Cover Page

Project Name

BY

STUDENT: EOIN LERNIHAN

Number: G00365942

SUPERVISOR: JOSEPH CORR

GitHub: https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22

Table of Contents

Introduction

Overview of Project

This project investigates the feasibility of serverless in supporting a website.

Serverless is defined as a method of providing backend services on an as-used basis. Servers are still used, but a company that gets backend services from a serverless vendor is charged based on usage, not a fixed amount of bandwidth or number of servers.[1] (https://www.cloudflare.com/learning/serverless/what-is-serverless/ .)

This feasibility of serverless been used in a website was validated by creating a small website developed in React. All the backend was provided via serverless technologies. The backend consisted of a REST API (created in AWS API Gateway) which was backed by several AWS lambda functions. AWS Lambda is in Amazon version of serverless, other examples are Google Cloud Functions and Microsoft Azure Functions. The AWS Lambda functions were written in Java and built using Maven. The database was provided by MongoDB Serverless.

The scope of the project was to see was it possible for serverless to compete with a traditional server for a small sized business or site

The criteria used for evaluation

- Response time
- Potential costs
- Ability to deliver Functionality
- Learning curve

Overview of website

The website demonstrates the functionality to allow players to join a gaming Tournament, after they are registered as a User. The Tournaments are associated with shops which are administrated by Admin. Each Tournament will have a max number of players. When the website is reached initially the User is prompted to login via their username and password. It then they can see their User profile and the Tournaments that have already joined.

The website is inspired by the Tournaments that game shops such as Dungeons and Donuts run on a weekly basis. The traffic on would be from the local area and would be of a low volume in nature. The traffic would be around when the tournament is released. Subsequent traffic/visits would be close to zero.

The website could be extended to allow for

1. The opportunity for game shops to promote tournaments that they are hosting to a target audience
2. In Game Tracking
    a. Allows shop admins to keep track of games in tournament results

b. Allows shop admins to keep track of the players in the games

Overview of Serverless

This section describes Serverless and some of the perceived advantages

Serverless definition

Serverless computing is a method of providing backend services on an as-used basis. Servers are still used, but a company that gets backend services from a serverless vendor is charged based on usage, not a fixed amount of bandwidth or number of servers. The following is a brief review of Serverless

Pros and Cons of Serverless

Pros

LOWER COSTS

With serverless, you only pay for what you use—there are no hardware costs and no costs when your services are not in use. Reduced cost is one of the main advantages of going serverless. If your services aren't heavily used all the time, then the 'pay-as-you-go' model gives you optimal resource utilization without paying for idle server time.

FEWER THINGS TO WORRY ABOUT

You are relieved from worrying about if the latest security has been applied. Because your servers are now in the hands of a third-party service provider, you no longer need to be concerned with keeping up with patches and bug fixes.

ENHANCED SCALABILITY

You may want to build a viral app like Instagram or Uber, but would you risk provisioning infrastructure just in case? With serverless architecture, you do not have to think twice about it because of its ability to automatically scale with along with the traffic volumes. Scaling also depends on the location of users and their network connection. Serverless providers have points of presence around all users, which diminishes delays and allows apps to perform as they should, regardless of your geographic location.

MORE FOCUS ON USER EXPERIENCE

Your users do not care about infrastructure; they care about features. With server maintenance out of mind, companies can dedicate more time and effort to improving the customer-facing elements. A quality UX design is essential if you want to keep users engaged with your app.

Serverless computing allows developers to build apps without the headache of managing infrastructure. More specially, it enables them to write in serverless code without having to provision a server to ensure its functionality or having to create test environments on a server and maintain the server uptime

Cons

VENDOR LOCK-IN

You are playing by their rules now. Once you give control of your servers to third-party providers, you lose control over the hardware, the run times, and updates. This can create issues in consistency and limit the resources you have available.

Additionally, when you commit to a service provider, you are in it for the long run. If you build your application on one serverless infrastructure, and then you want to transition to another vendor, they don't make it easy to switch. It is a very difficult process, and you may need to re-engineer your application if you wish to do so. Plus, there is a risk that a vendor will change its pricing or service terms or even stop offering serverless options.

UNSUITABLE FOR LONG-TERM TASKS

Serverless is great for short-term or real-time processes like sending out emails. However, for longer duration tasks, where functions are running constantly, you may end up paying more for computer time than when paying for a reserved instance. A task such as uploading large video files would require additional functions to be called on.

man sending email for example, Lambda gives you five minutes to complete a task. If it takes longer, you must call another function, and so on until the task is complete. Lambda also imposes a limit on how many simultaneous functions can be running, so if you execute too many functions, you will "denial of service" (DoS) your production applications.

COLD STARTS

You only pay for what you use, but if you do not use your function often, you will pay with a dramatic performance penalty. Hosted functions suffer from a cold-start penalty and can be terribly slow the first time they are called in a while.

However, you can attempt to minimize cold starts by keeping your functions small and precise, as cold starts increase with code size and memory. This will help keep your functions warm (but the drawback is that it is complicated and inefficient to manage numerous small functions).

COMPLEXITY

The learning curve in serverless application is a steep one. Units of integration with serverless are a lot smaller than with other architectures. This requires extra time to go into organizing the functions so they work in line with your data. There can also be problems with deployment and versioning.

AWS

Functions are time-restricted, allowing you a maximum of five minutes. If you have a task that requires a large amount of data that could exceed these runtime limits, a lot of effort will have to go into rewriting the code in another architecture. You may need to deploy a separate piece of code for every function in your entire logical application.

There are frameworks that allow you to deploy a collection of AWS resources, but it's more difficult to update multiple functions than to update a monolithic architecture. However, if you want to migrate your application over to serverless you need to tackle the complicated task of splitting your monolithic application to microservices.

From a business perspective, the disadvantages of vendor lock-in and loss of control could be damaging. Even though it may be cost-effective, it is also unpredictable because the number of executions is not predefined.

For developers, serverless is a relatively new technology. It can be very complex and testing becomes tricky. If not developed correctly, companies using serverless can lose customers due to poor performance.

Methodology

Overview

Over the course of the 2 semesters there were several phases to the development of the project. These phases were

1. Technology research and discovery
2. Proof of concept of the technologies
3. Architecture Design and Functional scope
4. Backend development (iterative) and deployment
5. Frontend Development
6. Issue Resolution/Write up

During each of these phases, the agile process was used to prioritize what next piece of work should be worked on. The work was broken into weekly targets. Although weekly targets were set, they were often missed, due to other demand of other projects especially towards the end of semester one. When this occurred, a reprioritization exercise was performed so that to ensure the most critical work was been performed. A more detail how and what occurred in each phase is detailed below

A weekly meeting with the project supervisor allowed for review of progress, feedback on progress and some direction in terms of planning and prioritization.

Technology research and discovery

This phase researched the technologies to know whether it was possible to see what components were available. The research started with technologies that were used during the course. Research areas included DB, serverless function providers, Frontend technology and how to integrate the frontend to the serverless functions. Another aspect of the research was to ensure that any technology selected would have no major costs.

The outcome of this phase was the selected technologies that were going to be used.

Proof of concept of the technologies

This phase was to create a hello world application using each of the technologies. An account(student) was created in MongoDB and an AWS.

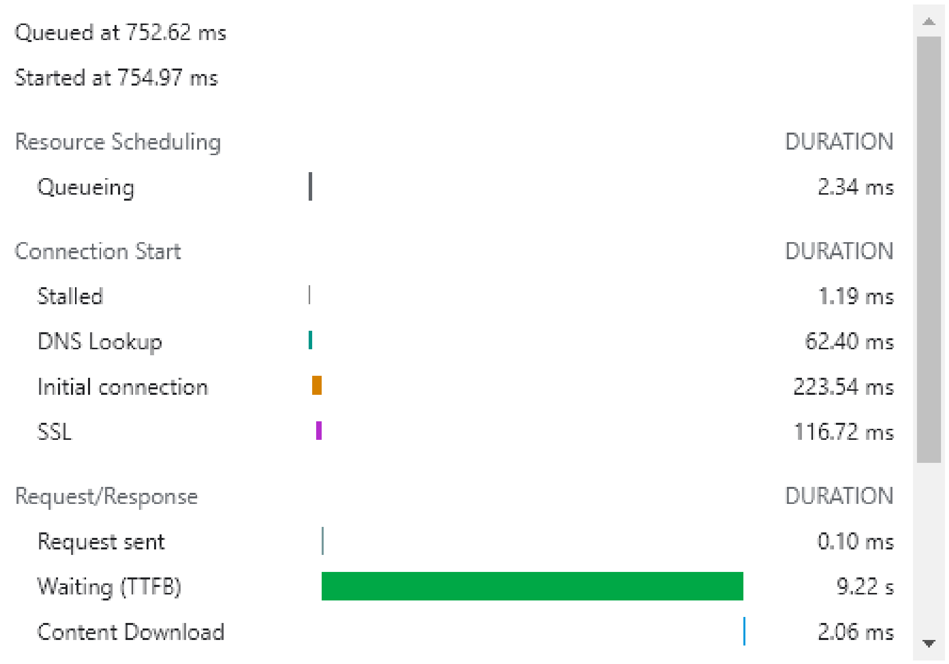
- In MongoDB a Database was created with one collection, data inserted manually using the console.
- A java jar was created with one large class which had a method to receive the lambda event. (link to commit). This was uploaded directly to the lambda and tested using the lambda test (picture). The major challenge was the connection to the MongoDB serverless. At this stage Maven was

introduced to help introduce the necessary jars/libraries to find the correct libraries to use for MongoDB connection and interaction. [2](https://www.baeldung.com/java-mongodb)

- An API gateway was created to connect to the lambda without proxy integration.[3] (https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html)
  https://xxxxxxx.execute-api.us-east-2.amazonaws.com/Develop/user

- A basic page was created to call the API Gateway and received information fetched from the DB.

The outcome of this phase was that it was possible to link these technologies together and the performance of the HTTP calls after Cold Start were acceptable



Queued at 752.62 ms
Started at 754.97 ms

Resource Scheduling	DURATION
Queueing	2.34 ms

Connection Start	DURATION
Stalled	1.19 ms
DNS Lookup	62.40 ms
Initial connection	223.54 ms
SSL	116.72 ms

Request/Response	DURATION
Request sent	0.10 ms
Waiting (TTFB)	9.22 s
Content Download	2.06 ms

Architecture Design and Functional scope

This phase was to create an architecture design and to scope the functionality that was to be provided. The architecture design was created in powerpoint and the scope functionality was a set of mockup of the UI screens. These were presented to a group of lecturers. on 14 December 2021

[4]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/tree/main/WireFrames

[5]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/blob/main/Document/Christmas-Presentation.pdf

[6]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/commit/26cccc049279ae12e38033d3b619299ae00a011d

The outcome of the phase was that this project warranted a final year project. Also, feedback was taken on board regarding the scope of the project.

Backend development (iterative) and deployment

This phase was done on an iterative basis. The first couple of weeks was the creation of collection in the Database. Once the collections were created, focused shifted to the java backend. The structure of the java was created with some examples showing a Data access layer (DAO) and Controllers. A decision to only have one jar for all lambdas was made. This was to allow for easy development and to easily share code between the lambda, such as Database connection code.

Initially tests of this phase main function in the java. Upload to the existing lambda function with testing of the integration with one of the API Gateway.

During this phase there was an issue with the serialization to JSON and HTTP response codes. The issue was

1. There tournament structure had a nested JSON structure and AWS Lambda was doing the serialisation to JSON.
2. The API gateway was handling the HTTP response codes and it was difficult to get the correct response codes, such as NOT_FOUND 404.

The project then introduced proxy integration based on this example.

[7](https://www.baeldung.com/aws-lambda-api-gateway)

During this phase 12 Lambda were created, and the upload of the jar to each lambda was slow. The discovery of the ability to upload once to a S3 location and then link to lambda reduced this upload time.

Finally, in the AWS APIgateway console the API Resource structure was configured and integrated with the AWS Lambda. Each Resource was tested using the console on the API Gateway.

The outcomes of this phase were

1. A MongoDB Database with 3 collections (including the Games collections having a nest list of players id)
2. A Maven project
3. 12 AWS lambdas (Serverless)
4. A REST API provided by the AWS
5. An upload, API gateway deployment and AWS debugging process via AWS Cloudwatch logs.

Frontend Development

This phase was done on an iterative basis. Visual code was used as the IDE , the REACT the technologies.

REACT was used in 3 previous projects, hence the reason for the selection of this technology. Challenges during this phase was CORS issues which resulted from different scenarios

1. A REST endpoint not available – normally this would result in a 404 as the endpoint would not exists
2. No CORS headers been returned from the Java – proxy integration means the http response must be created in the JAVA layer
3. No OPTIONS method been defined on the Resources - this was to faciltate pre flight requests
4. Enabling CORS on API gateway – Not sure if this is required if proxy integration is used.

The outcomes of this phase were

1. A website that could demonstrate functionality from serverless
2. An application that is well integrated and tested
3. CORS issue resolution

Issue Resolution/Write up

This phase was the write up of the project. The outcome of this phase is this document.

