CS 443 Cloud Computing

Project Midpoint Milestone

Bilgehan Akcan 21802901
Miray Ayerdem 21901987
Ferhat Korkmaz 21901940

Contents

Section 1 - Project Overview

1.1 Project Name and Team

The project name is Veni Vidi Movie. The team name is Veni Vidi Cloud. The team consists of the following students:

- Bilgehan Akcan 21802901
- Miray Ayerdem 21901987
- Ferhat Korkmaz 21901940

1.2 Purpose

The project aims to present an online movie ticket purchase system, which will be a web application utilizing Google Cloud services. Through a user-friendly interface, users will be able to purchase tickets for movies in a particular movie theater, offering a range of movie sessions. Veni Vidi Movie, which is planned to be a platform that will facilitate the process of movie selection and ticket purchase with its periodically updated movie list, aims to offer a solution to everyone who likes to watch movies and thinks of going to the cinema.

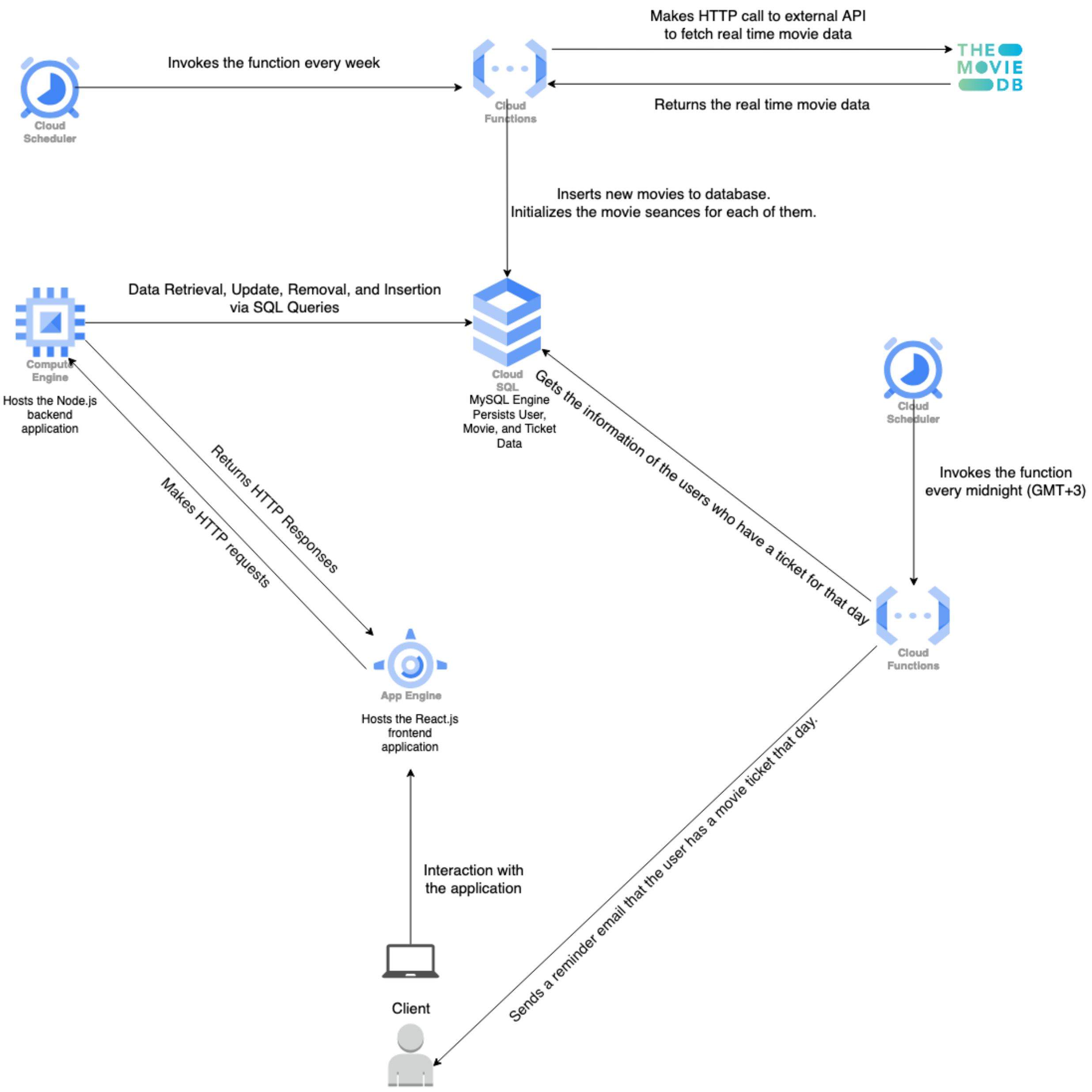
1.3 Scope

The scope of the system includes the features and functionalities required to enable the process of purchasing a movie ticket. Veni Vidi Movie will allow users to view all the movies and their showtimes currently showing in theaters. It will enable them to select a preferred seat. The system will be integrated with a movie database (TMDb) such that movies offered on the application will be updated periodically. Also, it will send an auto-generated confirmation email when a ticket is purchased and a reminder email when the date of the movie that a ticket is purchased is approaching.

