May 25, 2022

Movie Ticket Booking System

Implementation Details and Instructions

Megana Reddy Boddam
Sahana Pandurangi Raghavendra

Instruction for building the code:

1. We have deployed our frontend on AWS API gateway. Since the first endpoint doesn't have an authentication key we have removed the deployment from the AWS server as of now.

   Professor, please send us an email when you will be grading our work. We will re-deploy our project on AWS.

2. To access our project please post the below URL on the web browser(chrome/safari)
   https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/pages/home
3. Please follow the instructions below to recreate our demo. You may inspect the website page to check the API responses and requests between frontend→AWS API gateway→Lambda→ database.
4. Once you have completed the demo. Please ask us for the logs and database updates we will be happy to provide.
5. We have also provided detailed configuration steps for AWS API gateway, Lambda , database and frontend in case our project needs to be configured from scratch on another AWS account.

Executing the steps in demo of Working Website

- Go to the following URL.
  - https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/pages/home
- You should observe Figure 1. Select "Sign Up". Input details in text boxes. "Phone" should have only integers. Select "Register".

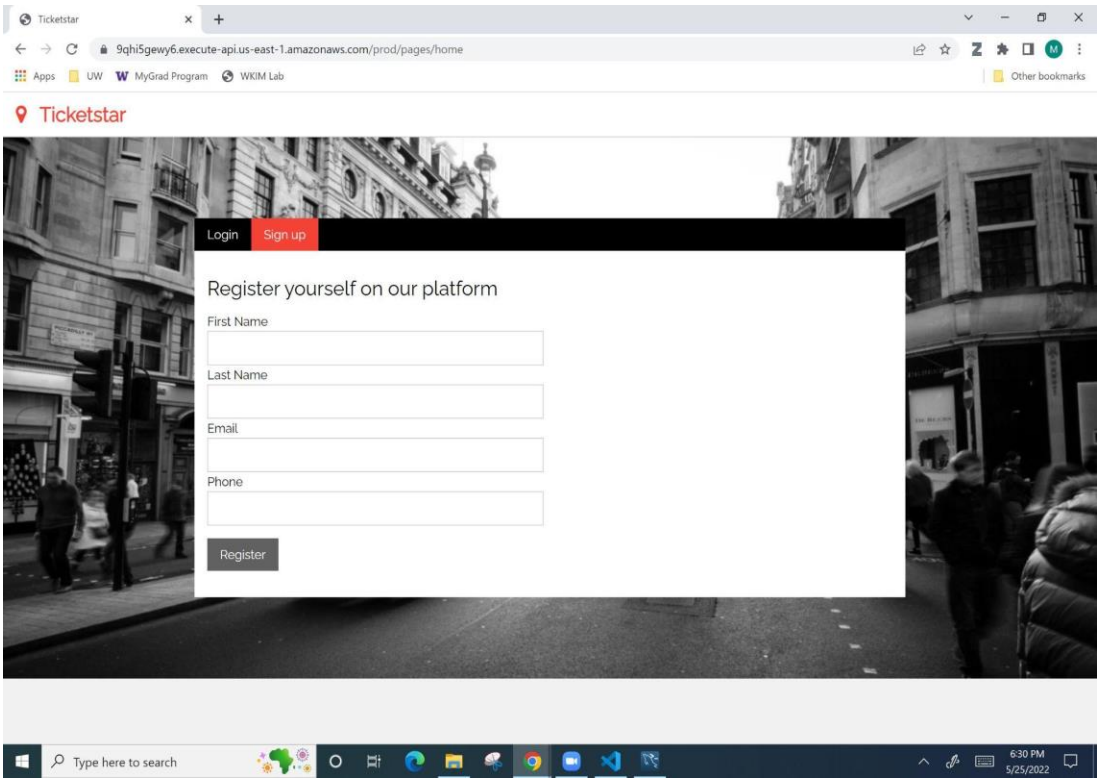


Figure 1: Sign up page of website.

- Observe a pop-up window with "used_id" number. Please save this number.
- Click "Login".
- Input the saved "user_id" number in user_id field on the login screen.

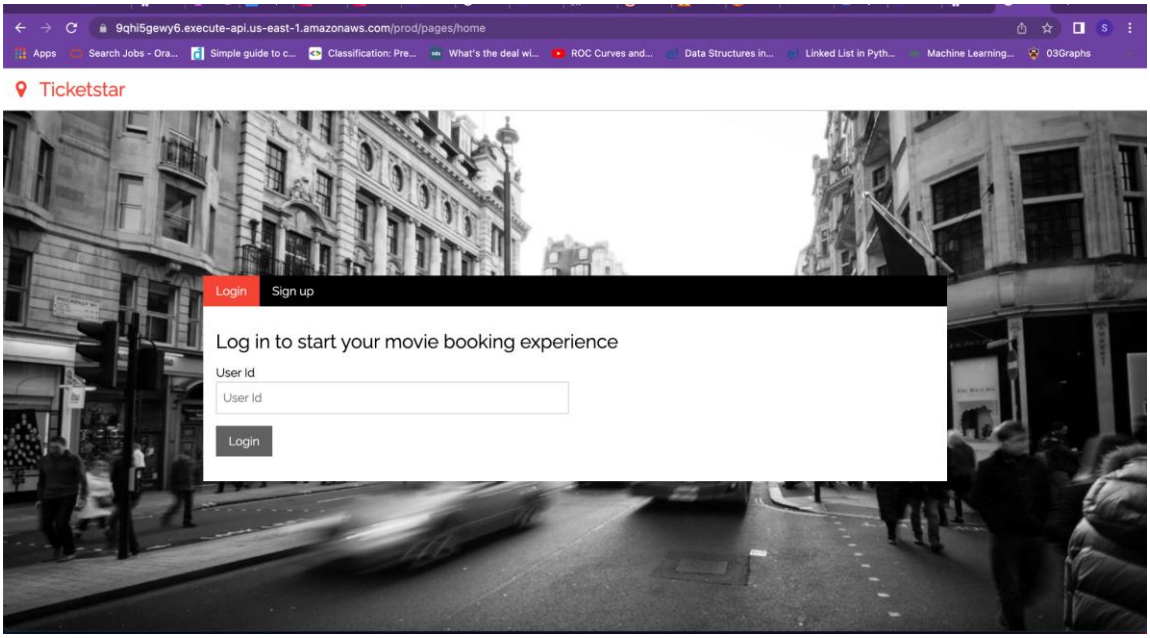


Figure 2: Login screen

- You should observe Figure 3. Select the “Seattle” radio button. Select the “Next: Select Theater” button.

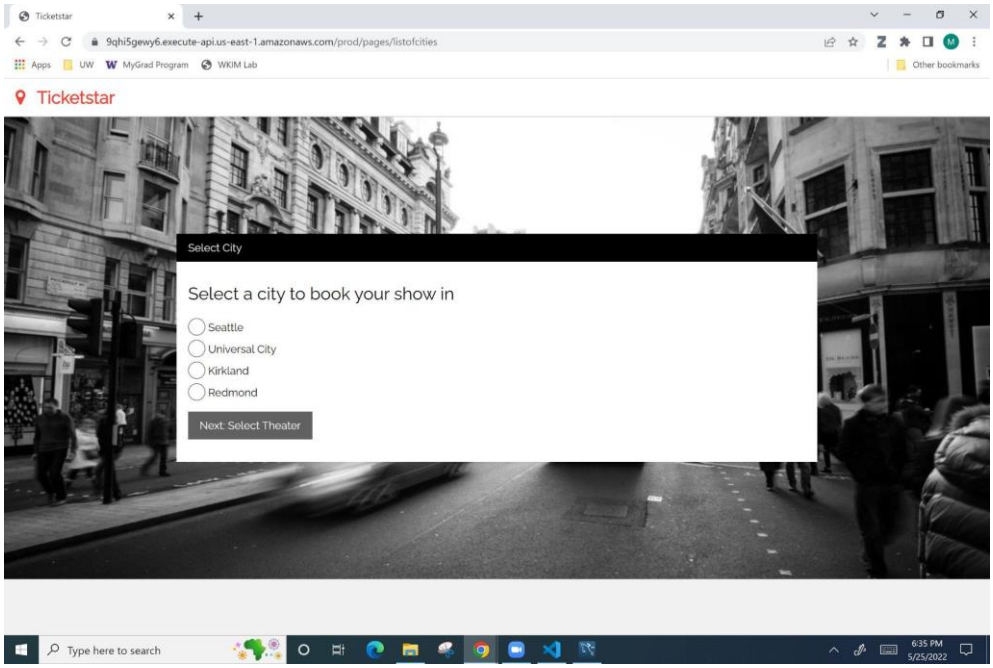


Figure 3: City selection page of website.

- You should observe Figure 4. Select the “Cinemark” option in the drop down menu. Select the “Next: Select Movie” button.

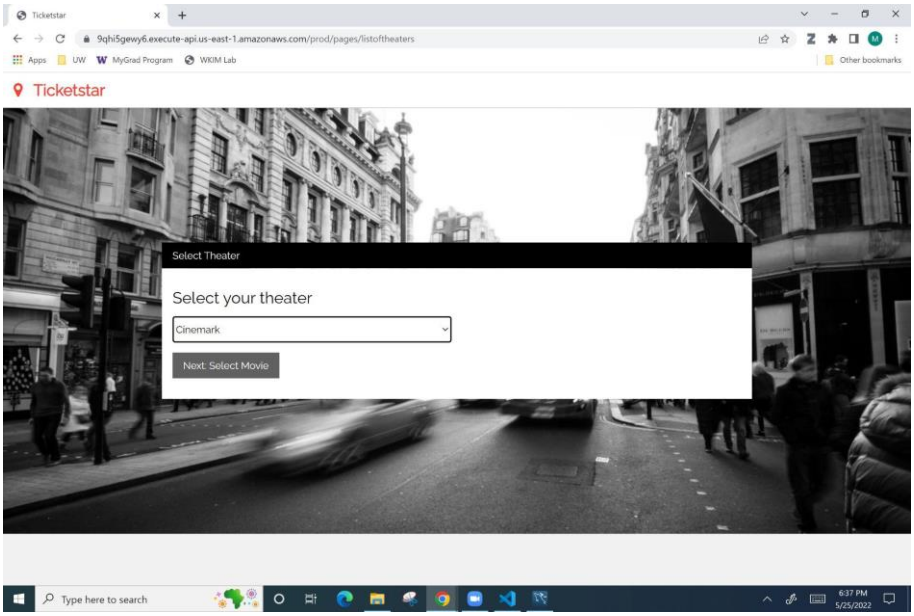


Figure 4: Theater selection page of website.

- You should observe Figure 5. Select the “Top Gun: Maverick” option in the drop down menu. Select the “Next: Select show” button.

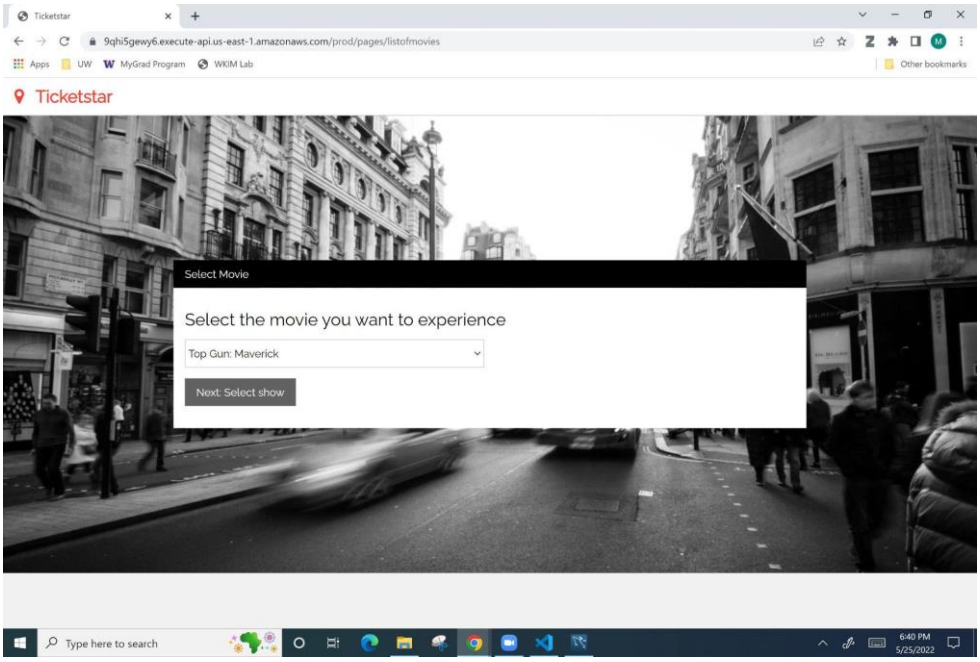


Figure 5: Movie selection page of website.

- You should observe Figure 6. Select the “2022-06-01” option in the drop down menu for “Show dates”. Select the “12:30:00” option in the drop down menu for “Show”. Select the “Next: Select seats” button.

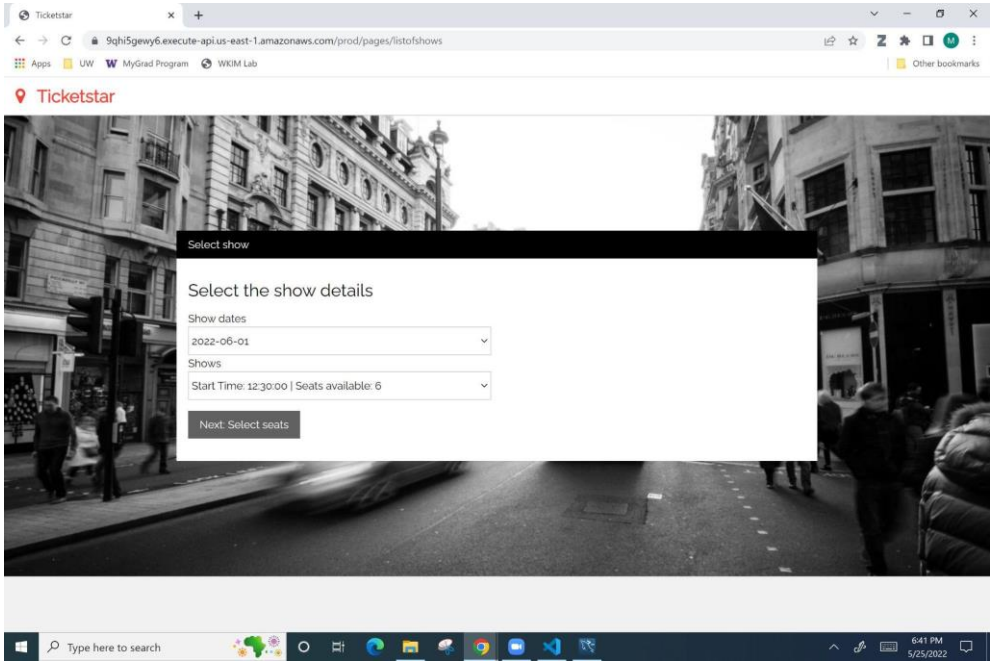


Figure 6: Show date and time selection page of website.

- You should observe Figure 7. Select the seat number “12” and “13”. Select the “Submit Selection” button. You will not observe any changes to the page. There is an internal update of the database. Please click on the submit selection button twice each with a 5 seconds interval. This is because sometimes the database updates fail as there are multiple requests at the same time.Select the “Next: Select food” button.

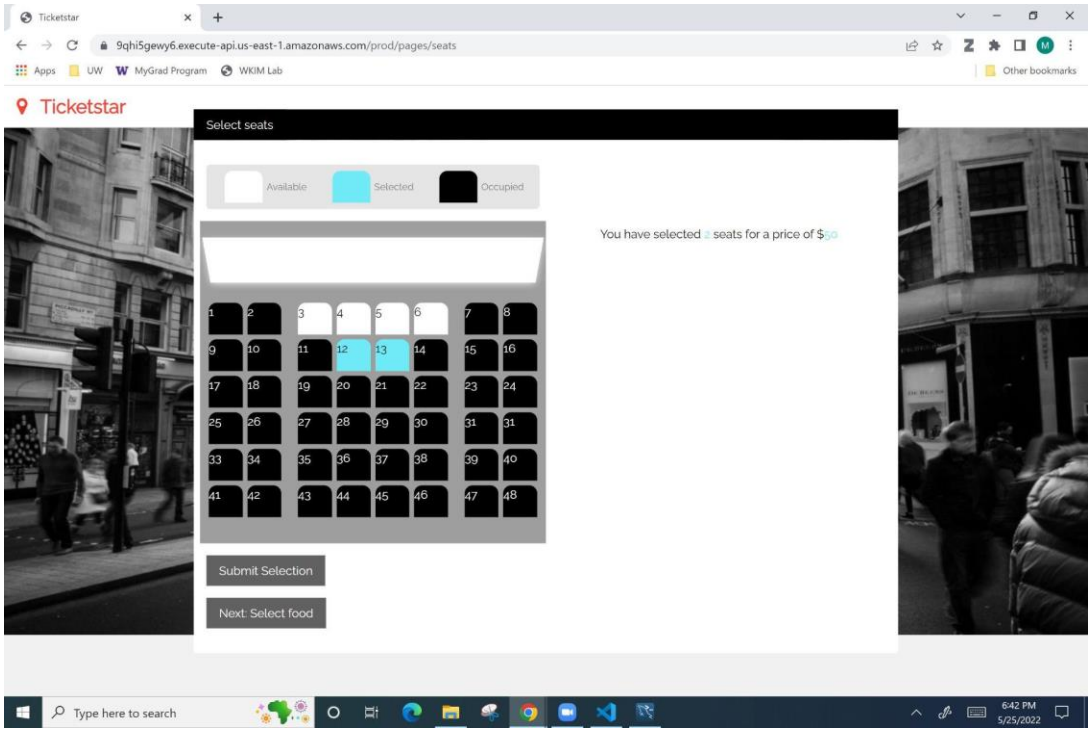


Figure 7: Seat selection page of website

- You should observe Figure 8. Select any option in the drop down menu. Select any quantity option. Select the “Next: Order summary” button.

