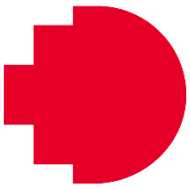


RMIT University Vietnam

School of Science, Engineering and Technology (SSET)

Assignment 2

Build a scalable app on cloud

Course: COSC2638 – Cloud Computing

Group: Group 2 - Exsper

Members: Tran Phuong Anh – s3914138

Nguyen Minh Anh – s3927589

Bui Minh Nhat – s3878174

Thieu Tran Tri Thuc – s3870730

Lai Nghiep Tri – s3799602

Lecturers: Thanh Ngoc Nguyen

Submission Date: September 15th 2022

Contents

ABBREVIATION	MEANING
AWS	Amazon Web Service
BAAS	Back-end as a Service
ENV	Environment

I. Contribution

Table 1: Contribution among members (%)

Members	Percentage (%)
Tran Phuong Anh	20
Nguyen Minh Anh	20
Bui Minh Nhat	20
Thieu Tran Tri Thuc	20
Lai Nghiep Tri	20

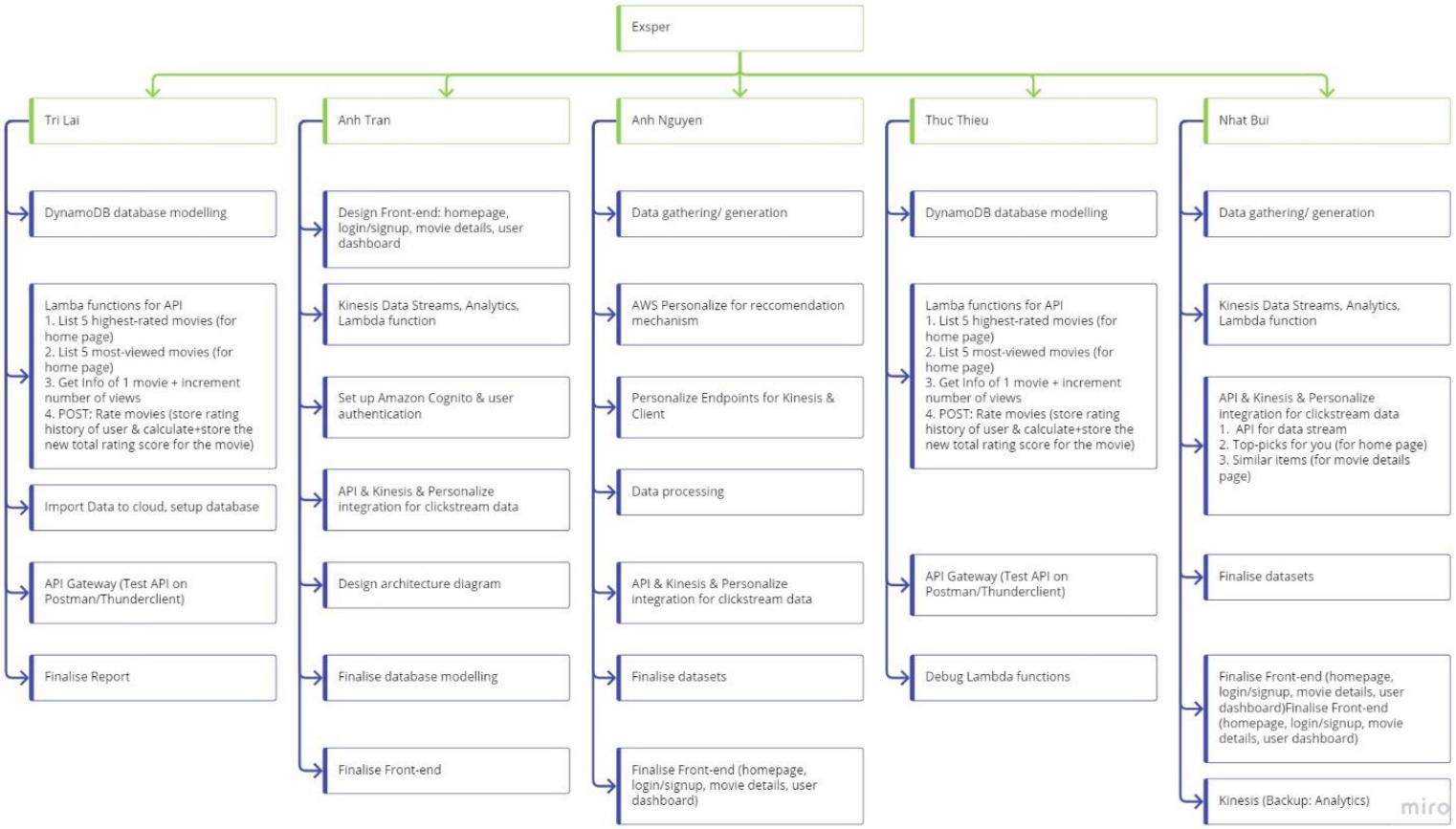


Figure 1: Work Breakdown Structure (WBS)

II. Links

Github Repository:

[baubau09/exsper-imdb (github.com)]

Demo Application Link:

https://www.exsper.rmitvn.io

Demo user login information

User 1

Username: s3914138@rmit.edu.vn

Password: 12345678

User 2

Username: s3927589@rmit.edu.vn

Password: 12345678

III. Summary

In this project, the team utilizes the fundamentals of cloud computing to develop a web application, which is deployed on Amazon Web Services. By integrating multiple cloud services such as AWS Amplify, Amazon Cognito, Amazon S3, Amazon API Gateway, AWS Lambda, Amazon DynamoDB, Amazon Kinesis Data Streams & Firehose, and Amazon Personalize, we could quickly ship a complex system where users' data could be collected and reproduce for analytics and improve the business logic.

IV. Introduction

The team decided to build a more basic version that nevertheless does the essential tasks after being inspired by the IMDB web application, whose purpose is to offer a variety of lists of movies along with other details (IMDB, 2022). Since users must actively search for each movie, it is inefficient to pick a film that satisfies both personal preferences and cinematic trends. The problem can escalate if users' viewing habits are visualized without their data. The purpose of this program is to allow users to explore, rate certain movies, and suggest additional films in the same genre as a result of the system's ability to track their behaviour. The app's benefit

over other movie review websites is its unique and beautiful interface design, which allows users to explore movies and ratings in a seamless manner.

V. Related Work

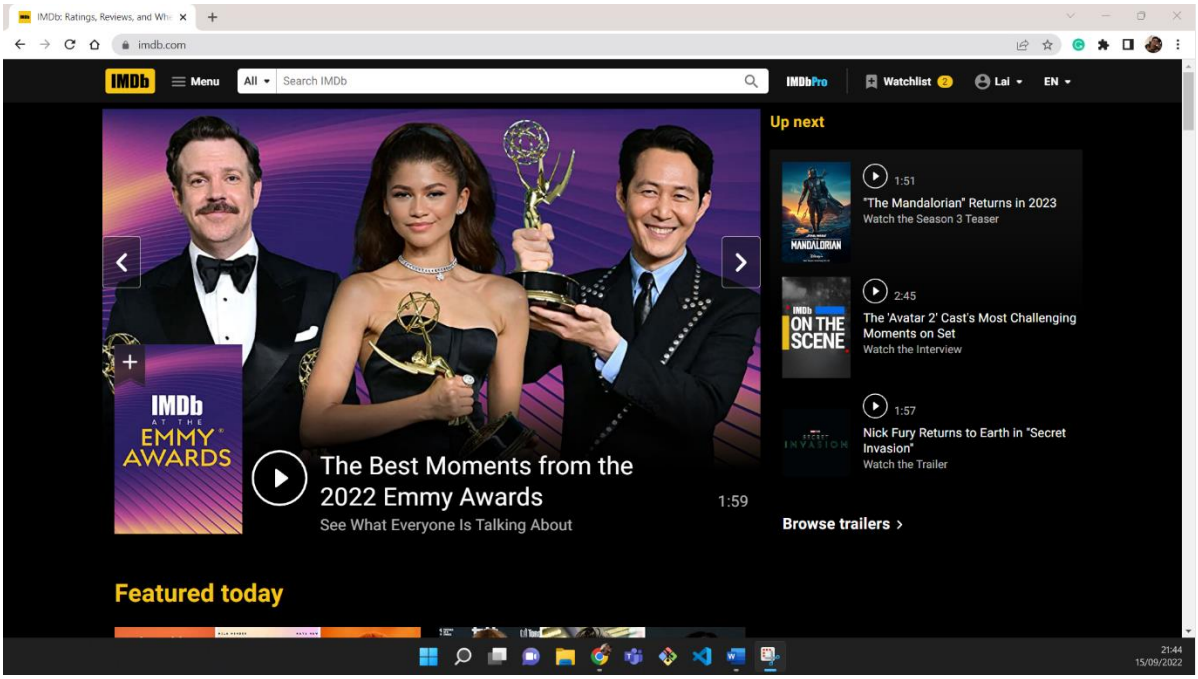


Figure 2: IMDb's Homepage

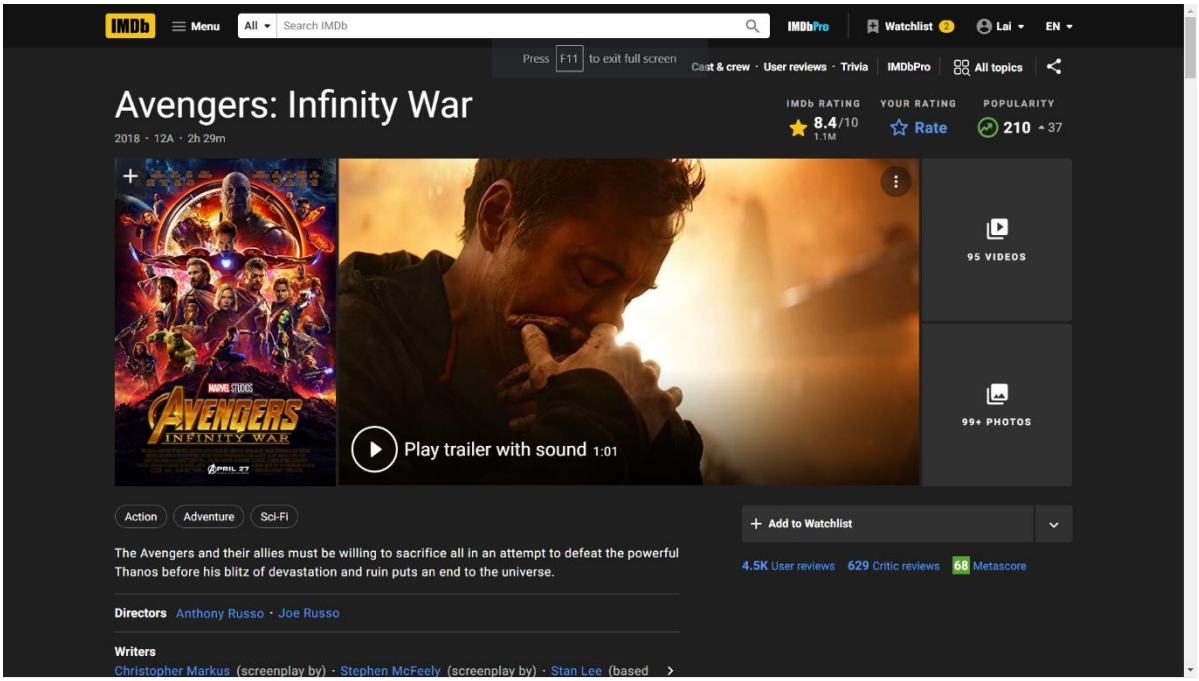


Figure 3: IMDb's film details page

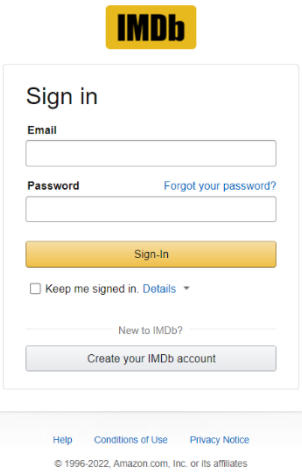


Figure 4: IMDB's sign-in form

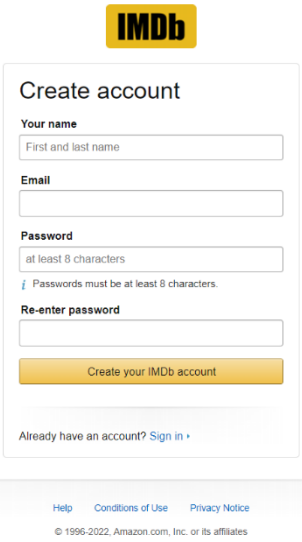


Figure 5: IMDb's sign-up form

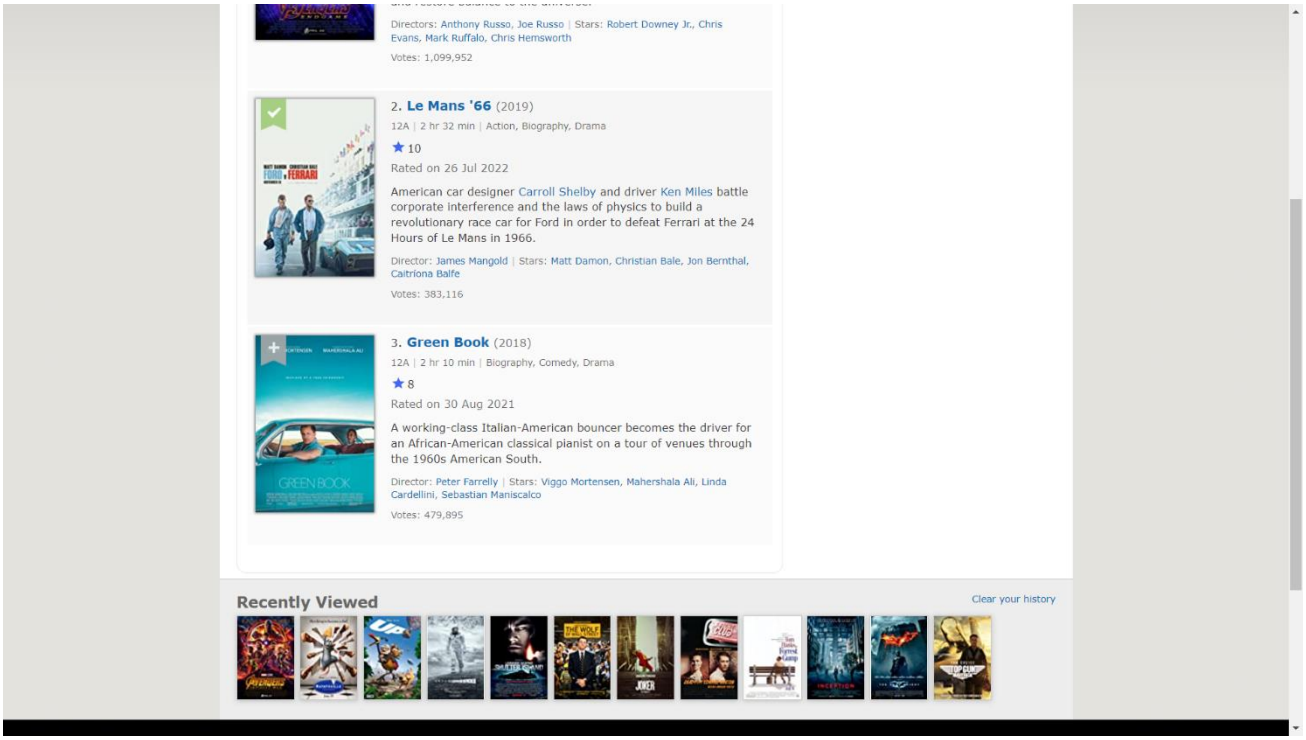


Figure 6: IMDb's User Ratings history and Recent Views

VI. Software Design and Architecture

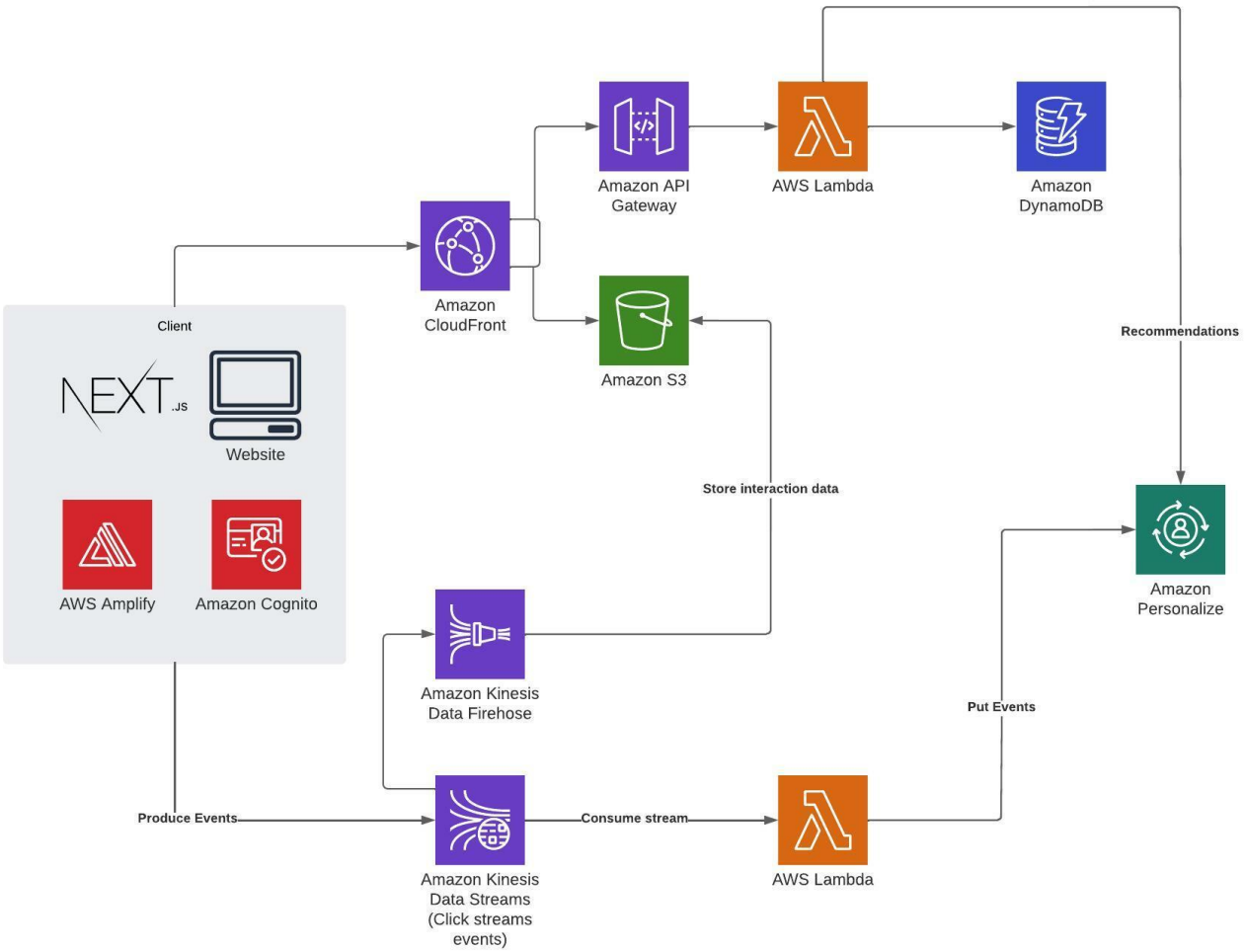


Figure 7: Overview of the Exsper-IMDB components

The app is designed using NEXTJS on the client side, which is based on React and supports hybrid static and server-side rendering, allowing the webpage to load more faster with fully rendered content at the server and the full-stack application will be hosted by AWS Amplify, which serves as a back-end as a service (BaaS). Additionally, we utilize AWS Cognito to build sign-in forms, then authenticate users through an external identity provider or credentials to access the app's resources. Then, using AWS API Gateway to make API calls in order to connect to the back-end side.

The datasets gathered from GroupLens [1] are slightly adjusted on the database side to fit the project context before being transferred to the AWS S3 bucket, which

follows a NoSQL model, will be later imported into AWS DynamoDB. Connecting the database through AWS Lambda, a serverless computing service, in order to conduct CRUD operations that are accessed through AWS API Gateway. We have integrated AWS Kinesis Data Streams to deal with the mechanism of movie suggestions or mouse click handlers. This allows us to record and store data streams (events), then feed them to AWS Kinesis Data Firehose to process them in real-time, which later store those processed data to S3. At the final stage, the team is using Amazon Personalize as machine learning to model the recommendation algorithm.

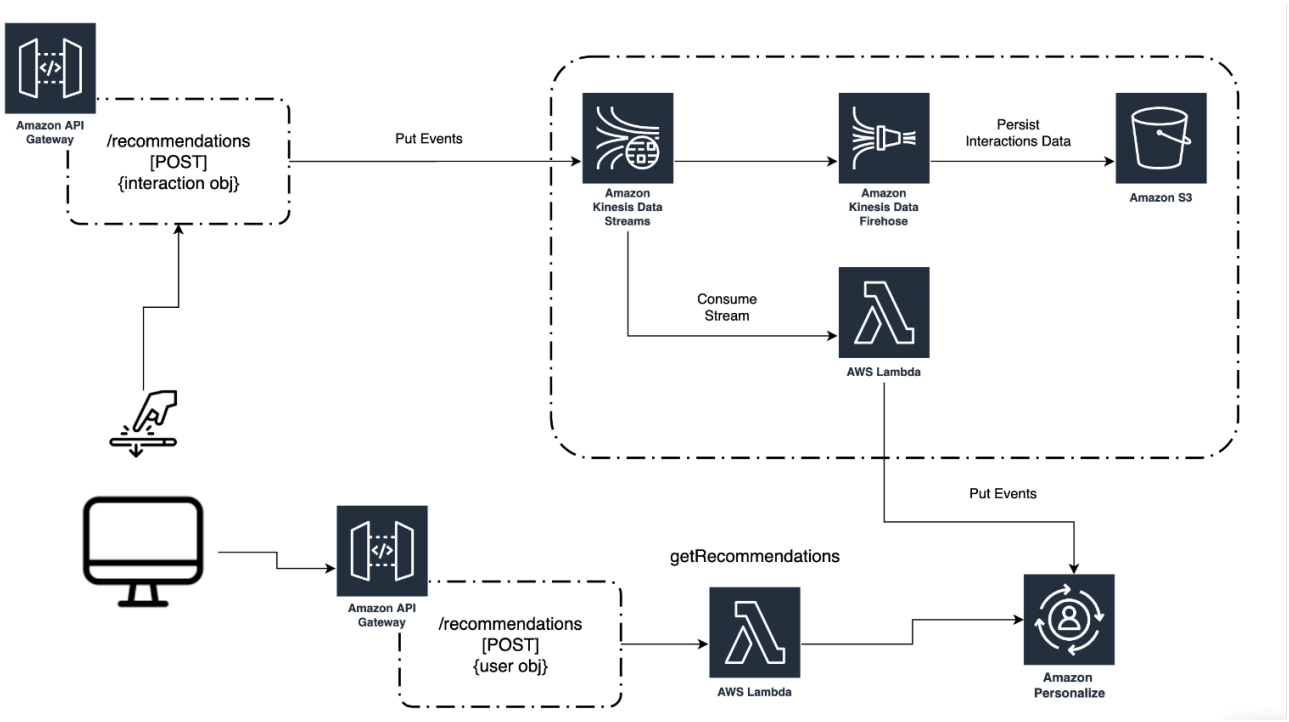


Figure 8: The architecture to construct API Layer consuming AWS Personalize recommendations and produce real time events [2]

AWS Personalize is a machine learning service which provides real-time recommendations. With the help of modern AI technology, AWS Personalize outperforms the traditional rule-based recommendation systems. In our movie system, this service is used to provide top-picked movies for a specific user and find similar movies for the target one. The frontend can get recommendations by making a request to the API Gateway which connects to an AWS Lambda function to trigger the corresponding Personalize Campaign. When users click on or watch a particular movie, the event will be sent to another API which then forwards the data to AWS Kinesis. Kinesis will record all the events until there are enough records for a batch (which is set by the developer). After that, Kinesis would trigger an AWS Lambda function which will send the events to the AWS Personalize's event tracker. After 7 days of collecting new data, Personalize will retrain the AI model to provide up to date recommendations.

VII. Implementation – Developer Manual

**Note: In this project, we assume that readers have basic understandings on AWS in order to go through this tutorial sheet. All these below Amazon services are mostly based on the tutorial note so it shares the similarities in configurations and developments.*

Create Cloud storage S3

Step 1: Create buckets

User has to create a bucket where data files or folders will be added later and there should be hierarchical structure of folders which highly depends on the team preferences.

