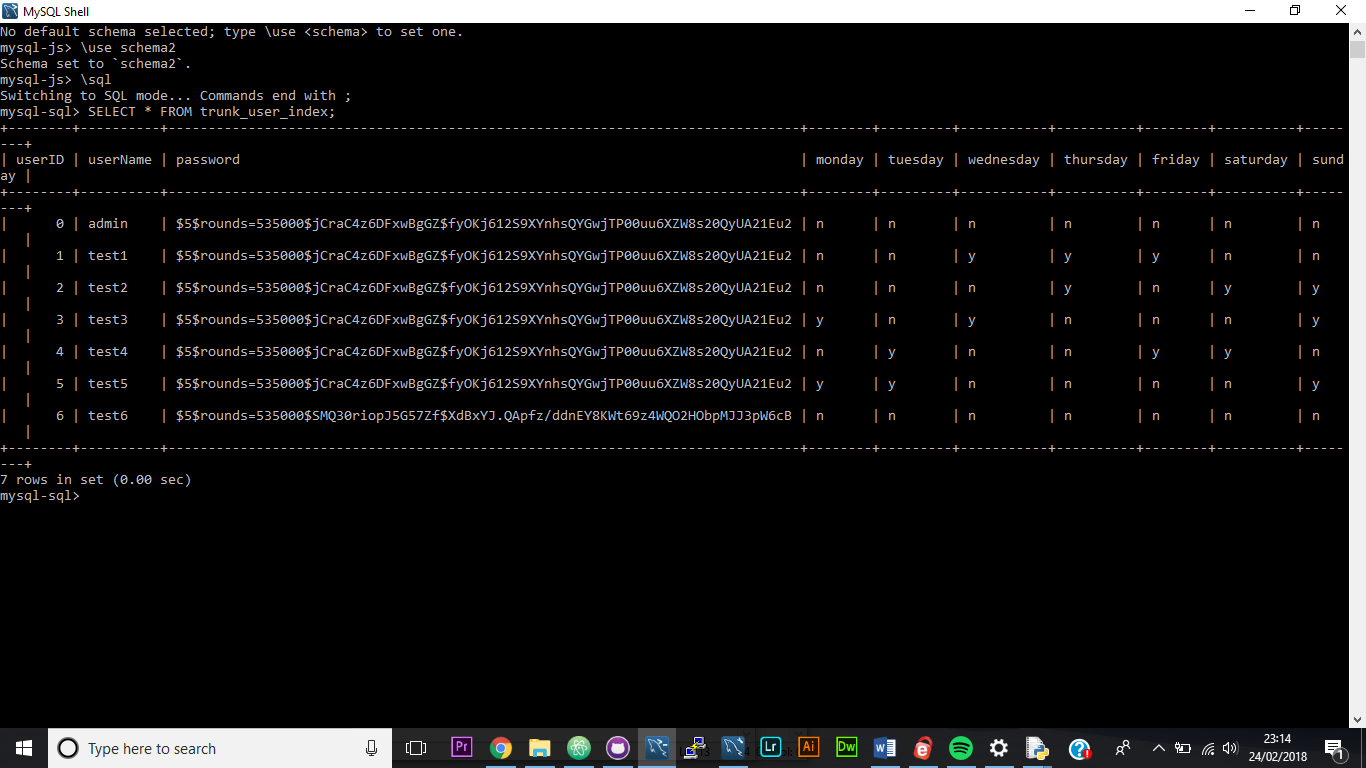
### The ‘trunk\_user\_index’:



This script creates a table named ‘trunk\_user\_index’, in which the primary key column is an automatically incrementing integer with up to 11 characters named ‘userID’, which has to be a value (not null). The next column is a VARCHAR field (can contain various types of character) which can contain up to 45 characters but can again not be null, named ‘userName’. The third column is another VARCHAR field, this one called ‘password’, that can contain up to 100 characters in order to allow it to hold a SHA256 hash of the user’s password. The next seven columns are one-character VARCHARS with the default ‘n’ that correspond to the seven days of the week. If a user has a shift on a given day than the value of the column will be changed to ‘y’.

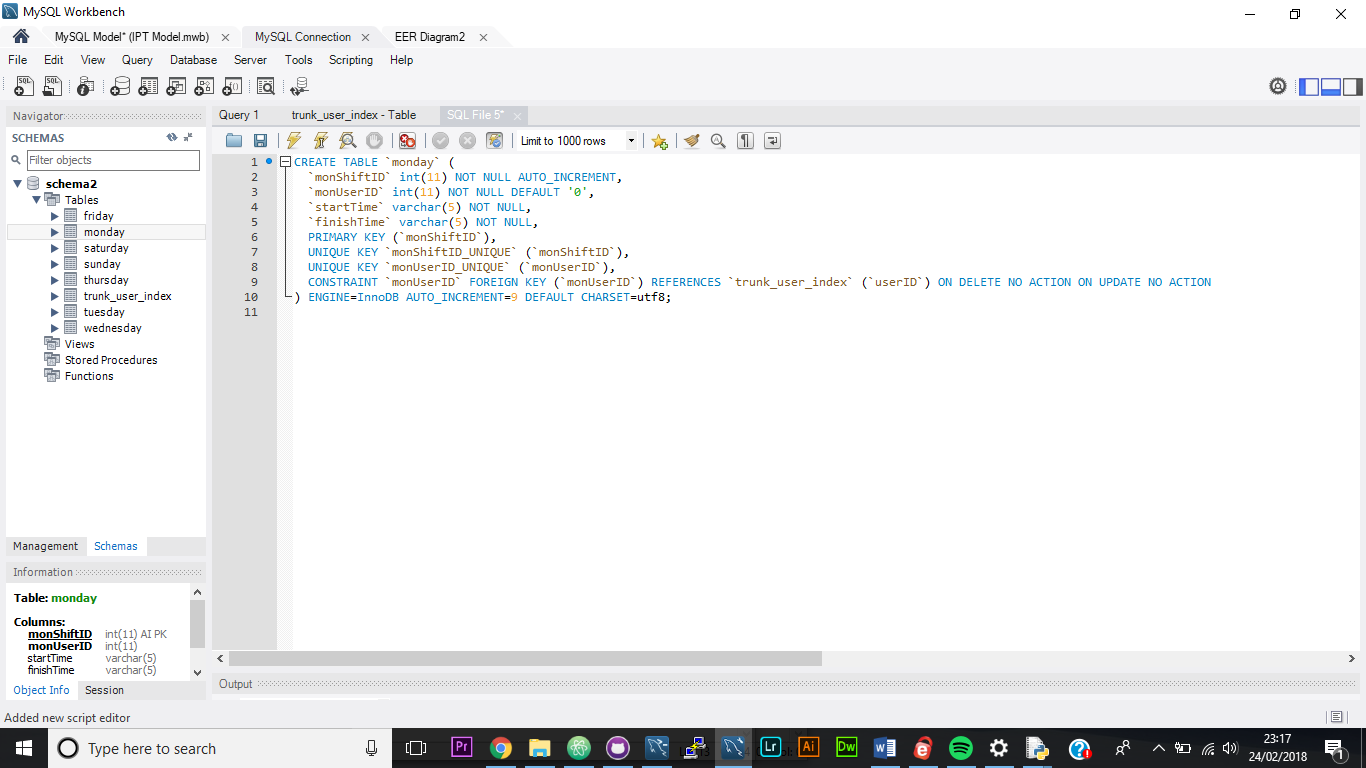
To insert into the database ‘with MySQL Shell’, the following command is used:

INSERT INTO trunk\_user\_index(userName, password, monday, tuesday, wednesday, thursday, friday, saturday, sunday) VALUES(“test6”, “$5$rounds=535000$SMQ30riopJ5G57Zf$XdBxYJ.QApfz/ddnEY8KWt69z4WQO2HObpMJJ3pW6cB”, “n” , “n” , “n” , “n” , “n” , “n” , “n”);

This then (when repeated) produces a database that looks like this: 

### The shift tables:

The shift tables are the tables that contain the shift information for each day. The script to create them looks like this:

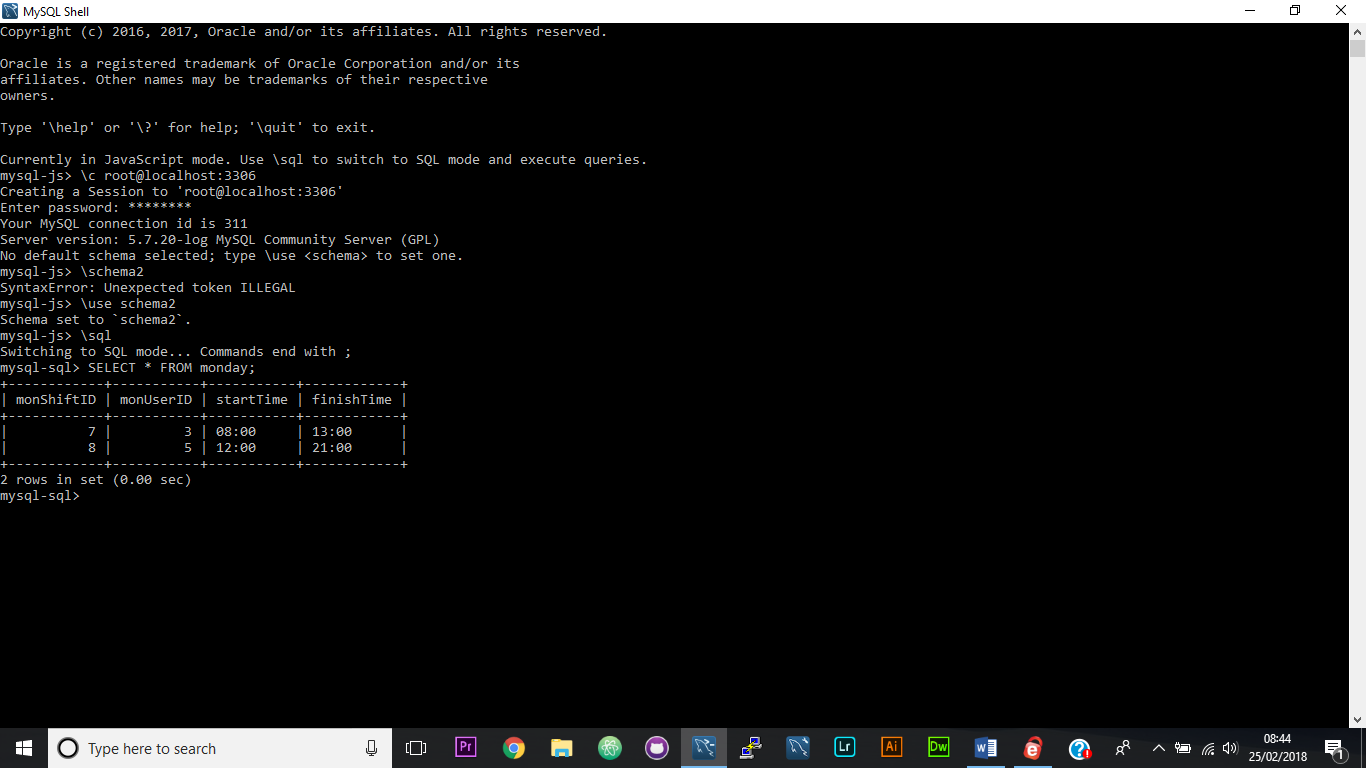


This script creates a table with four columns, the first of which is an auto incrementing INT field called ‘dayShiftID ‘, which can contain up to 11 characters. The second column is another 11-character INT field, this time named ‘dayUserID’. This is what acts as the foreign key connecting the shift data table to the ‘trunk\_user\_index’. This value is unique in the table and cannot be null. The final two columns store two 5-character VARCHAR values, which correspond to the start and finish time of the user’s shift in the form ‘hh:mm’.

The insert script to this table looks like this:

INSERT INTO monday(monUserID, startTime, finishTime) VALUES (3, “08:00”, “13:00);

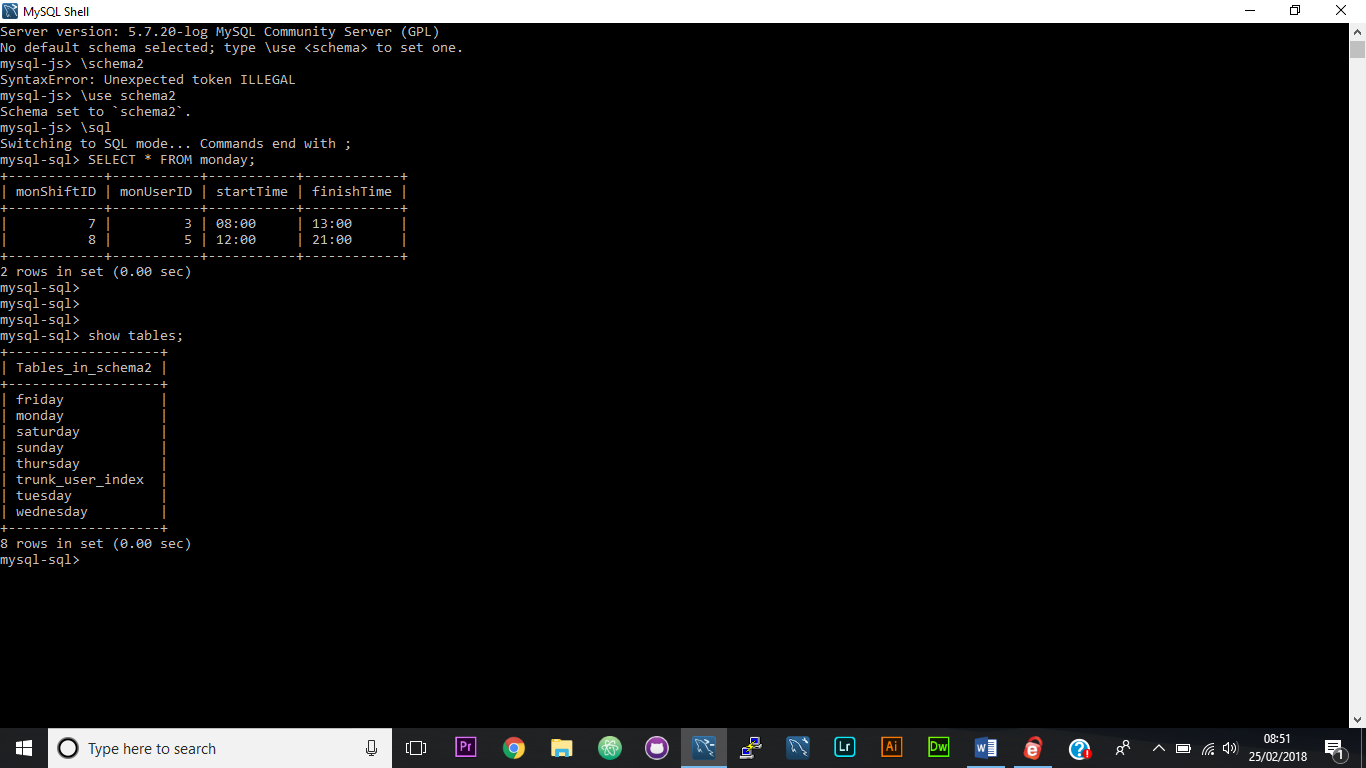
When repeated multiple times, this creates a table that looks like this:



I repeated the create table script seven times with the table name and two ID variable names changed to correspond with the days of the week. I then tested that the tables had in fact been created with the command:

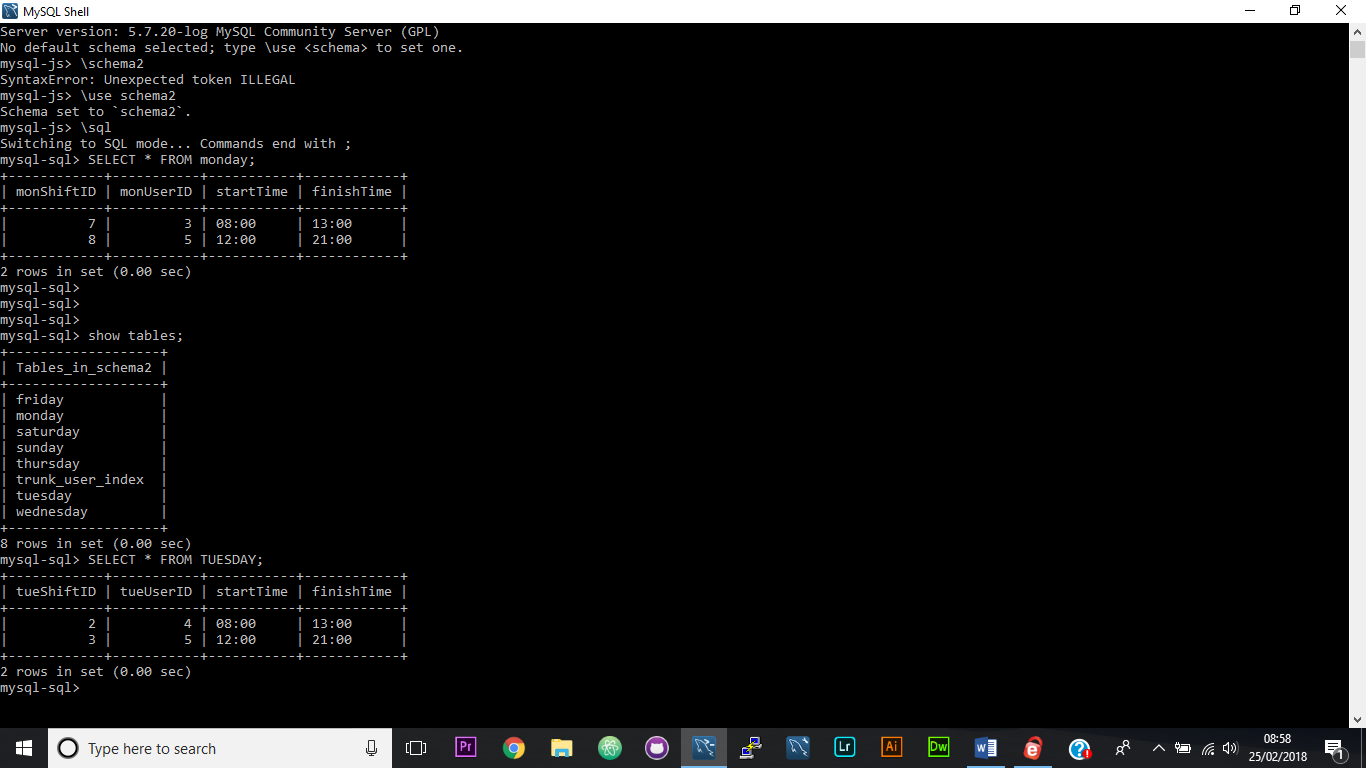
Show tables;

Which resulted in the following list of tables being displayed:

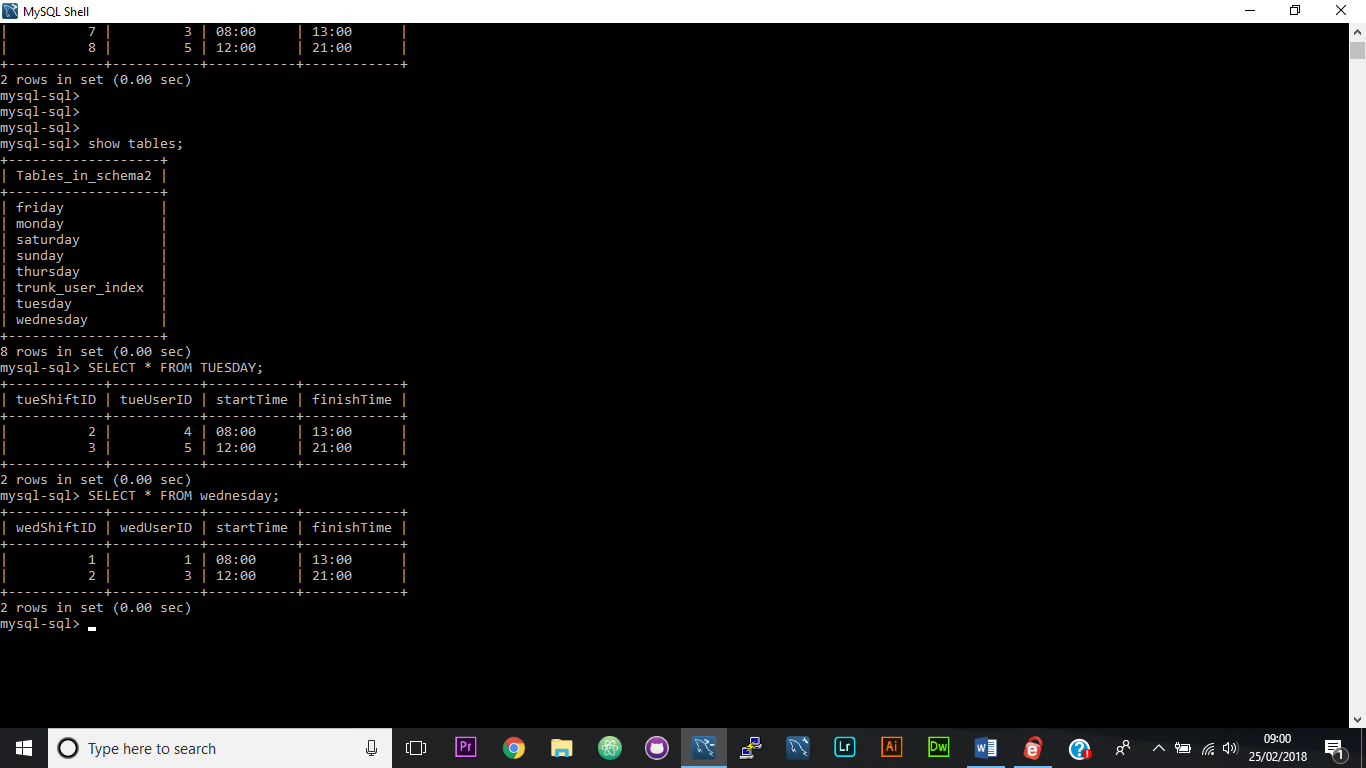


Which verifies that I have created eight tables with the correct names. I have already verified that I created the tables ‘trunk\_user\_index’ and ‘monday’ earlier when I explained how to create the two classes of table. Next, I will clarify that all of the other tables contain the correct data.

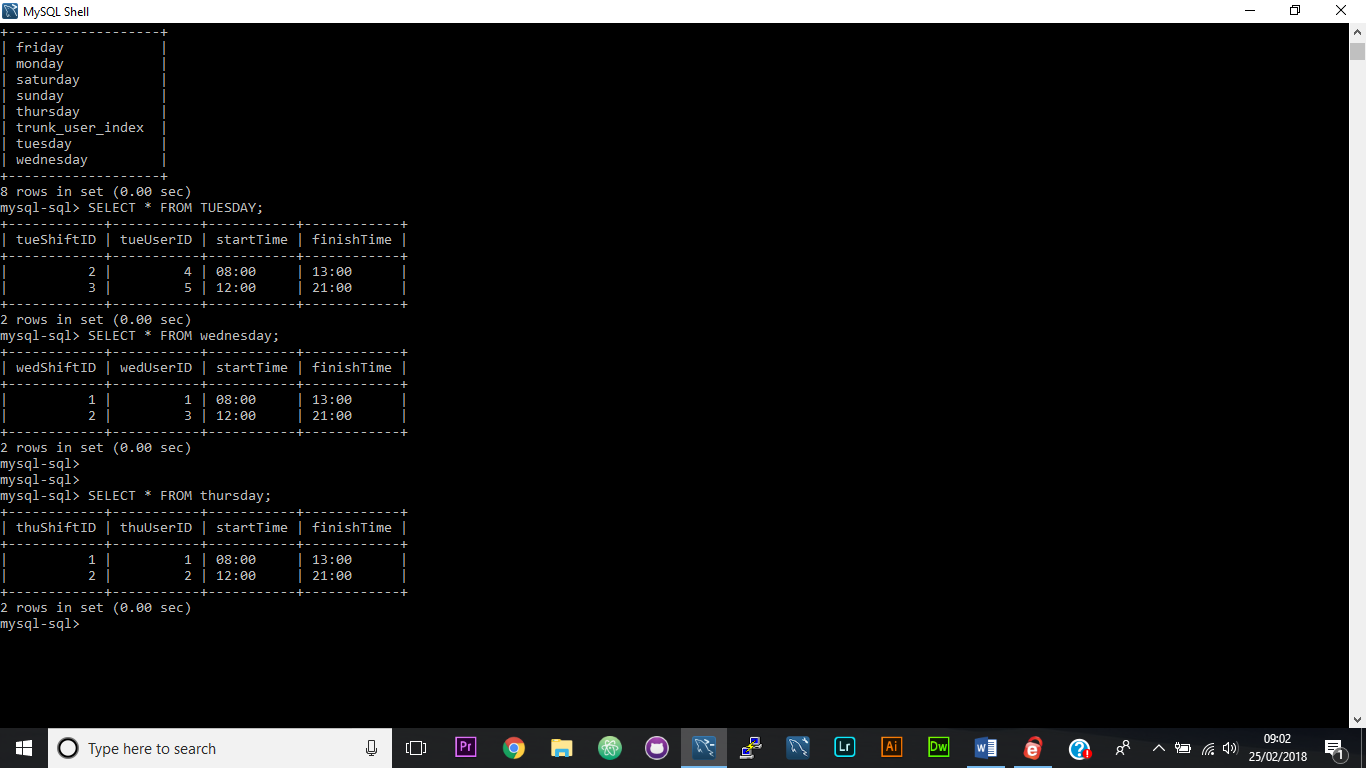
Tuesday:



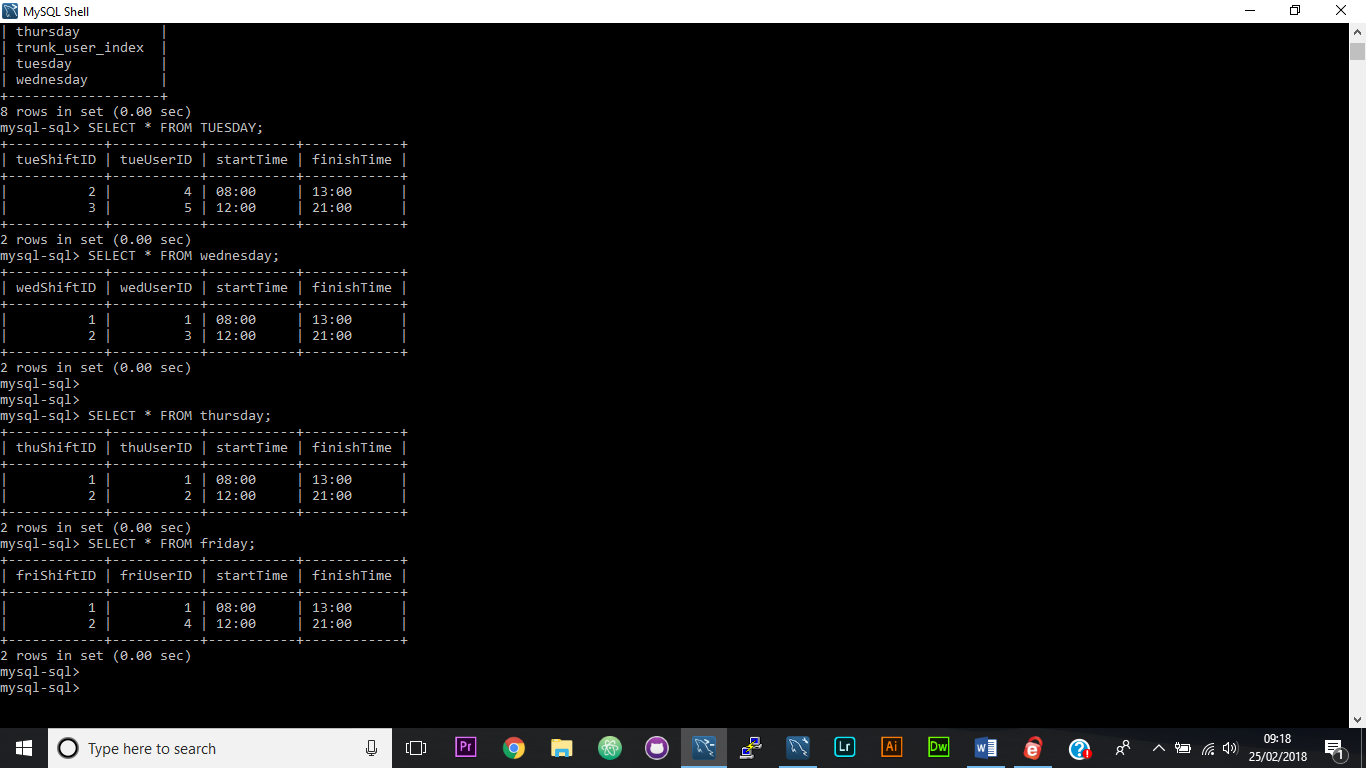
Wednesday:



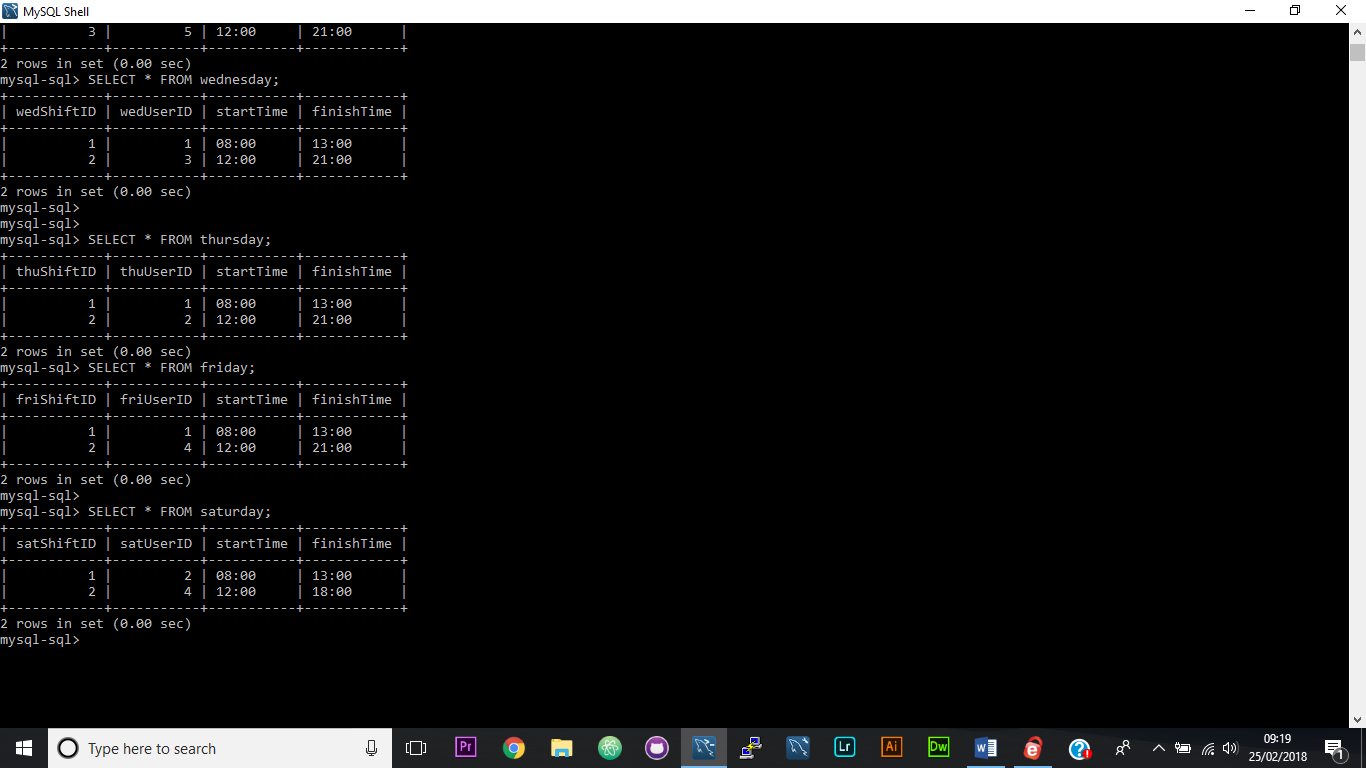
Thursday:



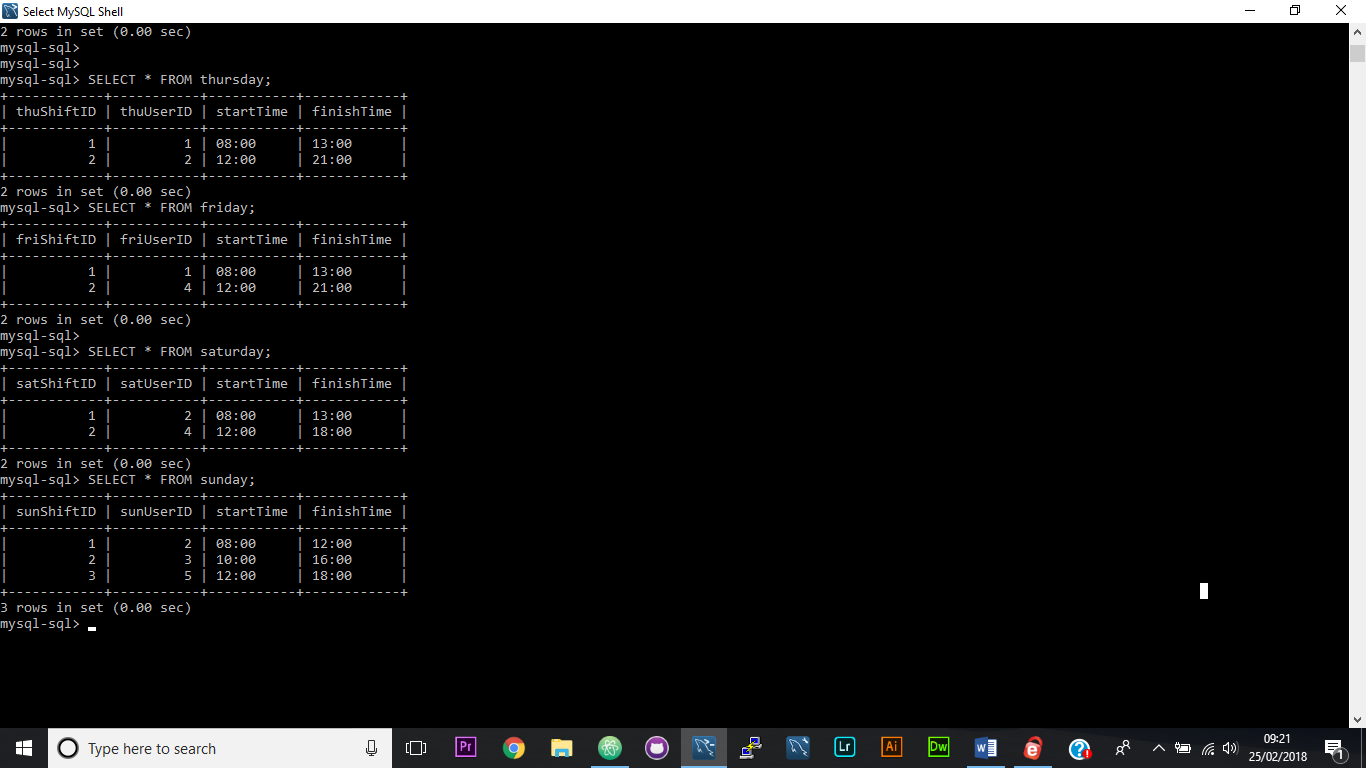
Friday:



Saturday:



Sunday:



Now that the SQL database is set up and populated, I can start building the Flask web app portion of the project.

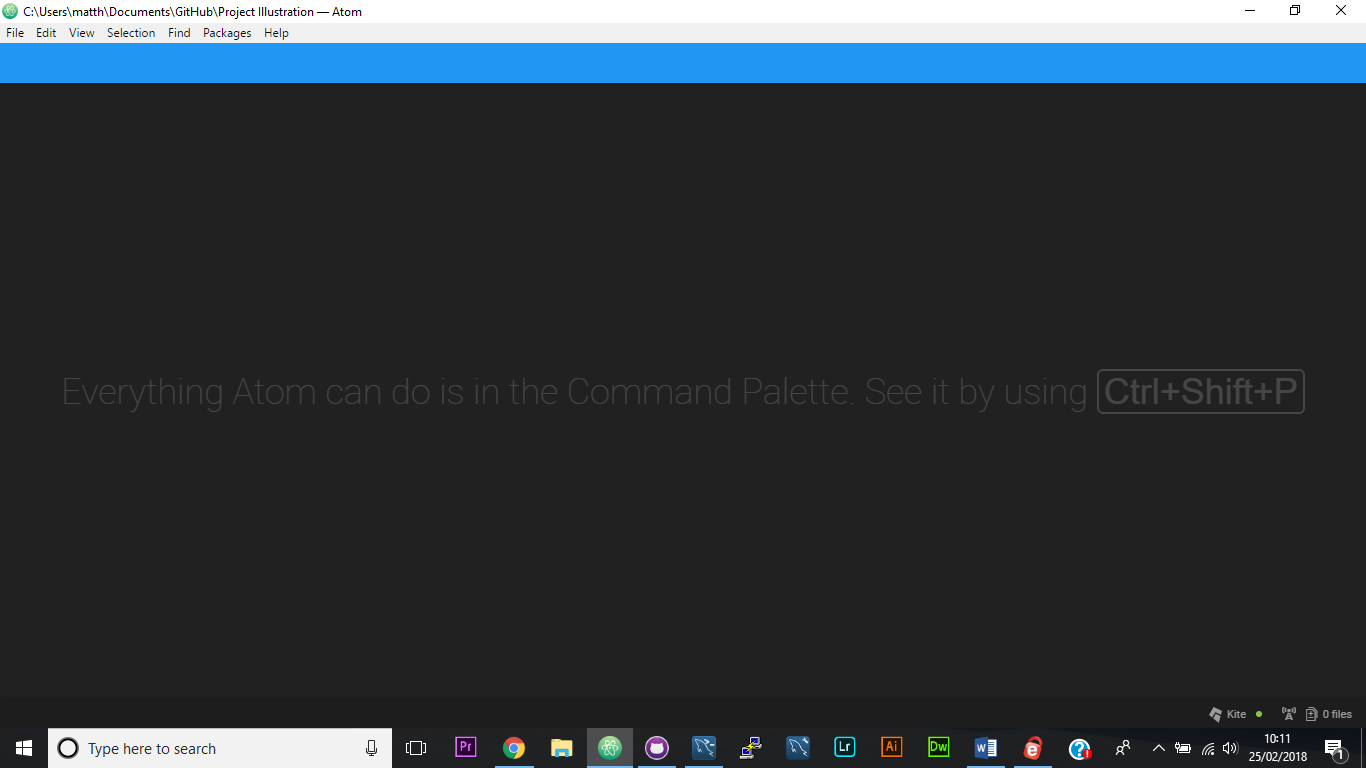
## Step 2) setting up the IDE

The next step of this project is setting up my development environment. First, I created a GitHub repository, which will enable me to both keep track of versions of my program and to store the program on the cloud. This will reduce the likelihood of me losing my project greatly.

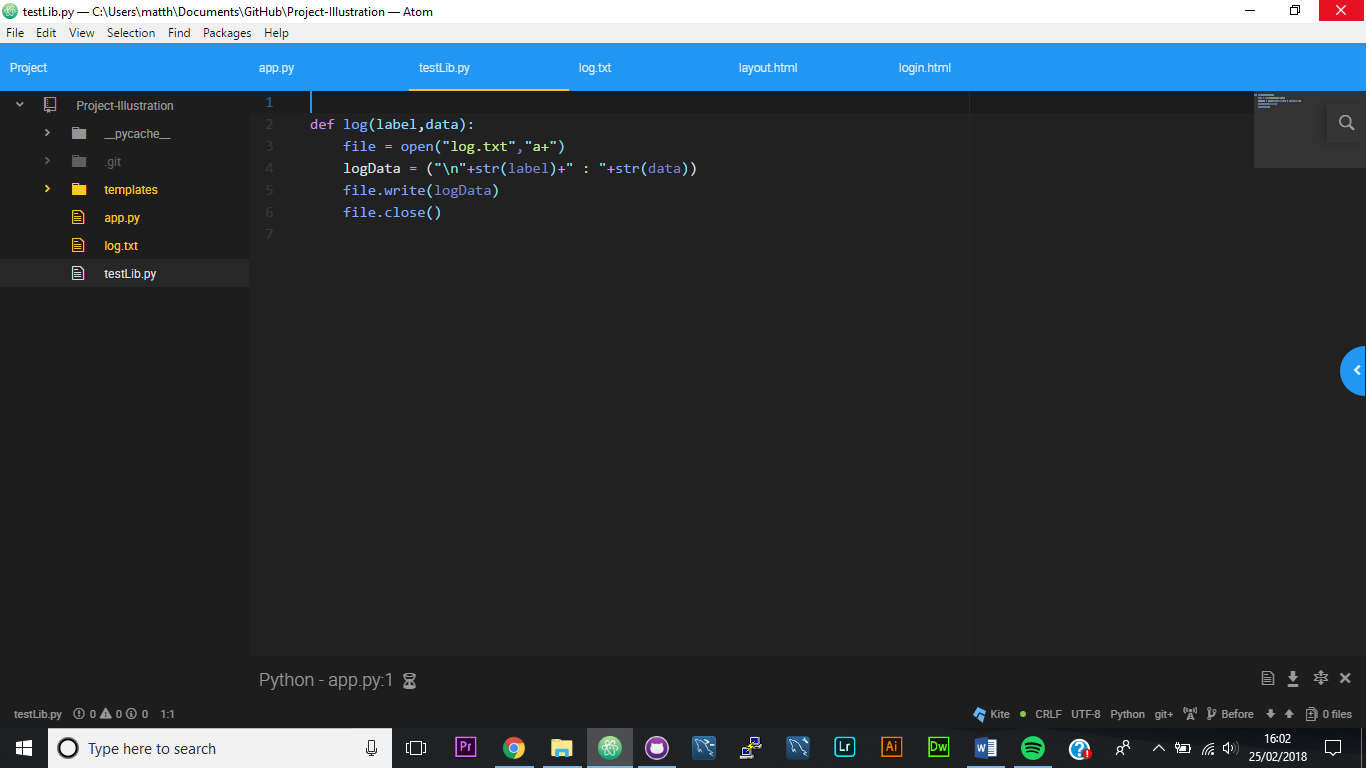
Next, I chose an IDE. My choice was Atom, an open source multi-language IDE which you can add packages to. These packages add extra functionality to the IDE, for example I used ‘kite’ to auto complete variables that had shown up in the code before, ‘minimap’ to show a visual representation of all of my code on a minimap on the right, and ‘script’ to run code in atom as stock Atom does not include an interpreter. As well as this I used a host of other packages and themes to make the coding experience more comfortable.

Atom also has integrated GitHub support to make version control easier as well.

When I had finished setting it up, my IDE looked like this:



Before I started writing the flask app I also created a python test library that logs data to a file as I couldn’t find how to log things to the console.