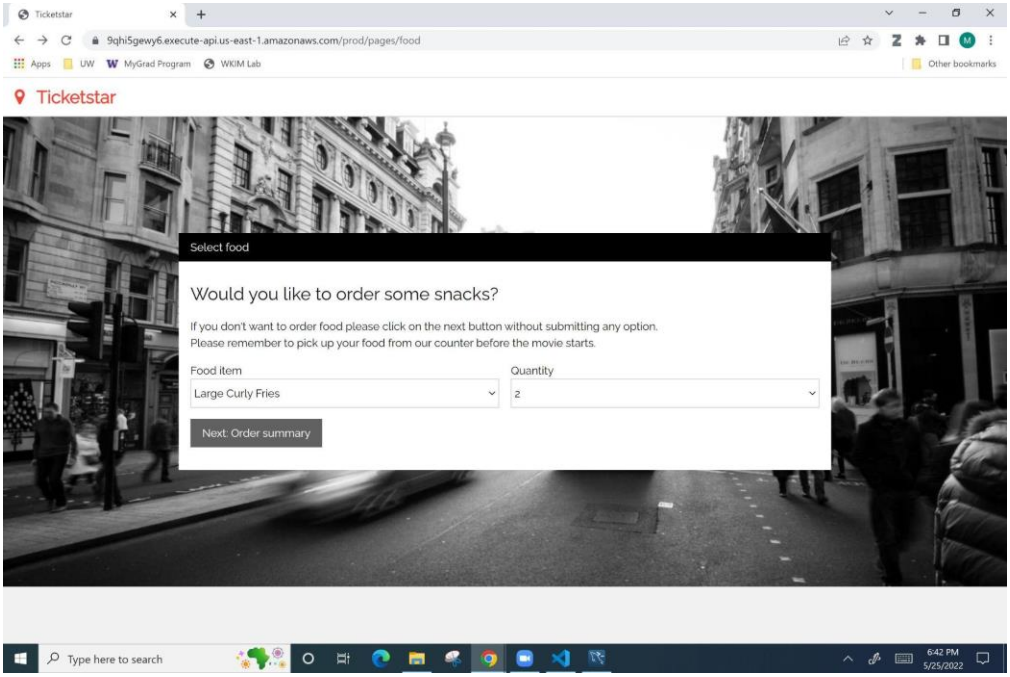


Figure 8: Food selection page of website.

- You should observe Figure 9. The exact values and numbers may not be the same based on what was selected in the previous page. Select the “Next: Payment Processing” button.

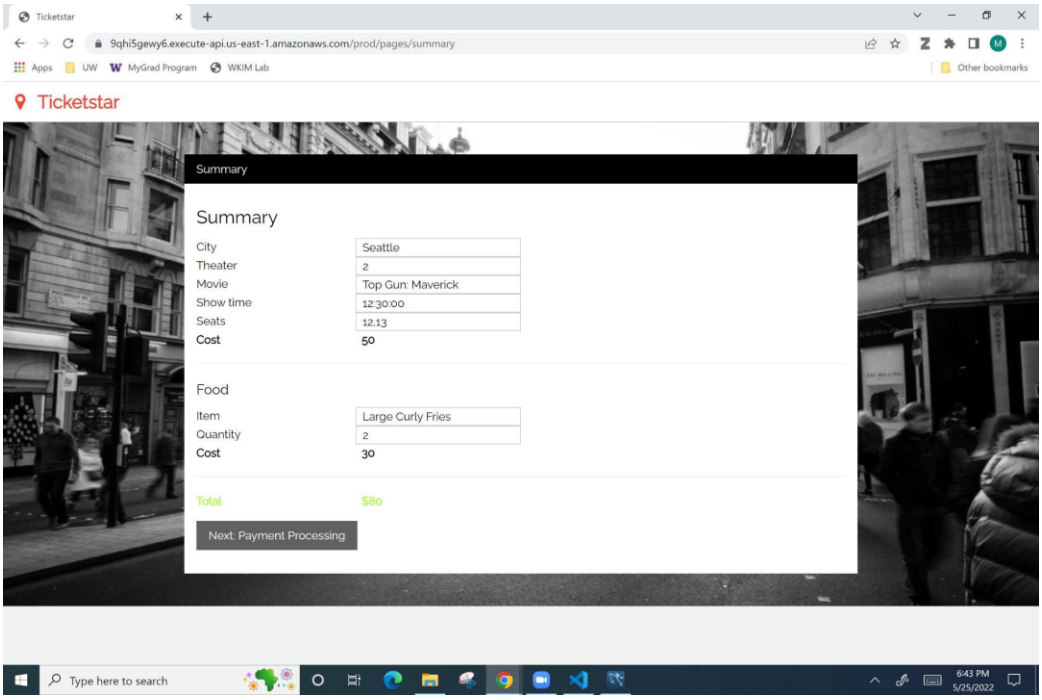


Figure 9: Order summary page of website.

- You should observe Figure 10. Input your details. Select “Submit”. You should observe the message “Your tickets are booked! Thank you for choosing us. Enjoy your movie!”

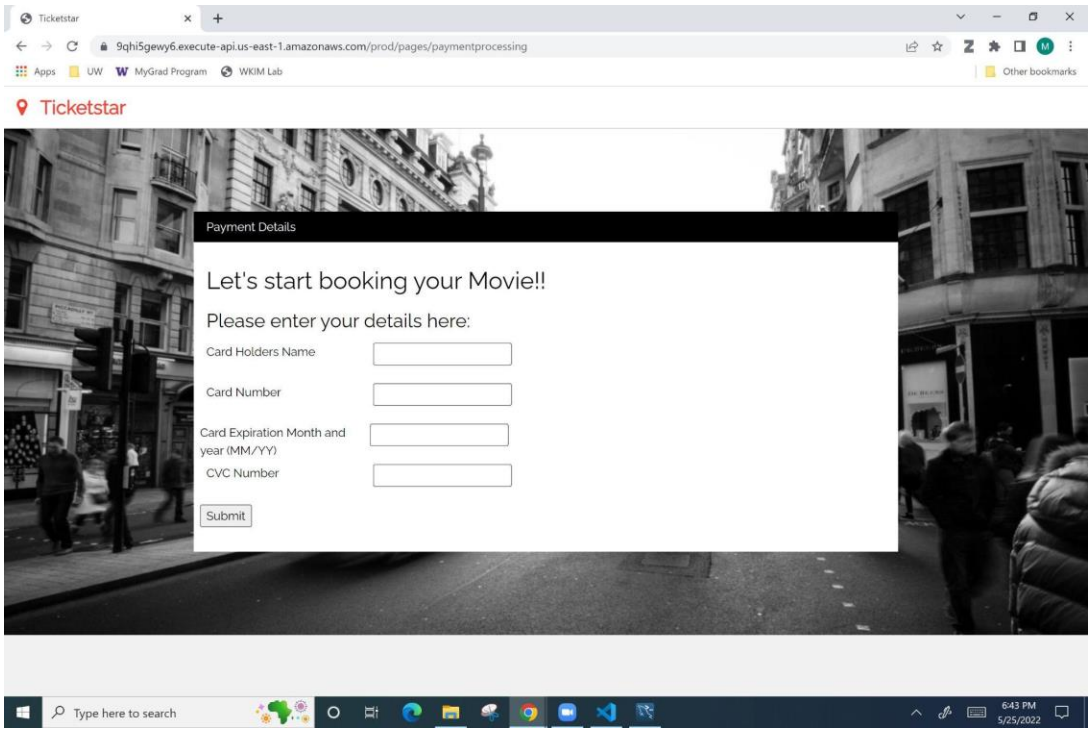


Figure 10: Payment Processing page of website

Configuration Steps to deploy the system on a new AWS account.

Note: Below are the steps to configure and populate a new database for the project. However you can use our already populated database by uploading the following zip files: “search_movies.zip” , “ticket_booking.zip”, “food_booking.zip” and “payment_processing.zip” (provided in the implementation zip file) into the aws lambdas for “SearchMoviesServer”, “TicketBookingServer”, “FoodBookingServer”, and “PaymentProcessingServer” respectively.

Database Tier

- We used Amazon's Relational Database Service (RDS). Below are the steps we followed to configure the database.
- We used the MySQL Community Engine in the Free Tier.
- We used a T3 Micro instance named "db.t3.micro" in Amazon Web Services.
- It provided us with 1 Gibibyte of memory and used an Intel Enterprise-level 2.5 GHz processor. It provided us with 64-bit CPU architecture and in Amazon's terms, "low-moderate" network performance.
- We used the "Easy Create" database creation method in the RDS. This automatically selected best recommended practices such as enabled encryption, default virtual private cloud with its security groups, default connection port of 3306, enabled monitoring, and more.
- Figures 11 and 12 show screenshots of the pages in AWS that created the database.

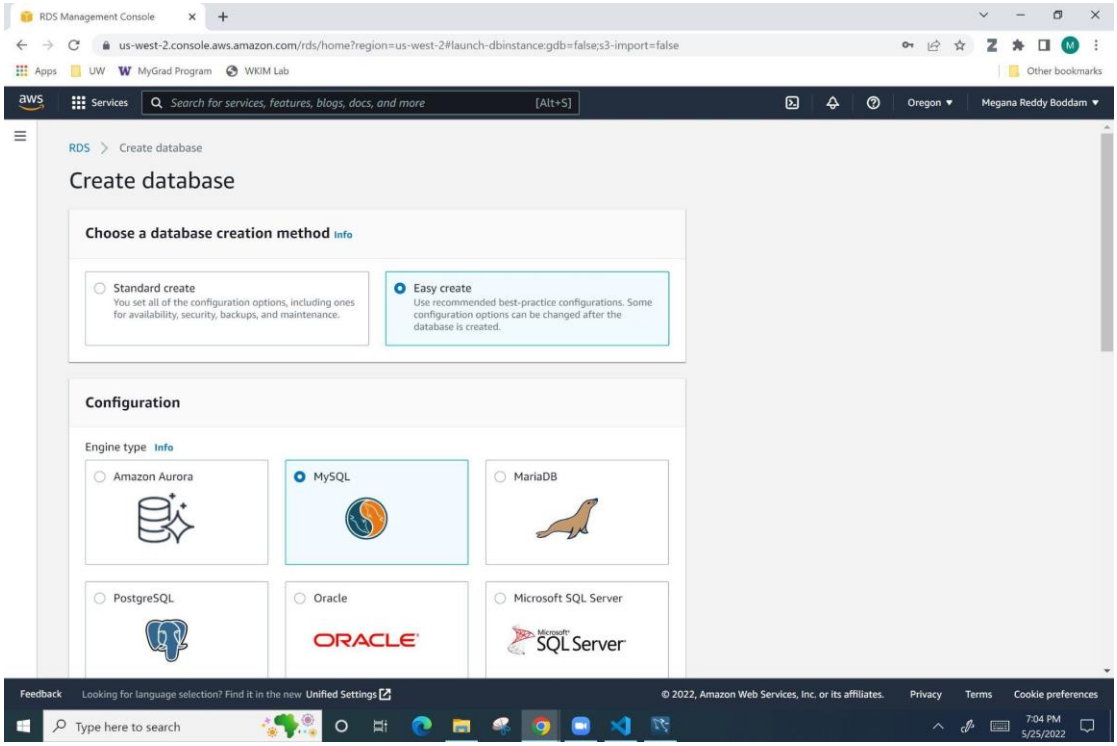


Figure 11: Database creation, "Easy create" and "MySQL" options selected.

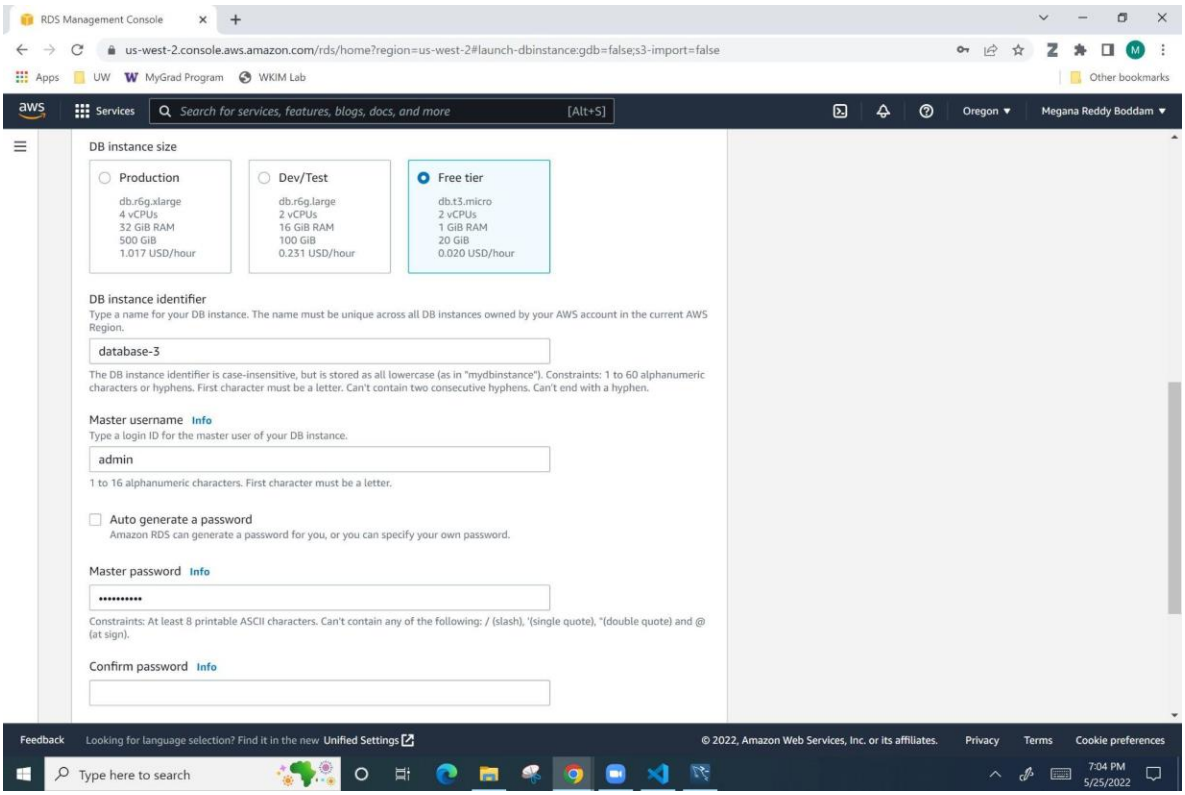


Figure 12: Database creation, "Free tier" option selected. Username and Password chosen.

- We had to change the connectivity configuration setting as shown in Figures 13 and 14 to make the database connectable in the middle tier. We made the RDS accessible publicly. We could have placed all our services in a virtual private network in Amazon; however, we did not have the time to set up our services in that manner.

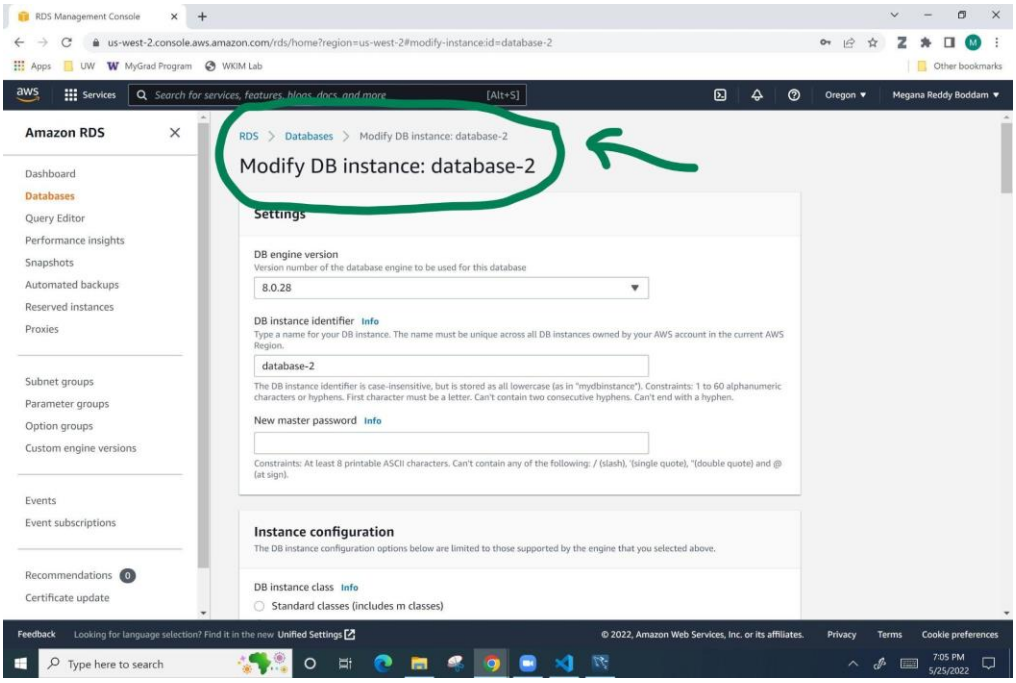


Figure 13: The location in AWS to modify the database connectivity.