Step 2: Upload csv data files to the buckets

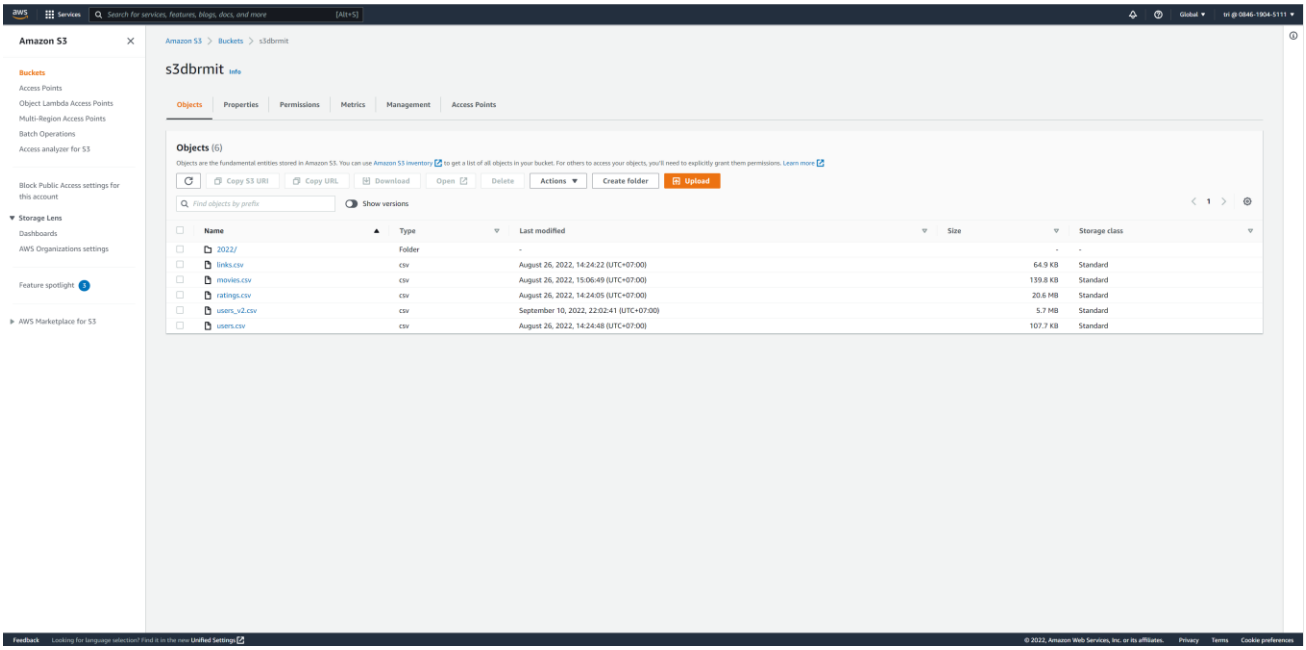


Figure 9: Data files uploaded into S3 bucket

Create data tables on DynamoDB

Step 1: Import existing sample data files from s3 buckets

Instead of creating a raw table without any data or items, we only need to import these tables, which were uploaded so far in S3 bucket, contains a list of sample data. User can refer to the ***Imported from S3*** keyword in the left-side menu [3],

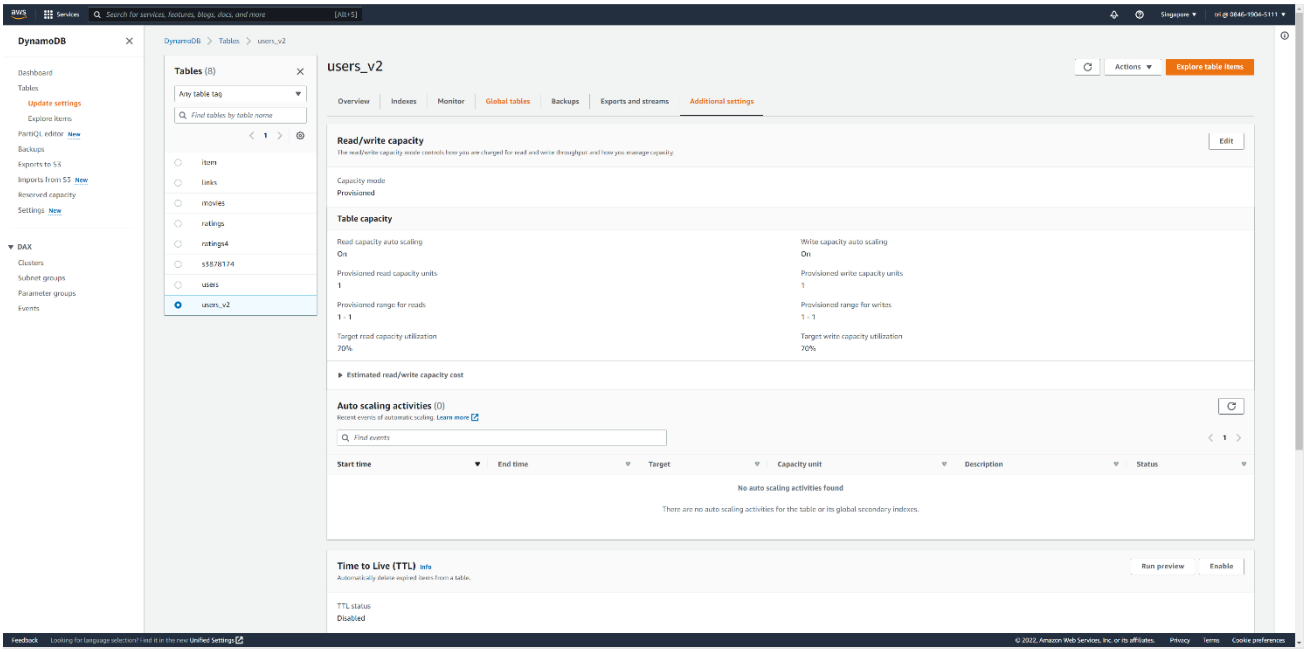


Figure 10: One of the sample buckets in our project

Step 2: Setting the import configurations [3]

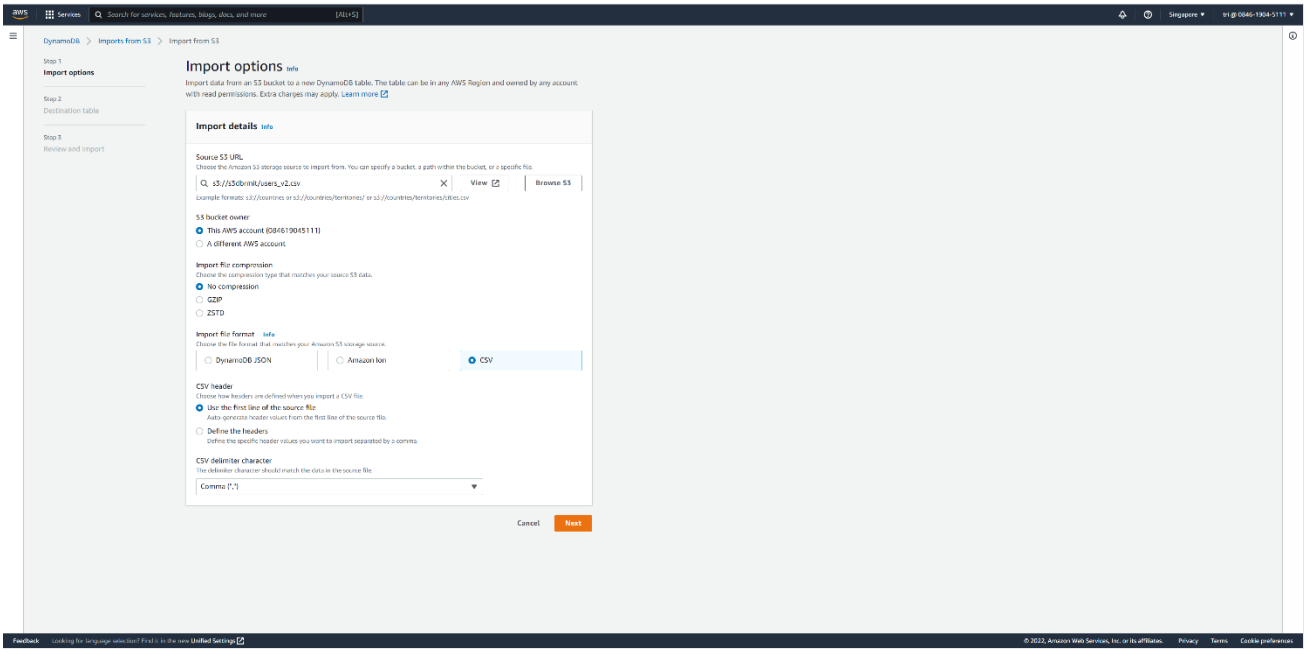


Figure 11: Set location path of data file in s3

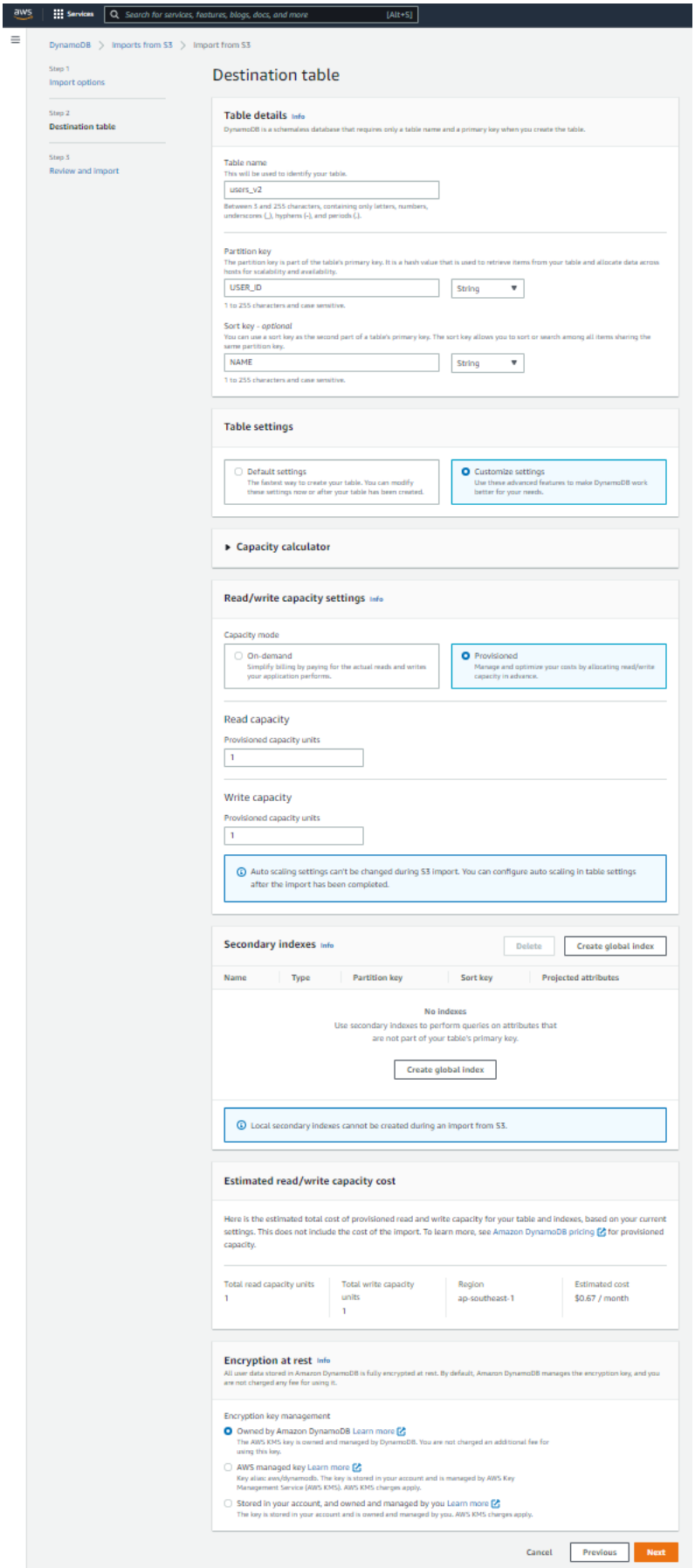


Figure 12: Set new table details with appropriate WCU and RCU (1 unit for both)

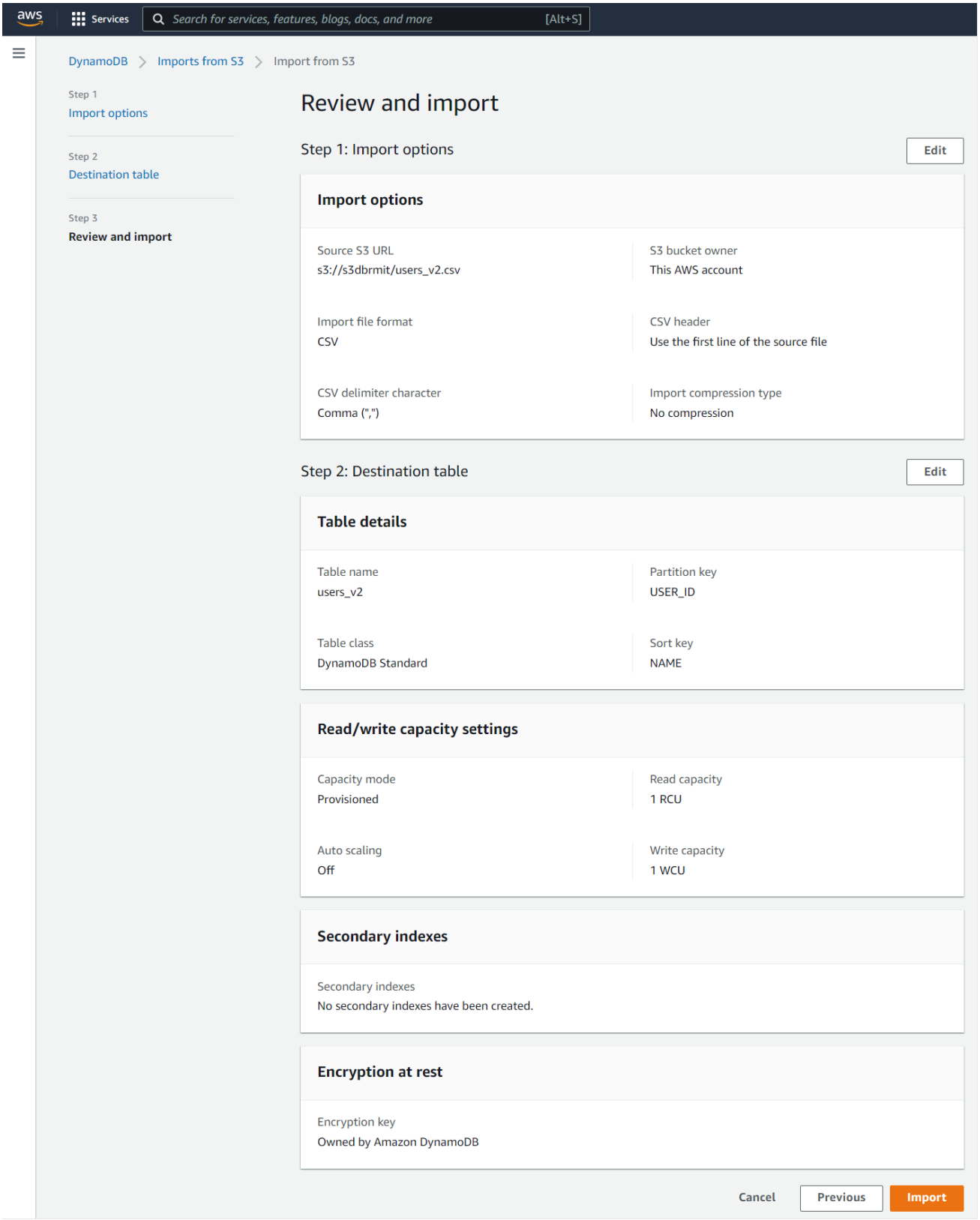


Figure 13: Review all the settings before importing the data file to DynamoDB

Create customized Lambda functions and connect to DynamoDB via API Gateway

Step 1: Create lambda function

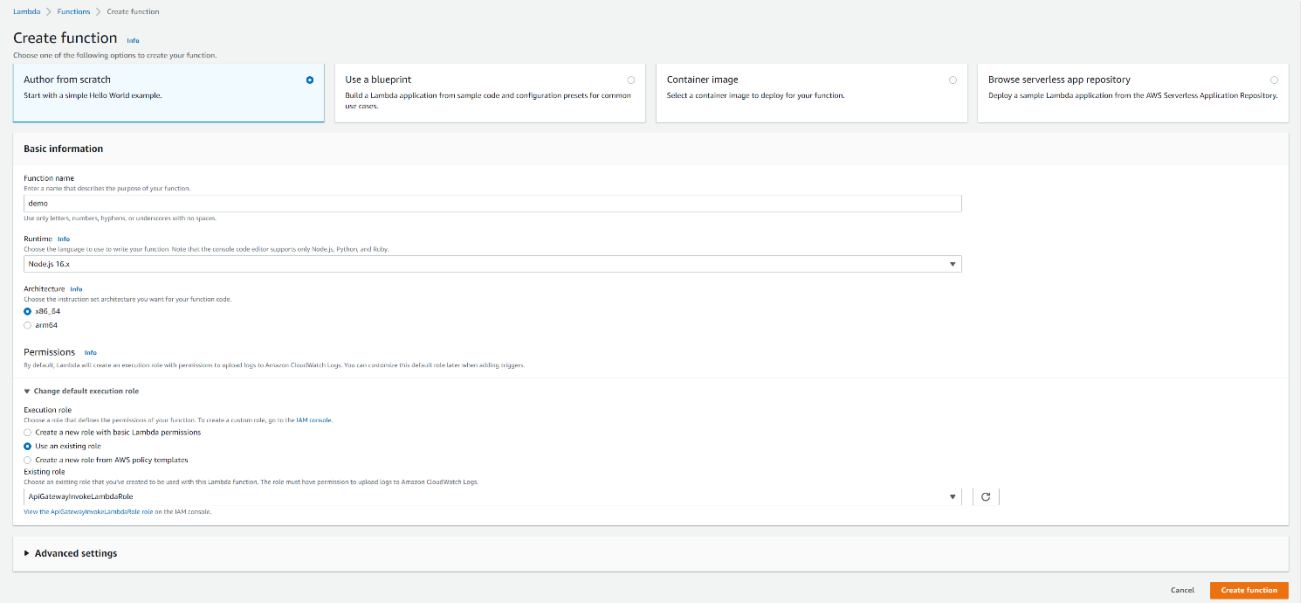


Figure 14: Create a new Lambda function

After creating the lambda function, we have to add a trigger is API Gateway in this stage since we want to the lambda function will be invoked whenever the corresponding API was called [4].

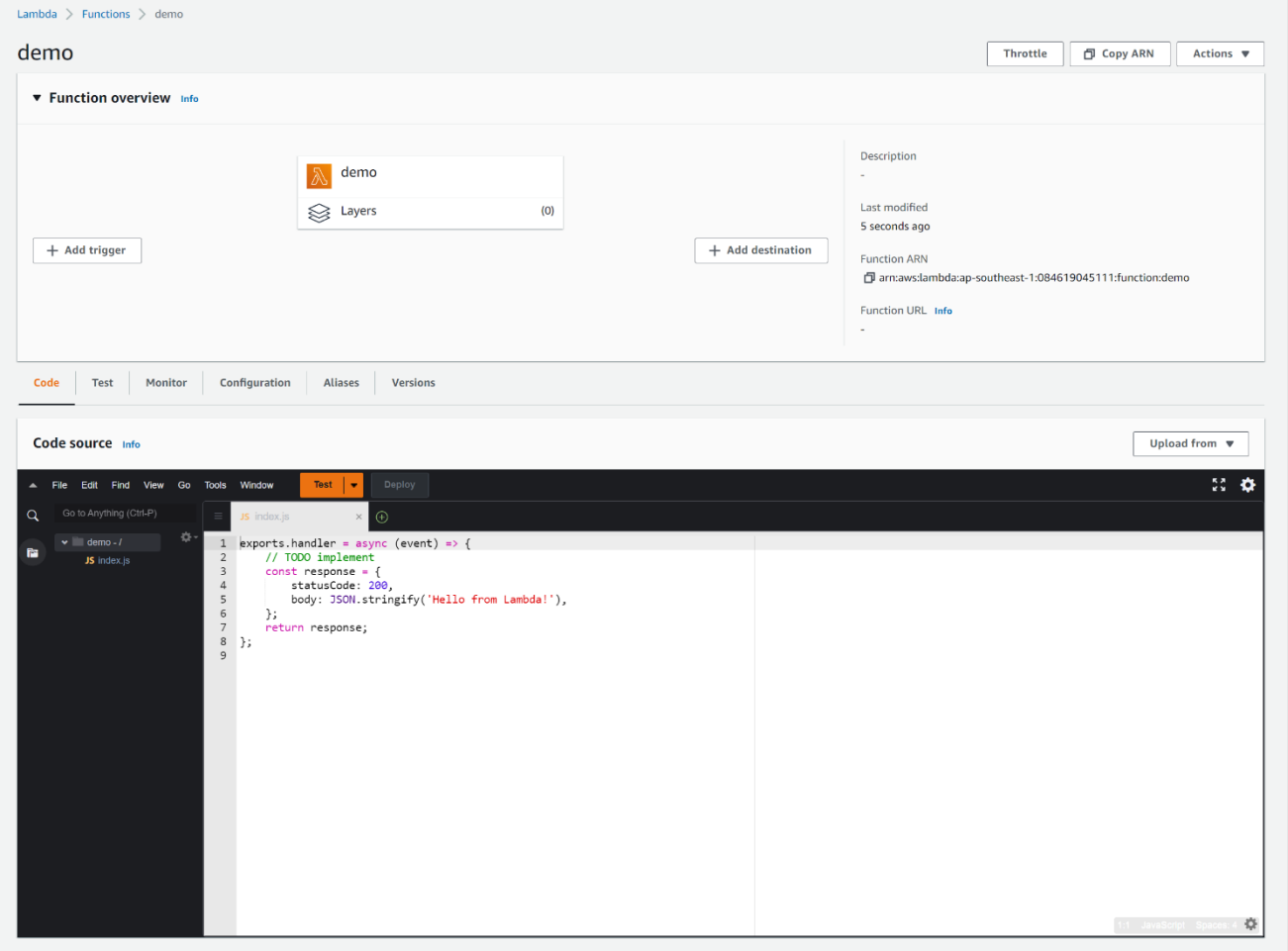


Figure 15: A newly created lambda function "demo"

Step 2: Configurate API Gateway

Searching for the AWS API Gateway service the create a new REST API with the IAM security [5].

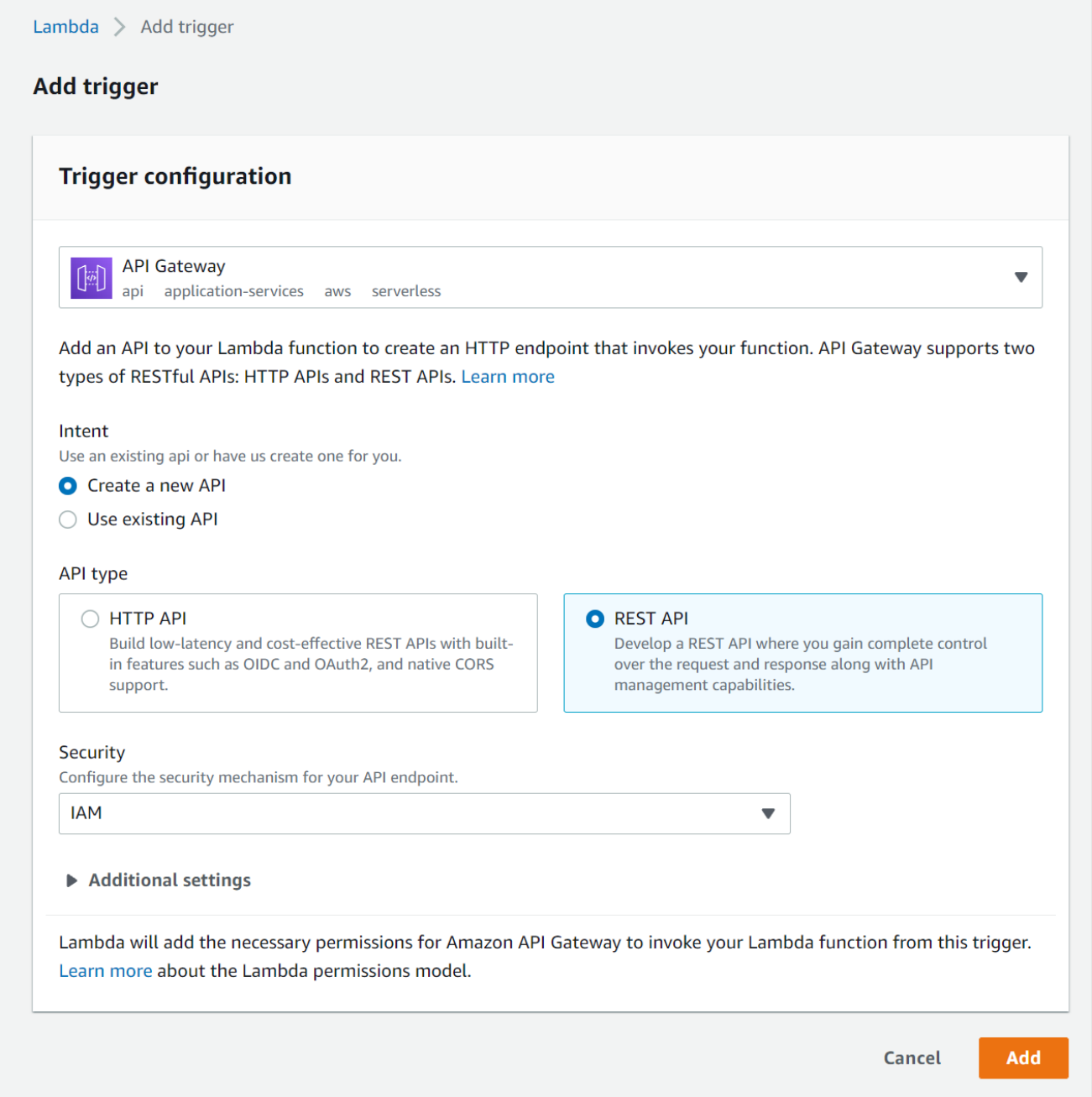


Figure 16: Trigger configurations for lambda function

After adding the API Gateway to the Lambda function, the system automatically initiates a new endpoint in the Configuration tab in order to trigger the corresponding function whenever the client has a call to that such API [5].