1.4 Out of Scope

The system will not be interactive, and it will not allow users to review, rate, and comment on the movies. Also, when users try to purchase tickets, they will enter their credit card information, but this information will not be used or saved in our application. In other words, our application does not do a real-world purchase system with the user's credit card information. Also, In the future, dockerizing the client-side and server-side code might help migrations and recovery. Also, renting another Compute Engine instance to host CI/CD pipeline, like Jenkins, might help with continuous integration and delivery of the application. Right now, any CI/CD pipelines or tokenization will be implemented to pursue simplicity.

Section 2 - System Architecture



Cloud Scheduler
Invokes the function every week
Cloud Functions
Makes HTTP call to external API to fetch real time movie data
THE MOVIE DB
Returns the real time movie data
Inserts new movies to database. Initializes the movie seances for each of them.
Compute Engine
Hosts the Node.js backend application
Data Retrieval, Update, Removal, and Insertion via SQL Queries
Cloud SQL
MySQL Engine Persists User, Movie, and Ticket Data
Gets the information of the users who have a ticket for that day
Cloud Scheduler
Invokes the function every midnight (GMT+3)
Cloud Functions
Returns HTTP Responses
Makes HTTP requests
App Engine
Hosts the React.js frontend application
Interaction with the application
Sends a reminder email that the user has a movie ticket that day.
Client

Figure: Revised System Architecture

The Cloud Architecture that we are going to use mainly has 1 Compute Engine Instance, 1 Cloud SQL Instance, 1 App Engine Instance, 2 Cloud Functions deployed, and 2 Cloud Schedulers. The compute engine will host the backend application, which is a Node.js application. The deployment of it will include the following steps:

1. Developing the backend application in our local machines. To manage the version control, we are using Git and GitHub.

2. Running the finalized backend application in our local machines. Once we make sure that every RESTful endpoint works, we are ready to deploy to the Google Cloud.
3. Creation of Compute Engine instance under our project in Google Cloud.
4. SSH into the instance and install required packages such as NPM, git, and forever (Keeps a Node.js application running) [1].
5. Clone the backend application's repository, CD, into the directory, and run the following command "forever start server.js" [1].
6. By using Postman, we will check if the endpoints are publicly available. If not, set the network settings of the Compute Engine instance from the cloud console.

The web application will listen to HTTP requests and, based on the requests, generate responses; the backend application will access the database, which is a Cloud SQL instance. The App Engine instance will host the React.js frontend application. The deployment of the frontend application will have the following steps [2]:

1. Developing the frontend application in our local machines. To manage the version control, we are using Git and GitHub.
2. Running the finalized frontend application in our local machines. Once we make sure that every page on the website works, we are ready to deploy to the Google Cloud.
3. Creation of the App Engine instance under our project in Google Cloud.
4. By using Cloud Shell, clone the frontend application's repository.
5. Add some config files and run the build.
6. By using the command "gcloud app deploy", deploy the website. And test if it is accessible from the public address.

Users use that website to interact with the application. The Cloud Function at the top will be invoked by a Cloud Scheduler every week. The function will fetch the currently showing movies in Turkey and update our movie database respectively. The other Cloud Function on the right-hand side of the diagram will be invoked by a Cloud Scheduler every day at midnight (Istanbul Timezone). It will send an email to the users who have a movie ticket on that day. The function can do it by accessing the data that is stored in the Cloud SQL instance.

2.1 Choice of Technologies, APIs, Algorithms

First of all, we chose to build a web application hosted by Google Cloud. The reasons why we chose web applications are that multiple users can access the same version of the application through various platforms like laptops, desktops, or mobile, and users don't have to install the app. Our system will have Three-Tier Architecture, which is described in section 5 in more detail because it provides scalability and security to our application. Through this architecture, our data tier and presentation tier will not communicate directly [3]. On the presentation layer, we will have a React.js front-end application. On the application layer, we will have a Node.js backend application. Finally, on the data layer, we will have Google's Cloud SQL instance running in the MySQL database engine. We chose a relational database for our application because it is important to store data about movies, showtime, theatre, seats, and movie seances in an organized and structured manner. Also, it provides our application with consistent and accurate data since relational databases let us implement proper checks and constraints to prevent errors and inconsistencies. For example, if a ticket is sold by a user, another user cannot buy this ticket again as we apply proper constraints and checks to our database. Google Cloud provides Cloud Spanner and Cloud SQL as a relational database. We chose Cloud SQL because it is easier to set up and use. Moreover, Cloud SQL is a better option for our application since our application is a relatively less complex and smaller application compared to other applications. The reasons why we went with MySQL engine are that we have experience with MySQL, it provides high performance, and its popularity makes finding resources for help easier [4].

