



---

## MEDIA RELEASE

---



JANUARY 1, 2022  
DUNDALK INSTITUTE OF TECHNOLOGY  
Seainin Keenan

## Acknowledgements

I would like to thank Frances Byrne for helping with this project and guiding me on what was required for the project documentation.

I would also like to thank my lecturers, classmates, friends, and family for helping out with the project

## Declaration

I hereby declare that the work described in this project is, except where otherwise stated, entirely my own work and has not been submitted as part of any degree at this or any other Institute/University.

---

Seainin Keenan, 2022

## Abstract

The main objective of this project was to create an application/website which will track new releases of a different variety of entertainment media, i.e., movies, books, games, and shows. Users will be able to add different entertainment media to their in-app calendar; The user receives a notification when their media is released, by either phone or email. The user will use a mobile application that interacts with an API for media related data and a Firestore database for user related data.

## Table of Contents

Acknowledgements .....	i
Declaration .....	ii
Abstract.....	iii
Introduction .....	8
Aims .....	8
Objectives.....	8
Literature Review .....	9
Existing Applications.....	9
Hosting Platforms .....	9
Amazon Web Services .....	9
Google .....	9
Azure .....	9
Technologies .....	10
What kind of app will be developed?.....	11
Programming Platforms .....	11
Programming Languages.....	11
Web Scraper.....	12
Design.....	13
Brief Overview of Project.....	13
APP .....	13
API .....	15
Model .....	15
Interface .....	16
Controller .....	16
Data .....	16
Azure SQL Database.....	17
Models/schemas .....	17
Stored procedures.....	18
Triggers.....	19
Web Scraper.....	19
Push Notifications .....	19
Implementation.....	21
API Development.....	21
Timeline .....	21
Evaluation .....	26
APP Development .....	27

Timeline .....	27
Evaluation .....	31
Web Scraper Development.....	31
Timeline .....	31
Evaluation .....	35
Push Notifications .....	35
Conclusions .....	38
Aims.....	38
Objectives.....	38
Future Work.....	38
API .....	38
Web Scraper .....	38
Mobile Application .....	39
Source Code .....	40

Figure 1 Login and Signup Service using Firebase .....	15
Figure 2 Book Model in API.....	15
Figure 3 Book Interface in API.....	16
Figure 4 Get Books Method in API controller.....	16
Figure 5 Get Books Method in API Data Class .....	17
Figure 6 Diagram of Database Schema.....	17
Figure 7 Simplified Diagram of Database Schema .....	18
Figure 8 Delete Duplicates Stored Procedure.....	18
Figure 9 Trigger for Stored Procedure .....	19
Figure 10 Movie Model in API .....	21
Figure 11 Part of Movie Controller.....	22
Figure 12 Book Interface .....	22
Figure 13 Tables in Database .....	23
Figure 14 API Migration to Database.....	23
Figure 15 Using Postman to Call API .....	24
Figure 16 Using Postman to Add to API .....	25
Figure 17 Cors in API .....	25
Figure 18 Adding Images to API using Json Form.....	25
Figure 19 One to Many Model .....	26
Figure 20 Date Entity Frameworks .....	26
Figure 21 Data Interfaces .....	26
Figure 22 Feed SQL Interface .....	26
Figure 23 Ionic Pages .....	27
Figure 24 Adding Firebase to Enviroment .....	28
Figure 25 App Module .....	28
Figure 26 Login Page Function .....	28
Figure 27 Firebase Login and Signup.....	29
Figure 28 Signup Function .....	29
Figure 29 Ionic Login & Signup Pages .....	29
Figure 30 API Call .....	30
Figure 31 Cors Error in Browser .....	30
Figure 32 Ionic More Details Function .....	30
Figure 33 Ionic Feed Page, First Implementation .....	30
Figure 34 Ionic Feed Page .....	31
Figure 35 Websites to be Scraped .....	32
Figure 36 Create Chrome Driver .....	32
Figure 37 Webscraper Parser Function .....	32

Figure 38 For Loop Parser .....	33
Figure 39 Python Pandas .....	33
Figure 40 Python Imports .....	33
Figure 41 Change Date Function.....	33
Figure 42 Add to Database Function .....	34
Figure 43 Stored Procedure .....	34
Figure 44 RDS Prototype .....	35
Figure 45 Set up Cloud Formation.....	36
Figure 46 Teardown Cloud Formation .....	36
Figure 47 SNS Python Functions .....	36



## Introduction

This project looks at designing and implementing an application that can be used as a centralized point to find release dates for movies, games, and music. First, users will be able to create an account on the ionic mobile application. Once logged in, they will be able to browse a list of movies, games, shows, and books released or will be released soon. Then, the Media API supplies the mobile application with data for the media release dates. Finally, a web scraper fills and adds the media data to a SQL database connected to the media API.

## Aims

1. Design a mobile application that will allow users to access entertainment release dates.
2. Implement the application backend on a cloud platform.
3. Design a database to maintain user and media information.
4. Use a web scraper to find release dates automatically on the web.
5. Implement push notifications.

## Objectives

1. Understand the tools and applications used to develop mobile applications
2. Research cloud platforms and the different services provided, i.e., SQL database, lambda functions, virtual machines, and notification services.
3. Research database options such as SQL and NoSQL
4. Research the operation of web scrapers to find information on the Internet.

## Literature Review

### Existing Applications

Are not any mobile applications to keep track of different media releases. However, multiple websites show when specific media releases, i.e., movie releases or game releases. They had the information but no straightforward method to keep track of them. They can be tracked manually on a calendar app. There are multiple websites to find information on media releases, including a list. All have their way of showing the data. For example, games and books have a higher quantity of releases compared to movies and shows. So, the information is there, but there could be an easier way to follow these media, mainly if someone follows multiple media, it can be hard to keep track of them.

### Hosting Platforms

#### Amazon Web Services

Amazon has a lot of simple services it provides, including Simple Notification Service (SNS), Simple Queue Service (SQS), Simple Storage Service(S3), and Elastic Compute Cloud (EC2) instances. All these services can be used independently from one other. However, they require some knowledge to set up and can get confusing. Some of the services I might use include SNS, Relational Database Service (RDS), EC2, and Lambda. SNS is used to push notifications to other services in different formats. These formats include text and email. SNS has clients subscribe to a topic that can then publish/send messages to all clients who subscribe to that topic. RDS is an amazon used to set up different databases, including Microsoft SQL Server, MySQL, aurora, and PostgreSQL. The Media API will store all information in an SQL database, which amazon could provide. EC2 is used to create virtual machines in the cloud. There are many possibilities when using virtual machines, including hosting websites/services, and could benefit any project that works with the cloud. Finally, Lambda provides a service to run code without needing a server.

#### Google

Google also provides cloud services, like AWS. Firebase is a platform provided by Google to help create and develop mobile and web applications. They offer both a real-time and a Firestore database. A real-time database is Firebase's original database. It is low latency compared to the Firestore Database. The Firestore database is a newer database that uses faster queries and can scale better than the real-time database (Firebase Documentation, 2022). The Firebase SDK/software development kit can be used with multiply languages like C++, Java, and JavaScript. Firebase is widely used with JavaScript frames like angular and react

#### Azure

Microsoft provides Azure's web services. Azure provides SQL databases both on a server and serverless like the other services. Although Microsoft also developed Visual Studio (VS). Azure web services are easily integrated and used in VS

applications. In addition, VS provides much help when deploying services, making interaction with VS convenient.

## Technologies

Ionic is an open-source software development kit used to develop hybrid apps, i.e., web applications that are put into a native app shell. They help web designers create fully functional mobile applications by allowing the developer to use HTML, CSS, and JavaScript frameworks to create applications and use mobile functionality (Ionic, 2022). Ionic was founded in 2012 by Ben Sperry and Max Lynch. Ionic sits in a Cordova WebView. Cordova allows Ionic to use mobile device functions, i.e., contacts, and emails. Cordova was developed by Apache (Griffith, 2022)

Angular is a JavaScript framework used to create web applications. It is used as a front-end framework. Helps to create web pages with functionality. It is written mostly in typescript. Angular was developed by Google on September 14, 2016.

Bootstrap structures web pages and allow websites to be responsive. However, Bootstrap can cause trouble with ionic. They both manipulate the DOM. The Bootstrap node package allows Ionic to use Bootstrap. It uses HTML and CSS to do this. Bootstrap splits a webpage into 12 columns, and each HTML element can be formatted using these columns. Bootstrap also supplies more effortless scalability in terms of screen size, i.e., changing the look of the webpage on computer screens and mobile devices

Node Js executes JavaScript code on the server-side. However, it also runs on the front end of the web application. Node Js computes requests on a single thread instead of using a pool that can run out of connections in the connection pool. Node Js handles requests on a single thread. It can handle thousands of requests. (Capan, 2022)

.Net Core is used to build web apps and services in C# and other languages. .Net Core is the latest version of the .Net framework, a software development framework to build and run applications on windows. Microsoft developed the .Net framework

*Selenium* is a tool used to perform automated tasks on a browser, including testing and automation on a web browser. Many web scrapers use selenium to execute a web page rather than download the web page HTML. Because it can execute the JavaScript on a page, it provides all the availability to a program that a person sees when browsing a webpage. Selenium provides a chrome extension and selenium libraries for different programming languages. A web driver is needed to use selenium on a local machine. the web driver interacts directly with a browser and uses the browsers engine to control the webpage. (tutorialpoint, 2022)

Git is a version control system. It helps keep track of files & folders and stores them in the cloud. Git handles any size project, from enterprise projects to single files. Git was released in April 2005. It makes use of a bash command line to control different resources. GitHub and GitLab are two internet hosts that make use of git.

## What kind of app will be developed?

Ionic provides the capability to create apps on IOS, Android, and the web. Each platform required minor changes to be compatible with ionic. This app will be developed for android as android is more popular than IOS (Statista Research Department, 2022), although there are benefits to developing with both. This app is a utility app with a few social media options. It provides the user with a reminder of when a media is released.

## Programming Platforms

Visual Studio, Visual Studio Code, and Replit are all environments that are used to develop applications/products/code. In addition, they help create a wide range of projects in different languages.

Visual Studio is an Integrated Development Environment used to develop websites, web apps, and APIs. Microsoft developed visual Studio and integrated its cloud services into the IDE, making deploying an application created in Visual Studio easier and integrating Azure services into the applications.

Visual Studio Code is a code editor. It provides an area where code is written and edited, and it supports multiply different coding languages, with the ability to add extensions, which can help format code to a designated format or add an animated cat to the code-editor

Replit is an online IDE based in San Francisco. Supports multiply languages including C, C#, Machine Assembly, Node Js, and Python. Replit can be used to develop applications on the web, suited for weaker laptops, or if the developer moves around a lot, can save CPU and ram. Although Replit does not provide private projects by default, there is a payment option to increase CPU and ram limits, more storage, and make projects private. The free tier provides the user with the tools to create projects, including importing libraries, some drivers, and compiling code in the browser.

## Programming Languages

JavaScript allows developers to create front-end websites. It allows web pages to be more interactive while browsing. A *high-level programming language* is a language that can be easily read by humans and is abstract from the computer running it (Beal, 2022).

C# can be used with .Net core to create web applications and APIs. It is also a high-level programming language.

Typescript allows developers to add type support to JavaScript, creating classes and objects. Some JavaScript frameworks are written in typescript, including Angular and Vue, while react supports it. Typescript simplifies JavaScript code to read and modify more easily.

HTML is a language used to design displays for websites in a browser. CSS assists in formatting HTML pages. They care about a web page's design and what the user sees when interacting with a web page.

JSON is a format used to store data. It helps store objects in a key-value pair and lists of different objects. It can be easily read and written by humans. JSON stands for JavaScript object notation. It can be beneficial for sending data across HTTP

Python is a high-level, general-purpose programming language. It can be lightweight but not used for mobile or game dev because slow. Python was developed to be easily read by humans (geeksforgeeks, 2022).

SQL stands for Structured Query Language. It is used to query databases. It is capable of other processes such as stored procedures which automatically perform SQL queries on a database

## Web Scraper

Web scrapers are used to get/scrape data from websites, getting the data from html elements or browsing webpages.

Web scrapers are used to get/scrape data from websites by getting the data from HTML elements or browsing webpages.

Creating a web scraper is C++ with gumbo or using python and selenium to interpret dynamic web pages. Basic formatted of a python web scraper would be. First, import libraries into python file, then initiate selenium driver and add parameters, then load the webpage and query the elements of the webpage.

## Design

### Brief Overview of Project

The client will have a mobile application in which they can create an account and log in. The main page will be a feed that will display a list of different types of newly released or upcoming media, including movies, games, books, and shows. The user will then add the media to their in-app calendar. The user will also be able to message other users and review media. The mobile application will store its user data in a Firestore database and other information specific to the user, i.e., the calendar entries. Users will be notified when an upcoming media is released via text or email.

The mobile application will get the media data from an API (Application Protocol Interface). This API will store its data in a SQL database on the cloud. The API will perform CRUD (Create, Read, Update, Delete) on the data in the SQL database. The API will have multiple endpoints, conform to the REST API criteria, be stateless, and be designed using the model view controller pattern. Splitting all the data in models and used by their controller/endpoints to improve the readability of the code.

