D424 – Software Engineering

Task 3

Capstone Proposal Project Name: University Course Tracker

Student Name: Lance Roller

Table of Contents

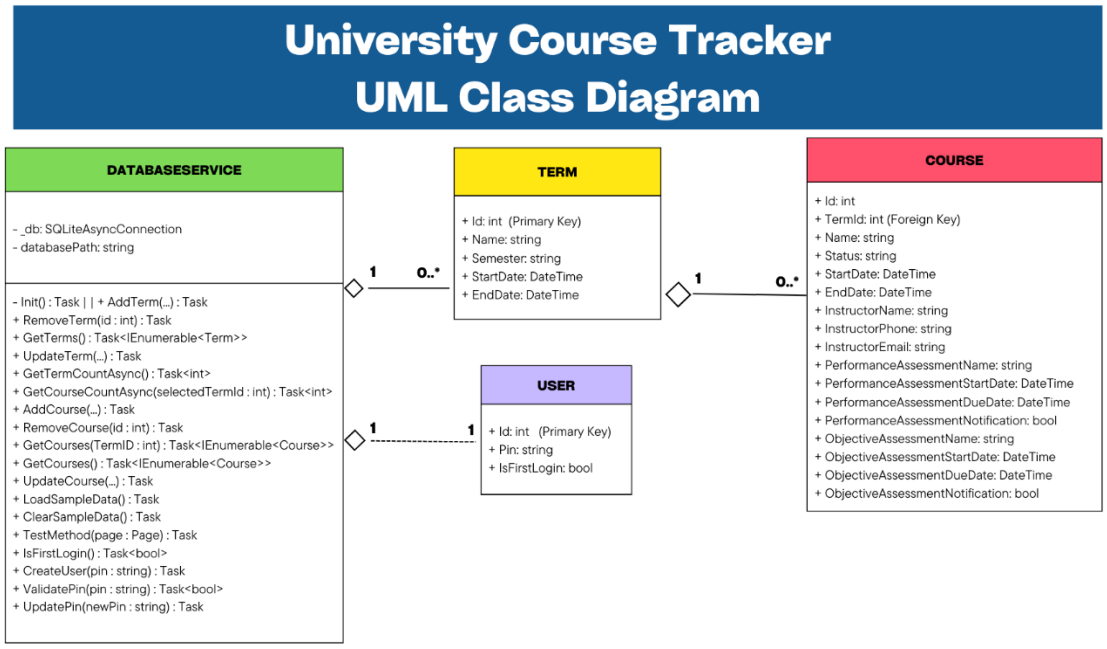
Task 3: Design Document

Task 3 Design Document

Part C1

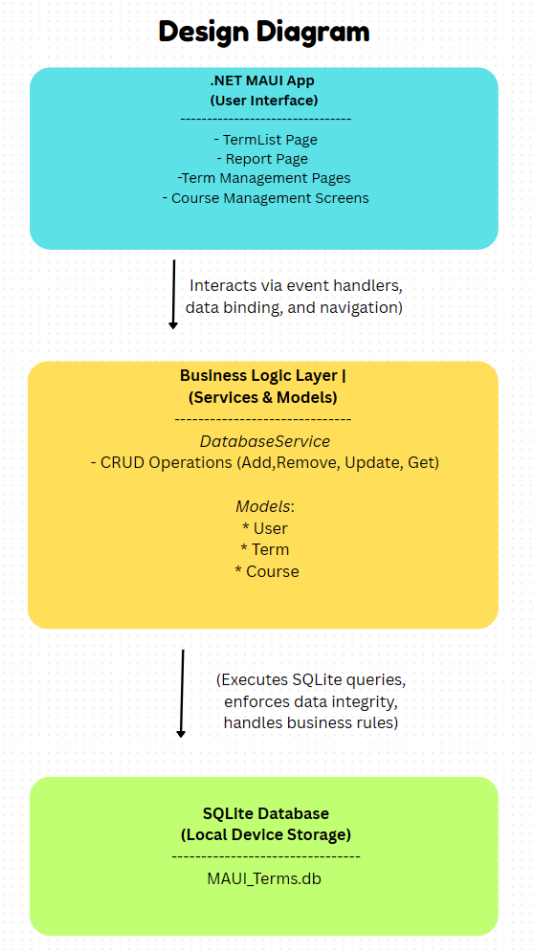
Class Diagram:

The UML (Unified Modeling Language) Class Diagram below shows the properties, functions, and relationships between each class.



