Project Documentation: Movie Ticket Booking System

Hrushikesh Adya (700749931)

Big Data Analytics and IT, University of Central Missouri

Readings in Computer Information Systems

Professor: Silvana Faja

Index

Project Overview:

The Movie Ticket Booking System will serve as a bridge between movie enthusiasts and the Ticket Selling Company through multiple theatre chains. A user will be able to see the available shows in the theatre and book a movie show. The admin will have access to control the available movies on the platform. All the ticket booking and selling details will be stored in the database for multiple use cases.

Technology Stack:

Front End Development:

- Scripting Language: HTML, CSS
- Programming Language: Typescript
- Framework: React JS

Back End Development:

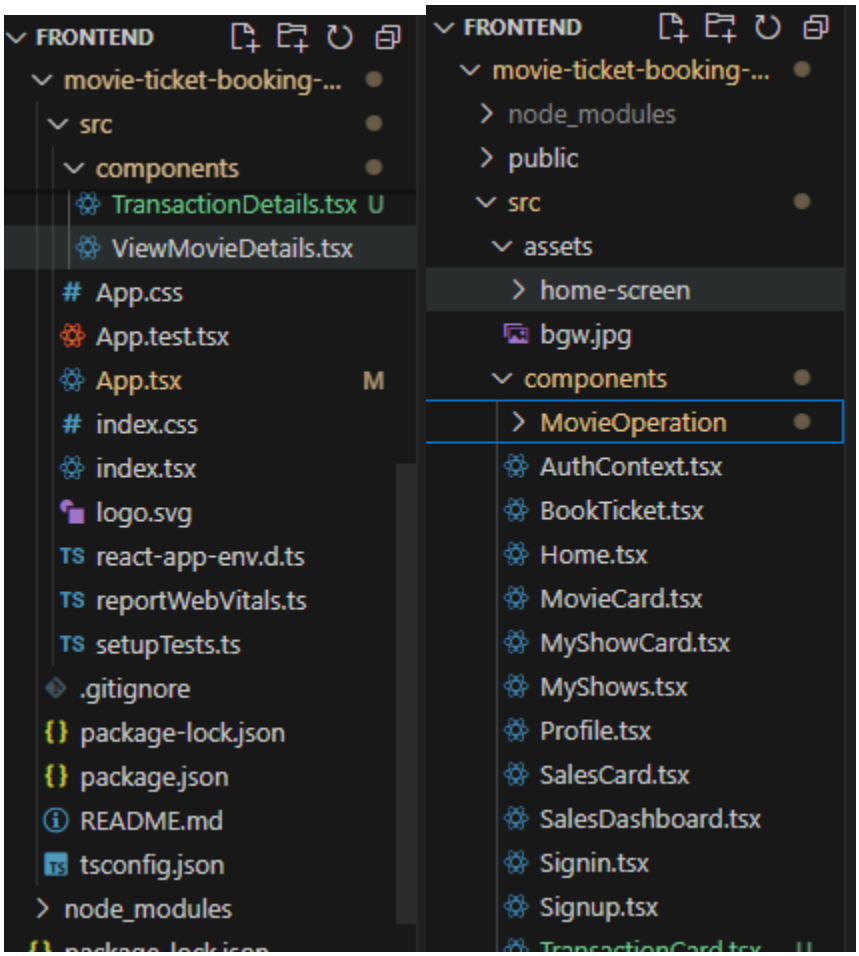
- Programming Language: Python

Cloud Platform:

- Amazon Web Services (AWS)

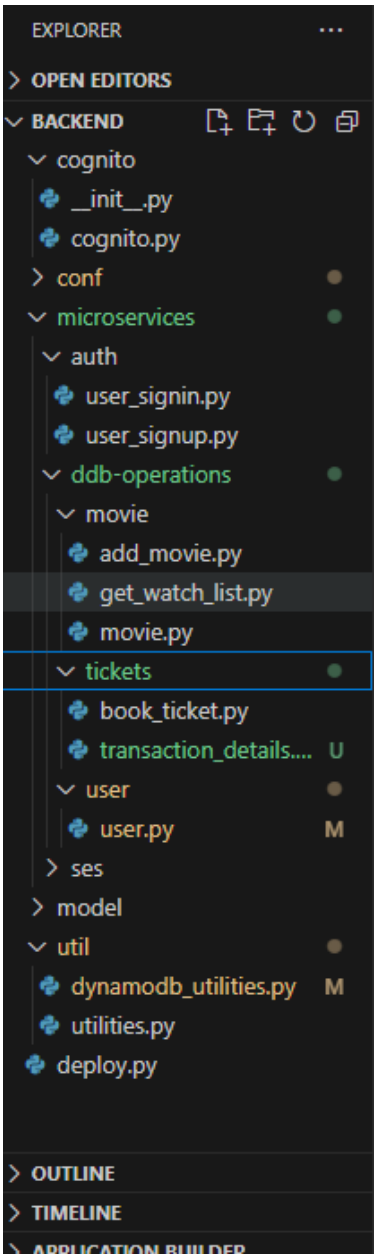
Folder Structure:

Front End Folder Structure:



Above screenshots show folder structure for front end code development done with React JS Framework.

