

NCTTS ReadMe file

Running the software:

There are two ways to test the software.

1. Use the software from the cloud

- a. Visit and try out the website at <https://www.nctts-essex.com>
- b. Click on Login.
- c. Login using the admin credentials.
- d. Click on Add users.
- e. Enter the user username, password, and public IP address.
 - i. **Note: it is important you add the public IP address of the user otherwise you will not be able to log into the operator page.**
- f. Click on save and logout.
- g. Click on login and enter the operator user credentials.

2. Run the software locally

- a. Install the following softwares:
- b. Install the latest version of Python. As of writing this document it is 3.11
 - i. <https://www.python.org/downloads/>
- c. Install the latest version of Django. As of writing this document it is 4.1.3
 - i. <https://www.djangoproject.com/download/>
- d. Install the latest version of Git. As of writing this document it is 2.38.1
 - i. <https://git-scm.com/downloads>
- e. Clone the project from GitHub using the following link
https://github.com/mustafa-sibai-essex/4_SSDCS_PCOM7E_2_Coding_Output
- f. Navigate to the nctts folder in the cloned project
- g. Running the following command to run the http server
 - i. Python manage.py runserver
 - ii. **Note: When running the above command, Django will start a development server running on the less secure http protocol rather than the https protocol. This is ok since it is a development server. To view the website using the https protocol please visit the <https://www.nctts-essex.com> website.**
- h. Click on Login.
- i. Login using the admin credentials.
- j. Click on Add users.

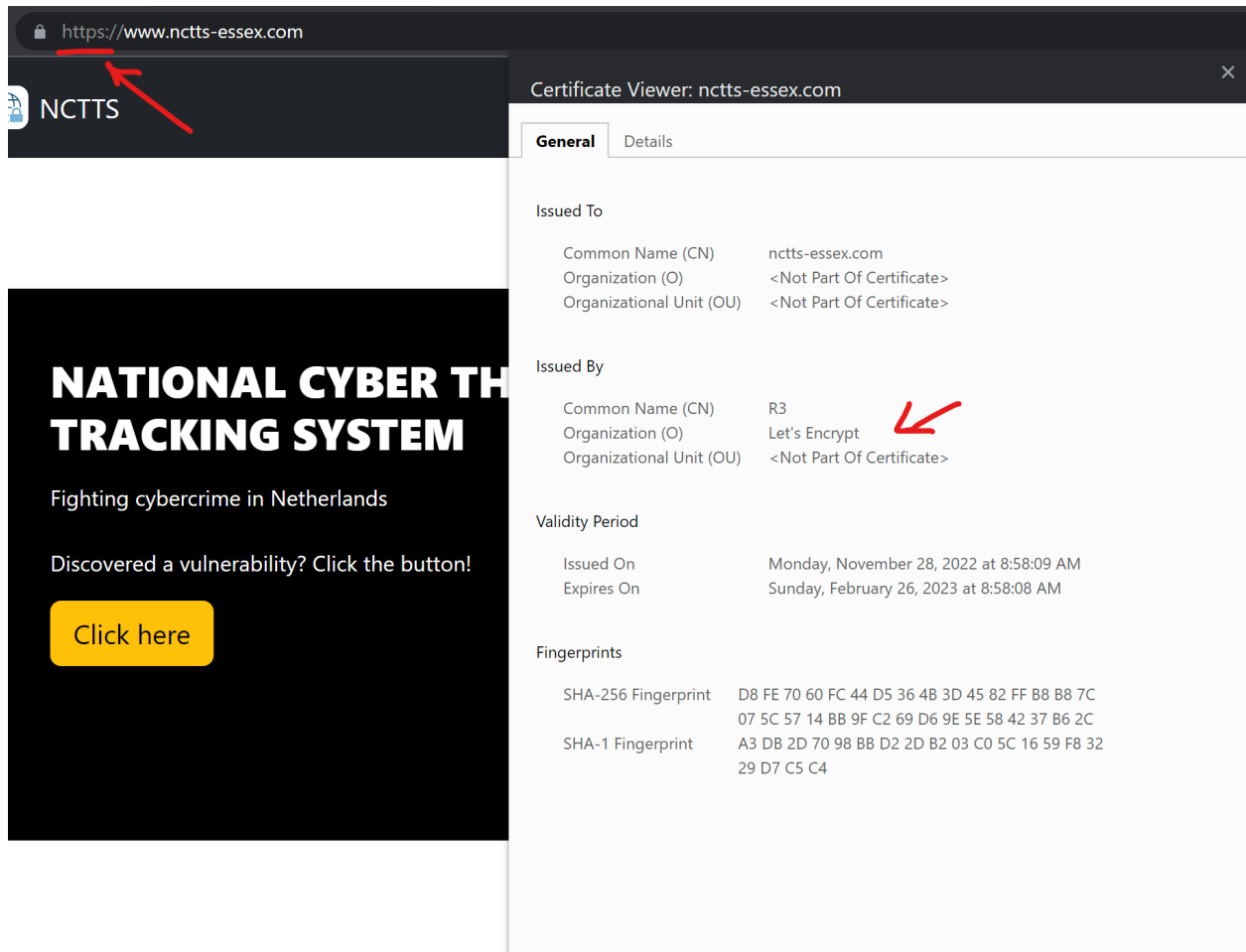
- k. Enter the user username, password, and add the loopback IP address which is 127.0.0.1
 - i. **Note: it is important you add the loopback IP address otherwise you will not be able to log into the operator page. The loopback IP address is only used in the development server. In the production server the user public IP address must be used instead.**
- l. Click on save and logout.
- m. Click on login and enter the operator user credentials.

