2022

SteamFlow

CAB432 Assignment 1

Thomas Peters

10810251

9/19/2022

Contents

Introduction

Mashup Purpose & description

SteamFlow fills an unknown gap prevalent in majority of game related sites. This gap is the ability to view wish listed games on demand, and the reason for its lack of implementation is accessibility.
Steam has provided minimal documentation for wish list access, with what has been provided shifting formats, endpoints and authentication requirements, and that lack of access has hindered users' abilities to quickly access relative content on the game itself.

Services used

In the case of SteamFlow, the primary API is an undocumented internal API they publicly provide.

For clarity, the used API is an official one, however, it is generally implemented for use in the platforms front end store. This does not mean use of the service is not allowed as Steam themselves stated they are fine with public use of the API; however, they would not be supplying documentation as the webAPI is the preferred manor of access.

The purpose of using this API over that of the Steam webAPI was that some functions of the documented API require authentication. Steam themselves are an openID provider, which would require the application to become stateful and therefor no longer meet the requirements of the assignment. Additionally, many of steams webAPI results that are not authentication required function the same as their internal counterpart.

Additionally, as the Internal API is used for the storefront, it includes the natural endpoints for accessing information such as Steams currently featured games which is not accessible through the webAPI.

Internal Steam API

Documentation: [Community Compiled] https://github.com/Revadike/InternalSteamWebAPI/wiki

Base URI: `https://store.steampowered.com`

API: Wishlist

A paginated json endpoint that provides 2 forms of results.

Result 1: the status of the request is 500, referring to an inaccessible user profile.

Result 2: is a successful request returning upwards of 100 keys paired with a miniature appdetails object.

Endpoint: `/wishlist/profiles/`*`steamID`*`/wishlistdata`

API: Featured

Returns an object with 4 notable values. “large_capsules” is formally reserved for the featured carousel, whilst the other 3 follow the naming scheme “featured_*os*” representing a platform specific recommendation.

These each are an array of minimalistic apps containing important factors such as name, id, images, price, discounts, currency key, compatible platforms, and other information.

Endpoint: `/api/featured`

API: Appdetails

Appdetails provides the in-depth information about an application. This data is the same used on the applications home page and includes all store available information.

It requires an “appids” parameter to function and will provide a 400 should the ID not match any available game.

This endpoint is the only validated one to maintain a rate limit of 200 requests per 5 mins.

Endpoint: `/api/appdetails`

Google API (Youtube enabled)
Documentation: https://developers.google.com/youtube/v3/docs

Base URI: `https://www.googleapis.com/youtube/v3`

API: Search

Search in the applied sense makes use of the “part” parameter set to “snippet” to acquire the results thumbnail URI.

It will return the very basics otherwise, including each entries videoID more commonly known as the watch ID, and internal ID.

Endpoint: `/search`

Services

Docker	Redis
	Redis for this project is used as a response caching server to help avoid reaching rate limits on repeatedly accessed pages in addition to providing performance benefits. This is configurable through environmental variables meaning the image can be used as a standalone or with any Redis based service such as MemoryDB or ElastiCache. An accepted caching service is not required but is highly recommended. https://hub.docker.com/_/redis
AWS	DynamoDB
	DynamoDB is accessed server-side to handle the update and retrieval of the view count displayed on the index page. This is a fixed layout and is mandatory to the image's functionality. https://aws.amazon.com/dynamodb/

Mashup Use Cases and Services

Game Developer, Gamer, Reporter

Wishlist Management

As an	Gamer
I want	Quickly see all me of my wish list and related content.
So that	I can make a judgment on if I would like the game or not.

Content Access

As an	Reporter
I want	Quickly evaluate the game and the games community.
So that	I can write articles faster.

Community monitoring

As an	Game Developer
I want	Quickly find content on my game
So that	I can effectively monitor community response and possible issues.

The application is extremely simplistic in its use cases. This is namely due to the restrictions and time frame imposed by the assignment (refer to Extensions for a more in-depth idea as to why and what I would have changed to benefit these use cases more).

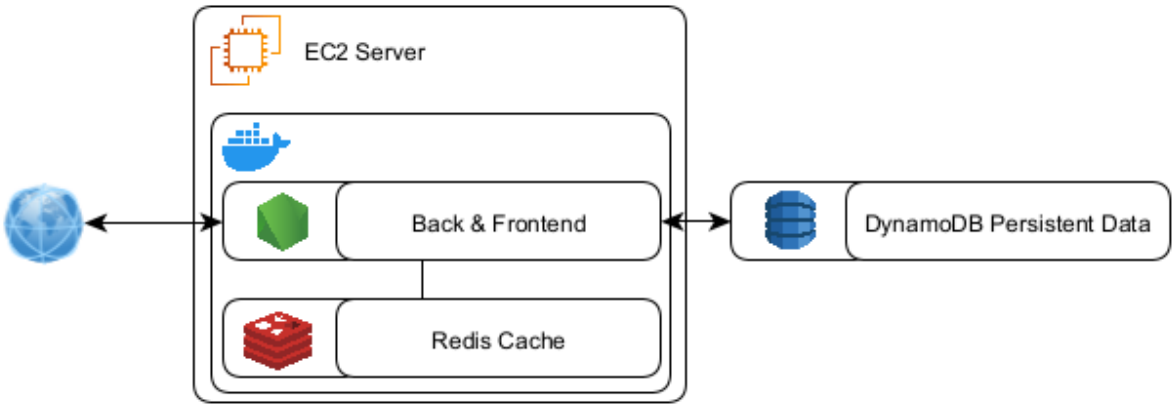
For reporters and developers, the intended application of the service is to rapidly access high quality videos for insight. Reporters in this sense would use the videos to gage currently popular gimmicks and opinions within the community, whilst developers would benefit more

from the ability to quickly see upcoming trends, discover possible problems with mechanics and monitor the community response to a feature.

For gamers, the wish list page and featured are the primary targets. Whilst the featured page provides no more than steam's own list, it maintains the ability to peak a user's interest in a title. The wish list page itself grants the simplified view of each title allowing a user to use the service to either identify titles they no longer show interest in or continue through to the app page to find impartial content to judge.