Back End Folder Structure:



Here is the folder structure for backend code development done with Python and AWS boto3 SDKs.

Technical Implementation Details:

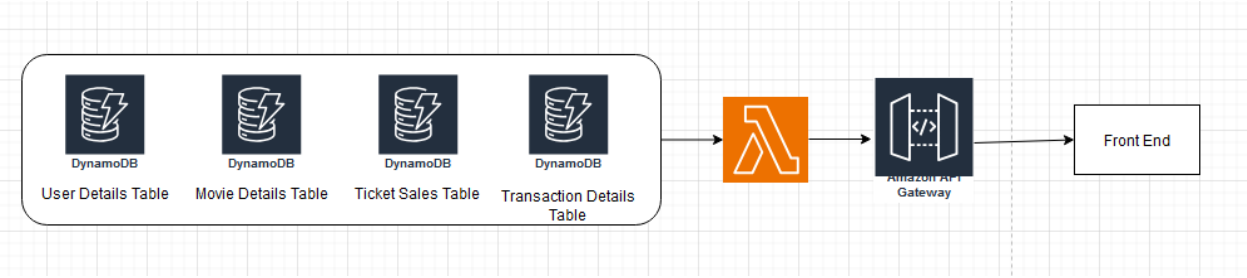
Movie ticket booking system app is developed using React JS Framework, HTML, CSS, and Python for backend development with boto3 AWS SDKs.

For every screen in the UI where data needs to be pulled from the backend or database, we are calling HTTP API using fetch () function with desired HTTP method. The response is carefully filtered and showcased into UI.

This API was created by using AWS API Gateway Service. API Gateway service in AWS allows you to create an API Endpoint which can be used to link other AWS services to third party applications.

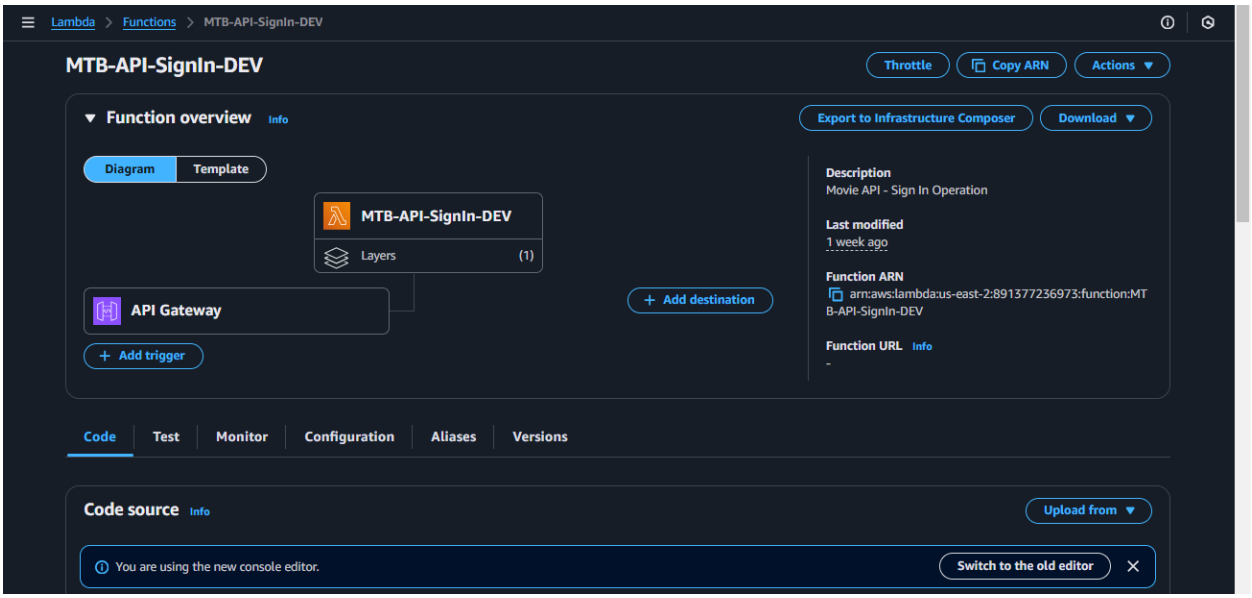
In our case, we are using AWS Lambda function written in Python behind every API which performs multiple DB operations on DynamoDB Tables.