The controllers are the backbone of the back-end system. They run in a Compute Engine instance as a Node.js application. The Compute Engine has the following properties:

Property Name	Property	Reasoning
Machine Type	e2-micro	We do not expect high traffic to the system since the application is a part of a course project. Therefore, to reduce the cost and get the work done, e2-micro has been chosen as the machine type.
Architecture	x86/64	It is the default industry standard.
Zone	europe-west3-b	To reduce the response time of the requests, the closest zone to Turkey has been chosen.
GPUs	0	We are not running any processes in that machine that requires GPU acceleration.

The data layer's "Movie" table will be fed by The Movie Database (TMDb) API, providing access to a wide range of information about movies containing titles, summaries, posters, trailers, release dates, ratings, and more. It has a request path that returns the movies that are currently playing around Turkey [5]. In our project, we will specify the region parameter as Turkey to narrow down the result of the request.

We have chosen Javascript as the main programming language around the system since it has a readable syntax, automatic memory management (garbage collector) [6], and a large variety of frameworks. Also, we, as a team, have a considerable amount of experience with Javascript. We have chosen React.js as the framework for the client-side application since it has modularity and reusability component-based architecture, allowing for easy development and maintenance of complex user interfaces. Also, we have chosen Node.js as the framework for the server-side application since it is fast and scalable [7]. Also, on the server-side application, the Express library will be used to ease the development experience even more.

From a security perspective, while signing up a new user, the user is added to the database by hashing the concatenation of a random password salt with it with the user's password. Also, we are using JSON Web Token (JWT) for each request other than the login and register to avoid unauthorized requests. Those practices will probably be sufficient in order for the application to be secure enough in its scope.

In terms of error detection, we will, for sure, utilize the language properties such as exception handling and logging. Also, we can create an alert policy on Google Cloud to see if the CPU, memory, and network bandwidth of the Compute Engine is overloaded [8]. To recover, we can create backups of the instances we have. Moreover, as we mentioned, MySQL has various features to eliminate errors related to data. We have used primary keys, foreign keys, unique constraints, and check constraints to ensure data consistency and prevent our application from the insertion of incorrect or duplicate data, which causes significant errors for our application.

The presentation layer, the client-side application, talks to the application layer, the server-side application, through the HTTP protocol, whose details are explained in section 7. It uses RESTFul APIs and sends/receives JSON objects as request and response models.