Technology Review

Overview

In this chapter the technologies used on this project will be explored and the reasons why these technologies were selected. It will include an explanation on what the technology used for, how it was configured/coded and how it was utilized to ensure that the online website was performant. The chapter discusses the technologies in the following categories

1. Database
    a. MongoDB
2. Middle layer
    a. Java , Libaries used, Maven build plugins
    b. AWS Lambda
    c. AWS Gateway
    d. S3
    e. AWS CloudWatch logs
    f. AWS Identify and Access Management
3. Frontend
    a. React

Database

The primary function of the database is to persist client data and to allow for retrival of such data. There were a few possible solutions. The following was some of the considerations such as Realtional such as MySQL, nonrelational such as MongoDB or DynamoDB, how was it charged and were was it located. MongoDB was selected because

- A serverless version released in July 2016
- [8] (https://www.mongodb.com/blog/post/introducing-serverless-instances-mongodb-atlas-now-available-preview). This offered a "hand free solution" with the operations and provisioning of servers.
- A course module in MongoDB had been completed in 3rd Year (XXXX)

Cost

According to MongoDB [9](https://www.mongodb.com/pricing) their pricing is as follows:

Read Processing Unit (RPU)	Number of read operations* to the database *You are charged one RPU for each document read (up to 4KB)	$0.30 for the first 5 million per day* *Daily RPU tiers: Next 20 million: $0.25/million

	or for each index read (up to 256 bytes).	Next 50 million: $0.20/million Next 100 million: $0.15/million
Write Processing Unit (WPU)	Number of write operations* to the database *You are charged one WPU for each combined document and index written (up to 1KB).	$1.25 per million
Storage	Data and indexes stored on the database	$0.25/GB-month
Backup	Download and restore of backup snapshots* *2 free daily snapshots included per serverless instance	$2.50/hour* *To download or restore the data
Data Transfer	Inbound/outbound data to/from the database	$0.015 - $0.10/GB* *Depending on traffic source and destination

For the duration of this project, it was mainly free with the exception of 0.52 Euro which was caused by several infinite loops on the frontend.

Middle layer

Overview

The primary function of the Middle Layer is to provide a bridge between the Frontend and the Database. It is also known as an application or logic layer. In this project the layer translates requests from the frontend in the form of HTTP Requests into interaction with the database and returns the result of those interaction as HTTP responses. An example is a HTTP request, method is GET, with a URL https://XXXXXXXX.execute-api.us-east-2.amazonaws.com/Develop/touraments?inGame=0&userName=u . The middle layer executes a search on MongoDB for all tournaments that include a player call u . The data is returned to the middle is then transforms into a JSON string in the payload of the HTTP response with a HTTP Status of 200.

```
{"tournaments":[{
 "owner": "owner337407",
 "game": "game337407",
 "gameMode": "gameMode337407",
 "maxPlayers": 12,
 "players": [
 "gkj"
 ],
 "number": 337407,
 "description": "description337407",
 "time": "Apr 5, 2022, 11:00:00 PM",
 "duration": 60,
 "isPublic": true,
 "numRounds": 3
}]}
```

The technologies used in this layer were

AWS API Gateway

AWS Lambda

Java, Libraries, Maven build plugins

S3

AWS CloudWatch logs

AWS Identify and Access Management

AWS API Gateway

According to AWS document an API Gateway is "is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications."[10](https://aws.amazon.com/api-gateway/#:~:text=Amazon%20API%20Gateway%20is%20a,functionality%20from%20your%20backend%20services).

AWS API Gateway was selected

1. The free tier
2. There are ample examples and tutorials integrating it AWS Lambda

Cost

Under AWS Free Tier there is one million API calls received for REST APIs per month for up to 12 months(https://aws.amazon.com/api-gateway/pricing/?loc=ft#Free_Tier). Subsequently charges range from $1.00 per million to $0.90 per million per month.

AWS Lambda

AWS Lambda is Amazon version of a serverless functional platform. It provided the computer processing and memory for the execution of the Java layer. Other alternative solutions are Google Functions or Microsoft Azure functions. It handles requests from the API gateway and executed the configured Java method.

ie.gmit.sw.controller.TournamentsController::getAllTournaments

AWS Lambda was selected

1. The free tier
2. Had some basic experience in AWS

Cost

The cost is calculated based on the number of executions, duration of the computation by the memory assigned for that duration. If you assign a large memory allocation to the lambda the cost will increase. More details here (https://aws.amazon.com/lambda/pricing/?loc=ft#Free_Tier) . The free tier was sufficient for this project as it included 1 million requests 4 hundred thousand GB-seconds of computer power per month.

Java, Libraries, Maven build plugins

The Java layer was the primary component of this layer. The Java jar is executed and invoked on the AWS lambda. The lambda executes a java method which then translates request into an interaction on the MongoDB and returns the data and/or a result.

Java 8 was selected as most examples were with Java 8. Alternative technologies were Go, Python, Ruby, Nodejs, .Net (C# or PowerShell). The alternative version of java was 11.

Cost

All software used is Free

Apache Maven

Apache Maven is a software used in this project for managing

1. The libraries used in the Project
2. The build process of creating the JAR, including compiling the project and running the unit tests

Gson

"Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have *access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals."*

[11] https://github.com/google/gson

Gson was selected due to

- baeldung examples contained it.
- Simplicity of toJson/fromJson in the simple cases

aws-lambda-java-core
"Minimal set of interface definitions for Java support in AWS Lambda"

aws-lambda-java-core was selected because it is required for AWS Lambda

junit-jupiter-api
JUnit 5 testing framework. Needed for JUnit tests.

mongodb-driver-core
"The Java operations layer for the MongoDB Java Driver. Third parties can wrap this layer to provide custom higher-level APIs"

Required for interacting with MongoDB

mongodb-driver-sync
"The MongoDB Synchronous Driver"

It is required for interacting with MongoDB

logback-classic
Loggin framework to ensure the formatting of logs are consistent.

junit-jupiter-engine
The platform for running JUnit 5

com.googlecode.json-simple
"A simple Java toolkit for JSON"

It is Required as a companion for Gson

maven-surefire-plugin
Platform for executing JUnit test in maven build cycle

maven-shade-plugin
"Repackages the project classes together with their dependencies into a single uber-jar, optionally renaming classes or removing unused classes."

It is used for creating fat jar which contained all of the libraries and classes that a required by the AWS Lambdas

CloudWatch

"CloudWatch enables you to monitor your complete stack (applications, infrastructure, and services) and use alarms, logs, and events data to take automated actions and reduce mean time to resolution (MTTR). This frees up important resources and allows you to focus on building applications and business value." [12](https://aws.amazon.com/cloudwatch/features/#:~:text=CloudWatch%20enables%20you%20to%20monitor,building%20applications%20and%20business%20value.)

The project use it for viewing logs from the AWS Lambdas and AWS API Gateway. It is available via system console in AWS. It provided for quicker and more accurate live debugging

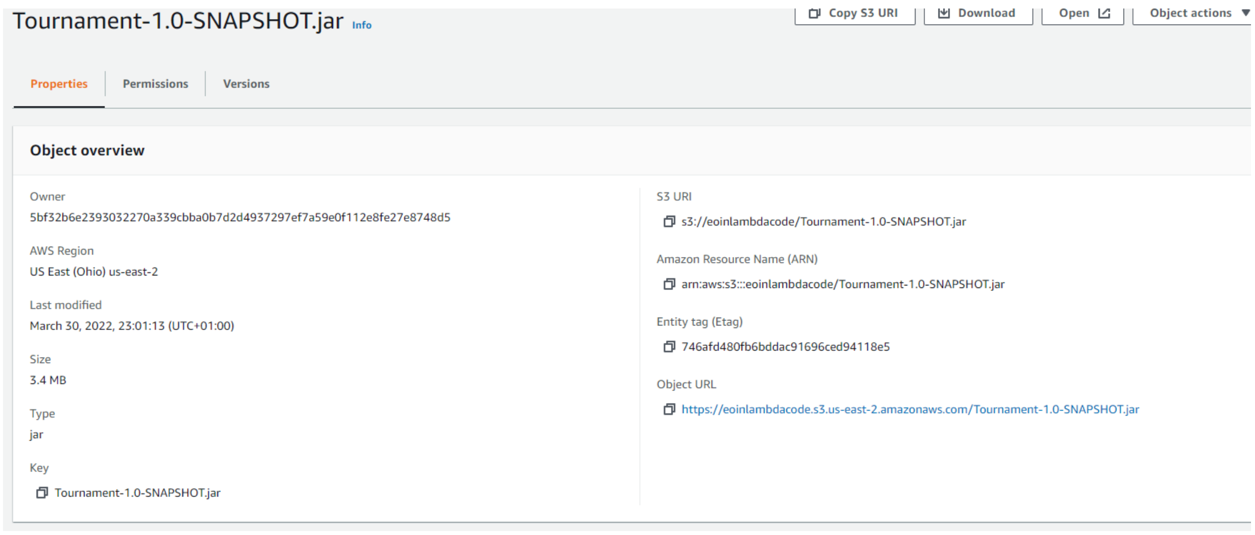
Cost

Free - View, filter, and download the most recent 90 days of your account activity for all management events in supported AWS services.

Additional rates will apply for any data event and for additional copies of management events beyond the first copy in each region. See product information for more detail

S3

S3 Buckets are objects files system in the cloud. They are used to contain the current most updated maven built jar on the cloud



Tournament-1.0-SNAPSHOT.jar Info

Object overview

Owner
5bf32b6e2393032270a339cbba0b7d2d4937297ef7a59e0f112e8fe27e8748d5

AWS Region
US East (Ohio) us-east-2

Last modified
March 30, 2022, 23:01:13 (UTC+01:00)

Size
3.4 MB

Type
jar

Key
Tournament-1.0-SNAPSHOT.jar

S3 URI
s3://eoinlambdacode/Tournament-1.0-SNAPSHOT.jar

Amazon Resource Name (ARN)
arn:aws:s3:::eoinlambdacode/Tournament-1.0-SNAPSHOT.jar

Entity tag (Etag)
746afd480fb6bddac91696ced94118e5

Object URL
https://eoinlambdacode.s3.us-east-2.amazonaws.com/Tournament-1.0-SNAPSHOT.jar

Cost

1. 5 GB of Standard Storage
2. 20,000 Get Requests
3. 2,000 Put Requests

This is limited to 12 months.

Frontend

The purpose of the frontend is to display a GUI to the user. The frontend takes instructions from the user and sends those instructions as requests to the Middle Layer. These requests include requests for information retrievals and updates to the Middle layer.

React

React is a JavaScript library for building web pages. It allows the developer to create components that can be combined to create complex UIs.

Cost

Free, although there was insufficient to investigate AWS Amplify

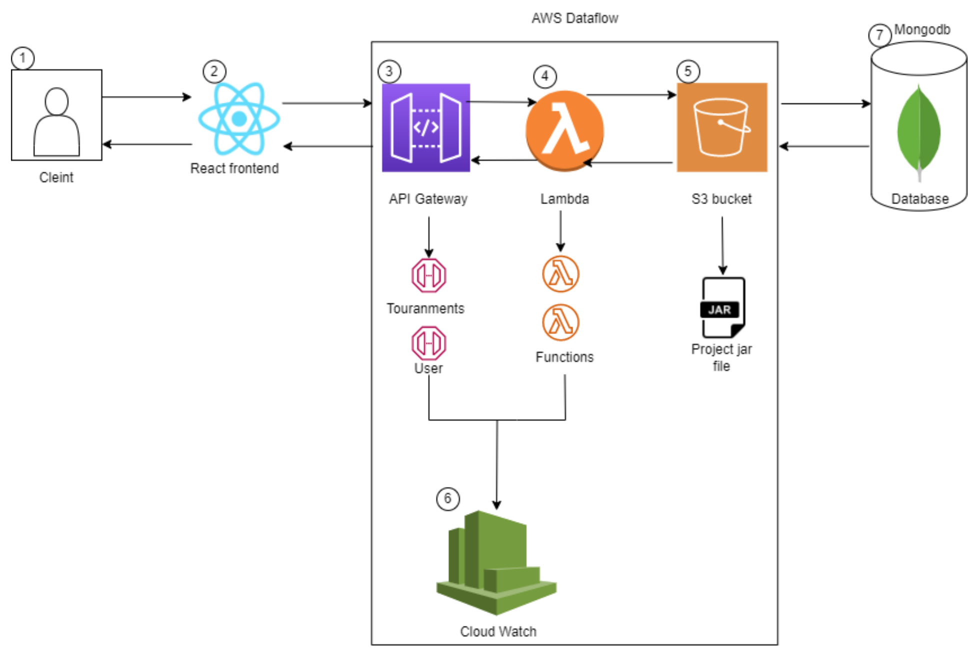
System Design

Overview

This chapter describes technically and functional how the overall is designed and constructed. It details Five points of view

1. Integrations between
    a. Frontend and AWS API Gateway
    b. AWS API Gateway and AWS Lambda
    c. AWS Lambda and Java
    d. Java and MongoDB
2. Configuration of the AWS and MongoDB components
    a. AWS API gateway
    b. AWS Lambda
    c. AWS IAM
3. Java Class Construction
4. React Construction
5. Build process

Highlevel Data flow



The above diagram describes the flow of data/interaction in the system.