A web scraper will fill the SQL database with media. The web scraper will use python to scrape specific websites and store the data in the SQL database. The libraries used will be selenium, pyodbc, UUID, web driver manager, and Date Time, and the drivers used are chrome web driver and ODBC Driver 18 for SQL Server. pyodbc python library allows python to communicate with an MSSQL database using the ODBC Driver for SQL Server. The web driver's manager library downloads the required web driver for selenium when the python script is run. UUID Library creates universally unique identifiers, like the guide, globally unique identifiers.

### APP

The mobile application will be written in angular, HTML, CSS, and Ionic. The mobile application will communicate with the mobile using Cordova. The following pages will be added to the mobile application:

#### *Login*

Allow a user to log in if they already have an account using fire auth. The HTML page will give the user an option to log in with their email and password or sign up if they do not have an account. Firebase will store the user data, which will allow the mobile application to keep track of users, i.e., using the user data to keep track of followed media

#### *Signup*

Allow a user to create an account that they will be able to sign up with stored in a Firestore database. An email, username, and password are required to make an account.

#### *Feed*

The application would have had a separate page to list each media based on type, i.e., movies, games, shows, and books. Instead, the application will use a single feed

page that will display a list of all media, making the application more engaging by decreasing the number of pages the user needs to use. The user will also have the option to search by media type and by media name, date, theme, genre, or actor.

The media API will supply the media entries for the media app. When a request is made to the API, the API will return a list of media ordered by release date.

If a user wants more information on a media, they will be able to click on a media, bringing the user to another page with more information on the media.

### *Forum*

The forums page will have a forum for each media in the MediaAPI, which should be automatically generated by either the media API, the ionic app, or a separate program. The user will be able to create posts on forums. The user will also be able to comment on posts.

### *Messages*

The messages page will allow the user to message other users. The user will be able to search for and open a chat page with messages and receive messages from other users. The messages will be stored in a Firestore database along with the sender and receiver. The user will also have an option to block other users.

### *Profile*

The profile page will allow users to create a profile. The profile will include a username, a profile picture, a short description, and a list of media they like/follow.

### *Sidenav*

The application will use a Sidenav to organize all the pages into an easily accessible side menu that will be displayed on every page to allow the user to access all other pages.

The following services will be added to the mobile application:

### *Chat*

The messages page will use this service, which will allow users to add message data to the Firestore database. The service will also be able to get data from the database and delete data if they are the owner/creator of it.

### *Forum*

This will be like the chat service. The forum page will use the forum service. The forum service will be able to add post and comment data, get post and comment data and delete post and comment data by the owner/creator

### *Media/Feed*

Service to allow the app to use the MediaAPI. Allow feed app to create dynamic requests to get specific information from the API, including actors, themes, genres, sellers, and media creation properties. These requests will return a list of media associated with the above parameters.

```

export class FireserviceService {
  constructor(public firestore: AngularFirestore, public auth: AngularFireAuth) { }

  loginWithEmail(data) {
    return this.auth.signInWithEmailAndPassword(data.email, data.password);
  }

  signup(data) {
    return this.auth.createUserWithEmailAndPassword(data.email, data.password);
  }

  saveDetails(data) {
    return this.firestore.collection("users").doc(data.uid).set(data);
  }

  getDetails(data) {
    return this.firestore.collection("users").doc(data.uid).valueChanges();
  }
}

```

Figure 1 Login and Signup Service using Firebase

## Firestore

Allow users to use Firestore to save login and signup detail.

The login and signup page will make use of this service.

The service will make use of the Firestore Auth service.

As shown in Figure 1, there is

a 'login with email' method and a 'signup' method. The other functions will save the

login/signup data in a collection and get the data from the collection.

## Notifications

The notification service will send an API request to the media API to add a user's email, number, and the media they wish to follow. This data will be saved in a SQL table. The media topic ARN will also be saved in the table.

## API

There will be many different parts of this media API. There are not as clear to split as the media App. But here is a list of the different components of the Media API

## Model

A Model is representation of data (techopedia, 2022). The data model will allow the API to use the data retrieved from a different source (SQL database). The model is a way for C# to understand the data. We can migrate this model into an MSSQL server using the Microsoft Entity Framework package. As Shown in Figure 2, The model is a class that stores different objects together.

The models will be like the Figure 2 and include movies, books, and shows. All media will also have a series, with SeriesId being the foreign key. Most of the data will be a string of information with the release date stored and represented as a date in the model and SQL database.

```

namespace MediaApi.Models
{
    ///in dbset
    public class Book
    {
        [Key]
        public Guid MediaId { get; set; }

        1 reference
        public Guid SeriesId { get; set; }

        0 references
        public Guid CreatingPropertyId { get; set; }

        [Required]
        1 reference
        public String MediaType { get; set; }

        [Required]
        1 reference
        public String Title { get; set; }

        1 reference
        public String ImageName { get; set; }

        0 references
        public String Description { get; set; }

        0 references
        public int NumberofTimesSearched { get; set; }

        0 references
        public int Length { get; set; }

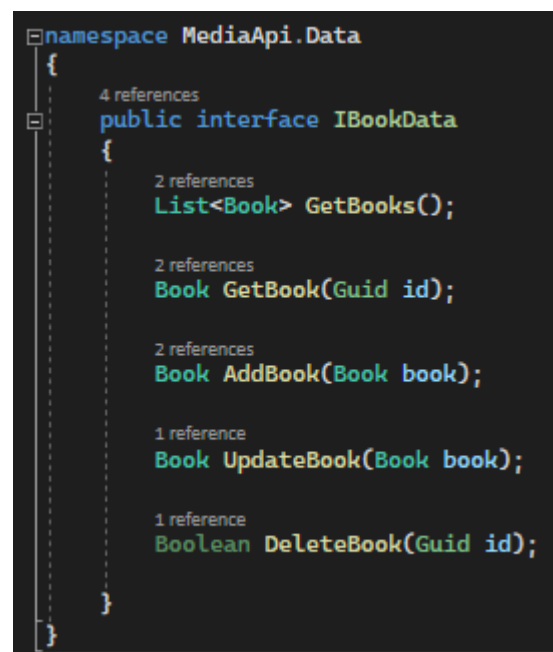
        [Required]
        [Column(TypeName = "Date")]
        0 references
        public DateTime ReleaseDate { get; set; }
    }
}

```

Figure 2 Book Model in API



## Interface



```

namespace MediaApi.Data
{
    4 references
    public interface IBookData
    {
        2 references
        List<Book> GetBooks();

        2 references
        Book GetBook(Guid id);

        2 references
        Book AddBook(Book book);

        1 reference
        Book UpdateBook(Book book);

        1 reference
        Boolean DeleteBook(Guid id);
    }
}

```

Figure 3 Book Interface in API

*Interfaces* are methods that classes must implement. Interfaces are used when a controller is interacting with a data class. The interface allows a coming protocol meaning both controller and data files are using the same method format. In addition, interfaces allow us to change the class the controller interacts with to another class that uses the same interface.

In C#, this can be done by adding a scope to the service in the startup.cs file. Which tells the controller which class using the interface it should use.

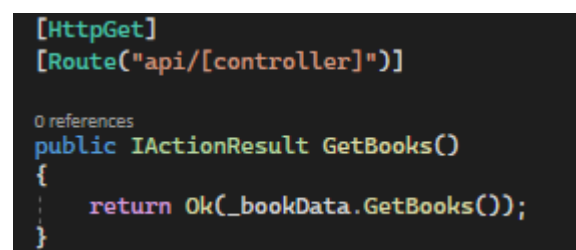
Keep track of the GetBooks() class.

Notice it returns a list of books, the model of the book is shown in Figure 3, and does not require any

parameters

## Controller

Controllers control how a user interacts with a system; this can be by a webpage, where the controller will help the users by showing the correct view when the user requires it. The Media API makes use of controllers as the endpoint for the API. The controller allows the developer to allow different requests and handle these requests. For example, as shown in Figure 4, this controller allows a get request on the book controller. The GetBooks controller method returns the GetBooks method from Figure 4. `_bookData` is the interface the controller is currently using.



```

[HttpGet]
[Route("api/[controller]")]

0 references
public IActionResult GetBooks()
{
    return Ok(_bookData.GetBooks());
}

```

Figure 4 Get Books Method in API controller

## Data

The Data classes in this API are used to get data from the Azure SQL database. The `System.data.sqlclientpackage` provides C# with functions to query an SQL database.

For example, the GetBooks method returns a list of book objects from the

SQL database.

```
2 references
public List<Book> GetBooks()
{
    return _allContext.Books.ToList();
}
```

Figure 5 Get Books Method in API Data Class

Entity Frameworks allows developers to create databases off of C# classes. Moreover, it provides the tools to query the database.

## Azure SQL Database

### Models/schemas

Below is a diagram of the MediaAPI database used in this project. The database was created using Entity Frameworks and the Media API models. This is the database used by the media API and the python web scraper

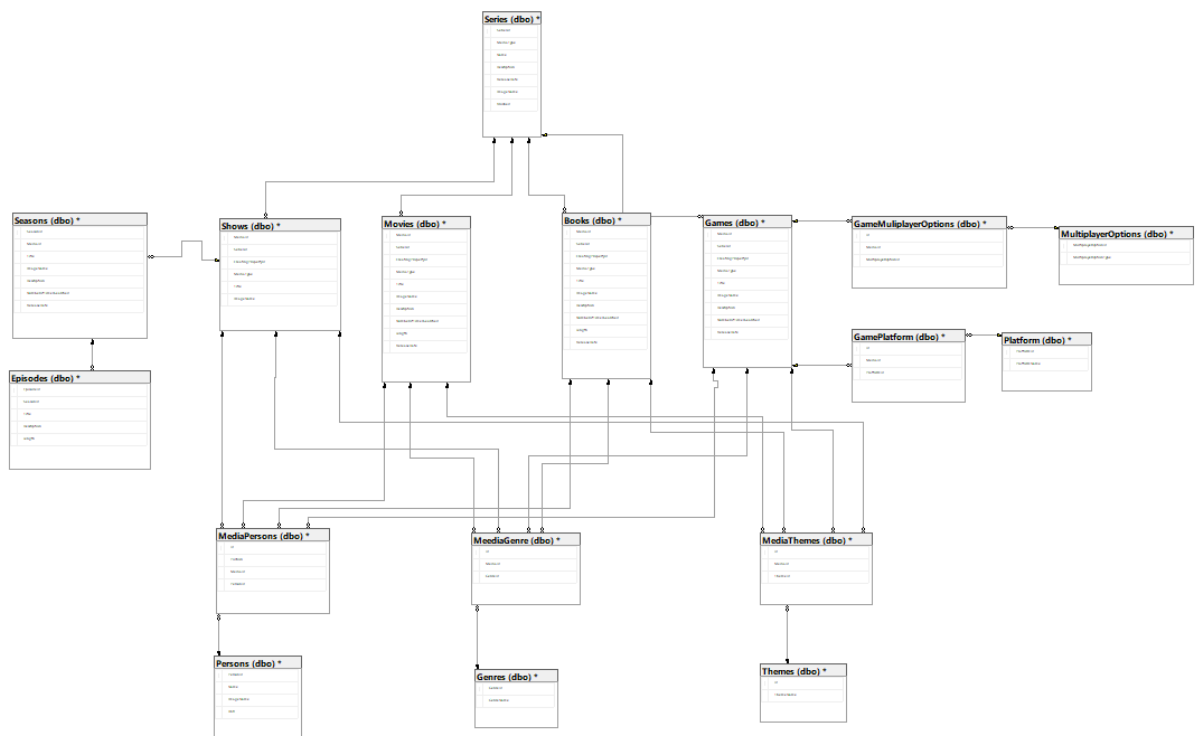


Figure 6 Diagram of Database Schema

Shown in Figure 6 is a diagram of the SQL Database. Figure 7 simplifies this schema while still containing all the critical parts. There are four different media types; movies, books, shows, and games. The games and season tables both have some other tables associated with them. Games have a many-to-many connection with Platforms. Each game can be played on multiple platforms, and each platform can play multiple games. Therefore, a middle table was created that records all the games and the platforms they can be played on called GamesPlatform. Shows have a many-to-one relationship with seasons. Each show has multiple seasons, and each season only has one show.