Section 3 - Data Dictionary

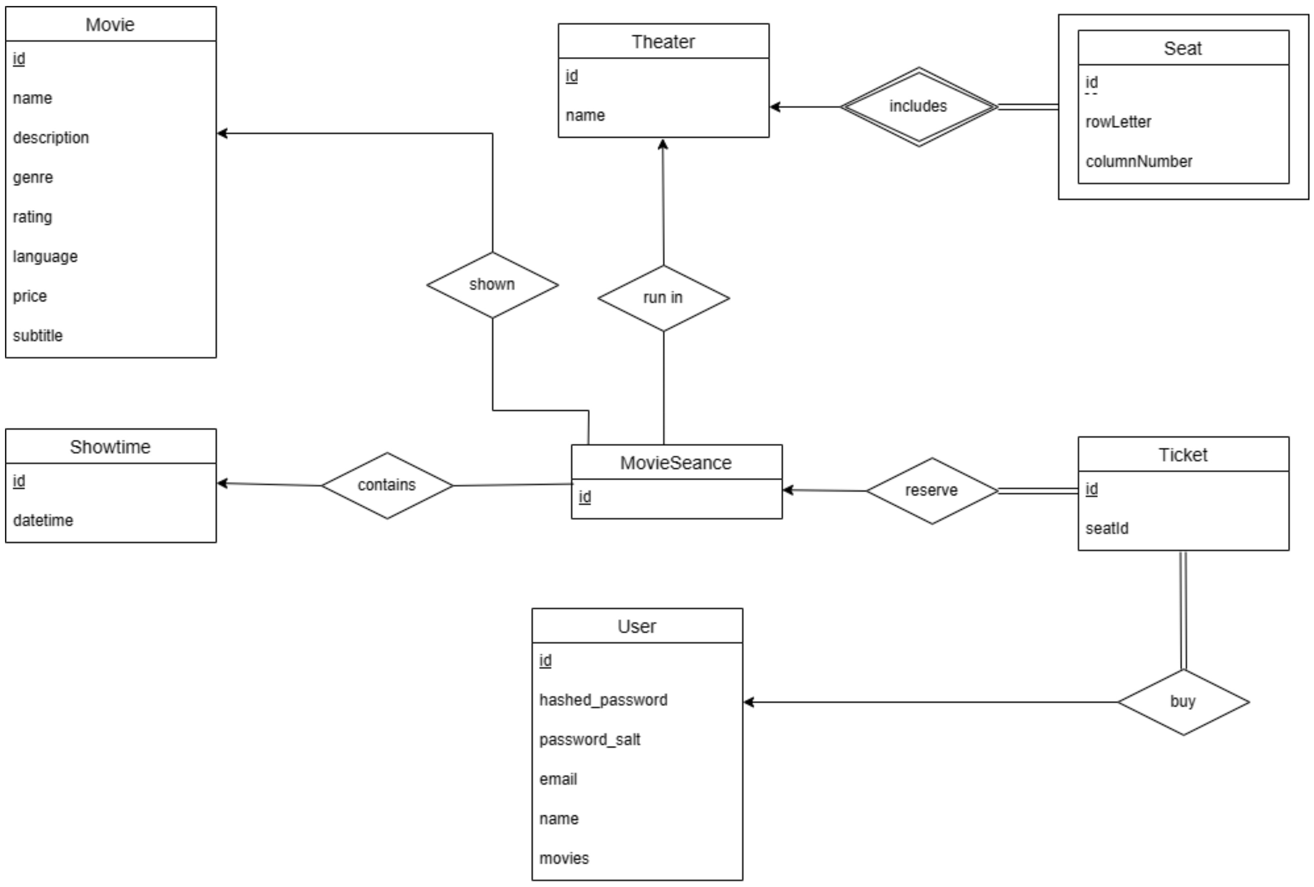


Figure: ER Diagram

The description of our database tables:

User

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
hashed_password	User's hashed password. Retrieved by hashing password + salt	VARCHAR(64)	NOT NULL
password_salt	User's randomly generated password salt	VARCHAR(32)	NOT NULL
email	User's email address	VARCHAR(255)	NOT NULL, UNIQUE
name	User's name.	VARCHAR(255)	NOT NULL
movies	Movie List which user watched	JSON	DEFAULT '[]'

Movie

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
name	Movie's name	VARCHAR(255)	NOT NULL
description	Movie's description	LONGTEXT	NOT NULL
rating	Movie's rating	FLOAT	NOT NULL
genre	Movie's genre	VARCHAR(255)	NOT NULL
language	Movie's original language	VARCHAR(255)	NOT NULL
price	Movie ticket price	FLOAT	NOT NULL
subtitle	Option indicating whether the subtitle is available	BIT	NOT NULL

Theater

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
name	Name of the theater	VARCHAR(255)	NOT NULL

Showtime

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
datetime	Date and time of the show	DATETIME	NOT NULL

MovieSeance

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
movieId	Id of the related movie.	INT	FOREIGN KEY (Movie.id)
showtimeId	Id of the related showtime.	INT	FOREIGN KEY (Showtime.id)
theatreId	Id of the related theatre.	INT	FOREIGN KEY (Theater.id)

Seat

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
theaterId	Id of the related theater	INT	FOREIGN KEY (Theater.id)
rowLetter	Row of the seat	CHAR	NOT NULL
columnNumber	Column of the seat	INT	NOT NULL

Ticket

Field	Notes	Type	Constraint
id	Unique Identifier	INT	PRIMARY KEY
userId	Id of the ticket owner	INT	FOREIGN KEY (User.id)
seatId	Id of the selected seat	INT	FOREIGN KEY (Seat.id)
movieSeanceId	Id of the movie seance	INT	FOREIGN KEY (MovieSeance.id)

reserve

Field	Notes	Type	Constraint
movieSeanceId	Id of the related movie seance.	INT	FOREIGN KEY (MovieSeance.id)
id	Unique Identifier	INT	FOREIGN KEY (Ticket.id)

In our movie ticket purchase application, we use a relational database system to make sure that all data about our users, movies, tickets, showtimes, theaters, and seats are saved in a dependable and long-lasting way. We have the following entities:

- User: This entity contains information about the user, such as their unique ids, two types of passwords, names, and emails.
- Movie: This entity has detailed information about movies like unique ids, names, descriptions, genres, ratings, languages, and ticket prices.
- Showtime: This entity contains unique ids and the times of the movie seances.
- Theater: This entity has a unique id and name. In our application, we have several theaters showing movies.
- Seat: This entity has id, and seat information like row and column numbers. The seat entity is a weak entity because every theater has its own seats, and it is not possible to have a seat without a theater.
- Movie Seance: This entity has a unique id.
- Ticket: This entity has a unique id and a seat id.

All of these entities have relations with each other:

- Movie Seance can only be associated with one showtime, one movie, and one theater, but every showtime, theater, and movie can have many movie seances. This relation will provide to have unique movie seances and their details like time, theater, and movie.
- Seat is a weak entity of Theater entity, and they have a one-to-many relationship. The seat is a dependent entity which means that it cannot exist without theater, and every theater has its own seats so that every theater can have its own seat plan.
- A ticket can be associated with only one user and one movie seance while a user can buy many tickets.
- A ticket should be only associated with one movie seance, whereas a movie seance can have many tickets.