The Client browser (1) renders the website with HTML that is created by React frontend (2). The React frontend is served from a local webserver in visual studio. Once rendered on the web browser, data is fetched via HTTP Ajax calls to the REST API that is hosted on the API Gateway (3).

These requests utilize aspects of the HTTP request to fetch/update the appropriate data. The elements include the body, header, http method, path as part of the URL and query parameters. The API gateway has a corresponding definition for each combination of resource and HTTP method. The API gateway forwards the request to the integrated AWS Lambda.

The AWS Lambda java (4 + 5) receives as a byte stream of the above json structure. Each lambda connects to the Mongo DB to query and fetch or update the data. These requests are served over HTTP.

AWS Lambda java processes the return data and sends it back to the frontend via the API Gateway.

Integrations

This section details how the integrations between the compoents

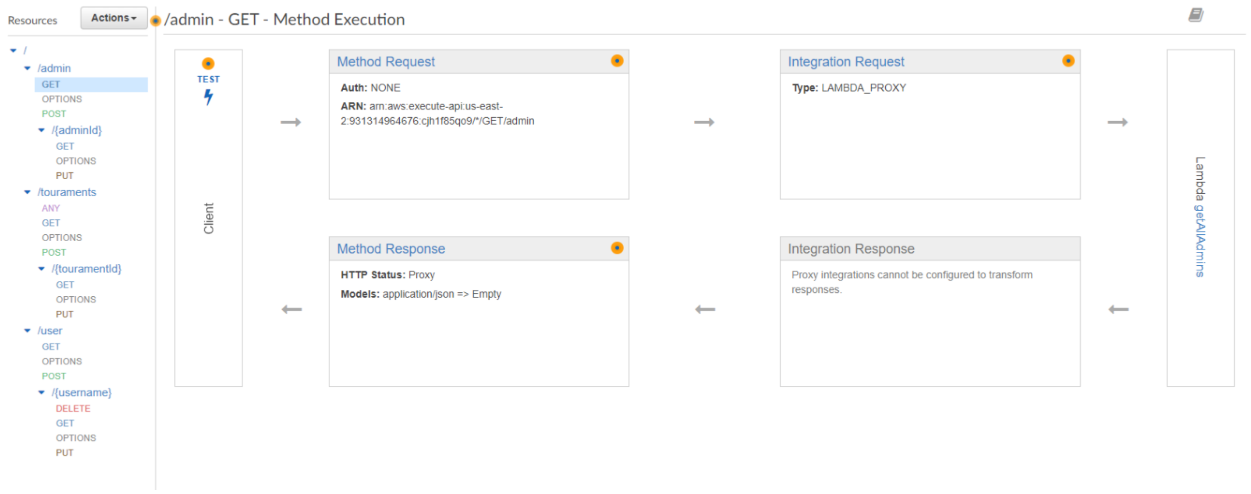
Frontend and AWS API Gateway

This integration is achieved in the react frontend via axios calls

```
axios.get(`https://XXXXXXX.execute-api.us-east-2.amazonaws.com/Develop/touraments`, { params: {
            username: userName,
            inGame: '0',
            dateAt : Date.now()
          }})
```

AWS API Gateway and AWS Lambda

On the API Gateway the is a REST (Resource structure) created for all the HTTP Methods (GET, POST, PUT ,OPTIONS, DELETE) for the Resource admin ,user and tournaments. Each of these resource and method combination has an integration configuration. The following configuration was applied



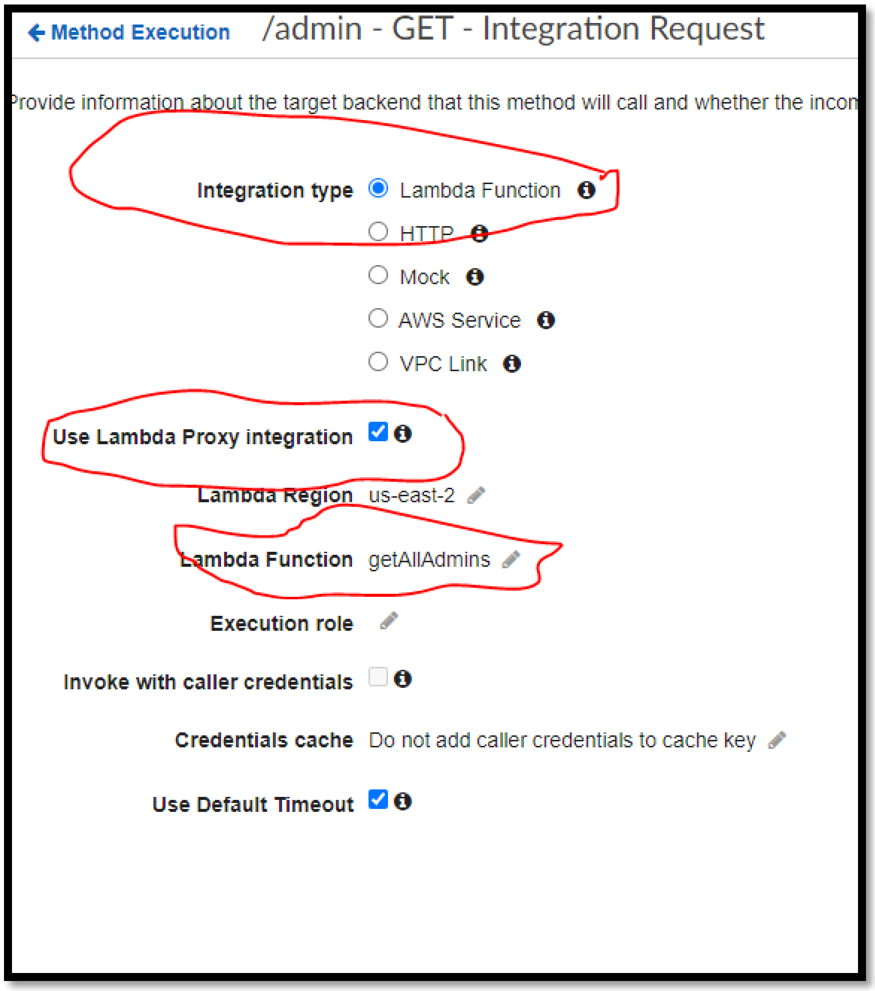
```
Resources   Actions▾      /admin - GET - Method Execution

▾ /
  ▾ /admin
      GET
      OPTIONS
      POST
    ▾ /{adminId}
        GET
        OPTIONS
        PUT
  ▾ /touraments
      ANY
      GET
      OPTIONS
      POST
    ▾ /{touramentId}
        GET
        OPTIONS
        PUT
  ▾ /user
      GET
      OPTIONS
      POST
    ▾ /{username}
        DELETE
        GET
        OPTIONS
        PUT
```

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-2:931314964676:cjh1f85qo9/*/GET/admin

Integration Request
Type: LAMBDA_PROXY

Method Response
HTTP Status: Proxy
Models: application/json => Empty

Integration Response
Proxy integrations cannot be configured to transform responses.



← Method Execution /admin - GET - Integration Request

Provide information about the target backend that this method will call and whether the incon

Integration type: [✓ Lambda Function] HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration ✓

Lambda Region us-east-2

Lambda Function getAllAdmins

Execution role

Invoke with caller credentials ☐

Credentials cache Do not add caller credentials to cache key

Use Default Timeout ✓

The above image shows the admin and GET combinations. Integration Type is set to Lambda Function, Lambda Function is set the Lambda that serves that endpoint. (Appendix A has the Full list of the matching Endpoints to Lambdas). The Use Lambda Proxy integration is set to true. When this is set to the Java/Lambda has to parse the request and produce the full HTTP response.

AWS Lambda and Java

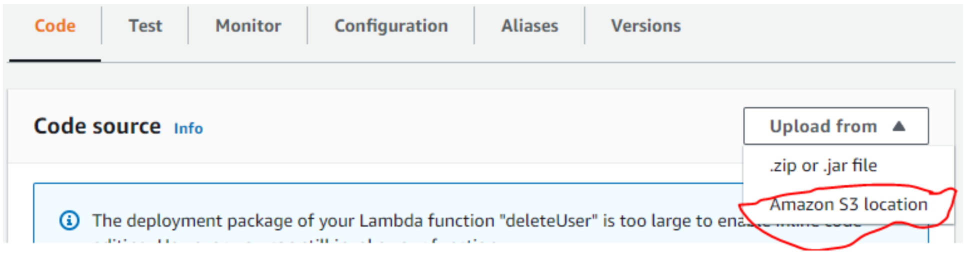
This section details the following

1. Lambda configuration to a Jar and appropriate java method
2. The Java method signature
3. The input format to the Java method
4. The output format required from the Java method

Setting the "Use Lambda Proxy integration" to true affects 2, 3, 4 above. For more details please see [13] https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html

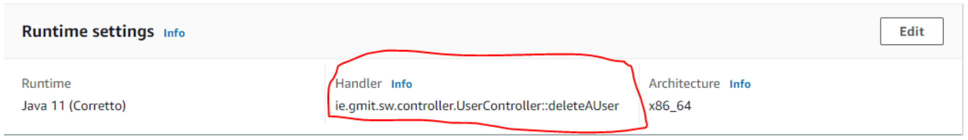
Lambda configuration to a Jar and appropriate java method

When the Lambda is created initially Java 8 is selected. Under the Code Source section, select S3 to enter the S3 AWS Resource Number (ARN)



The deployment package of your Lambda function "deleteUser" is too large to ena

This JAR is the Uber JAR that is created in the Maven Build.



Runtime	Handler	Architecture
Java 11 (Corretto)	ie.gmit.sw.controller.UserController::deleteAUser	x86_64

The Lambda is associated with a Java method in the Jar by setting the handler field in the runtime setting to the *package.Class::method*.

Finally, the Lambda memory configuration can be increased, this will reduce the cold start duration. Should be increased where lambdas are used less frequently

The Java method signature

```
/**
* Gets all admins in the database and tranforms them by putting in a google json
* format
*
* @return
*/
public void getAlAdmin(InputStream input, OutputStream output, Context context) throws IOException
{
```

This is the required signature to communicate the AWS Lambda that has been configured using proxy integration in the API gateway.

The input format to the Java method

When proxy integration is used the following the Json structure

```
{
  "resource": "/my/path",
  "path": "/my/path",
  "httpMethod": "GET",
  "headers": {
    "header1": "value1",
    "header2": "value2"
  },
  "multiValueHeaders": {
    "header1": [
     "value1"
    ],
    "header2": [
     "value1",
     "value2"
    ]
  },
  "queryStringParameters": {
    "parameter1": "value1",
    "parameter2": "value"
  },
  "multiValueQueryStringParameters": {
    "parameter1": [
     "value1",
     "value2"
    ],
    "parameter2": [
     "value"
    ]
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "id",
    "authorizer": {
     "claims": null,
     "scopes": null
    },
    "domainName": "id.execute-api.us-east-1.amazonaws.com",
    "domainPrefix": "id",
    "extendedRequestId": "request-id",
    "httpMethod": "GET",
    "identity": {
     "accessKey": null,
     "accountId": null,
     "caller": null,
     "cognitoAuthenticationProvider": null,
     "cognitoAuthenticationType": null,
     "cognitoIdentityId": null,
     "cognitoIdentityPoolId": null,
     "principalOrgId": null,
     "sourceIp": "IP",
     "user": null,
     "userAgent": "user-agent",
     "userArn": null,
     "clientCert": {
      "clientCertPem": "CERT_CONTENT",
      "subjectDN": "www.example.com",
      "issuerDN": "Example issuer",
        "serialNumber": "a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1:a1",
        "validity": {
          "notBefore": "May 28 12:30:02 2019 GMT",
          "notAfter": "Aug  5 09:36:04 2021 GMT"
        }
      }
    },
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "requestId": "id=",
    "requestTime": "04/Mar/2020:19:15:17 +0000",
    "requestTimeEpoch": 1583349317135,
    "resourceId": null,
    "resourcePath": "/my/path",
    "stage": "$default"
  },
  "pathParameters": null,
  "stageVariables": null,
  "body": "Hello from Lambda!",
  "isBase64Encoded": false
}
```

[14] (https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format)

The key elements are the queryStringParameters and the pathParameters, as these elements determine the filtering of the MongoDB rows. In case where there is request body this located in the body json node. Examples of these values in the project are

```
"queryStringParameters": { "inGame": "0", "userName": "u" }
"body":
"{\"number\":568593,\"userName\":\"game568593\",\"gameMode\":\"gameMode568593\",\"owner\
":\"owner568593\",\"maxPlayers\":12,\"time\":\"Apr 5, 2022, 11:00:00
PM\",\"description\":\"description568593\",\"players\":[\"u\"]}"
```

Note that body contains a json structure as a String, so therefore the quotes inside this String are escaped.

The output format required from the Java method

```
{

  "isBase64Encoded": true|false,

  "statusCode": httpStatusCode,

  "headers": { "headerName": "headerValue", ... },

  "multiValueHeaders": { "headerName": ["headerValue", "headerValue2", ...], ... },

  "body": "..."
```

}

[15]https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-output-format

Note that body contains a json structure as a String, so therefore the quotes inside this String are escaped.