AWS Infrastructure Details:

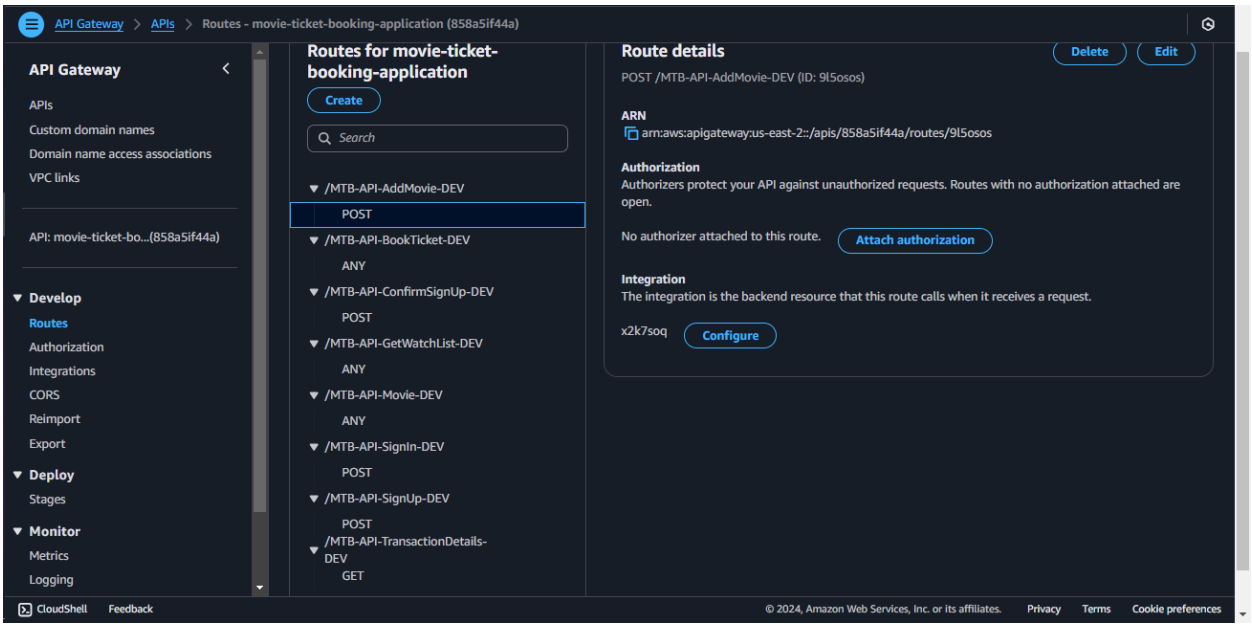


All the details are stored in DynamoDB Tables as shown in the above architecture diagram. These DynamoDB tables are linked with Lambda Functions which are triggered by APIs created with API Gateway service.

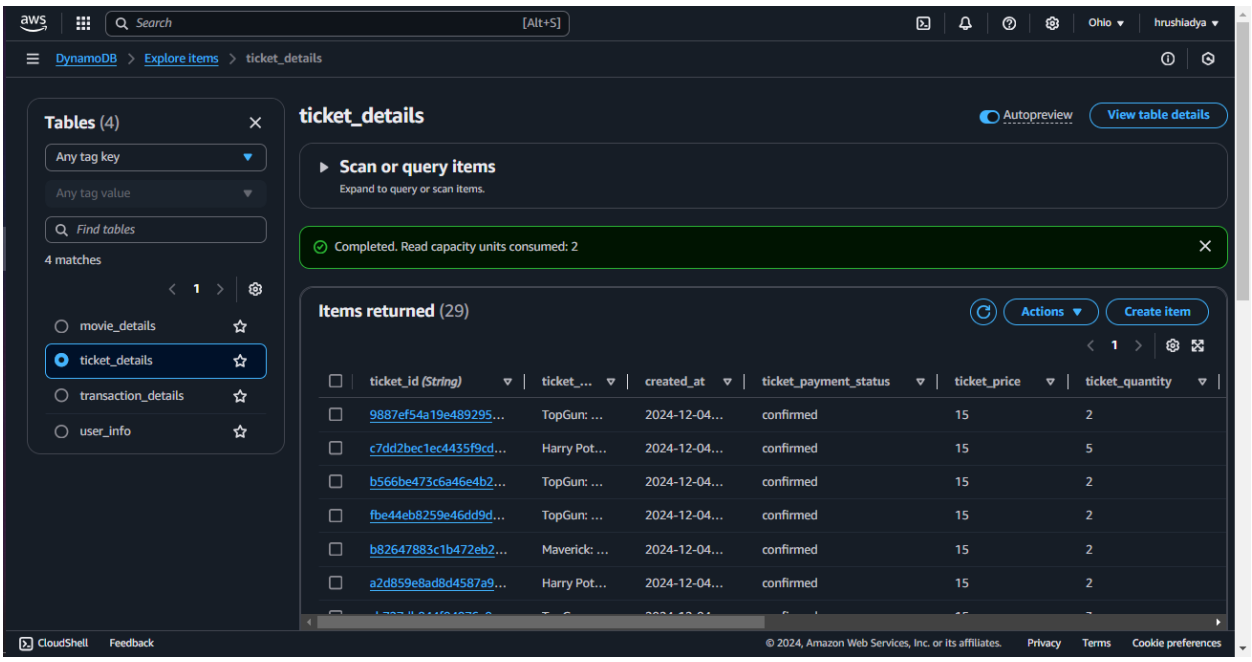
Lambda Function Integration with API Gateway:



API Gateway Bundles:



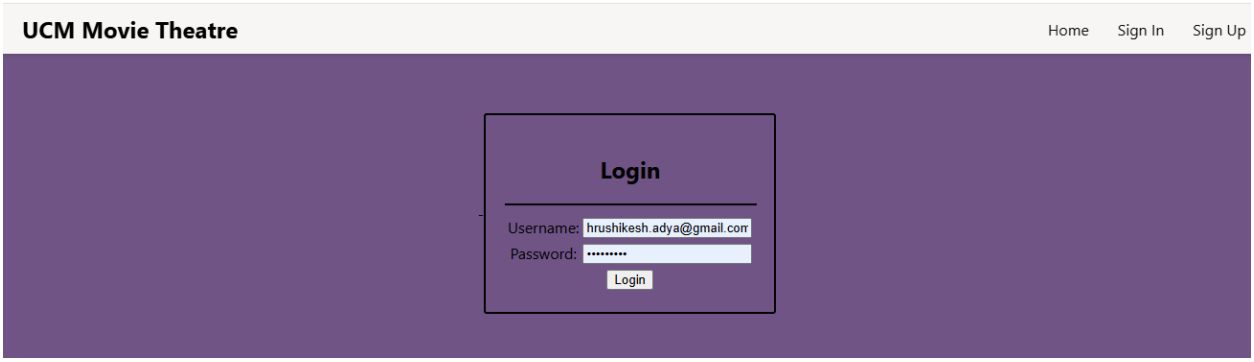
DynamoDB Tables:



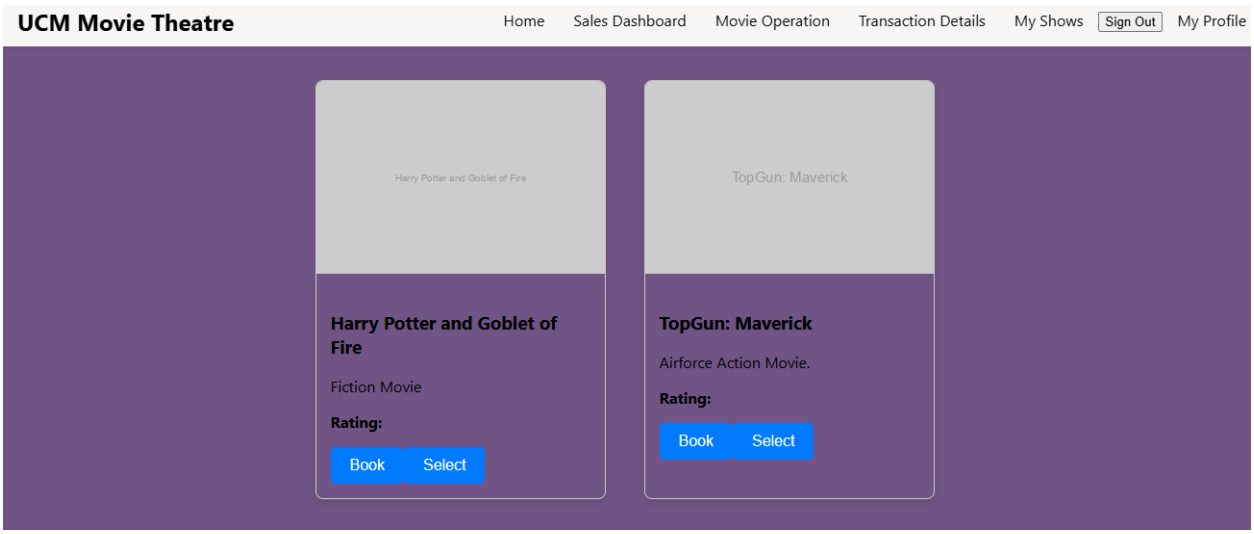
Screenshots of User Interface:

Sign Up Page:

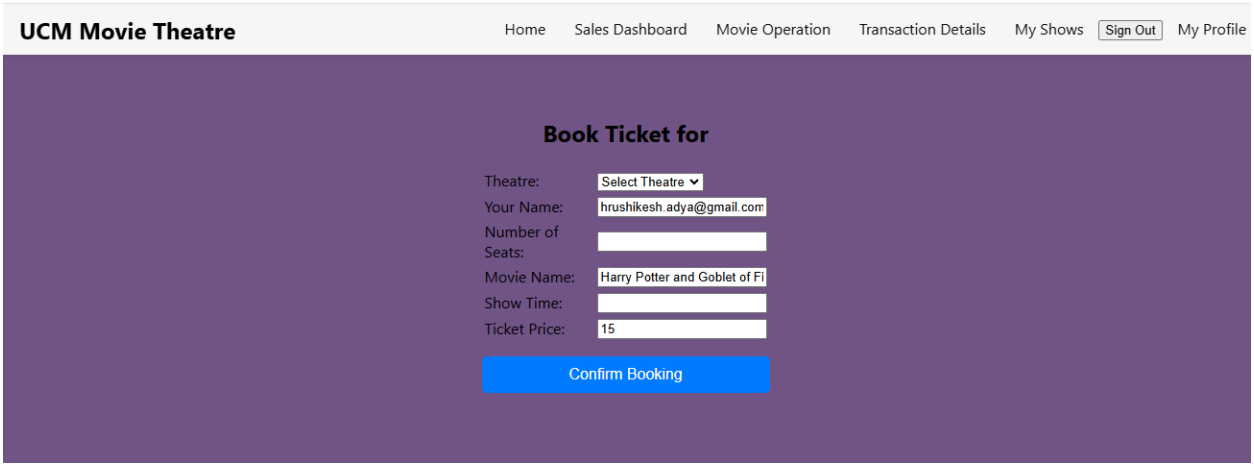
Sign In Page:



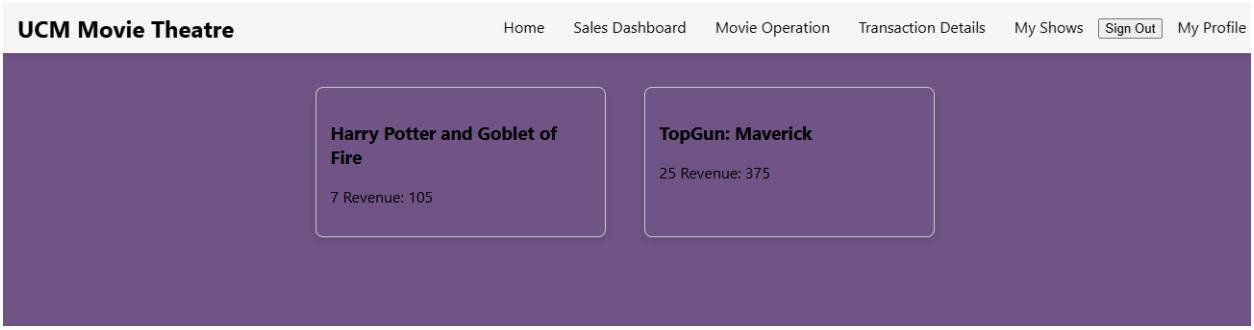
Home Page:



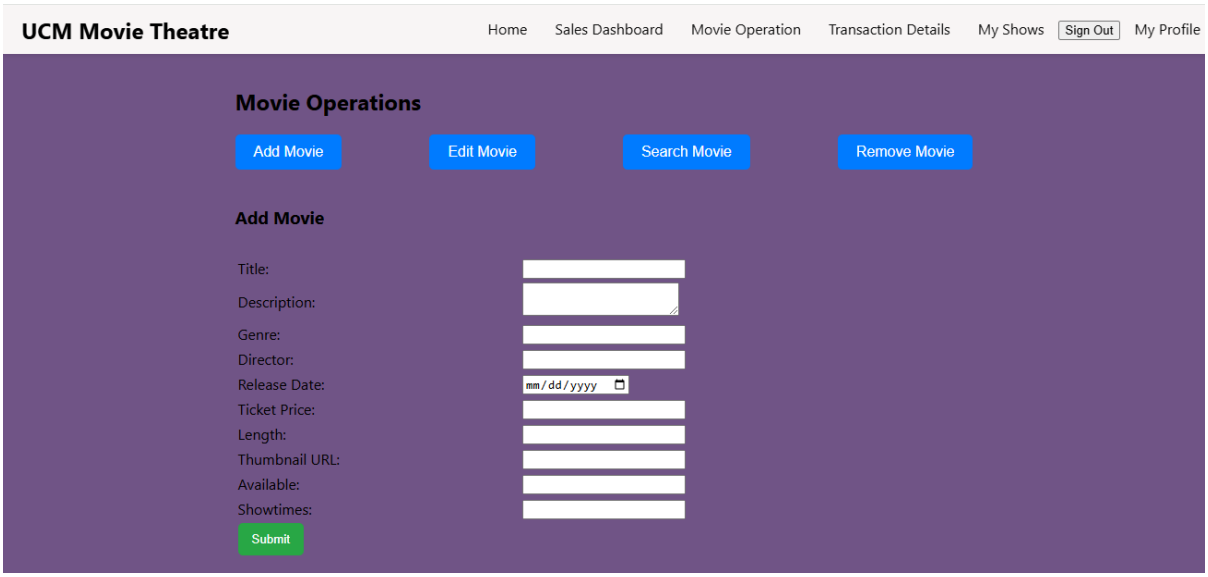
Book Ticket Page:



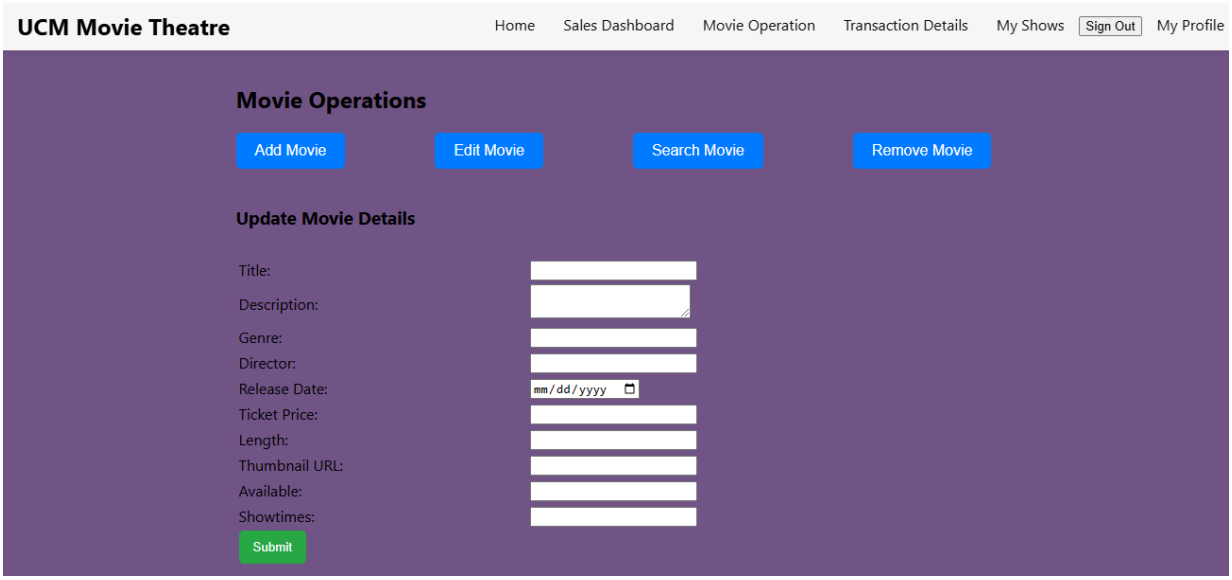
Sales Dashboard:



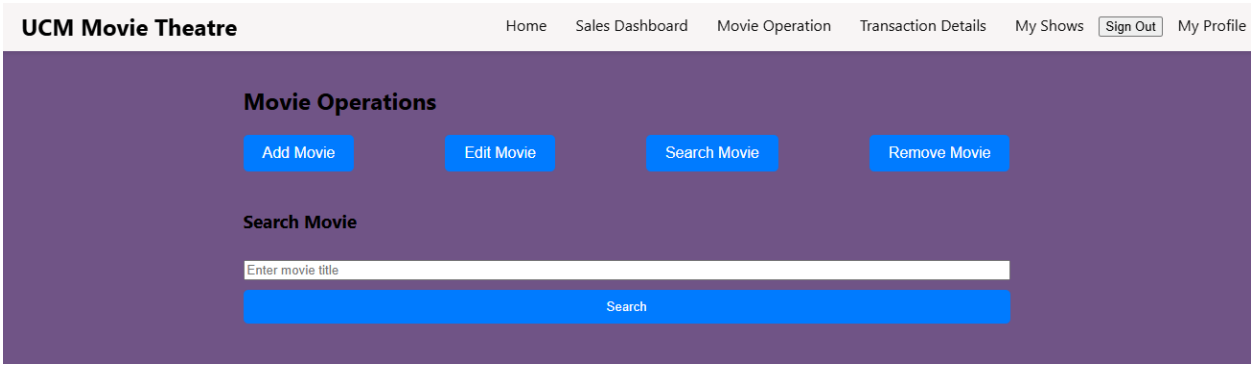
Add Movie Operation Page:



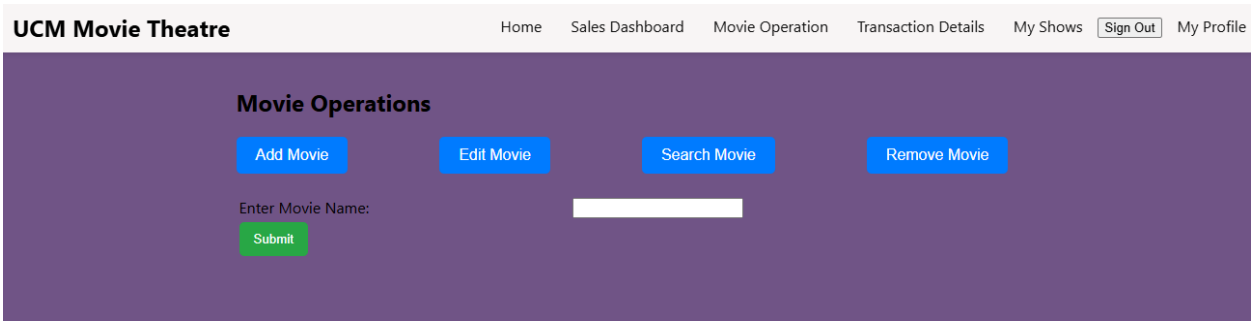
Update Movie Operation:



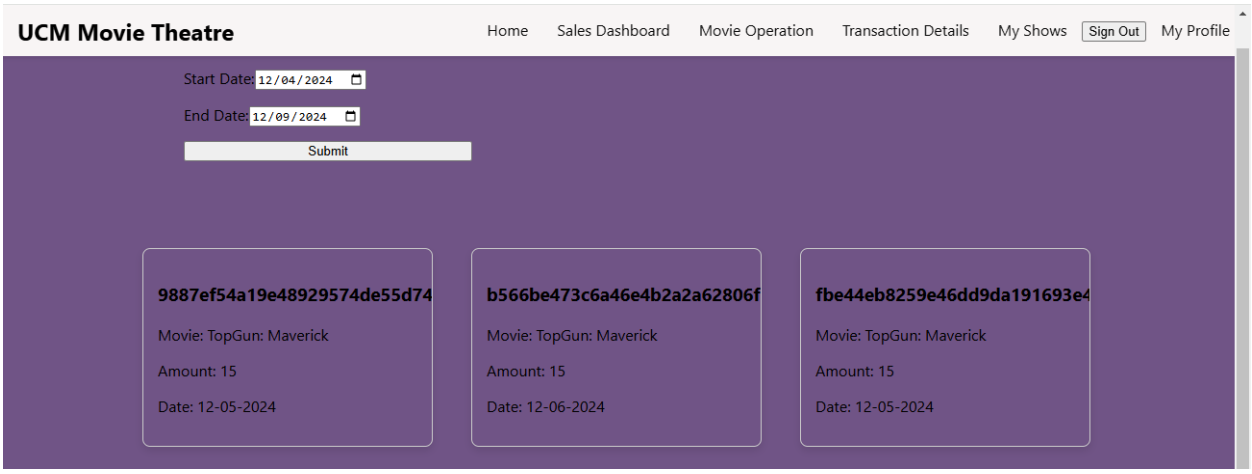
Search Page:



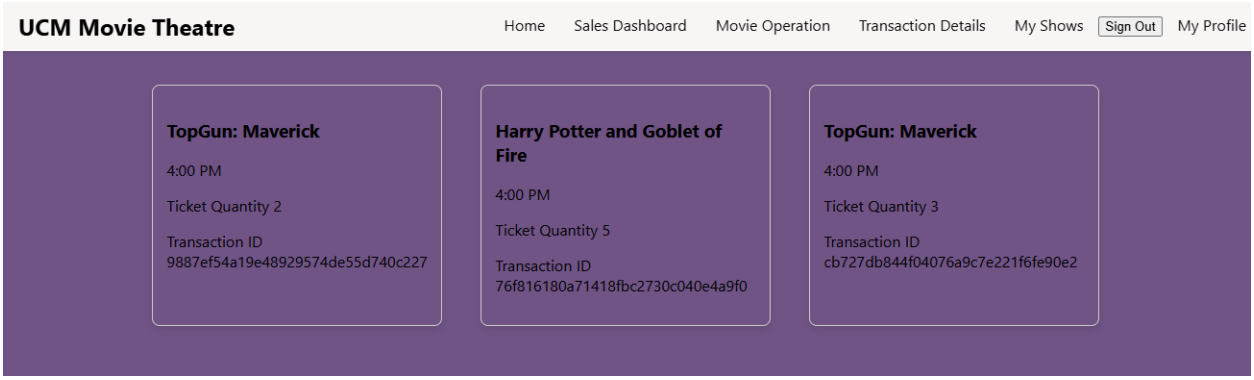
Delete Movie Page:



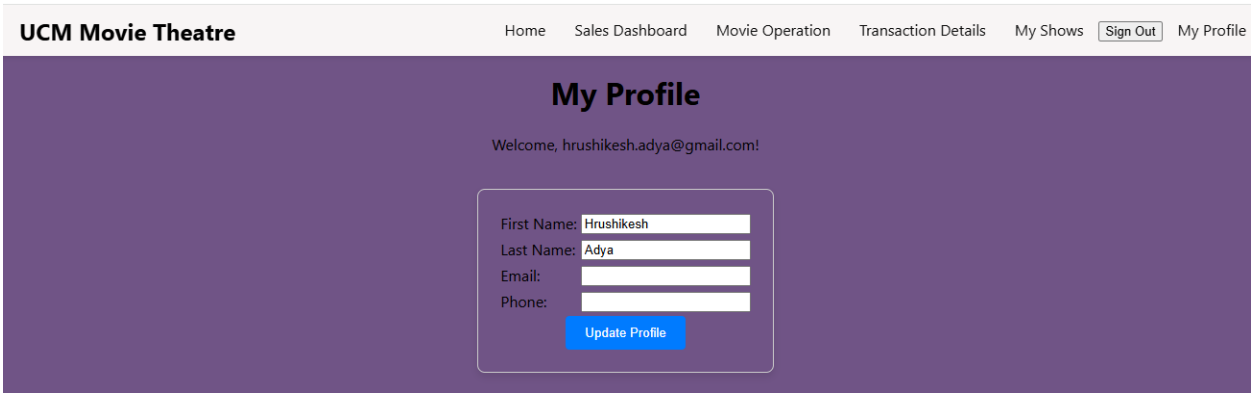
Transaction Details between two dates feature:



My Shows Page:

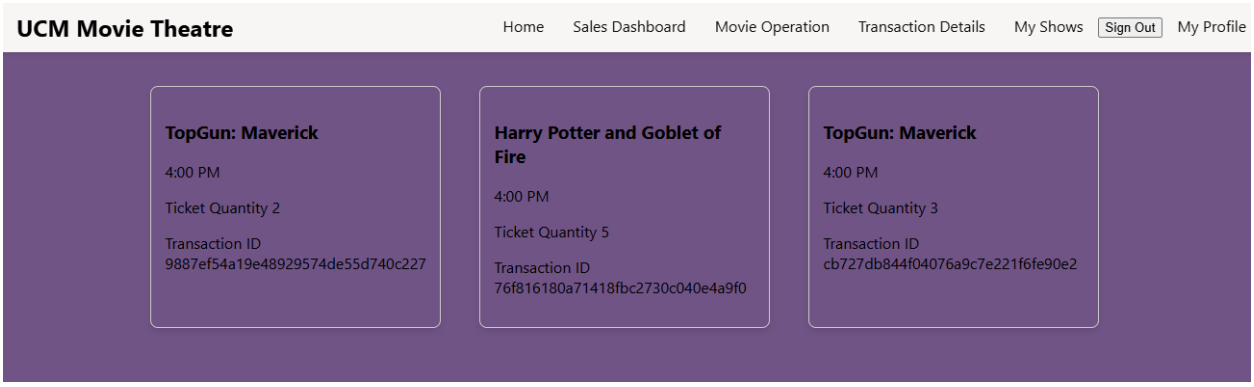


My Profile Page:



Code Explanation:

In this section we will explore functionality of one API from front end to back end.



My shows tab shows the user all the shows user has booked on movie ticket booking system.

```tsx
const MyShows: React.FC = () => {
    const { isLoggedIn, username } = useAuth();
    const location = useLocation();
    const [responseMessage, setResponseMessage] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [errorMessage, setErrorMessage] = useState<string | null>(null);
    const navigate = useNavigate();

    const apiGatewayUrl = 'https://858a5if44a.execute-api.us-east-2.amazonaws.com/dev/MTB-API-GetWatchList-DEV';
    const [movies, setMovies] = useState<any[]>([]);

    console.log("Username: ", username);
    const fetchData = async () => {
        try {
            const response = await fetch(`${apiGatewayUrl}?user_id=${username}`, {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json',
                },
            });

            const data = await response.json();
            console.log('Data:', data);
            if (response.ok) {
                console.log('Data:', data);
                setMovies(data);
            } else {
                setErrorMessage(data.message || 'Failed to get WatchList from API.');
            }
            console.log("MOVIEE")
```

In MyShows.tsx file we have used GetWatchList API which is returning a list of shows watched by logged in user.

```tsx
    return (
        <div>
            {isLoggedIn ? (
                <div>
                    <div style={gridStyle}>
                        {movies.map((movie, index) => (
                            <MyShowsCard
                                key={index}
                                title={movie.movie_name}
                                show_date={movie.show_date}
                                ticket_quantity={movie.ticket_quantity}
                                transaction_id={movie.transaction_id}
                            />
                        ))}
                    </div>
                </div>
```

The data returned with API is showcased in front end UI using HTML and CSS.

```python
def lambda_handler(event, context):
    print("Event: ", event)
    if 'httpMethod' not in event:
        raise RuntimeError('No HttpMethod')
    logger.info("Event:")
    logger.info(json.dumps(event))
    http_method = event["httpMethod"]

    try:
        if http_method == "GET":
            if event['queryStringParameters'] is not None:
                user_id = event['queryStringParameters']['user_id']
                print("user_id ", user_id)
                user = get_user_by_key(user_id, user_id)
                print("user: ", user)
                response = user['watch_list']
                print("response: ", response)
                return {
                    'statusCode': 200,
                    'body':  json.dumps(response, cls=DecimalEncoder)
                    }
    except JSONDecodeError as e:
        raise JSONDecodeError('Error when decoding json body', inner=e)
    except Exception as e:
        raise Exception('Error when performing GET API', e)
```

This is the backend lambda function code for get_watch_list functionality which is helping us get the users watch list from DynamoDB table.