Section 4 – Data Design

Our movie ticket purchase system includes storage and management of various types of data. These data can be divided into three major categories: persistent/static data, transient data, and external interface data.

4.1 Persistent/Static Data

4.1.1 Static Data

The only thing that we are willing to treat as configuration data is the public IPv4 address of the Compute Engine instance that runs our Node.js backend code. Other than that, we do not have any configuration data for the system. Also, we have movie genres, they are kept as static data, and users can specify their genre choices from a drop-down list, which is an HTML component with the tag “<ul>”.

4.1.2 Persisted data

Data persistence is the longevity of data after the application that created it has been closed. Hence, persistent data refers to data that is kept and preserved for a long time, even when the system or program is not in use. We created a database of related data that is organized in a structured manner. The ER diagram which we presented in Section 3 demonstrates the visual presentations of entities, as well as their corresponding relationships, and detailed information about the database can be seen there too.In our application, all of the information that is stored in our database is persisted. In other words, all of the user information, movie details, ticket, showtime, theaters, seats, and movie seances are persisted so that because this information is always saved and accessible in the database, a user can quickly access movie specifics and available showtimes when they want to buy a movie ticket. Additionally, after a user purchases a ticket and reserves a spot, storing this data makes it impossible for other users to book that same seat.

In our database, we have some entities that are not modified. For example, our application serves as a system to manage the booking system of a specific number of theaters. Hence, the “Theater” entity will stay the same after its creation. Also, the “Seat” entity will not be modified because the theater seats are fixed and do not change unless there are unforeseen events like physical damage or deformation. In other words, each theater's seating arrangement and plan are characteristics that do not change frequently.

On the other hand, in our application, movies are updated periodically, accessing the new movies from the external API, The Movie Database [5]. In this manner, movie data is an example of dynamic data. Moreover, users can update their personal information, which makes “User” dynamic too. “Showtimes” is also dynamic because they should be updated over time.

4.2 Transient Data

In our application, we will have forms that will be filled by end-users ie. movie, theater, and seance forms. Those forms will contain transient data before the user submits them. Also, we are planning to use JWT (JSON Web Token) to control user authentication and authorization [9]. The token will mimic the session info of the user. Therefore, the tokens can also be considered transient data in our system.

4.3 External Interface Data

We will only have communication between the client side and server side through the HTTP protocol. To accomplish that, request and response JSON data will be sent bi-directionally. Those JSON objects are solid examples of external interface data.

4.4 Transformation of Data

In our application, we will not do a transformation of data in order to be stored in a different form.

Section 5 - Detailed of Backend / Cloud Design

5.1 Software Application Domain Chart

Veni Vidi Movie uses a Three-Tier Architectural Style, which organizes applications into three logical and physical computing tiers [3]. It consists of presentation, application, and data tiers. By organizing the application into three layers, some software architecture design principles, such as modularity and scalability, are intended to be satisfied. It allows each tier to be developed, maintained, and updated independently, such that changes made in a tier do not affect others.

The presentation tier accounts for providing a user interface enabling users to interact with the application. It displays information and receives requests from users. In our application, the presentation tier runs on a web browser. As can be seen from the figure below, it includes a package of components, namely Login/Sign Up, Home, Profile, Tickets, and Movie/Seance/Seat Selection. These components will be explained in detail in the following section. All the components use the application tier to access the data tier to retrieve data, insert a new record into the database, or delete or update an existing record.

The application tier connects the other two layers. Its responsibility is to make the necessary calculations, process the information collected in the presentation layer, and send data retrieval, insertion, update, and deletion requests to the data layer. Our application includes a package of controllers, namely Authentication, Movie, Theater, User, Ticket, and Payment Controller. These controllers will be explained in detail in the following section.

The data tier is responsible for the management and maintenance of the database. In our application, the data layer is a relational database management system. It includes a package of data tables managed by the Google Cloud Storage Manager, namely User, Ticket, Movie Seance, Showtime, Movie, Theater, and Seat Data. They correspond to the entities of the ER diagram presented in Section 3. They will be explained in detail in the following section.