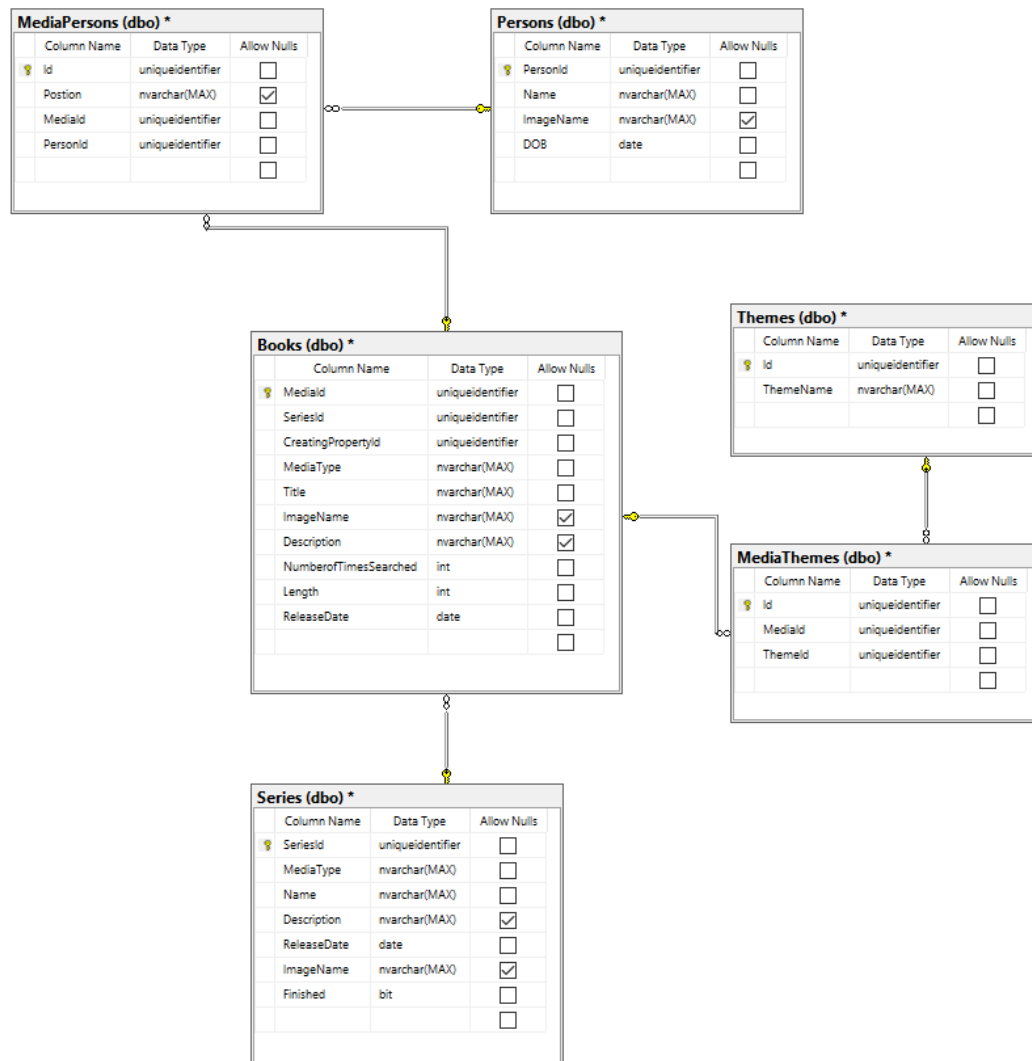


Figure 7 Simplified Diagram of Database Schema

All media contain one series Id linked to the series table in a one-to-many relationship. All media have a many-to-many relationship with genres, themes, and persons. They are shown above in Figure 7, a middle table to store all many to many relationships.

### Stored procedures

The SQL database will make use of two different stored procedures. The first is the procedures shown in Figure 8, which deletes all the duplicates in the Books table. There is a procedure to delete duplicates for all four media tables. The procedure selects all media in a table where their title, description,

```

--
ALTER PROCEDURE [dbo].[BooksDeleteDuplicates]
AS
BEGIN
    WITH CTE([title],
             [description],
             [releaseDate],
             DuplicateCount)
    AS (SELECT [title],
              [description],
              [releaseDate],
              ROW_NUMBER() OVER(PARTITION BY [title],
                                         [description],
                                         [releaseDate]
                                  ORDER BY mediaId) AS DuplicateCount
    FROM [dbo].[Books])
    DELETE FROM CTE
    WHERE DuplicateCount > 1;
END
  
```

Figure 8 Delete Duplicates Stored Procedure

and release date are the same. Then every field with a partition count over one is deleted.

The second procedure will delete all media that have been released for over a month. This is to save space in the database. The MediaAPI and media app are based on newly released media and inform users when a new media is released. There are hundreds of websites and applications that allow users to follow media well after its release. Including Reddit and IMDB.

### Triggers

A trigger is needed to use the stored procedure in Figure 9. After a record is inserted into the azure database, the store procedure will delete the duplicates. The web scraper will activate daily, so there will be duplicates of the different media. It will be easier to delete duplicates in the database than use the web scraper or some other resource to check if there is a duplicate and then delete it. The web scraper also sends all insert data in one request so the database can process it as one request. Hence the trigger is only activated once per day

```
ALTER TRIGGER [dbo].[MoviesInsertTrigger]
on [dbo].[Movies]
After INSERT
AS
BEGIN
    EXEC [dbo].[MoviesDeleteDuplicates]
END
```

*Figure 9 Trigger for Stored Procedure*

The second stored procedure will not be able to be activated by a trigger as they only work on insert, delete, and select queries. Not trigger at a set time. A scheduled job will be used to trigger the second stored procedure.

### Web Scraper

The Web scraper will be written in python. It will use the selenium, web driver, and pyodbc libraries. Selenium will load the web pages and get different HTML elements from the webpage. All the information provided by the website that will be useful for the project will be inserted into the SQL database using the pyodbc library.

### Push Notifications

Simple Notification Service (SNS) sends messages via email or text; This will notify the users when a media they follow has been released. The SNS will be run in an AWS lambda function using python. The libraries used were pyodbc, time, and boto3. Boto3 is a python library that allows developers to use AWS services in their python code. In this case, it would be used to set up the SNS. each media will have its topic. Every time a user follows a media, they send an API request to the media API with their email, number, and the media they wish to follow. The API will then add the email and number into a table with the topic Amazon Resource Name (ARN) of the

game they wish to follow. The SQL table will be called 'userSub.' The lambda function will be triggered daily. It will subscribe to all users in the 'userSub' table with the topic that matches their ARN if they are not yet subscribers. Then the function will get all media from the database where the date is today's date. There topic ARN will be used to publish a message to all subscribers, letting them know their followed media has been released. Then all users in the 'userSub' that match that topic will be deleted to clear up the table

## Implementation

Step by step, even or especially when things don't work, it demonstrates how you spent your time.

### API Development

Focus, most time spent

```
namespace MediaApi.Models
{
    //in dbset
    16 references
    public class Movie
    {
        [Key]
        0 references
        public Guid MediaId { get; set; }

        1 reference
        public Guid SeriesId { get; set; }

        0 references
        public Guid CreatingPropertyId { get; set; }

        [Required]
        1 reference
        public String MediaType { get; set; }

        [Required]
        1 reference
        public String Title { get; set; }

        1 reference
        public String ImageName { get; set; }

        0 references
        public String Description { get; set; }

        0 references
        public int NumberOfTimesSearched { get; set; }

        0 references
        public int Length { get; set; }

        [Required]
        [Column(TypeName = "Date")]
        0 references
        public DateTime ReleaseDate { get; set; }
    }
}
```

Figure 10 Movie Model in API

### Timeline

The media project began with the media API, using the API template provided by Visual Studio .net framework. Next, the models, controller, and data folders were added. The model classes were then created with primary fields required by the mobile app. The four different media that were used in this project were movies, games, books, and series. The series was changed to shows later in the project, as the meaning of the series changed. All media can be a part of a series. The series media will be described as shows from now on to avoid confusion). Each media was a subclass of media. Media had similar information found in all media, i.e., name, release date, and description. The subclass movies, for example, had length, stored as an integer, and

actors, stored as a list of actors. The data classes for the media and subclasses were created. To begin with, a data class was created to get data from a method, instead of getting the data from a database. Lists of data were created for each model. This was to test the controllers first without complicating the API with SQL connections and setup.

The controllers were created along with the data classes. Interfaces were used with the data classes to provide a standard interface the controller could use. The add scope method was used to specify with data class the controller used based on the interface.

There was a problem with the computer being used, and I could not access the project. This set me back a few weeks as I had to start the project again on my laptop, including setup.

When I started again on the project, I could design the model/controllers and interfaces a bit better. Moreover, because I had just done the project in the previous version, I could quickly create a new project like the old one.

Implement Entity frameworks and classes, downloaded Microsoft entity frameworks. I used a MySQL Apache server to test out the connection first, which I was able to get working. The steps required were using an MYSQL library to communicate. I had to remove subclasses and create a separate class as they were stored in the database weirdly. I could not find a solution to this at the time but have found that Entity frameworks do support subclasses. Also, because I had lists of objects, i.e., actors in the movies model, the models could not migrate into the database; I decided to remove these lists as they did not integrate into the database.

Adding migrations, the problem with models as all media were children of the media class. It would not work with Entity Frameworks. I tried looking up a solution but could not find one. I separated each child of class media into their separate class, I did not like this, but I wanted to make it work. Later, in development found out you could add children's classes to the database

```
[ApiController]
1 reference
public class MoviesController : ControllerBase
{
    private IMovieData _movieData;
    private static IWebHostEnvironment _environment;

    0 references
    public MoviesController(IMovieData allData, IWebHostEnvironment environment)
    {
        _environment = environment;
        _movieData = allData;
    }

    [HttpGet]
    [Route("api/[controller]")]

    0 references
    public IActionResult GetMovies()
    {
        return Ok(_movieData.GetMovies());
    }
}
```

Figure 11 Part of Movie Controller

```
namespace MediaApi.Data
{
    4 references
    public interface IBookData
    {
        2 references
        List<Book> GetBooks();

        2 references
        Book GetBook(Guid id);

        2 references
        Book AddBook(Book book);

        1 reference
        Book UpdateBook(Book book);

        1 reference
        Boolean DeleteBook(Guid id);
    }
}
```

Figure 12 Book Interface

Problem connecting to Apache database was not sure what format the connection string had to be but then found Azure provided the string like what I have used before. I also had to add DB context in the startup C# file and DB set for all the models.

Another problem was using Apache to test MySQL and using MySQL package with API to create and connect to the database. I tried to connect with azure MSSQL but did not realize why it would not work; I finally coped on realizing that there are different SQL databases.

Creating an SQL database in Azure and setting up the database was simple enough as I had done it before. Just a bit of setup. I called server MySQL because I did not realize there was a difference.

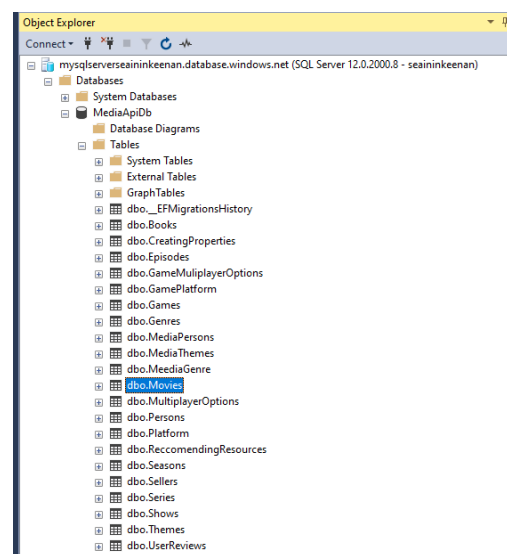


Figure 13 Tables in Database

Start using git and continue to use it for version control; I had a little experience with git with group projects.

Computer fixed, so I started using it again, the same SSD, so no real trouble

Compatibility issues between computer and laptop net5.0 on laptop net6.0 on computer. Not sure how this happened. I think I tried to update my old project using git with my new project. When I got back to my laptop, it would not work, told me I needed a newer version or changed the project version.

Updated laptop

When I tried to connect was rejected because my IP address was not accepted. So added an IP address to the SQL server firewall. It allowed me to add a range, so I added a range for my home pc in case of DNS while adding and

```
1 reference
public partial class InitialMigration : Migration
{
    [reference]
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Books",
            columns: table => new
            {
                MediaId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                SeriesId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                CreatingPropertyId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                MediaType = table.Column<string>(type: "nvarchar(max)", nullable: false),
                Title = table.Column<string>(type: "nvarchar(max)", nullable: false),
                ImageName = table.Column<string>(type: "nvarchar(max)", nullable: true),
                Description = table.Column<string>(type: "nvarchar(max)", nullable: true),
                NumberOfTimesSearched = table.Column<int>(type: "int", nullable: false),
                Length = table.Column<int>(type: "int", nullable: false),
                ReleaseDate = table.Column<DateTime>(type: "Date", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Books", x => x.MediaId);
            });
    }
}
```

Figure 14 API Migration to Database



removing the IP address from the security firewall on the laptop as I would use the laptop on public Wi-Fi at work and college.

I added the Migrations to the MSSQL Azure SQL database.