Technical breakdown

This is a deeper discussion of the architecture, the technology used in creating the mashup, any issues encountered, and overall, how you implemented the project.



As previously stated, the service implements Redis for caching to provide performance benefits and reducing the impact on the implemented services. To begin with, the cache has 3 notable expirations. Featured *(Steam)* generally resets daily, however, due to the unknown time zone of this event, the expiration is set to midnight locally rather than relative.
Any app requests will expire 3 hours from the latest request with one exception.
In the event Youtube should result in an error when requesting related videos, the resulting cache entry will expire at midnight PT. The reason for this is that in the event of the quota being reached, any cached entries will maintain their data on Youtube videos for 3 hours, any non-cached applications will be cached till the next daily reset occurs, at which they will expire with any further requests falling within the new quota.

Beyond the use of AWS-SDK for DynamoDB, the backend additionally implements Axios for performing requests.

The flow of the backend programmatically is more akin to a modular system. Apart from the central API, both Youtube and Steam have accessible endpoints defined as functions within in object. In retrospect however, this modular schema would have better served as an inheritable class system, which would have opened the possibility to individual functionality per call. That is to say, that the import of a single object that provides all respective API calls, may have been beneficial as a set of individual classes that could be imported on a per requirement basis.

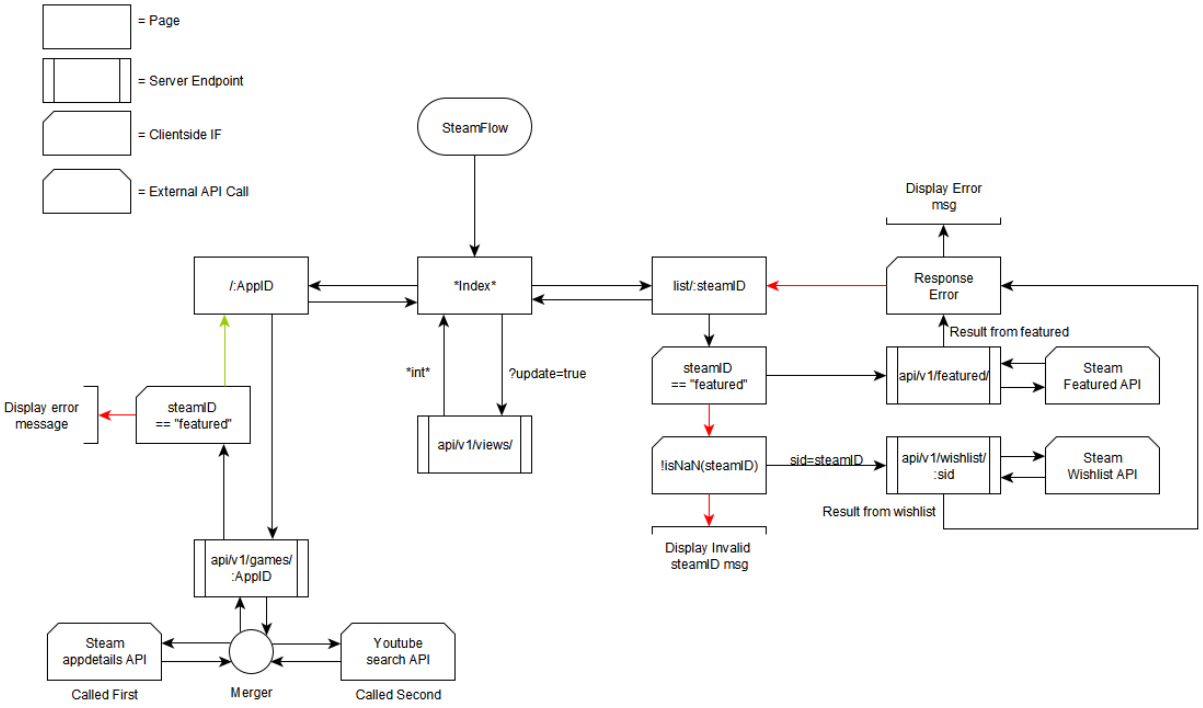
In terms of issues, the overarching personal problem was frontend. My experience as a full stack developer saw me never actually using or implementing React, an unfortunate truth. Due

to this lack of experience, it took me sometime to adjust to the stylization. Inline with this, I was unaware the full difference between functional and class components and was forced to wrap some specific class components in functional components as a response.

Architecture and Data Flow

Frontend & Backend

Explain how your system operates, making it clear how data flows around the system through requests and responses. You are describing the overall architecture of your application at a source code level. The description above tells us something of the application's use. Now we want to see how that maps to the code organization – show us how your code is organized and tell us how you have split the responsibilities. We should get some sense of how the application works and how the data and control flows around. You may also find it helpful to show us screen grabs of code if that makes your points clearer. Tell us anything you think we need to know about how you have structured the application and made it work, but there also a section below to describe problems.



In the case of individual API's both API's follow this structure:

```
router.get("/api/v1/steam/:endpoint", Token.VerifyAuth, async (req, res) => {
    if (req.params.endpoint in exports.paths) {
        var result = await exports.paths[req.params.endpoint](req.query);
        if (result.status == 200) {
            res.status(200).json(result.data);
            return;
        }
        else {
            res.sendStatus(result.status);
        }
    }
    else
        res.sendStatus(404);
});
```

The joint API however, accesses `exports.paths` directly through a common module removing any additional delay or possible problem on other platforms imposing loopback restrictions.

Client side

In terms of client-side data flow, any data outlined above is handled through the joint API with any additional formatting handled client side. The instances of client-side post processing are the lists and app page.

The list page uses several functions to convert a featured list and a wish list into a common data structure that is then displayed.

The app page implements a small amount of math to convert steam reviews into their shorthand version, and additionally, extracts the applications major information into its own state for looping and displaying.

Deployment and the Use of Docker

For deployment and submission, there is an included docker-compose file. This is present as the application can use Redis for response caching. It is highly recommended that it is used with Redis as it helps with performance and reducing the stress on Steam and Youtubes API's. To run in cache-less mode the environment variable redisConfig must contain `"Disabled":true` in the root, Redis will however, be mandatory otherwise.