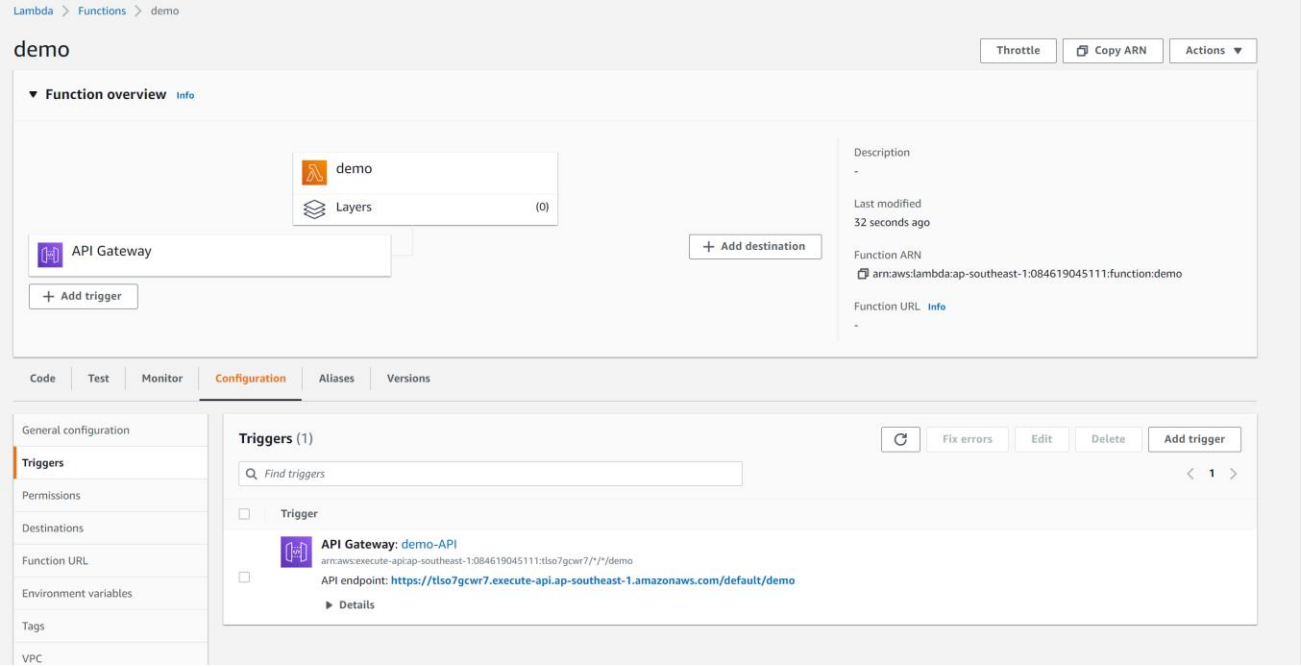


Figure 17: A new API endpoint is added to invoke to lambda function

Set Up AWS Personalize

Step 1: Upload Data to S3

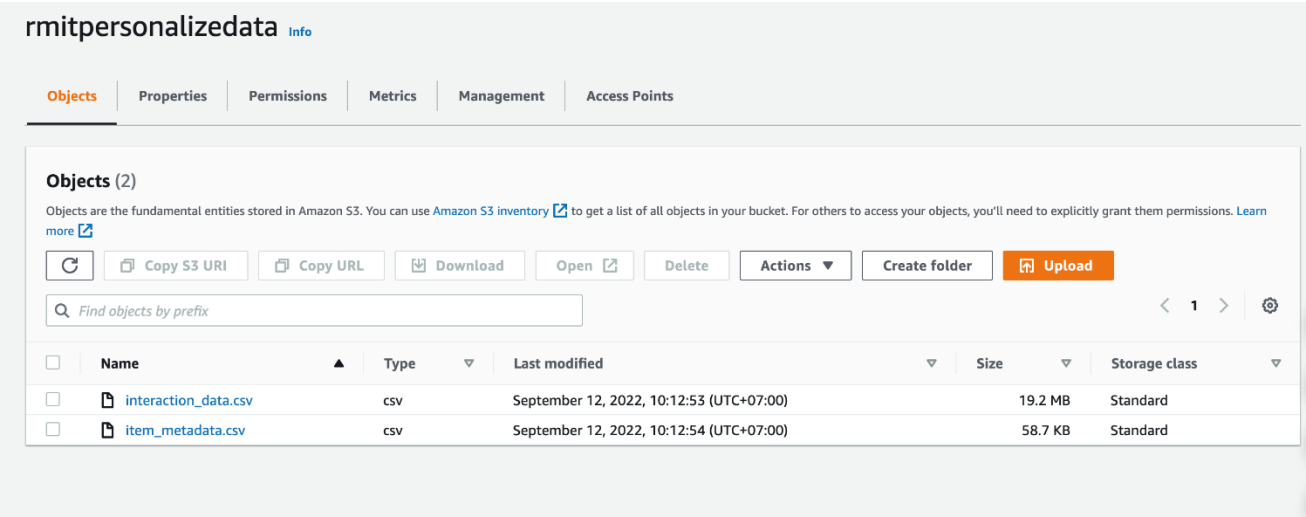


Figure 18: CSV data files uploaded into S3

Step 2: Update S3 policy

Update S3 policy so that AWS Personalize can access the data

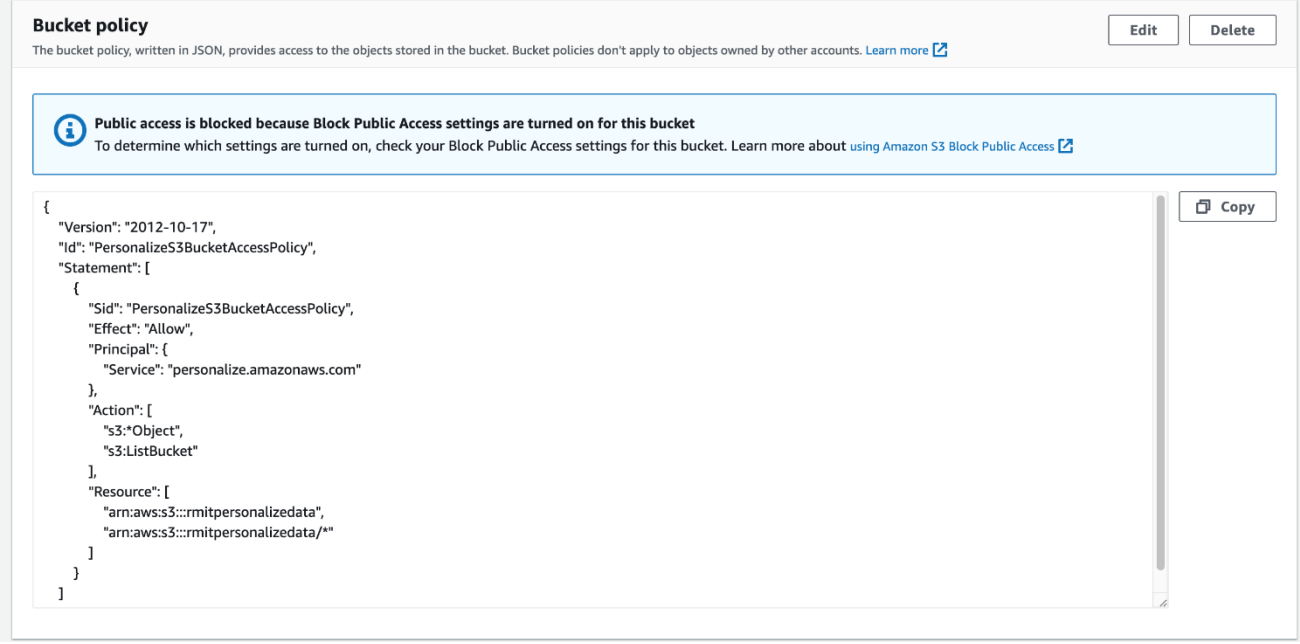


Figure 19: Policy allows AWS Personalize fully access a specific S3 resources

Step 3: Create a role for Personalize

Create a role to grant all necessary permissions for Personalize.

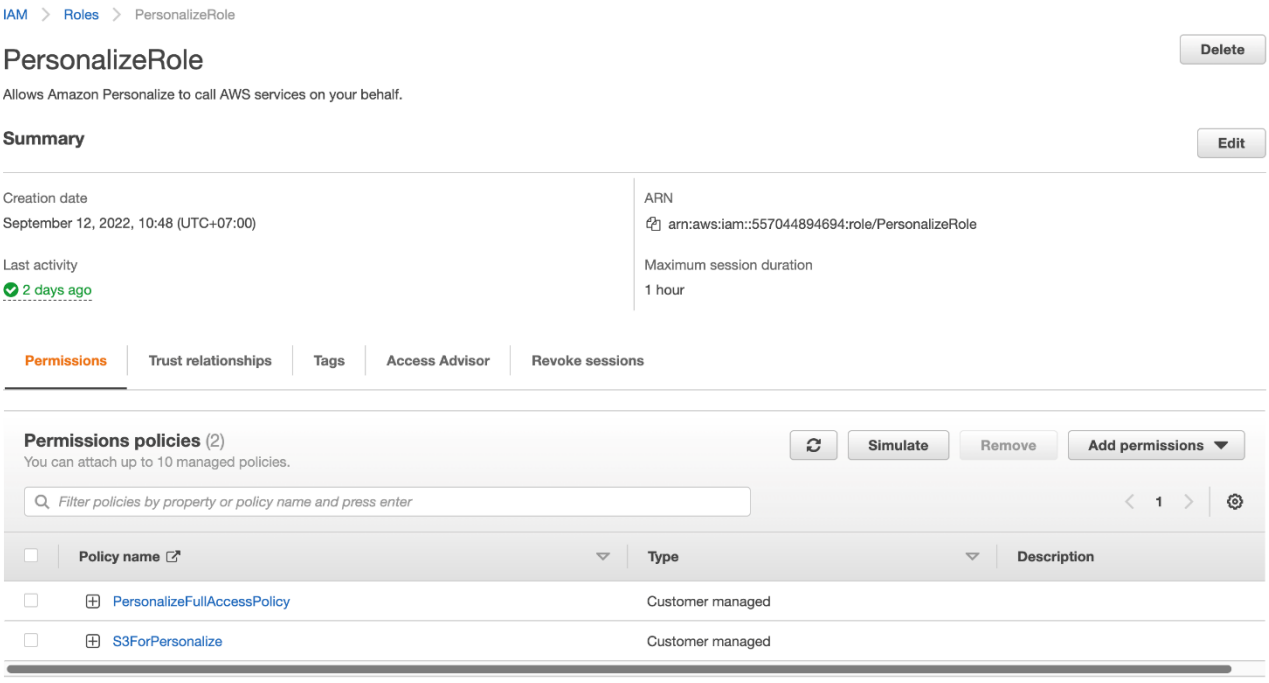


Figure 20: A new role (including built-in and customized) for AWS Personalize

Step 4: Create a Personalize dataset group

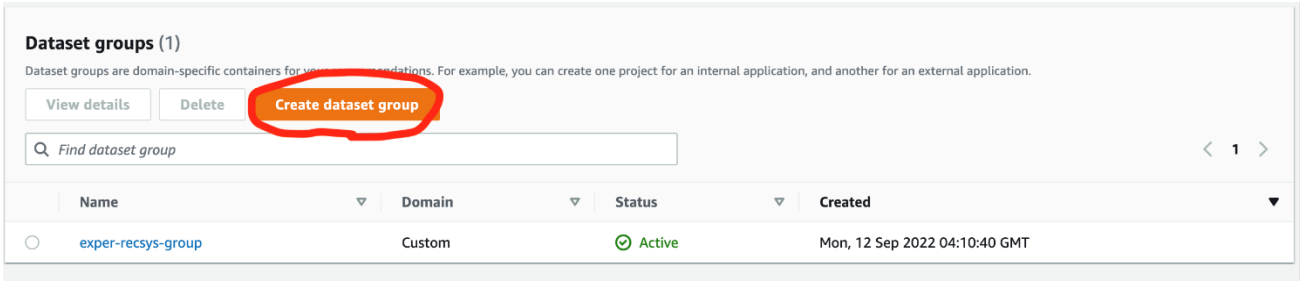


Figure 21: Create a new dataset

Step 5: Configure the dataset group

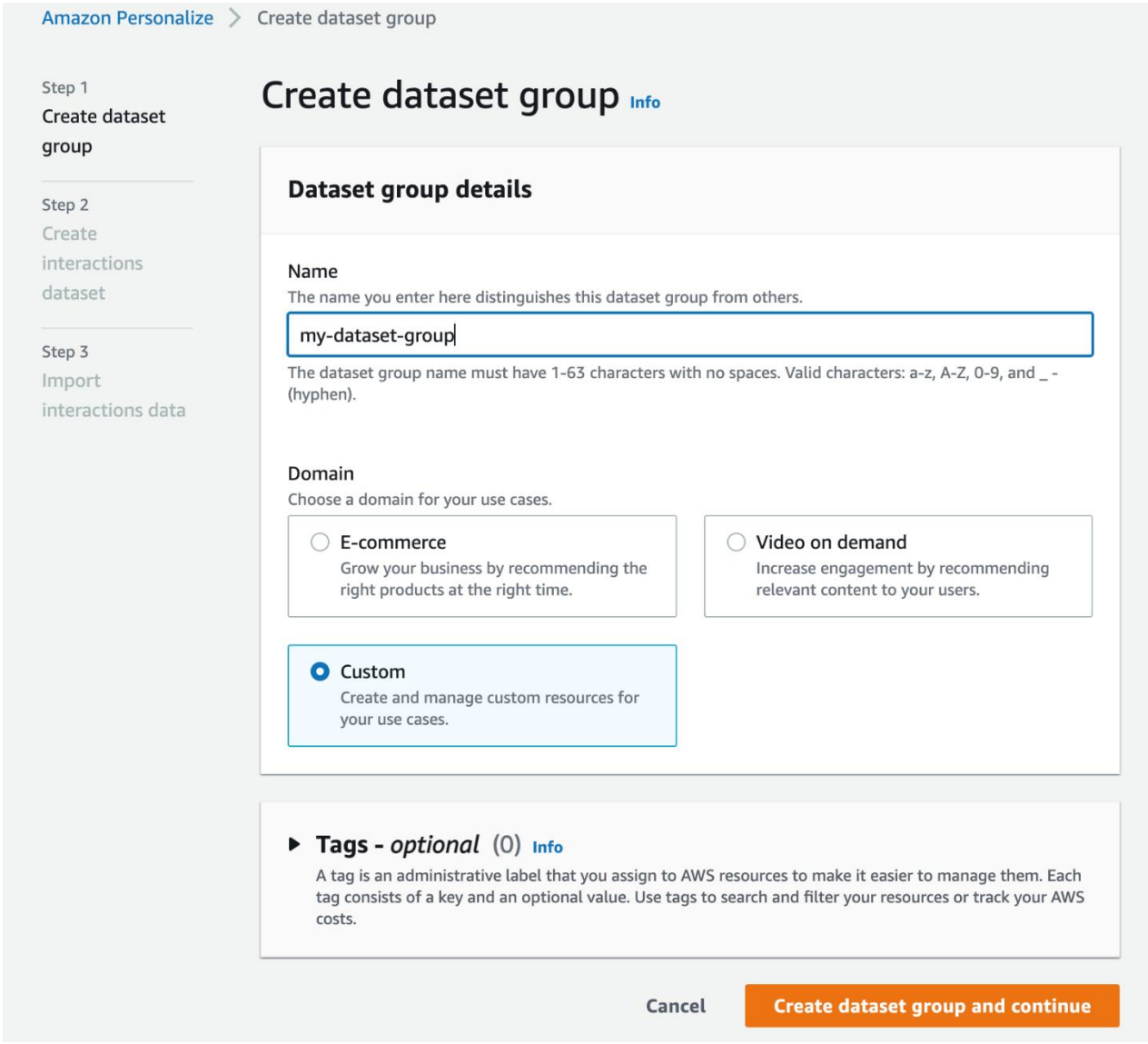


Figure 22: Configurate details for dataset group

Step 6: Create the interaction dataset

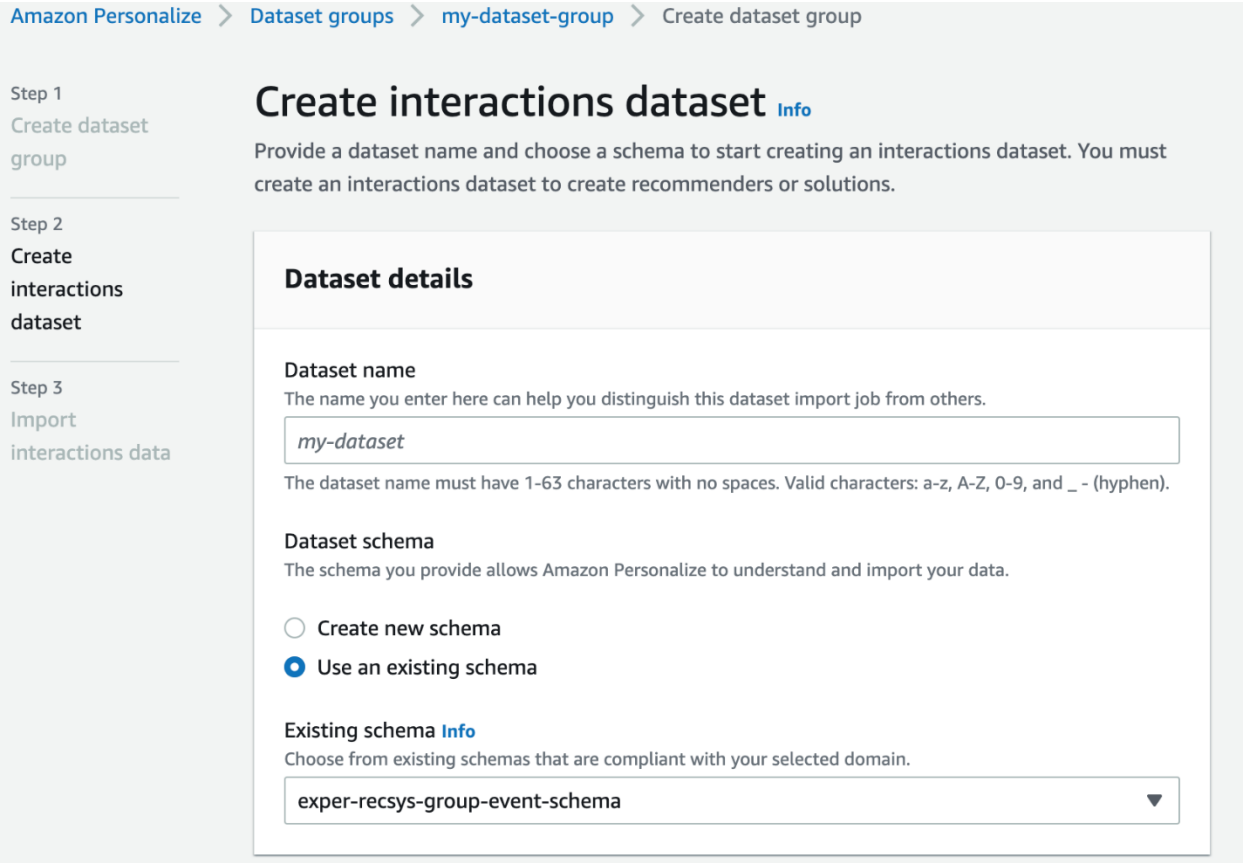


Figure 23: Create an interaction dataset in order to create recommendations

Step 7: Setup the dataset schema

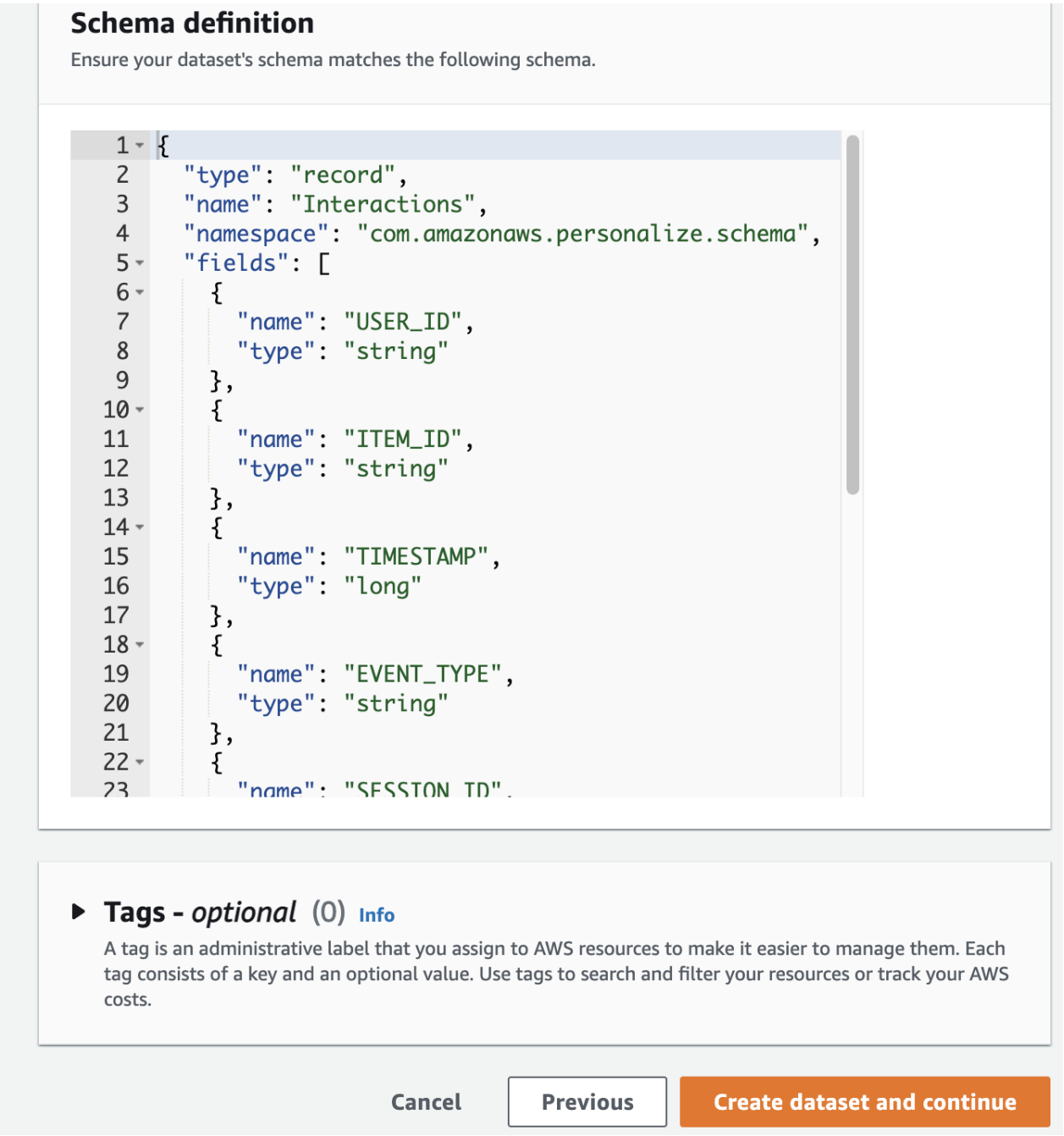


Figure 24: Create a dataset schema based on an existing schema

Step 8: Import file from S3 and attach the above-created role

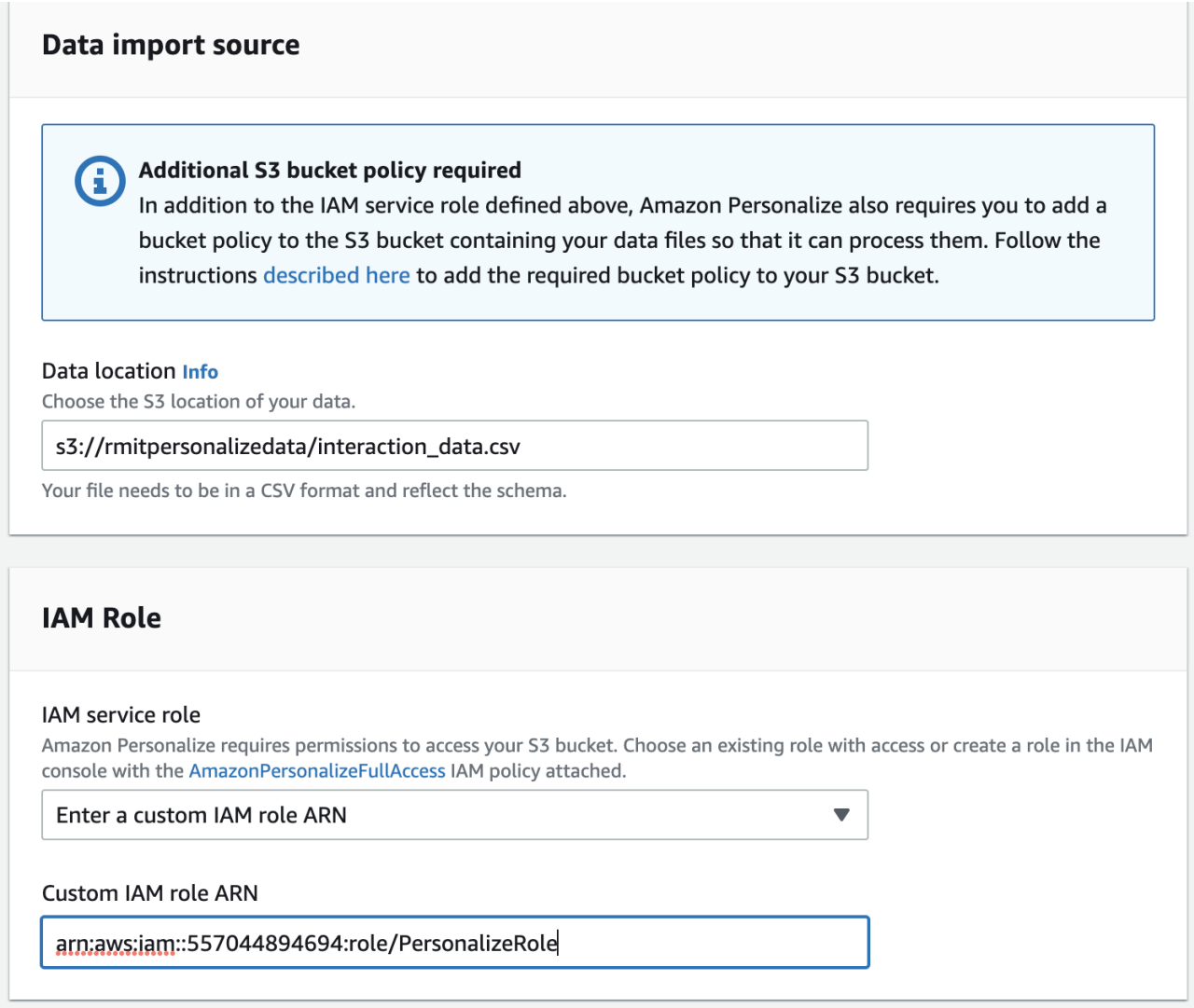


Figure 25: Import CSV data from S3 into the new dataset group along with create a new IAM role based on the customized role so far

Step 9: Repeat step 6 to 8 to create another dataset for movie items

Step 10: Create and train solutions

The solution will retrieve data from S3 and start training a machine learning model for the chosen task. Below is the setup for the user aws-personalize-ranking solution.