Login credentials:

- 1. Admin login:
 - a. Username: admin
 - b. Password: Admin_useR@1962

Security measures:

- 1. **Server running on the HTTPS protocol**
 - a. The server is using the free LetsEncrypt SSL certificate for the domain nctts-essex.com



2. SQL injection protection

By default Django provides a very secure authentication system using the auth module. Rather than reinventing the wheel we have decided to use it. The auth module and the entire Django system has SQL injection protection within it. The way it does this is by using query parameterization.

“Django’s queriesets are protected from SQL injection since their queries are constructed using query parameterization. A query’s SQL code is defined separately from the query’s parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.”

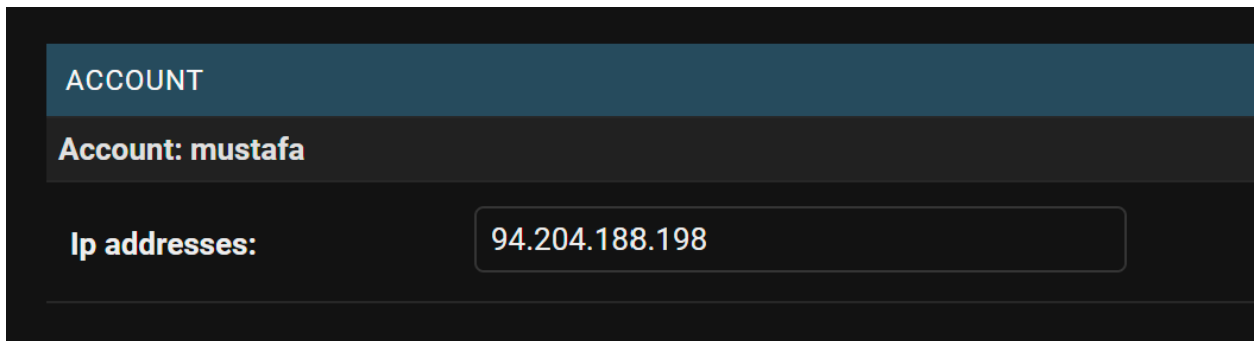
(Django 2022 SQL injection security)

We rely on Django query parameterization and we make sure to never use raw SQL query in any of our code.

3. Whitelisting IP address

Part of our security measure is we make sure to never grant access to operators who aren't logged in from a whitelisted IP address. When an admin creates the operator account, they are required to add the user IP address. This way if some third party gains access to the operator login credentials, they cannot log into the system because their IP address would be different unless they are logging in from the operator location.

The below picture shows how the admin can whitelist an IP address for the mustafa account.



The screenshot shows a dark-themed user interface for account management. At the top, there is a header bar labeled 'ACCOUNT'. Below this, the account name 'Account: mustafa' is displayed. Underneath, there is a section labeled 'Ip addresses:' followed by a text input field containing the IP address '94.204.188.198'.

4. SSH server protection

From time to time we have to access the server in which our website is hosted on. There are two methods of doing this, one is by typing the password and the other is by generating an SSH key which is a lot more secure. We have opted to generate an SSH key and only allow certain IP addresses to access the server in Google Cloud Platform.