Additionally, as this form of deployment is highly recommended as all environment variables contained within the image are present within the compose and are preconfigured for deployment.

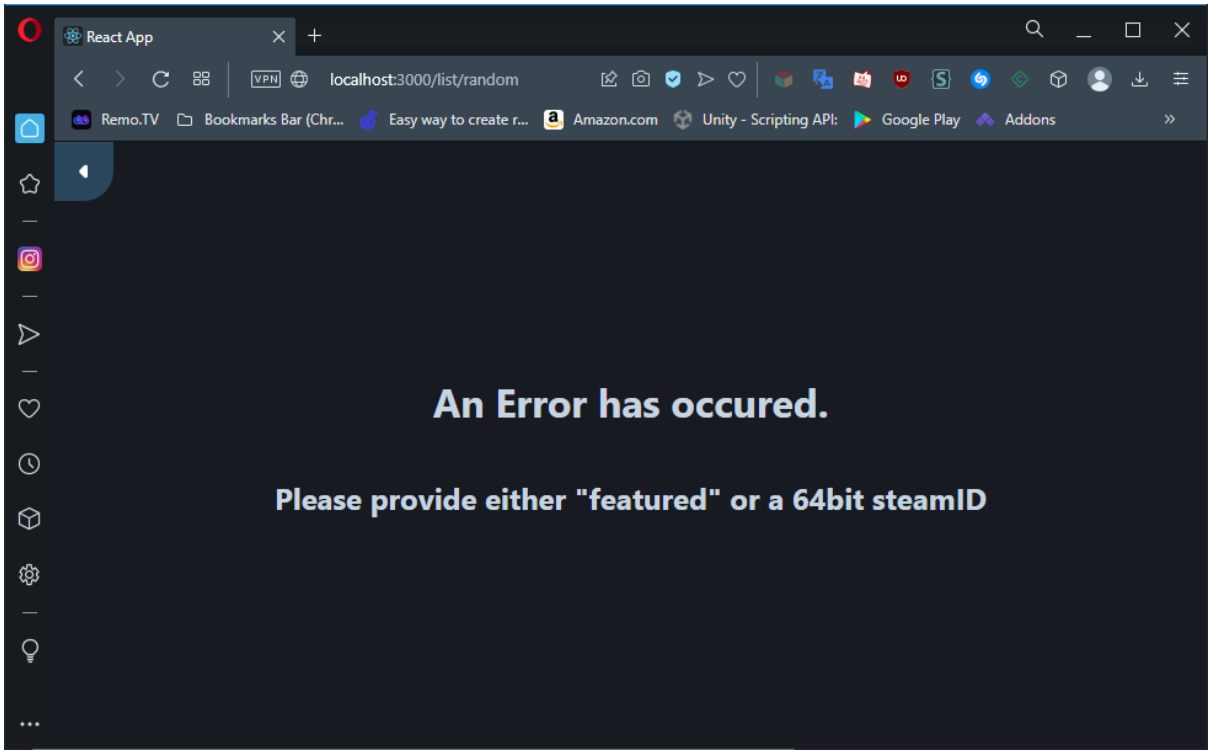
Test plan

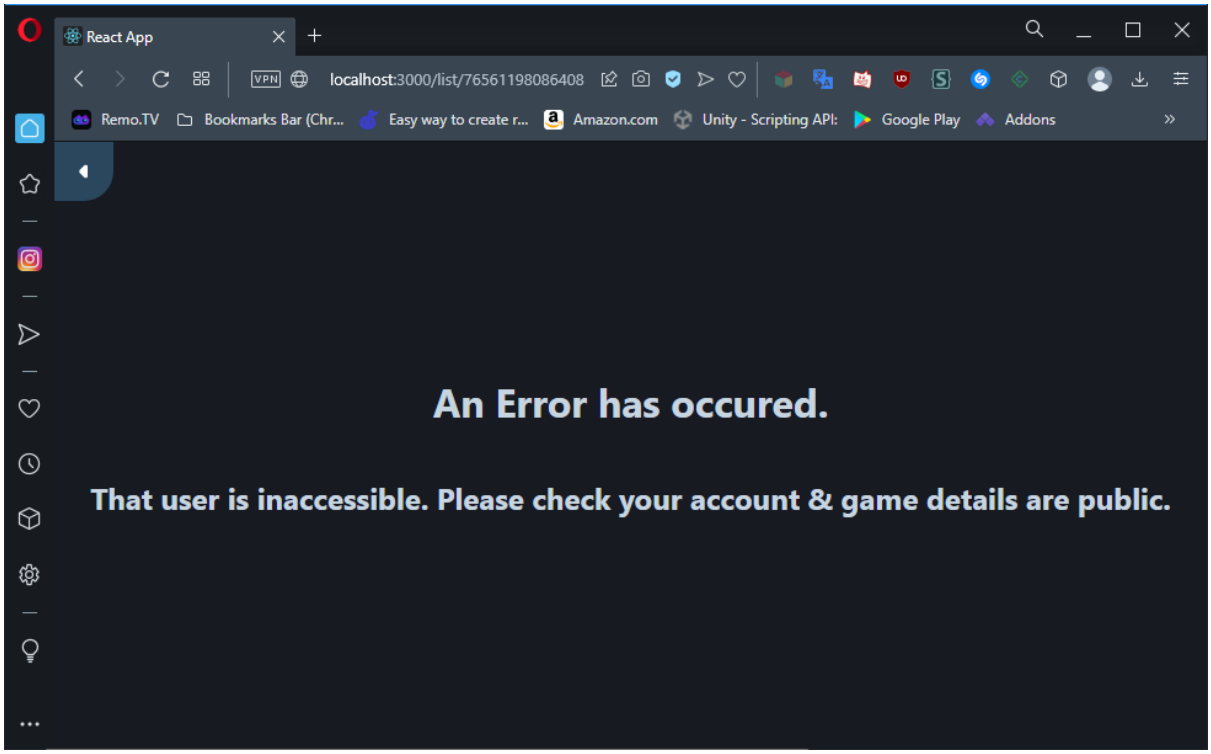
The dominant form of testing applied was manual testing in tandem with Integrated testing. As stated prior the structure of endpoints are asynchronous functions which reflect the action of requesting from their specific endpoint. As such it was easy to identify and determine notable errors that can occur and plan for them, at which point the testing scheme shifted from manual towards integration testing as errors are handled by the caller