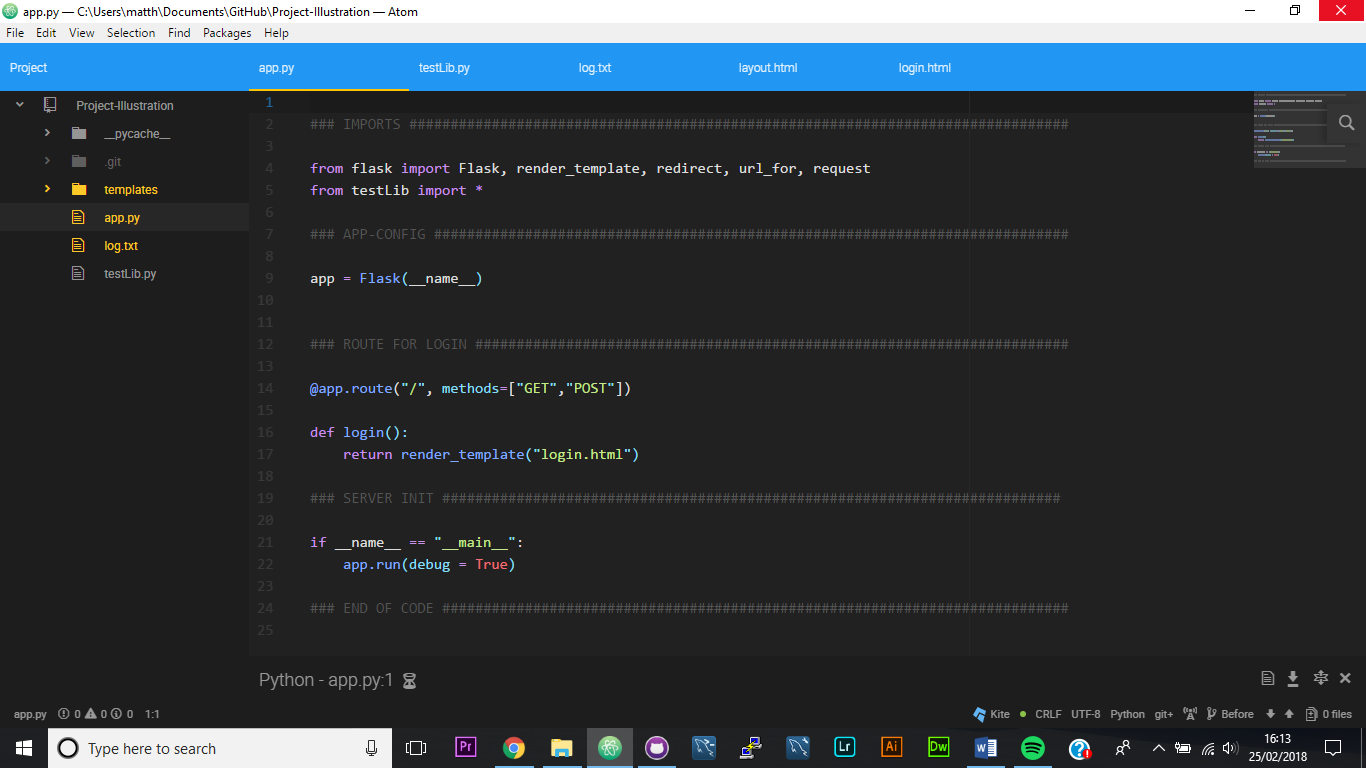


## Step 3) Creating a flask app

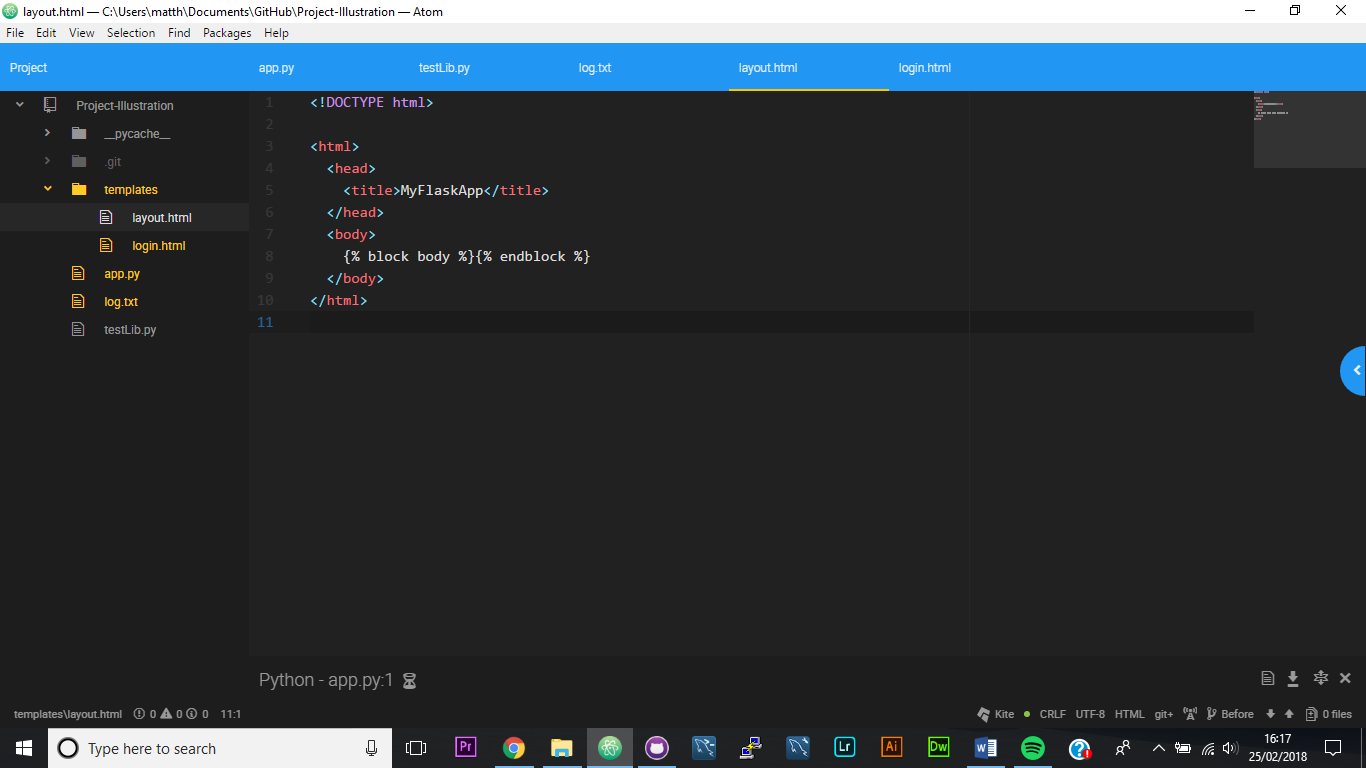
First, I imported all of the classes and functions I was going to use into ‘app.py’.

Next, I created an instance of the class ‘Flask’ called app to act as the WSGI (the interface between web servers and the web app). I used the variable ‘\_\_name\_\_’ as the parameter, which returns the string ‘\_\_main\_\_’ when the code is run.

Next, I mapped the app route ‘/’ to run the function ‘login’, so when the URL is ‘localhost:5000’ the program runs the function defined as ‘login’ and loads the page ‘login.html’.



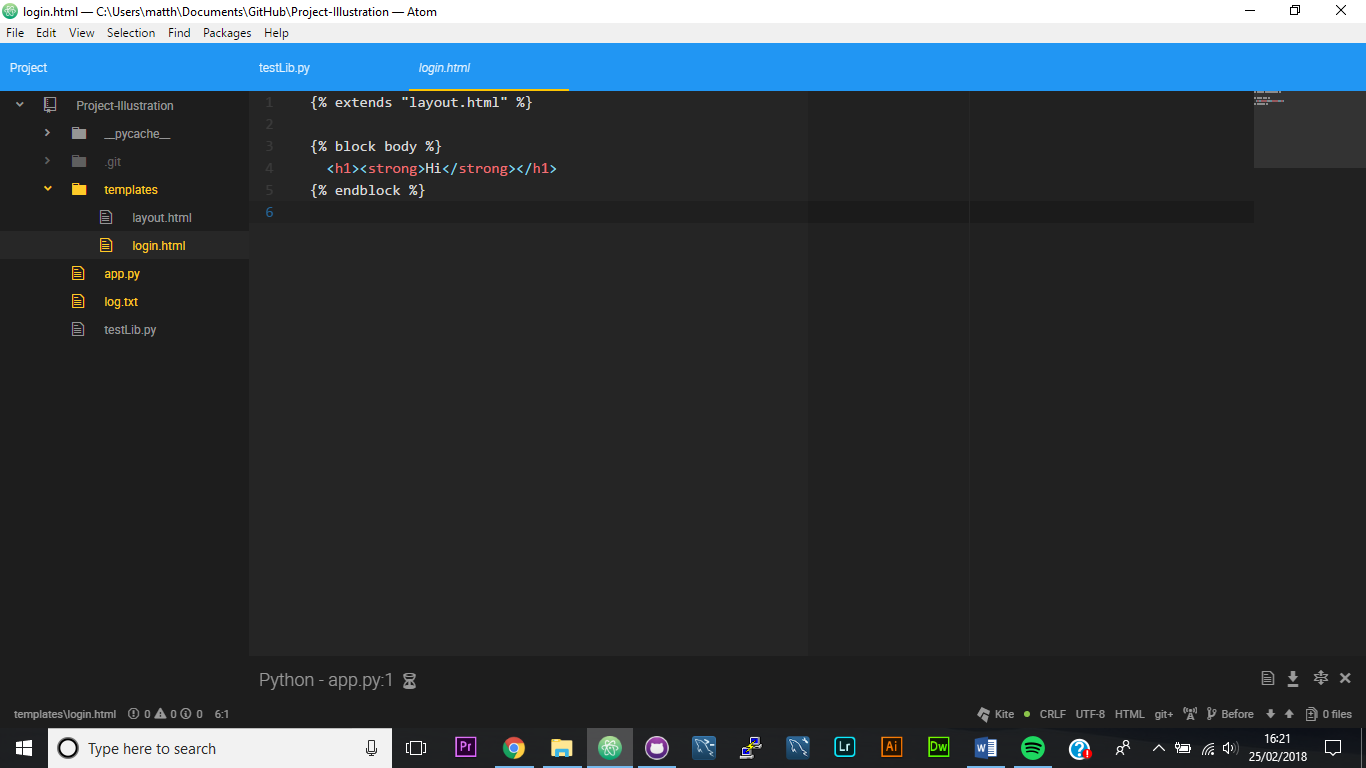
Next, I made the page layout.html which includes all the html head information. It also contains the two tags ‘{% block body %}’ and ‘{% endblock %}’ in between which all of the html for the pages that extend ‘layout.html will go.



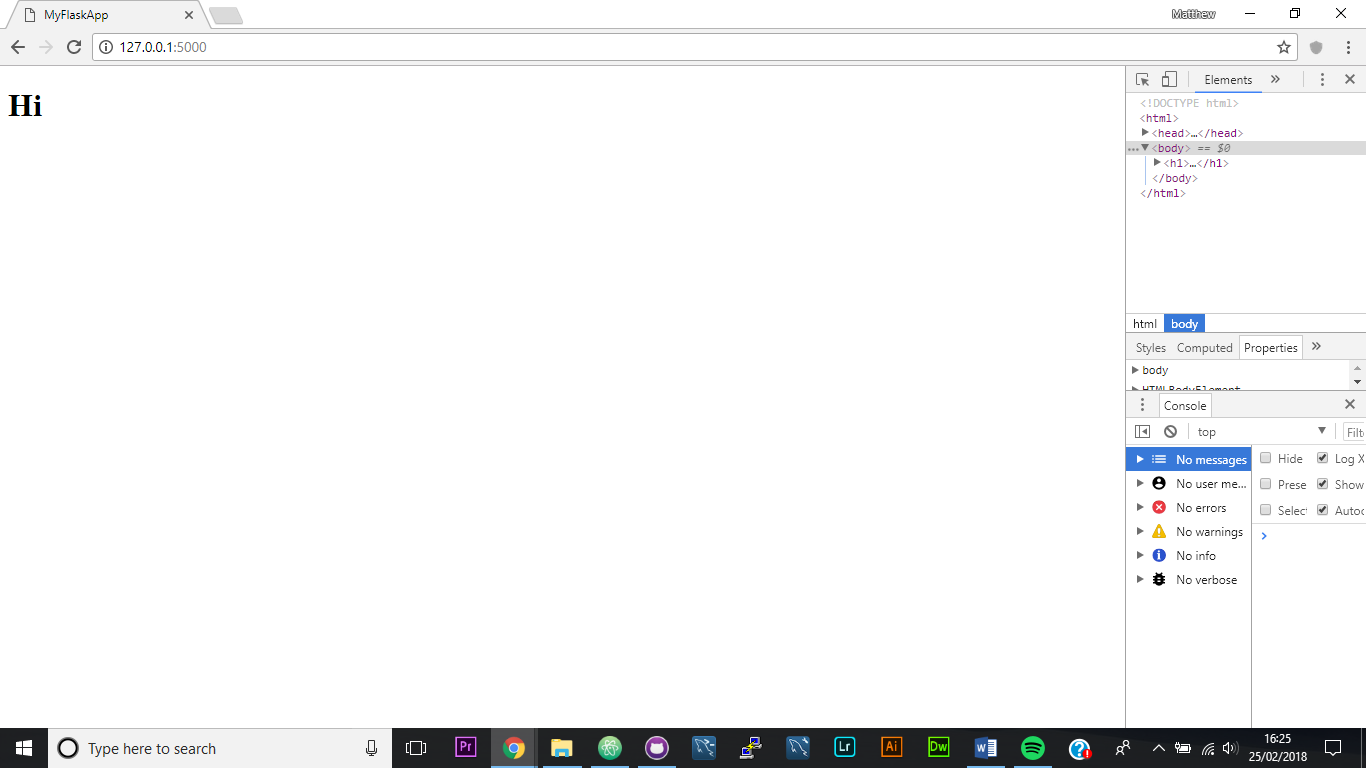
Finally, I wrote a page called ‘login.html’, which has the tag

{% extends "layout.html" %}

As well as a set of ‘block body’ tags, in-between which the html for home is written.



To make sure that the code is correct I ran the python script and navigated to the URL ‘localhost:5000’. It produced the following response, proving that the code was correct.