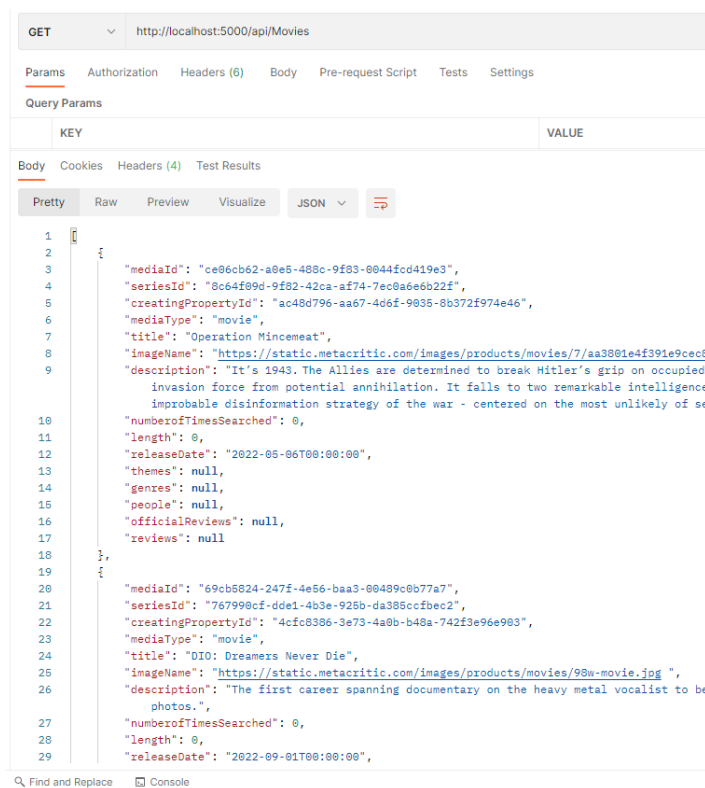


Figure 15 Using Postman to Call API

the SQL database, [NotMapped] ignores class fields for migrations; I used this to ignore lists of objects in the model classes. I also used it with media, as each media has a list of themes and genres and actors associated with them.

Reworked season models from series-season-episode to series-shows-season-episodes. Now had, movies, books, shows, and games were all a part of a series. It is a bit confusing as I changed the meaning of series in the API context.

Use postman to add and get models from API. Tested the API, which helped create and redesign the controllers, data interfaces, and data SQL classes. Used methods provided by data context instead of raw SQL queries. I learned more about HTTP requests and how they are formed. Postman gives much information. Form-data to add multiple bits of data

Used entity framework to add not mapped lists, which would later be used to fill the list in the API. These lists stored lists of objects. I had

an object called series, which would have a media list. I could not see the list in

Used postman to add dummy data to API so there would be data to test the mobile app. So all the data in media models were filled, which turned out different when using the web scraper.

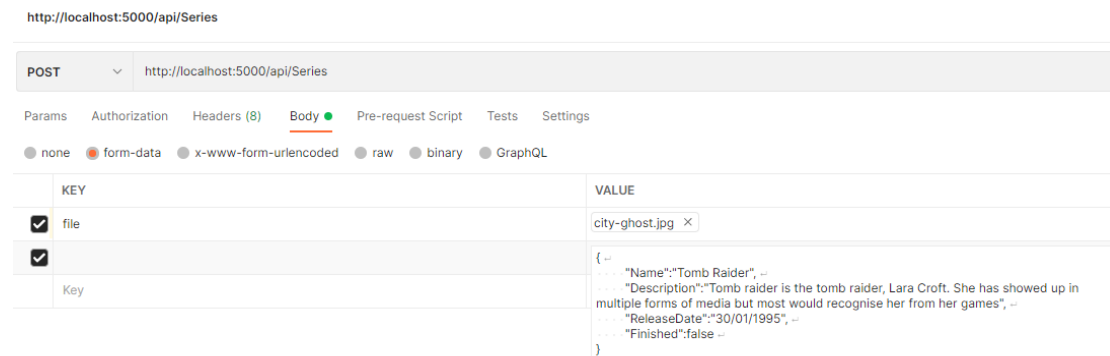


Figure 16 Using Postman to Add to API

I had a problem with the mobile app and API communicating due to Cors. After looking for a solution and finding that it could be solved by adding `app.useCors()` to the C# startup file. Cors allows the API to return the address of the mobile application from the request and adds it to the response.

```
services.AddCors(option => {
    option.AddDefaultPolicy(builder =>
    {
        builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod();
    });
});
```

Figure 17 Cors in API

I also allowed the API to except image, so I could add images to the dummy data.

```
[HttpPost]
[Route("api/[controller]")]
public IActionResult AddMovie([FromForm]String data, [FromForm]FileUploadAPI objFile)
{
    try
    {
        if (objFile.file.Length > 0 && !objFile.Equals(null))
        {
            if (!Directory.Exists(_environment.WebRootPath + "\\Movie\\"))
            {
                Directory.CreateDirectory(_environment.WebRootPath + "\\Movie\\");
            }
            using FileStream fileStream = System.IO.File.Create(_environment.WebRootPath + "\\Movie\\" + objFile.file.FileName);
            objFile.file.CopyTo(fileStream);
            fileStream.Flush();

            Movie movie = JsonConvert.DeserializeObject<Movie>(data);
            movie.ImageName = objFile.file.FileName;
            movie.MediaType = "Movie";
            return Ok(_movieData.AddMovie(movie));
        }
        else
        {
            return Ok("Movie was NOT!! added, no image");
        }
    }
    catch (Exception e)
    {
        return Ok(e.Message.ToString());
    }
}
```

which save the images in a folder in the project. They can be accessed on the web when the API is deployed.

```
app.UseStaticFiles();
```

Figure 18 Adding Images to API using Json Form

Added one too many tables to connect media with lists of people, themes, and genres in SQL database, although only themes are used now as not all data was gotten from the websites scraped.

```

namespace MediaApi.Models
{
    // In dbset
    // 7 references
    public class MediaPerson
    {
        [Key]
        // 1 references
        public Guid Id { get; set; }

        // 1 references
        public string Position { get; set; }
        [Required]
        // 1 references
        public Guid MediaId { get; set; }
        [Required]
        // 1 references
        public Guid PersonId { get; set; }
    }
}

```

Figure 19 One to Many Model

```

[Required]
[Column(TypeName = "Date")]
// 0 references
public DateTime ReleaseDate { get; set; }

```

Figure 20 Date Entity Frameworks

Update models to store the date as a date, instead of a string, in the models and database instead of a string. I removed dummy data when I created and got the web scraper working. However, used kept reading that I should never store dates as strings in a SQL database. I also thought I would have a problem in the future if I tried to sort/order by date.

Updated models to store extra data that could be gotten from the web scraper. Deleted dummy data/decided not to use API to add to SQL database and add straight to database. It would have required a lot of repetitive classes connect the python to the API and then the database

```

Data
├── Interface
│   ├── IBookData.cs
│   ├── ICreatingPropertyData.cs
│   ├── IEpisodeData.cs
│   ├── IFeedData.cs
│   ├── IGameData.cs
│   ├── IGameMultiplayerOptionData.cs
│   ├── IGamePlatformData.cs
│   ├── IGenreData.cs
│   ├── IMediaGenreData.cs
│   ├── IMediaPersonData.cs
│   ├── IMediaThemeData.cs
│   ├── IMovieData.cs
│   ├── IMultiplayerOptionData.cs
│   ├── IPersonData.cs
│   ├── IPlatformData.cs
│   ├── IRecommendingResourceData.cs
│   ├── ISeasonData.cs
│   ├── ISellerData.cs
│   ├── ISeriesData.cs
│   ├── IShowData.cs
│   ├── IThemeData.cs
│   └── IUserReviewData.cs

```

Figure 21 Data Interfaces

Added feed API to get multiple media with one API call. This would merge all media. The database is already ordered by date. I thought this method was better than sorting them while in an API response request. Cut downtime to process requests.

```

// 2 references
public List<object> GetAllMedia()
{
    List<object> list = new List<object>();
    list.AddRange(_allContext.Movies.ToList());
    list.AddRange(_allContext.Books.ToList());
    list.AddRange(_allContext.Games.ToList());
    List<Show> shows = _allContext.Shows.ToList();
    foreach (Show show in shows)
    {
        show.Seasons = _allContext.Seasons.Where(x => x.MediaId == show.MediaId).ToList();
    }
    list.AddRange(shows);
    return list;
}

```

Figure 22 Feed SQL Interface

## Evaluation

After designing and implementing the API, one of the biggest problems was that the models and database were design using hypothetical data. The models were design with what data I would like to display in the application instead of the data that I would be able to obtain. If I had a chance to start again or start a similar project I would design and implement the web scrapper first to figure out what data, I can acquire.

The design of the API will allow me to scale easily. All models, controllers, and data classes are formatted in a consistent way. It will be easy to add other models and

controller to the API. The code for the API is also well structured and requires minimal refactoring, that I am aware of.

I wasted a lot of time because, 1, I designed the models off what data I thought I would need and 2, not having all the models or controllers designed before beginning to program. I started programming first and designing the API while I was writing it.

The API does achieve what I needed it to. I connect to the database and returns that data to the API calls. Although the API calls to take too long to process. This is because the API gets all data from the database, which is impractical. Will need to use pagination to make the API calls quicker.

## APP Development

Took backseat because I thought other parts were more important and that the app was fickle in design, could put a lot of work in and decide to change it, which I did do

## Timeline

Generated application. Generated using ionic, I tried using different templates, including sidebar and tab. I deleted and restarted as I kept running into problems when I changed some aspects of the routing. So, delete them instead of figuring out the problem.

Generated tabs. Decide to use this template. Have a tab for each media\*\*\*media list page for each media type\*\*\*\*. This was later changed to a feed that would get all media with a parameter to search specific media.

I added a page separately for each media. At the start, this was all I thought I needed. And a calendar. I did not do much after that.

I had a problem with my personal computer and could not use it for two weeks. This happened near the start of development, so it was not too big of a problem. I was able to use a laptop in the meantime.

Computer fixed.

I experienced compatibility issues between laptop and computer. The laptop had up-to-date ionic, while the computer had an older version (December). The used version of the computer on both. So downgraded the laptop to ionic 5.

After, I reworked the application to use a single media page instead of a separate page for each media. Added calendar, details-media, forums, login, signup, media/feed, messages, profile, and side nav pages. A placeholder was created for each of these pages.

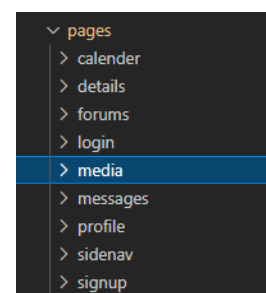


Figure 23 Ionic Pages

The calendar page would have a calendar using a module specifically for android. Details-media was to bring up more detail on a media when clicked on.

Created a login and signup page utilizing the Firestore

database. I added the firebase API information into environment.ts and added the AngularFiremodule.

```
export const environment = {
  production: false,

  firebase: {
    apiKey: "my own data",
    authDomain: "",
    projectId: "",
    storageBucket: "",
    messagingSenderId: "",
    appId: "",
    measurementId: ""
  }
};
```

Figure 24 Adding Firebase to Enviroment

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
import { SplashScreen } from '@ionic-native/splash-screen/ngx';
import { StatusBar } from '@ionic-native/status-bar/ngx';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';

import { HttpClientModule } from '@angular/common/http';

import { AngularFireModule } from '@angular/fire/compat';
import { AngularFireStoreModule } from '@angular/fire/compat/firestore';
//import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { environment } from 'src/environments/environment';

@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports: [AngularFireModule.initializeApp(environment.firebase), AngularFireStoreModule, BrowserModule, IonicModule.forRoot(), AppRoutingModule, HttpClientModule],
  providers: [SplashScreen, StatusBar, { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Figure 25 App Module

Created Firestore service to login in sign up and get user information. Sign up = the information email, password and generated Uuid is added to a Firestore database. Then a user could sign up and login successfully.

```
login(){
  this.fireService.loginWithEmail({email:this.email,password:this.password}).then(
    res=>{console.log(res);
    if(res.user.uid){
      this.fireService.getDetails({uid:res.user.uid}).subscribe(res=>{
        console.log(res);
        this.router.navigate(['/sidenav/nav/calender'])
        //alert('Welcome '+res['name']);
      },err=>{
        console.log(err);
      });
    }
  )
}

signup(){
  this.router.navigate(['/signup']);
}
```

Figure 26 Login Page Function

```

export class FireserviceService {

  constructor(public firestore: AngularFirestore, public auth: AngularFireAuth) { }

  loginWithEmail(data) {
    return this.auth.signInWithEmailAndPassword(data.email, data.password);
  }

  signup(data) {
    return this.auth.createUserWithEmailAndPassword(data.email, data.password);
  }

  saveDetails(data) {
    return this.firestore.collection("users").doc(data.uid).set(data);
  }

  getDetails(data) {
    return this.firestore.collection("users").doc(data.uid).valueChanges();
  }
}

```

Figure 27 Firebase Login and Signup

```

signup(){
  this.fireService.signup({email:this.email,password:this.password}).then(res=>{
    if(res.user.uid){
      let data = {
        email:this.email,
        password:this.password,
        name:this.name,
        uid:res.user.uid
      }
      this.fireService.saveDetails(data).then(res=>{
        alert('Account Created!');
        this.navCtrl.navigateForward('login')
      })
    }
  },err=>{
    alert(err.message);
  })
}

```

Figure 28 Signup Function

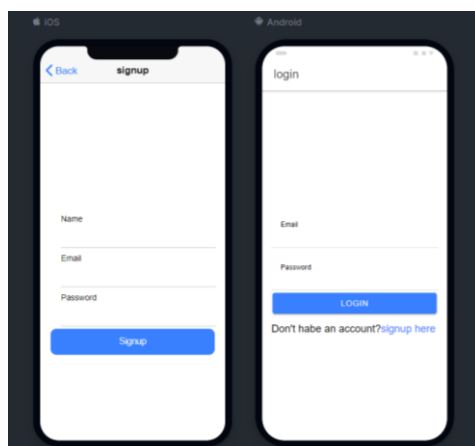


Figure 29 Ionic Login & Signup Pages

Figure 27 is of the services to login users in or sign them up. It also returns user data to be kept in the app and allow it to track what users follow

forums and messages were placeholders that were going to be used to message other users on the app and create posts on different forums