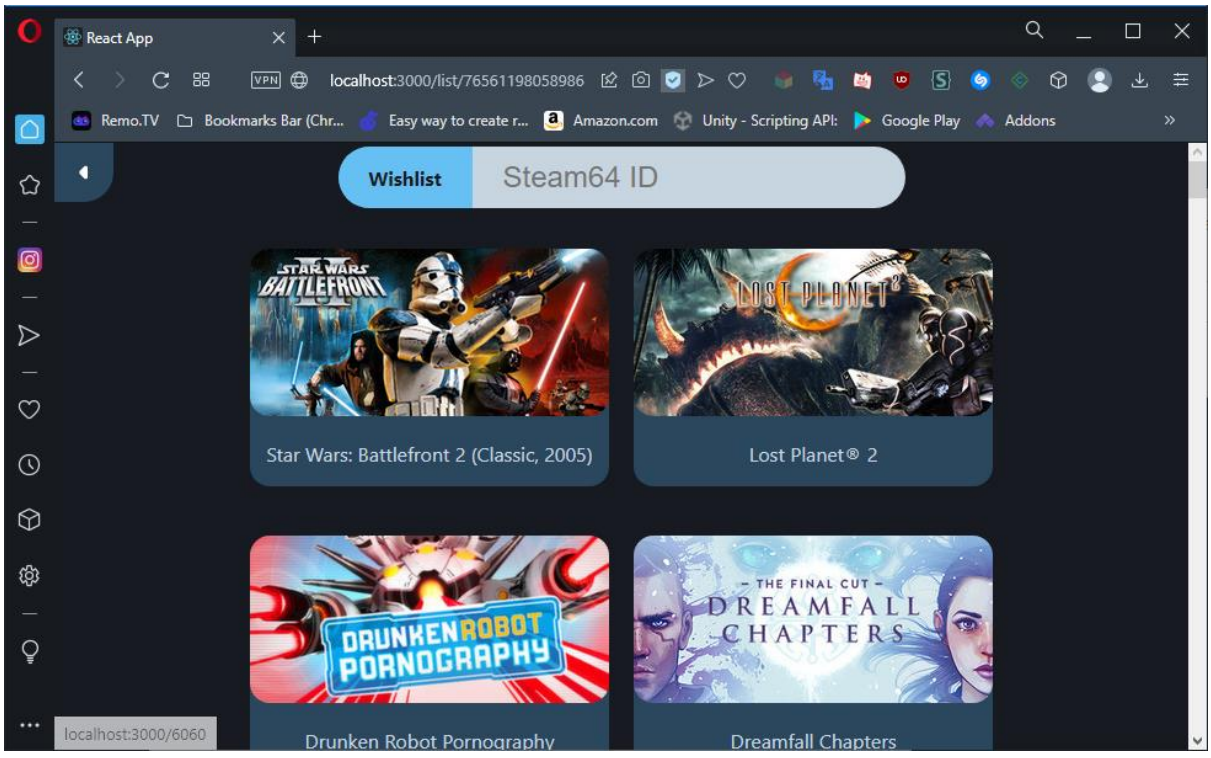
```
Redis cache Disabled
Attempting to validate DynamoDB...
DynamoDB table Validated
Application listening on 3080
730 targeted
730 not cached
730 steam resolved valid
730 Yt passed
730 joined result passed
```

rather than the endpoint function.

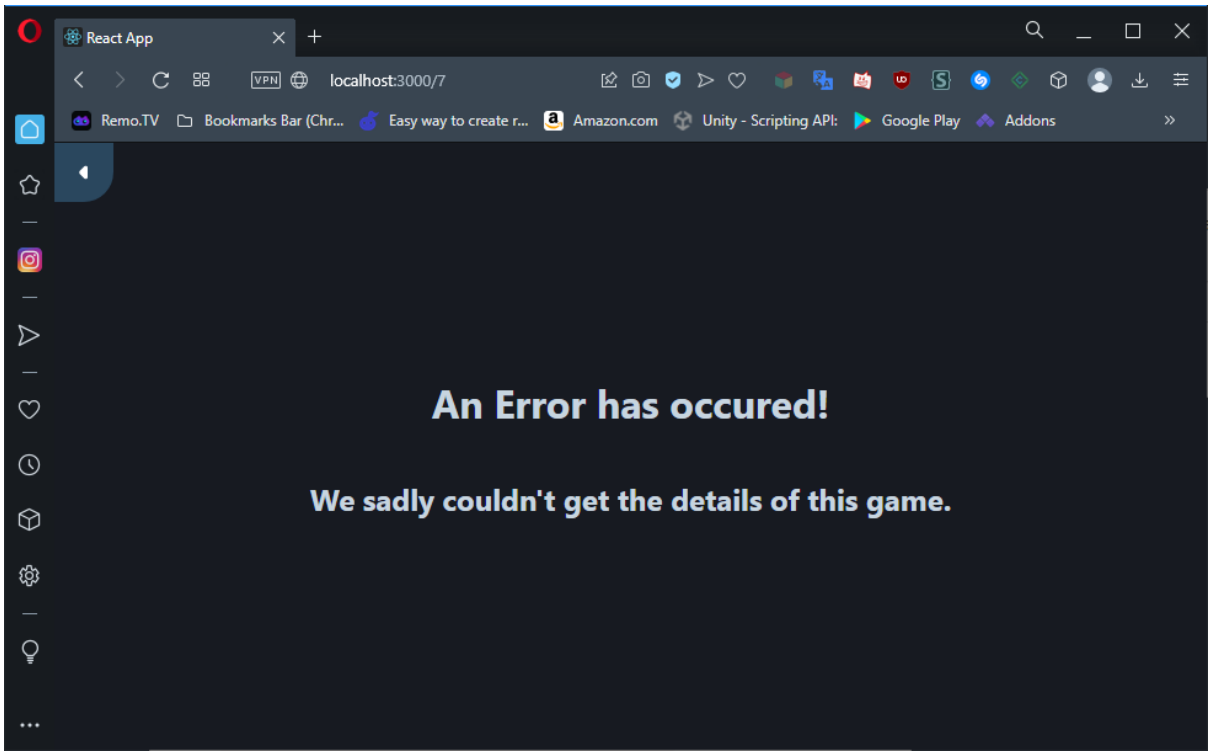
```
Attempting to validate DynamoDB...
Redis client connected.
DynamoDB table Validated
Application listening on 8080
730 targeted
730 not cached
730 steam resolved valid
730 Yt passed
730 joined result passed
730 now cached
730 expirer set
```