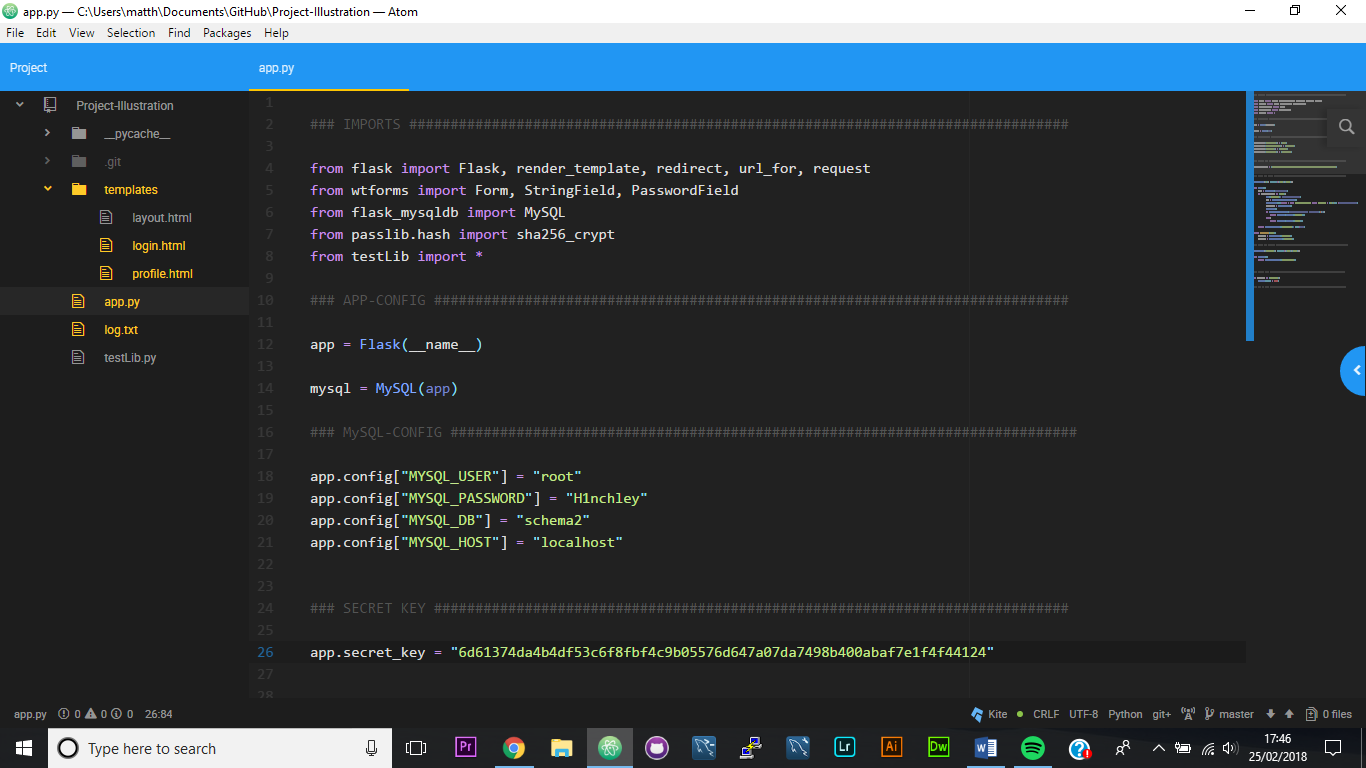


### Step 3 test table:

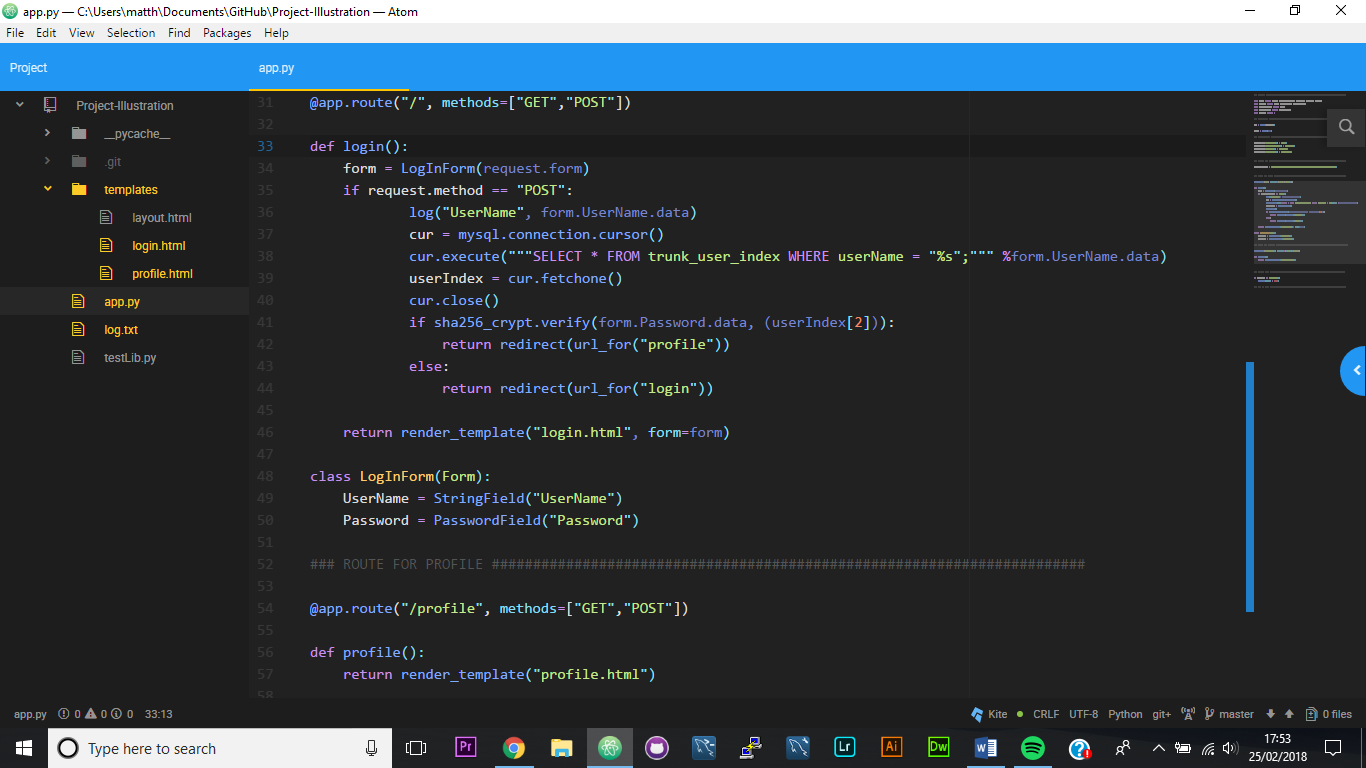
|  |  |  |
| --- | --- | --- |
| Test scenario | Expected response | Actual response |
| I navigated to ‘localhost:5000’ | I see ‘Hi’ in bold | I saw ‘Hi’ in bold |

## Step 4) adding login functionality

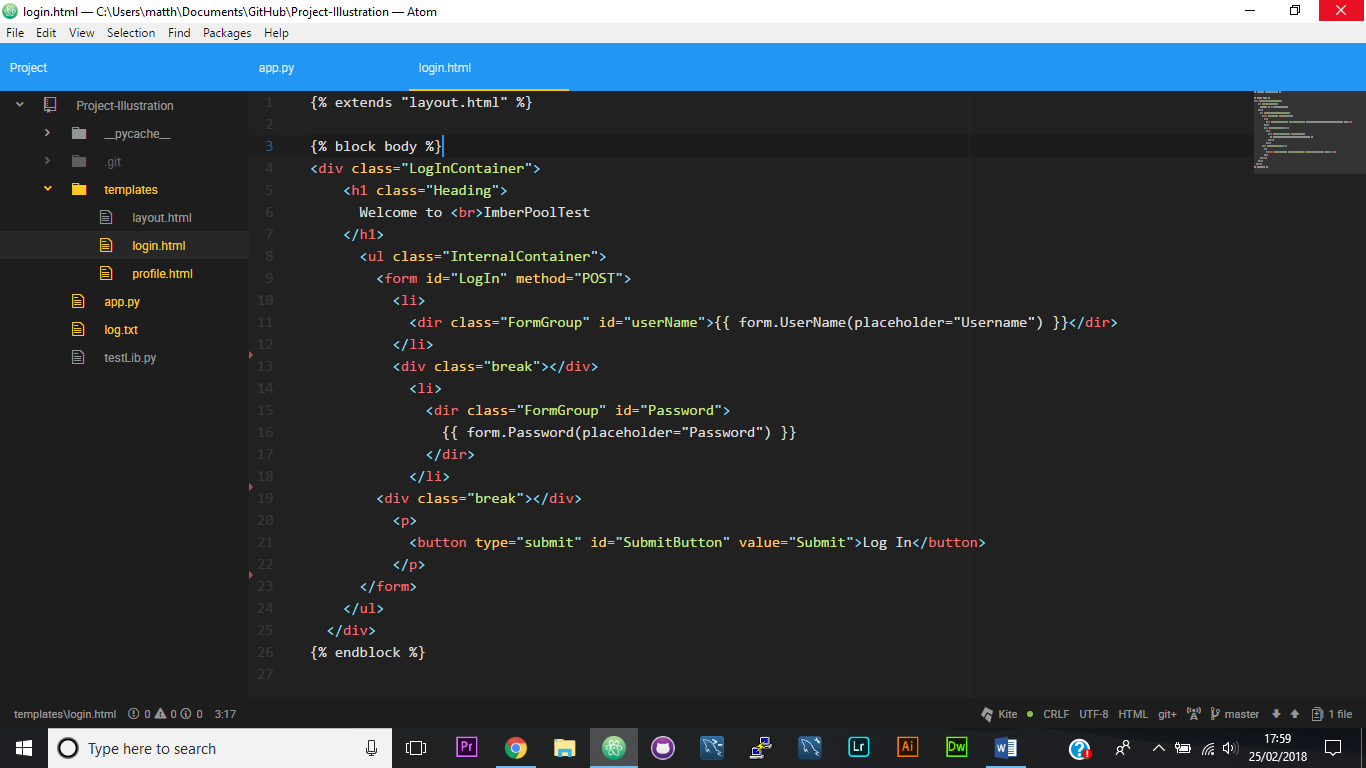
In order to add login functionality I first imported the ‘wtforms’, ‘passlib.hash’ and ‘flask\_mysqldb’ libraries. Next, I passed the variable app into the function ‘MySQL’ and set the returned value to the variable ‘mysql’. Next, I configured the sql connection variables so that the app connects to my MySQL database. I then set the secret key.



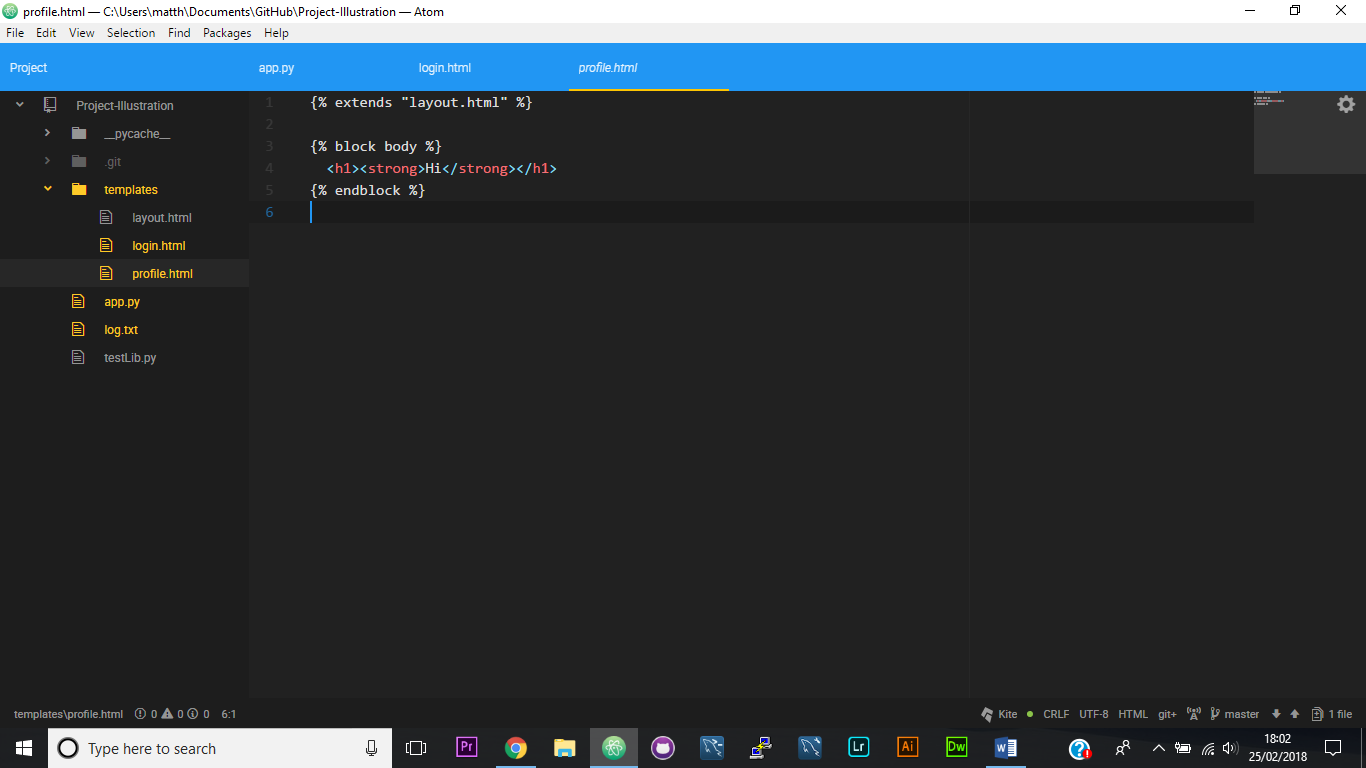
Next, I edited the function for login and created the form ‘Form’ which will take data from the inputs from the HTML and use them as values. When the button for login is pressed, login is used to select a row in the table ‘trunk\_user\_data’ and compared the value stored in the password column to a hash of the password input. If they are both hashes of the same value the function returns the URL for profile, if not it redirects you back to the login page. Later I will add form validation and error handling. Finally, I created an app route for the suffix ‘/profile’ which will render the template ‘profile.html’ when the code is run.



Next, I added to ‘login.html’, adding a heading, two text inputs that have placeholders in them to aid usability, and a button that when pressed submits the form.

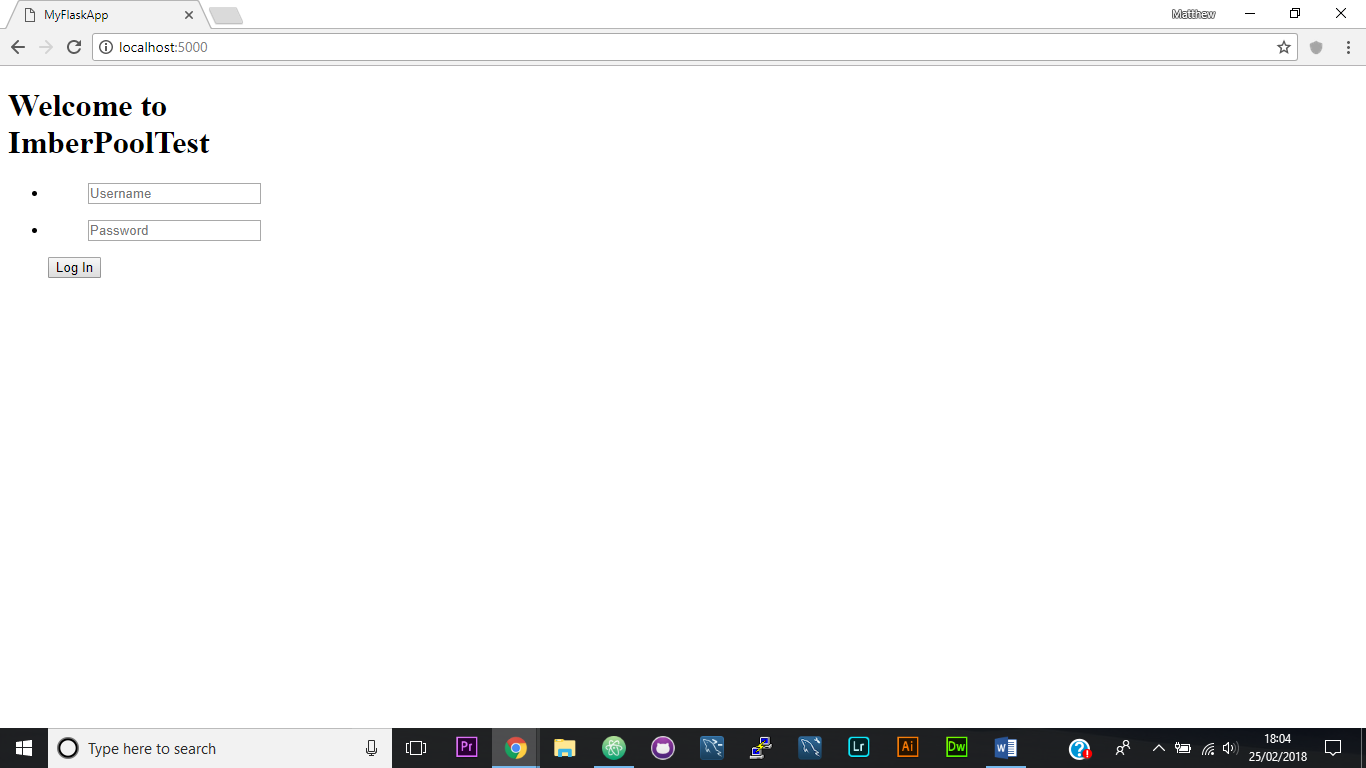


Finally, I made a file called ‘profile.css’ and pretty much just copied what used to be in ‘login.html’ into it so that I could test if the additions were working.