I had problems with sidenav. New releases of ionic quickly made it hard to find a solution for the ionic version I was using. Different methods to navigate ionic between ionic 4, 5, and 6. So it was hard to find a tutorial for ionic 5. I decided to move on to more critical parts of the project. I was still having trouble with it. I tried using methods from ionic 4 and 6. Furthermore, I watched many tutorials, and I could only

get it to work once and only once. I think it is something to do with the folder placement or the sidenav only loading once as all pages are loaded inside sidenav. Can figure out a solution. Low priority because

I tried adding the media API I created as a service in ionic, but I could not get it working, so I moved the request to the feed page. The problem was not in the service. It was with how is used the observable. I have not movies the service back yet, but I could see how that could be useful.

```
data = [];
results: Observable<any>;

constructor(public navCtrl: NavController, public httpClient: HttpClient, private route: Router, private alertController: AlertController) { }

ngOnInit() {
  this.results = this.httpClient.get('http://localhost:5000/api/feed');
  this.results
    .subscribe(data => {
      console.log('my data: ', data);
    })
}
```

Figure 30 API Call

When I got the API connected, I ran into another problem: Cors. Cross-origin resource sharing is a mechanism that restricts HTTP requests from scripts. From what I understand, it sends a request to an API, the API returns the response, and if the domain is not the same as the request, the response is rejected

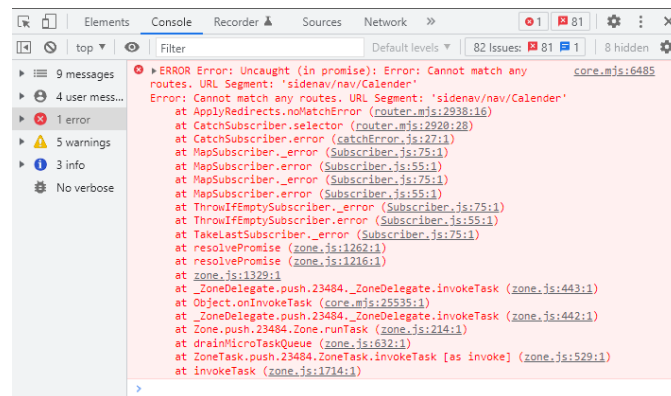


Figure 31 Cors Error in Browser

I tried setting up requests to get rid of the error, but they did not work. So, I found a solution to fix this Cors problem on the API instead.

Added functionality on media.page.html to show data received from the API. Also added the ability to get more details by saving the tapped media in a state and bringing

```
<ion-content>
  <ion-list>
    <ion-card button *ngFor="let item of (results | async); let i = index" (click)="moreDetails(item)">
      {{item.title}}
      <br>
      {{item.releaseDate}}
      <br>
      {{item.mediaType}}
      <br>
      <ion-img src="{{item.imageName}}"></ion-img>
    </ion-card>
  </ion-list>
</ion-content>
```

Figure 33 Ionic Feed Page, First Implementation

that state to another page where more information was show as well as more medias in the same series as the current media

I tried using different if statements in the HTML to sort the list of data into different media types.

```
moreDetails(media: any){
  let navigationExtras: NavigationExtras = {
    state: {
      media: media
    }
  };
  this.route.navigate(['movie'], navigationExtras);
}
```

Figure 32 Ionic More Details Function

```

<ion-card button *ngFor="let item of (results | async); let i = index" (click)="moveDetails(item)">
  <div *ngIf="item.mediaType=='Movie'" class="container">
    <div class="image">
      <ion-img src="http://localhost:5000/{{item.mediaType}}/{{item.imageName}}"/></ion-img>
    </div>
    <div class="title movie">
      <h3>{{item.title}}</h3>
      <h5>{{item.mediaType}} {{item.releaseDate}}</h5>
    </div>
    <div class="Description">
      {{item.description}}
    </div>
  </div>
  <div *ngIf="item.mediaType=='Game'" class="container">
    <div class="image">
      <ion-img src="http://localhost:5000/{{item.mediaType}}/{{item.imageName}}"/></ion-img>
    </div>
    <div class="title game">
      <h3>{{item.title}}</h3>
      <h5>{{item.mediaType}} {{item.releaseDate}}</h5>
    </div>
    <div class="Description">
      {{item.description}}
    </div>
  </div>
  <div *ngIf="item.mediaType=='Book'" class="container">
    <div class="image">
      <ion-img src="http://localhost:5000/{{item.mediaType}}/{{item.imageName}}"/></ion-img>
    </div>
    <div class="title book">
      <h3>{{item.title}}</h3>
      <h5>{{item.mediaType}} {{item.releaseDate}}</h5>
    </div>
    <div class="Description">
      {{item.description}}
    </div>
  </div>
  <div *ngIf="item.mediaType=='Show'" class="container">
    <div class="image">
      <ion-img src="http://localhost:5000/{{item.mediaType}}/{{item.imageName}}"/></ion-img>
    </div>
    <div class="title show">
      <h3>{{item.title}}</h3>
      <h5>{{item.mediaType}}</h5>
    </div>
  </div>
  <ion-list>
    <ion-card class="container" *ngFor="let season of item.seasons">
      <div class="image">
        <ion-img src="http://localhost:5000/Season/{{season.imageName}}"/></ion-img>
      </div>
      <div class="title season">
        {{season.title}}
      </div>
    </ion-card>
  </ion-list>
</ion-card>

```

As each media had different bits of information, this did not matter too much as the information will not show up if it does not have data. Information is wanted on the feed page. The problem came when I tried to display the images. The page failed to load as the image URLs store using just their name, i.e., picture.jpeg. I added the URL in the ionic app. Each Image was stored in a folder in the wwwroot file according to its media. So, I had to check what type the media was to determine the folder in which its Image was stored. I could have prevented this by adding the full URL straight into the database, which I did later when using the data from the web scraper. The user interface did not look too good after I spent a

reasonable amount of time developing it, and it did not look too pretty in the code either, so I deleted it and decided to use bootstrap.

Later, I found that ionic and bootstrap do not work well together as both use the DOM in different ways. However, found out after that Ionic has a bootstrap package.

## Evaluation

I found ionic to be useful in that it has a lot of modules that can be used to enhance the application, saving me from programming everything from scratch. I found CSS, HTML, and a lot of ionic to be tedious. Angular Typescript can be hard to understand, I do not find the syntax to be easy to read. Although I can see why it is used as it can be extremely powerful when creating and operating websites, front-end and backend.

I was not able to complete the Ionic application as much as I would have liked. I wanted to have the backed i.e., the API, Database, and web scraper, to be operating properly before I created the user interface. The object of the application and project was to allow users to follow media releases and be notified when they are released. I believe that the backend was more important than the Ionic app as without the backend the Ionic app does not accomplish the goals of this project. The app is important but took a backseat when developing the project.

## Web Scraper Development

Turned out the best as was the smallest part but a very important piece

## Timeline

Watch a tutorial about using a web scraper with python and selenium

Use Replit to create a web scraper using python. I did not want to install all the python libraries, drivers, and dependencies locally, especially when the final web scraper would run on an AWS lambda.



Imported selenium with web drivers. These dependencies are automatically installed with Replit, so I did not have to worry about setting up the selenium web driver.

I tested getting a page and printed the results. After getting the web page's information, I started looking for websites with the release dates of movies, books, games, and shows. I also wanted them to be reoccurring as I wanted the lambda to run daily and retrieve new information every time it ran. So, if they were reoccurring, I could scrape them once a day and have up-to-date data in my SQL database. I got two or three websites from each media.

```
#Movie urls to be scraped
MOVIE_METACRITIC_COMING_SOON_URL = 'https://www.metacritic.com/browse/movies/release-date/coming-soon/date?view=detailed'
MOVIE_IMDB_COMING_SOON_URL = 'https://www.imdb.com/movies-coming-soon/'

#Game urls to be scraped
GAME_METACRITIC_COMING_SOON_URL = 'https://www.metacritic.com/feature/major-upcoming-video-game-release-dates-xbox-ps4-pc-switch'
GAME_STEAM_COMING_SOON_URL = 'https://store.steampowered.com/explore/upcoming/'

#Book urls to be scraped
BOOK_FANTASTIC_COMING_SOON_URL = 'https://www.fantasticfiction.com/coming-soon/'
BOOK_AMAZON_COMING_SOON_URL = 'https://www.amazon.com/Books-Coming-Soon/s?rh=nr3A2B3155%2cp_n_publication_date%3A1250228011'
BOOK_RISINGSHADOW_COMING_SOON_URL = 'https://www.risingshadow.net/library/comingbooks/'

#Show urls to be scraped
SHOW_METACRITIC_COMING_SOON_URL = 'https://www.metacritic.com/browse/tv/release-date/coming-soon/date'

#Path to chrome driver
PATH_TO_CHROMEDRIVER = 'C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe'
```

Figure 35 Websites to be Scraped

I tested the movie website Metacritic first before implementing it with other media i.e., repeat with books, shows, games. After setting up the driver with chrome using selenium, I was able to retrieve the page.

```
def get_driver():
    chrome_options = Options()
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_argument('--disable-dev-shm-usage')
    chrome_options.add_argument('--headless')

    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()),options=chrome_options)
    return driver
```

Figure 36 Create Chrome Driver

Then used the find\_elements (By.TAG\_NAME,'tr'). This would get every tag and everything in that tag and return a list. The tutorial I was following was using a find\_elements method that was deprecated, so I had to search for the current way to use the method. Luckily, it had not changed much. I have never formally learned python, only in past small projects, so I was not familiar with all the syntax. I did watch a few basic tutorials so I would understand what I was doing a bit better. Some of the syntax used in the web scraper tutorial I did not understand at all, but by the time I finished the web scraper, I understood how the web scraper functioned.

To parse the media. I created a function for each media. Parse\_movie\_metacritic would take in one

movie element from the webpage and return a Python object I could use. The web scraper tutorial returned a data structure I did not understand, and python threw no exceptions. I found it odd that python did not require me to

```
try:
    title_tag = movie.find_element(By.CLASS_NAME, 'title')
    title = title_tag.text

    img_url_tag = movie.find_element(By.TAG_NAME, 'img')
    img_url = img_url_tag.get_attribute('src')

    score_tag = movie.find_element(By.CLASS_NAME, 'metascore_w')
    score = score_tag.text

    multi_tag = movie.find_element(By.CLASS_NAME, 'clamp-details')
    multi = multi_tag.text
    m = multi.split('|')
    date = m[0]
    rating = m[1]

    description_tag = movie.find_element(By.CLASS_NAME, 'summary')
    description = description_tag.text

    return {
        'title': title,
        'img_url': img_url + ' ',
        'date': parse_date(date),
        'rating': rating,
        'score': score,
        'description': description
    }
```

Figure 37 Webscraper Parser Function

initiate objects manually. Parse\_movie\_metacritic does not specify a return type. The return type was a dictionary, storing string key and object value pair.

Parse\_movie\_metacritic threw an exception because each second tr, webpage element of a row in a table, was empty with other empty ones. Had problems figuring out how to handle exceptions. Try except- syntax.

```
movies = driver.find_elements(By.TAG_NAME, 'tr')
movies_data = []
for movie in movies:
    parsedMovie = parse_movie_metacritic(movie)
    if parsedMovie is not None:
        if parsedMovie['date'] is not None:
            movies_data.append(parsedMovie)
add_movies_to_db(movies_data)
```

I had two loops. One went through a list of web table rows and removed empty tags. Then another loop parses the data into a list of dictionaries. Refactor using "if is None". which split the runtime in half, especially when all four media web pages were queried

Figure 38 For Loop Parser

Turned list of dictionaries into csv file using panda data frame. This helped to visualize the data and correct the web scraper to

```
movies_df = pd.DataFrame(movies_data)
movies_df.to_csv(file)
```

show the information I needed.

Figure 39 Python Pandas

Some movies didn't have release dates, so I didn't want to use them, added another if statement to handle if the dictionary[date] is None, to skip their rows.

I ran into this problem with both shows and games websites.

Then I had to researched how to add the data to a SQL database. Replit doesn't support pyodbc as it requires a SQL driver to connect to a SQL database.

Set up python and pip on my computer and laptop and download an ODBC driver for the SQL database. Now must configure selenium on pc. Finally, install web driver manager to install the correct chrome driver during runtime.