React App
localhost:3000/list/76561198086408
Remo.TV
Bookmarks Bar (Chr...
Easy way to create r...
Amazon.com
Unity - Scripting API:
Google Play
Addons
An Error has occured.
That user is inaccessible. Please check your account & game details are public.


React App
localhost:3000/list/76561198058986
Remo.TV
Bookmarks Bar (Chr...
Easy way to create r...
Amazon.com
Unity - Scripting API:
Google Play
Addons
Wishlist
Steam64 ID
Star Wars: Battlefront 2 (Classic, 2005)
Lost Planet® 2
Drunken Robot Pornography
Dreamfall Chapters
localhost:3000/6060


React App
localhost:3000/730
Remo.TV
Bookmarks Bar (Chr...
Easy way to create r...
Amazon.com
Unity - Scripting API:
Google Play
Addons
Counter-Strike: Global Offensive
Total Reviews: 3.49M
Counter-Strike: Global Offensive (CS: GO) expands upon the team-based action gameplay that it pioneered when it was launched 19 years ago.
CS: GO features new maps, characters, weapons, and game modes, and delivers updated versions of the classic CS content (de_dust2, etc.).
"Counter-Strike took the gaming industry by surprise when the unlikely MOD became the most played online PC action game in the world almost immediately after its release in August 1999," said Doug Lombardi at Valve. "For the past 12 years, it has continued to be one of the most-played games in the world, headline competitive gaming tournaments and selling over 25 million units worldwide across the franchise. CS: GO promises to expand on CS' award-winning gameplay and deliver it to gamers on the PC as well as the next gen consoles and the Mac."
Counter-Strike: Global Offensive (2021) - Gameplay (PC UHD) [4K60FPS]
Counter-Strike: Global Offensive Trailer
Counter-Strike: Global Offensive (2022) - Gameplay UHD


React App
localhost:3000/7
Remo.TV
Bookmarks Bar (Chr...
Easy way to create r...
Amazon.com
Unity - Scripting API:
Google Play
Addons
An Error has occured!
We sadly couldn't get the details of this game.

Service	Task	Result	Notes
Backend	Youtube Query		
Backend	Caching		
Backend	Cache-less functionality		Manages this by replacing all functions with returns if disabed
Backend	Per step test		Per step test are post Promise Redis actions
F/B	Invalid AppID		The common caller for steam used by the joint API returns null and handles sending the error status.
F/B	Inavlid List		
F/B	Invalid SteamID		
F/B	Inaccessible account		
F/B	Valid account		

Difficulties / Exclusions / unresolved & persistent errors /

Majority of development roadblocks were short lived courtesy of numerous developer friends with more specific experience than myself.

One worry, however, is the Steam API. As previously stated, its undocumented in any official manner and whilst allowed by steam the lack of documentation presents a large possibility for unidentified errors to occur which may result in some unforeseen error. Unfortunately, the only real method of avoiding this edge case is to perform a general catch of an error code, then work through the known response codes.

Originally, I intended to implement Cheapshark with Steam, however, after submitting that idea it was made apparent that the deviation in both API's wasn't enough to meet the assignments requirement and thus pivoted me towards the current submission.

Another problem space during development was remaining stateless. Steam themselves do not support any specific endpoint for the request of a user's steam ID as a user's ID is the only defining value for account identification due to the allowance of name overlap. To compensate for this, they act as an OpenID provider for steam verification. Unfortunately, this would require the application to become stateful thus invalidating the submission.

Extensions

Ideally given more time I would have loved to redesign the backend into the aforementioned class driven system and implement Metacritic for scoring and reviews, Twitter for insight into the community and twitch for current streams.

Each of these are underrated sources of insight into a games community and popularity as Twitter is notorious for hastily tweeting about recent in-game updates, Metacritic has a wide scoring system whilst most scores are never added to Steams product pages and twitch not only grants additional content, but also unfiltered gameplay in comparison to YouTube.

Whilst Youtube is an excellent source of videos, I additionally would have loved to integrate more video platform APIs to provide a wide range of content to a user. This includes accessing review websites and filtering the recent results to help users get a first hand opinion of the game.

User guide

Valid Testing entries

Steam User ID's

Technocide: 76561198058986389

This user is a friend who has agreed to allow use of his wish list, as such I am not responsible for any content displayed.

This ID should be valid, and an example of an intended use

Hoggy077: 76561198086408838

This is my personal account where "Game details" is set to private and should be reflected with a 500-server response.