Design Diagram:

The Design Diagram below shows the different layers of the University Course Tracker application. It shows a high-level diagram of the different tiers from the Presentation Layer to the Application Layer, then the Data Layer.



Part C2

Link to Application:

Here is a link to the application being hosted on Google Play Store. Currently the application requires being an "Internal Tester" to download.

Link: **https://play.google.com/apps/internaltest/4701649778042781273**

Part C3

Link to GitLab Repository:

Here is a link to the Git Lab repository 'Working' branch.

Link: **https://gitlab.com/wgu-gitlab-environment/student-repos/lroll47/d424-software-engineering-capstone/-/tree/Working?ref_type=heads**

Part C4

User Guide: Maintenance

Maintaining the application requires ensuring that your development environment is correctly set up. You will need to have the **.NET 8.0 SDK** installed along with the **.NET MAUI** workload. Additionally, having **Visual Studio 2022 with MAUI support** is essential for development, while **SQLite Database Browser** can be helpful for debugging database issues. A working **Git installation** is also necessary for version control and managing repository updates

.NET 8.0 SDK: https://dotnet.microsoft.com/en-us/download/dotnet/8.0

Visual Studio: https://visualstudio.microsoft.com/downloads/

GIT: https://git-scm.com/downloads

To begin, you must clone the repository from GitLab. Open a terminal or command prompt, navigate to your preferred working directory, and run the **git clone** command using the repository URL

Repository URL: https://gitlab.com/wgu-gitlab-environment/student-repos/lroll47/d424-software-engineering-capstone.git

```
C:\Users\chron>cd F:\GithubProjects

C:\Users\chron>F:

F:\GithubProjects>git clone https://gitlab.com/wgu-gitlab-environment/student-repos/lroll47/d424-software-engineering-ca
pstone.git
```

Once cloned, navigate into the project folder and check out the latest stable version using git checkout, specifying the appropriate branch or tag ("Working" in this case).

```
F:\GithubProjects\TEST\d424-software-engineering-capstone>git checkout Working
Switched to a new branch 'Working'
branch 'Working' set up to track 'origin/Working'.

F:\GithubProjects\TEST\d424-software-engineering-capstone>
```

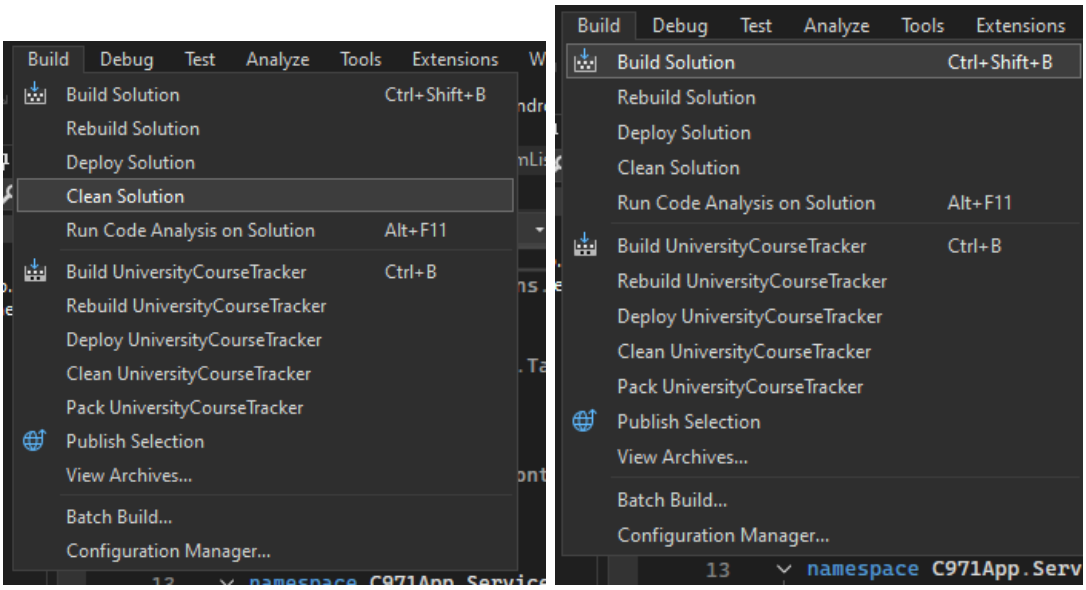
Once the repository is ready, open the project in **Visual Studio 2022**. Before running the application, restore dependencies by executing dotnet restore. Ensure that the correct platform is selected based on whether you are running it on Windows or deploying to an Android emulator or physical device.

```
dotnet restore
```