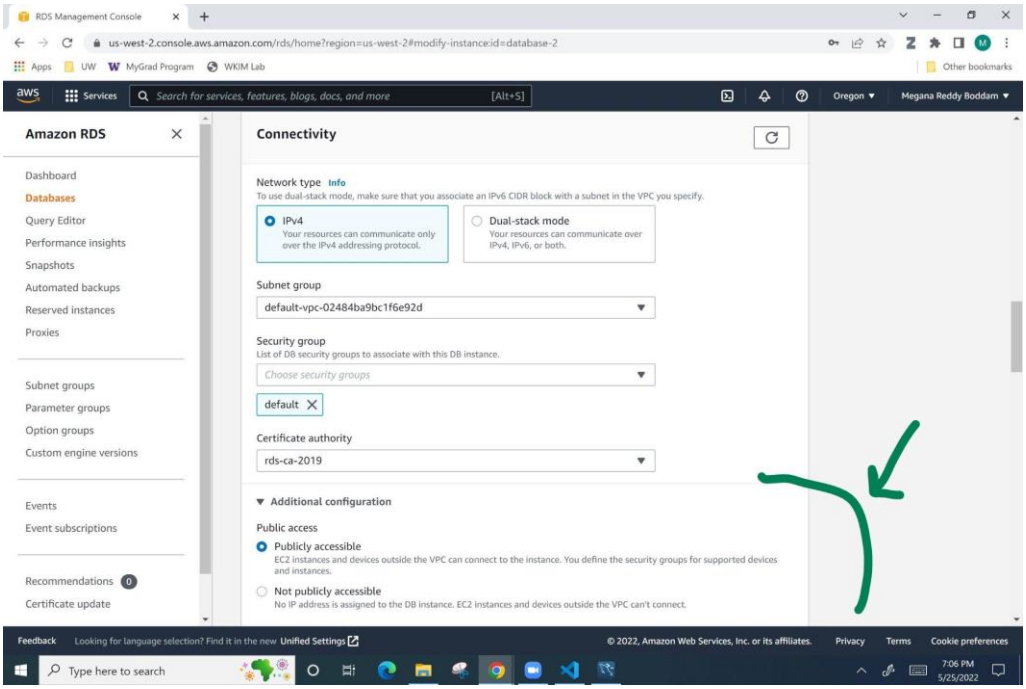
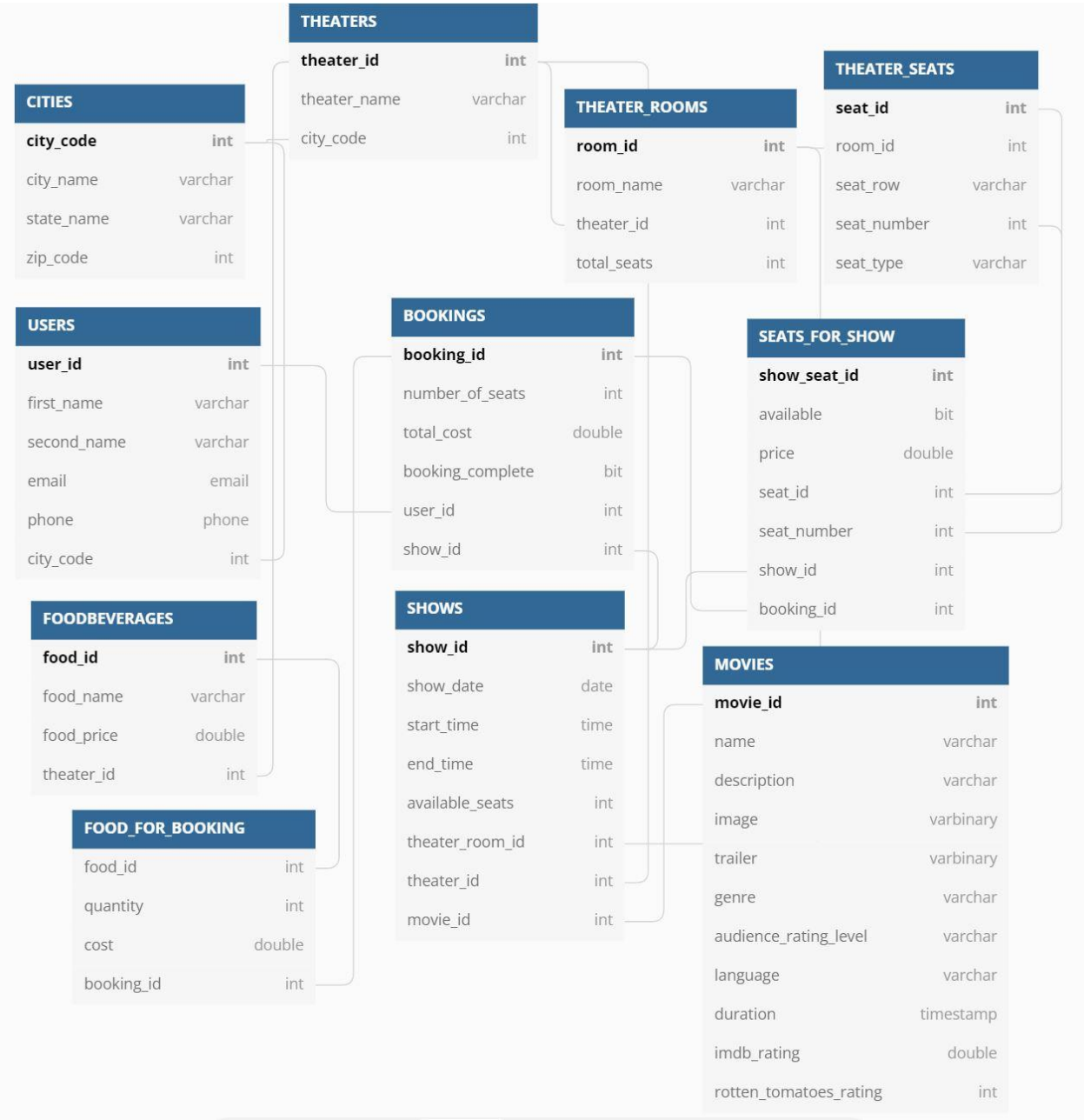


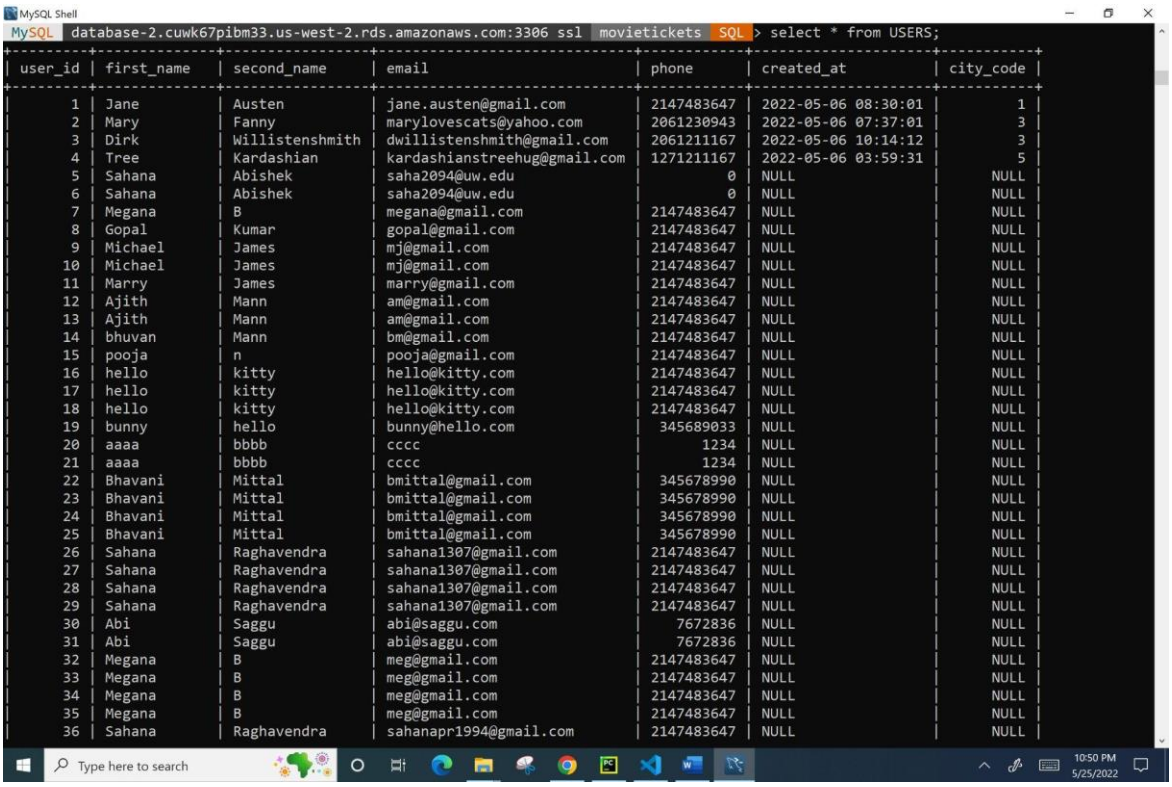
Figure 14: The setting to change to "Publically accessible" in the form from Figure 13.

- The exact database (db) host name, db name, db administrative username, and db administrative password are in the **"config.json"** file.
    - These will need to be accessed in our servers when we try to connect to the db to modify it.
- We did not have the time to create an administrative interface that would allow an admin to add or remove tables or data in the database such as the currently playing movies, list of theaters in a city, the seat details in those theaters, etc.
- Therefore, we used the Oracle MySQL Shell (downloaded from this link in References - (1)) to access the Amazon RDS instance, create a "movietickets" database, create relational tables, and populate those tables with preliminary data.
- The schema of the relational tables is in figure 15, and the SQL code for the tables in the **"dbcreation.sql"** file.

Figure 15: "movietickets" database schema.

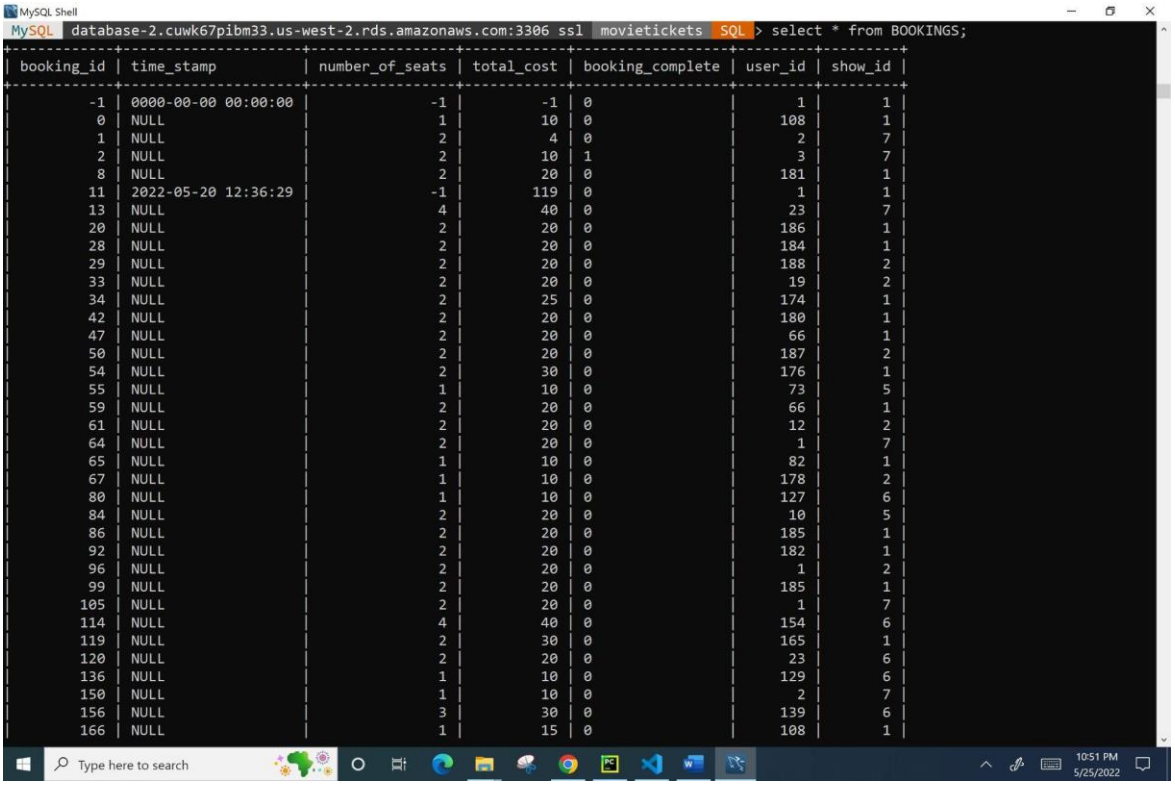


- We used the data in the **"testdata_small.sql"** file to populate these tables.
- Following Figures 16 to 23 are snapshots of the database once it has been filled with data. The captions show the SQL queries used to retrieve this data.



```
MySQL Shell
MySQL  database-2.cuwk67pibm33.us-west-2.rds.amazonaws.com:3306 ssl  movietickets  SQL > select * from USERS;
+---------+-------------+----------------+-----------------------------+------------+---------------------+-----------+
| user_id | first_name  | second_name    | email                       | phone      | created_at          | city_code |
+---------+-------------+----------------+-----------------------------+------------+---------------------+-----------+
|       1 | Jane        | Austen         | jane.austen@gmail.com       | 2147483647 | 2022-05-06 08:30:01 |         1 |
|       2 | Mary        | Fanny          | marylovescats@yahoo.com     | 2061230943 | 2022-05-06 07:37:01 |         3 |
|       3 | Dirk        | Willistenshmith| dwillistenshmith@gmail.com  | 2061211167 | 2022-05-06 10:14:12 |         3 |
|       4 | Tree        | Kardashian     | kardashianstreehug@gmail.com| 1271211167 | 2022-05-06 03:59:31 |         5 |
|       5 | Sahana      | Abishek        | saha2094@uw.edu             |          0 | NULL                |      NULL |
|       6 | Sahana      | Abishek        | saha2094@uw.edu             |          0 | NULL                |      NULL |
|       7 | Megana      | B              | megana@gmail.com            | 2147483647 | NULL                |      NULL |
|       8 | Gopal       | Kumar          | gopal@gmail.com             | 2147483647 | NULL                |      NULL |
|       9 | Michael     | James          | mj@gmail.com                | 2147483647 | NULL                |      NULL |
|      10 | Michael     | James          | mj@gmail.com                | 2147483647 | NULL                |      NULL |
|      11 | Marry       | James          | marry@gmail.com             | 2147483647 | NULL                |      NULL |
|      12 | Ajith       | Mann           | am@gmail.com                | 2147483647 | NULL                |      NULL |
|      13 | Ajith       | Mann           | am@gmail.com                | 2147483647 | NULL                |      NULL |
|      14 | bhuvan      | Mann           | bm@gmail.com                | 2147483647 | NULL                |      NULL |
|      15 | pooja       | n              | pooja@gmail.com             | 2147483647 | NULL                |      NULL |
|      16 | hello       | kitty          | hello@kitty.com             | 2147483647 | NULL                |      NULL |
|      17 | hello       | kitty          | hello@kitty.com             | 2147483647 | NULL                |      NULL |
|      18 | hello       | kitty          | hello@kitty.com             | 2147483647 | NULL                |      NULL |
|      19 | bunny       | hello          | bunny@hello.com             |  345689033 | NULL                |      NULL |
|      20 | aaaa        | bbbb           | cccc                        |       1234 | NULL                |      NULL |
|      21 | aaaa        | bbbb           | cccc                        |       1234 | NULL                |      NULL |
|      22 | Bhavani     | Mittal         | bmittal@gmail.com           |  345678990 | NULL                |      NULL |
|      23 | Bhavani     | Mittal         | bmittal@gmail.com           |  345678990 | NULL                |      NULL |
|      24 | Bhavani     | Mittal         | bmittal@gmail.com           |  345678990 | NULL                |      NULL |
|      25 | Bhavani     | Mittal         | bmittal@gmail.com           |  345678990 | NULL                |      NULL |
|      26 | Sahana      | Raghavendra    | sahana1307@gmail.com        | 2147483647 | NULL                |      NULL |
|      27 | Sahana      | Raghavendra    | sahana1307@gmail.com        | 2147483647 | NULL                |      NULL |
|      28 | Sahana      | Raghavendra    | sahana1307@gmail.com        | 2147483647 | NULL                |      NULL |
|      29 | Sahana      | Raghavendra    | sahana1307@gmail.com        | 2147483647 | NULL                |      NULL |
|      30 | Abi         | Saggu          | abi@saggu.com               |    7672836 | NULL                |      NULL |
|      31 | Abi         | Saggu          | abi@saggu.com               |    7672836 | NULL                |      NULL |
|      32 | Megana      | B              | meg@gmail.com               | 2147483647 | NULL                |      NULL |
|      33 | Megana      | B              | meg@gmail.com               | 2147483647 | NULL                |      NULL |
|      34 | Megana      | B              | meg@gmail.com               | 2147483647 | NULL                |      NULL |
|      35 | Megana      | B              | meg@gmail.com               | 2147483647 | NULL                |      NULL |
|      36 | Sahana      | Raghavendra    | sahanapr1994@gmail.com      | 2147483647 | NULL                |      NULL |
Type here to search
10:50 PM
5/25/2022
```