An important note is the return statusCode and headers are created in Java. The headers are important to resolve CORS issues. The Headers Class is used to insert these headers into the response.

```
@SerializedName("Access-Control-Allow-Headers") String accessControlAllowHeader = "Content-Type";
@SerializedName("Access-Control-Allow-Origin") String accessControlAllowOrigin = "*";
@SerializedName("Access-Control-Allow-Methods") String accessControlAllowMethods =
"OPTIONS,POST,GET,PUT,DELETE";
```

The response code is injected in the controllers. The example of success

```
private void successResponseProxy(JSONObject responseJson, JSONObject responseBody) {
responseJson.put("headers", new Headers());
responseJson.put("body", responseBody.toString());
responseJson.put("statusCode", 200);
}
```

The body element is created with escaped quotes, this is achieved by creating a JSON string by using GSON. The DBObject overrides the toString method and is the base Class for all model objects.(Admin, User, Tournament)

```
public abstract class  DBObject {
   /**
    * used to ensure that the body response has a json structure with escaped quotes
    */
   public String toString() {
       Gson gson = new GsonBuilder().setPrettyPrinting().create();
       return gson.toJson(this);
   }
}
```

Java and MongoDB

There are three details here

1. The connection between Java and MongoDB
2. The construction of the Filters
3. The return mapping of the data
4. The update of the data

The connection between Java and MongoDB

The mongodb-driver-core-4.3.0 jar is used to create the connections between Java and MongoDB Serverless. The details were obtained from the MongoDB console. This is contained in the DBConnection Class

```
ConnectionString connectionString = new
ConnectionString("mongodb+srv://user1:password@serverlessinstance0.dgmiv.mongodb.net/Serverle
ssInstance0?retryWrites=true&w=majority&authSource=admin");
```

The construction of the Filters

In certain times there are complex queries that need to be created. e.g. show only tournaments where the logged in user is a player or show only tournaments where the logged in user is a not player

```
if (entry.getKey().equalsIgnoreCase("number")) {
filtereq = eq("number", Integer.valueOf(entry.getValue()));
} else if ( (entry.getKey().equalsIgnoreCase("userName")) && (isATourament) ) {
if (excludePlayer) {
filtereq = ne("players", entry.getValue()) ;
} else {
filtereq = eq("players", entry.getValue());
}
} else {
filtereq = eq(entry.getKey(), entry.getValue());
}
```

These filters are then passed to the MongoDB

The return mapping of the data

Data is returned from the MongoDB in a Document format, The value in this document are mapped to the model values using a mapper class per model (UserMapper is a mapper class for User). An example of a mapping is

```
user1.setEmail(returnData.getString(EMAIL));
```

The update of the data

To update the data the Java fields of the model object (User, Admin and Tournament) are mapped into fields in the MongoDB document.

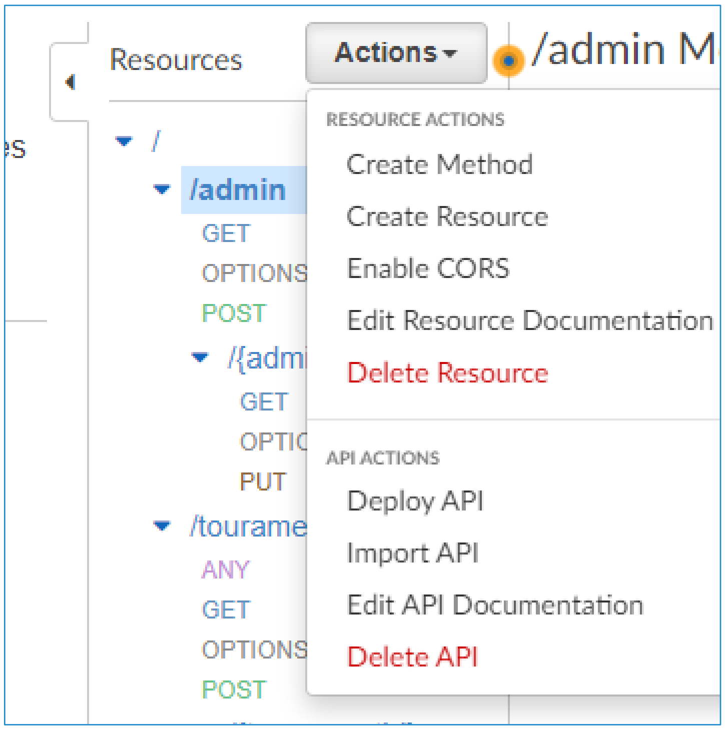
```
Document newDocument;
newDocument = new Document("_id", new ObjectId());
newDocument.append(NUMBER, reqUser.getNumber());
newDocument.append(USERNAME, reqUser.getUserName())
.append(EMAIL, reqUser.getEmail())
.append(FIRST_NAME, reqUser.getFirstName())
.append(PHONE_NUMBER, reqUser.getPhoneNumber())
.append(PASSWORD, reqUser.getPassword())
.append(LAST_NAME, reqUser.getLastName());
```

Configuration of the AWS and MongoDB components

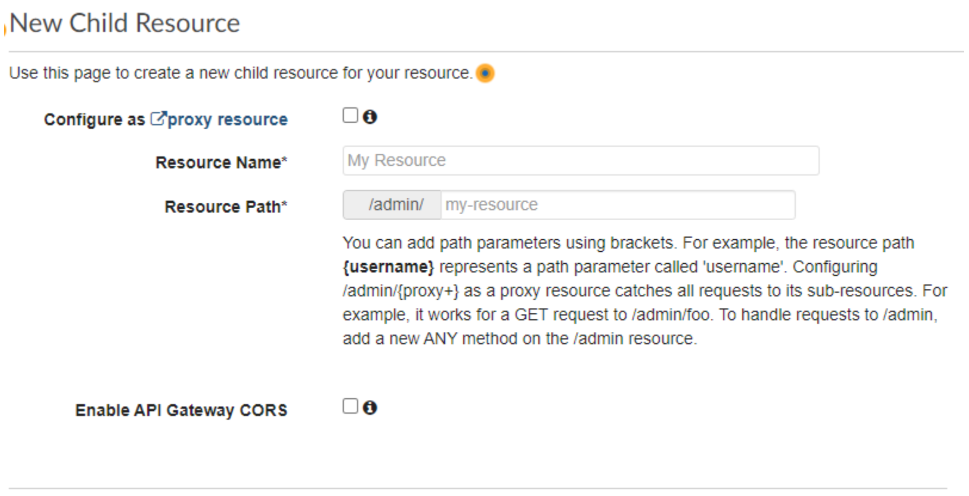
This section details the configuration of AWS

AWS API gateway

This section describes the configuration of an API Gateway resource.   To create a resource the appropriate level is selected, the Actions button is clicked, then the Create Resource is clicked



You are then presented with configuration screen (There is no edit facility after creation, so delete and recreate if mistaken)



New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource	☐
Resource Name*	My Resource
Resource Path*	/admin/ my-resource
	You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring /admin/{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /admin/foo. To handle requests to /admin, add a new ANY method on the /admin resource.
Enable API Gateway CORS	☐

Note adding path parameters is defined here using brackets ({variable}). Note this is done in the project for adminId, touramentId and username.

Particular attention is required here so that names are correct as these defines the path parameters names for the java layer.

The resources defined in the API gateway are as follows

Resource	Method	Description
admin	GET	Gets a list of admins, with no query parameters it will get all admins
admin	POST	Creates a new Admin. It takes an Admin as a body.
admin	OPTIONS	This is required for preflight request to eliminate CORS issue. More details provide later
admin/{adminUsername}	GET	Gets one admin as defined by the adminUsername. Can be called with a query parameter of password is used to login. If a combination of username and password is not found, then 404 is returned
admin/{adminUsername}	PUT	Updates the specified admin as defined by the adminUsername. The new admin is passed in the request body to the API gateway.
admin/{adminUsername}	DELETE	Deletes the specified admin as defined by the adminUsername.
touraments	GET	Gets a list of tournaments, with no query parameters it will get all tournaments. If a query parameters of userName and inGame are passed, then it gets tournaments that either the user has joined or can join. inGame can have a value of 1 for all tournaments that has joined and 0 for all tournaments that they can join.
touraments	POST	The is create a new tournament. It takes a tournament as a body.
touraments	OPTIONS	This is required for preflight request to eliminate CORS issue. More details provide later.
/touraments/{touramentId}	GET	Get one admin as defined by the tournamentId. If not found, then 404 is returned.
/touraments/{touramentId	PUT	Updates the specified tournament as defined by the tournamentId. The new tournament is passed in the request body to the API gateway. This endpoint is used to add or remove players to a tournament. To add a new player the username is added to the players list in the json object. this.state.tourament.players.push(userName)
/touraments/{touramentId}	DELETE	Deletes the specified tournament as defined by the tournamentId.
user	GET	Gets a list of users, with no query parameters it will get all user.
user	POST	Creates a new User. It takes a User as a body.

user	OPTIONS	This is required for preflight request to eliminate CORS issue. More details provide later.
user/{username}	GET	Get one user as defined by the username. Can be called with a query parameter of password is used to login. If a combination of username and password is not found, then 404 is returned.
user/{username}	PUT	Updates the specified user as defined by the username. The new admin is passed in the request body to the API gateway.
user/{username}	DELETE	Deletes the specified user as defined by the username.

AWS Lambda

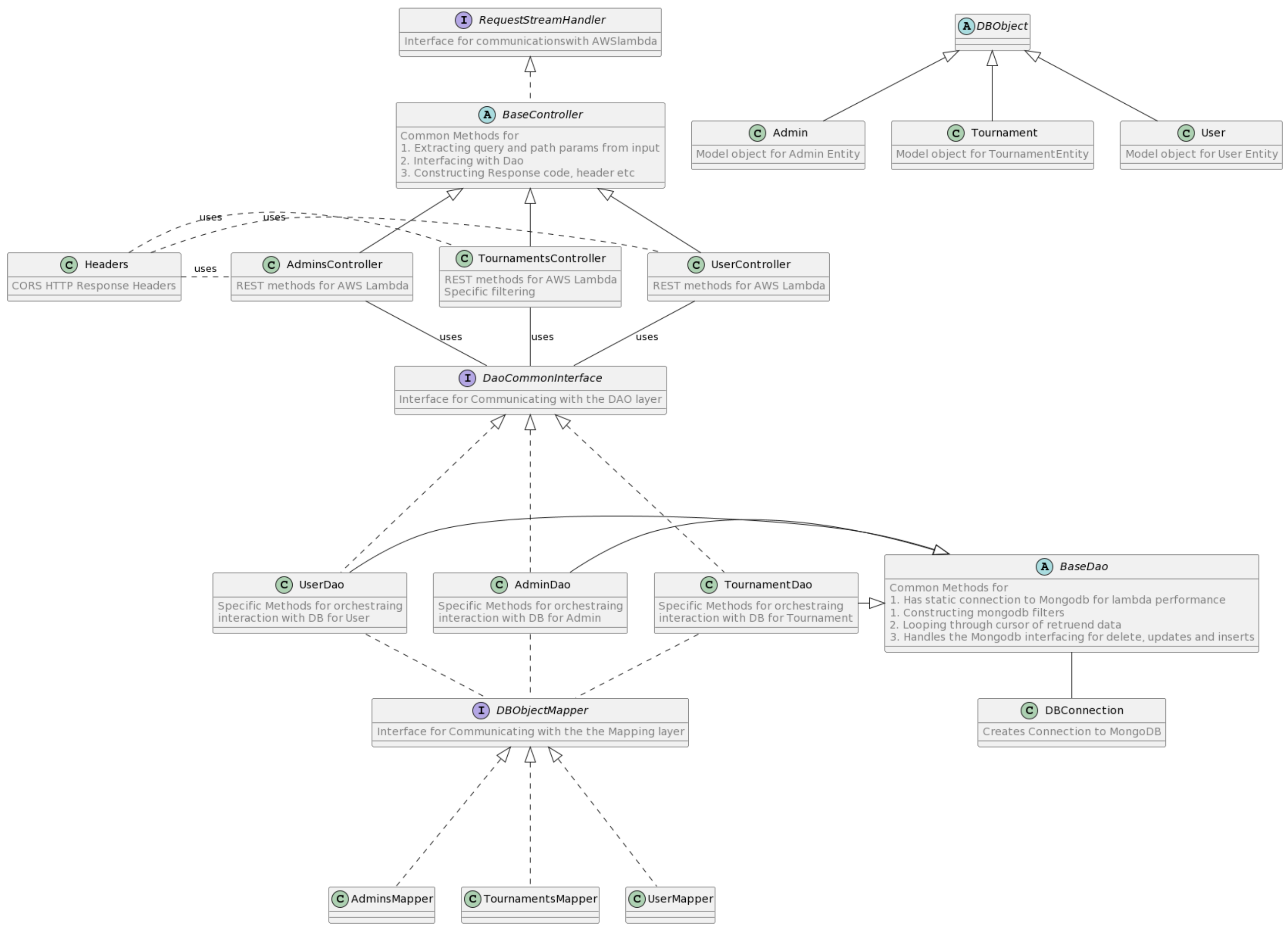
No extra configuration required here in addition to the details described in Integrations...AWS Lambda and Java

AWS IAM

This role was created help with a CORS issue, creating the role and adding to API Gateway allowed for log belong to the API Gateway to be created in AWS CloudWatch. More details here [16](https://aws.amazon.com/premiumsupport/knowledge-center/api-gateway-cloudwatch-logs/)

Java Class Construction

The following diagram describes the structure of the application. The structure consists of controllers Daos, Mappers and Models.