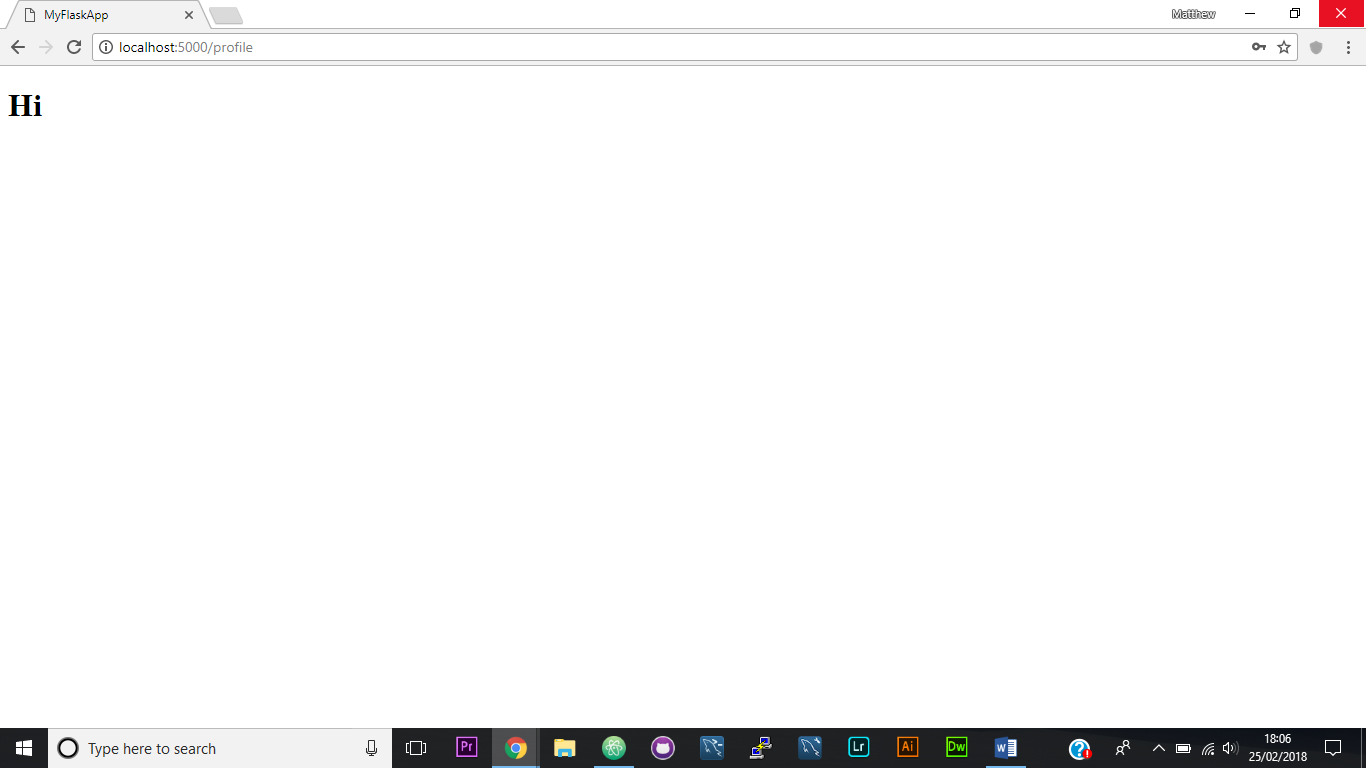


Then ran the program and tested the site

Login.html:



Profile.html:

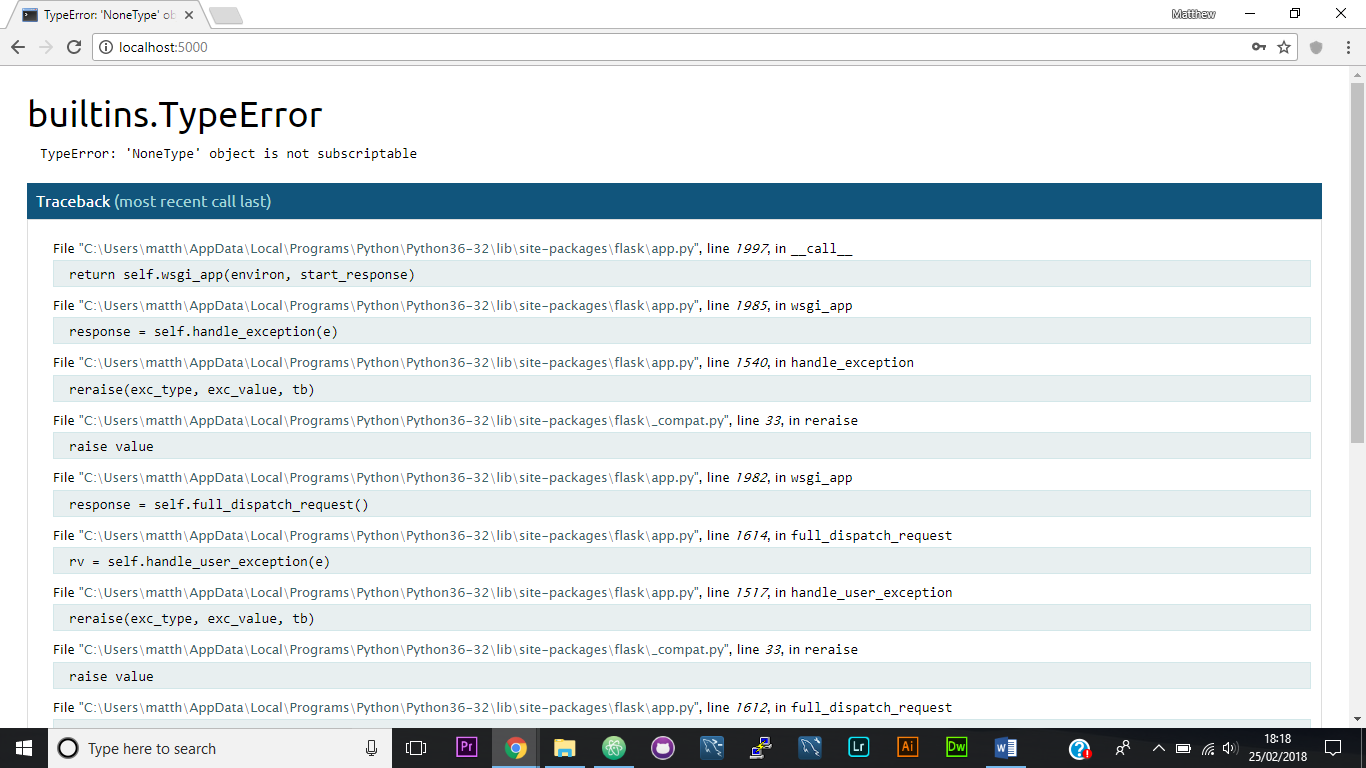


### Step 4 test table:

|  |  |  |
| --- | --- | --- |
| Test scenario | Expected response | Actual response |
| I navigated to ‘localhost:5000’ | See login page | I saw login page |
| I input correct values | Loads profile page | Loaded profile page |
| I input wrong username | Reloads login page | ERROR nonetype not subscriptable |
| I input wrong password | Reloads login page | Reloaded login page |

The test was not a success, there was an error in the code.

### Troubleshooting:



I think the issue was that if there is no userName value that is the same as what is inputted, there is no row fetched so the program throws an error.

To remedy this I will put a try statement before the database query and an except that returns the login page after it.

### Step 4 test table 2:

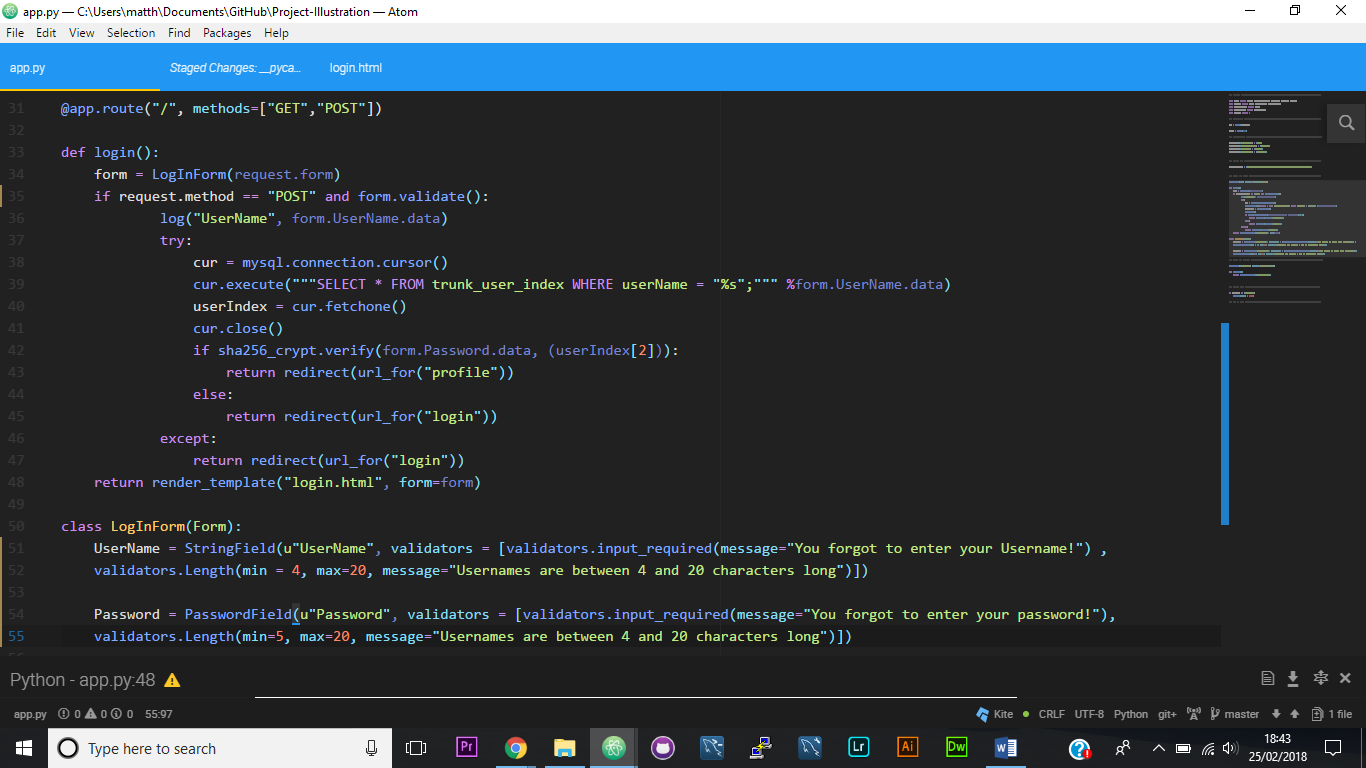
|  |  |  |
| --- | --- | --- |
| Test scenario | Expected response | Actual response |
| I inputted an incorrect username | Reloads login page | Reloaded login page |

The code is fixed, and the test was a success.

## Step 5) adding form validation

The first thing I did for this step was import ‘validators’ from wtforms, as well as ‘flash’ from flask to allow me to return messages that will tell users what they have done wrong if the page reloads.

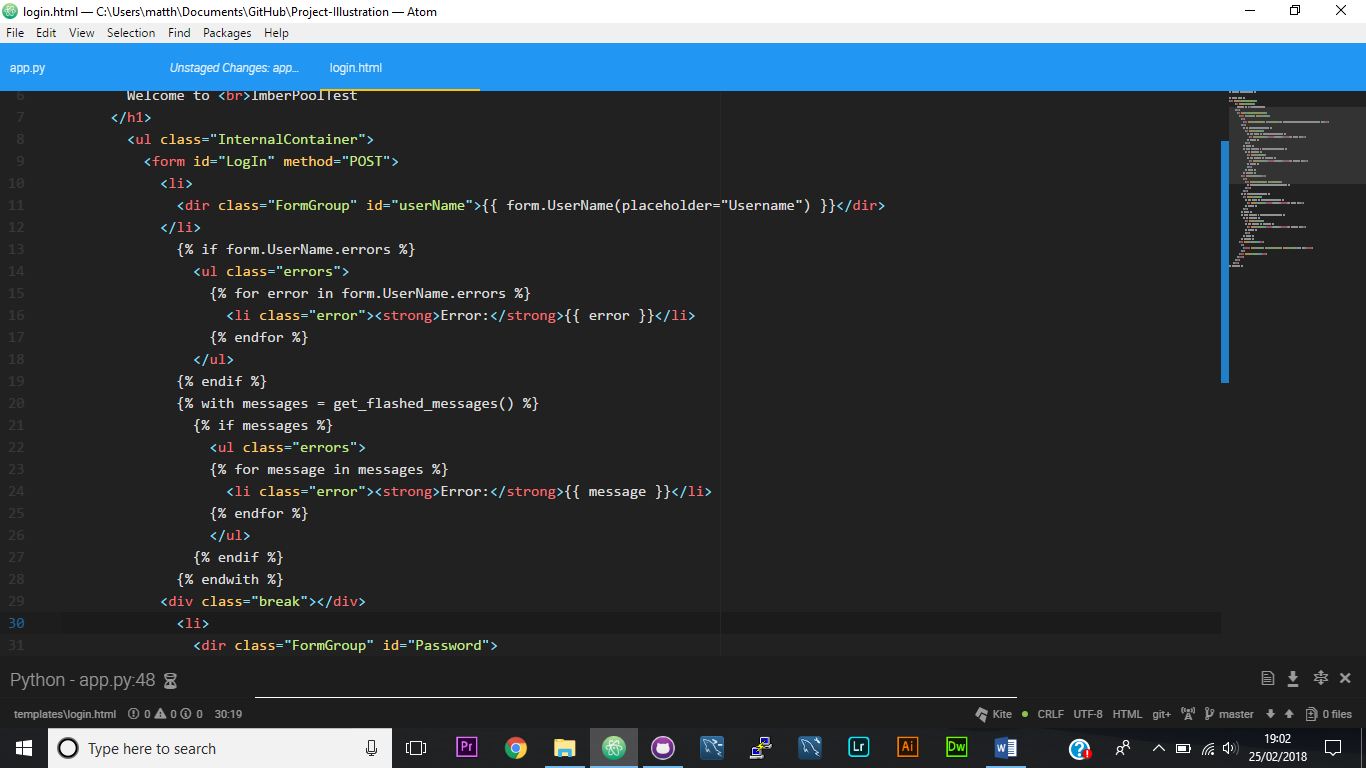
Next, I added a ‘form.validate()’ condition to the if statement that attempts to log the user in. furthermore I added validators to both of the fields that detect if the form has an input and if that input is between 4 and 20 characters. If either of those conditions is not met, ‘form.validate’ returns False and a message relevant to the error is shown.