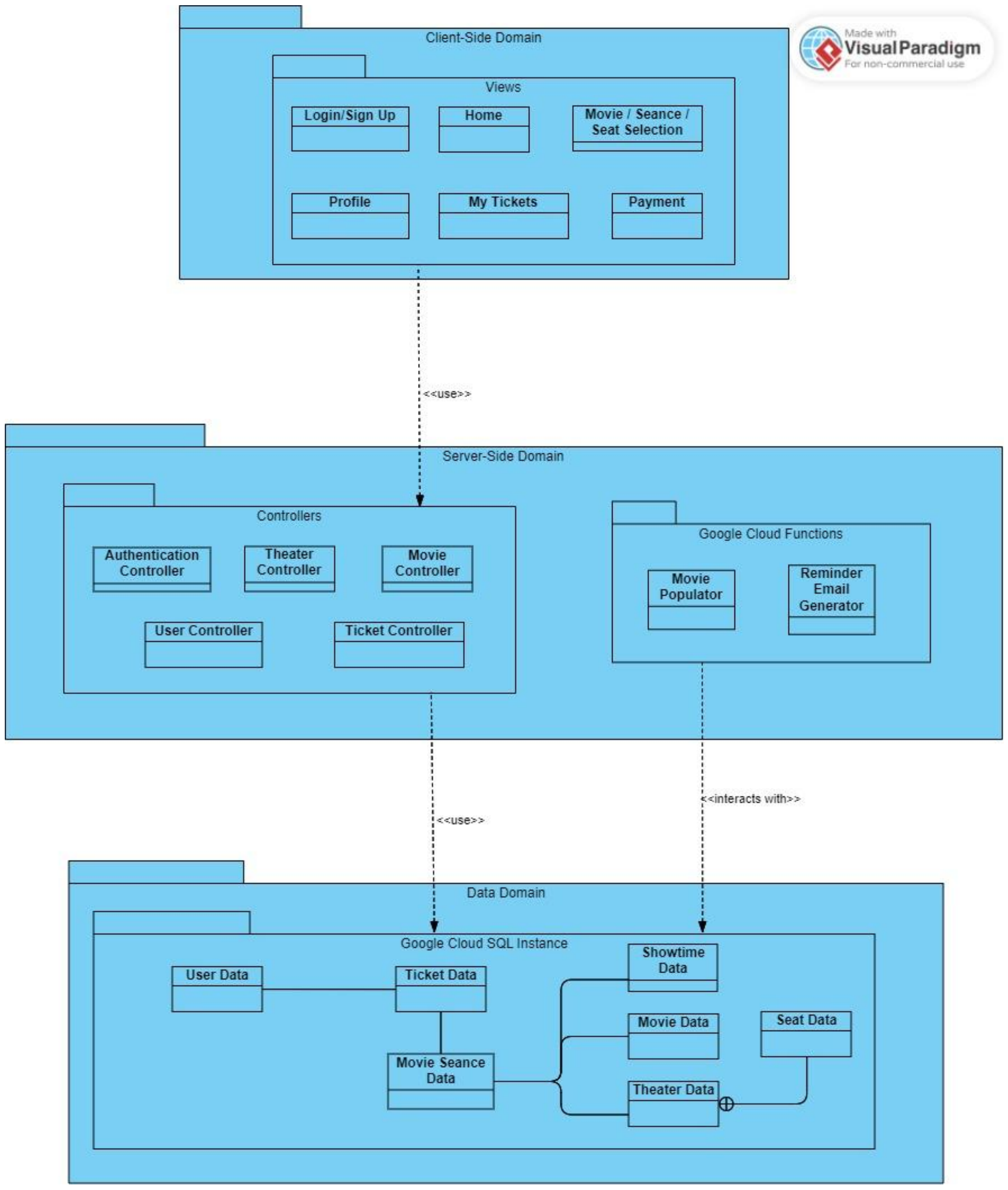


Figure: Application Domain Diagram

5.2 Components

5.2.1 REST Controllers

Our backend application consists of a bunch of REST controllers, each of them being responsible for managing user flows such as authentication, authorization, movie management, ticket purchase, etc. REST Controller will run in a Google Cloud Compute Engine instance.

5.2.1.1 Authentication Controller

Manages user authentication of the application.

5.2.1.1.1 /register

Handles a user’s sign-up process by checking if another account exists with the same email. If not, it adds the user to the database by hashing the concatenation of a random password salt with it with the user's password.

URL	{baseUrl}/authentication/register
Method	POST
Request Parameters	N/A

Request Body	`{`
	`"name": "John Smith",`
	`"email": "johnsmith@gmail.com",`
	`"password": "myPassword123"`
	`}`

Sample Response Body	`{`
	`"key": "SUCCESS",`
	`"userId": 1122,`
	`"email": "johnsmith@gmail.com",`
	`"token": "Bearer`
	`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Ii`
	`s5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le4`
	`8d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
	`}`

5.2.1.1.2 /login

Handles user’s login process.

URL	{baseUrl}/authentication/login
Method	POST
Request Parameters	N/A

Request Body	`{`
	`"email": "johnsmith@gmail.com",`
	`"password": "myPassword123"`
	`}`

Sample Response Body	`{`
	`"key": "SUCCESS",`
	`"userId": 1122,`
	`"email": "johnsmith@gmail.com",`
	`"token": "Bearer`
	`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Ii`
	`s5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le4`
	`8d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
	`}`

5.2.1.1.3 /changePassword

Handles password change of the user.