There was a little problem when writing add\_to\_db () methods, with sanitization and dictionary (which is what I was adding to the database) being null. Created method to change string date to type date so it would be appropriately stored in the database. Methods can change multiple date string formats into suitable date format for the database to understand.

```
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
import pyodbc
import uuid
import datetime
from code import parse_date
```

Figure 40 Python Imports

```
def parse_date(unformatted_date):
    try:
        months = {
            'January':1,
            'February':2,
            'March':3,
            'April':4,
            'May':5,
            'June':6,
            'July':7,
            'August':8,
            'September':9,
            'October':10,
            'November':11,
            'December':12,
        }

        date_part = unformatted_date.replace(",","").split()

        date_part_year=1990
        date_part_month=1
        date_part_day=1

        throw = True

        for key, value in months.items():
            if date_part[0].casefold() == key.casefold():
                date_part_month = value
                date_part_day = date_part[1]
                date_part_year = date_part[2]
                throw = False
            elif date_part[1].casefold()[0:3] == key.casefold()[0:3]:
                date_part_month = value
                date_part_day = date_part[0]
                date_part_year = date_part[2]
                throw = False

        if throw:
            raise Exception

        return date(
            day = int(date_part_day),
            month = int(date_part_month),
            year = int(date_part_year)
        )
```

Figure 41 Change Date Function

Problem with null fields and unique ids for mediaId. Imported uuid<> which is the same as guid<>. Learned what these both do<detail>.

```
def add_games_to_db(games_data):
    try:
        with pyodbc.connect(DB_CONNECTION) as conn:
            with conn.cursor() as cursor:
                games = []
                q = 'INSERT INTO [dbo].[Games] (MediaId, SerialId, CreatingPropertyId, '
                for game in games_data:
                    mediaId = uuid.uuid4()
                    serialId = uuid.uuid4()
                    creatingPropertyId = uuid.uuid4()

                    games.append([
                        mediaId,
                        serialId,
                        creatingPropertyId,
                        game['title'],
                        game['img_url'],
                        game['description'],
                        game['date']
                    ])

                cursor.executemany(q, games)
                conn.commit()
    except Exception as e:
        print(e)
```

Figure 42 Add to Database Function

The method is inside a loop, so each element adds to the database. Many SQL queries should try to do less. I could have let the add\_to\_db method take in a list, and all the data inserted together and let the SQL database handle it as one SQL statement. However, I did not want to set up a second loop to handle the now complete set(no null columns) data. It would have increased the runtime by double. Lambda has a 15-minute limit, so I

wanted to keep the runtime down, especially if I want to add more to this webspace, although I could create

another separate web scraper to handle more methods.

Created stored procedure to delete duplicates if title release date and description are the same. Little bit complicated. What if media changes date? Or two different media by the same name are coming out??

Created triggers for stored procedure, to trigger when insert into media/movies/shows/games/books. (If so, then stored procedure is triggered every time there is a request, will need to refactor if I want to be more efficient)

```
--
ALTER PROCEDURE [dbo].[BooksDeleteDuplicates]
AS
|
|
|
BEGIN
    WITH CTE([title],
             [description],
             [releaseDate],
             DuplicateCount)
    AS (SELECT [title],
              [description],
              [releaseDate],
              ROW_NUMBER() OVER(PARTITION BY [title],
                                      [description],
                                      [releaseDate]
                                  ORDER BY mediaId) AS DuplicateCount
    FROM [dbo].[Books])
    DELETE FROM CTE
    WHERE DuplicateCount > 1;
END
```

Figure 43 Stored Procedure

## Evaluation

I believe the web scraper achieves the goals set out when developing this project. It supplies the database with data automatically, while also deleting duplicates. The task is automated. I had to learn python to complete this task and have enjoyed using it. The syntax is easy to understand and how python must be formatted to run properly. I have tried to adopt a python or similar format when using other languages. I understand the reason for readability and its importance when refactoring projects.

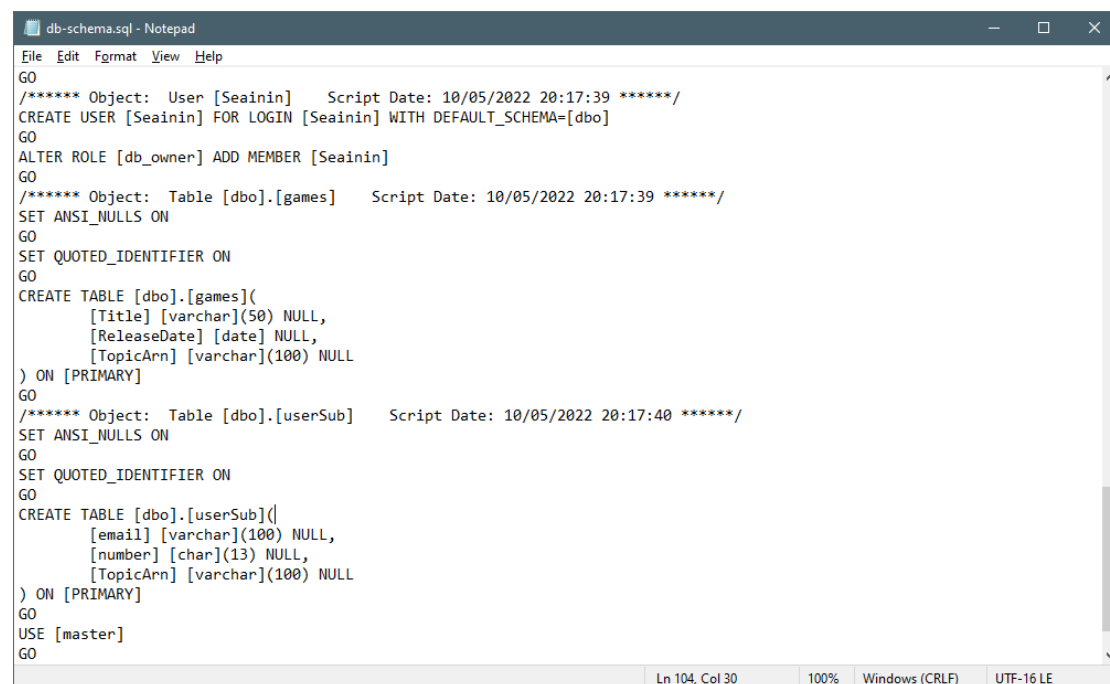
## Push Notifications

Created prototype, have not added in yet.

I would be using the knowledge learned in cloud architecture for push notifications. This includes two lambda functions, simple notification services, and a relational database service. These services are all provided by amazon web services.

When a user follows a media, the application will send an API call with a payload to a lambda function. This lambda function will add their contact information into either the azure database or a new RDS database run on AWS. In addition, the lambda function will check the azure database to see if a media has a TopicArn, used for SNS. Suppose the media does have a TopicArn continue. If it does not, create a new topic and add the topicArn to the media. This part has not been implemented yet.

The second Lambda function will run once a day. It will check the azure database and check if any media is released today. If so, publish a message to topicArn.



```

db-schema.sql - Notepad
File Edit Format View Help
GO
/***** Object: User [Seainin] Script Date: 10/05/2022 20:17:39 *****/
CREATE USER [Seainin] FOR LOGIN [Seainin] WITH DEFAULT_SCHEMA=[dbo]
GO
ALTER ROLE [db_owner] ADD MEMBER [Seainin]
GO
/***** Object: Table [dbo].[games] Script Date: 10/05/2022 20:17:39 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[games](
    [Title] [varchar](50) NULL,
    [ReleaseDate] [date] NULL,
    [TopicArn] [varchar](100) NULL
) ON [PRIMARY]
GO
/***** Object: Table [dbo].[userSub] Script Date: 10/05/2022 20:17:40 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[userSub](
    [email] [varchar](100) NULL,
    [number] [char](13) NULL,
    [TopicArn] [varchar](100) NULL
) ON [PRIMARY]
GO
USE [master]
GO
Ln 104, Col 30 100% Windows (CRLF) UTF-16 LE

```

Figure 44 RDS Prototype

This is the schema of the database used for the prototype. The system was deployed using cloud formation, to deploy in one go. The database has a media table and a userSub table. Will have to separate userSub by topicArn as the current userSub table will store a lot of data.

```
#!/usr/bin/env
Write-host "Waiting for cloudformation to be created. will take a few minutes"
$StackId = (aws cloudformation create-stack --stack-name dbstack --template-body file://template.json --capabilities CAPABILITY_IAM)
write-host "Cloud Initiated please wait while rds is created, may take up to 20 minutes"
aws cloudformation wait stack-create-complete --stack-name dbstack
aws rds wait db-instance-available --db-instance-identifier sqldatabase

Write-host "Creating pyodbc driver layer"
$Layer = (aws lambda publish-layer-version --layer-name pyodbcdriver --zip-file fileb://pyodbc-layer.zip --compatible-runtimes python3.8)

Write-host "Adding .Zip to lambda function"
aws lambda update-function-code --function-name lambdaSNSFromRDS --zip-file fileb://function.zip

Write-host "Getting latest pyodbc layer"
$LayerArn = (aws lambda list-layer-versions --layer-name pyodbcdriver | ConvertFrom-Json)

Write-host "Adding layer to lambda"
aws lambda update-function-configuration --function-name lambdaSNSFromRDS --layers $LayerArn.LayerVersions.LayerVersionArn[0]

#-get route table id
$RouteTableId = (aws ec2 describe-route-tables | ConvertFrom-Json).RouteTables.RouteTableId
#-get internet gateway id
$InternetGatewayId = (aws ec2 describe-internet-gateways | ConvertFrom-Json).InternetGateways.InternetGatewayId
Write-host "Adding Internet Gateway to RouteTable"
#-add route to route table//allow all connections to database instance
aws ec2 create-route --route-table-id $RouteTableId --destination-cidr-block 0.0.0.0/0 --gateway-id $InternetGatewayId
```

Figure 45 Set up Cloud Formation

Set up a PowerShell file to create the cloud formation for the prototype of the Push notifications. Creates the stack from an existing template I created. Then Adds a pyodbc layer to allow the lambda function to communicate with a SQL database. The PowerShell then adds a route to allow the database to be accessed through the internet.

```
teardown.ps1 > ...
1 Write-host "Deleting layer as it was not created in the cloud fromation"
2 aws lambda delete-layer-version --layer-name $Layer.LayerArn --version-number $LayerArn.LayerVersions.Version[0]
3
4 $topics = (aws sns list-topics | ConvertFrom-Json).Topics
5 Foreach ($topic in $topics){
6     aws sns delete-topic --topic-arn $topic.TopicArn
7 }
8 Write-host "Deleting dbstack"
9 aws cloudformation delete-stack --stack-name dbstack
10 aws cloudformation wait stack-delete-complete --stack-name dbstack
11 Write-host "Deleted, GOODBYE"
```

Figure 46 Teardown Cloud Formation

Teardown PowerShell, to take down the AWS resources

```
#!/usr/bin/env python3
import boto3
import sys
import os
import json
import logging
import argparse
import time
import random
import string

username = 'seanin'
password = 'password'

#conn = pyodbc.connect('Driver={ODBC Driver 17 for SQL Server};Server=tcp:mysqldbserverseaninkeeenan.database.windows.net,1433;Database=MediaApiDb;Uid=seaninkeeenan;Pwd=password;Encrypt=yes;TrustServerCertificate=yes;')
#conn = pyodbc.connect('Driver={ODBC Driver 17 for SQL Server};Server='+server+';Database='+database+';Uid='+username+';Pwd='+password+';Encrypt=yes;TrustServerCertificate=yes;')
return conn.cursor()

def subscribe(topic, protocol, endpoint):
    sns = boto3.client('sns')
    subscription = sns.subscribe(TopicArn=topic, Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True)
    return subscription

def publish_message(topic, message):
    sns = boto3.client('sns')
    response = sns.publish(TopicArn=topic, Message=message)
    messageId = response['MessageId']
    return messageId

def delete_subscription(subscription):
    subscription.delete()

def delete_topic(topic):
    topic.delete()

def create_topic(name):
    sns = boto3.resource('sns')
    topic = sns.create_topic(Name=name)
    return topic

def list_topics():
    sns = boto3.resource('sns')
    topics_iter = sns.topics.all()
    return topics_iter
```

Figure 47 SNS Python Functions

These are the functions that were used in the prototype and will be used in the push notification lambda functions.

## Conclusions

The aims and objective set at the start of this project were achieved.

### Aims

A mobile application was designed, which allowed users to view media releases. It was not fully completed to a satisfying level. The mobile application could use a lot more time to fully design the user interface. The application currently is very basic and shows the bare bones of the capabilities of the Ionic Framework.

The API, Azure Database and web scraper are all deployed on cloud platforms. The project uses the cloud platforms Azure and AWS to operate.

The database design was able to hold all the data I required it to. There are empty tables that I plan to fill, using more dynamic web scraper or other APIs. I believe the design of the database functions well for the purposes required of it.

I created a web scraper by learning a new language, python, and the libraries needed to operate a python web scraper. I also familiarized myself with the functions of drivers so I could better understand the web scraper I was creating. Along with the web scraper I learned how to deploy small bits of code to AWS lambda functions to automate a task, which came in useful in this project.

I also learned how to make use of AWS's Simple Notification Service and use it with my project to notify users of media releases.

### Objectives

I have learned a lot since starting this project. I have not added anything into this project that I felt I didn't have a good understanding of. I am not an expert in all the technologies used but have a good understanding of them. When I required new functions to be added into the project, I researched ways that they could be accomplished and if I would be able to understand how to utilize them. I only added in functions that I thought I would be able to understand.

### Future Work

#### API

I plan to implement API keys into the media API. I believe I could make use of them in this project. To record and restrict users' use of the API. I would like to have the API public so that other people may use it for their own purposes, as I do believe it will be useful for some projects.