The below picture shows a firewall rule which only allows a single IP address 94.204.188.198 to access port 22 which is the SSH port on Google Cloud Platform server.

The screenshot shows the Google Cloud Firewall console. On the left is a sidebar with navigation links: VPC network, VPC networks, IP addresses, Bring your own IP, Firewall (selected), Routes, VPC network peering, Shared VPC, Serverless VPC access, and Packet mirroring. The main content area is titled 'Firewall' and includes buttons for 'CREATE FIREWALL POLICY' and 'CREATE FIREWALL RULE'. Below these are links for 'GO TO NETWORK INTELLIGENCE CENTER' and 'REMIND ME LATER'. A notification banner states: 'Easy to deploy network threat detection with Google Cloud IDS. [Learn more](#)'. The section 'VPC firewall rules' explains that firewall rules control incoming or outgoing traffic and provides a link to 'Learn more'. A note mentions that App Engine firewalls are managed in the 'App Engine Firewall rules section'. A warning icon indicates 'SMTP port 25 disallowed in this project'. Below this are buttons for 'REFRESH', 'CONFIGURE LOGS', and 'DELETE'. A filter bar is present above a table of firewall rules.

Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network
default-allow-http	Ingress	http-server	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	default
default-allow-https	Ingress	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	default
gunicorn	Ingress	gunicorn	IP ranges: 0.0.0.0/0	tcp:8000	Allow	1000	default
ssh-access	Ingress	ssh-access	IP ranges: 94.204.188.198/32	tcp:22	Allow	1000	default

GDPR (General Data Protection Regulation) compliance:

Part of our GDPR compliance, we allow the general public to request for personal information deletion.

Multithreading

Sending an email takes quite a bit of time and can freeze the page until the email is sent. Sending many emails can make the page unresponsive for a very long time. Therefore it is necessary to execute this operation in a background thread to keep the website responsive.