URL	{baseUrl}/authentication/changePassword
Method	PATCH
Request Parameters	N/A

Request Headers	`"token": "Bearer`
	`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Ii`
	`s5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le4`
	`8d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`

Request Body	`{`
	`"userId": 1122,`
	`"currentPassword": "myPassword123",`
	`"newPassword":`

	"myNewPasswordButHighlySecure??123." }

Sample Response Body	{
	  "key": "SUCCESS"
	}

5.2.1.1.4 /forgottenPassword

Handles the forgotten password process of the user whose email is given. It sends a link to the user's email to reset their password.

URL	{baseUrl}/authentication/forgottenPassword
Method	POST
Request Parameters	N/A
Request Headers	"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"

Request Body	{
	  "userEmail": "johnsmith@gmail.com",
	}

Sample Response Body	{
	  "key": "SUCCESS"
	}

5.2.1.2 User Controller

Manages user-related information transactions.

5.2.1.2.1 /profile

Returns the profile information of the user whose id is given.

URL	{baseUrl}/user/profile
Method	GET
Request Parameters	?userId=1122

Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`{ "key": "SUCCESS", "name": "John Smith", "email": "johnsmith@gmail.com", "pastMovies": [{ "movieName": "John Wick", "movieId": 44, "dateTime": "2023-04-01T22:15:00+03:00", "theatre": "Mithat Çoruh Amfi", "theatreId": 2, "seat": { "row": "D", "column": 15 }, "price": 15 }, { "movieName": "Recep İvedik 8", "movieId": 41, "dateTime": "2023-02-28T10:45:00+03:00", "theatre": "En güzel salon", "theatreId": 1, "seat": { "row": "J", "column": 9 }, "price": 20 }] }`

5.2.1.2.2 /editProfile

Edits the basic information of the user if the new email is not taken by another user in the system.

URL	{baseUrl}/user/editProfile
Method	PATCH
Request Parameters	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`

Request Body	`{`
	`"userId": 1122,`
	`"newEmail": "dennis.ritchie@ug.bilkent.edu.tr",`
	`"newName": "Dennis Ritchie"`
	`}`

Sample Response Body	`{`
	`"key": "SUCCESS",`
	`}`

5.2.1.3 Movie Controller

Manages the movie related user interactions.

5.2.1.3.1 /showing

Returns the currently showing movies in all of the theaters.

URL	{baseUrl}/movie/showing
Method	GET
Request Parameters	-
Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`

Sample Response Body	`[`
	`{`
	`"movieName": "John Wick",`
	`"movieId": 44,`

```
       "genres": ["Action", "Crime"],
       "director": "Chad Stahelski",
       "language": "en-us",
       "subtitle": true,
       "description": "John Wick uncovers a path to
defeating The High Table. But before he can earn
his freedom, Wick must face off against a new enemy
with powerful alliances across the globe and forces
that turn old friends into foes.",
       "showTimes": [
           {
               "id": 102,
               "dateTime":
"2023-04-11T10:45:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 103,
               "dateTime":
"2023-04-11T13:30:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 104,
               "dateTime":
"2023-04-12T10:40:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 404,
               "dateTime":
"2023-04-11T10:45:00+03:00",
               "theatre": "En güzel salon",
               "theatreId": 1
           },
           {
               "id": 405,
               "dateTime":
```

```
"2023-04-11T13:30:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 406,
               "dateTime":
"2023-04-12T10:40:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           }
       ],
       "price": 15
   },
    {
       "movieName": "Recep İvedik 8",
       "movieId": 41,
       "genres": ["Comedy"],
       "director": "Şahan Gökbakar",
       "language": "tr-tr",
       "subtitle": false,
       "description": "Recep, doğayı tehdit eden
büyük bir tehlikeye karşı heyecanlı bir maceraya
atılır. Yepyeni bir maceraya hazır olun.",
       "showTimes": [
           {
               "id": 802,
               "dateTime":
"2023-04-21T10:45:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 803,
               "dateTime":
"2023-04-21T13:30:00+03:00",
               "theatre": "Mithat Çoruh Amfi",
               "theatreId": 2
           },
           {
               "id": 804,
                "dateTime":
"2023-04-22T10:40:00+03:00",
                "theatre": "Mithat Çoruh Amfi",
                "theatreId": 2
            },
            {
                "id": 1904,
                "dateTime":
"2023-04-21T10:45:00+03:00",
                "theatre": "En güzel salon",
                "theatreId": 1
            },
            {
                "id": 1905,
                "dateTime":
"2023-04-21T13:30:00+03:00",
                "theatre": "Mithat Çoruh Amfi",
                "theatreId": 2
            },
            {
                "id": 1906,
                "dateTime":
"2023-04-22T10:40:00+03:00",
                "theatre": "Mithat Çoruh Amfi",
                "theatreId": 2
            }
        ],
        "price": 15
    }
]
```

5.2.1.3.2 /information

Returns the information about a movie whose id is given and its occupied seats in each theater.