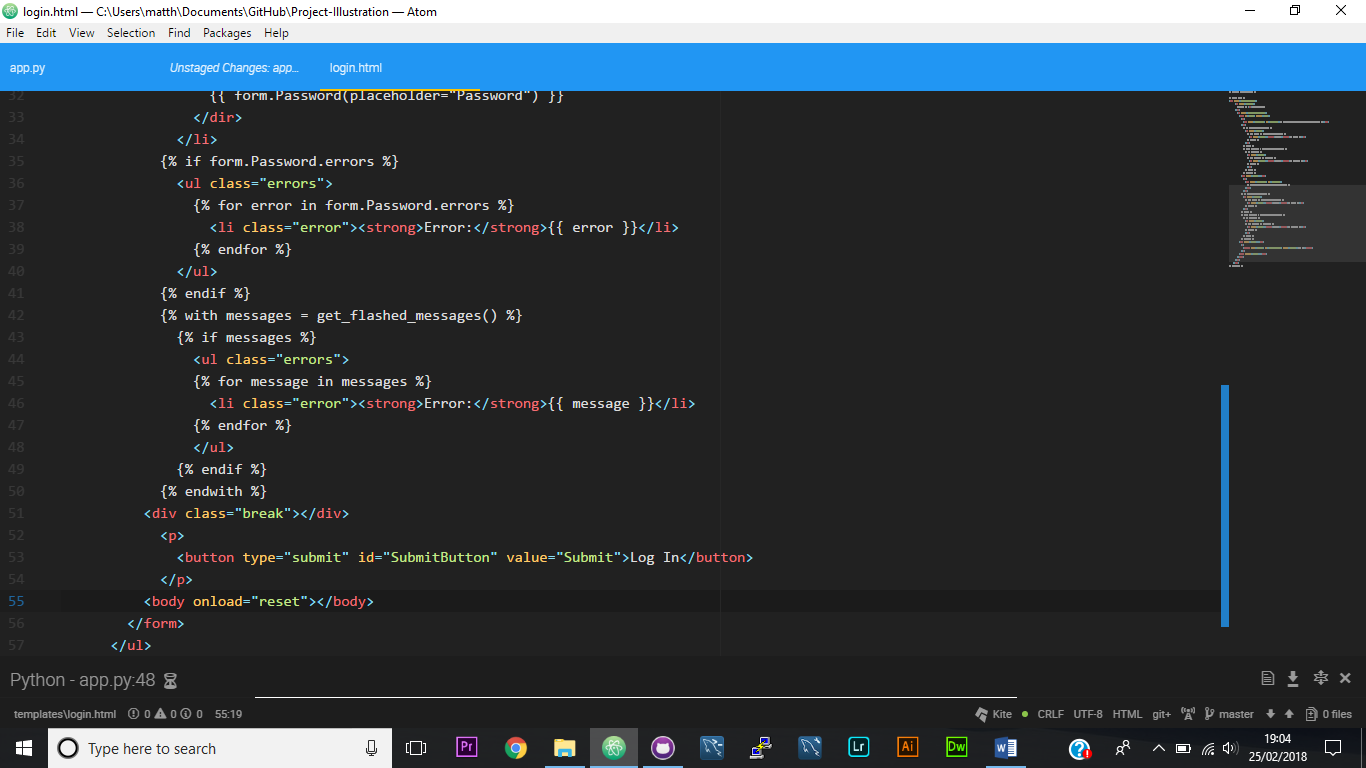
The next step was including the messages returned on the login page. For the wtforms messages I used jinja to write a for loop in an if statement that lists the the messages that are relevant below the relevant field.

Next, I used a for loop, nested within an if or statement, nested within a with statement to get and display the flashed messages below both fields if an error takes place.

For Username:

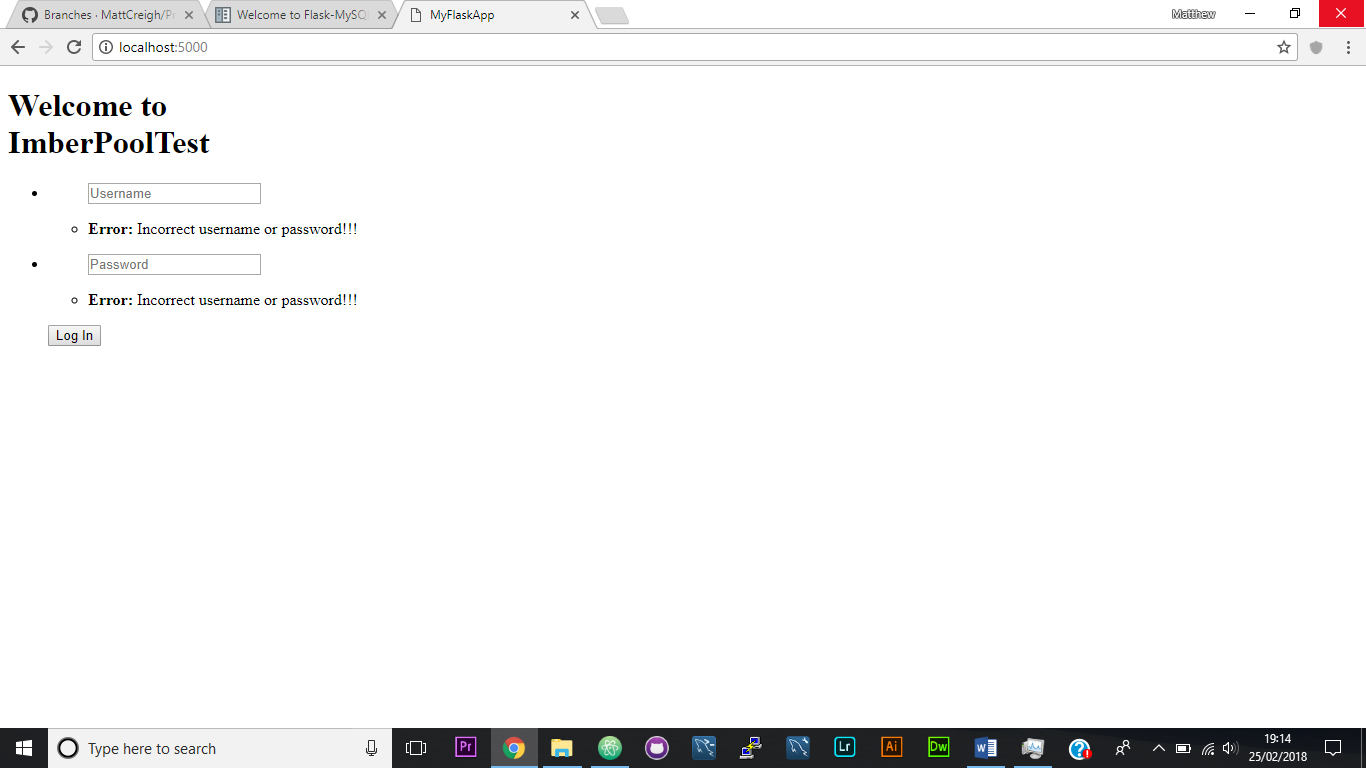


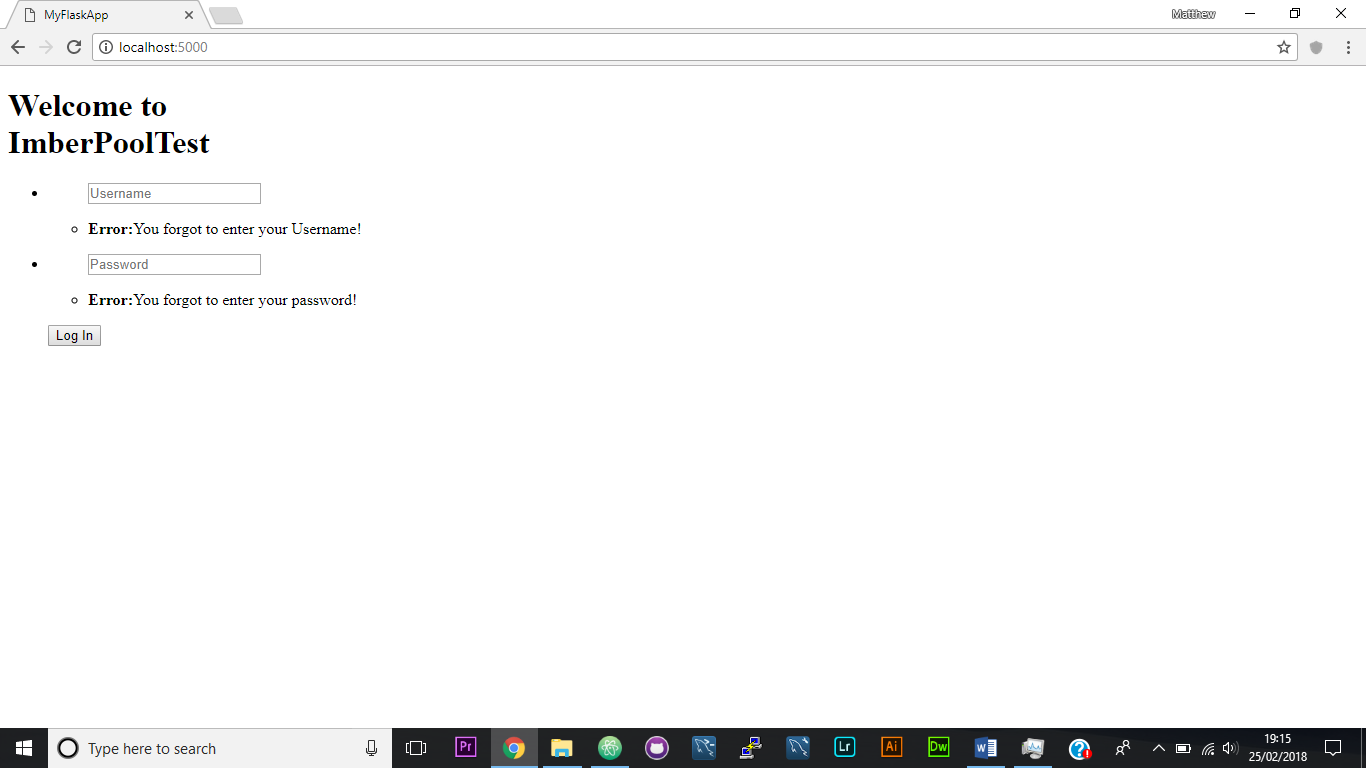
For Password:

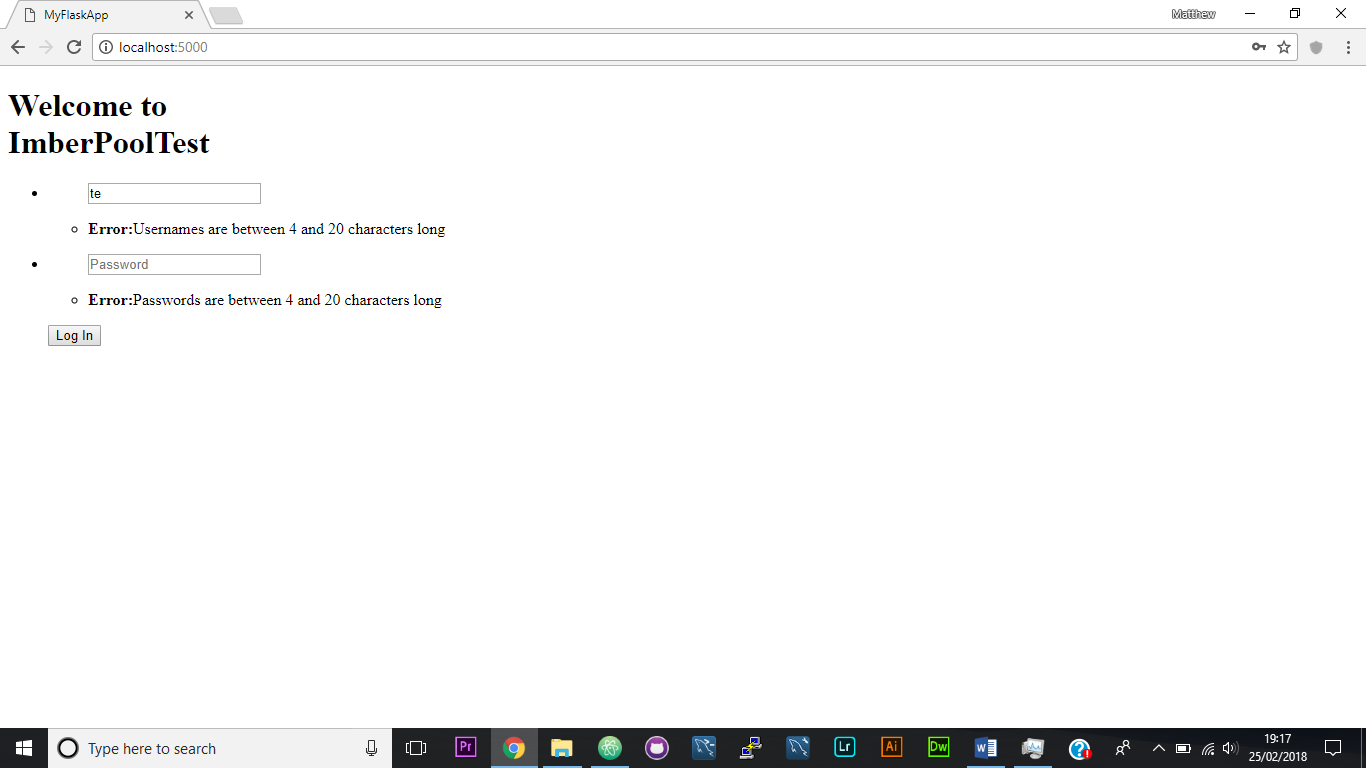


### Step 5 test table:

|  |  |  |
| --- | --- | --- |
| Test scenario | Expected response | Actual response |
| I navigated to ‘localhost:5000’ | Loads login page with no messages | Loaded login page with no messages |
| Input correct username and password | Loads profile page | Loaded profile page |
| Input no username or password | Messages under fields telling me they require input | Displayed messages |
| Input username and password that are too short | Messages under both fields telling me they’re too short | Messages displayed |
| Input username that doesn’t exist | Error wrong username or password displayed after reloading | Error displayed after reloading |
| Input password that doesn’t exist | Error wrong username or password displayed after reloading | Error displayed after reloading |