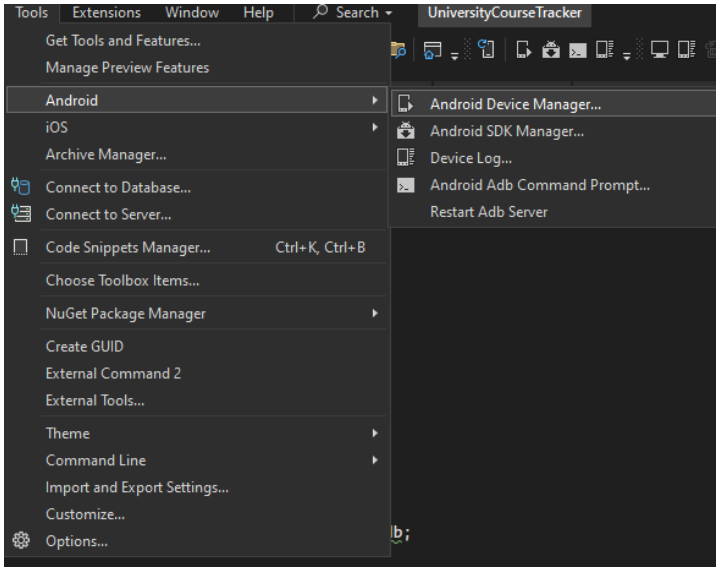
The application uses an SQLite database for local storage. If you need to modify the database schema, you can update the table definitions within the DatabaseService file. If resetting the database is required, delete the existing database file and restart the application to generate a new one. This can be done by removing the MAUI_Terms.db file and running dotnet run to apply the changes.

```
rm -rf MAUI_Terms.db
dotnet run
```

Using the Visual Studio interface can be used as well instead of the command line. To build the application first go to **Build > Clean Solution**, then **Build > Build Solution (or Rebuild Soltuion)**



If using an emulator make sure you have the appropriate emulated devices downloaded along with the appropriate SDKs. For testing purposes it was most commonly used to use API 31. To check installed SDKs and Devices, go to **Tools > Android > Android Device Manager** for Devices, and for **SDKS go to Tools > Android > Android SDK Manager**.



For deployment updates, commit your changes to the repository using **git add .** and **git commit -m**, followed by git push to the relevant branch.

```
git add .
git commit -m "Updated feature X"
git push origin <branch>
```

If issues arise, **debugging tools in Visual Studio** can be used, including breakpoints and application logs. Checking SQLite logs can also provide insight into database-related errors. If necessary, additional logging can be implemented within the application for better error tracking.

