EAD2 CA2 – Android Project

An Book tracker app (BOOKTRACKR)

Ciaran Dunne

2021

Table of Contents (Page Numbers will change)

**IntroductionPage 1**

**GitHub InformationPage 2**

**Database SchemaPage 2**

**ServicePages 3 - 6**

**Application BreakdownPages 7**

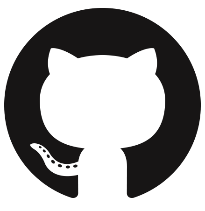
**ConclusionPage 7**

**Introduction**

For this CA I wanted to create an APP that would create a list of books user has. The user will be able to create a list by assigning it a name and author.

I wanted the user to be able to go through the list of books to view and read.

The Web API methods I created in ASP.NET Core and the client side of the application will be using Java in Android Studio.

**GitHub Information**

Repo URI: https://github.com/CodeCiaran/EAD2CA2.git

I made this repo in order to store the Web API so it could be pulled. I was first having issues commiting to the repo so I had to manually add the stored local Web API and Android App to this repo. These can be unzipped and used. I have got a understanding of repo and using the commits and branches correctly in order to get latest version and previous versions.

**Database Schema**

Entity Relationship graph generated from SQL Server.

BLists

* ID
* Username
* Name

Book

* Id
* Author
* Name

**Service**

**Our Operations**

**GET Methods:**

**GetLatestLists():** This GET method returns the **latest** album lists that were added to the service. It does this by ordering the list in a descending order through the date of creation using LINQ.

**GetMyLists():** This GET method returns the lists that have been added by the user with a device-specific username generated using from metadata of the device. This ensures that they can keep their own lists in a convenient location, and also helps identify which lists they can and cannot perform CRUD actions on. It does this by sorting the lists by ensuring that the username of the current device matches that of the created list. The username attribute is present in the **AlbumList** class.

**GetById():** This GET method the returns the lists for the **ID** that is supplied by the user. It does this by taking in the ID from the user, and comparing it to the ID present in the lists. It returns the first element if found, and throws a 404 if there is not a match.

**POST Methods:**

**CreateAlbumList():** This POST method allows the user to create an AlbumList. It does this using the username, the name of the list and a description. Only the name of the list and the description need to be supplied by the user for the method to work. It works by essentially creating an AlbumList object based on these parameters.

**AddToList():** This POST method allows the user to add an album to an existing AlbumList. The name of artist and album title are then used in a query to the Last.FM API, using the excellent [Inflatable.Lastfm library](https://github.com/inflatablefriends/lastfm) for .NET Core. If the Album is null (in the case of non-existence, for example), a bad request is returned. Additionally, if the Album is found successfully, it will retrieve the album art for that album automatically.

**DELETE Methods:**

**DeleteList():** This DELETE method simply allows the user to delete an AlbumList. It used to ID to do this. If the ID is found, the list is delete.

**DeleteFromList():** This DELETE method allows the user to delete an Album from an AlbumList. It uses the list ID and the Album ID to accomplish this. If the IDs match up, the album is removed from the list.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Screenshot of Swagger UI Test Page**

Graphical user interface, text, application

Description automatically generated

**Azure App Service Settings**

Although I could not get the API working exactly how I had liken it to, I did deploy to azure SQL database and deployed this app.

A screenshot of a computer

Description automatically generated

**Application Breakdown**

In this section I would have liken to being able to show case the app using the device manager but this could not build and due to time to upload I could not get running. I have got the understanding of this but could just not get this working and would have liken more time.

I created different CRUD methods with buttons that would create new book lists and add new books with the UI for this. I configured the build gradle to my azure database server.

**Conclusion**

In conclusion I would have liken to fully develop this to display on my github. When I finish I will come back to this project to complete and have a working Android App with the backend ASP.NET Core API holding the application together. I would have like to have another member to do this assignment with to bounce ideas of each other and solve problems that occurred and make use of CI/CD on github.