In order to make sure that the implementation is secure a few tests were carried out as mentioned in Chapter 3. The first was **SQL queries** being executed directly from the any of the POST and GET forms. When the form was submitted the back-end directly took the data as being unsafe and produced an error. As stated in Django's **security documentation**[?] the SQL commands were **escaped and ignored** hence making it safe to use.

Another test executed on the system was **Cross Site request forgery(CSRF)**, this meant duplicating a cookie from a browser where the user was already logged in and trying to send POST requests from another browser. Each POST form in Django requires a **Secret Key** to be present, and when the duplicated cookie POST request did not have a Secret Key, Django automatically **denied access** to the action. This security feature is extremely important as common **Remote Administration Tools** can access your cookies and duplicate them.

Cross Site Scripting(XSS) had its limitations when testing, for example if a CSS Style was defined within the HTML code, a simple execution of a JavaScript code would be possible to access unauthorised data but as there was no such code present a significantly professional code would be required to test and penetrate through Django's security system for auto escaping unknown code.

Simple **Automated tests** called '**unittest**' were also executed using Django's inbuilt features to test out the execution of **POST and GET forms**. A simple unittest for POST forms such as login, profile update, and request more files was executed where the result was displayed in console. The result were verified using **response code** received when the code was executed as well as the page location. For example when testing;

```
client.post('/account/login/', {'email': 'rshah5@sheffield.ac.uk',
 'password': 'complexp@ssw0rd'})
```

The response received was '200' with the HTML code for the dashboard page.

Lastly, the system was manually tested for bugs and security loop holes as well as user reviews based on ease of access, confidence and comparison. When running manual tests, a user with Student privileges would be able to access pages where only a Secretary would have access. For example simply typing

```
<link to website>/secretarypanel/viewform?id=1
```

would allow the student to access and change data as though they were the secretary. Same applied for the Scrutiny panel as well as regular forms created by other Students. In order to prevent this, validation checks were implemented which would check the privileges of the user and only give them access to specific pages in the system. The checks would also make sure that the logged in session user is accessing only data created by them and not by other users(mainly for the Student Panel as the Secretary has access to all).