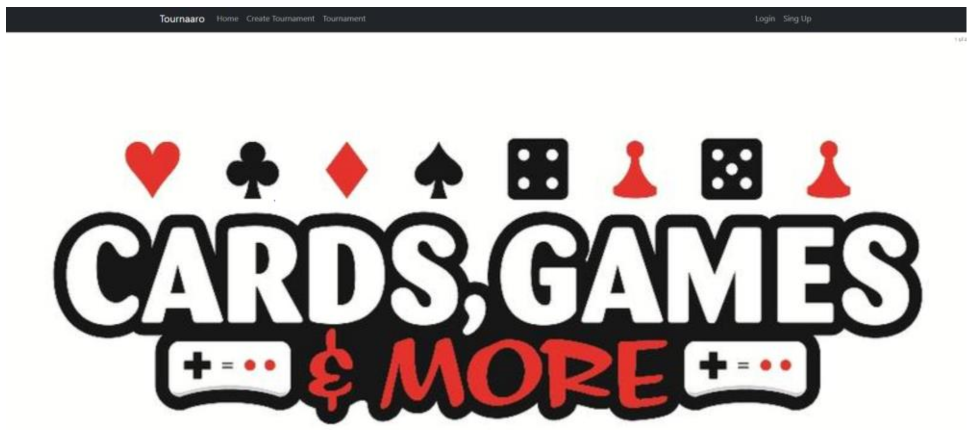

RequestStreamHandler
Interface for communicationswith AWSlambda
BaseController
Common Methods for
1. Extracting query and path params from input
2. Interfacing with Dao
3. Constructing Response code, header etc
DBObject
Admin
Model object for Admin Entity
Tournament
Model object for TournamentEntity
User
Model object for User Entity
uses
uses
Headers
CORS HTTP Response Headers
uses
AdminsController
REST methods for AWS Lambda
TournamentsController
REST methods for AWS Lambda
Specific filtering
UserController
REST methods for AWS Lambda
uses
uses
uses
DaoCommonInterface
Interface for Communicating with the DAO layer
UserDao
Specific Methods for orchestraing interaction with DB for User
AdminDao
Specific Methods for orchestraing interaction with DB for Admin
TournamentDao
Specific Methods for orchestraing interaction with DB for Tournament
BaseDao
Common Methods for
1. Has static connection to Mongodb for lambda performance
1. Constructing mongodb filters
2. Looping through cursor of retruend data
3. Handles the Mongodb interfacing for delete, updates and inserts
DBObjectMapper
Interface for Communicating with the the Mapping layer
DBConnection
Creates Connection to MongoDB
AdminsMapper
TournamentsMapper
UserMapper

The controllers are the interface to the Lambda. The DAOs are the Data Access Objects that connect to the DB. The model represents the main entities in the system. The mapper are used to transform data to and from MongoDB documents.

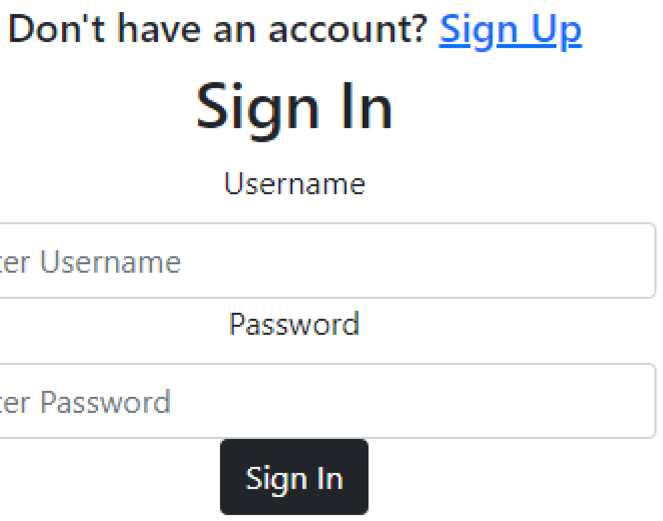
Fronted - React Construction

The Frontend consists of several pages to allow user/player to do the following

- Home Screen: to be display when no user is login , Limited features are shown (login)



- Login: Allows the player to supply a username and password. Once submitted the username and password will be validated via the backend. If successful, the tournaments that the player is already registered, and the tournaments that they are not registered in. Only tournaments that are in the future are displayed. These tournaments filters are passed to the API Gateway and the filtering is done in the backend.


Don't have an account? Sign Up

Sign In

Username

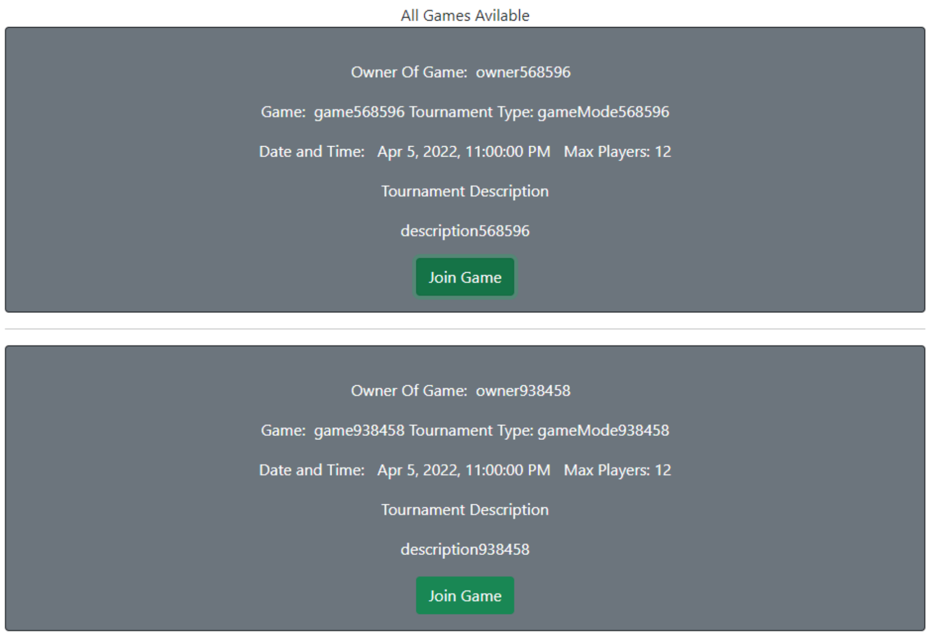
Enter Username

Password

Enter Password

Sign In

- Join a tournament: When viewing the list of tournaments that player is not registered in, they are given the option to join that tournament. When clicked a post called is passed to the API gateway to update the tournament with the player added.


All Games Avilable

Owner Of Game: owner568596

Game: game568596 Tournament Type: gameMode568596

Date and Time: Apr 5, 2022, 11:00:00 PM Max Players: 12

Tournament Description

description568596

Join Game

Owner Of Game: owner938458

Game: game938458 Tournament Type: gameMode938458

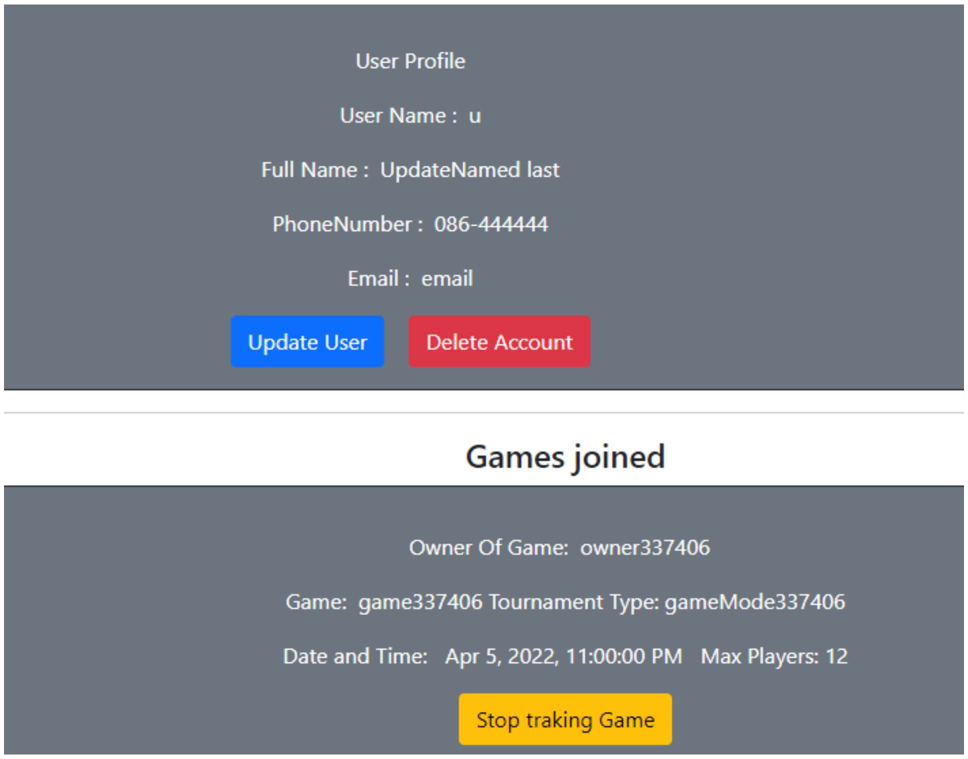
Date and Time: Apr 5, 2022, 11:00:00 PM Max Players: 12

Tournament Description

description938458

Join Game

- Profile and Leave a tournament: when clicking of the User the user profile is displayed along the list of tournaments that player has joined, they are given the option to leave that tournament. When clicked a post called is passed to the API gateway to update the tournament with the player removed.



User Profile

User Name : u

Full Name : UpdateNamed last

PhoneNumber : 086-444444

Email : email

Games joined

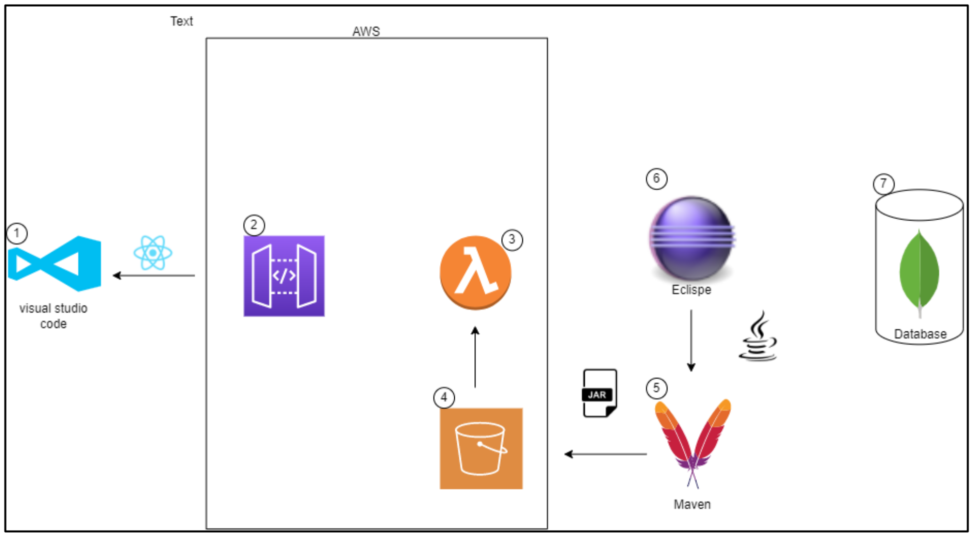
Owner Of Game: owner337406

Game: game337406 Tournament Type: gameMode337406

Date and Time: Apr 5, 2022, 11:00:00 PM Max Players: 12

- Logout: An option for the player to logout
- Delete: A page to delete the player's account. This will pass a Delete to the API Gateway.

Build process



This section describes how each component was built and/or deployed. The detail of the configuration is described above

- MongoDB: This was manually configured using MongoDB console. No deployment

- Lambda: Eclipse was used for developing the java, Maven (install) created a Fat jar. The jar was uploaded to an S3 bucket. Each lambda was connected to the S3 using the ARN (AWS Resource Number). The Lambdas are configured using the AWS Console.
- The AWS API Gateway is configured using the AWS Console, and is deployed using the Action Deploy API in the API Gateway console
- The react Front is started using NPM start inside Visual Code.

AWS Lambda

Each API gateway as defined by in The Component section above will have a corresponding AWS Lambda that executes the appropriate Method in the JAR. There is no AWS lambda for the OPTIONS methods.