App ID's

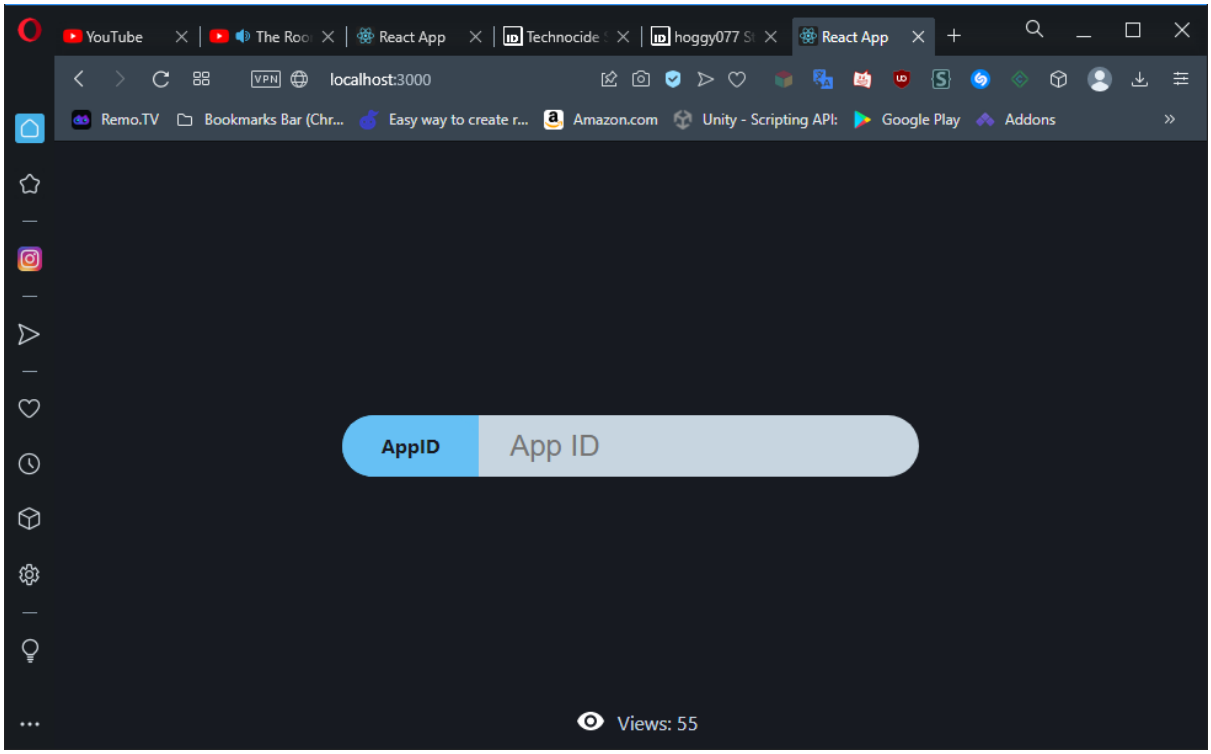
CS:GO: 730 Destiny 2: 1085660 GTA 5: 271590 Payday 2: 218620 Hololive Error: 2062550

Subnautica Below Zero: 848450

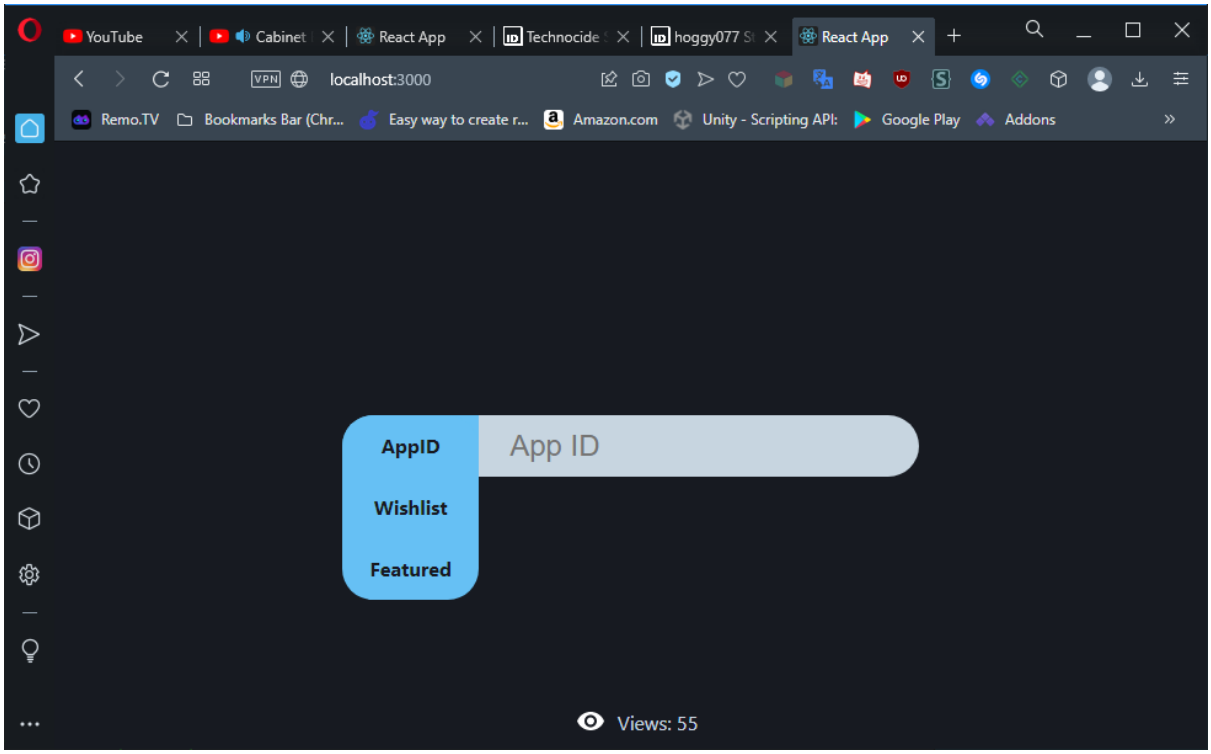
For more appID's I suggest referring to https://steamdb.info