*Figure 16: SELECT * FROM USERS;*



```
MySQL Shell
MySQL  database-2.cuwk67pibm33.us-west-2.rds.amazonaws.com:3306 ssl  movietickets  SQL > select * from BOOKINGS;
+------------+---------------------+-----------------+------------+------------------+---------+---------+
| booking_id | time_stamp          | number_of_seats | total_cost | booking_complete | user_id | show_id |
+------------+---------------------+-----------------+------------+------------------+---------+---------+
|         -1 | 0000-00-00 00:00:00 |              -1 |         -1 | 0                |       1 |       1 |
|          0 | NULL                |               1 |         10 | 0                |     108 |       1 |
|          1 | NULL                |               2 |          4 | 0                |       2 |       7 |
|          2 | NULL                |               2 |         10 | 1                |       3 |       7 |
|          8 | NULL                |               2 |         20 | 0                |     181 |       1 |
|         11 | 2022-05-20 12:36:29 |              -1 |        119 | 0                |       1 |       1 |
|         13 | NULL                |               4 |         40 | 0                |      23 |       7 |
|         20 | NULL                |               2 |         20 | 0                |     186 |       1 |
|         28 | NULL                |               2 |         20 | 0                |     184 |       1 |
|         29 | NULL                |               2 |         20 | 0                |     188 |       2 |
|         33 | NULL                |               2 |         20 | 0                |      19 |       2 |
|         34 | NULL                |               2 |         25 | 0                |     174 |       1 |
|         42 | NULL                |               2 |         20 | 0                |     180 |       1 |
|         47 | NULL                |               2 |         20 | 0                |      66 |       1 |
|         50 | NULL                |               2 |         20 | 0                |     187 |       2 |
|         54 | NULL                |               2 |         30 | 0                |     176 |       1 |
|         55 | NULL                |               1 |         10 | 0                |      73 |       5 |
|         59 | NULL                |               2 |         20 | 0                |      66 |       1 |
|         61 | NULL                |               2 |         20 | 0                |      12 |       2 |
|         64 | NULL                |               2 |         20 | 0                |       1 |       7 |
|         65 | NULL                |               1 |         10 | 0                |      82 |       1 |
|         67 | NULL                |               1 |         10 | 0                |     178 |       2 |
|         80 | NULL                |               1 |         10 | 0                |     127 |       6 |
|         84 | NULL                |               2 |         20 | 0                |      10 |       5 |
|         86 | NULL                |               2 |         20 | 0                |     185 |       1 |
|         92 | NULL                |               2 |         20 | 0                |     182 |       1 |
|         96 | NULL                |               2 |         20 | 0                |       1 |       2 |
|         99 | NULL                |               2 |         20 | 0                |     185 |       1 |
|        105 | NULL                |               2 |         20 | 0                |       1 |       7 |
|        114 | NULL                |               4 |         40 | 0                |     154 |       6 |
|        119 | NULL                |               2 |         30 | 0                |     165 |       1 |
|        120 | NULL                |               2 |         20 | 0                |      23 |       6 |
|        136 | NULL                |               1 |         10 | 0                |     129 |       6 |
|        150 | NULL                |               1 |         10 | 0                |       2 |       7 |
|        156 | NULL                |               3 |         30 | 0                |     139 |       6 |
|        166 | NULL                |               1 |         15 | 0                |     108 |       1 |
Type here to search
10:51 PM
5/25/2022
```