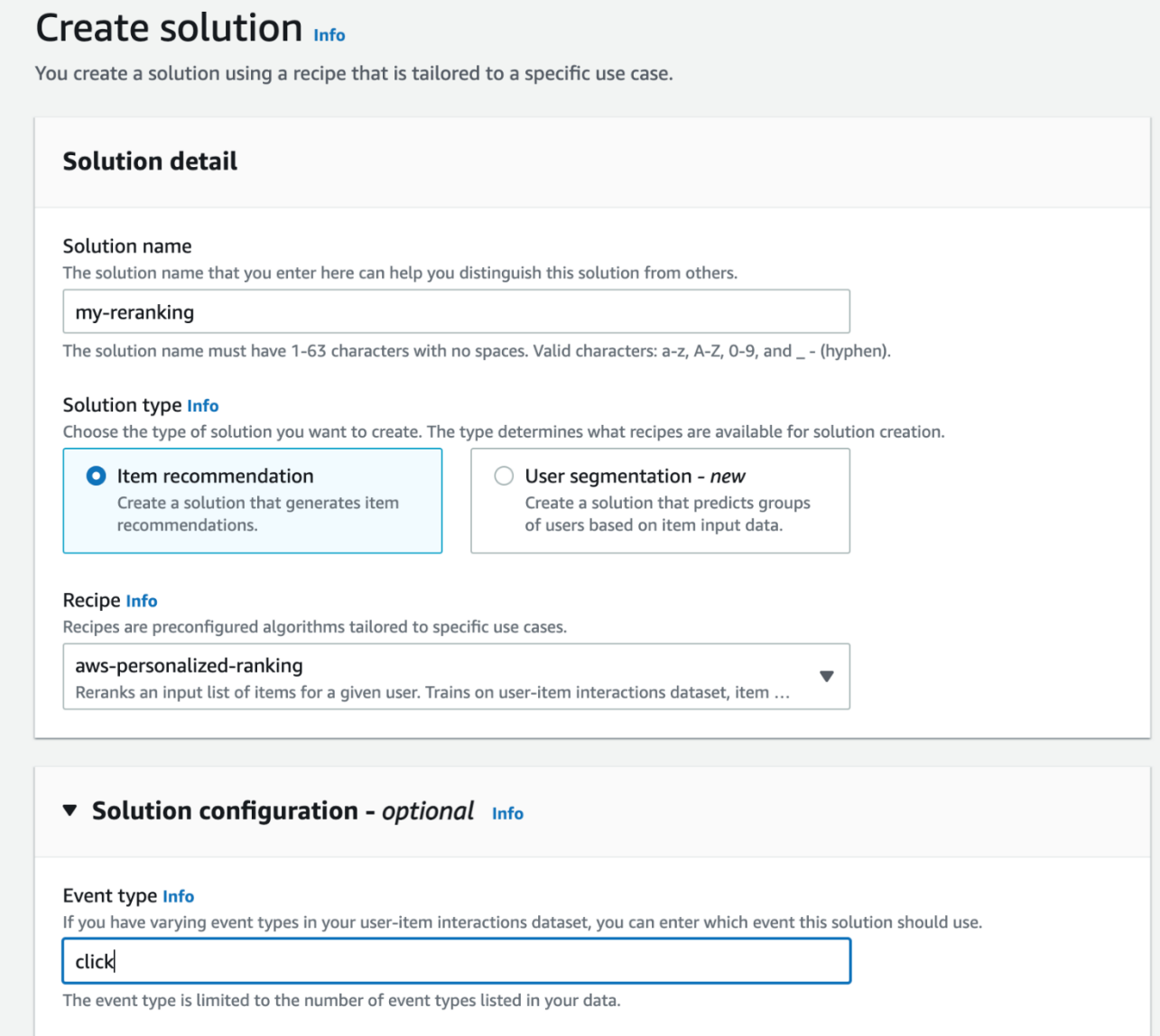


Figure 26: Create a solution call "my-reranking" attached by "aws-personalized-ranking" recipe

In our system, we also created 2 more solutions: aws-similar-items and aws-user-personalize.

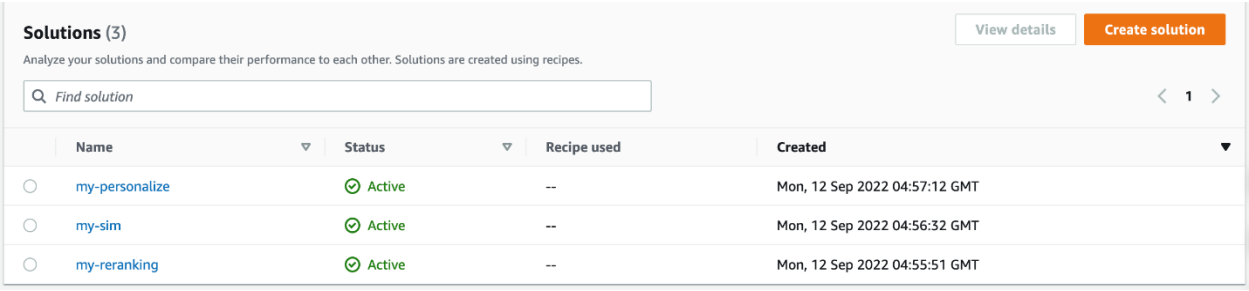


Figure 27: Create two more solutions "aws-similar-items" and "aws-user-personalize"

Step 11: Create a campaign for each solution

A campaign will serve a solution so that we and send a request to get recommendations via AWS Client API.

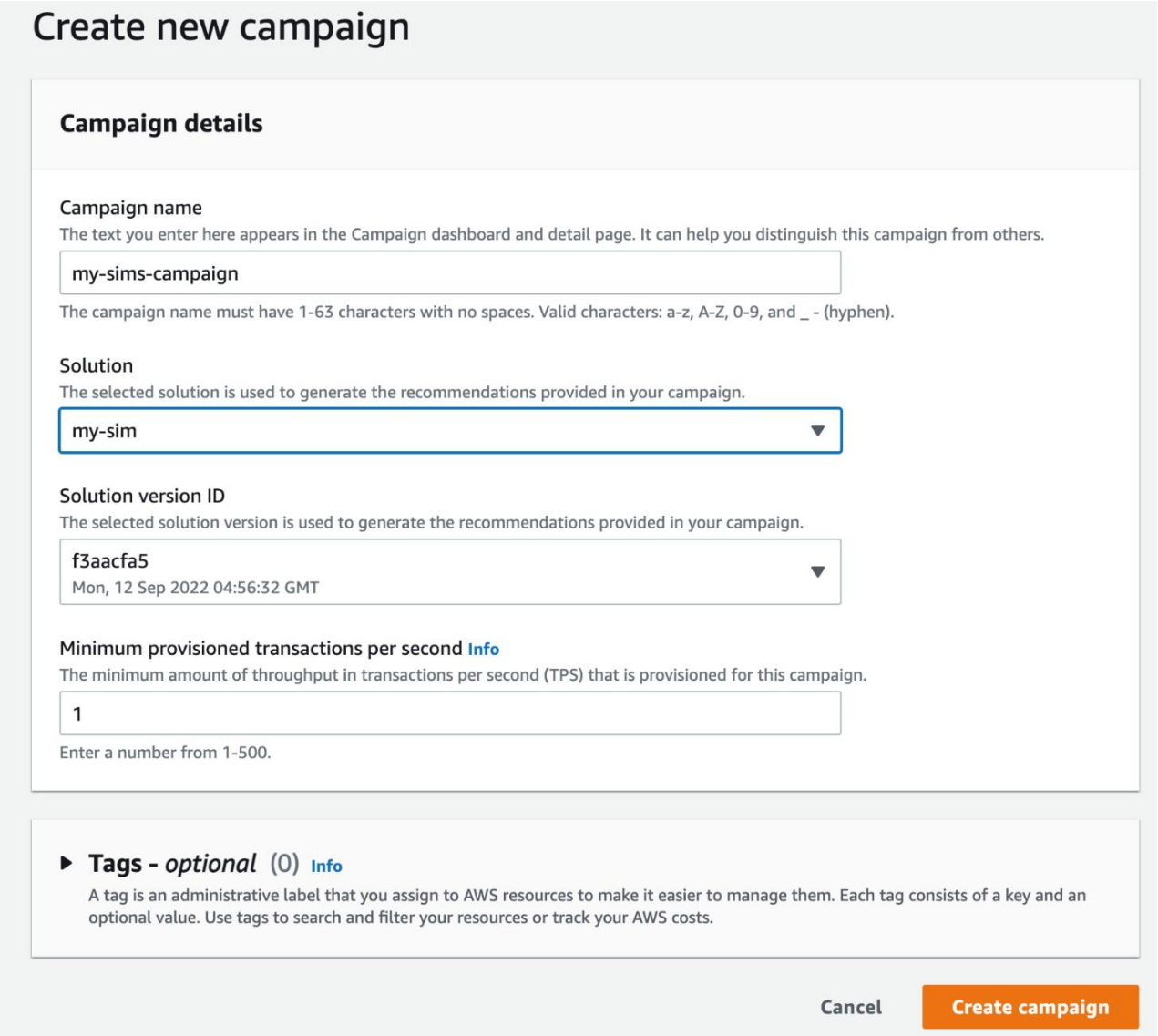


Figure 28: Create a campaign

Step 12: Create an Event Tracker

The Event Tracker will be responsible for collecting Put Events sent from Kinesis.

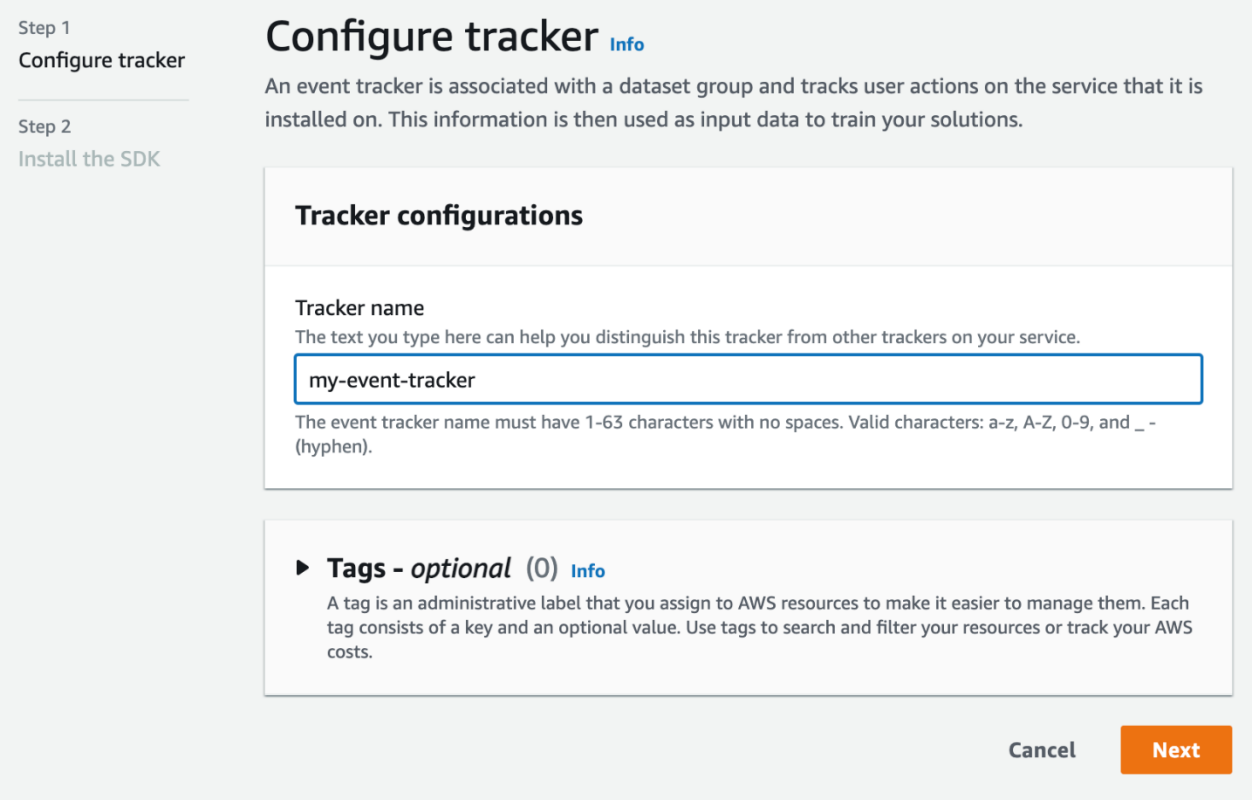


Figure 29: Create event tracker

Step 13: Create an API and Lambda function

Create an API using AWS API Gateway and build a Lambda function to get recommendations from Personalize.

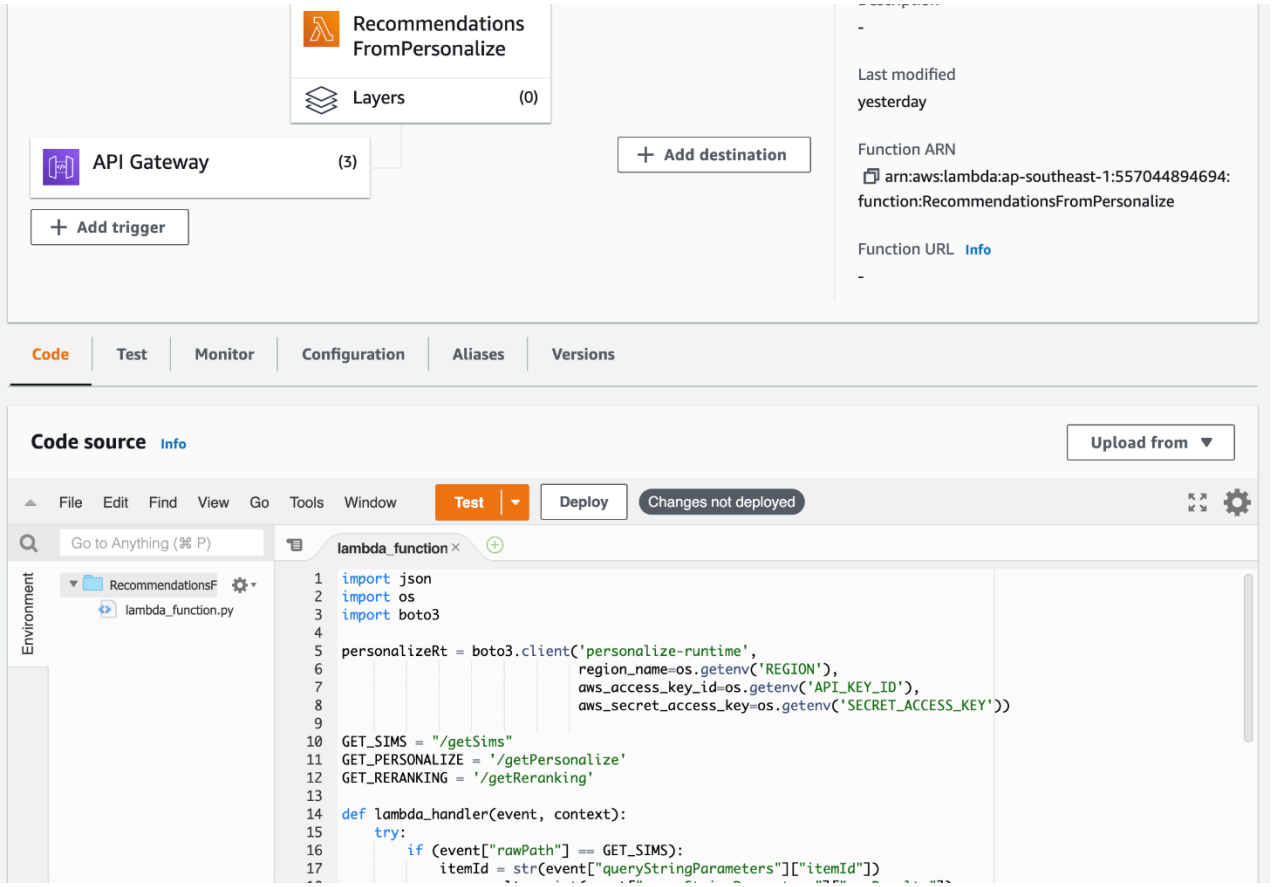


Figure 30: Create Lambda function associated to Personalize and a API Gateway as a client API

Set up AWS Amplify

Step 1: Create a new app by connecting with Github in Amplify console

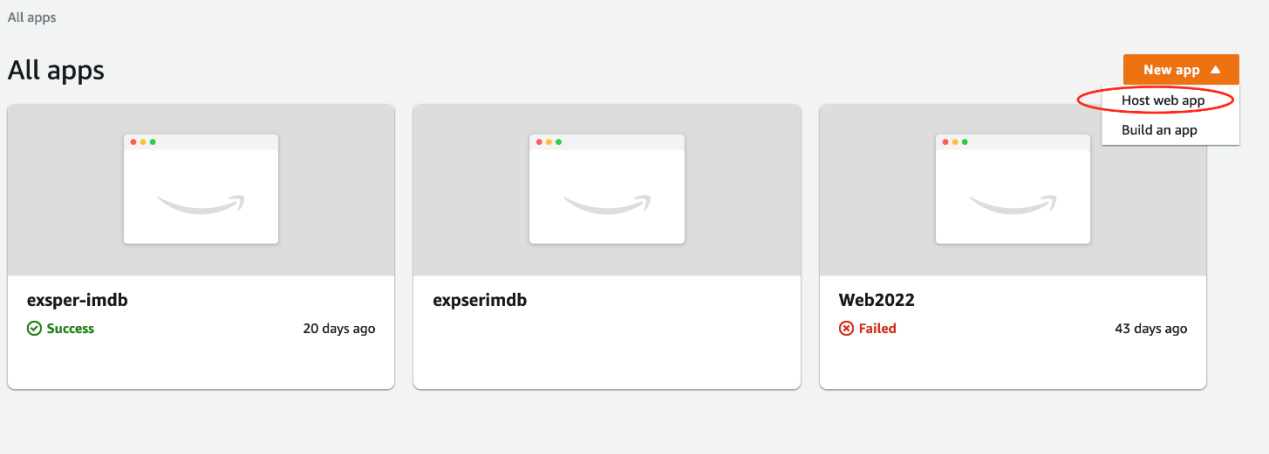


Figure 31: Create an empty app at first

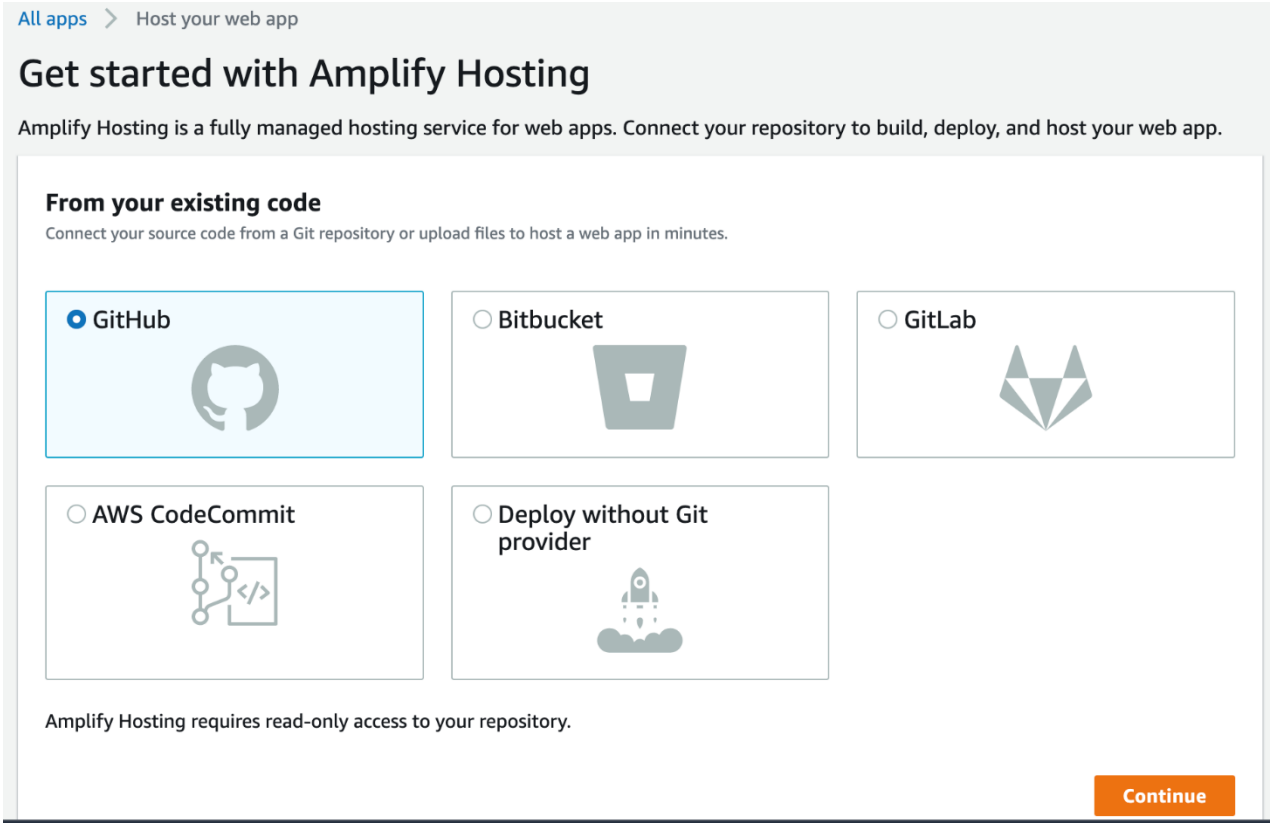


Figure 32: Use source code from GitHub repo

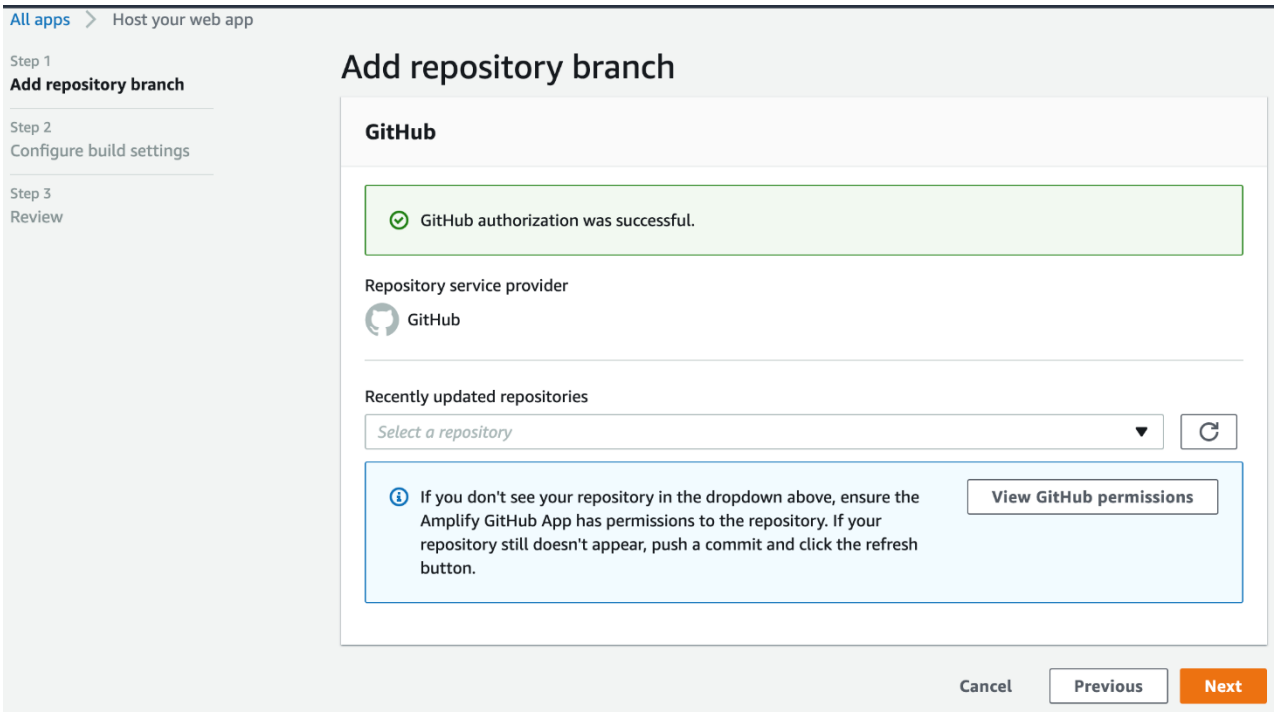


Figure 33: Locate to a repo branch

Step 2: Commit and push latest update to Github repository to build and deploy the frontend

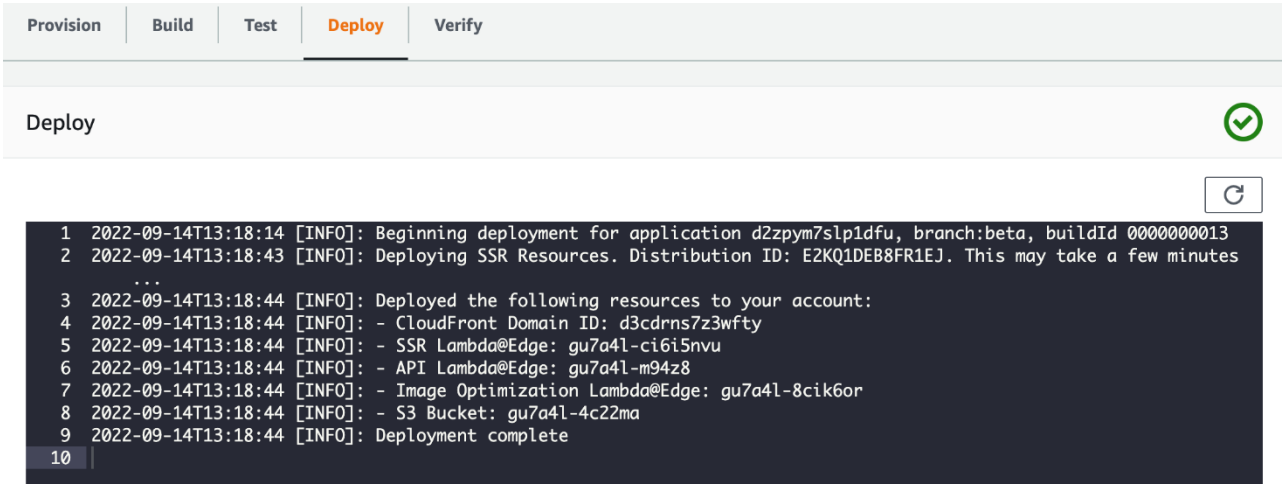


Figure 34: Deployment successfully with new resources: CloudFront, Lambda Edge, and S3 bucket

