For the Introduction to Cybersecurity project, I created a simple to-do list web application that allows for a user to create a to-do list, add tasks to it, mark them as complete, and delete the to-do list altogether. There are two example accounts that were created:

| | |
|---|---|
| User: bob | Password: squarepants |
| User: alice | Password: redqueen |

**Flaw 1: Identification and Authentication Failures**

A web application is said to have identification and authentication failures if the application permits weak, default, and well-known passwords. This application is a prime example of this flaw, as there are no checks made to see if the password for a new account is strong enough. The link to this flaw can be found here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/models.py#L5-L6
As noticed, there are no checks done to the password, just that the character limit is 200. We fix this flaw by making checks on the password, which is done in models.py. We do this by making a function within the List model, and performing multiple checks to verify the strength of the password. If the password fails any of the tests (password less than 8 characters, password not containing a lower-case letter, password not containing an upper-case letter, password not containing a digit), the application simply raises a validation error, prompting the user to create a stronger password for their account. The fix can be viewed here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/models.py#L8-L20

**Flaw 2: Broken Access Control**

This flaw maintains that users should not be able to act outside their intended permissions. In the case of this web application, this means that a user should not be able to access the "view list" page of other users. However, this flaw is included in the application. Thus, a user can simply find the user ID of another user and use it to access the view page. For example, if a user's ID number is 21, anyone could simply type the URL "http://localhost:8000/view_list/21/" to access their to-do list. The flaw can be found in the verify_password function and view_list function in views.py. We fix this flaw by creating a session variable every time a password has been verified for a specific user. The session variable can be seen here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L53-L56
If the password is verified, then the view_list page will be accessible. However, if the password has not been verified for a specific user, then the web application will automatically be redirected to the "verify password" page. This way, if a user tries to switch the user ID in the URL, it will redirect to the "verify password" page, instead of allowing the hacker to access anyone's page. The fix can be found through the following link:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L81-L85

**Flaw 3: Injection**

A web application is vulnerable to an injection attack if the user-supplied data is not validated, filtered, or sanitized properly by the application. Additionally, if any unsafe SQL queries are performed within the web application, that leaves it vulnerable to an attack. In this web application, an unsafe query is performed to delete lists from the database. This would allow for injections to exploit the rest of the data, and possibly delete all the to-do lists. The link to this flaw can be found here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L123-L129
In order to fix this flaw, we will need to completely remake the delete_list function, and use Django's ORM to delete the list instead. Since we are now using ORM instead of the unsafe SQL query, we are no longer vulnerable to SQL injections. The fix can be viewed here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L131-L137

**Flaw 4: CSRF Attacks**
Cross-Site Request Forgery (CSRF) is a type of attack that allows an attacker to perform unwanted actions on a web application in which they currently have authentication. In the case of this web application, the attacker could do things such as delete the to-do list, mark tasks in the to-do list as complete, and add unwanted tasks to the list. There is no location of this flaw in the web application, as it is more about the lack of the CSRF protection rather than invalid code. Fixing this flaw in the application is quite simple; for all post requests in our HTML templates, we simply add the CSRF token template tag. Then, in the views file, we import the csrf_protect tag and use it to add additional protection against CSRF attacks. Although Django has built-in protection against CSRF attacks, the inclusion of these tags will provide enough protection to make it so CSRF attacks are no longer feasible and the web application will be protected against them. An example of the csrf_protect tag being used in the code can be viewed here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L27-L30
And an example of a CSRF token template tag being used can be viewed here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/509d5d40802d6723f26b4cd8344f0c96434cecc2/Cyber-Security-Project/mysite/src/pages/templates/pages/verify_password.html#L9

**Flaw 5: Security Logging and Monitoring Failures**
This flaw occurs in a web application when there is not sufficient logging, detection, and monitoring taking place. Events such as logins and failed logins should be monitored to detect suspicious activity in the webpage. This web application has the flaw of security logging and monitoring failures, as there is no use of logging at all. Therefore, there is again no location for this flaw as the lack of logging is the flaw. This flaw can be fixed by creating a logger, and then tracking certain things on the web application, such as logins, failed logins, and invalid forms. This can be seen throughout the code, as invalid forms and incorrect passwords are reported to the logger. For example, if an incorrect password is submitted, an error is logged, which can be viewed here:

https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L60-L63
Also, the importation of logging can be viewed here:
https://github.com/JesseHantula/Flawed-Website-Project/blob/main/Cyber-Security-Project/mysite/src/pages/views.py#L13-L17