*Figure 17: SELECT * FROM BOOKINGS;*

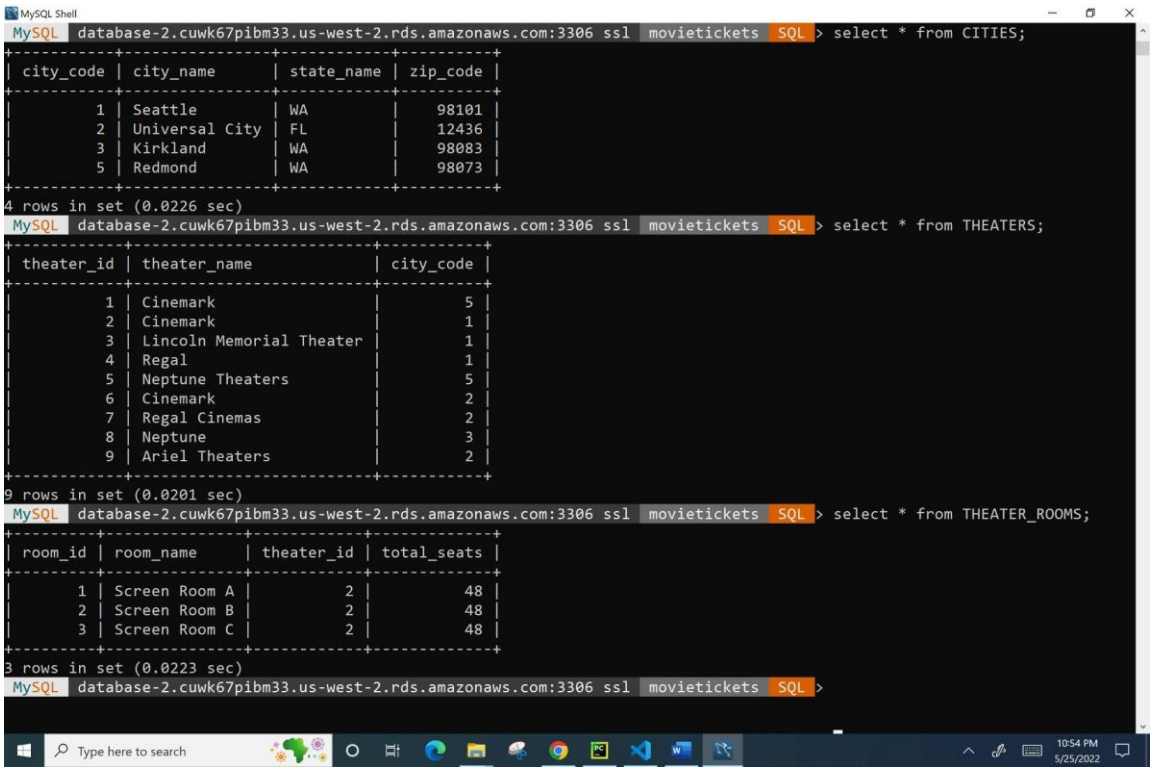


*Figure 18: SELECT * FROM CITIES; SELECT * FROM THEATERS; SELECT * FROM THEATER_ROOMS;*

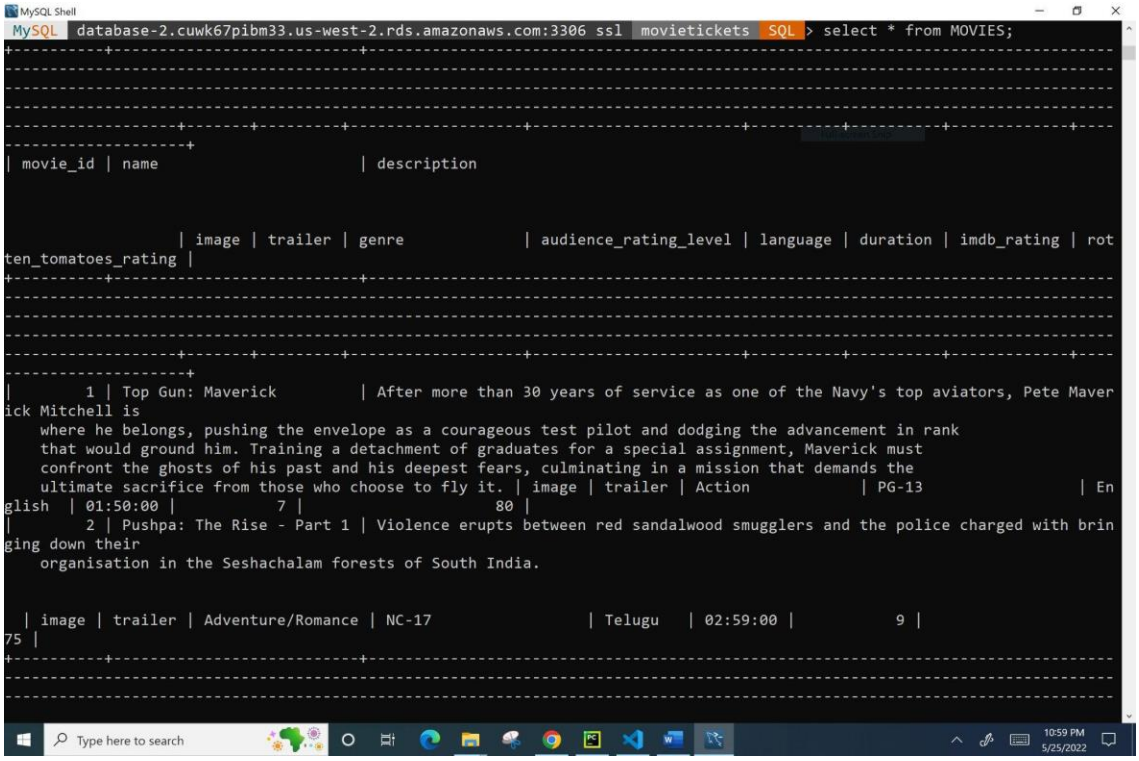


*Figure 19: SELECT * FROM MOVIES; We apologize for the formatting.*

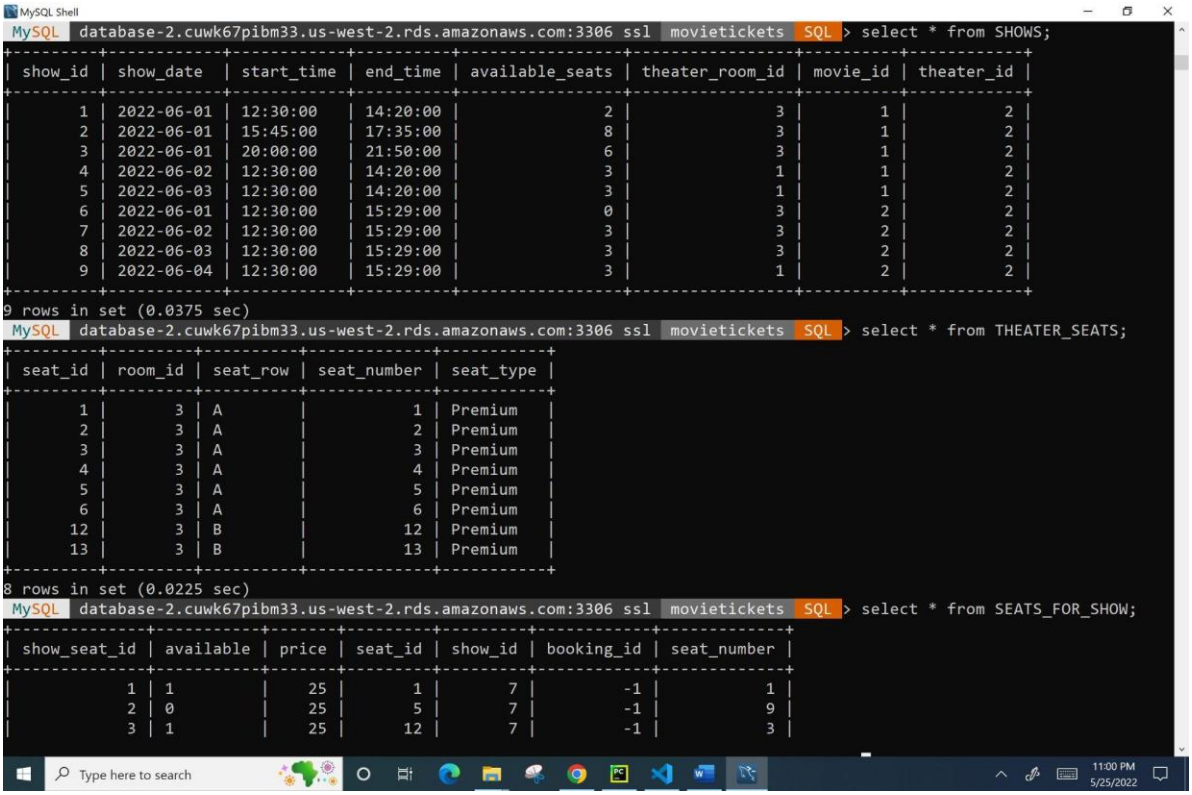


*Figure 20: SELECT * FROM SHOWS; SELECT * FROM THEATER_SEATS;*

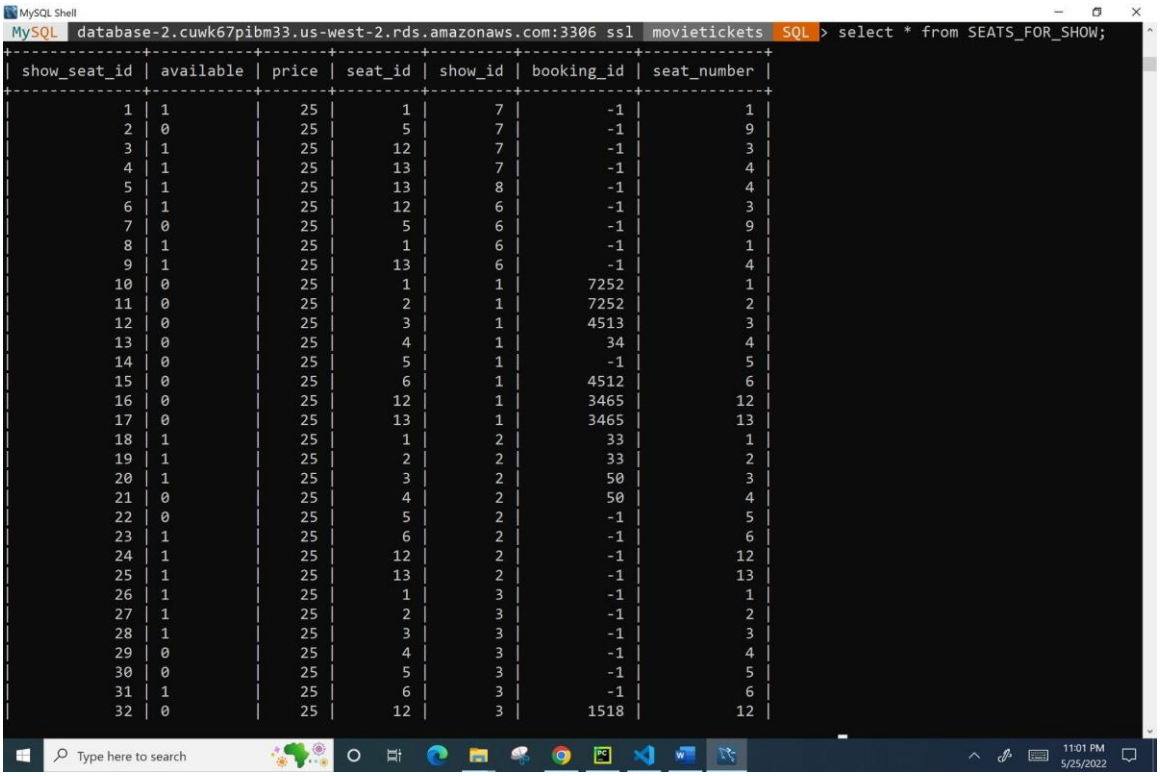
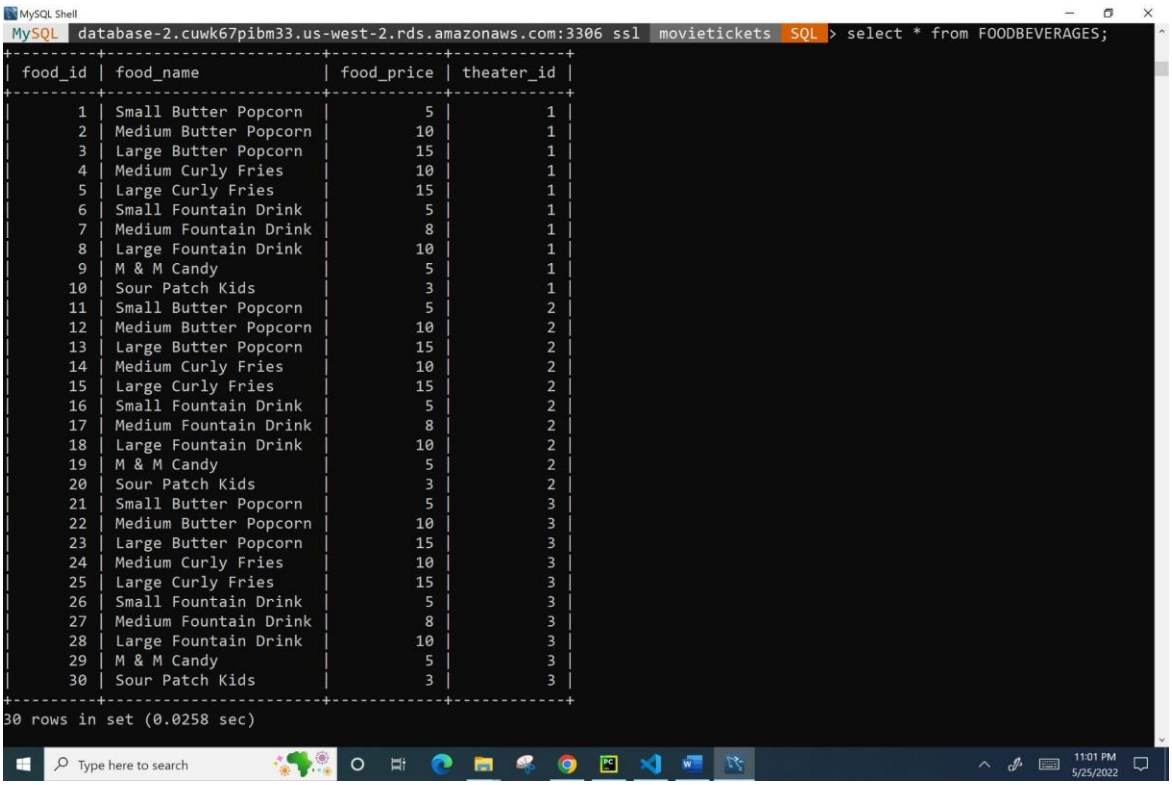
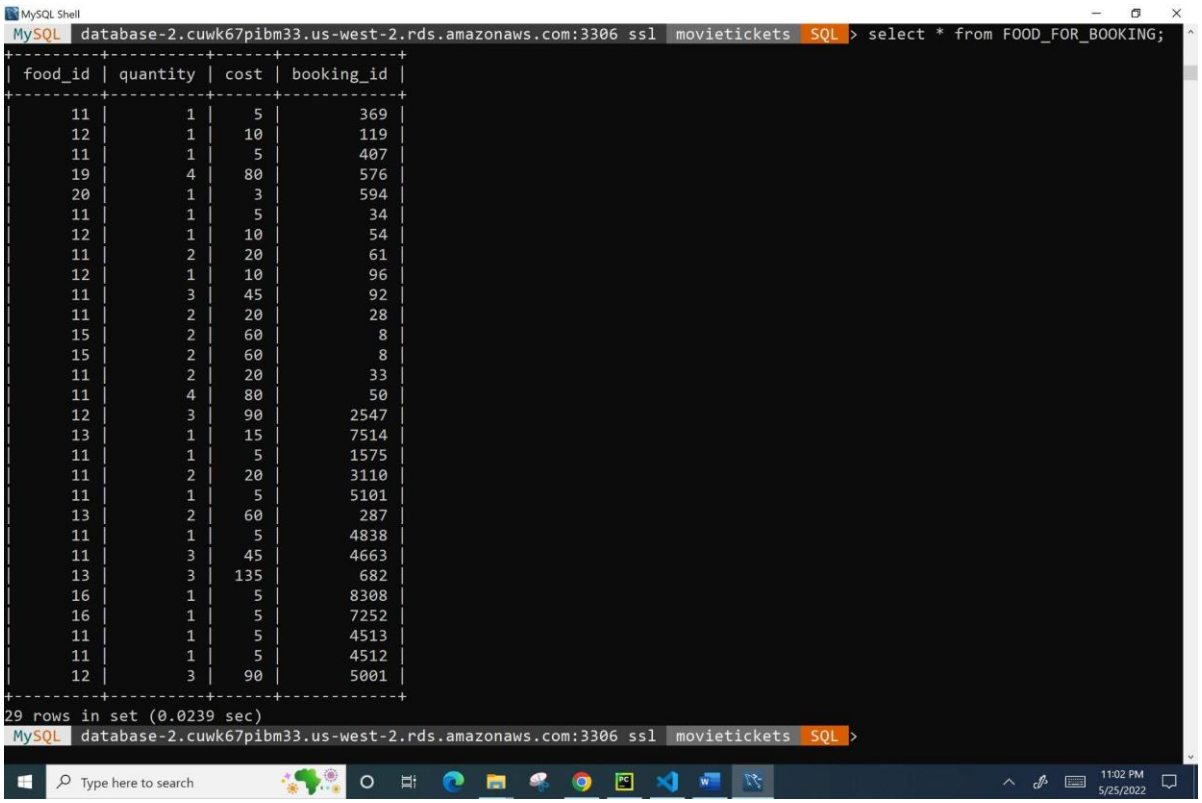


*Figure 21: SELECT * FROM SEATS_FOR_SHOW;*



```
MySQL Shell
MySQL  database-2.cuwk67pibm33.us-west-2.rds.amazonaws.com:3306 ssl  movietickets  SQL > select * from FOODBEVERAGES;
+---------+----------------------+------------+------------+
| food_id | food_name            | food_price | theater_id |
+---------+----------------------+------------+------------+
|       1 | Small Butter Popcorn |          5 |          1 |
|       2 | Medium Butter Popcorn |         10 |          1 |
|       3 | Large Butter Popcorn |         15 |          1 |
|       4 | Medium Curly Fries   |         10 |          1 |
|       5 | Large Curly Fries    |         15 |          1 |
|       6 | Small Fountain Drink |          5 |          1 |
|       7 | Medium Fountain Drink |         8 |          1 |
|       8 | Large Fountain Drink |         10 |          1 |
|       9 | M & M Candy          |          5 |          1 |
|      10 | Sour Patch Kids      |          3 |          1 |
|      11 | Small Butter Popcorn |          5 |          2 |
|      12 | Medium Butter Popcorn |         10 |          2 |
|      13 | Large Butter Popcorn |         15 |          2 |
|      14 | Medium Curly Fries   |         10 |          2 |
|      15 | Large Curly Fries    |         15 |          2 |
|      16 | Small Fountain Drink |          5 |          2 |
|      17 | Medium Fountain Drink |         8 |          2 |
|      18 | Large Fountain Drink |         10 |          2 |
|      19 | M & M Candy          |          5 |          2 |
|      20 | Sour Patch Kids      |          3 |          2 |
|      21 | Small Butter Popcorn |          5 |          3 |
|      22 | Medium Butter Popcorn |         10 |          3 |
|      23 | Large Butter Popcorn |         15 |          3 |
|      24 | Medium Curly Fries   |         10 |          3 |
|      25 | Large Curly Fries    |         15 |          3 |
|      26 | Small Fountain Drink |          5 |          3 |
|      27 | Medium Fountain Drink |         8 |          3 |
|      28 | Large Fountain Drink |         10 |          3 |
|      29 | M & M Candy          |          5 |          3 |
|      30 | Sour Patch Kids      |          3 |          3 |
+---------+----------------------+------------+------------+
30 rows in set (0.0258 sec)
```

*Figure 22: SELECT * FROM FOODBEVERAGES;*



```
MySQL Shell
MySQL  database-2.cuwk67pibm33.us-west-2.rds.amazonaws.com:3306 ssl  movietickets  SQL > select * from FOOD_FOR_BOOKING;
+---------+----------+------+------------+
| food_id | quantity | cost | booking_id |
+---------+----------+------+------------+
|      11 |        1 |    5 |        369 |
|      12 |        1 |   10 |        119 |
|      11 |        1 |    5 |        407 |
|      19 |        4 |   80 |        576 |
|      20 |        1 |    3 |        594 |
|      11 |        1 |    5 |         34 |
|      12 |        1 |   10 |         54 |
|      11 |        2 |   20 |         61 |
|      12 |        1 |   10 |         96 |
|      11 |        3 |   45 |         92 |
|      11 |        2 |   20 |         28 |
|      15 |        2 |   60 |          8 |
|      15 |        2 |   60 |          8 |
|      11 |        2 |   20 |         33 |
|      11 |        4 |   80 |         50 |
|      12 |        3 |   90 |       2547 |
|      13 |        1 |   15 |       7514 |
|      11 |        1 |    5 |       1575 |
|      11 |        2 |   20 |       3110 |
|      11 |        1 |    5 |       5101 |
|      13 |        2 |   60 |        287 |
|      11 |        1 |    5 |       4838 |
|      11 |        3 |   45 |       4663 |
|      13 |        3 |  135 |        682 |
|      16 |        1 |    5 |       8308 |
|      16 |        1 |    5 |       7252 |
|      11 |        1 |    5 |       4513 |
|      11 |        1 |    5 |       4512 |
|      12 |        3 |   90 |       5001 |
+---------+----------+------+------------+
29 rows in set (0.0239 sec)
MySQL  database-2.cuwk67pibm33.us-west-2.rds.amazonaws.com:3306 ssl  movietickets  SQL >
```

*Figure 23: SELECT * FROM FOOD_FOR_BOOKING;*

- The data in **"testdata_large.sql"** is not fully ready to be populated into the website for rigorous testing yet. However, running the scripts in this file populates all the tables with large sets of data.

Middle Tier

- We used AWS Lambda functions to develop and produce our four servers: **"SearchMoviesServer", "TicketBookingServer", "FoodBookingServer", and "PaymentProcessingServer"**.
- Create 4 lambda functions in AWS using the below steps for **"SearchMoviesServer", "TicketBookingServer", "FoodBookingServer", and "PaymentProcessingServer"** and upload the respective zip files from our source code into the respective lambda functions. Eg: Upload search_movies.zip into "**SearchMoviesServer"** lambda function.

- Each server has its own lambda function, and the following are the standard configuration steps we used. Figure 24 shows a screenshot of all of these settings.
    - Choose to "Author from scratch".
    - The function names are those provided in the previous bullet point.
    - The runtime is "Node.js 16.x".
    - The architecture is "x86_64".
    - The permissions to access each Lambda function are default; this will also upload our logs to Amazon CloudWatch Logs, which immensely helped us during our development and testing phases.

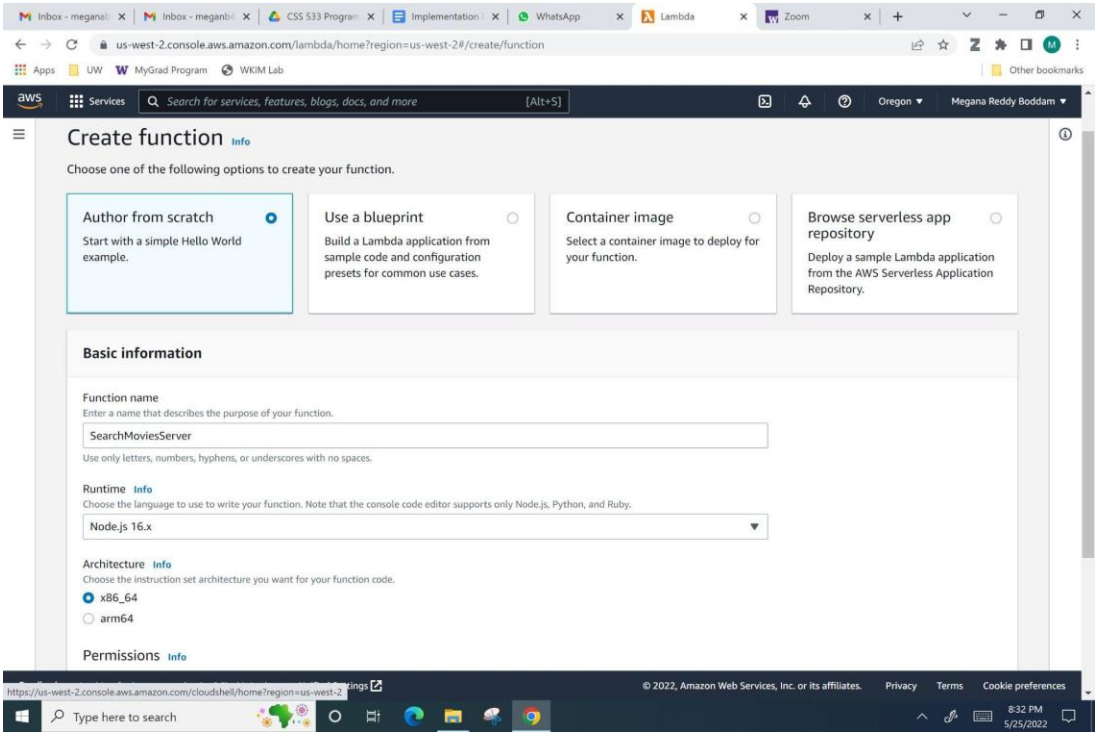


Figure 24: Serverless lambda function creation configuration settings.

- To access the previously created db, we had to locally, on our computers, install various MySQL and sql script parsing packages through Javascript Node Package Manager.
- All of our dependencies are described in **"package-lock.json"** and the Javascript libraries are downloaded in the **"node-modules"** folder.
- Each of the following folders **"SearchMoviesServer", "TicketBookingServer", "FoodBookingServer", and "PaymentProcessingServer" have "node-modules", "config.json", "package-lock.json", and "index.js"**. "index.js" contains the main programs with our business logic.
- Each folder was zipped and uploaded to its appropriate AWS Lambda Function.
- Below are the instructions to connect AWS Lambda with AWS API Gateway.
- After reaching Figure 25, choose "Build" in the "REST API" section.

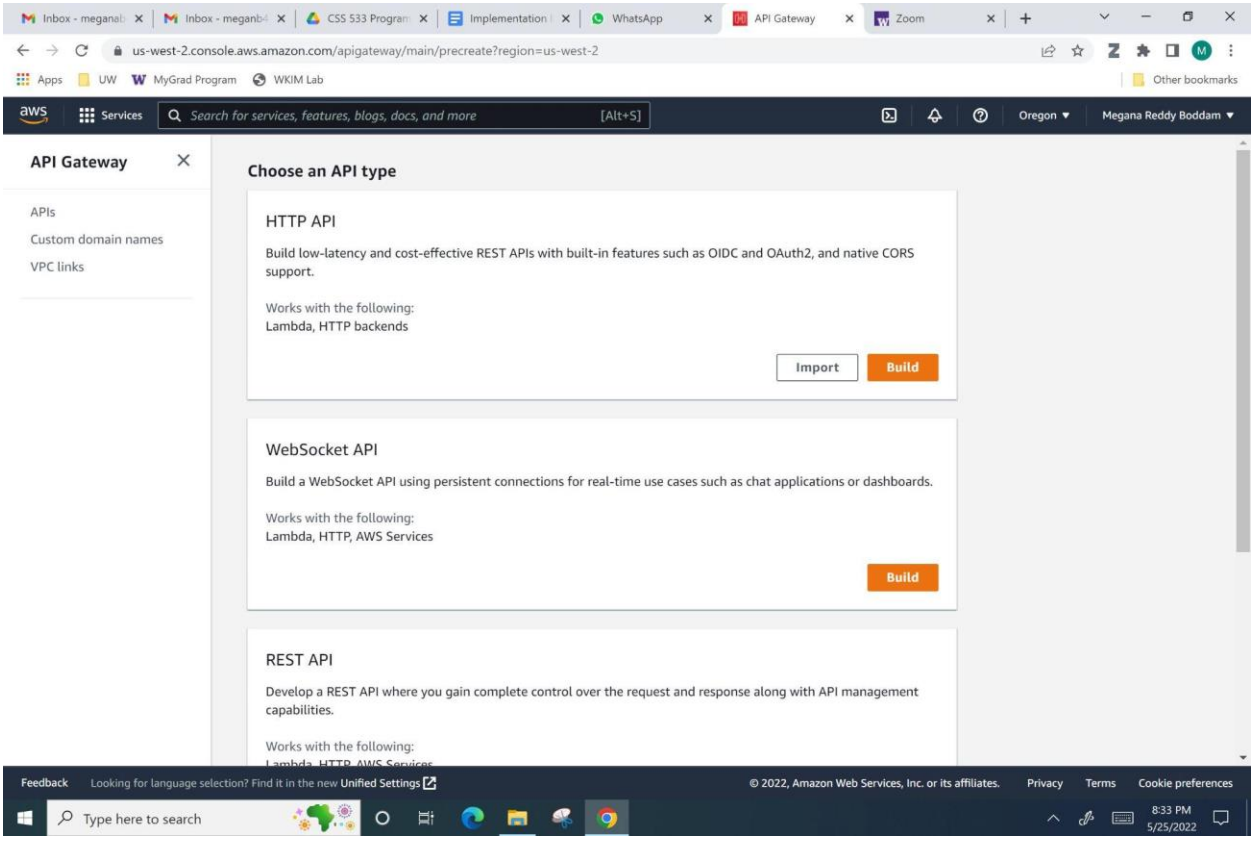


Figure 25: Choosing the "REST API" architecture to build APIs.