I would also like to redesign the application to handle requests like this API <https://the-one-api.dev/v2>. This Lord of The Rings API lets the user add in any number of parameters into one endpoint and returns the desired data to the user. I did try to implement this into the media API but moved on to other parts of the project that were more functional.

#### Web Scraper

I think there is a lot I could do with this web scraper or even a new one. I will refactor this web scraper as there is a lot of repetition in it. There are three functions for each type of media. I do believe that with better utilization of the code I could create

dynamic functions to handle any media, either putting it into a database or parse the webpages results. This would also help with adding new webpages to the web scraper.

I would also like to add a profanity filter as some of the results can be inappropriate. I have not decided whether to add the filter into the web scraper, meaning the database will have nothing inappropriate in it or if I will allow the user to decide if they want to see that content, meaning the inappropriate content will be stored in the database.

Make another web scraper to search for trailers for media in SQL database and add them to respective media. I would also like to find information to fill the rest of the database as I believe it would create a more useful API.

Add another stored procedure to order all medias by release date. Faster than order every request.

### Mobile Application

Add calendar functionality to keep track of media added to calendar. Just so the user could see them. Plan to add soon but wasn't completed by the time of this document

Allow users to submit correction to media data. Either have someone manually check and add correct information or cross examine submits of the same media see if it's the same correction and implement that correction.

I would also like to create a better User Interface and experience, as the application is bare bones and the minute and isn't hugely interactive.

I have learned and accomplished a lot since starting this project, with technologies that I will be using in the future.



## Source Code

<https://github.com/Seainin2/MediaProjectWebScraper>

<https://github.com/Seainin2/MediaProject>

```
export class MediaPage{

  data = [];
  results: Observable<any>;

  constructor(public navCtrl: NavController, public httpClient: HttpClient,private route: Router,private alertController: AlertController) { }

  ngOnInit() {
    this.results = this.httpClient.get('http://localhost:5000/api/Feed');
    this.results
    .subscribe(data => {
      console.log('my data: ', data);
    })
  }

  moreDetails(media: any){
    let navigationExtras: NavigationExtras = {
      state: {
        media: media
      }
    };
    this.route.navigate(['movie'], navigationExtras);
  }
}
```

```
/*async searchDataFuckCors() {
  let loading = await this.loadingCtrl.create();
  await loading.present();

  this.httpClient.get('http://localhost:5000/api/movies').pipe(
    finalize(() => loading.dismiss())
  )
  .subscribe(data => {
    this.data = data[0];
  }, err => {
    console.log('JS Call error: ',err);
  })
}

searchData(){
  this.results = this.httpClient.get('http://localhost:5000/api/movies');
  this.results
  .subscribe(data => {
    console.log('my data: ', data);
  })
}
*/
```

## Media release

```
}

import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {
    path: '',
    redirectTo: 'sidenav',
    pathMatch: 'full'
  },
  {
    path: 'login',
    loadChildren: () => import('./pages/login/login.module').then( m => m.LoginPageModule),
  },
  {
    path: 'signup',
    loadChildren: () => import('./pages/signup/signup.module').then( m =>
m.SignupPageModule)
  },
  {
    path: 'sidenav',
    loadChildren: () => import('./pages/sidenav/sidenav.module').then( m =>
m.SidenavPageModule)
  },
  {
    path: 'movie',
    loadChildren: () => import('./pages/details/movie/movie.module').then( m =>
m.MoviePageModule)
  },
  {
    path: 'games',
    loadChildren: () => import('./pages/details/games/games.module').then( m =>
m.GamesPageModule)
  }
];
```

## Source Code

```
},

];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
import { SplashScreen } from '@ionic-native/splash-screen/ngx';
import { StatusBar } from '@ionic-native/status-bar/ngx';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';

import { HttpClientModule } from '@angular/common/http';

import { AngularFireModule } from '@angular/fire/compat';
import { AngularFireStoreModule } from '@angular/fire/compat/firestore';
//import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { environment } from 'src/environments/environment';
```

## Media release

```
@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports:
[AngularFireModule.initializeApp(environment.firebase),AngularFirestoreModule,BrowserModule
, IonicModule.forRoot(), AppRoutingModule,HttpClientModule],
  providers: [SplashScreen,StatusBar,{ provide: RouteReuseStrategy, useClass:
IonicRouteStrategy }],
  bootstrap: [AppComponent],
})
export class AppModule {}
import { Injectable } from '@angular/core';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { AngularFirestore } from "@angular/fire/compat/firestore"

@Injectable({
  providedIn: 'root'
})
export class FireserviceService {

  constructor(public firestore: AngularFirestore,public auth: AngularFireAuth) { }

  loginWithEmail(data) {
    return this.auth.signInWithEmailAndPassword(data.email,data.password);
  }

  signup(data) {
    return this.auth.createUserWithEmailAndPassword(data.email,data.password)
  }

  saveDetails(data) {
    return this.firestore.collection("users").doc(data.uid).set(data);
  }
}
```

## Source Code

```
}

getDetails(data) {
  return this.firestore.collection("users").doc(data.uid).valueChanges();
}

}

import { Injectable } from '@angular/core';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { AngularFirestore } from "@angular/fire/compat/firestore"

@Injectable({
  providedIn: 'root'
})
export class FireserviceService {

  constructor(public firestore: AngularFirestore,public auth: AngularFireAuth) { }

  loginWithEmail(data) {
    return this.auth.signInWithEmailAndPassword(data.email,data.password);
  }

  signup(data) {
    return this.auth.createUserWithEmailAndPassword(data.email,data.password)
  }

  saveDetails(data) {
    return this.firestore.collection("users").doc(data.uid).set(data);
  }

  getDetails(data) {
```

## Media release

```
        return this.firestore.collection("users").doc(data.uid).valueChanges();
    }
}
<ion-header>
  <ion-toolbar color="primary">
    <div>
      <ion-title>Newsfeed</ion-title>
    </div>
    <div>
      <ion-buttons slot="start">
        <ion-menu-button autoHide="false"></ion-menu-button>
      </ion-buttons>
    </div>
    <div>
      <ion-searchbar showCancelButton="focus" placeholder="FilterSearch" animated></ion-searchbar>
    </div>
    <ion-item>
      <ion-label>Media Type</ion-label>
      <ion-select multiple="true" cancelText="Nah" okText="Okay!">
        <ion-select-option value="Game">Game</ion-select-option>
        <ion-select-option value="Book">Book</ion-select-option>
        <ion-select-option value="Movie">Movie</ion-select-option>
        <ion-select-option value="Show">Show</ion-select-option>
        <ion-select-option value="Series">Series</ion-select-option>
      </ion-select>
    </ion-item>
  </div>
</ion-toolbar>
</ion-header>
```

## Source Code

```
<ion-content>
  <ion-list>
    <ion-card button *ngFor="let item of (results | async); let i = index"
      (click)="moreDetails(item)">
      {{item.title}}
      <br>
      {{item.releaseDate}}
      <br>
      {{item.mediaType}}
      <br>
      <ion-img src="{{item.imageName}}"></ion-img>
    </ion-card>
  </ion-list>
</ion-content>

using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace MediaApi.Models
{
    //in dbset
    public class Game
    {
        [Key]
        public Guid MediaId { get; set; }

        public Guid SeriesId { get; set; }

        public Guid CreatingPropertyId { get; set; }
    }
}
```

## Media release

```
[Required]

public String MediaType { get; set; }

[Required]
public String Title { get; set; }

public String ImageName { get; set; }

public String Description { get; set; }

public int NumberofTimesSearched { get; set; }

[NotMapped]
public List<MultiplayerOption> MultiplayerOptions { get; set; }

[NotMapped]
public List<Platform> Platforms { get; set; }

[Required]
[Column(TypeName = "Date")]
public DateTime ReleaseDate { get; set; }

[NotMapped]
public List<Theme> Themes { get; set; }

[NotMapped]
public List<Genre> Genres { get; set; }

[NotMapped]
public List<Person> People { get; set; }
```

## Source Code

```
[NotMapped]
public List<RecommendingResource> OfficialReviews { get; set; }

[NotMapped]

public List<UserReview> Reviews { get; set; }

}

//in dbset
public class GameMultiplayerOption
{
    [Key]

    public Guid Id { get; set; }
    [Required]
    public Guid MediaId { get; set; }
    [Required]
    public Guid MultiplayerOptionId { get; set; }
}

//in dbset
public class MultiplayerOption
{
    [Key]

    public Guid MultiplayerOptionId { get; set; }

    [Required]
    public String MultiplayerOptionType { get; set; }

}
```

## Media release

```
//in dbset
public class GamePlatform {

    [Key]

    public Guid Id { get; set; }

    [Required]

    public Guid MediaId { get; set; }

    [Required]

    public Guid PlatformId { get; set; }

}

//in dbset
public class Platform

{

    [Key]

    public Guid PlatformId { get; set; }

    [Required]

    public String PlatformName { get; set; }

}

using MediaApi.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace MediaApi.Data
{
    public class SqlGameData : IGameData
    {

```

## Source Code

```
private AllContext _allContext;

public SqlGameData(AllContext allContext)
{
    _allContext = allContext;
}

public Game AddGame(Game game)
{
    _allContext.Games.Add(game);
    _allContext.SaveChanges();
    return game;
}

public bool DeleteGame(Guid id)
{
    throw new NotImplementedException();
}

public Game GetGame(Guid id)
{
    return _allContext.Games.Find(id);
}

public List<Game> GetGames()
{
    return _allContext.Games.ToList();
}

public Game UpdateGame(Game game)
{
    throw new NotImplementedException();
}

```

## Media release

```
    }
}

using MediaApi.Models;
using System;
using System.Collections.Generic;

namespace MediaApi.Data
{
    public interface IGameData
    {
        List<Game> GetGames();

        Game GetGame(Guid id);

        Game AddGame(Game game);

        Game UpdateGame(Game game);

        Boolean DeleteGame(Guid id);
    }
}

using MediaApi.Data;
using MediaApi.Models;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System;
using System.IO;
using static MediaApi.Controllers.ImageUploadController;
```

## Source Code

```
namespace MediaApi.Controllers
{
    [ApiController]
    public class GamesController : ControllerBase
    {
        private IGameData _Data;
        private static IWebHostEnvironment _environment;

        public GamesController(IGameData allData, IWebHostEnvironment environment)
        {
            _environment = environment;
            _Data = allData;
        }

        [HttpGet]
        [Route("api/[controller]")]

        public IActionResult GetGames()
        {
            return Ok(_Data.GetGames());
        }

        [HttpGet]
        [Route("api/[controller]/{id}")]

        public IActionResult GetGame(Guid id)
        {
            var media = _Data.GetGame(id);
            if (media != null)
            {
                return Ok(media);
            }
        }
    }
}
```

## Media release

```
    }

    return NotFound($"Game with Id: {id} was not found");
}

[HttpPost]
[Route("api/[controller]")]

public IActionResult AddGame([FromForm] String data, [FromForm] FileUploadAPI
objFile)
{
    try
    {
        if (objFile.file.Length > 0 && !objFile.Equals(null))
        {
            if (!Directory.Exists(_environment.WebRootPath + "\\Game\\"))
            {
                Directory.CreateDirectory(_environment.WebRootPath + "\\Game\\");
            }

            using FileStream fileStream =
System.IO.File.Create(_environment.WebRootPath + "\\Game\\" + objFile.file.FileName);
            objFile.file.CopyTo(fileStream);
            fileStream.Flush();

            Game game = JsonConvert.DeserializeObject<Game>(data);
            game.ImageName = objFile.file.FileName;
            game.MediaType = "Game";
            return Ok(_Data.AddGame(game));
        }
        else
        {
```

## Source Code

```
        return Ok("Show was NOT!! added, no image");
    }

    catch (Exception e)
    {
        return Ok(e.Message.ToString());
    }
}

}

import pyodbc
import time
import boto3
from botocore.exceptions import ClientError
from datetime import datetime

def connect_to_db(key):
    #server = ' 'key
    server = 'tcp:'+key+',3306'
    database = 'GamesDatabase'
    #port = '3306'
    username = 'Seainin'
    password = 'password'

    #cnxn = pyodbc.connect('Driver={ODBC Driver 17 for SQL
Server};Server=tcp:mysqlserverseaininkeenan.database.windows.net,1433;Database=MediaApiDb;U
id=seaininkeenan;Pwd=;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;')

    cnxn = pyodbc.connect('Driver={ODBC Driver 17 for SQL
Server};Server='+server+';Database='+database+';Uid='+username+';Pwd='+password+';Encrypt=y
es;TrustServerCertificate=no;Connection Timeout=15;')
```



## Media release

```
    return cnxn.cursor()

def subscribe(topic, protocol, endpoint):
    sns = boto3.client('sns')

    subscription =
sns.subscribe(TopicArn=topic, Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True
)

    return subscription

def publish_message(topic, message):
    sns = boto3.client('sns')

    response = sns.publish(TopicArn=topic, Message=message)

    messageId = response['MessageId']

    return messageId

def delete_subscription(subscription):
    subscription.delete()

def delete_topic(topic):
    topic.delete()

def create_topic(name):
    sns = boto3.resource("sns")

    topic = sns.create_topic(Name=name)

    return topic

def list_topics():
    sns = boto3.resource("sns")

    topics_iter = sns.topics.all()

    return topics_iter

if __name__ == "__main__":
```