Step 3: Add Amplify backend environment to connect with Cognito and Kinesis Data Stream for Analytics

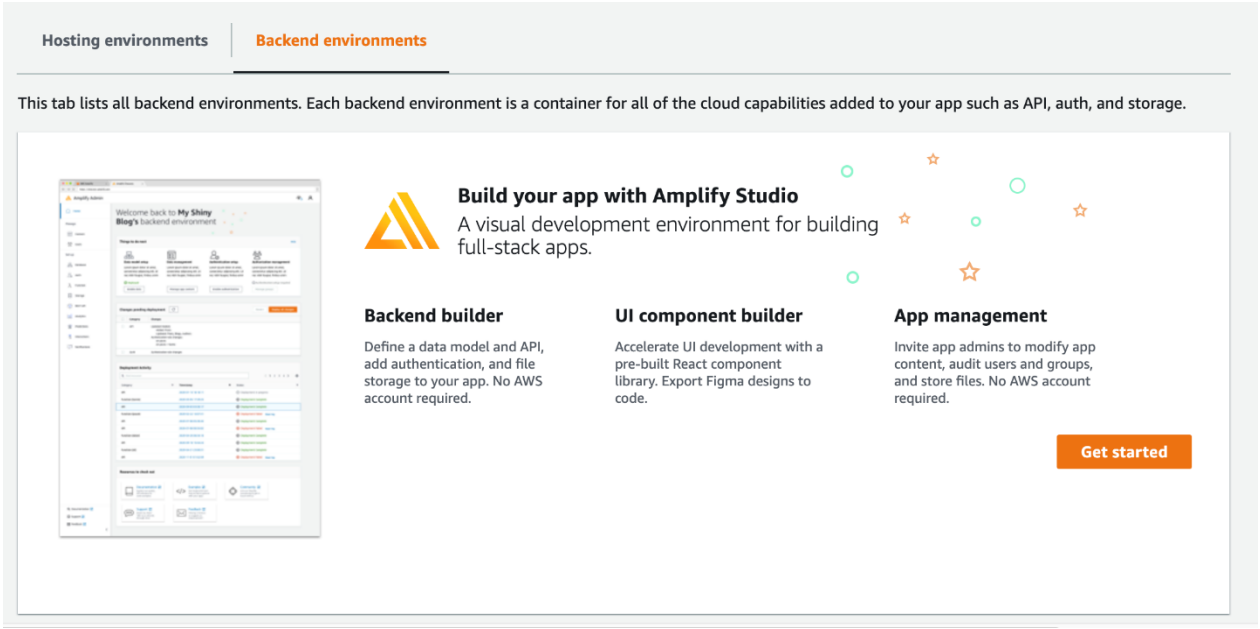


Figure 35: Connect Amplify to Cognito and Kinesis Data Stream

Step 4: Set up Amplify CLI on local machine using NPM

Figure 36: Install AWS Amplify CLI with npm

Step 5: Pull the Amplify backend to local machine

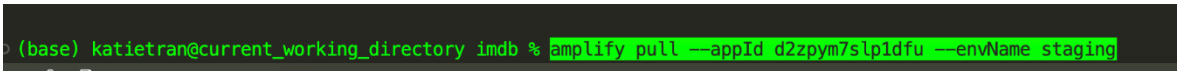


Figure 37: Pull the Amplify back-end to local

Set up Cognito on Amplify

Step 1: Add Cognito and config through Amplify CLI

```
(base) katietran@current_working_directory imdb % amplify add auth
Using service: Cognito, provided by: awscloudformation

 The current configured provider is Amazon Cognito.

 Do you want to use the default authentication and security configuration? Manual configuration
 Select the authentication/authorization services that you want to use: User Sign-Up, Sign-In, connected with AWS IAM controls (
 Provide a friendly name for your resource that will be used to label this category in the project: exsperPoolWithAnalytics
 Enter a name for your identity pool. exsperPoolWithAnalytics
 Allow unauthenticated logins? (Provides scoped down permissions that you can control via AWS IAM) Yes
 Do you want to enable 3rd party authentication providers in your identity pool? No
 Provide a name for your user pool: exsperPoolWithAnalytics
 Warning: you will not be able to edit these selections.
 How do you want users to be able to sign in? Email
 Do you want to add User Pool Groups? No
 Do you want to add an admin queries API? No
 Multifactor authentication (MFA) user login options: OFF
 Email based user registration/forgot password: Disabled (Uses SMS/TOTP as an alternative)
 Please specify an SMS verification message: Your verification code is {####}
 Do you want to override the default password policy for this User Pool? Yes
 Enter the minimum password length for this User Pool: 8
 Select the password character requirements for your userpool:
 Warning: you will not be able to edit these selections.
 What attributes are required for signing up? Email
 Specify the app's refresh token expiration period (in days): 30
 Do you want to specify the user attributes this app can read and write? Yes
 Specify read attributes: Address, Birthdate, Email, Family Name, Middle Name, Gender, Locale, Given Name, Name, Nickname, Phone
 Specify write attributes: Address, Birthdate, Family Name, Middle Name, Gender, Locale, Given Name, Name, Nickname, Phone Numbe
 Do you want to enable any of the following capabilities?
 Do you want to use an OAuth flow? No
? Do you want to configure Lambda Triggers for Cognito? Yes
? Which triggers do you want to enable for Cognito
✅ Successfully added auth resource exsperPoolWithAnalytics locally
```

```
(base) katietran@current_working_directory imdb % amplify push
✔ Successfully pulled backend environment staging from the cloud.

    Current Environment: staging

| Category | Resource name | Operation | Provider plugin |
| --- | --- | --- | --- |
| Auth | exsperPoolWithAnalytics | Create | awscloudformation |
| Auth | exsperimdb | Unlink | awscloudformation |
| Function | ExsperAmplifyPresignupAutoconfirm | No Change | awscloudformation |

? Are you sure you want to continue? Yes

Deployment completed.
Deployed root stack exsperimdb [ ====================================== ] 3/3
        amplify-amplifyd5bd157199814-… AWS::CloudFormation::Stack     UPDATE_COMPLETE              Thu Sep 15 2022 08:43:24…
        functionExsperAmplifyPresignu… AWS::CloudFormation::Stack     UPDATE_COMPLETE              Thu Sep 15 2022 08:39:32…
        authexsperPoolWithAnalytics    AWS::CloudFormation::Stack     CREATE_COMPLETE              Thu Sep 15 2022 08:43:04…
Deployed auth exsperPoolWithAnalytics [ ====================================== ] 11/11
        SNSRole                        AWS::IAM::Role                 CREATE_COMPLETE              Thu Sep 15 2022 08:40:14…
        UserPool                       AWS::Cognito::UserPool         CREATE_COMPLETE              Thu Sep 15 2022 08:40:19…
        UserPoolClient                 AWS::Cognito::UserPoolClient   CREATE_COMPLETE              Thu Sep 15 2022 08:40:24…
        UserPoolClientWeb              AWS::Cognito::UserPoolClient   CREATE_COMPLETE              Thu Sep 15 2022 08:40:24…
        UserPoolClientRole             AWS::IAM::Role                 CREATE_COMPLETE              Thu Sep 15 2022 08:41:02…
        UserPoolClientLambda           AWS::Lambda::Function          CREATE_COMPLETE              Thu Sep 15 2022 08:41:13…
        UserPoolClientLambdaPolicy     AWS::IAM::Policy               CREATE_COMPLETE              Thu Sep 15 2022 08:41:50…
        UserPoolClientLogPolicy        AWS::IAM::Policy               CREATE_COMPLETE              Thu Sep 15 2022 08:42:28…
        UserPoolClientInputs           Custom::LambdaCallout          CREATE_COMPLETE              Thu Sep 15 2022 08:42:34…
        IdentityPool                   AWS::Cognito::IdentityPool     CREATE_COMPLETE              Thu Sep 15 2022 08:42:40…
        IdentityPoolRoleMap            AWS::Cognito::IdentityPoolRol… CREATE_COMPLETE              Thu Sep 15 2022 08:42:44…
```

Figure 38: Configurate AWS Cognito using Amplify CLI

Step 2: Revise User Pool configurations on the Cognito Console in Amazon AWS Console

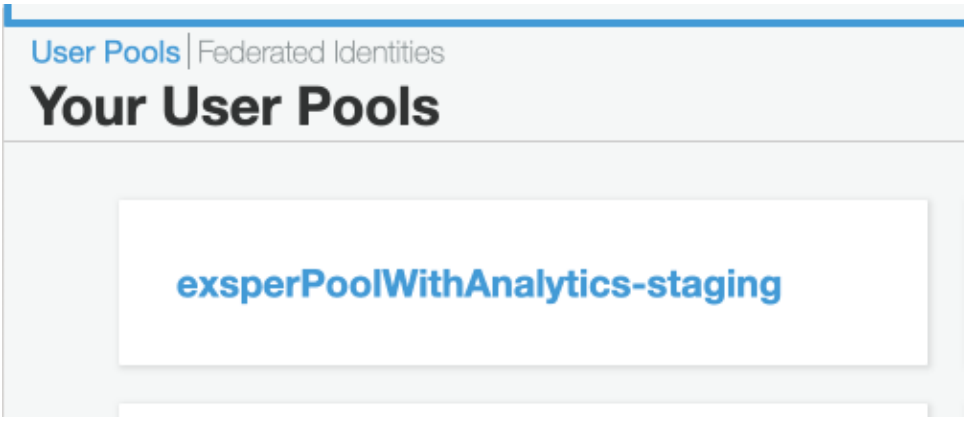


Figure 39: Create a new user pool

Step 3: Add custom attributes and turn off all verifications and unrelated services such as Email, SNS, SES

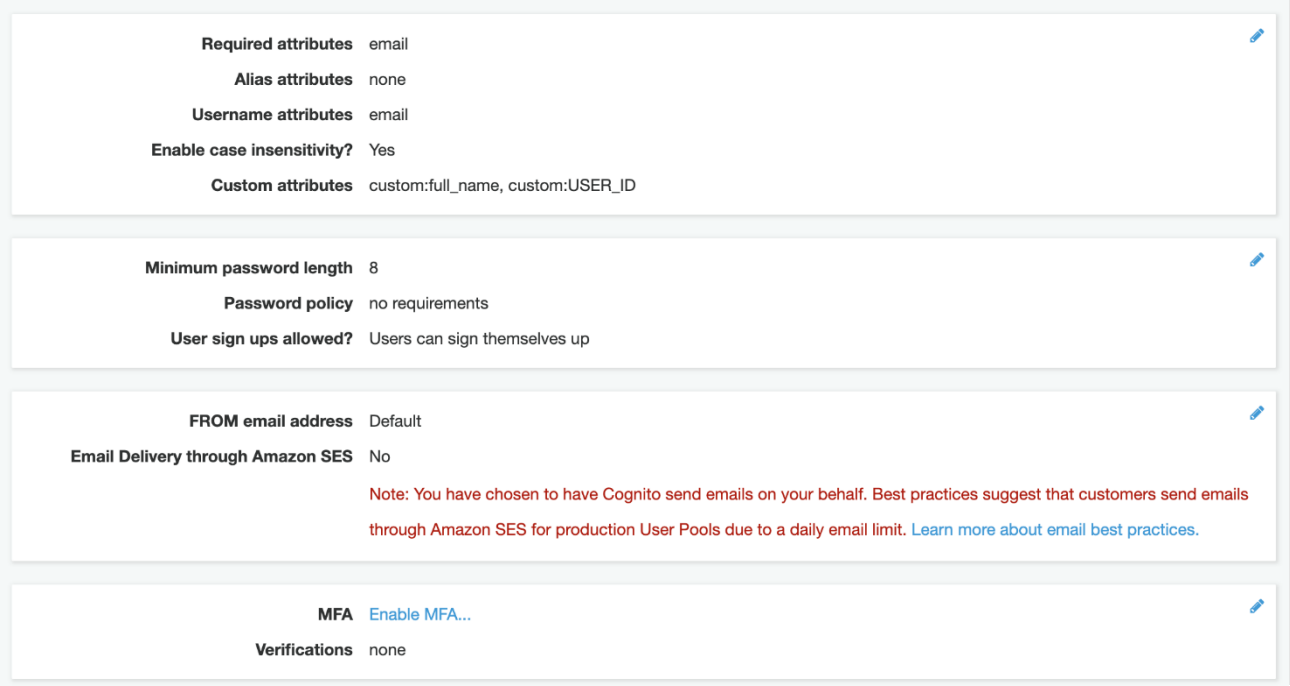


Figure 40: Configurate User Pools

Step 4: Add domain name

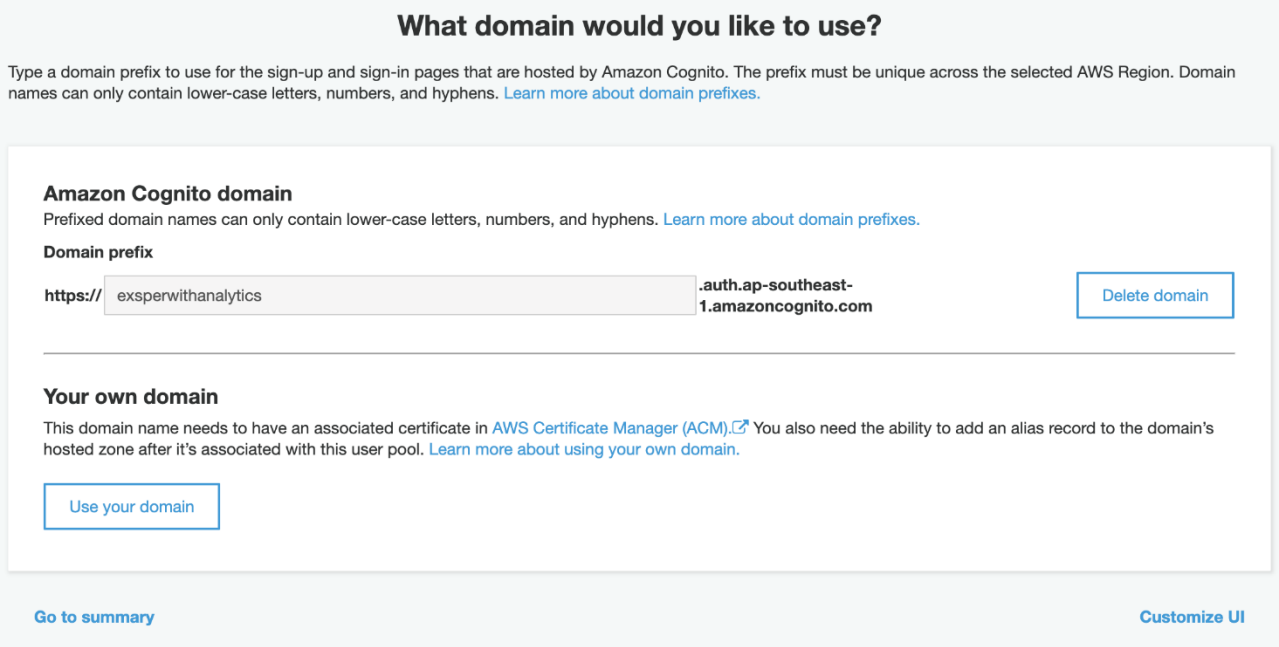


Figure 41: Create AWS Cognito domain

Step 5: Create a Lambda Function to automatically confirm user on signing up

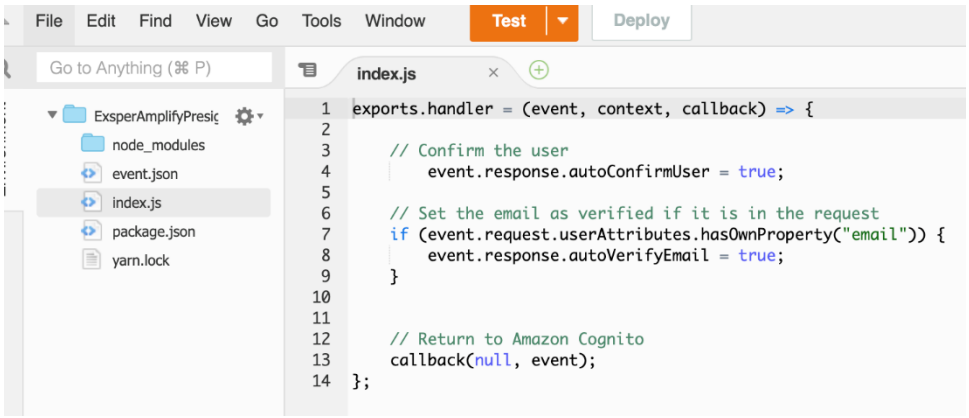


Figure 42: Create lambda function to verify user email on sign-up

Step 6: Add Cognito credentials to Amplify Environment variables

Variable	Value	Branch
AMPLIFY_NATIVECLIENT_ID	287fjl00jdtp2qk7henrkqiqv2	All branches
AMPLIFY_USERPOOL_ID	ap-southeast-1_1eyeYTptG	All branches
AMPLIFY_WEBCLIENT_ID	2grsb7tll46df8772dnrisbjbr	All branches
COGNITO_APPCLIENT_WEB	2grsb7tll46df8772dnrisbjbr	All branches
COGNITO_DOMAIN	exsperwithanalytics.auth.ap-southeast-1.amazoncognito.com	All branches
COGNITO_ID	287fjl00jdtp2qk7henrkqiqv2	All branches
COGNITO_POOL_ID	ap-southeast-1_1eyeYTptG	All branches
COGNITO_REGION	ap-southeast-1	All branches
COGNITO_SECRET	1kvjhdl6cmlq7tqglu5c05c1q63n3tpobeijb71p7s2mnei2t7m3	All branches

Figure 43: Attach Cognito to Amplify

Set up Amplify Analytics and Kinesis

Step 1: Create Kinesis Stream

To prepare for streaming data, we need to set up a

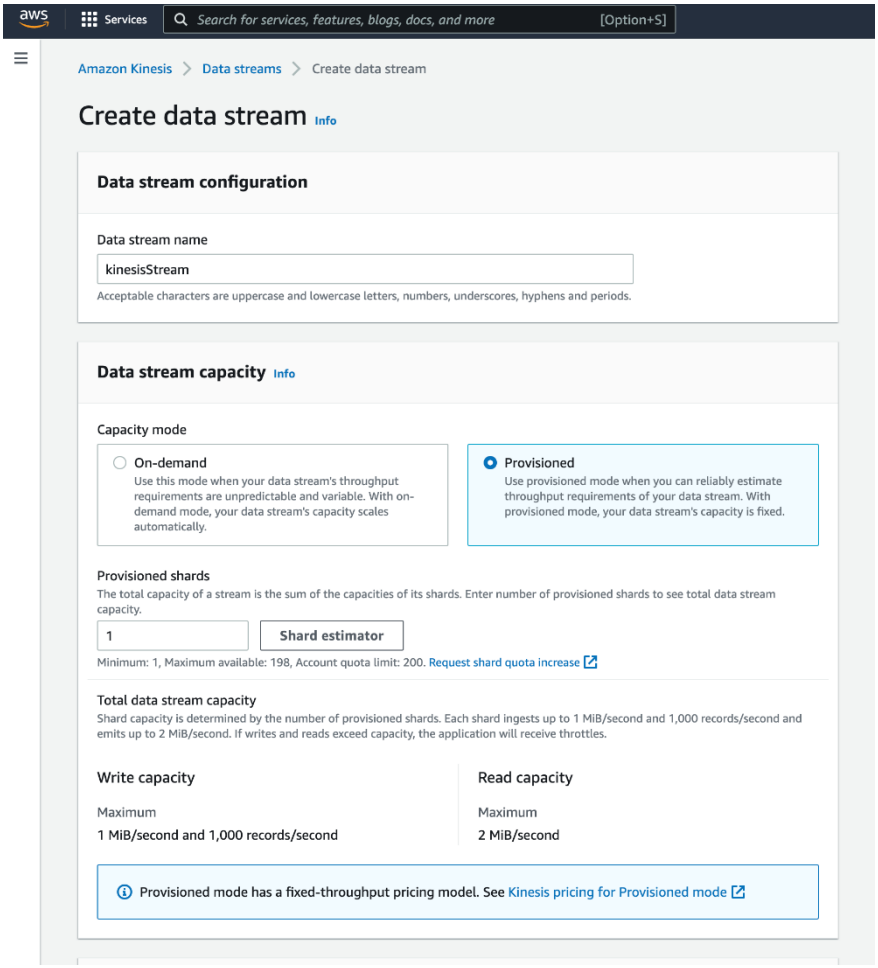


Figure 44: Create and Configurate data stream

Step 2: Add Kinesis Stream to Amplify using the Amplify CLI

```
(base) katietran@current_working_directory imdb % amplify add analytics
? Select an Analytics provider Amazon Kinesis Streams
? Enter a Stream name amplifyAnalyticsExsper
? Enter number of shards 1
Successfully added resource amplifyAnalyticsExsper locally

Some next steps:
"amplify push" builds all of your local backend resources and provisions them in the cloud
"amplify publish" builds all your local backend and front-end resources (if you have hosting category added) and provisions them in

(base) katietran@current_working_directory imdb % amplify push
✔ Successfully pulled backend environment staging from the cloud.

    Current Environment: staging

| Category | Resource name | Operation | Provider plugin |
| --- | --- | --- | --- |
| Analytics | amplifyAnalyticsExsper | Create | awscloudformation |
| Auth | exsperPoolWithAnalytics | No Change | awscloudformation |
| Function | ExsperAmplifyPresignupAutoconfirm | No Change | awscloudformation |

? Are you sure you want to continue? Yes

Deployment completed.
Deployed root stack exsperimdb [ ======================================== ] 4/4
        amplify-amplifyd5bd157199814-… AWS::CloudFormation::Stack     UPDATE_COMPLETE              Thu Sep 15 2022 09:10:30…
        analyticsamplifyAnalyticsExsp… AWS::CloudFormation::Stack     CREATE_COMPLETE              Thu Sep 15 2022 09:10:27…
        functionExsperAmplifyPresignu… AWS::CloudFormation::Stack     UPDATE_COMPLETE              Thu Sep 15 2022 09:09:26…
        authexsperPoolWithAnalytics    AWS::CloudFormation::Stack     UPDATE_COMPLETE              Thu Sep 15 2022 09:09:28…
Deployed analytics amplifyAnalyticsExsper [ ==================================== ] 3/3
        KinesisStream                  AWS::Kinesis::Stream           CREATE_COMPLETE              Thu Sep 15 2022 09:09:37…
        CognitoAuthPolicy              AWS::IAM::Policy               CREATE_COMPLETE              Thu Sep 15 2022 09:10:14…
        CognitoUnauthPolicy            AWS::IAM::Policy               CREATE_COMPLETE              Thu Sep 15 2022 09:10:22…
```

Figure 45: Configurate AWS Kinesis Data Stream using Amplify CLI

Step 3: Create S3 bucket to store data

To store event data after done processing for future reference, we create an s3 bucket using the following command

```
(base) nhatbuiminh@MBP-de-Nhat ~ % aws s3 mb s3://exper-movie-data
make_bucket: exper-movie-data
(base) nhatbuiminh@MBP-de-Nhat ~ %
```

Figure 46: Create S3 bucket for data storage

Step 4: Create Kinesis Firehose Delivery Stream to send data to S3

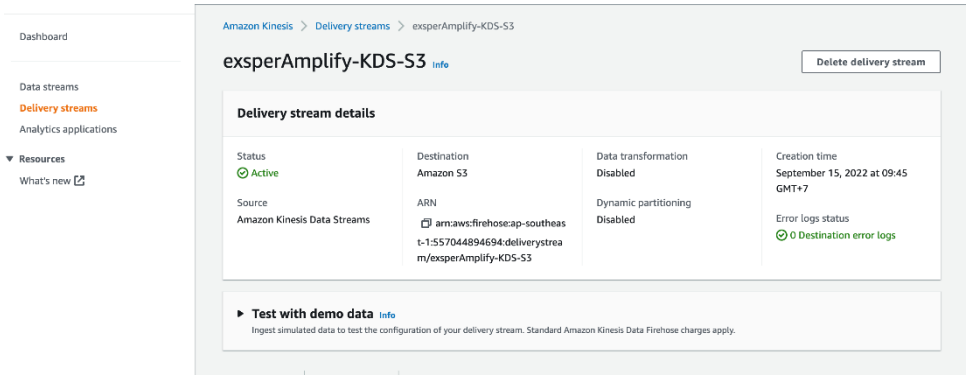


Figure 47: Configure Kinesis Firehose

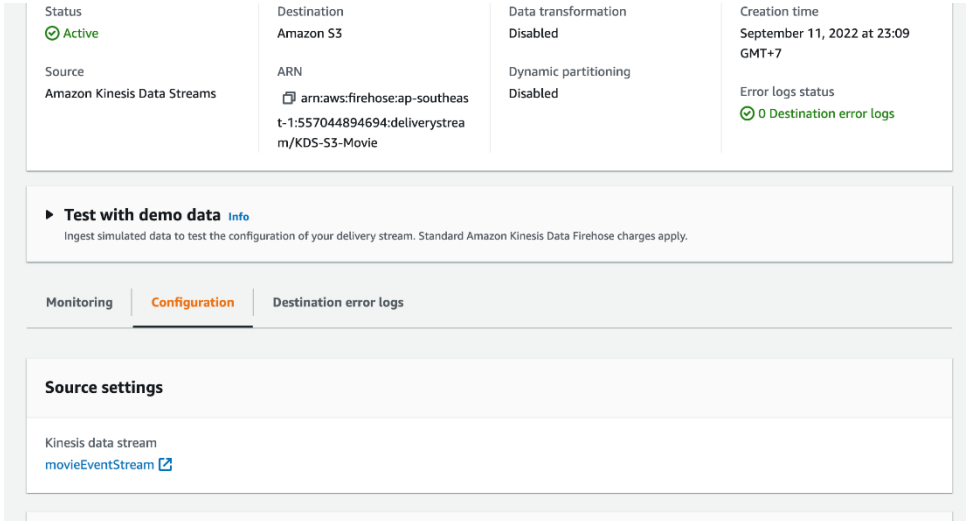


Figure 48: Select the source stream as the Kinesis

Step 5: Create a Lambda function to handle streaming events and connect with AWS Personalize

After setup the Kinesis and Kinesis Firehose, we create a streaming function for processing click event data.