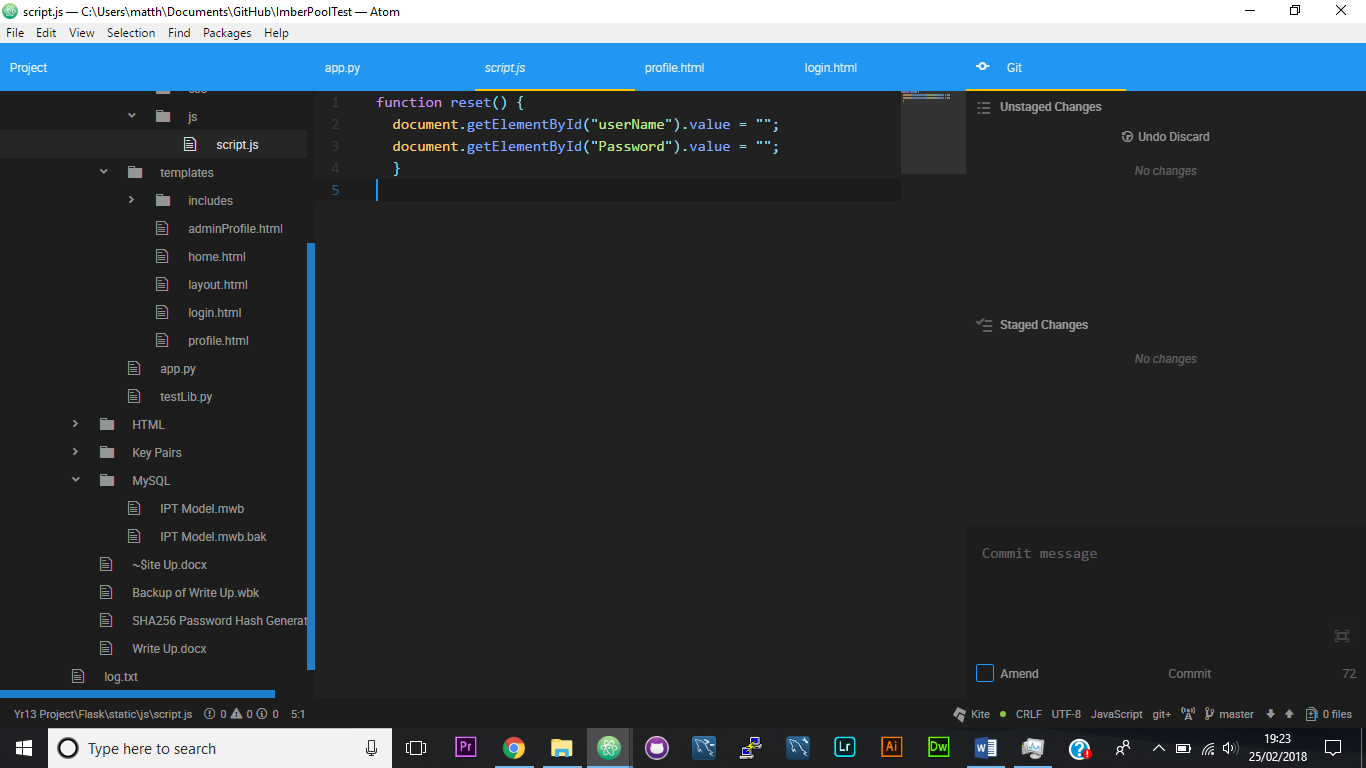




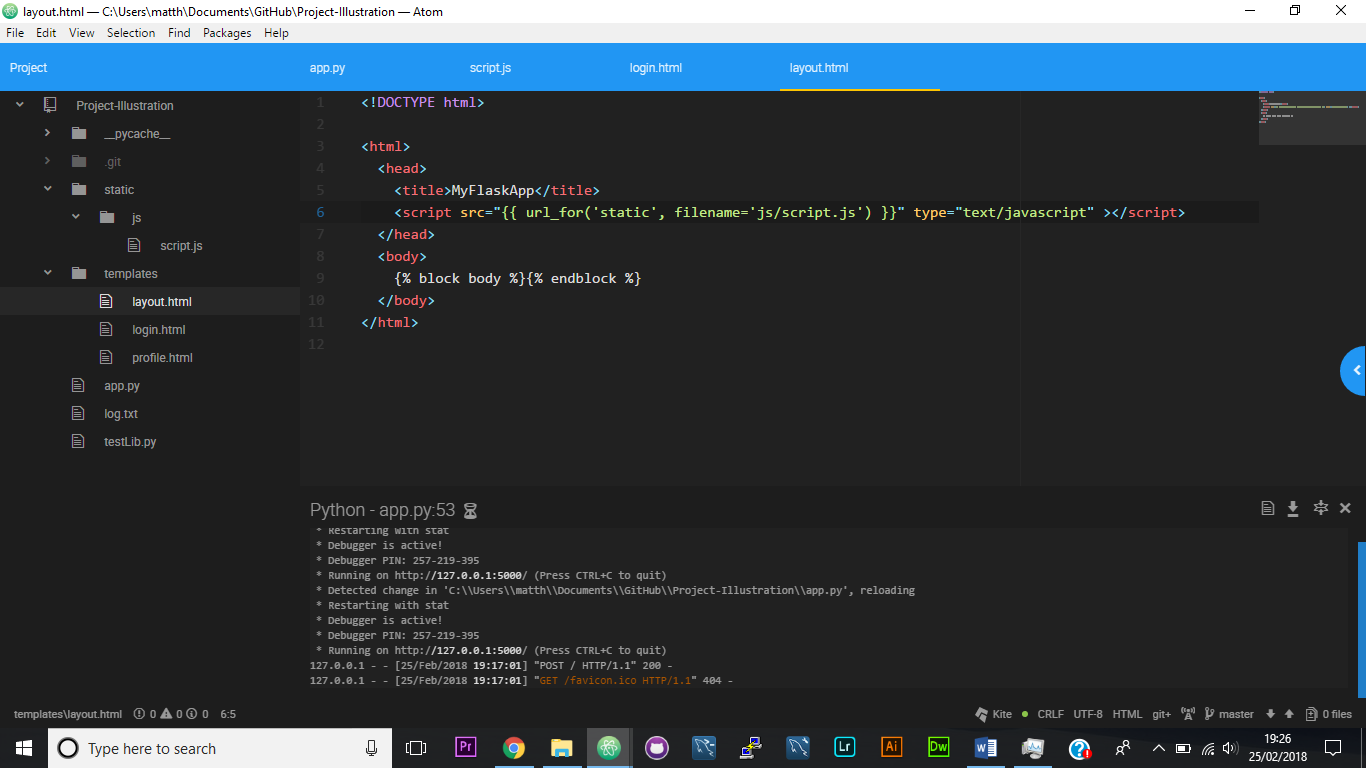
Testing was a success!

## Step 6) removing the form data on reload

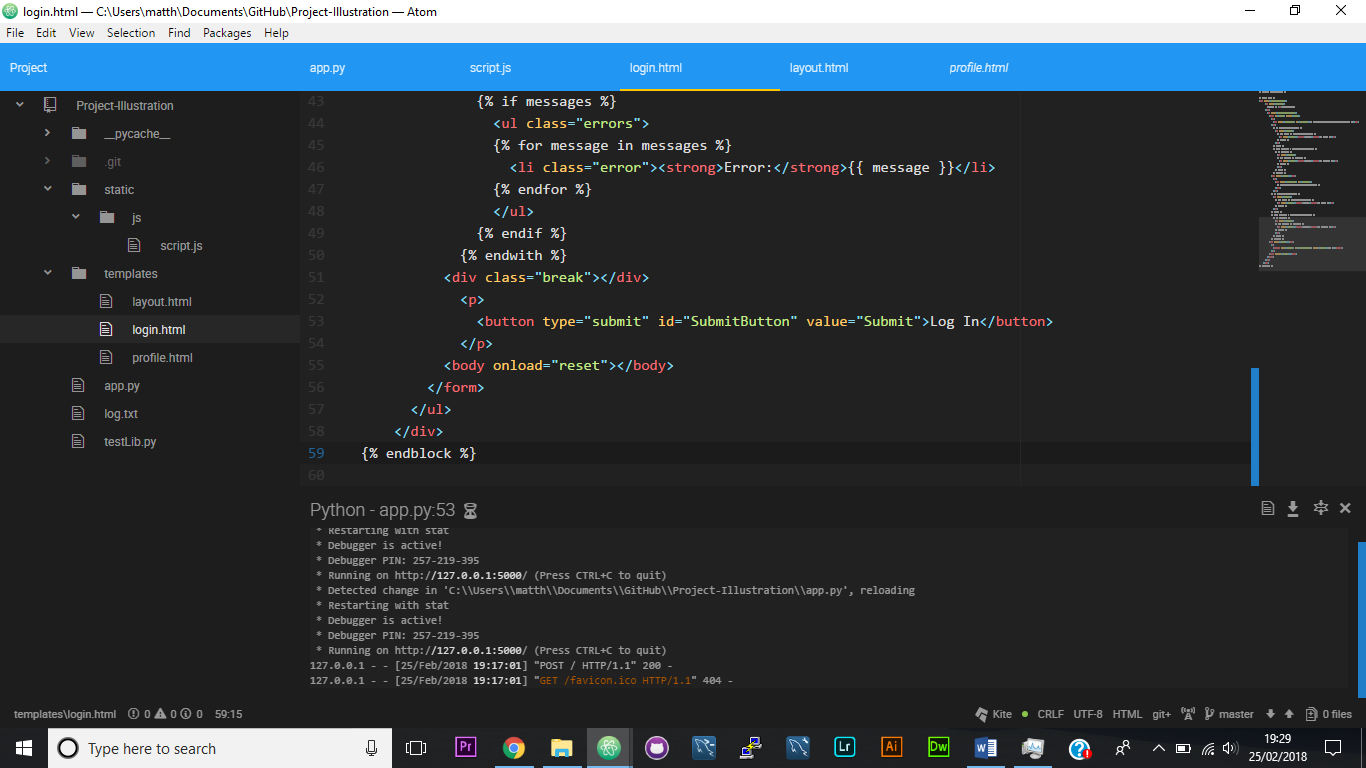
To do this I will first create a folder called ‘static’, and in that create a folder called ‘js’. I will then create a file called ‘script.js’ within which I will write a function that will reset the fields on reload.



Then I will link the file to the ‘layout.html’ page.



Finally, I will call the JavaScript function in my HTML on reload.



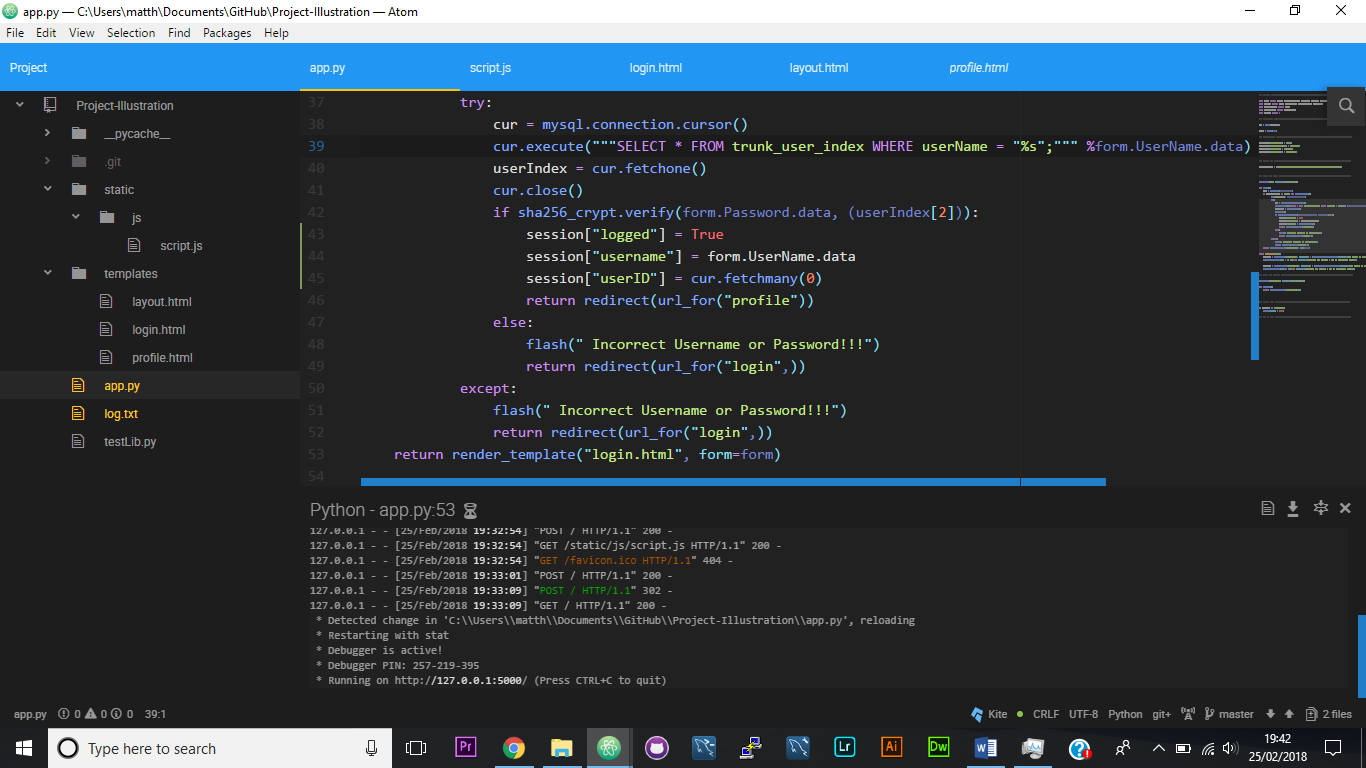
Finally, I will test it.

### Step 6 test table:

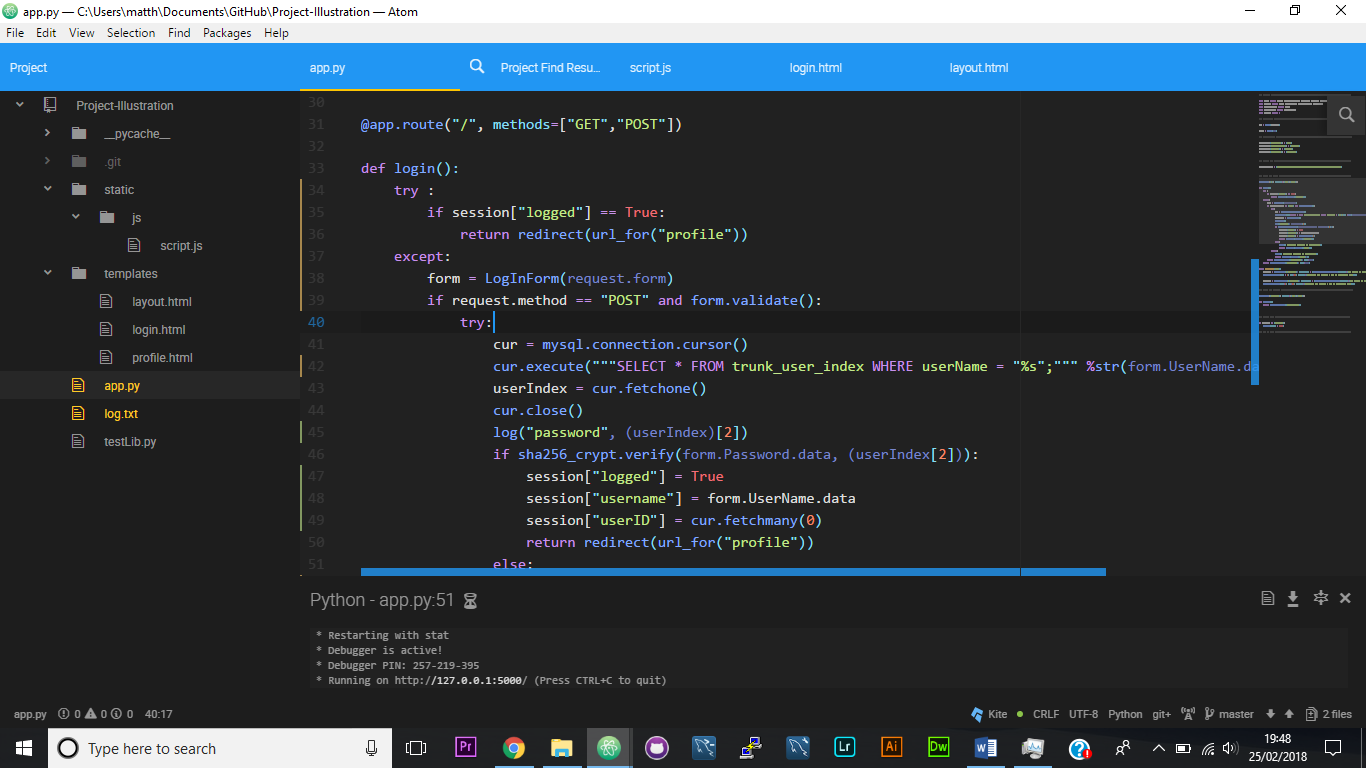
|  |  |  |
| --- | --- | --- |
| Test scenario | Expected response | Actual response |
| Input wrong name, examine reload | Fields empty after reloading | Fields were empty |

## Step 7) Return user info

To do this step I will first have to import ‘session’ from ‘flask’. Next I will write a piece of python code in ‘app.py’ so that if the user name and password are correct, three variables called session[“logged”], session[“username”] and session[“userID”] are set to True, the form data for username and the first column in the row queried for the password validation.



Before the main block of code, I put an if else statement that directs you directly to profile if session[“logged”] == True.



Next, I put a block of code in profile that requires you be logged in to view it, and that redirects you back to the login page if you’re not.

Then I made a navbar