The below picture shows the code responsible for creating a background thread and sending an email from it.

```

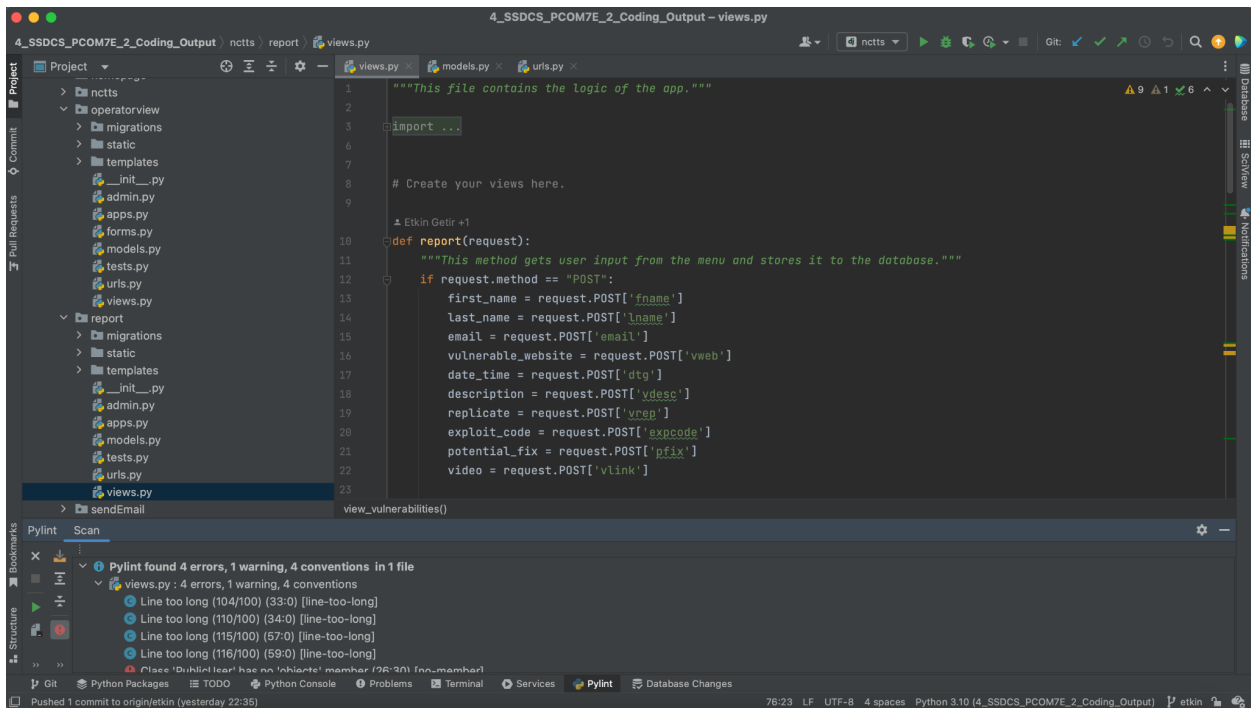
class SendEmail(threading.Thread):
    """Creates a background thread to send the email. Rather than freezing the page until the operation
    .. def __init__(self, title, body, email):
    ..     self.title = title
    ..     self.body = body
    ..     self.email = email
    ..     threading.Thread.__init__(self)

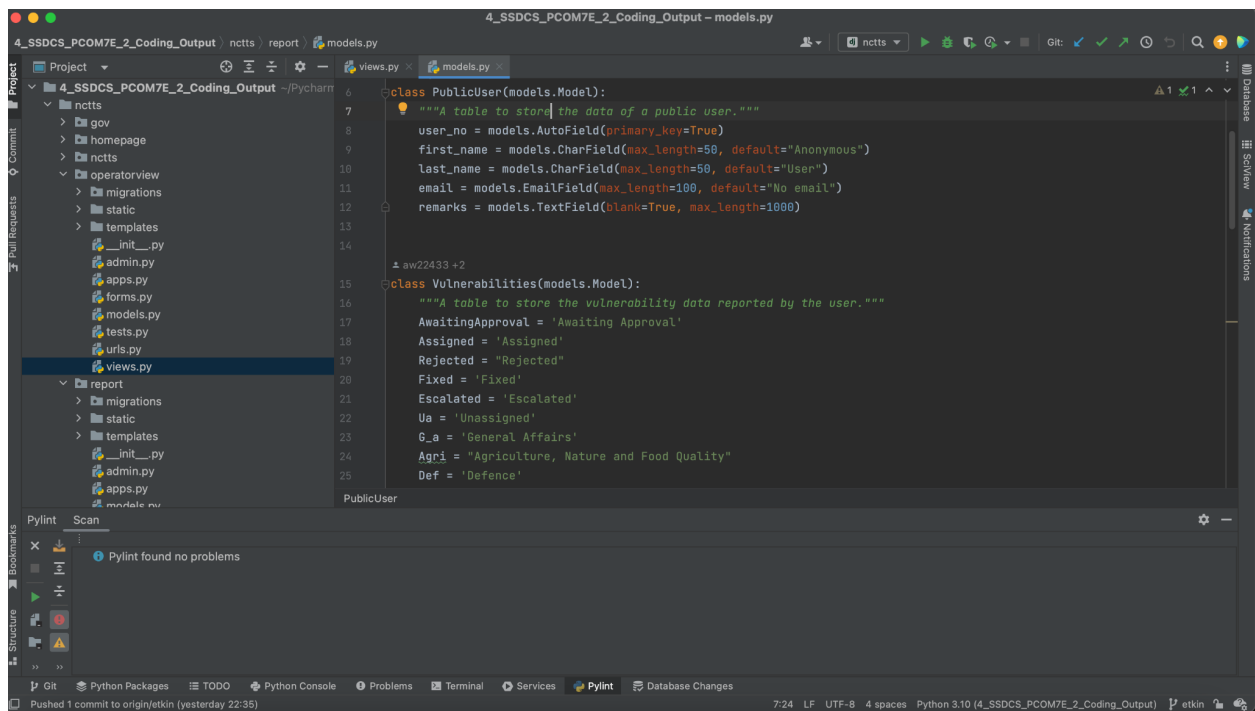
    .. def run(self):
    ..     """Runs the thread in the background"""
    ..     try:
    ..         send_mail(
    ..             self.title,
    ..             self.body,
    ..             None,
    ..             [self.email],
    ..             False,
    ..         )
    ..     except Exception as e:
    ..         print(e)

```

Linters

Pylint was used as the linter of choice for this project. All of the files in the each app were scanned by pylint:

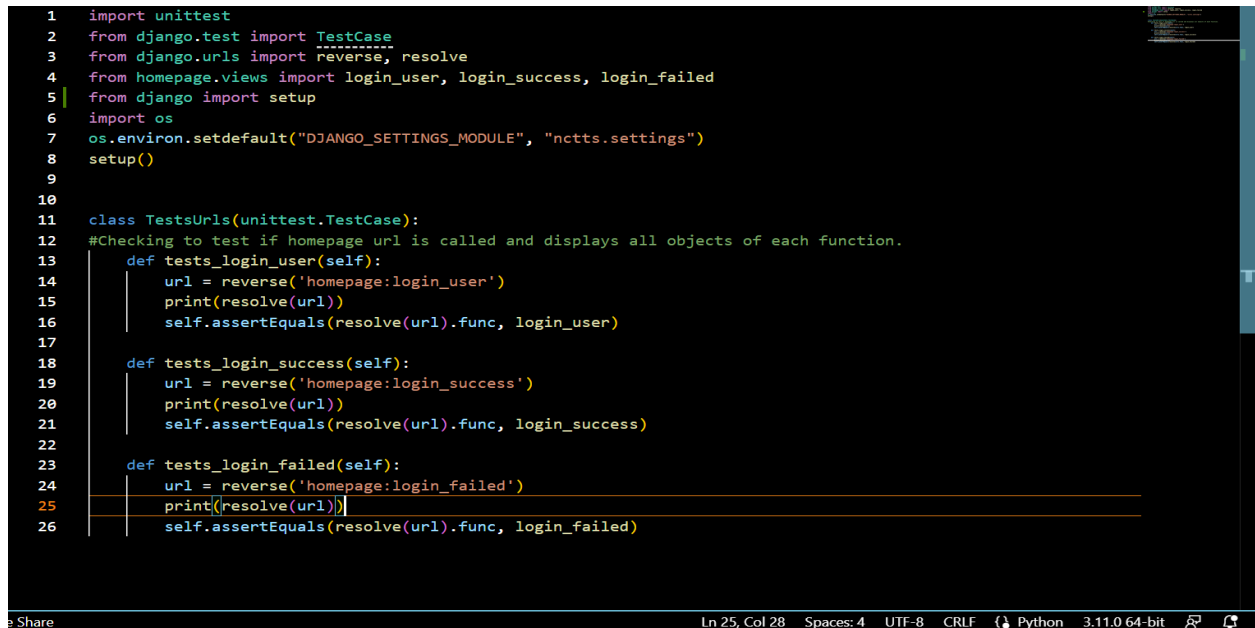




Testing with Unittest

This is a concise description of the unit test which was used to test part of the code, the unittest is used to test a component or an isolated part of code which ensures code should function as intended.

The code below tests the Url within the homepage and return the correct Url link.



Below we can see each function which was tested has passed successfully.

```
Found 3 test(s).
System check identified no issues (0 silenced).
ResolverMatch(func=homepage.views.login_failed, args=(), kwargs={}, url_name='login_failed', app_names=['homepage'], namespaces=['homepage'], route='login_failed')
.ResolverMatch(func=homepage.views.login_success, args=(), kwargs={}, url_name='login_success', app_names=['homepage'], namespaces=['homepage'], route='login_success')
.ResolverMatch(func=homepage.views.login_user, args=(), kwargs={}, url_name='login_user', app_names=['homepage'], namespaces=['homepage'], route='login_user')
.
-----
Ran 3 tests in 0.006s
```

References:

Django Software Foundation (N.D.) Django Documentation - SQL Injection security. Available at: <https://docs.djangoproject.com/en/4.1/topics/security/> (Accessed: December 5, 2022).

Django Software Foundation (N.D.) Django Documentation – ModelForms. Available from: <https://docs.djangoproject.com/en/4.1/topics/forms/modelforms/> (Accessed 04/12/22)

Django Software Foundation (N.D.) Django Documentation – Widgets. Available from: <https://docs.djangoproject.com/en/4.1/ref/forms/widgets/> (Accessed 04/12/22)

Django Software Foundation (N.D.) Django Documentation – Queries. Available from: <https://docs.djangoproject.com/en/4.1/topics/db/queries/> (Accessed 04/12/22)

Pylint (N.D.) What is Pylint? Available from: <https://pylint.pycqa.org/en/latest/> (Accessed 05/12/22)

Voigt, P., von dem Bussche, A. (2017) *The EU General Data Protection Regulation: A Practical Guide* Cham: Springer. DOI: <https://doi.org/10.1007/978-3-319-57959-7>