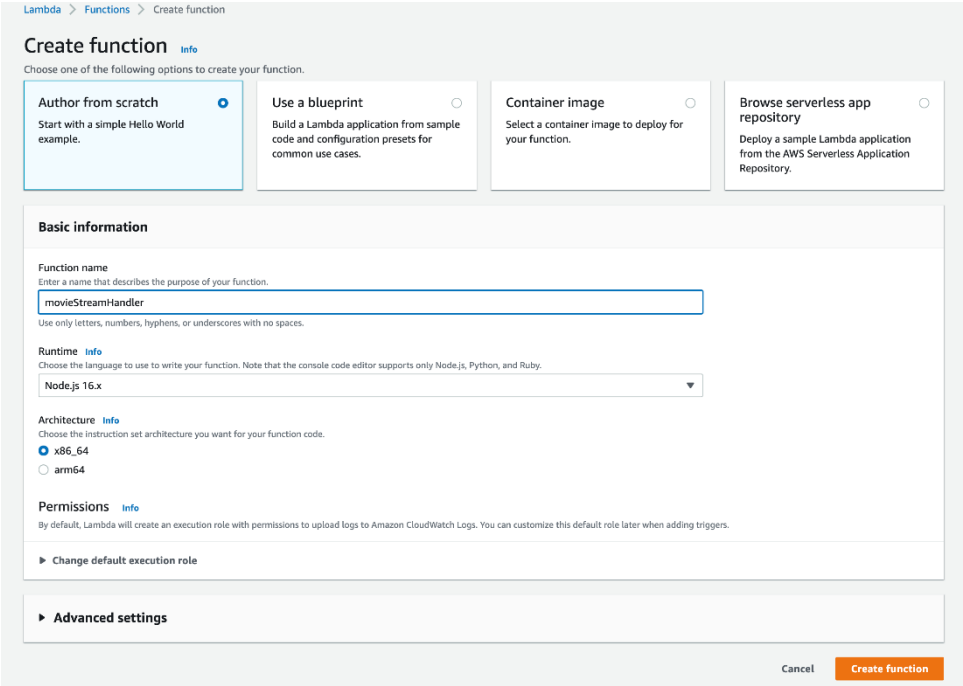


Figure 49: Create lambda function to handle events

Deploy the following code to lambda:

The code part contains necessary functions to received payload from Kinesis and process data for AWS Personalize

```
AWS.config.update({
    region:'ap-southeast-1',
    accessKeyId: process.env.ACCESS_KEY_ID,
    secretAccessKey: process.env.SECRET_ACCESS_KEY
});

var personalizeevents = new AWS.PersonalizeEvents();
// var dynamoClient = new AWS.DynamoDB.DocumentClient();

console.log('Loading function');

exports.handler = (event, context, callback) => {
    console.log(JSON.stringify(event, null, 2));

    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = Buffer.from(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
        payload = JSON.parse(payload);
        var eventDate = new Date();
        var putEventsParams= {
            'sessionId': payload.SessionId, /* required */
            'trackingId': process.env.TRACKING_ID, /* required */
            'userId': payload.UserId,
            eventList: [
                {
                  'eventType': payload.EventType, /* required */
                  'itemId': payload.ItemId, /* required */
                  'sentAt': eventDate
                },
            ]
        }
        console.log("THIS IS THE OBJECT = " + JSON.stringify(putEventsParams,null,3))
        //Create a personalize event
        personalizeevents.putEvents(putEventsParams, function (err, data) {
          if (err) {
                console.log(err, err.stack); // an error occurred
          }
          else{
                console.log(data);           // successful response
                putEventsParams['eventList'][0]['sentAt']=putEventsParams['eventList'][0]['sentAt'].toTimeString();
                const putEventsErrResponse = {
                    statusCode: 500,
                    body: JSON.stringify(err),
                };
                callback(null, putEventsErrResponse);
                const response = {
                    statusCode: 200,
                    body: JSON.stringify(putEventsParams),
                };
                callback(null, response);
          }
        });
    });
};
```

Figure 50: Code deployment on Lambda function

Step 6: Set up environment variable

The following are the necessary variable for the lambda function to process the incoming streaming data

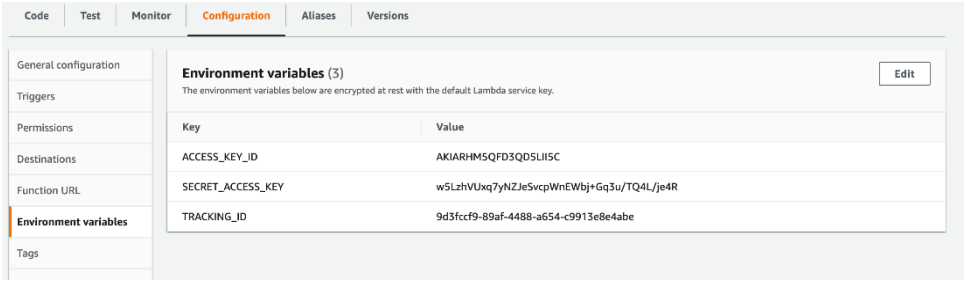


Figure 51: Create environment variables

Step 7: Config Lambda IAM Role permission to add trigger

In order for the lambda function to work properly, we have to assign IAM Role to allow it to access to S3, Kinesis and Firehose.

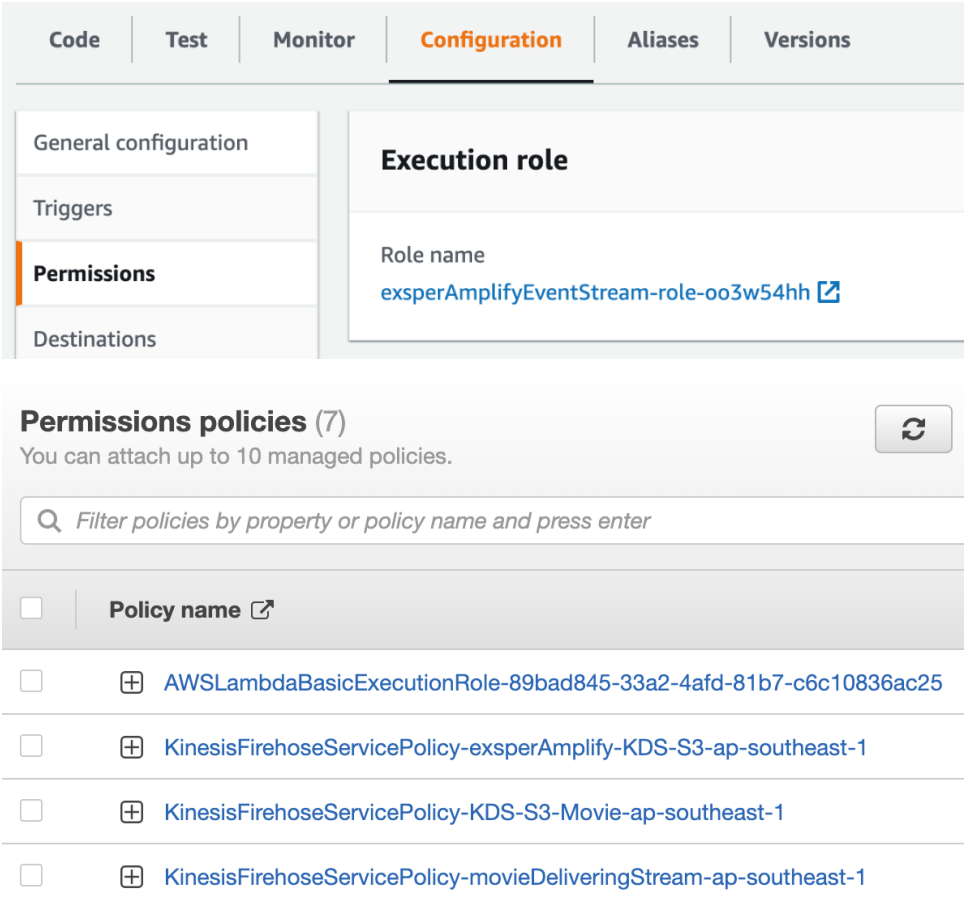


Figure 52: Create policy permissions for Lambda to access S3, Kinesis and Firehose

Step 8: Add Kinesis trigger to Lambda function

The function is trigger when the click event is broadcasted to Kinesis

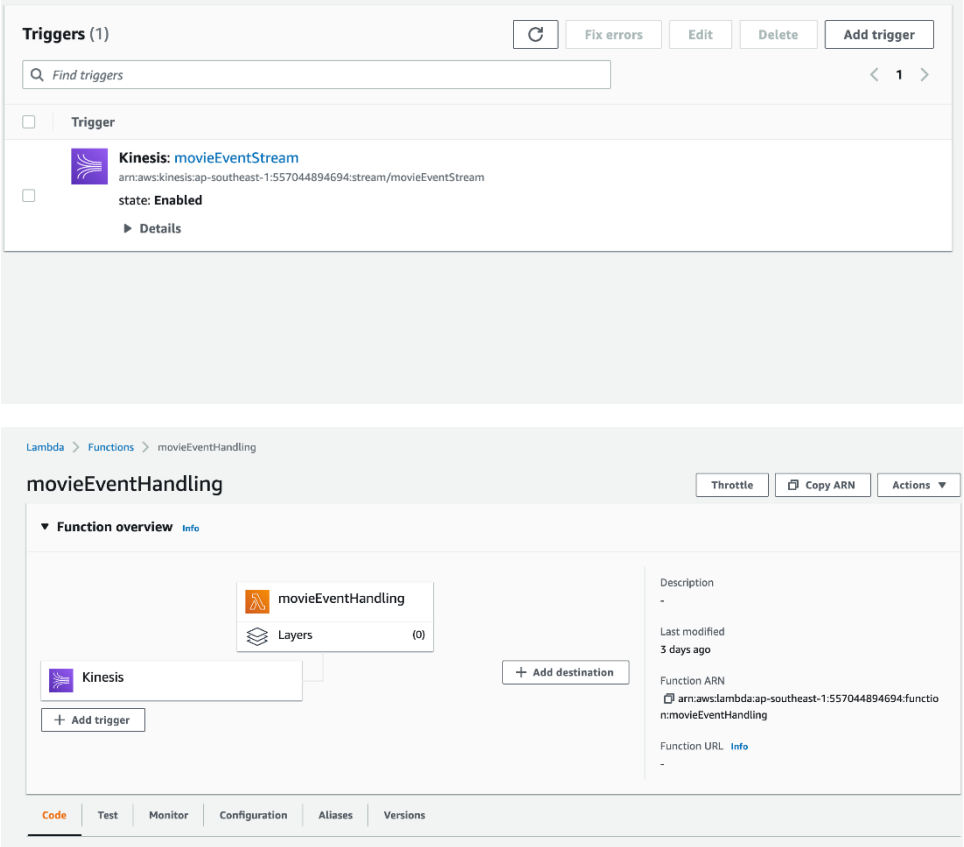


Figure 53: Add Kinesis as a trigger for Lambda function

Step 9 : Record the click stream event from the front end with Amplify SDK

After done setting up the server side, we set up built in AWS Amplify analytics event to record the click event and transmitted it to Kinesis

```
useEffect(() => {
    const runAnalytics = async () => {
        const currentUser = await Auth.currentAuthenticatedUser();
        if (currentUser) {
            Analytics.record(
                {
                    data: {
                        EventType: "clicked",
                        UserId: currentUser.attributes["custom:USER_ID"],
                        SessionId: "",
                        ItemId: movieID,
                    },
                    streamName: "amplifyAnalyticsExsper-staging",
                },
                "AWSKinesis"
            );
        }

    }
    runAnalytics()
}, [])
```

Figure 54: Collect click stream events using Amplify SDK

Step 10: Monitor events on Cloudwatch

After setting up properly, the data payload is streamed using Kinesis and process. The payload data is written in the log for verification purposes.

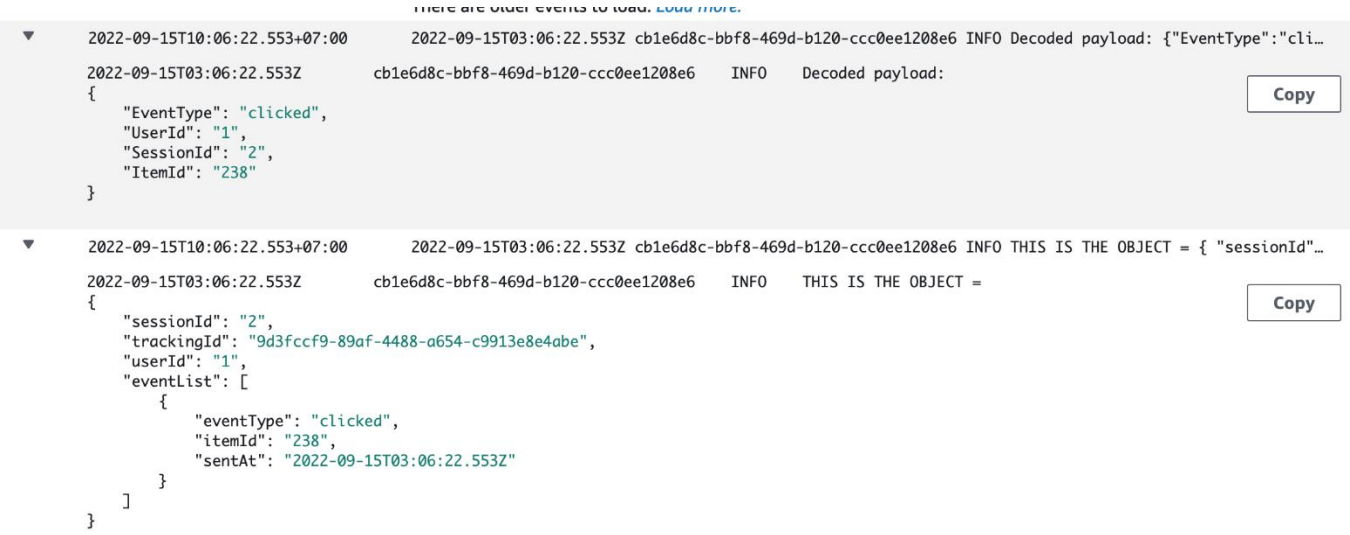


Figure 55: Observe click stream event on AWS Cloud Watch

VIII. A small user manual

**Note: Before experiencing the app, user must follow on the link in the section 2.*

1. Homepage

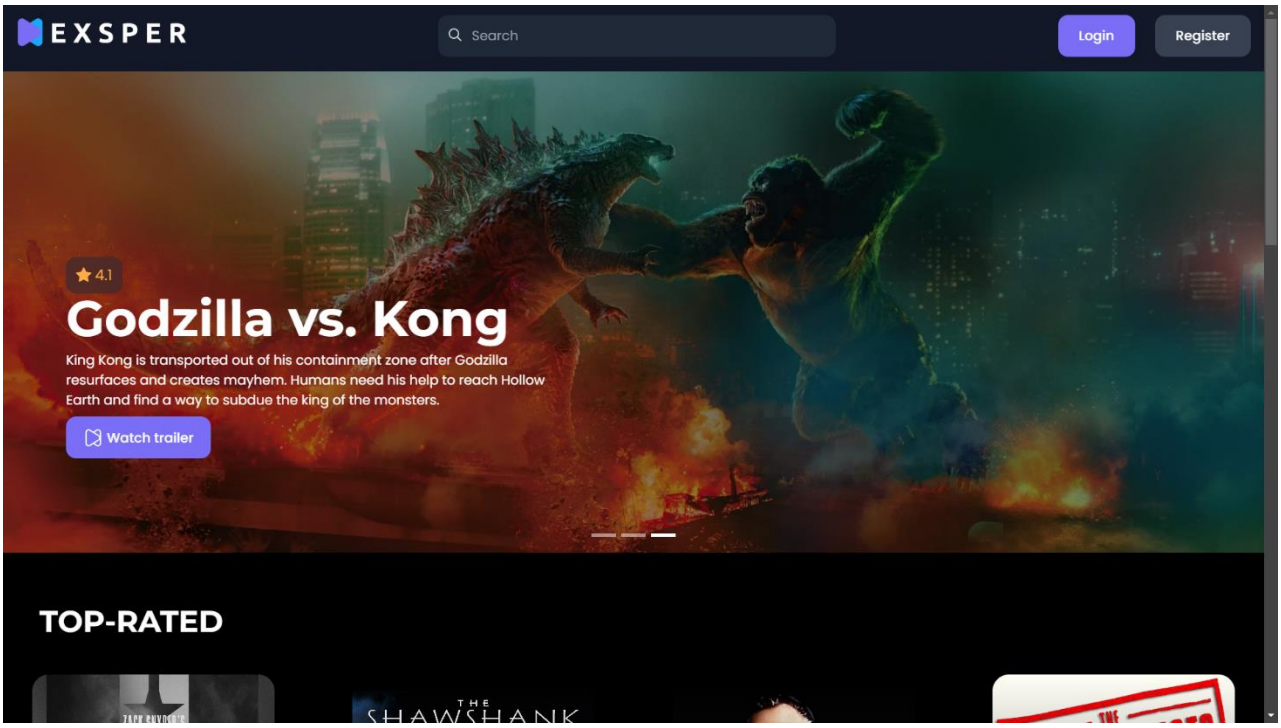


Figure 56: Exsper's Homepage

After being directed by AWS Amplify, the user will be on the homepage spotlighted by top trend movies. These such movies are briefly described by their names, rating stars, sub-headings, and a trailer, embedded with a corresponding YouTube link. The homepage also provides the basic features including the search bar, log-in option for existing users, and sign-up option for new ones.

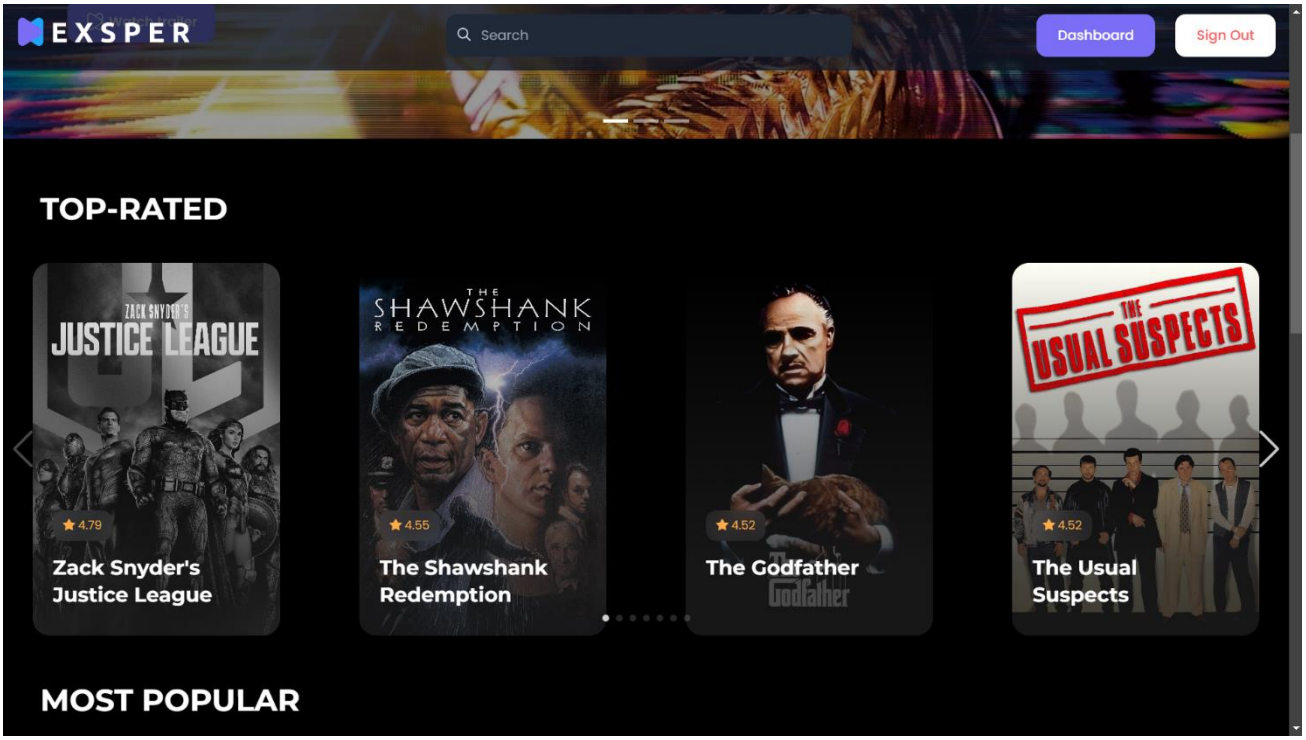


Figure 57: List of top-rated films

Scrolling down the homepage a bit, the application offers a list of movies that have a lot of favorable reviews from both experts and viewers. We design these movie items as a card list using image and text, and even icons to rapidly grab visitors' attention and make it simple for them to move between various films.

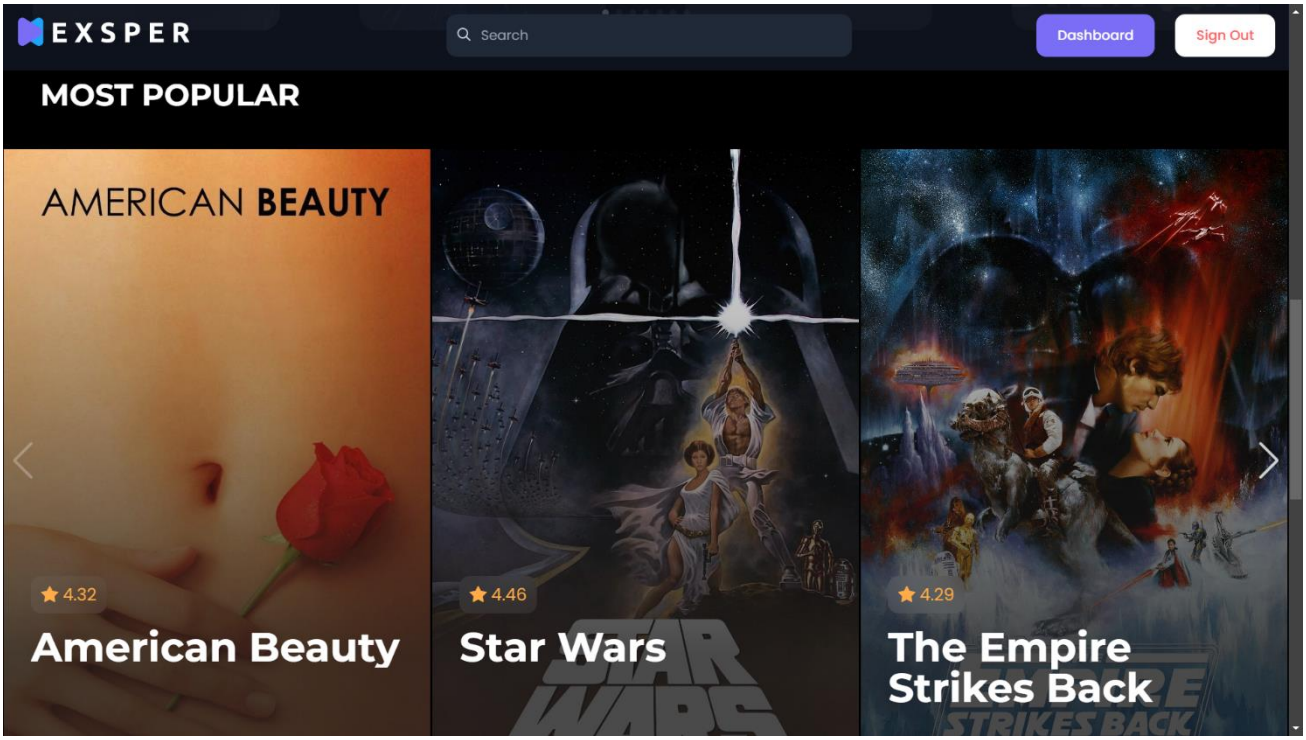


Figure 58: List of highest views movies

At the end of the homepage, there is the list of the series that attracted the most views all the time. The same design was chosen in this movie list because we expected it to be effective in terms of visual appeal.