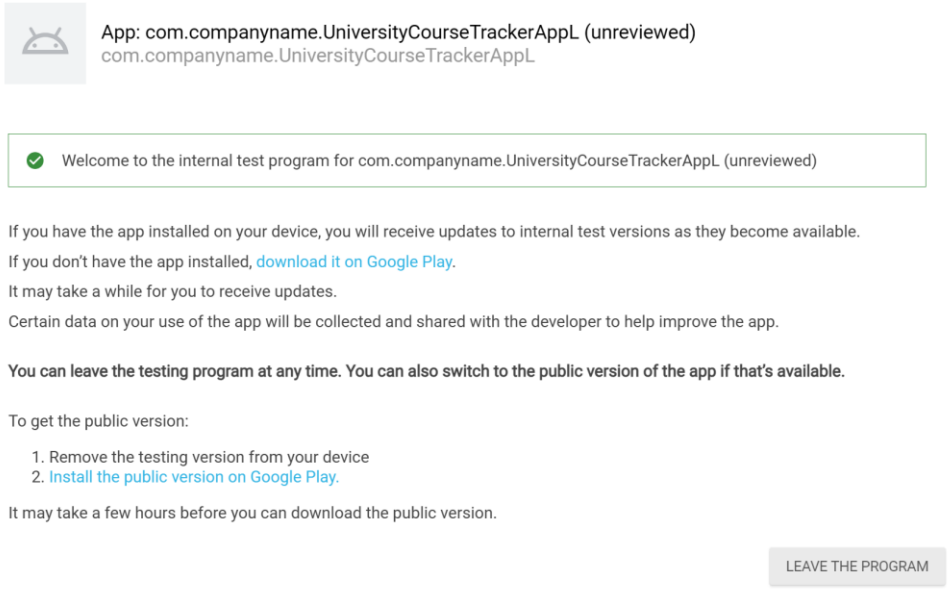
Part C5

User Guide: Running Application

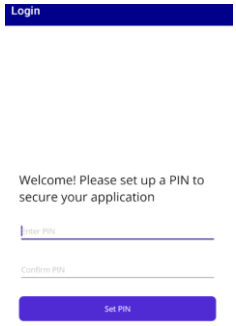
To access the application, internal testers need to navigate to the **Google Play Store** and ensure they are signed in with an email that has been added to the **Internal Testing** group. Once logged in, they should go to the **Beta & Internal Testing Apps** section, locate the application, and install it. It's also possible to access the internal testing track via hyperlink if your email is

added as a tester. Make sure to "**download it on Google Play".** If it's your first time, you may have to **"Opt In"** first.

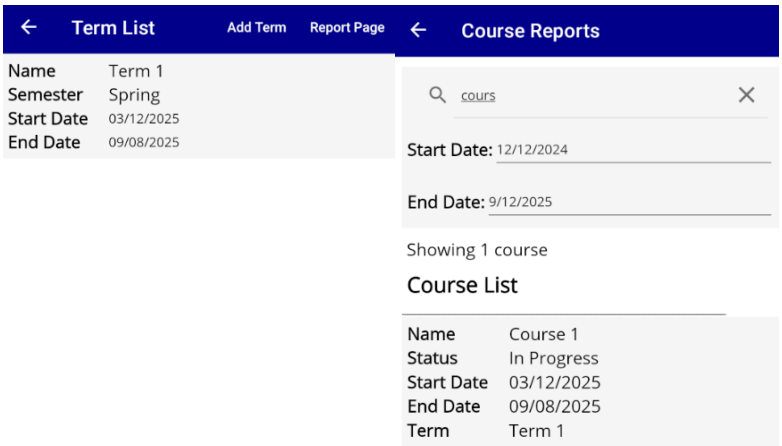
Link: https://play.google.com/apps/internaltest/4701649778042781273

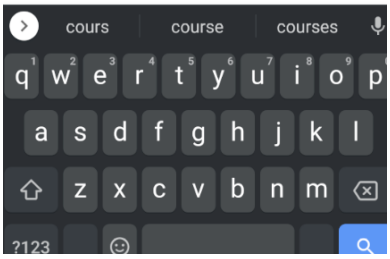


After installation, the application can be launched from the home screen or the app drawer. Upon opening the app for the first time, necessary permissions may need to be granted, such as storage and notifications. The application will request a to set up a **pin** for the first time upon opening. Enter your desired pin that you will later use to access your information.



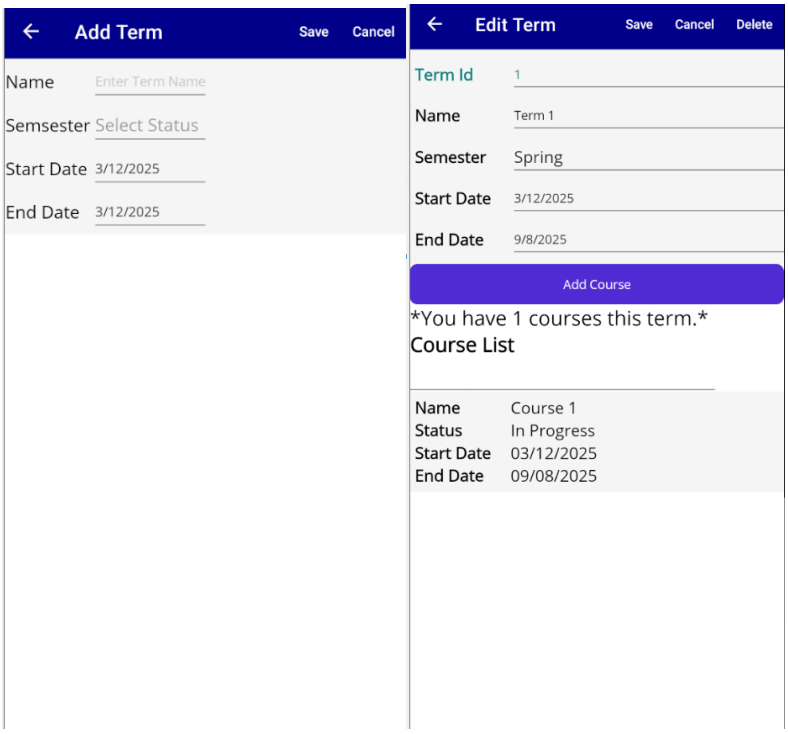
The application consists of several key sections. The **Term List Page** allows users to view available terms, while the **Report Page** provides summaries of which courses the user is enrolled in. On the Report Page, simply search for your name of the desired course, and put in the desired dates, and it will return all courses that meet the criteria.