- Observe Figure 26. Choose the options “REST” for protocol, “New API” for next section, name API “MovieTickets”, choose the “Regional” option in “Endpoint Type”.

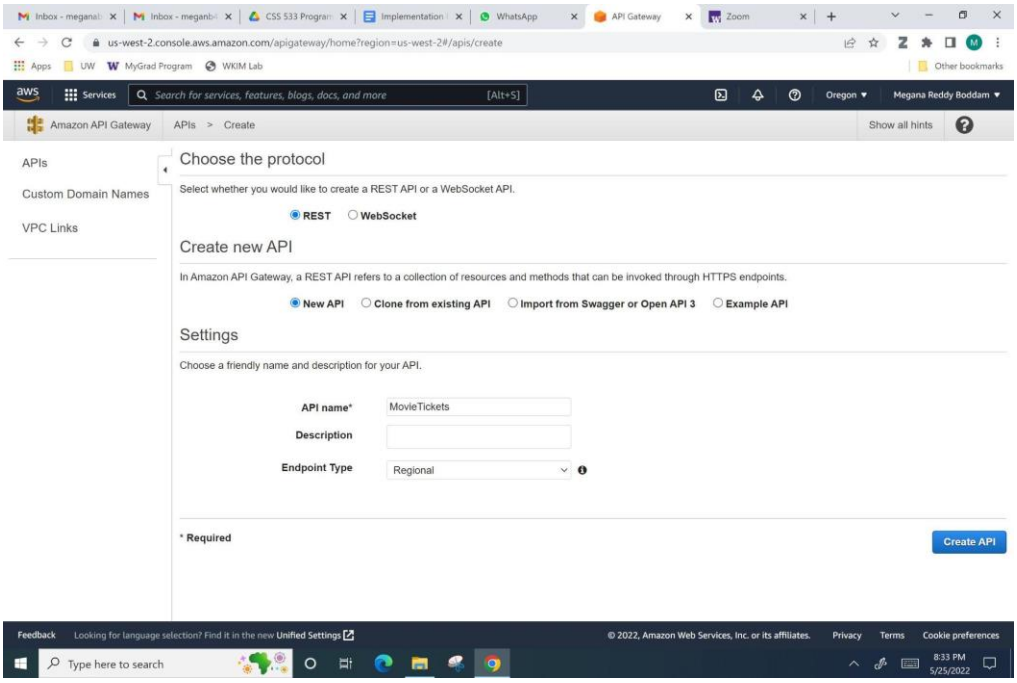


Figure 26: Initializing the API in the Gateway.

- Observe Figure 27. To reach the next stage described in Figure 26, click the “Actions” button. Choose “Create Resource”. Name the first API “addnewuser”. Enable the “API CORS Gateway”.

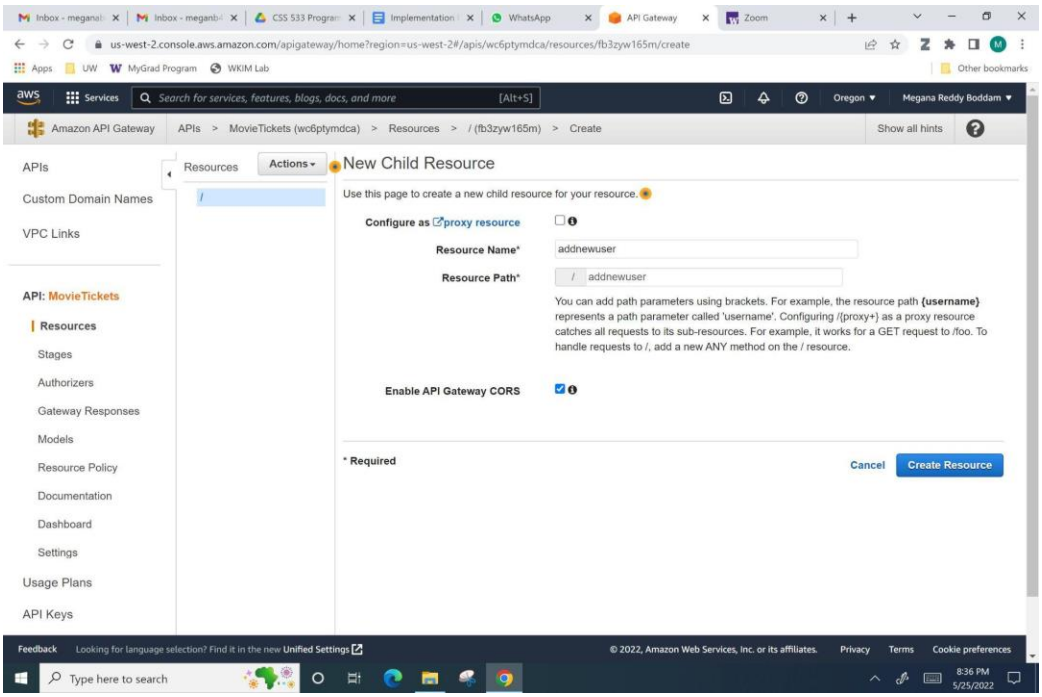


Figure 27: Create first API Resource.

- To reach Figure 28, select “Actions” again. Click “Create Method”. Choose “PUT”. Select the check mark.

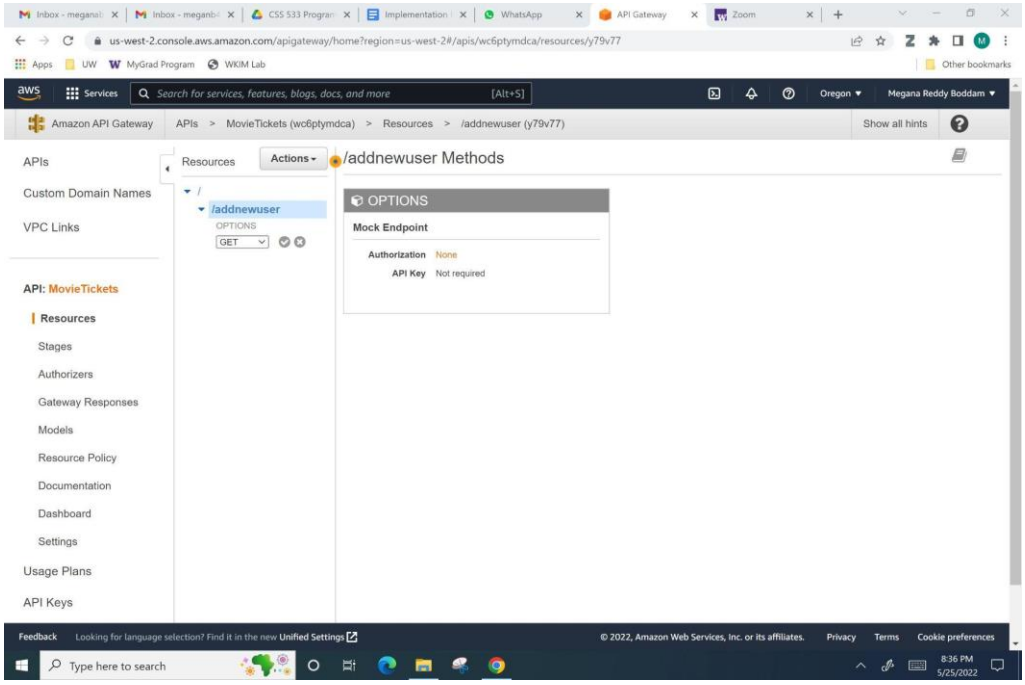


Figure 28: Creating an API method.

- Next, as shown in Figure 29, select “Lambda Function” for integration type, “Lambda Proxy Integration”, “Default Timeout”, and “SearchMoviesServer” for Lambda Function.

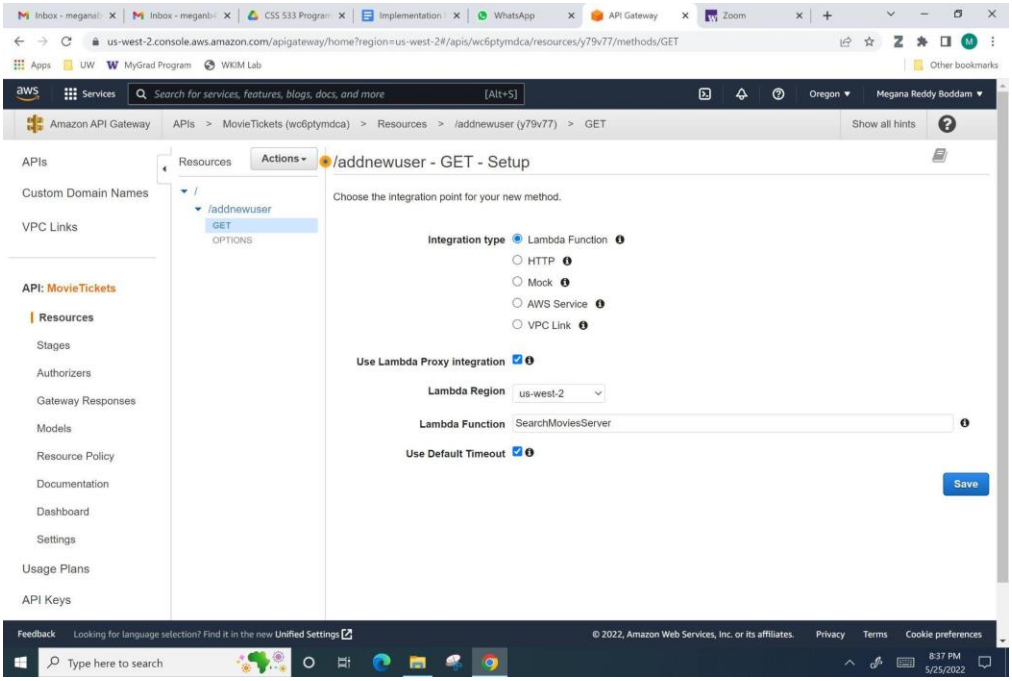


Figure 29: Configuring and Integrating the API method.

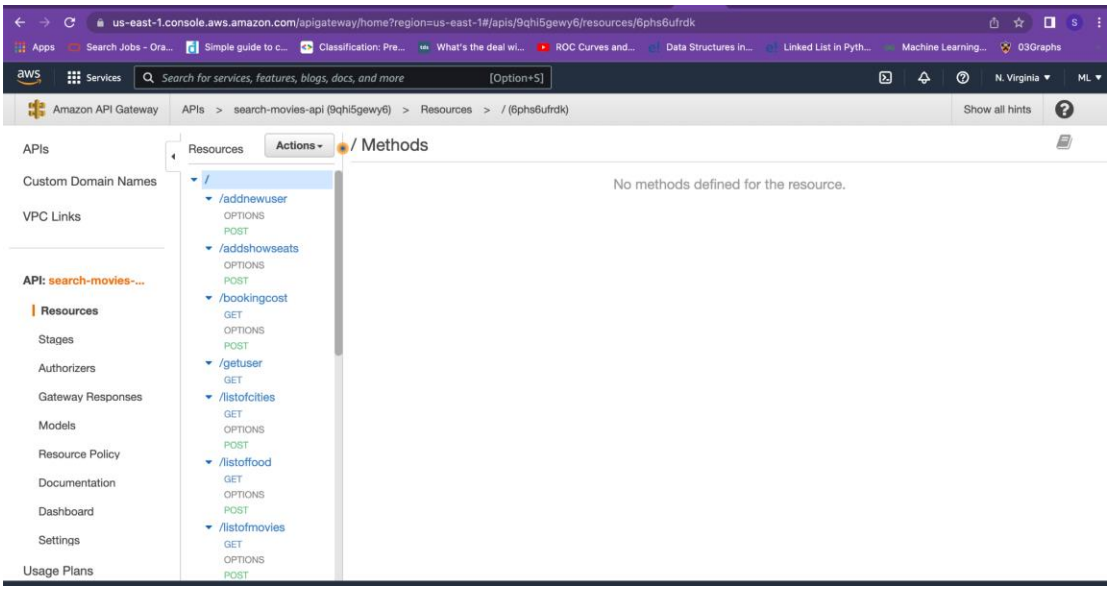


Figure 30: APIs configured in API gateway integrated with AWS Lambda.

- These steps create the barebones of one API.
- In this "MovieTickets" resource, we created 12 total API Resources, which had 1 or 2 associated methods. So, we repeated the above steps to achieve this.
    - addnewuser
        - POST method
        - Code in "SearchMoviesServer" lambda
    - getuser
        - GET method
        - Code in "SearchMoviesServer" lambda
    - listofcities
        - GET method
        - POST method
        - Code in "SearchMoviesServer" lambda
    - listoftheaters
        - GET method
        - Code in "SearchMoviesServer" lambda
    - listofmovies
        - GET method
        - Code in "SearchMoviesServer" lambda
    - listofshowdates
        - GET method
        - Code in "TicketBookingServer" lambda
    - listofshows
        - GET method
        - Code in "TicketBookingServer" lambda

    - listofshowseats
        - GET method
        - Code in “TicketBookingServer” lambda
    - addshowseats
        - POST method
        - Code in “TicketBookingServer” lambda
    - listoffood
        - GET method
        - POST method
        - Code in “FoodBookingServer” lambda
    - bookingcost
        - GET method
        - POST method
        - Code in “PaymentProcessingServer” lambda
    - listoftickets
        - GET method
        - Code in “PaymentProcessingServer” lambda
- We deployed these APIs in Amazon API Gateway, and this gave us a “https” URL to access our APIs and a security key. This helped us secure our resources in Amazon and be cost-efficient.
- The list of headers for our URL include “Content-type”, which is JSON, “Accept”, which is also JSON, and “x-api-key”, which is the previously mentioned security key.
- Both our integration tests and client interface, as described in below sections, use this URL and these headers to access the APIs.

Client Tier (Front-End)

- We created HTML, CSS, and Javascript files that access the movie ticket booking website and act as the interface between our APIs and the human user. Our CSS files reference the stylesheets developed in some open-source front-end examples found online - (2).
- We hosted these files in the “Prod” deployment stage (prod stands for production) on Amazon API Gateway.
- Clicking the below website should allow a customer to access our website and book a movie ticket.
    - https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/pages/home

- We have also hosted the web pages that front the business logic on AWS Lambda exposed through AWS API Gateway. On request for a page, the lambda interprets the page that is being requested, reads the corresponding HTML that was packaged with the lambda code and returns it to the client as a “text/html” response for display on the browser.

- Initially, the client manually navigates to the homepage by hitting the URL – https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/pages/home,

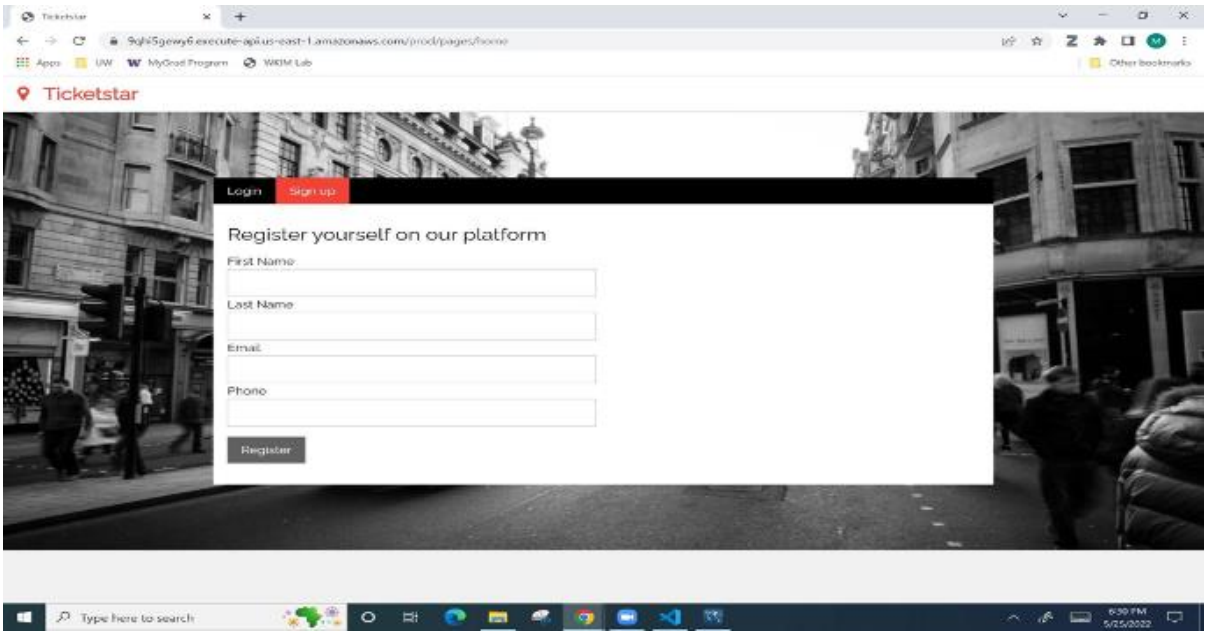


Figure 31: Frontend

From here, depending on how the user interacts with the browser UI, frontend code automatically issues various requests for pages to the API Gateway directly.