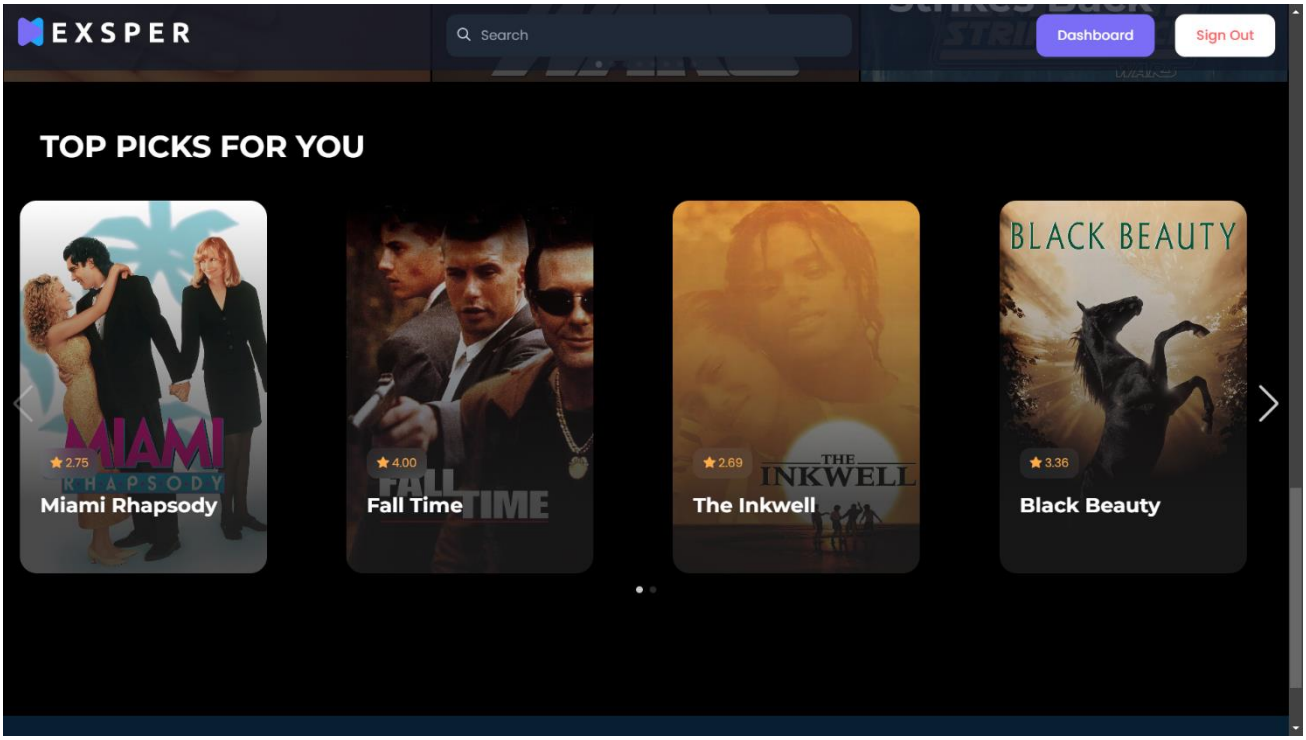


Figure 59: List of suggested movies

2. Log-in page

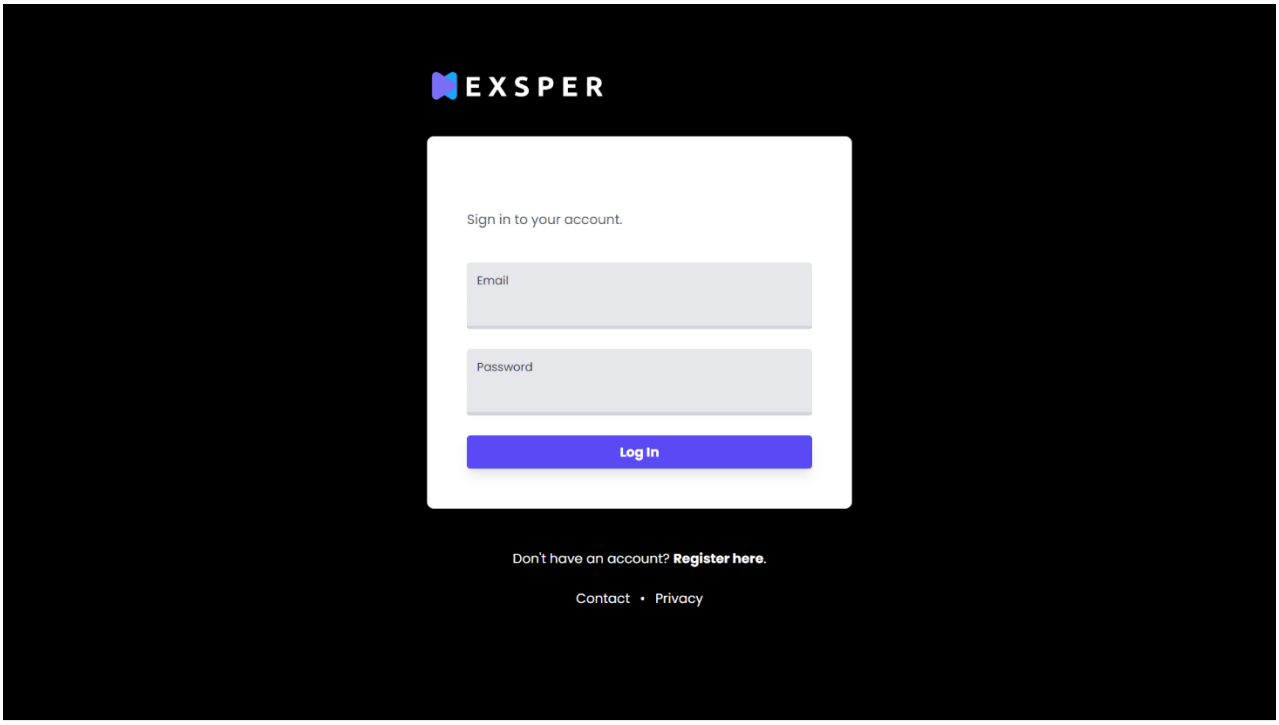


Figure 60: Login page after pressing Login button at the homepage

By clicking on the sign-in button, the system displays a login form containing two fields (Email and Password) then the user has to fill these with their account, which is provided or created beforehand, to jump into the app.

3. Sign-up page

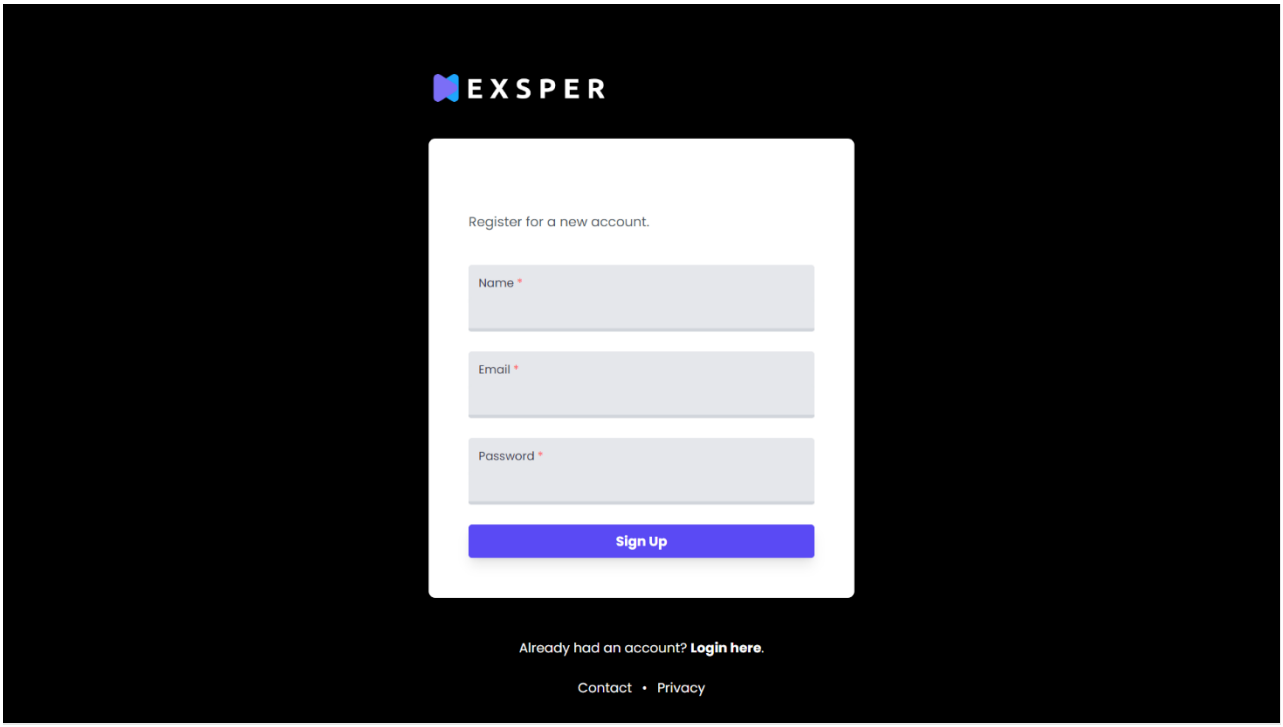
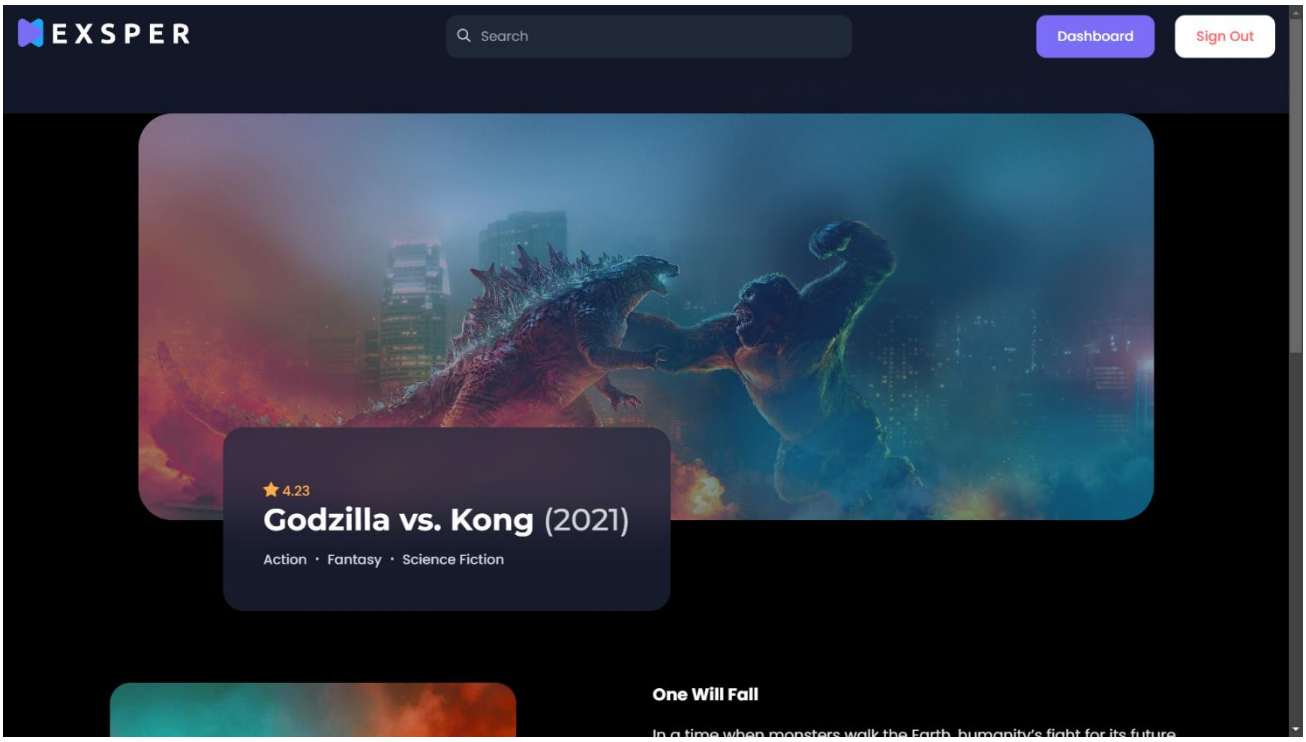


Figure 61: Signup page after pressing Register button from the homepage or login form

In case user is new to the app, there is a sign-up button in the homepage, or they can refer to the keyword **Register here**, located in log-in form, to get into the sign-up form. To pass this step, user must provide the required information (Name, Age, Email, Password, Confirmed Password) in order to get valid credentials after all.

4. Movie detail page



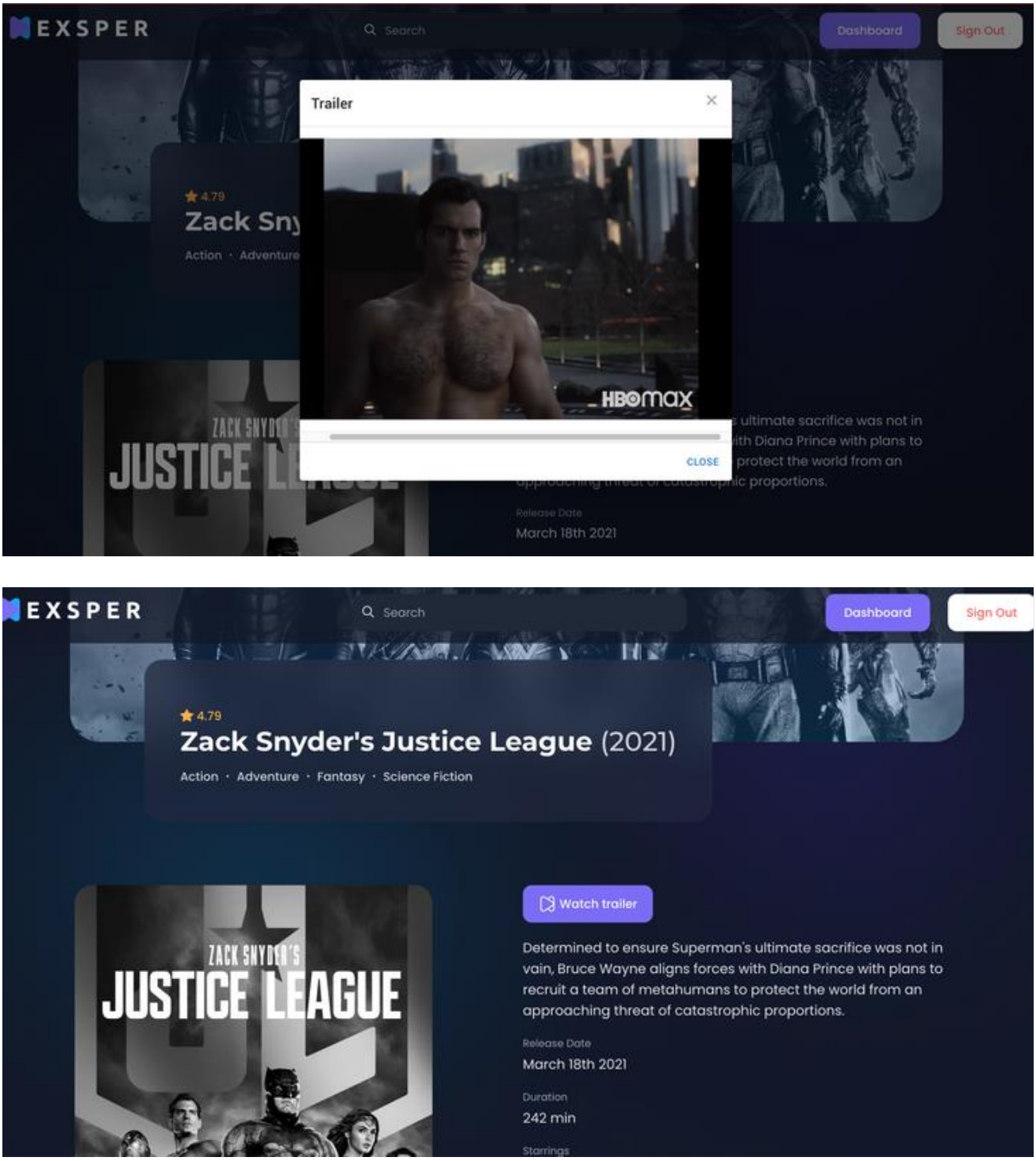
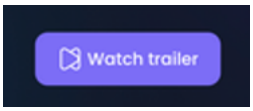


Figure 62: Movie details

When users click one of the spotlighted films on the homepage, the app will drive them to the expansive page containing more details about that movie. Some of the details could be shown up such as Description, Release Date, Duration, Director,

Writers, and a “Rate this movie” module where user can rate the film based upon their preference.

User can also view the trailer of that movie by clicking on the “Watch trailer” button



5. Search movie

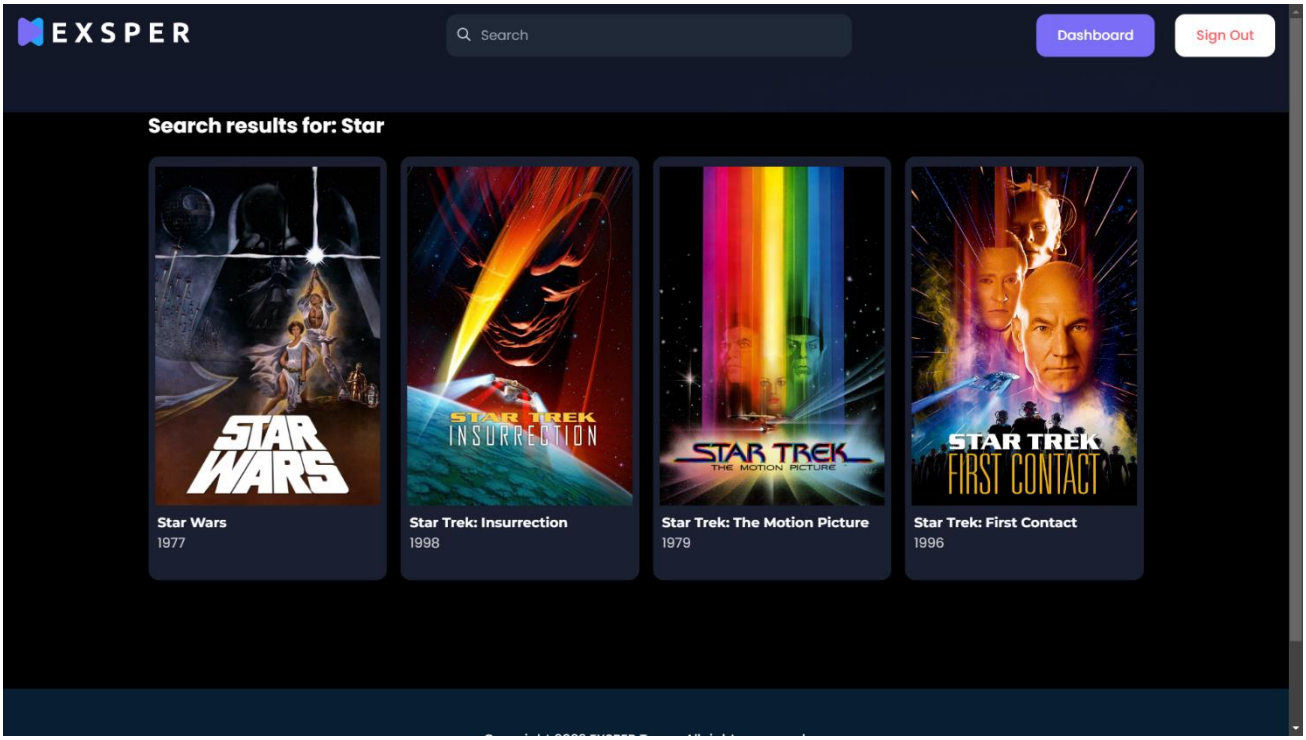


Figure 63: A returned list of film that include the searching keyword

The app provides a searching feature as an efficient way to find the specific film by looking up its name. It is notable that the search operation will search by word, meanings the searched result may return a list of movies whose title includes the searching keyword.

6. User Dashboard

In the user dashboard page, user can view their personal information such as Name, Age, Gender, and Email address. When scrolling down, users can see their own history of recently viewed items and recently rated items.

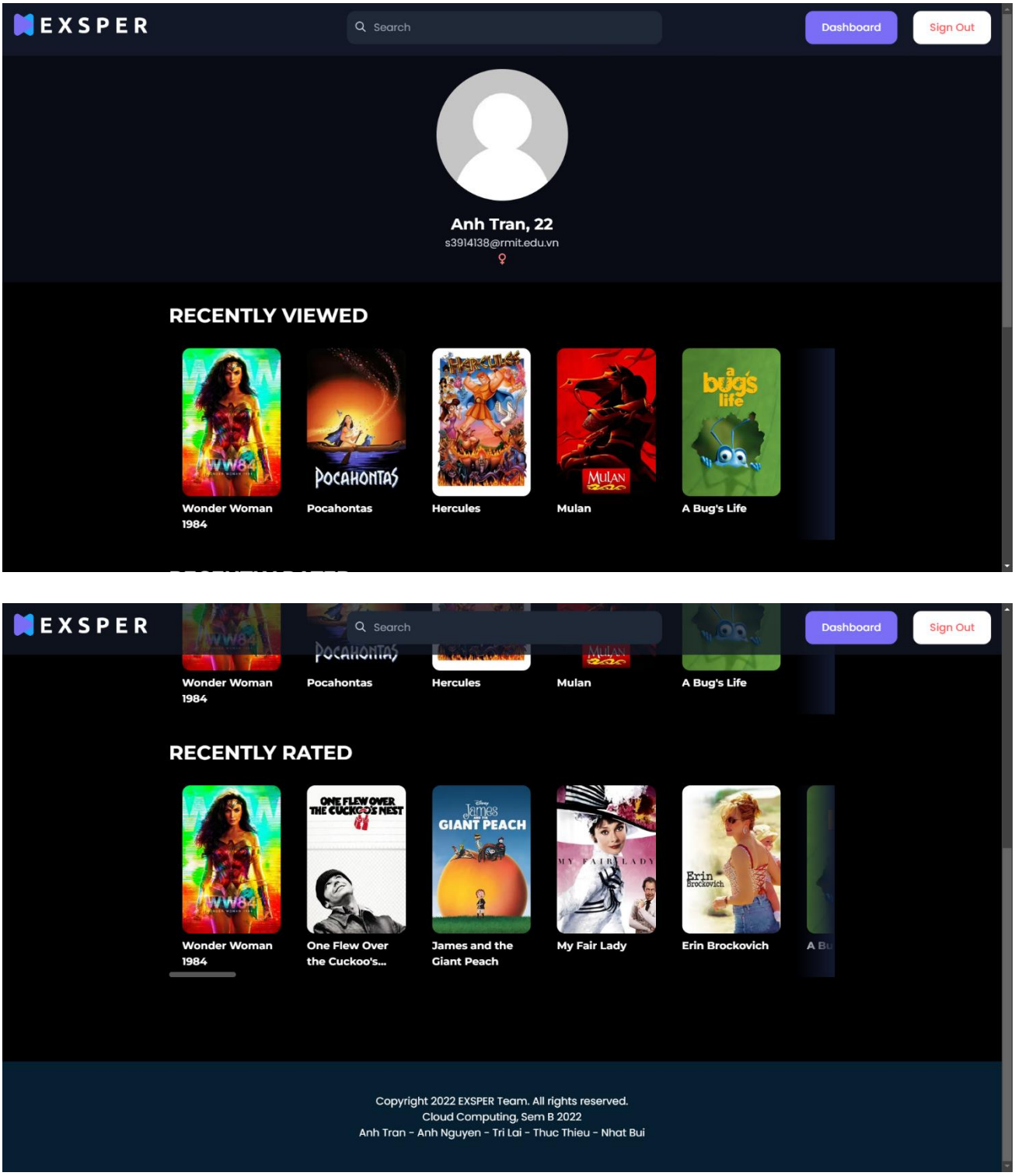


Figure 64: Dashboard contains list of recent viewed and recent rated movies

IX. References

[1] F. Harper and J. Konstan, "MovieLens 25M Dataset", *GroupLens*, 2015. [Online]. Available: https://grouplens.org/datasets/movielens/25m/. [Accessed: 28- Aug- 2022].

[2] "amazon-personalize-samples/next_steps/operations/streaming_events at master · aws-samples/amazon-personalize-samples", *GitHub*, 2022. [Online]. Available: https://github.com/aws-samples/amazon-personalize-samples/tree/master/next_steps/operations/streaming_events. [Accessed: 03- Sep- 2022].

[3] N. Thanh, "AWS DynamoDB", *Google Slides*, 2022. [Online]. Available: https://docs.google.com/presentation/d/1fDtnGRdunp1hvhLiM1FZdNy54Ved2-IH/edit?pli=1#slide=id.p1. [Accessed: 16- Aug- 2022].

[4] N. Thanh, "AWS Lambda", *Google Slides*, 2022. [Online]. Available: https://docs.google.com/presentation/d/12sTN8_4A0DPmnA1BYooOG-CMiyA4_51S/edit#slide=id.p1. [Accessed: 18- Aug- 2022].

[5] N. Thanh, "REST API", *Google Slides*, 2022. [Online]. Available: https://docs.google.com/presentation/d/1rA9-j5oCGn9GfFoE8IuxvV-uMWU41vRz/edit#slide=id.p1. [Accessed: 25- Aug- 2022].