Index

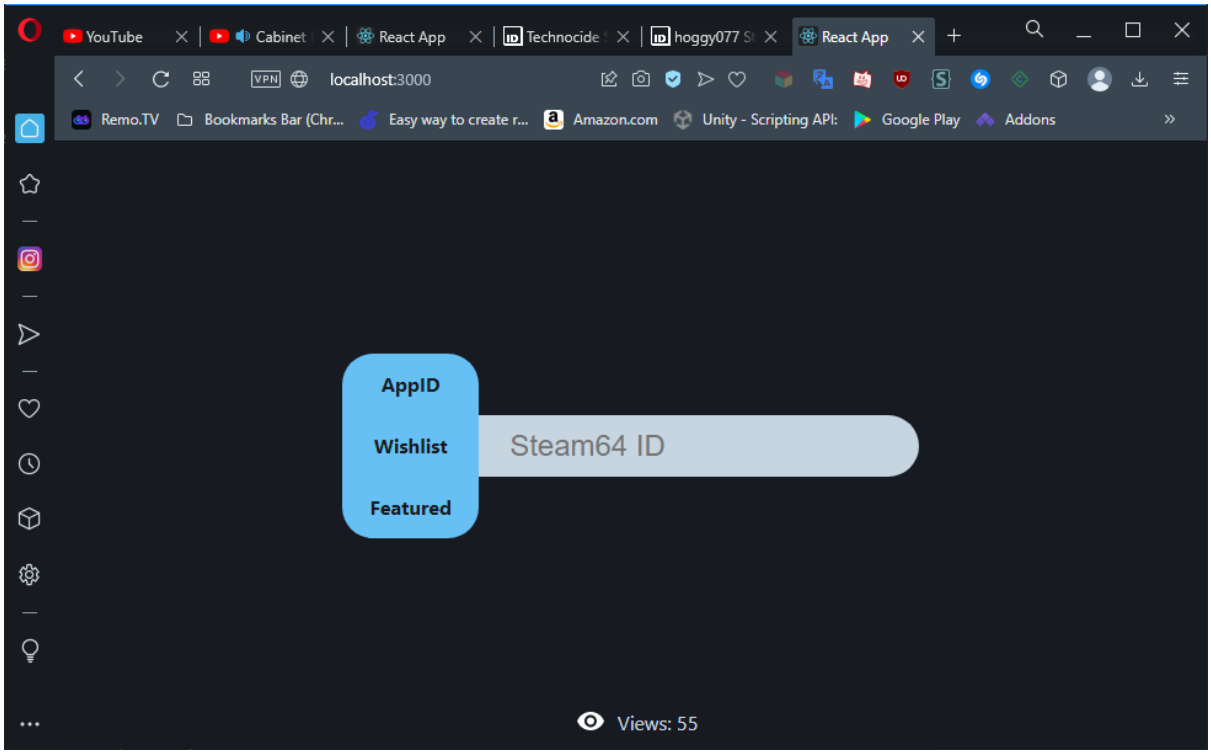
To view a specific game page, insert the app ID into the search bar and press enter.



Pressing "AppID" will open alternative search options



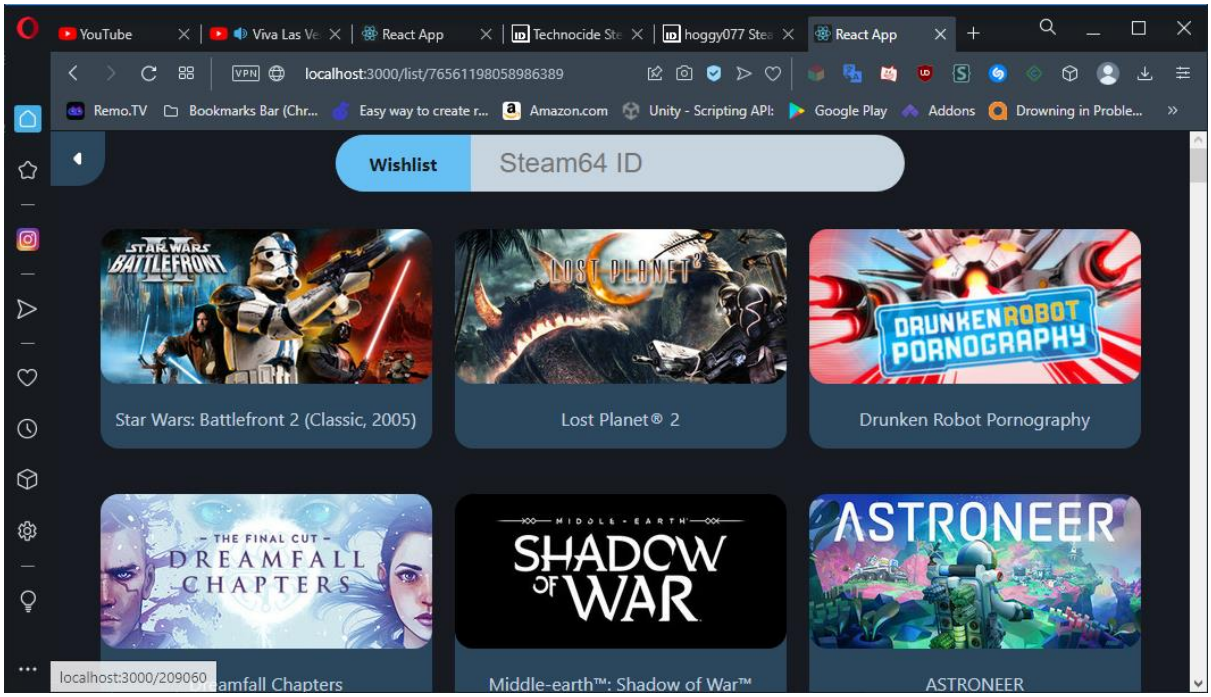
To access a wish list, select the wish list option, insert a steam ID and press enter.



To access featured, select the option, insert any value into the search bar and press enter.