- To host this setup,

1. We first created the "frontend" lambda function and added all our HTML page artifacts to the lambda codebase.
2. Next, we added the "index.js" Node.JS file that will be in-charge of receiving the requests from the API Gateway, interpreting which page was requested, reading the corresponding HTML artifact and then finally serving the read HTML as response back to the user. Here is a screenshot of the developer view of the "frontend" lambda showcasing code for the main Node.JS file as well as the uploaded HTML artifacts,

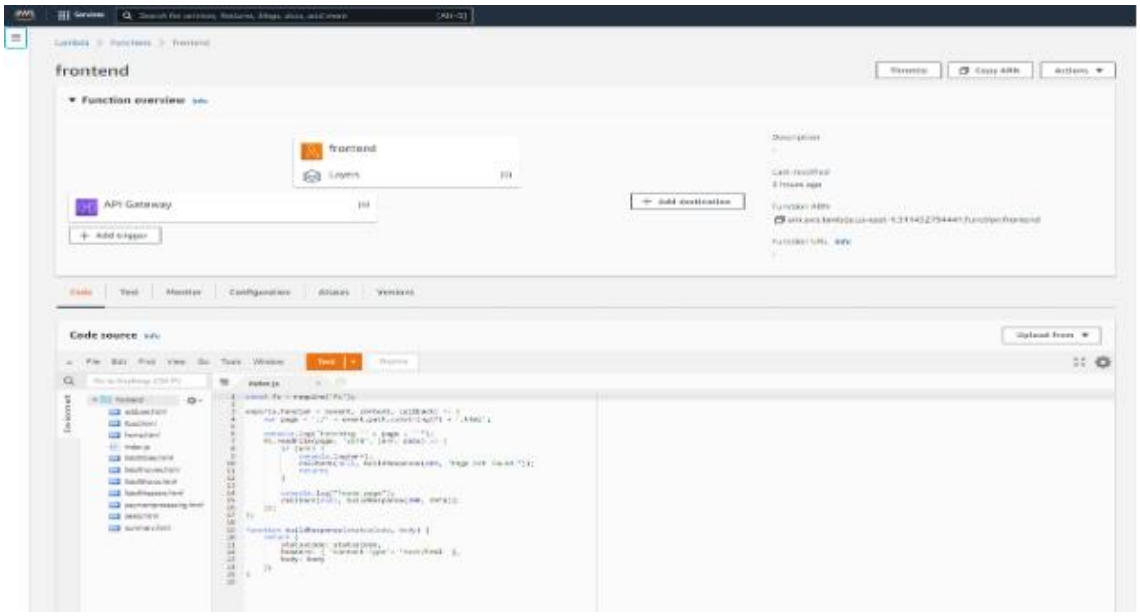


Figure 32: Frontend Lambda

3. Next, we created the API Gateway resource that will accept incoming requests for pages and forward it to the "frontend" lambda. We created this resource with a proxy resource so that it accepts all requests of the form "<API_GATEWAY_URL>/pages/<SUB_DOMAIN>" and forwards it to the lambda. The lambda would then extract the sub domain to identify the HTML artifact being requested. To do this,

   a. Add a "pages" resource to the root "/" API,

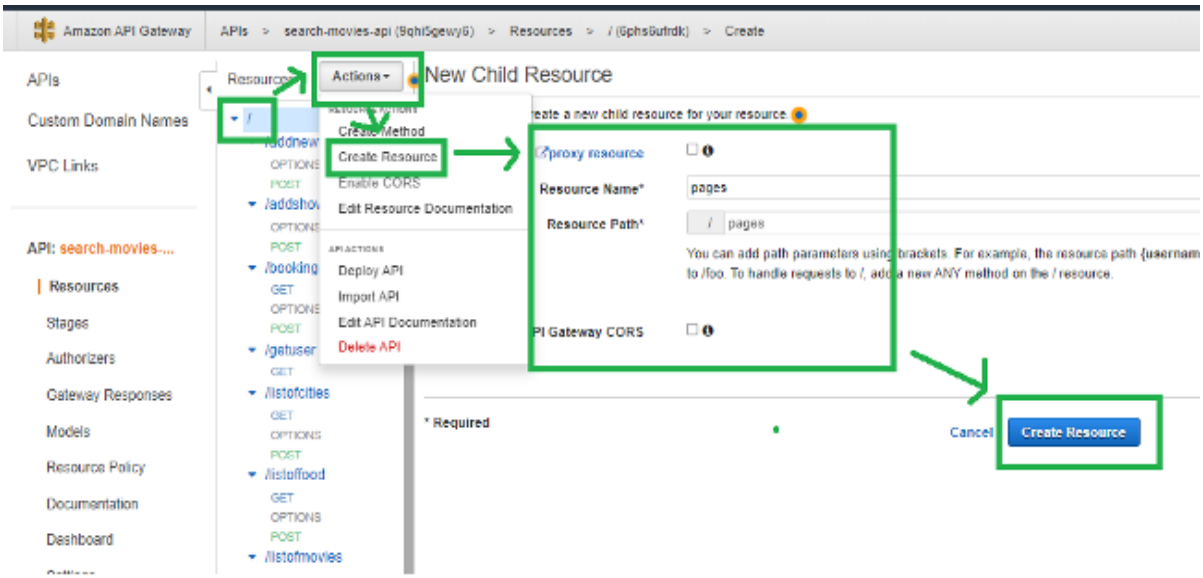


Fig 33: Configuring frontend on API gateway

b. Add a proxy resource under the “pages” resource,

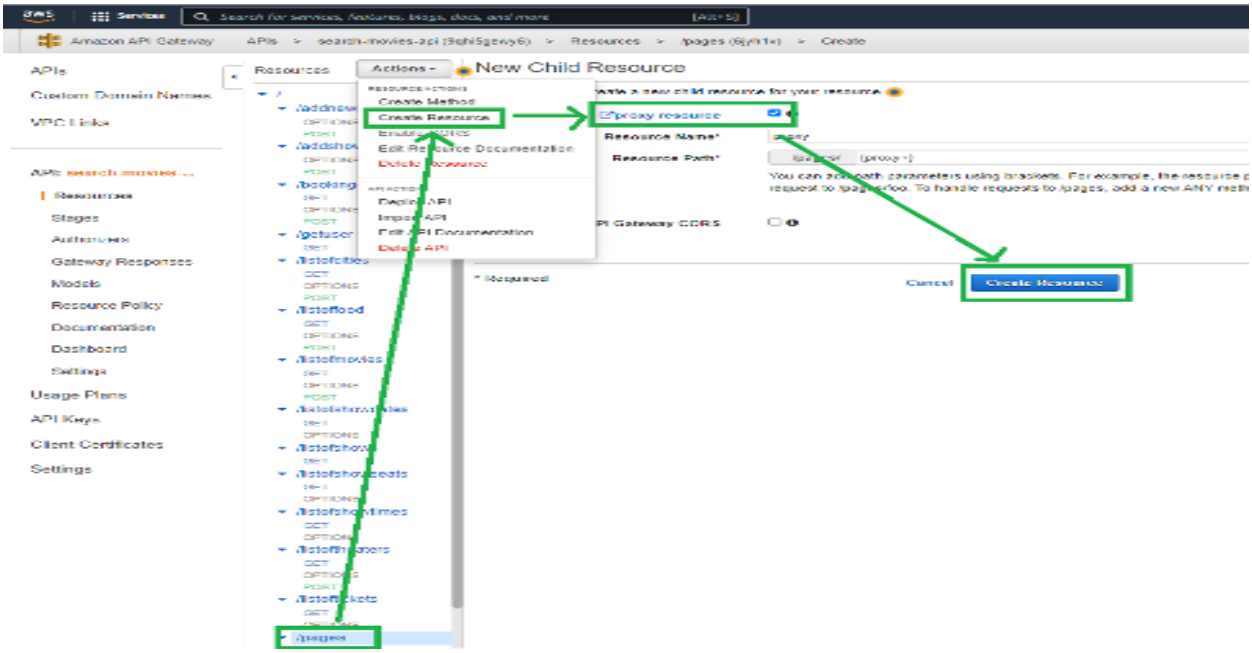


Fig 34: Configuring frontend on API gateway

c. Add a GET method under the proxy resource that performs a proxy integration with the “frontend” lambda we created earlier,

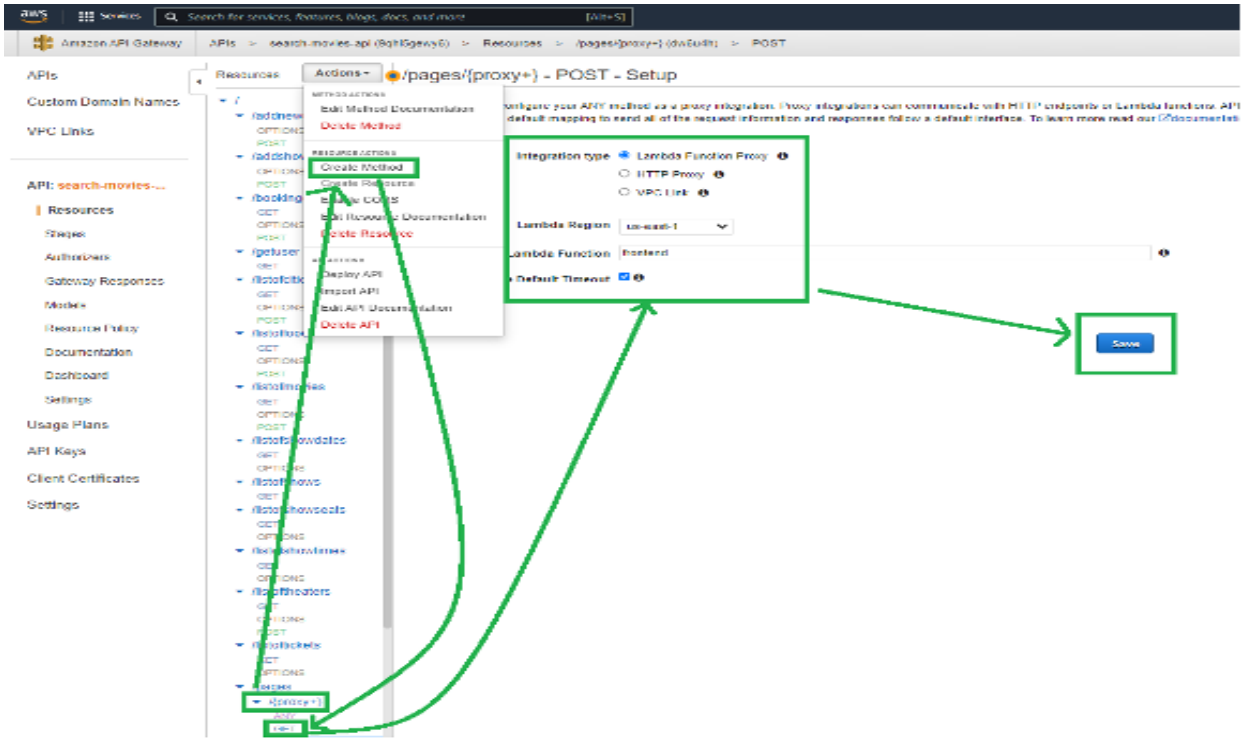


Fig 35: Configuring frontend on API gateway: Adding methods

Deploying the API gateway:

Once our Lambda function and API endpoints are integrated we need to deploy the API gateway to be publicly accessible. Please follow the steps to deploy the API gateway and add API key.

- Click on actions and then deploy API

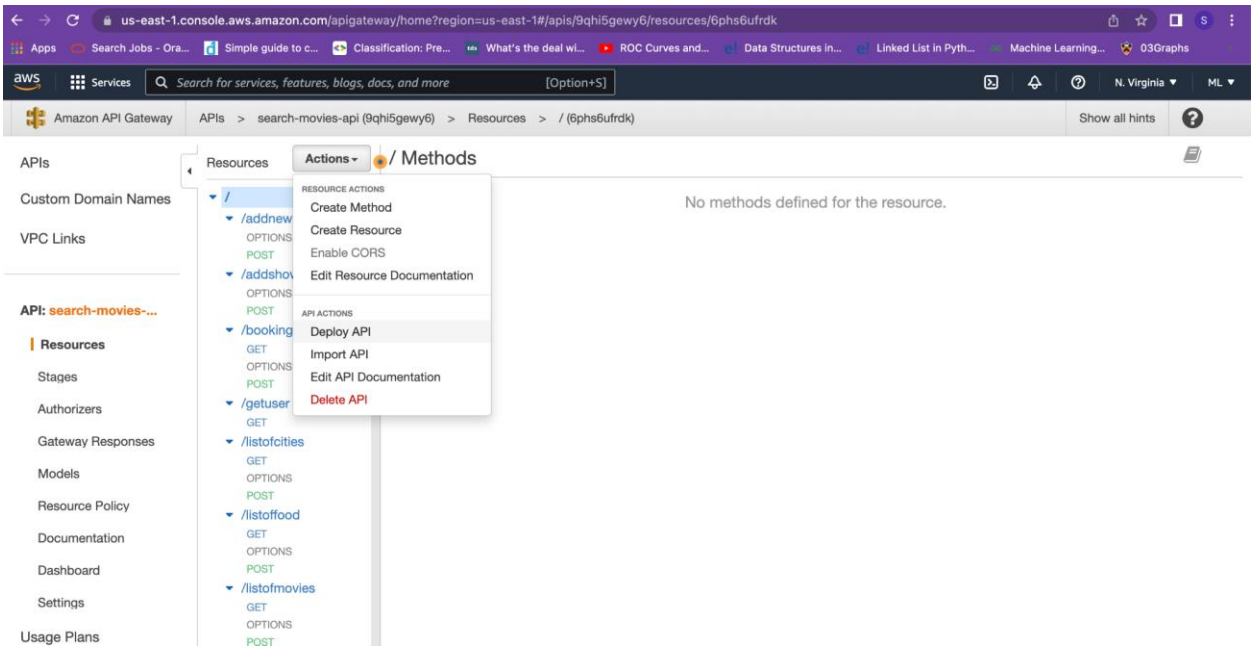


Fig 36: Deploying the API

- Create a deployment stage prod and the API is deployed

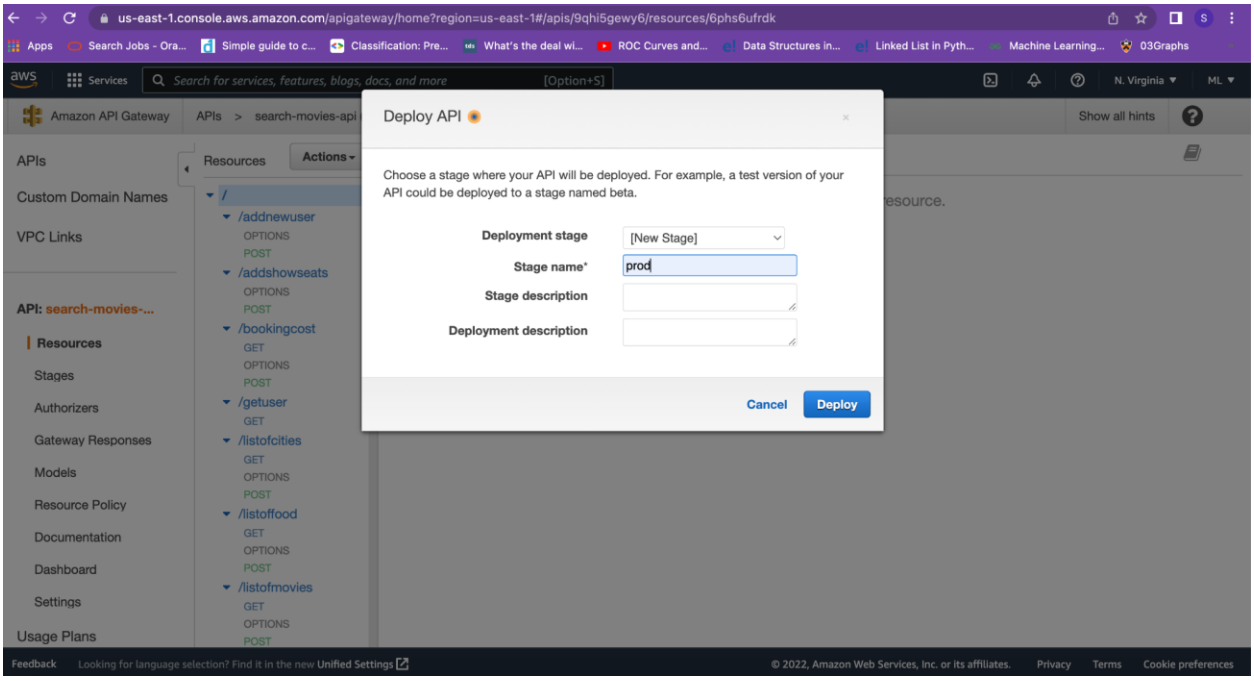


Fig 37: Creating deployment stage

- The deployment stage is now created.