X. Video

https://www.youtube.com/watch?v=bddrYQZe-Mk

XI. Appendices

1. Sample query user result on Thunder Client

Domain**:** **https://z1hi6ssas5.execute-api.ap-southeast-1.amazonaws.com**

1.1 Get recommended videos for a given user

GET: {domain}/getPersonalize?userId={userId}&numResults={number of recommendations}

Response: list of tmdbId (string) of recommended videos

Example

Request: https://z1hi6ssas5.execute-api.ap-southeast-1.amazonaws.com/getPersonalize?userId=15&numResults=5

Response: ["71754", “39953”, “9614”, “48787”, “12158”]

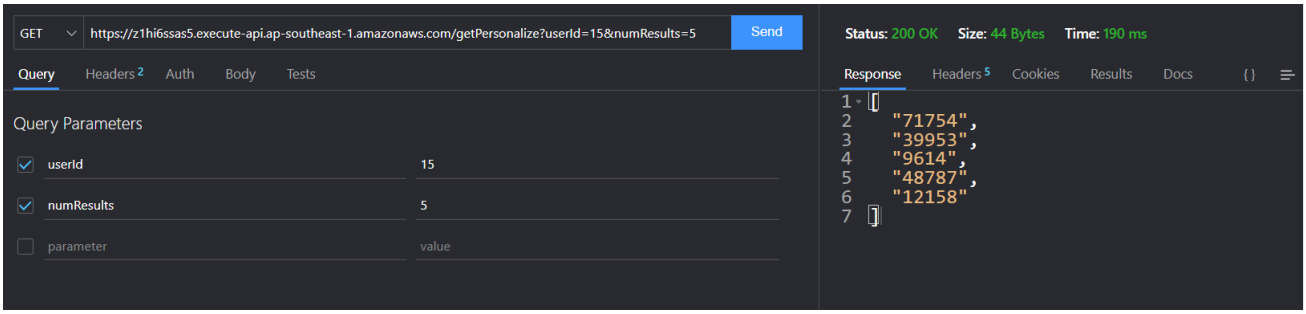


Figure 65: Top-picks for a certain user

1.2 Get similar videos to the target video

GET: {domain}/ getSims?itemId={itemId}&numResults={number of recommendations}

Response: list of tmdbId (string) of similar videos

Example

Request: https://z1hi6ssas5.execute-api.ap-southeast-1.amazonaws.com/getSims?itemId=862&numResults=10

Response: ["16934", "9073", "34444", "9623", "9073", "78802", "18256", "688", "8987", "4584"]

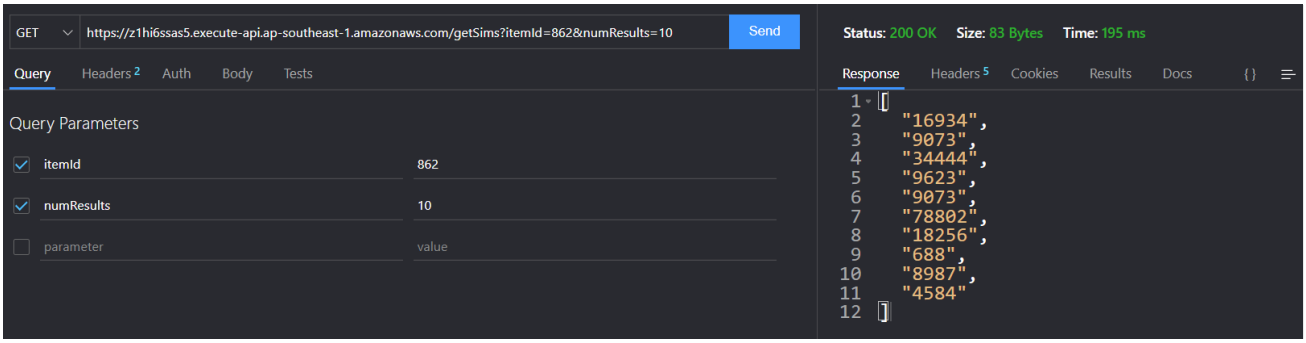


Figure 66: List of same genre films for an anonymous user

1.3 Get recommended videos given the target video and current user

GET: {domain}/ getReranking?

userId={userId}&itemId={itemId}&numResults={number of recommendations}

Response: list of tmdbId (string) of recommended videos

Example

Request: https://z1hi6ssas5.execute-api.ap-southeast-1.amazonaws.com/getReranking?userId=2&itemId=91&numResults=5

Response: ["11862", "11443", "2625", "25066", "2086"]

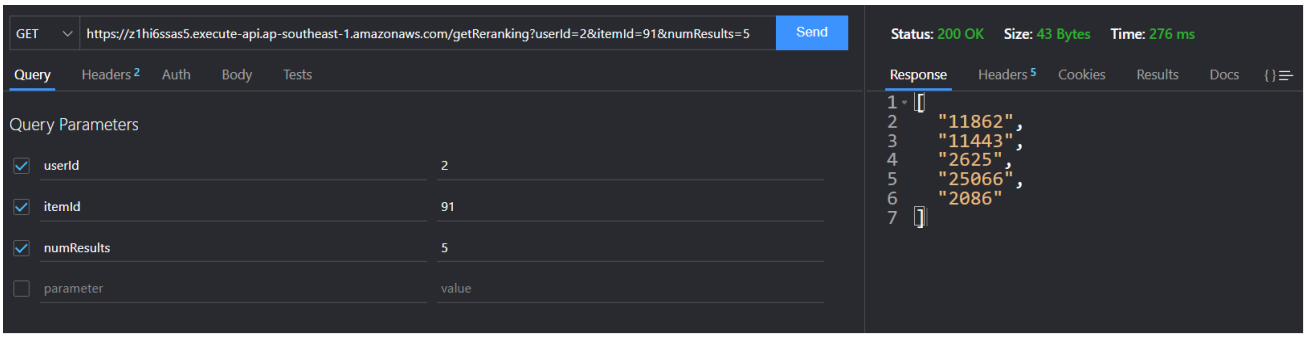


Figure 67: List of same genre films for an authenticated user

2. Sample query movie result on Postman

1. List 10 most-viewed movies

URL: Invoke URL + /movies/most-viewed

Method: GET

Sample result:

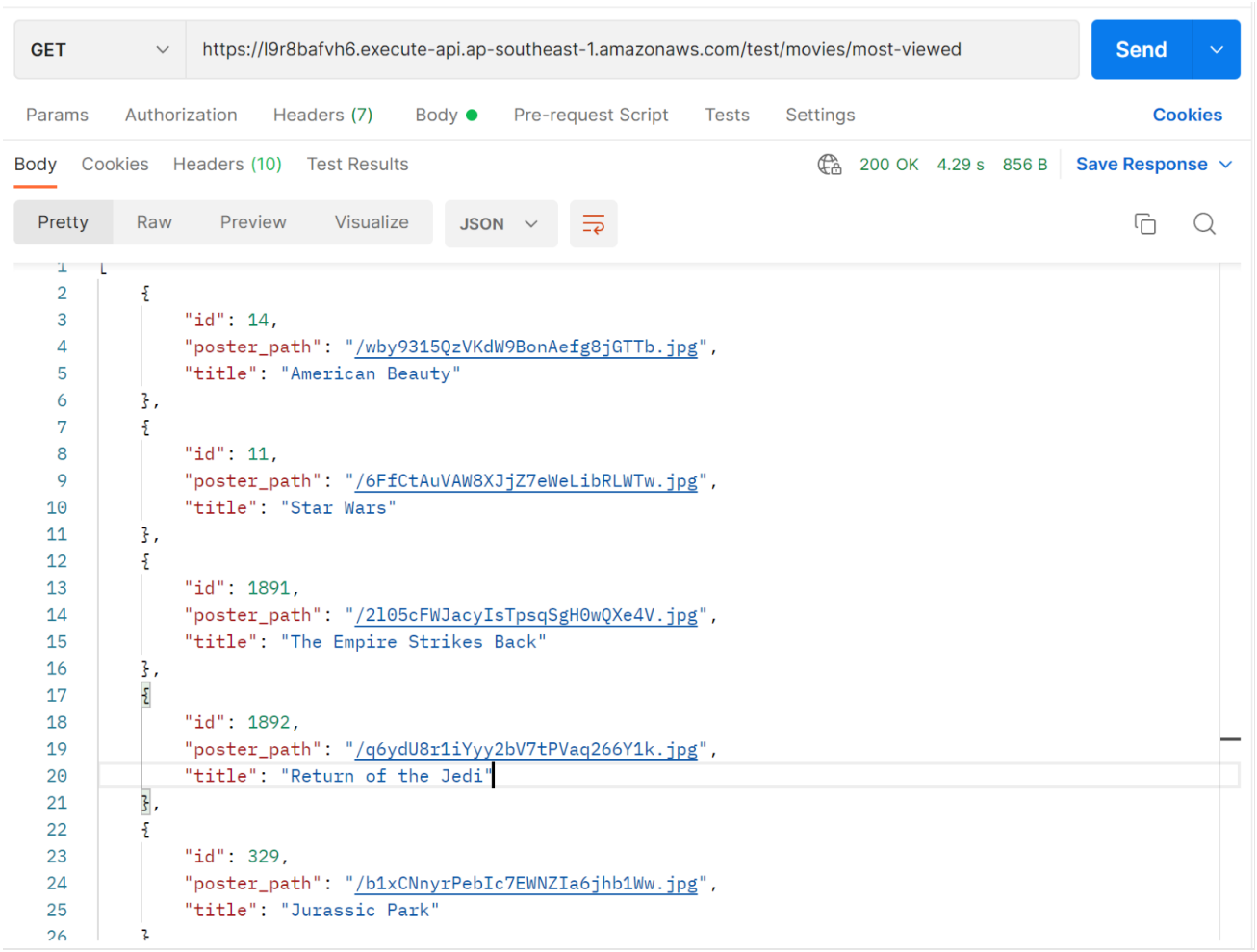


Figure 68: 10 most view films returned in JSON format

2. List 10 highest-rated movies

URL: Invoke URL + /movies/highest-rating

Method: GET

Sample result:

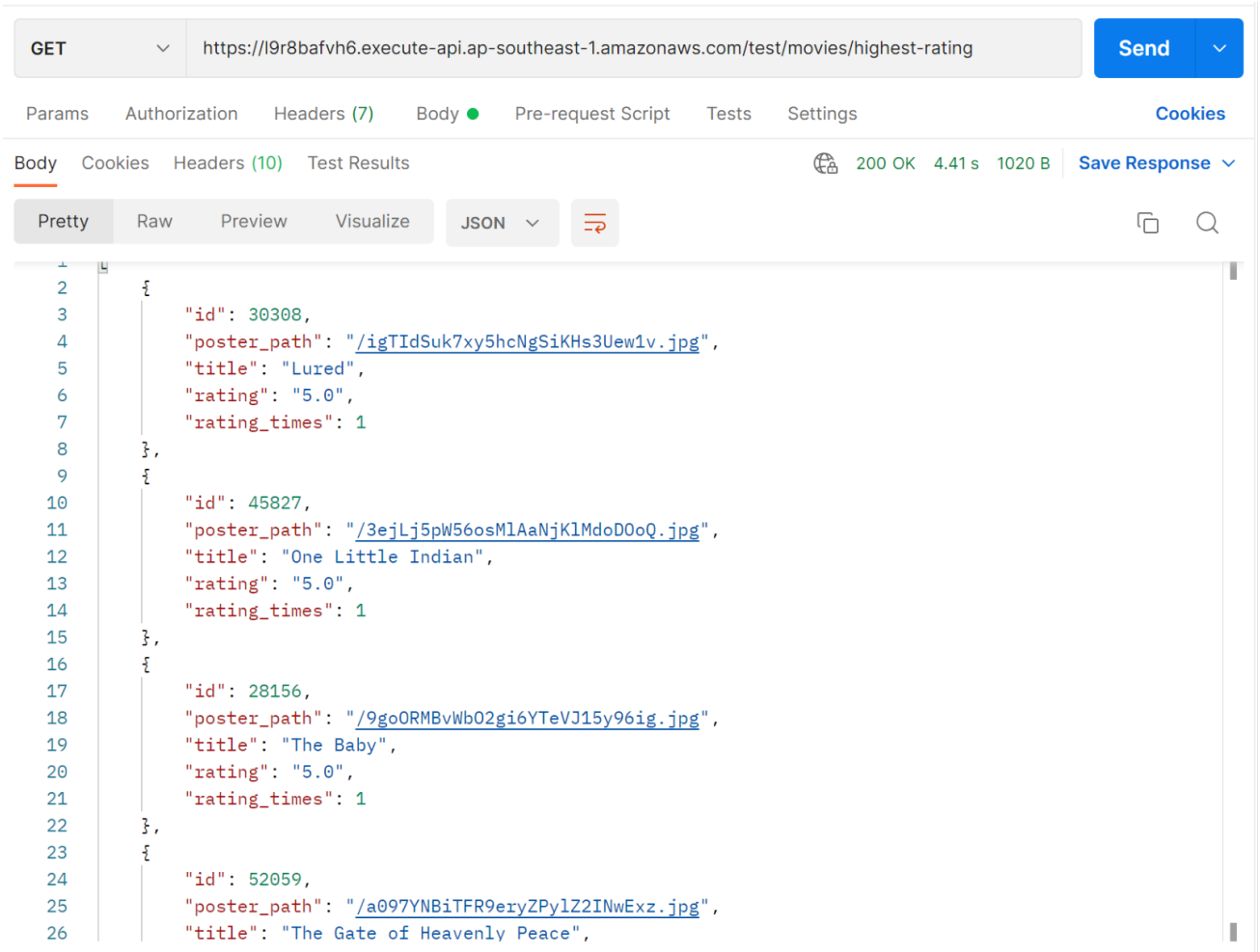


Figure 69: 10 highest rated films returned in JSON format

3. Search movies by title

URL: Invoke URL + /movies/search/{title}

Method: GET

Sample result:

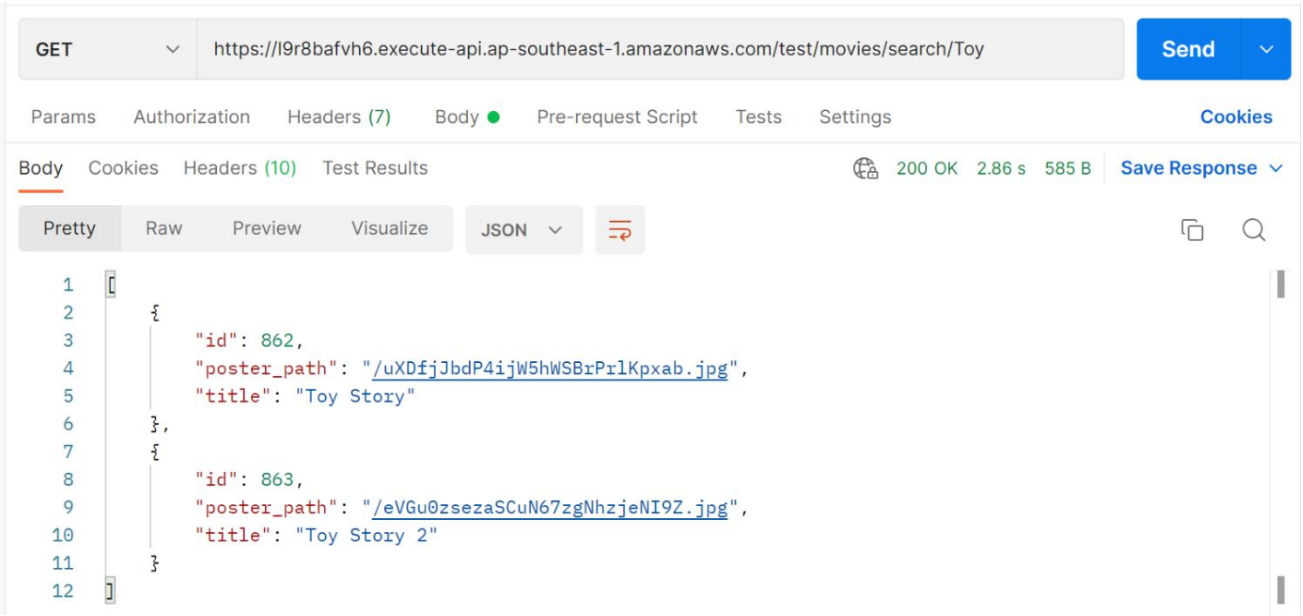


Figure 70: A list of movie contains search keyword returned in JSON format

4. Get movie information

URL: Invoke URL + /movie/{id}

Method: GET

Sample result:

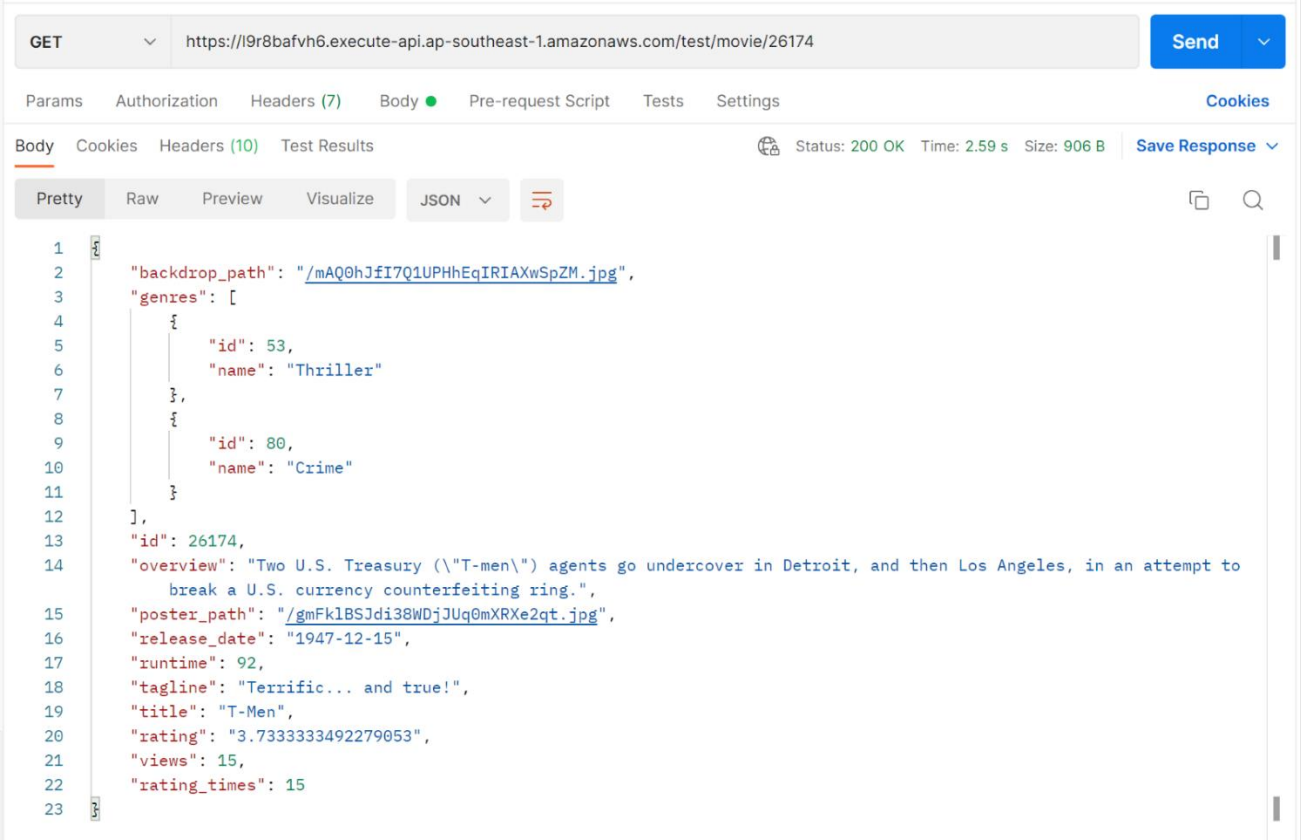


Figure 71: Movie details returned after sending a request

After calling:

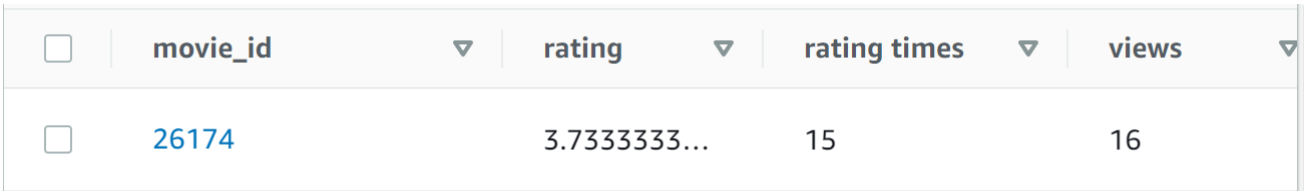


Figure 72: Increment number of view after sending a request

5. Rate movies

URL: Invoke URL + /movies/rate

Method: POST

Request body sample:

```
{
    "movie_id": "49299",
    "user_id": "1",
    "rating": 4
}
```

Sample result:

- Before call:

movie_id	rating	rating times	views
49299	4.5	2	2

Figure 73: Current movie rate number before rating

- Calling:

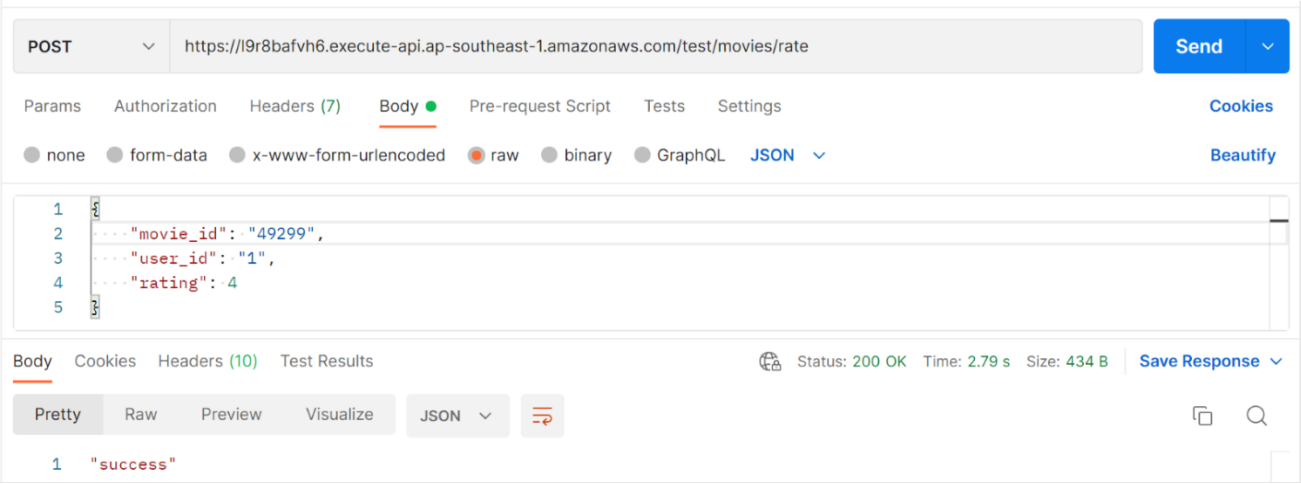


Figure 74: A request send with the body to rate the movie

- After call:

movie_id	rating	rating times	views
49299	4.3333333...	3	2

id	movie_id	rating	timestamp	user_id
2	49299	4	166211969...	1

Figure 75: Current movie rate number after rating

3. Sample query users result on Postman

3.1 Get all information of an user on data base by id

URL: Invoke URL + /users/{id}

Method: GET

Sample result:

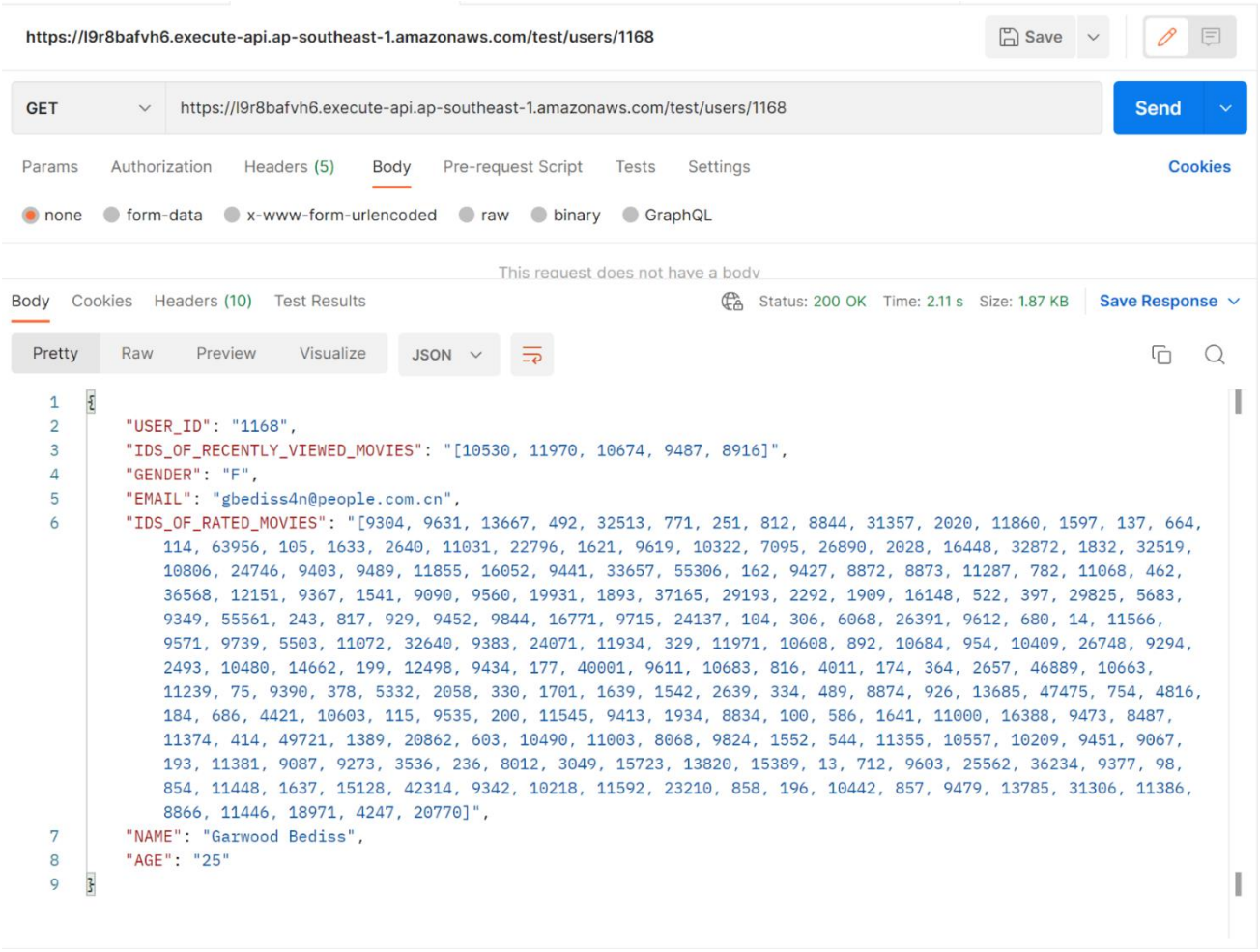


Figure 76: User details returned in JSON format

3.2 Check if an user has rated a movie, then return the infomation if exist

URL: Invoke URL + /ratings?user_id=213&movie_id=862

Method: GET

Sample result:

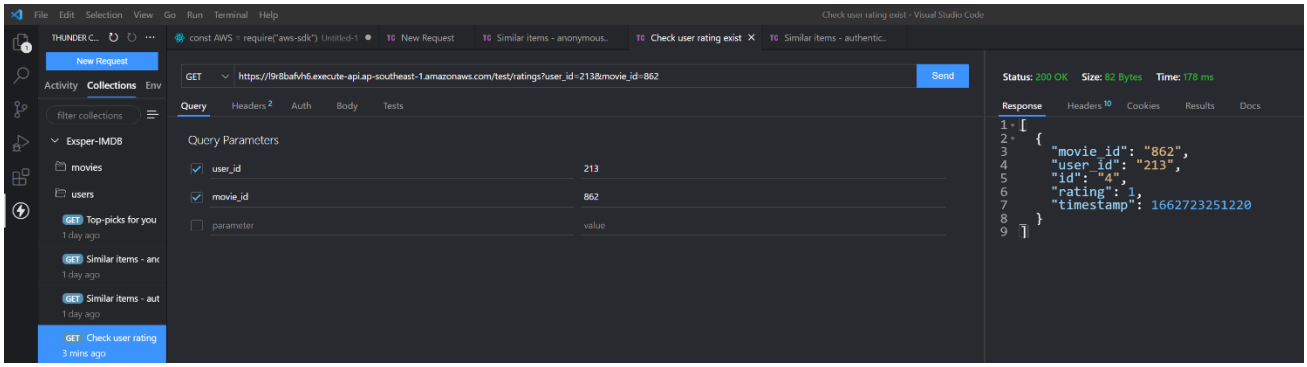


Figure 77: Response indicate the rating timestamp of a certain user on a certain movie