List

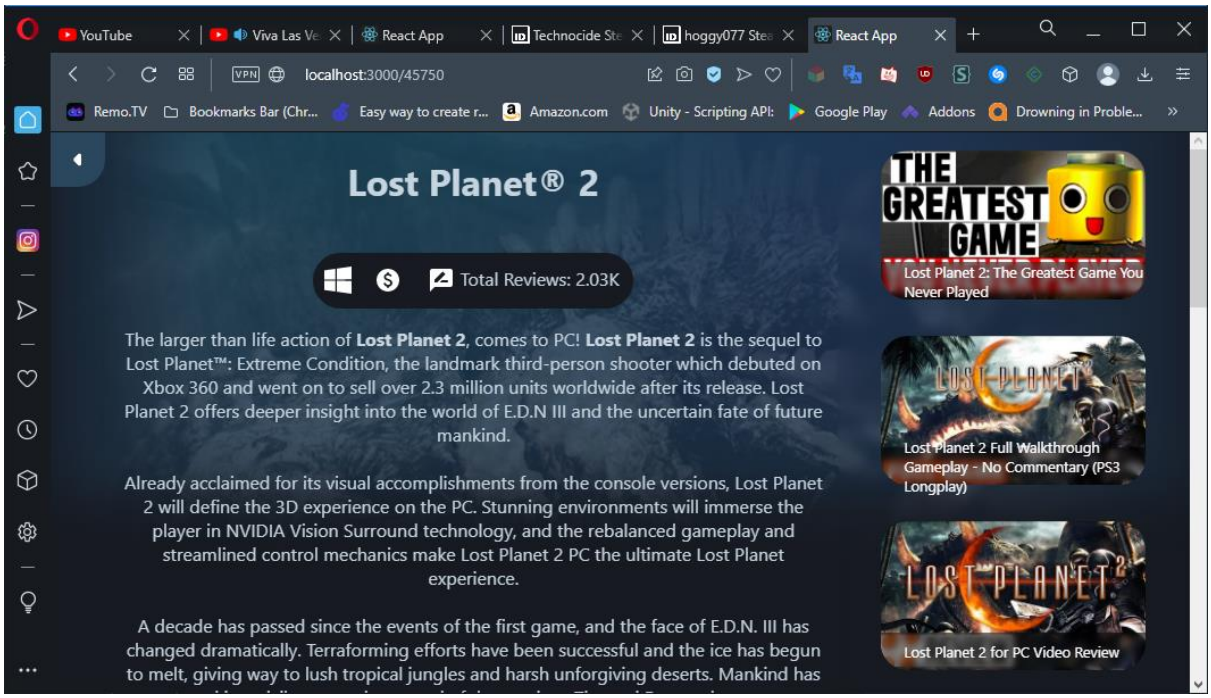
This path is used by both featured and wish list to display games.



Once on a list page, selecting any of the listed games will take you to the corresponding app page

---

App ID



Once on a list page, selecting any of the videos along the right edge will open the corresponding YouTube video. These videos are acquired by name.

Analysis

Question 1: Vendor Lock-in

Steam & Youtube

Frontend

Unfortunately, frontend it would be significant to pivot to any new service this is due to both page linking and data structure and the frontend on the App page handles the bulk of trimming steam entries and would require significant modification to respond to another API.

Backend

The modular system implemented would benefit in this case. With the exclusion of the individual service APIs provided, the only notable modification would be the replacement of an API request in its respective entry. Beyond this the only other points of modification would be error handling and error responses as response statuses are dependent on the service and endpoint.

Steam

Many of Steams competitors refuse to provide any storefront API meaning should Steam suddenly shut down the only alternative is web scraping APIs. Those that do provide an API restrict storefront access to their own services to avoid abuse of the endpoint. This means any major platform would either be accessible via an authorized third party or would require contact for access, which many document that authorization is for user verification and not storefront access.

Youtube

Youtube fortunately has numerous alternative sites, with the most notable one being DailyMotion as it follows a similar query and ID scheme as Youtube allowing for modification.

Question 2: Container Deployment

Substantial deployments have numerous benefits using docker over VM's. Namely the ability to rapidly deploy a change with minimal downtime as a segmented service would only lose functionality of the container for the period of stopping the container, pulling the new image, and starting it again.

For this I would like to focus on Cheapshark as they integrate several authorized game selling sites into a single uniform one. One notable feature is their API imposes no rate limit at the moment, however, given the amount of data exposed to the user, I would believe they implement horizontally scaled containers. Namely because any request to an API is an asynchronous event and should a container currently be processing several requests; it would be faster to launch another container then a VM. Given the nature of the service, it would also be safe to assume it manages a database in which different deals expire at their noted times.

As standard server's run a version of Linux generally, it would be safe to assume backend supplies data pulled from a database or cache, which is maintained by another portion of the service. Cheapshark also lacks any form of user login, benefitting from containers being designed for stateless deployment, whilst negating any possible problems that may arise from the reduced security in comparison to a VM. Additionally, if I am correct, an image would compose of a 2 individual sections, the module that serves the response and the module that accesses the database, meaning containers would have bot a size, resource and launch time advantage to a VM, which would allow for more instances to serve responses on a single machine then there would be VM's.

In this case I think it would be beneficial to deploy with docker over VM's despite the reduced security, as there is nothing that would suggest anything of value being kept between sessions or requests.

Personally however, if deploying at scale in this instance I would implement a NoSQL or Relational Database on a separate EC2 instance for security.

Redis would once again be my pick of choice, namely because the data they serve is re-obtainable quickly and can be served and stored on a per/request basis, this would make Redis

Database + Append Only File (RDB + AOF) desirable as ideally it would mean majority of the database would be safe in the event of a sudden shutdown, would require less forking, and therefor be up longer. Unfortunately, this comes with increased overhead to ensure data safety and avoid corruption including in-depth management of event timing.

Additionally, disaster recovery could be implemented using additional external services. For example, they could encrypt their RDB+AOF database and store it within an S3 bucket should something happen.

Amazon RDS and DynamoDB would both offer exceptional external persistence without the additional overhead of snapshotting and timing events as well.

Appendices

Docker-compose for response caching

```
services:
  web:
    build: .
    ports:
      - "8080:8080"
    environment:
      - DEV_Key=s6RjBwBMUo4uf0qZLqwvCXmxgm_AyEsFjl1PNH7rqqU
      - YOUTUBE_Key=AIzaSyCwaLoOE9q9pl4L1QLTEo73PZm6swHyi2w
      - redisConfig={"socket":{"host":"redis","port":6379}}
      - AWS_ACCESS_KEY_ID=*insert access key*
      - AWS_SECRET_ACCESS_KEY=*insert secret*
      - AWS_SESSION_TOKEN=*Insert token*
    depends_on:
      - redis
  redis:
    image: redis
```