## Source Code

```
def main_function(event, context):
    print(event)

    key = event['server']

    cursor = connect_to_db(key)

    Name = 'Witcher'

    topic1 = create_topic(Name)

    print(topic1.arn)

    cursor.execute("Insert into [dbo].[games] (Title, TopicArn) Values
(?, ?) ", Name, topic1.arn)

    cursor.commit()

    Name = 'Zelda'

    topic2 = create_topic(Name)

    cursor.execute("Insert into [dbo].[games] (Title, TopicArn) Values
(?, ?) ", Name, topic2.arn)

    cursor.commit()

    cursor.execute('SELECT * FROM [dbo].[games]')

    rows = cursor.fetchall()

    for row in rows:

        subscribe(row[2], 'email', event['email'])

    time.sleep(60)

    for row in rows:
```

## Media release

```
        publish_message(row[2],f"You will be notified when {row[0]} Releases!!!!!!  
YYYYAAAAYYYY")  
  
        cursor.execute("Delete from games")  
  
import pandas as pd  
from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from webdriver_manager.chrome import ChromeDriverManager  
from selenium.webdriver.chrome.options import Options  
from selenium.webdriver.common.by import By  
import pyodbc  
import uuid  
import datetime  
from code import parse_date  
  
#Movie urls to be scraped  
MOVIE_MEATCRITIC_COMING_SOON_URL = 'https://www.metacritic.com/browse/movies/release-  
date/coming-soon/date?view=detailed'  
MOVIE_IMDB_COMING_SOON_URL = 'https://www.imdb.com/movies-coming-soon/'  
  
#Game urls to be scraped  
GAME_METACRITIC_COMING_SOON_URL = 'https://www.metacritic.com/feature/major-upcoming-video-  
game-release-dates-xbox-ps4-pc-switch'  
GAME_STEAM_COMING_SOON_URL = 'https://store.steampowered.com/explore/upcoming/'  
  
#Book urls to be scraped  
BOOK_FANTASTIC_COMING_SOON_URL = 'https://www.fantasticfiction.com/coming-soon/'  
BOOK_AMAZON_COMING_SOON_URL = "https://www.amazon.com/Books-Coming-  
Soon/s?rh=n%3A283155%2Cp_n_publication_date%3A1250228011"  
BOOK_RISINGSHADOW_COMING_SOON_URL = "https://www.risingshadow.net/library/comingbooks"  
  
#Show urls to be scraped
```

## Source Code

```
SHOW_METACRITIC_COMING_SOON_URL = 'https://www.metacritic.com/browse/tv/release-  
date/coming-soon/date'  
  
#Path to chrome driver  
PATH_TO_CHROMEDRIVER = 'C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe'  
  
#Database connection string  
DB_CONNECTION = 'Driver={ODBC Driver 18 for SQL  
Server};Server=tcp:mysqlserverseaininkeenan.database.windows.net,1433;Database=MediaApiDb;U  
id=seaininkeenan;Pwd=;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;'  
  
def get_driver():  
    chrome_options = Options()  
    chrome_options.add_argument('--no-sandbox')  
    chrome_options.add_argument('--disable-dev-shm-usage')  
    chrome_options.add_argument('--headless')  
  
    driver =  
    webdriver.Chrome(service=Service(ChromeDriverManager().install()),options=chrome_options)  
    return driver  
  
def check_if_class_exists(type, tag, html):  
    try:  
  
        if type == 'class':  
            isText = html.find_element(By.CLASS_NAME, tag).text.strip()  
        if type == 'tag':  
            isText = html.find_element(By.TAG_NAME, tag).text.strip()  
  
        if not isText:  
            return False  
        return True
```

## Media release

```
except:
    return False

#Show-
START*****
*****

def handle_shows_metacritic(driver, url, file):
    driver.get(url)
    shows = driver.find_elements(By.TAG_NAME, 'tr')

    shows_data = []

    for show in shows:
        parsedShow = parse_show_metacritic(show)

        if parsedShow is not None:
            if parsedShow['date'] is not None:
                shows_data.append(parsedShow)
    add_shows_to_db(shows_data)

    shows_df = pd.DataFrame(shows_data)
    shows_df.to_csv(file)

def add_shows_to_db(shows_data):
    try:
        with pyodbc.connect(DB_CONNECTION) as conn:
            with conn.cursor() as cursor:
                shows = []
                q = "INSERT INTO [dbo].[Shows]
(MediaId,SeriesId,CreatingPropertyId,MediaType,Title,ImageName) VALUES (?, ?, ?, ?, ?, ?)"
                for show in shows_data:
```

## Source Code

```
mediaId = uuid.uuid4()
seriesId = uuid.uuid4()
creatingPropertyId = uuid.uuid4()

shows.append((
    mediaId,
    seriesId,
    creatingPropertyId,
    'show',
    show['title'],
    show['img_url'],
))

cursor.executemany(q, shows)
conn.commit()

except Exception as e:
    print(e)

def parse_show_metacritic(show):
    try:
        title_tag = show.find_element(By.CLASS_NAME, 'title')
        title = title_tag.text

        img_url_tag = show.find_element(By.TAG_NAME, 'img')
        img_url = img_url_tag.get_attribute('src')

        score_tag = show.find_element(By.CLASS_NAME, 'metascore_w')
        score = score_tag.text

        multi_tag = show.find_element(By.CLASS_NAME, 'clamp-details')
        multi = multi_tag.text
```

## Media release

```
description_tag = show.find_element(By.CLASS_NAME, 'summary')
description = description_tag.text

return {
    'title': title,
    'img_url': img_url+ ' ',
    'date': parse_date(multi),
    'score': score,
    'description': description
}
except:
    return None

#Shows-
END*****
*****

#Games-
START*****
*****

def handle_games_steam(driver, url, file):
    driver.get(url)
    page = driver.find_element(By.ID, 'tab_popular_comingsoon_content')
    games = page.find_elements(By.TAG_NAME, 'a')

    games_data = []

    for game in games:

        parsedGame = parse_game_steam(game)
        if parsedGame is not None:
            if parsedGame['date'] is not None:
                games_data.append(parsedGame)
```

## Source Code

```
add_games_to_db(games_data)

games_df = pd.DataFrame(games_data)
games_df.to_csv(file)

def add_games_to_db(games_data):
    try:
        with pyodbc.connect(DB_CONNECTION) as conn:
            with conn.cursor() as cursor:
                games = []

                q = "INSERT INTO [dbo].[Games]
(MediaId, SeriesId, CreatingPropertyId, MediaType, Title, ImageName, Description, NumberofTimesSea
rched, ReleaseDate) VALUES (?, ?, ?, ?, ?, ?, ?, ?)"

                for game in games_data:

                    mediaId = uuid.uuid4()
                    seriesId = uuid.uuid4()
                    creatingPropertyId = uuid.uuid4()

                    games.append((
                        mediaId,
                        seriesId,
                        creatingPropertyId,
                        'game',
                        game['title'],
                        game['img_url'],
                        'description',
                        0,
                        game['date']
                    ))

                cursor.executemany(q, games)
```

## Media release

```
        conn.commit()

    except Exception as e:
        print(e)

def parse_game_steam(game):
    try:
        title_tag = game.find_element(By.CLASS_NAME, 'tab_item_name')
        title = title_tag.text

        img_url_tag = game.find_element(By.TAG_NAME, 'img')
        img_url = img_url_tag.get_attribute('src')

        date_tag = game.find_element(By.CLASS_NAME, 'release_date')
        date = date_tag.text

        themes_tag = game.find_element(By.CLASS_NAME, 'tab_item_top_tags')
        themes = themes_tag.text

        url = game.get_attribute('href')

        ava = ['win', 'mac', 'linux']
        available = ''

        for x in ava:
            if check_if_class_exists('class', x, game):
                available = x + ', ' + available

        return {
            'title': title,
            'img_url': img_url + ' ',
            'date': parse_date(date),
            'tags': themes,
```

## Source Code

```
            'available': available,
            'URL': url
        }

    except:
        return None

#Games-
END*****

#Movies-
START*****

def handle_movies_metacritic(driver, url, file):
    movies = []
    driver.get(url)
    movies = driver.find_elements(By.TAG_NAME, 'tr')

    movies_data = []

    for movie in movies:
        parsedMovie = parse_movie_metacritic(movie)

        if parsedMovie is not None:
            if parsedMovie['date'] is not None:

                movies_data.append(parsedMovie)

    add_movies_to_db(movies_data)

    movies_df = pd.DataFrame(movies_data)
    movies_df.to_csv(file)
```

## Media release

```
def add_movies_to_db(movie_data):
    try:
        with pyodbc.connect(DB_CONNECTION) as conn:
            with conn.cursor() as cursor:

                movies = []

                q = "INSERT INTO [dbo].[Movies]
(MediaId,SeriesId,CreatingPropertyId,MediaType,Title,ImageName,Description,NumberofTimesSea
rched,Length,ReleaseDate) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

                for movie in movie_data:

                    mediaId = uuid.uuid4()
                    seriesId = uuid.uuid4()
                    creatingPropertyId = uuid.uuid4()

                    movies.append((
                        mediaId,
                        seriesId,
                        creatingPropertyId,
                        'movie',
                        movie['title'],
                        movie['img_url'],
                        movie['description'],
                        0,
                        0,
                        movie['date']
                    ))

                cursor.executemany(q,movies)

                conn.commit()

    except Exception as e:

        print(e)
```

## Source Code

```
def parse_movie_metacritic(movie):

    try:

        title_tag = movie.find_element(By.CLASS_NAME, 'title')
        title = title_tag.text

        img_url_tag = movie.find_element(By.TAG_NAME, 'img')
        img_url = img_url_tag.get_attribute('src')

        score_tag = movie.find_element(By.CLASS_NAME, 'metascore_w')
        score = score_tag.text

        multi_tag = movie.find_element(By.CLASS_NAME, 'clamp-details')
        multi = multi_tag.text
        m = multi.split('|')
        date = m[0]
        rating = m[1]

        description_tag = movie.find_element(By.CLASS_NAME, 'summary')
        description = description_tag.text

        return {
            'title': title,
            'img_url': img_url+' ',
            'date': parse_date(date),
            'rating': rating,
            'score': score,
            'description': description
        }
```

## Media release

```
except:

    return None

#Movies-
END*****
*****

#Books-
START*****
*****

def handle_books_risingshadow(driver, url, file):

    driver.get(url)

    books = driver.find_elements(By.CLASS_NAME, 'library')

    books_data = []

    for book in books:

        parsedBook = parse_book_risingshadow(book)

        if parsedBook is not None:

            if parsedBook['date'] is not None:

                #add_books_to_db(parsedBook)

                books_data.append(parsedBook)

    add_books_to_db(books_data)

    books_df = pd.DataFrame(books_data)

    books_df.to_csv(file)

def add_books_to_db(book_data):

    try:

        with pyodbc.connect(DB_CONNECTION) as conn:

            with conn.cursor() as cursor:

                books = []

                q = "INSERT INTO [dbo].[Books]
(MediaId,SeriesId,CreatingPropertyId,MediaTypeId,Title,Description,NumberofTimesSearched,Leng
th,ReleaseDate) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
```

## Source Code

```
for book in book_data:

    mediaId = uuid.uuid4()

    seriesId = uuid.uuid4()

    creatingPropertyId = uuid.uuid4()

    books.append((

        mediaId,

        seriesId,

        creatingPropertyId,

        'book',

        book['title'],

        'Description',

        0,

        0,

        book['date']

    ))

    cursor.executemany(q, books)

    conn.commit()

except Exception as e:

    print(e)

def parse_book_risingshadow(book):

    tr = book.find_elements(By.TAG_NAME, 'td')

    data = tr[1].text

    link = tr[0].find_element(By.TAG_NAME, 'a').get_attribute('href')

    img_url = tr[0].find_element(By.TAG_NAME, 'img').get_attribute('data-src')
```

## Media release

```
data = data.split('\n')

return {
    'title': data[0],
    'img_url': 'https://www.risingshadow.net/' + img_url+ ' ',
    'author': data[1],
    'date': parse_date(data[3]),
    'genre': data[4],
    'link': link
}

#Books-
END*****
*****

if __name__ == "__main__":
    print('Setting up driver')
    driver = get_driver()
    print('Driver set up')
```

## Source Code

```
print('Gettting Games')

#games on steam
handle_games_steam(driver, GAME_STEAM_COMING_SOON_URL, 'games_steam.csv')

print('Gettting Movies')

#movies on metacritic
handle_movies_metacritic(driver,
MOVIE_MEATCRITIC_COMING_SOON_URL, 'movies_metacritic.csv')

print('Gettting Books')

#books on risingshadow
handle_books_risingshadow(driver,BOOK_RISINGSHADOW_COMING_SOON_URL, 'books_risingshadow.csv'
)

print('Gettting Shows')

#shows on metacritic
handle_shows_metacritic(driver,SHOW_METACRITIC_COMING_SOON_URL, 'shows_metacritic.csv')

driver.quit()
```