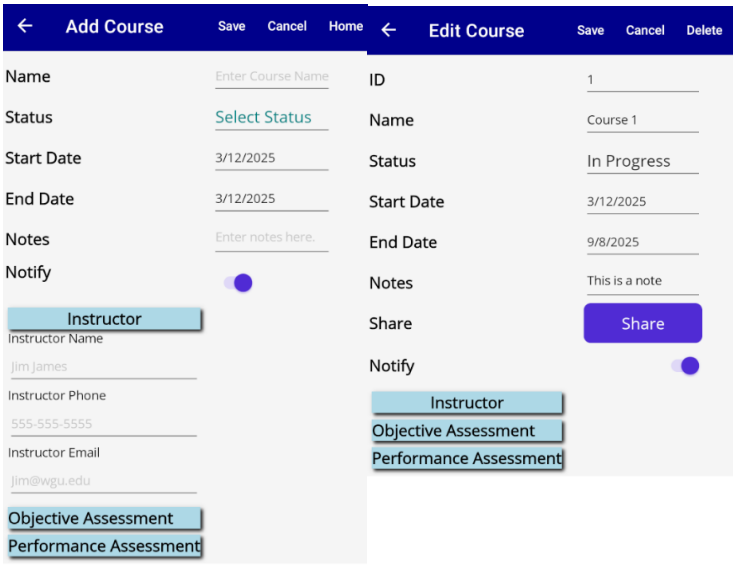
The **Add Term** page allow users to add new Terms. To add a term, users click “Add Term”, enter the necessary details, and save the changes.

To edit a Term, click on the desired term from the **Term List** page. It will bring you to the **Edit Term** page. Edit any of the desired term details then click “Save”. You can also delete the term by clicking “Delete”. If you want to cancel the changes, simply click “Cancel”.



The **Add Course** page allow users to add new courses. To add a course, from the **Edit Term** page select the option to add a new course, enter the necessary details, and save the changes. The “Notify” toggles allow the user to be sent mobile notifications regarding about upcoming start and end dates, which can be turned off in the **Edit Course** page.

To edit a course, click on the desired course to be edited from the **Edit Term** page. It will open the **Edit Course** page. Edit any of the desired course details then click “Save”. You can also delete the course by clicking “Delete”. If you want to cancel the changes, simply click “Cancel”. From this page you can also share your “Notes” to an external source.



If users encounter any issues or bugs, they should take a screenshot, note the steps to reproduce the issue, and report it using the **Google Play Internal Testing feedback system**. Regular updates to the application will be available through the Google Play Store, and testers should ensure they always have the latest version installed. Updates can be managed through the **Manage Apps** section in Google Play, where users can check for new versions and install them as needed.

Part D1

Unit Test Plan:

For our unit test plan, we went with using the XUnit package. We made a separate project that tests two different functions using a mock database and methods. The functions and tests are briefly explained in the table below.

Test	What is Expected	What Happened	Pass?
Tests validation with empty PIN	The app displays a message stating that a PIN is required (or returns false)	The test confirmed that "public async Task ValidatePin_EmptyPassword_ReturnsFalse()" returned false.	Yes
Tests validation with correct PIN	The user is authenticated and allowed to proceed.	The test confirmed that "public async Task ValidatePin_CorrectPassword_ReturnsTrue()" returned true.	Yes

Part D2

Unit Test Scripts:

Script 1:

```
[Fact]
| Invoke Codeium | 0 references
public async Task ValidatePin_EmptyPassword_ReturnsFalse()
{
    // Arrange: Define an empty PIN.
    string emptyPin = "";
    // Act: Attempt to validate the empty PIN.
    bool result = await MockDatabaseService.ValidatePin(emptyPin);
    // Assert: The result should be false, as an empty PIN is invalid.
    Assert.False(result, "Validating an empty PIN should return false.");
}
```

```
public static async Task<bool> ValidatePin(string pin)
{
    await Init();
    if (string.IsNullOrEmpty(_storedHashedPin))
        return false;

    // Simulate BCrypt verification
    return _storedHashedPin == "$2a$" + pin + "$";
}
```

Script 2:

```
[Fact]
| Invoke Codeium | 0 references
public async Task ValidatePin_CorrectPassword_ReturnsTrue()
{
    // Arrange: Define a test PIN.
    string testPin = "1234";
    // Ensure that a user is created with the test PIN.
    await MockDatabaseService.CreateUser(testPin);
    // Act: Validate the correct PIN.
    bool result = await MockDatabaseService.ValidatePin(testPin);
    // Assert: The result should be true, as the correct PIN is provided.
    Assert.True(result, "Validating the correct PIN should return true.");
}
```

```
public static async Task<bool> ValidatePin(string pin)
{
    await Init();
    if (string.IsNullOrEmpty(_storedHashedPin))
        return false;

    // Simulate BCrypt verification
    return _storedHashedPin == "$2a$" + pin + "$";
}
```

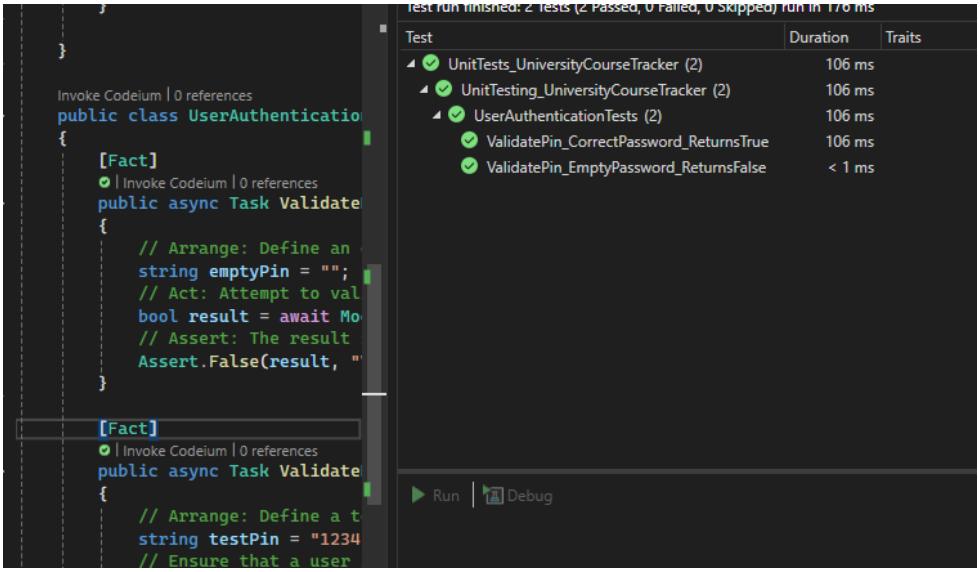
```
Invoke Codeium | 1 reference | 1/1 passing
public static async Task CreateUser(string pin)
{
    await Init();
    // Simulate BCrypt hashing
    _storedHashedPin = "$2a$" + pin + "$"; // Not actual BCrypt, just a simulation
    _isFirstLogin = false;
}
```

Part D3

Results of Unit Tests:

The two test scripts are put within one file “UnitTest1.cs”. The first script checks whether the PIN is empty, we “Assert” that it will be “False” due to the PIN being empty, which when entering an empty PIN turns out to be “True”.

The second script validates when a correct PIN is input. When a correct PIN is input, we “Assert” that the result will return “True”. We ensure to create a user with the desired PIN. We then used that PIN same pin and ensure to validate it. The validation returns the correct pin and our assertion becomes true, which returns “True”



Part D4

Summary of Changes Due Tests:

Fortunately, there were no changes due to our unit tests. The reasons for this being is that each function performed expectedly. Now, however, if our assertions were incorrect, and the tests failed. We would have had to change the functions in our database appropriately so that empty PINs show users' errors, and that correct PINs would validate successfully and the user could login in. These unit tests gave me more confidence in the code of our functions, and the application functionality.