Limitations And Known Bugs

This chapter outlines the issues that were resolved as the issues that remain open. It also shows how Cloudwatch logs were used in the debugging process of issues.

Closed Issues

The following issues were some of the issues encountered and fixed while developing the application

1. CORS – Preflight and Methods (non-proxy integrations), Fixed by enabling CORS on API Gateway
2. Timeout after 15 seconds to MongoDB. Fixed by changing libraries and connection details
3. MongoDB Puts Not Updating - Puts will push extra information up however will not update older information: Fixed Wrong attribute used in identifying ROW to updated
4. GetOne for both Tournaments and Admins - Seems to search for the Path parameters twice Fixed: Send down Correct parameters and cleanup API Gateway definition.
5. Cors issue with API gateway-Fixed When the API Gateway used proxy integration the response headers to disables cors had to be written in the java lambdas. These headers were added "Access-Control-Allow-Headers", "Access-Control-Allow-Origin" and "Access-Control-Allow-Methods"
6. Cors issue with post- Post sent to back end returns a CORS issue. Fixed frontend calling wrong endpoint.
7. Tournament Page-Filtering and Printing/Rendering all tournaments: Fixed introduced additional Filtering to the application.
8. Slowness in deploying Jar, leading to out of sync AWS Lambdas with the Java code : fixed introduced uploading to S3. The Jar was uploaded to a S3 bucket which speeded up the process of deploying.
9. Formatting the response body in a AWS API Gateway to AWS Lambda proxy integration with respect to the handling of Quotes: Fixed by creating a JSON string of the response body prior to inserting into the response map.

Open Issues

The following are the remaining open issues

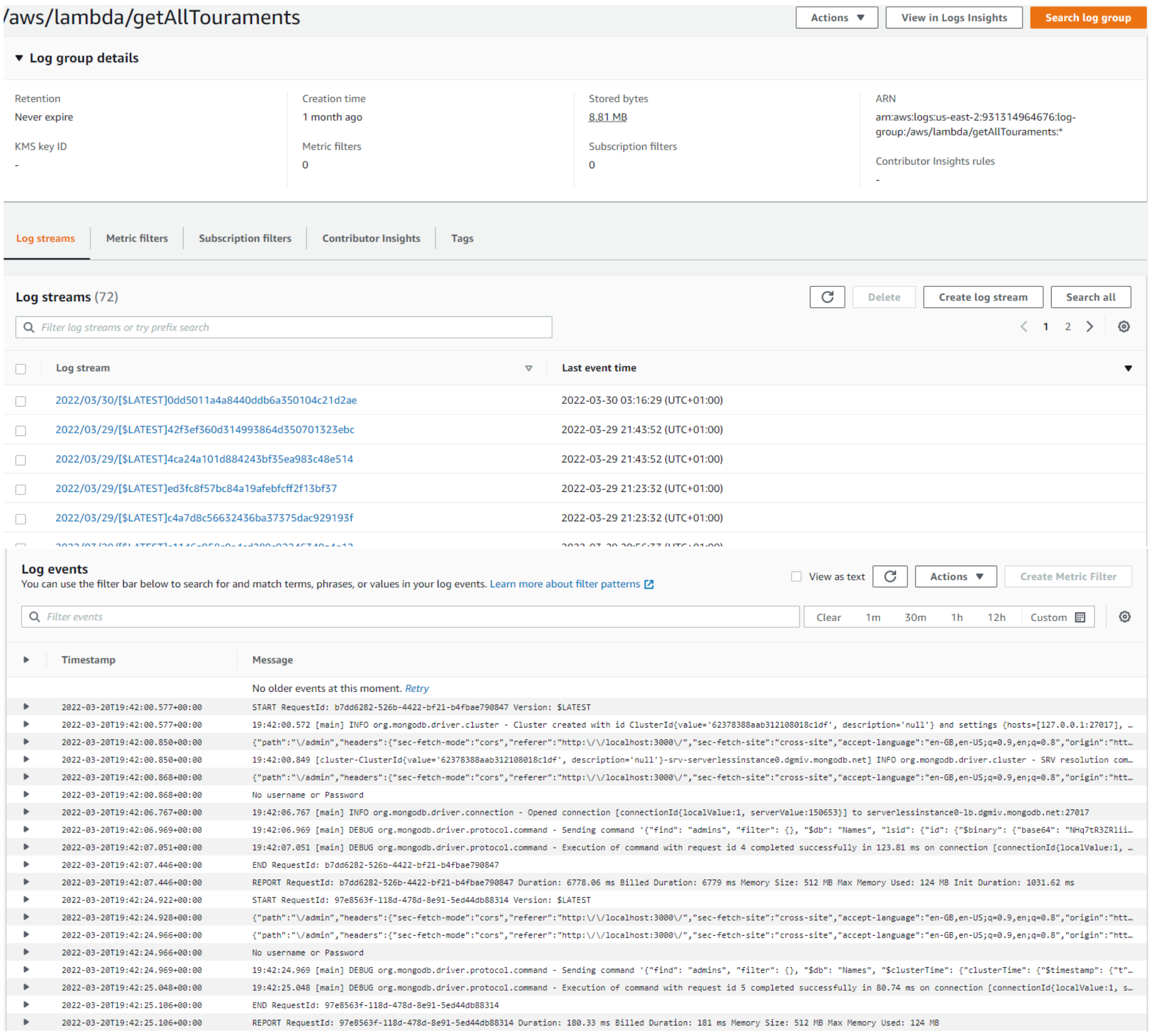
1. Reload Data not returning updated rows: When the user joins a tournament, the react fire the reload event on the parent component. The calls the tournament endpoint again which should remove the specific row from the list before rendering the page again. The returned response data still contains the rows, although a refresh of the page will not show the tournament
2. Build using jdk8- Java 8 end of life was 31-MAr-2022 [17](https://endoflife.date/java). Initially at the start of the project there was an issue with build in java 11. After some time, it was decided to build it on java 8 due to all examples found online used Java 8 and it cause no errors when uploading to the AWS lambdas. After resolution of many issues, it should be timely to revisit this issue.

3. MongoDB Database-The naming convention in MongoDB Database is not ideal. The database is called "Names" and the collections are called {Games, admins, users} a better set of names would be all lowercase and {tournaments, admins, users}

4. Proper Data- Need proper data to help the demo and functionality

CloudWatch Logs

The debugging process of the issues is difficult, as there is a reliance on the AWS CloudWatch logs. The starting and finishing events are logged in the appropriate logs. Although API gateway logs are not set on by default. The configuration of setting those logs are detailed above. The following image shows the level of details in the logs. Each entry can be expanded to show more details.



Retention	Creation time	Stored bytes	ARN
Never expire	1 month ago	8.81 MB	arn:aws:logs:us-east-2:931314964676:log-group:/aws/lambda/getAllTouraments:*
KMS key ID	Metric filters	Subscription filters	Contributor Insights rules
-	0	0	-

Log stream	Last event time
2022/03/30/[$LATEST]0dd5011a4a8440ddb6a350104c21d2ae	2022-03-30 03:16:29 (UTC+01:00)
2022/03/29/[$LATEST]42f3ef360d314993864d350701323ebc	2022-03-29 21:43:52 (UTC+01:00)
2022/03/29/[$LATEST]4ca24a101d884243bf35ea983c48e514	2022-03-29 21:43:52 (UTC+01:00)
2022/03/29/[$LATEST]ed3fc8f57bc84a19afebfcff2f13bf37	2022-03-29 21:23:32 (UTC+01:00)
2022/03/29/[$LATEST]c4a7d8c56632436ba37375dac929193f	2022-03-29 21:23:32 (UTC+01:00)

Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                       Message
                                No older events at this moment. Retry
2022-03-20T19:42:00.577+00:00   START RequestId: b7dd6282-526b-4422-bf21-b4fbae790847 Version: $LATEST
2022-03-20T19:42:00.577+00:00   19:42:00.572 [main] INFO org.mongodb.driver.cluster - Cluster created with id ClusterId{value='62378388aab312108018c1df', description='null'} and settings {hosts=[127.0.0.1:27017], …
2022-03-20T19:42:00.850+00:00   {"path":"\/admin","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"htt…
2022-03-20T19:42:00.850+00:00   19:42:00.849 [cluster-ClusterId{value='62378388aab312108018c1df', description='null'}-srv-serverlessinstance0.dgmiv.mongodb.net] INFO org.mongodb.driver.cluster - SRV resolution com…
2022-03-20T19:42:00.868+00:00   {"path":"\/admin","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"htt…
2022-03-20T19:42:00.868+00:00   No username or Password
2022-03-20T19:42:06.767+00:00   19:42:06.767 [main] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:1, serverValue:150653}] to serverlessinstance0-lb.dgmiv.mongodb.net:27017
2022-03-20T19:42:06.969+00:00   19:42:06.969 [main] DEBUG org.mongodb.driver.protocol.command - Sending command '{"find": "admins", "filter": {}, "$db": "Names", "lsid": {"id": {"$binary": {"base64": "NHq7tR3ZRlii…
2022-03-20T19:42:07.051+00:00   19:42:07.051 [main] DEBUG org.mongodb.driver.protocol.command - Execution of command with request id 4 completed successfully in 123.81 ms on connection [connectionId{localValue:1, …
2022-03-20T19:42:07.446+00:00   END RequestId: b7dd6282-526b-4422-bf21-b4fbae790847
2022-03-20T19:42:07.446+00:00   REPORT RequestId: b7dd6282-526b-4422-bf21-b4fbae790847 Duration: 6778.06 ms Billed Duration: 6779 ms Memory Size: 512 MB Max Memory Used: 124 MB Init Duration: 1031.62 ms
2022-03-20T19:42:24.922+00:00   START RequestId: 97e8563f-118d-478d-8e91-5ed44db88314 Version: $LATEST
2022-03-20T19:42:24.928+00:00   {"path":"\/admin","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"htt…
2022-03-20T19:42:24.966+00:00   {"path":"\/admin","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"htt…
2022-03-20T19:42:24.966+00:00   No username or Password
2022-03-20T19:42:24.969+00:00   19:42:24.969 [main] DEBUG org.mongodb.driver.protocol.command - Sending command '{"find": "admins", "filter": {}, "$db": "Names", "$clusterTime": {"clusterTime": {"$timestamp": {"t"…
2022-03-20T19:42:25.048+00:00   19:42:25.048 [main] DEBUG org.mongodb.driver.protocol.command - Execution of command with request id 5 completed successfully in 80.74 ms on connection [connectionId{localValue:1, s…
2022-03-20T19:42:25.106+00:00   END RequestId: 97e8563f-118d-478d-8e91-5ed44db88314
2022-03-20T19:42:25.106+00:00   REPORT RequestId: 97e8563f-118d-478d-8e91-5ed44db88314 Duration: 180.33 ms Billed Duration: 181 ms Memory Size: 512 MB Max Memory Used: 124 MB
```

The following log expansion show the issue with tournament PUT method not updating the row. There was two filters been sent to the MongoDB (one for "number": 337407 and one for "touramentId": "337407"). Only the filter "number": 337407 was required.