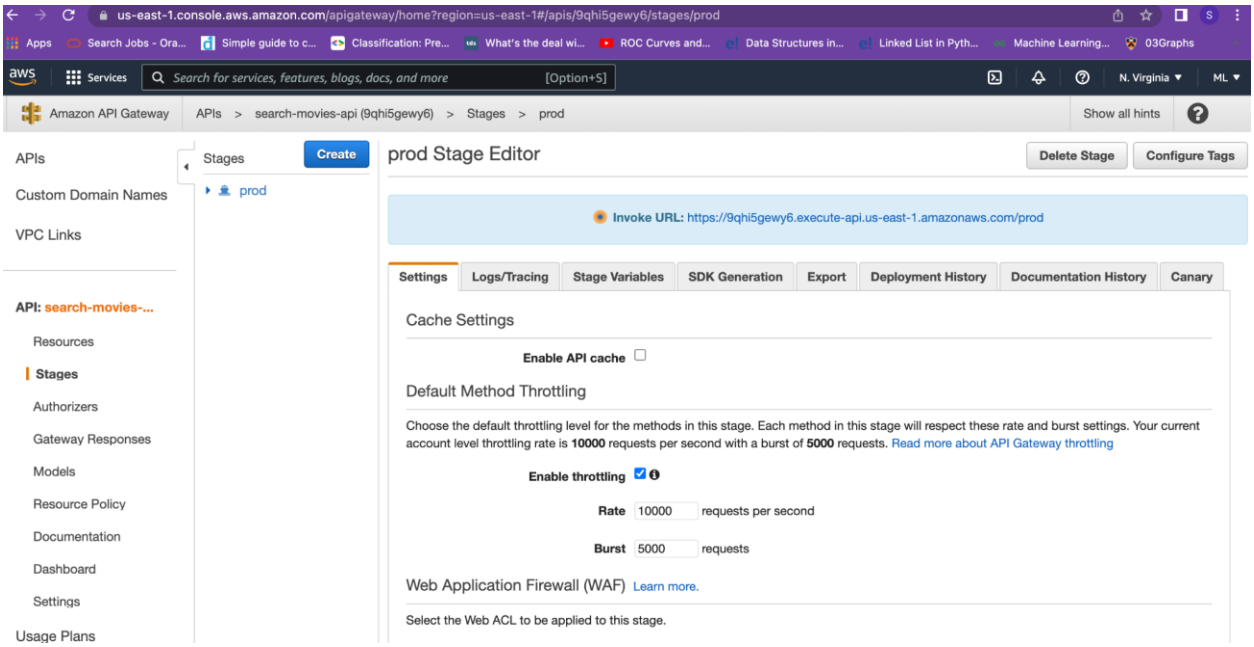


Fig 38 : API is deployed. Check the stages tab

- Add API keys by navigating to the API keys tab and clicking on actions-> create API

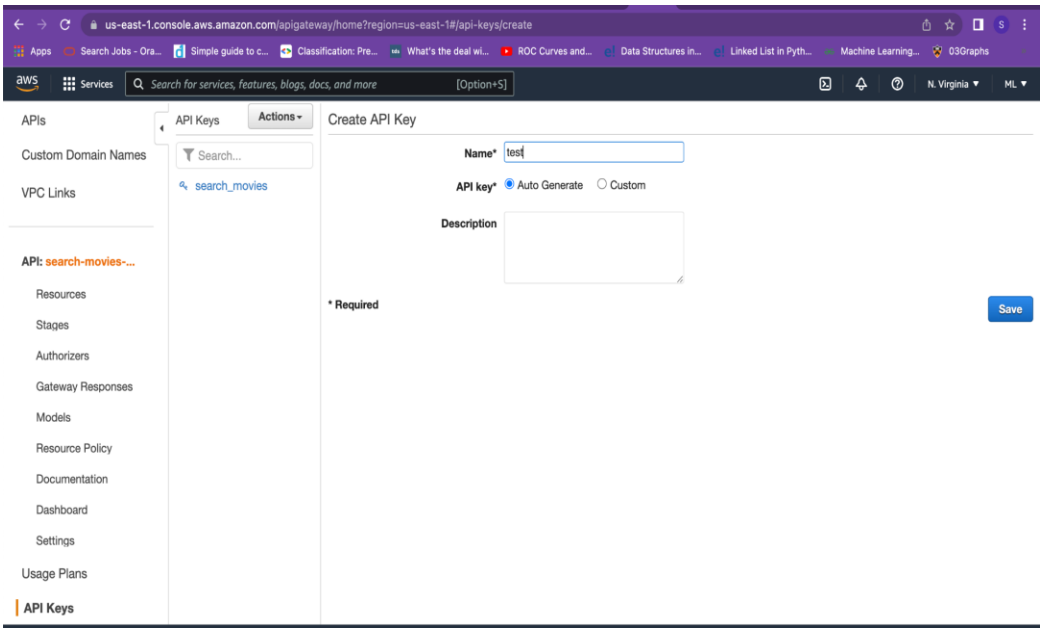


Fig 39 : Creating API keys

- The API key is now created

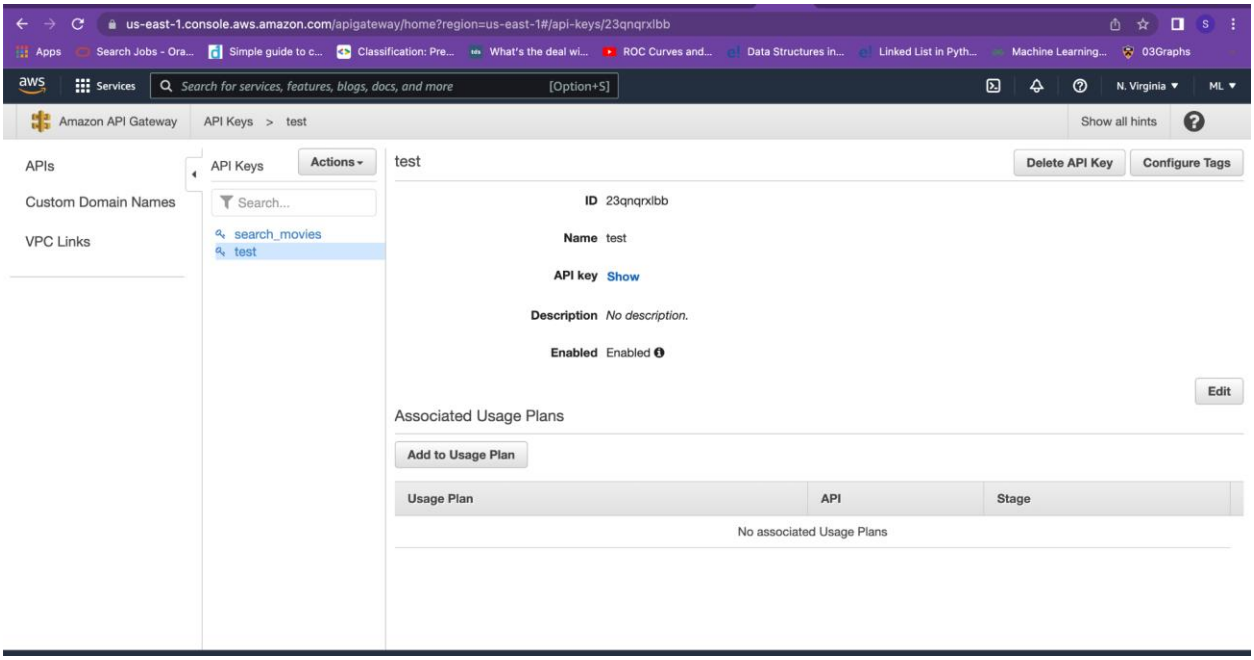


Fig 40 : Adding Usage Plan

- Click on the Add usage plan tab. Add appropriate throttling setting

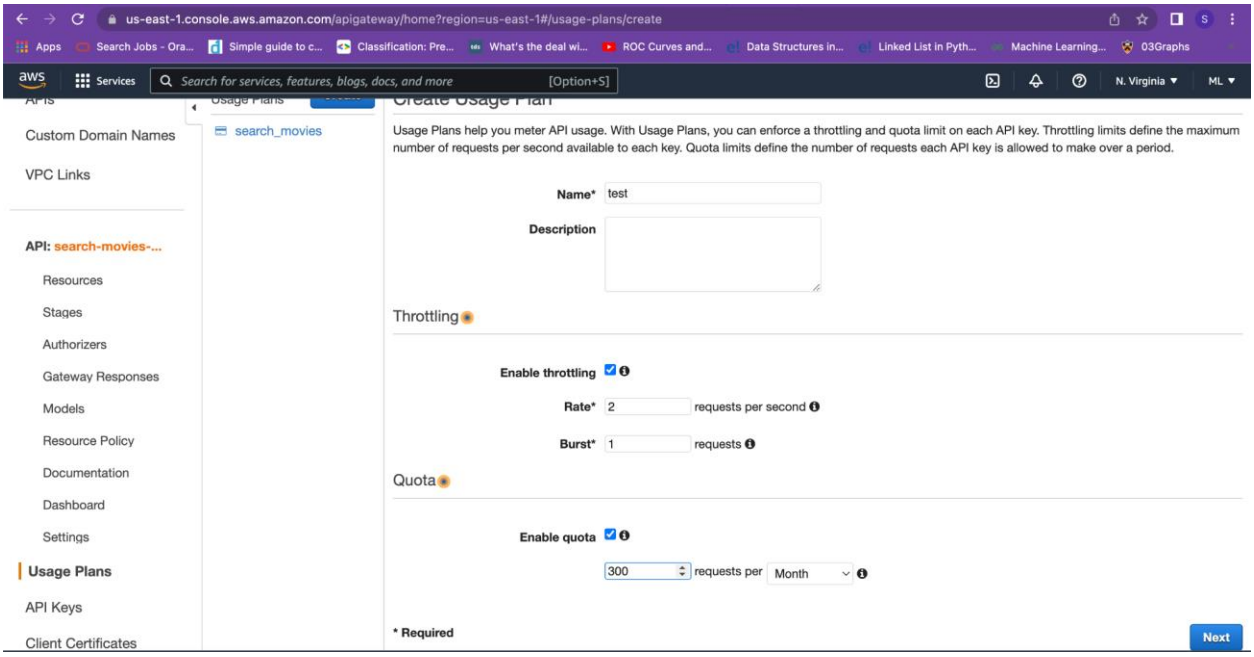


Fig 41 : Create the Usage Plan

- Associate the created API key with the deployed stage. Add the API key in the header x-api-key for every http request and response.

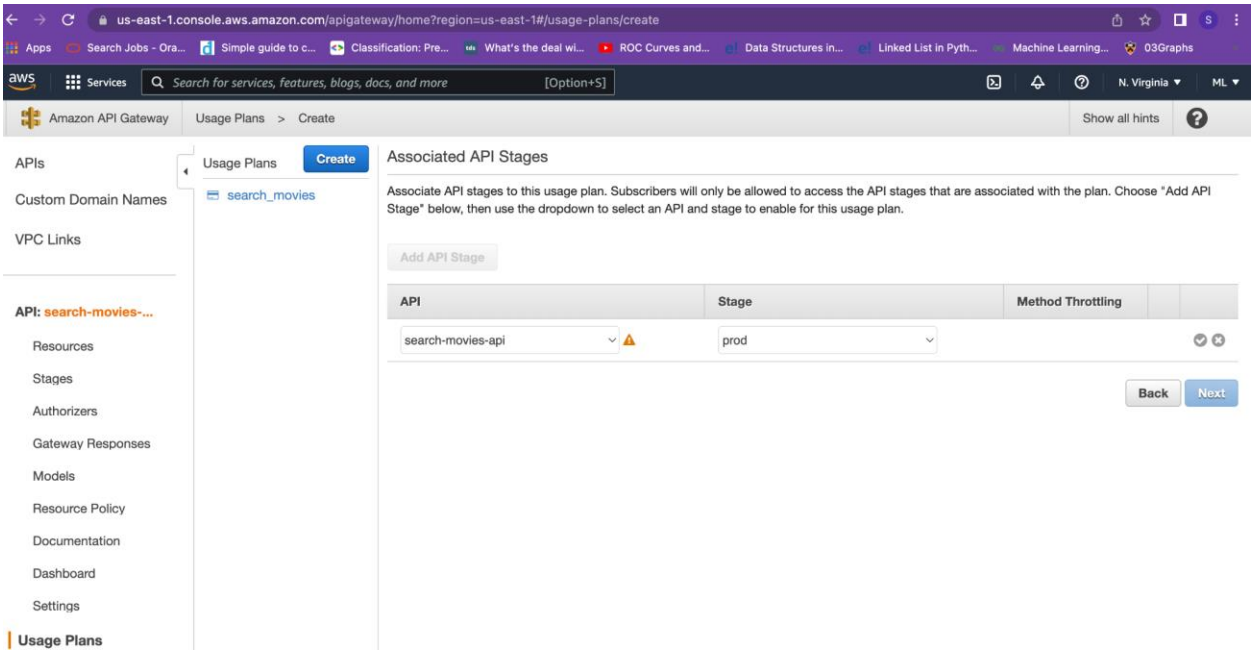


Fig 42 : Configure the API key with stages (deployment)

- Make sure to enable API Key as “true” for all theAPIs configured on the API gateway.

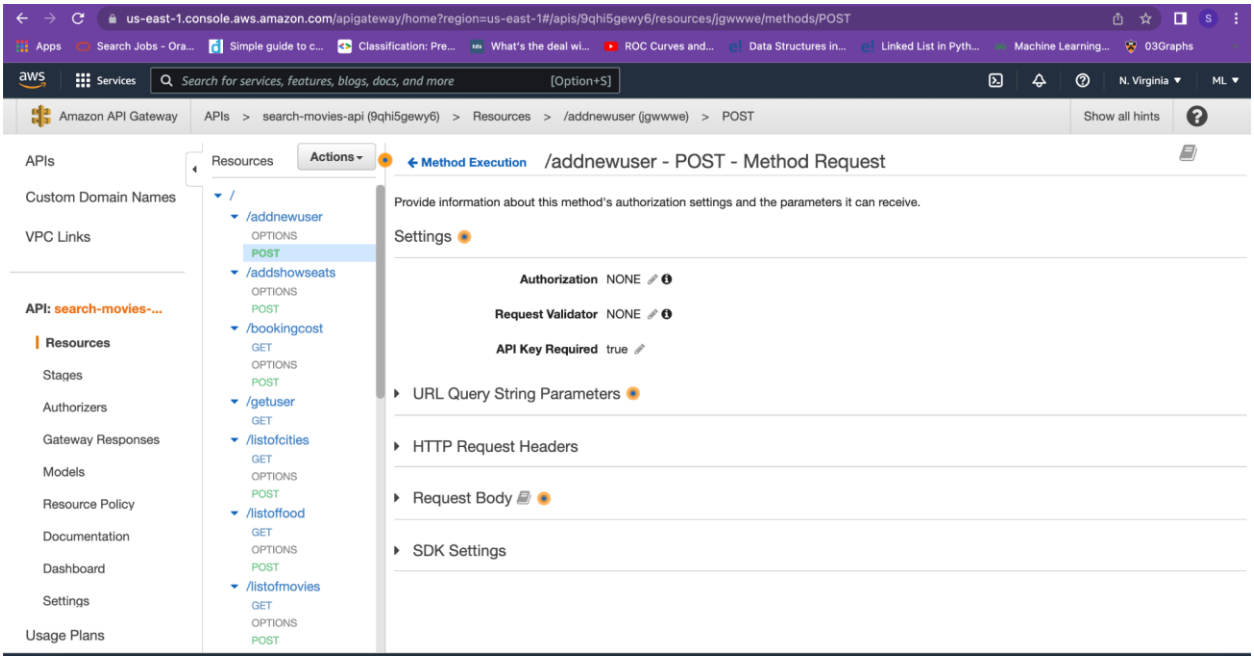


Fig 43: Enable the API key for each API method on API gateway.

Now the project is deployed on AWS and can be accessed.
Note: Please update the API keys with your newly configured API keys in the code if you are deploying the project on your own AWS account.

Integration Tests

- To accommodate some design requirements, we needed to modify some of our endpoints. We have written the integration tests in advance before we began our development effort.
- So, we needed to change our integration tests. We have resubmitted them.
- Please run the python scripts “search_movies_testsuite.py”, “ticket_booking_testsuite.py”, “food_booking_testsuite.py”, and “payment_processing_testsuite.py”**.**

References

1. https://dev.mysql.com/downloads/shell/
2. https://www.sourcecodeexamples.net/2020/12/movie-seat-booking-project-with.html
3. youtube.com/watch?v=Ut5CkSz6NR0&t=1270s
4. https://www.youtube.com/watch?v=V-ac_ZvdAW4