```
12:40:01.223 [main] DEBUG org.mongodb.driver.protocol.command - Sending command '{
    "update": "Games",
    "ordered": true,
    "writeConcern": {
      "w": "majority"
    },
    "txnNumber": 1,
    "$db": "Names",
    "lsid": {
      "id": {
        "$binary": {
          "base64": "73bGmA6AQ/SA0mnHk8zD1w==",
          "subType": "04"
        }
      }
    },
    "apiVersion": "1",
    "updates": [
      {
        "q": {
          "$and": [
            {
              "number": 337407
            },
            {
              "touramentId": "337407"
            }
          ]
        },
        "u": {
          "$set": {
            "owner": "owner337407",
            "gameType": null,
            "gameMode": "gameMode337407",
            "description": "description337407",
            "maxPlayer": 12,
            "players": [
              "gkj",
              "u"
            ],
            "time": {
              "$date": "2022-04-05T23:00:00Z"
            },
            "duration": null,
            "public": false,
            "numRounds": null
          }
        }
      }
    ]
}' with request id 4 to database Names on connection [connectionId{localValue:1, serverValue:44535}]
to server serverlessinstance0-lb.dgmiv.mongodb.net:27017
```

System Evalution/Conclusion

Overview

This section summarizes the observations and the follow on work that could be performed to provide further insights.

Observations

Cold starts

"Cold starts can be defined as the set-up time required to get a serverless application's environment up and running when it is invoked for the first time within a defined period. Cold starts are somewhat of an inherent problem with the serverless model." [18](https://builtin.com/software-engineering-perspectives/cold-starts-challenge-serverless-architecture).

These are more obvious in Java due to the starting of the JVM. In AWS the Lambda stay warm for approximately 15 minutes. In the project the cold starts took approximately 10 seconds initially.

It was recommend to keep the Jar as small as possible and this was facilitated by the examples by not using Spring framework.

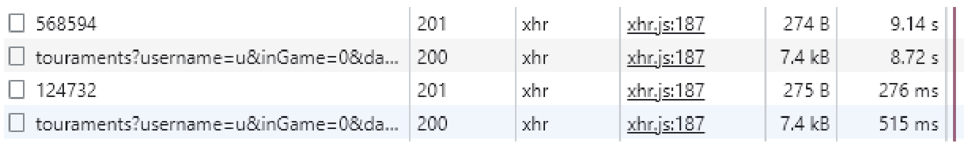
One experiment recommended by

- Captial one[19](https://www.capitalone.com/tech/cloud/aws-lambda-java-tutorial-reduce-cold-starts/)
- AWS [20](https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html#configuration-memory-console)

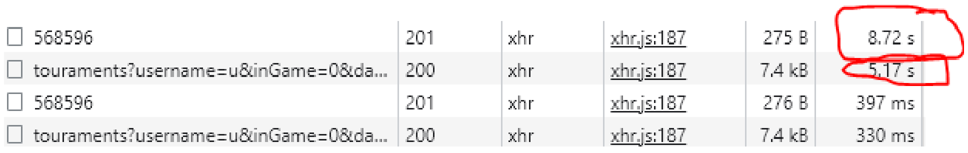
"Lambda allocates CPU power in proportion to the amount of memory configured. Memory is the amount of memory available to your Lambda function at runtime. You can increase or decrease the memory and CPU power allocated to your function using the Memory (MB) setting. To configure the memory for your function, set a value between 128 MB and 10,240 MB in 1-MB increments. At 1,769 MB, a function has the equivalent of one vCPU (one vCPU-second of credits per second)"

Memory

When memory was double on the lambda from 512 Kb to 1024 Kb. There was a decrease from 8.72 seconds to 5.17 seconds for the tournament GET lambda.



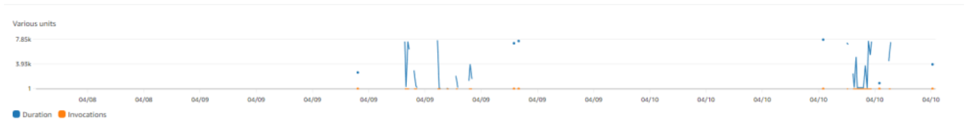
☐ 568594	201	xhr	xhr.js:187	274 B	9.14 s
☐ touraments?username=u&inGame=0&da...	200	xhr	xhr.js:187	7.4 kB	8.72 s
☐ 124732	201	xhr	xhr.js:187	275 B	276 ms
☐ touraments?username=u&inGame=0&da...	200	xhr	xhr.js:187	7.4 kB	515 ms



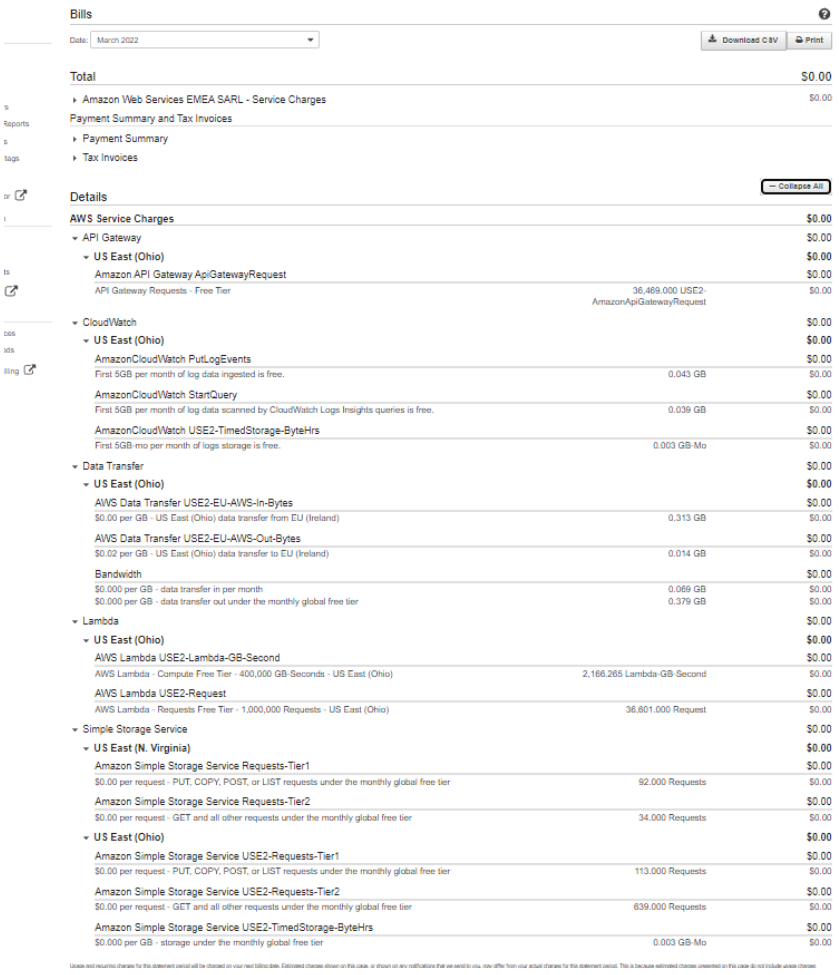
☐ 568596	201	xhr	xhr.js:187	275 B	8.72 s
☐ touraments?username=u&inGame=0&da...	200	xhr	xhr.js:187	7.4 kB	5.17 s
☐ 568596	201	xhr	xhr.js:187	276 B	397 ms
☐ touraments?username=u&inGame=0&da...	200	xhr	xhr.js:187	7.4 kB	330 ms

Costs for Downtime

The following diagram shows how much time when there was no traffic from development. The REST API, Lambda and the database server were always live (always available). The chart shows the time when there were no executions of any Lambda. This is the time when there was no charge.



If this was a traditional server, there much a continuous charge for this downtime. In addition there was no cost for any of the other services (S3, API Gateway, CloudWatch).



Total		$0.00
Amazon Web Services EMEA SARL - Service Charges		$0.00
Payment Summary and Tax Invoices		
Payment Summary		
Tax Invoices		
Details		
AWS Service Charges		**$0.00**
API Gateway		$0.00
US East (Ohio)		**$0.00**
Amazon API Gateway ApiGatewayRequest		$0.00
API Gateway Requests - Free Tier	36,469.000 USE2-AmazonApiGatewayRequest	$0.00
CloudWatch		$0.00
US East (Ohio)		**$0.00**
AmazonCloudWatch PutLogEvents		$0.00
First 5GB per month of log data ingested is free.	0.043 GB	$0.00
AmazonCloudWatch StartQuery		$0.00
First 5GB per month of log data scanned by CloudWatch Logs Insights queries is free.	0.039 GB	$0.00
AmazonCloudWatch USE2-TimedStorage-ByteHrs		$0.00
First 5GB-mo per month of logs storage is free.	0.003 GB-Mo	$0.00
Data Transfer		$0.00
US East (Ohio)		**$0.00**
AWS Data Transfer USE2-EU-AWS-In-Bytes		$0.00
$0.00 per GB - US East (Ohio) data transfer from EU (Ireland)	0.313 GB	$0.00
AWS Data Transfer USE2-EU-AWS-Out-Bytes		$0.00
$0.02 per GB - US East (Ohio) data transfer to EU (Ireland)	0.014 GB	$0.00
Bandwidth		$0.00
$0.000 per GB - data transfer in per month	0.069 GB	$0.00
$0.000 per GB - data transfer out under the monthly global free tier	0.379 GB	$0.00
Lambda		$0.00
US East (Ohio)		**$0.00**
AWS Lambda USE2-Lambda-GB-Second		$0.00
AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - US East (Ohio)	2,166.265 Lambda-GB-Second	$0.00
AWS Lambda USE2-Request		$0.00
AWS Lambda - Requests Free Tier - 1,000,000 Requests - US East (Ohio)	36,601.000 Request	$0.00
Simple Storage Service		$0.00
US East (N. Virginia)		**$0.00**
Amazon Simple Storage Service Requests-Tier1		$0.00
$0.00 per request - PUT, COPY, POST, or LIST requests under the monthly global free tier	92.000 Requests	$0.00
Amazon Simple Storage Service Requests-Tier2		$0.00
$0.00 per request - GET and all other requests under the monthly global free tier	34.000 Requests	$0.00
US East (Ohio)		**$0.00**
Amazon Simple Storage Service USE2-Requests-Tier1		$0.00
$0.00 per request - PUT, COPY, POST, or LIST requests under the monthly global free tier	113.000 Requests	$0.00
Amazon Simple Storage Service USE2-Requests-Tier2		$0.00
$0.00 per request - GET and all other requests under the monthly global free tier	639.000 Requests	$0.00
Amazon Simple Storage Service USE2-TimedStorage-ByteHrs		$0.00
$0.000 per GB - storage under the monthly global free tier	0.003 GB-Mo	$0.00

Usage and recurring charges for this statement period will be charged on your next billing date. Estimated charges shown on this page, or shown on any notifications that we send to you, may differ from your actual charges for this statement period. This is because estimated charges presented on this page do not include usage charges

Increased Compexity

As detailed above there are several different components to be configured, each of those have particular details in the configuration in order to work with another component. This requires a significant amount of research. Although the significant part of the application was AWS, this would be more difficult if components were picked from different providers.

Follow on work

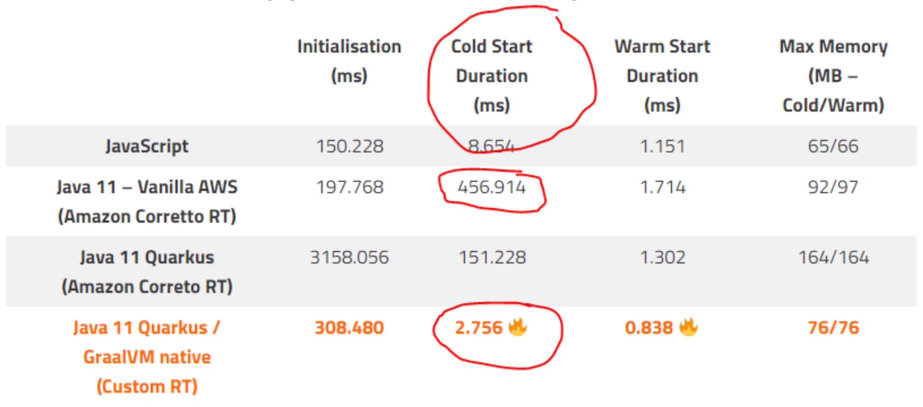
Warming the functions

It is possible when the website initially loads to call each of the REST end points with dummy data. This will have the effect of preloading all of the lambda functions.

GraalVM

“GraalVM is a high-performance JDK distribution designed to accelerate the execution of applications written in Java and other JVM languages along with support for JavaScript, Ruby, Python, and a number of other popular languages” [21](https://www.graalvm.org/22.0/docs/introduction/).

In the article [22](https://shinesolutions.com/2021/08/30/improving-cold-start-times-of-java-aws-lambda-functions-using-graalvm-and-native-images/)



	Initialisation (ms)	Cold Start Duration (ms)	Warm Start Duration (ms)	Max Memory (MB – Cold/Warm)
JavaScript	150.228	8.654	1.151	65/66
Java 11 – Vanilla AWS (Amazon Corretto RT)	197.768	456.914	1.714	92/97
Java 11 Quarkus (Amazon Correto RT)	3158.056	151.228	1.302	164/164
Java 11 Quarkus / GraalVM native (Custom RT)	**308.480**	**2.756** 🔥	**0.838** 🔥	**76/76**

GraalVM reduced the Cold start from 456 milliseconds to 2.7 milliseconds.

AWS Amplify

Investigate and deploy AWS Amplify to go serverless for the React Client on web.

Cut Content

Due to time constraints, some functionality of the website was omitted, such as the administration functionality. The backend services for this functionality were completed. Although not completed it did not hamper the project study. The study was about the feasibility of serverless. Other functionality such as profile pictures was omitted.

Some backend functionality should as

1. Validating the number of players in tournament was not greater than the max.
2. Only showing tournaments in the future

Appendix A REST to Controller Method Mapping

Resource	Method	Lambda	Method
admin	GET	getAllAdmins	
admin	POST	postAdmin	
admin/{adminUsername}	GET	getAdmin	
admin/{adminUsername}	PUT	putAdmin	
admin/{adminUsername}	DELETE		
touraments	GET		
touraments	POST	postTourament	
/touraments/{touramentId}	GET	getTourament	
/touraments/{touramentId	PUT	putTourament	
/touraments/{touramentId}	DELETE		
user	GET	getAllUser	
user	POST	postUser	
user/{username}	GET	getUser	
user/{username}	PUT	putUsers	
user/{username}	DELETE	deleteUser	

Appendix B CLoudWatch logs

2022-04-09T20:00:50.059+01:00	START RequestId: 4cd350b8-11d4-42ab-8db7-862ae8b74530 Version: $LATEST
2022-04-09T20:00:50.059+01:00	19:00:50.055 [main] INFO org.mongodb.driver.cluster - Cluster created with id ClusterId{value='6251d7e23c2fb136093827c5', description='null'} and settings {hosts=[127.0.0.1:27017], srvHost=serverlessinstance0.dgmiv.mongodb.net, mode=LOAD_BALANCED, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms'}
2022-04-09T20:00:50.172+01:00	event {"path":"\/touraments","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"http:\/\/localhost:3000","User-Agent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","X-Forwarded-Proto":"https","Host":"cjh1f85qo9.execute-api.us-east-2.amazonaws.com","dnt":"1","X-Forwarded-Port":"443","X-Amzn-Trace-Id":"Root=1-6251d7e0-7f1138d16c5b450b4607b618","accept":"application\/json, text\/plain, *\/*","sec-ch-ua":"\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\"","sec-ch-ua-mobile":"?0","sec-ch-ua-platform":"\"Windows\"","X-Forwarded-For":"185.34.122.187","accept-encoding":"gzip, deflate, br","sec-fetch-dest":"empty"},"pathParameters":null,"isBase64Encoded":false,"multiValueQueryStringParameters":{"inGame":["1"],"userName":["u"]},"multiValueHeaders":{"sec-fetch-mode":["cors"],"referer":["http:\/\/localhost:3000\/"],"sec-fetch-site":["cross-site"],"accept-

	language":["en-GB,en-US;q=0.9,en;q=0.8"],"origin":["http:\/\/localhost:3000"],"User-Agent":["Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36"],"X-Forwarded-Proto":["https"],"Host":["cjh1f85qo9.execute-api.us-east-2.amazonaws.com"],"dnt":["1"],"X-Forwarded-Port":["443"],"X-Amzn-Trace-Id":["Root=1-6251d7e0-7f1138d16c5b450b4607b618"],"accept":["application\/json, text\/plain, *\/*"],"sec-ch-ua":["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\""],"sec-ch-ua-mobile":["?0"],"sec-ch-ua-platform":["\"Windows\""],"X-Forwarded-For":["185.34.122.187"],"accept-encoding":["gzip, deflate, br"],"sec-fetch-dest":["empty"]},"requestContext":{"resourceId":"iq3rce","resourcePath":"\/touraments","httpMethod":"GET","extendedRequestId":"QU6rKEwkCYcFjeg=","requestTime":"09\/Apr\/2022:19:00:48 +0000","path":"\/Develop\/touraments","accountId":"931314964676","protocol":"HTTP\/1.1","stage":"Develop","domainPrefix":"cjh1f85qo9","requestTimeEpoch":1649530848856,"requestId":"69f0c156-bb06-4562-949e-6633d59bbe82","identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"sourceIp":"185.34.122.187","principalOrgId":null,"accessKey":null,"cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,"userAgent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","user":null},"domainName":"cjh1f85qo9.execute-api.us-east-2.amazonaws.com","apiId":"cjh1f85qo9"},"resource":"\/touraments","httpMethod":"GET","queryStringParameters":{"inGame":"1","userName":"u"},"stageVariables":null,"body":null}
2022-04-09T20:00:50.174+01:00	{"path":"\/touraments","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"http:\/\/localhost:3000","User-Agent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","X-Forwarded-Proto":"https","Host":"cjh1f85qo9.execute-api.us-east-2.amazonaws.com","dnt":"1","X-Forwarded-Port":"443","X-Amzn-Trace-Id":"Root=1-6251d7e0-7f1138d16c5b450b4607b618","accept":"application\/json, text\/plain, *\/*","sec-ch-ua":"\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\"","sec-ch-ua-mobile":"?0","sec-ch-ua-platform":"\"Windows\"","X-Forwarded-For":"185.34.122.187","accept-encoding":"gzip, deflate, br","sec-fetch-dest":"empty"},"pathParameters":null,"isBase64Encoded":false,"multiValueQueryStringParameters":{"inGame":["1"],"userName":["u"]},"multiValueHeaders":{"sec-fetch-mode":["cors"],"referer":["http:\/\/localhost:3000\/"],"sec-fetch-site":["cross-site"],"accept-language":["en-GB,en-US;q=0.9,en;q=0.8"],"origin":["http:\/\/localhost:3000"],"User-Agent":["Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36"],"X-Forwarded-Proto":["https"],"Host":["cjh1f85qo9.execute-api.us-east-2.amazonaws.com"],"dnt":["1"],"X-Forwarded-Port":["443"],"X-Amzn-Trace-Id":["Root=1-6251d7e0-7f1138d16c5b450b4607b618"],"accept":["application\/json, text\/plain, *\/*"],"sec-ch-ua":["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\""],"sec-ch-ua-mobile":["?0"],"sec-ch-ua-platform":["\"Windows\""],"X-Forwarded-For":["185.34.122.187"],"accept-encoding":["gzip, deflate, br"],"sec-fetch-dest":["empty"]},"requestContext":{"resourceId":"iq3rce","resourcePath":"\/touraments","httpMethod":"GET","extendedRequestId":"QU6rKEwkCYcFjeg=","requestTime":"09\/Apr\/202

	2:19:00:48 +0000","path":"\/Develop\/touraments","accountId":"931314964676","protocol":"HTTP\/1.1","stage":"Develop","domainPrefix":"cjh1f85qo9","requestTimeEpoch":1649530848856,"requestId":"69f0c156-bb06-4562-949e-6633d59bbe82","identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"sourceIp":"185.34.122.187","principalOrgId":null,"accessKey":null,"cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,"userAgent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","user":null},"domainName":"cjh1f85qo9.execute-api.us-east-2.amazonaws.com","apiId":"cjh1f85qo9"},"resource":"\/touraments","httpMethod":"GET","queryStringParameters":{"inGame":"1","userName":"u"},"stageVariables":null,"body":null}
2022-04-09T20:00:50.233+01:00	{"path":"\/touraments","headers":{"sec-fetch-mode":"cors","referer":"http:\/\/localhost:3000\/","sec-fetch-site":"cross-site","accept-language":"en-GB,en-US;q=0.9,en;q=0.8","origin":"http:\/\/localhost:3000","User-Agent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","X-Forwarded-Proto":"https","Host":"cjh1f85qo9.execute-api.us-east-2.amazonaws.com","dnt":"1","X-Forwarded-Port":"443","X-Amzn-Trace-Id":"Root=1-6251d7e0-7f1138d16c5b450b4607b618","accept":"application\/json, text\/plain, *\/*","sec-ch-ua":"\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\"","sec-ch-ua-mobile":"?0","sec-ch-ua-platform":"\"Windows\"","X-Forwarded-For":"185.34.122.187","accept-encoding":"gzip, deflate, br","sec-fetch-dest":"empty"},"pathParameters":null,"isBase64Encoded":false,"multiValueQueryStringParameters":{"inGame":["1"],"userName":["u"]},"multiValueHeaders":{"sec-fetch-mode":["cors"],"referer":["http:\/\/localhost:3000\/"],"sec-fetch-site":["cross-site"],"accept-language":["en-GB,en-US;q=0.9,en;q=0.8"],"origin":["http:\/\/localhost:3000"],"User-Agent":["Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36"],"X-Forwarded-Proto":["https"],"Host":["cjh1f85qo9.execute-api.us-east-2.amazonaws.com"],"dnt":["1"],"X-Forwarded-Port":["443"],"X-Amzn-Trace-Id":["Root=1-6251d7e0-7f1138d16c5b450b4607b618"],"accept":["application\/json, text\/plain, *\/*"],"sec-ch-ua":["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"99\", \"AVG Secure Browser\";v=\"99\""],"sec-ch-ua-mobile":["?0"],"sec-ch-ua-platform":["\"Windows\""],"X-Forwarded-For":["185.34.122.187"],"accept-encoding":["gzip, deflate, br"],"sec-fetch-dest":["empty"]},"requestContext":{"resourceId":"iq3rce","resourcePath":"\/touraments","httpMethod":"GET","extendedRequestId":"QU6rKEwkCYcFjeg=","requestTime":"09\/Apr\/2022:19:00:48 +0000","path":"\/Develop\/touraments","accountId":"931314964676","protocol":"HTTP\/1.1","stage":"Develop","domainPrefix":"cjh1f85qo9","requestTimeEpoch":1649530848856,"requestId":"69f0c156-bb06-4562-949e-6633d59bbe82","identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"sourceIp":"185.34.122.187","principalOrgId":null,"accessKey":null,"cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,"userAgent":"Mozilla\/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/99.0.4844.84 Safari\/537.36","user":null},"domainName":"cjh1f85qo9.execute-api.us-east-

	2.amazonaws.com","apiId":"cjh1f85qo9"},"resource":"\/touraments","httpMethod":"GET","queryStringParameters":{"inGame":"1","userName":"u"},"stageVariables":null,"body":null}
2022-04-09T20:00:50.636+01:00	19:00:50.636 [cluster-ClusterId{value='6251d7e23c2fb136093827c5', description='null'}-srv-serverlessinstance0.dgmiv.mongodb.net] INFO org.mongodb.driver.cluster - SRV resolution completed with hosts: [serverlessinstance0-lb.dgmiv.mongodb.net:27017]
2022-04-09T20:00:57.033+01:00	19:00:57.033 [main] INFO org.mongodb.driver.connection - Opened connection [connectionId{localValue:1, serverValue:32068}] to serverlessinstance0-lb.dgmiv.mongodb.net:27017
2022-04-09T20:00:57.216+01:00	19:00:57.216 [main] DEBUG org.mongodb.driver.protocol.command - Sending command '{"find": "Games", "filter": {"players": "u"}, "$db": "Names", "lsid": {"id": {"$binary": {"base64": "OErYlG2tQrKFDauvXkXtBQ==", "subType": "04"}}}, "apiVersion": "1"}' with request id 4 to database Names on connection [connectionId{localValue:1, serverValue:32068}] to server serverlessinstance0-lb.dgmiv.mongodb.net:27017
2022-04-09T20:00:57.300+01:00	19:00:57.300 [main] DEBUG org.mongodb.driver.protocol.command - Execution of command with request id 4 completed successfully in 125.16 ms on connection [connectionId{localValue:1, serverValue:32068}] to server serverlessinstance0-lb.dgmiv.mongodb.net:27017
2022-04-09T20:00:57.733+01:00	END RequestId: 4cd350b8-11d4-42ab-8db7-862ae8b74530
2022-04-09T20:00:57.733+01:00	REPORT RequestId: 4cd350b8-11d4-42ab-8db7-862ae8b74530 Duration: 7634.73 ms Billed Duration: 7635 ms Memory Size: 512 MB Max Memory Used: 127 MB Init Duration: 951.28 ms

Reference

[1] (https://www.cloudflare.com/learning/serverless/what-is-serverless/ .)

[2](https://www.baeldung.com/java-mongodb)

[3]https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html

[4]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/tree/main/WireFrames

[5]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/blob/main/Document/Christmas-Presentation.pdf

[6]https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22/commit/26cccc049279ae12e38033d3b619299ae00a011d

[7](https://www.baeldung.com/aws-lambda-api-gateway)

[8] (https://www.mongodb.com/blog/post/introducing-serverless-instances-mongodb-atlas-now-available-preview).

[9](https://www.mongodb.com/pricing)

[10](https://aws.amazon.com/api-gateway/#:~:text=Amazon%20API%20Gateway%20is%20a,functionality%20from%20your%20backend%20services).

[11]https://github.com/google/gson

[12](https://aws.amazon.com/cloudwatch/features/#:~:text=CloudWatch%20enables%20you%20to%20monitor,building%20applications%20and%20business%20value.)

[13] https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html

[14] (https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format)

[15]https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-output-format

[16](https://aws.amazon.com/premiumsupport/knowledge-center/api-gateway-cloudwatch-logs/)

[17](https://endoflife.date/java)

[18](https://builtin.com/software-engineering-perspectives/cold-starts-challenge-serverless-architecture).

[19](https://www.capitalone.com/tech/cloud/aws-lambda-java-tutorial-reduce-cold-starts/)

[20](https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html#configuration-memory-console)

[21](https://www.graalvm.org/22.0/docs/introduction/).

[22](https://shinesolutions.com/2021/08/30/improving-cold-start-times-of-java-aws-lambda-functions-using-graalvm-and-native-images/)

https://www.capitalone.com/tech/cloud/aws-lambda-java-tutorial-reduce-cold-starts/

https://www.cloudflare.com/learning/serverless/what-is-serverless

https://www.bmc.com/blogs/serverless-computing/

https://learntocodewith.me/posts/serverless-architecture/

https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-deploy-api.html

https://www.baeldung.com/java-mongodb

https://www.baeldung.com/aws-lambda-api-gateway

https://github.com/Eoin-Lernihan/Final-Year-Project-2021-22

https://www.mongodb.com/pricing

https://aws.amazon.com/free/?trk=ce1f55b8-6da8-4aa2-af36-3f11e9a449ae&sc_channel=ps&sc_campaign=acquisition&sc_medium=ACQ-P|PS-GO|Brand|Desktop|SU|AWS|Core|UK|EN|Text&s_kwcid=AL!4422!3!433803620879!e!!g!!aws%20pricing&ef_id=EAIaIQobChMIzZ_9jZGK9wIViLTtCh1BsQt0EAAYAiAAEgJUzfD_BwE:G:s&s_kwcid=AL!4422!3!433803620879!e!!g!!aws%20pricing&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=tier%23always-free&awsf.Free%20Tier%20Categories=*all

https://aws.amazon.com/premiumsupport/knowledge-center/api-gateway-cloudwatch-logs/