URL	{baseUrl}/movie/information
Method	GET
Request Parameters	?movieId=44

Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`

Sample Response Body

```
{
   "movieName": "John Wick",
   "movieId": 44,
   "genres": [
       "Action",
       "Crime"
   ],
   "director": "Chad Stahelski",
   "language": "en-us",
   "subtitle": true,
   "description": "John Wick uncovers a path to
defeating The High Table. But before he can earn
his freedom, Wick must face off against a new enemy
with powerful alliances across the globe and forces
that turn old friends into foes.",
   "showTimes": [
       {
           "id": 102,
           "dateTime": "2023-04-11T10:45:00+03:00",
           "theatre": "Mithat Çoruh Amfi",
           "theatreId": 2,
           "occupiedSeats": [
               {
                   "row": "D",
                   "column": 15
               },
               {
                   "row": "D",
                   "column": 14
               },
               {
                   "row": "J",
                   "column": 11
               }
           ]
       },
       {
           "id": 103,
           "dateTime": "2023-04-11T13:30:00+03:00",
           "theatre": "Mithat Çoruh Amfi",
           "theatreId": 2,
           "occupiedSeats": [
               {
                   "row": "D",
                   "column": 15
               },
               {
                   "row": "D",
                   "column": 14
               },
               {
                   "row": "J",
                   "column": 11
               },
               {
                   "row": "F",
                   "column": 8
               },
               {
                   "row": "F",
                   "column": 9
               },
               {
                   "row": "F",
                   "column": 10
               }
           ]
       },
       {
           "id": 104,
           "dateTime": "2023-04-12T10:40:00+03:00",
           "theatre": "Mithat Çoruh Amfi",
           "theatreId": 2,
           "occupiedSeats": []
       },
       {
           "id": 404,
           "dateTime": "2023-04-11T10:45:00+03:00",
           "theatre": "En güzel salon",
           "theatreId": 1,
           "occupiedSeats": [
               {
                   "row": "A",
                   "column": 10
               }
           ]
       },
       {
           "id": 405,
           "dateTime": "2023-04-11T13:30:00+03:00",
           "theatre": "Mithat Çoruh Amfi",
           "theatreId": 2,
           "occupiedSeats": [
               {
                   "row": "K",
                   "column": 21
               },
               {
                   "row": "K",
                   "column": 22
               }
           ]
       },
       {
           "id": 406,
           "dateTime": "2023-04-12T10:40:00+03:00",
           "theatre": "Mithat Çoruh Amfi",
           "theatreId": 2,
           "occupiedSeats": []
       }
   ],
   "price": 15
}
```

5.2.1.4 Theater Controller

The theater and seating data is static. Therefore, no POST, PATCH, or DELETE operation will be implemented to maintain the simplicity of the system.

5.2.1.4.1 /all

Returns to all theaters in the application with their names and IDs.

URL	{baseUrl}/theater/all
Method	GET
Request Parameters	-
Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`[ { "theaterId": 1, "name": "En güzel salon" }, { "theaterId": 2, "name": "Mithat Çoruh Amfi" } ]`

5.2.1.4.1 /seating

Returns the seating plan and name of a theater whose ID is given.

URL	{baseUrl}/theater/seating
Method	GET
Request Parameters	?theaterId=1
Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`{ "theaterId": 2, "name": "Mithat Çoruh Amfi", "defaultSeating": [`

```
        {
            "column": 1,
            "row": "A"
        },
        {
            "column": 1,
            "row": "B"
        },
        {
            "column": 1,
            "row": "C"
        },
        {
            "column": 1,
            "row": "D"
        },
        {
            "column": 1,
            "row": "E"
        },
        {
            "column": 1,
            "row": "F"
        },
        {
            "column": 1,
            "row": "G"
        },
        {
            "column": 1,
            "row": "H"
        },
        {
            "column": 1,
            "row": "I"
        },
        {
            "column": 1,
            "row": "J"
        },
        {
            "column": 1,
            "row": "K"
        },
        {
            "column": 1,
            "row": "L"
        },
        {
            "column": 1,
            "row": "M"
        },
        {
            "column": 1,
            "row": "N"
        },
        {
            "column": 1,
            "row": "O"
        },
        {
            "column": 1,
            "row": "P"
        },
        {
            "column": 1,
            "row": "Q"
        },
        {
            "column": 1,
            "row": "R"
        },
        {
            "column": 1,
            "row": "S"
        },
        {
            "column": 1,
            "row": "T"
        },
        {
            "column": 1,
```

	```
	"row": "U"
	},
	.
	.
	.
	.
	.
	{
	"column": 26,
	"row": "A"
	},
	{
	"column": 26,
	"row": "B"
	},
	{
	"column": 26,
	"row": "C"
	},
	{
	"column": 26,
	"row": "D"
	},
	{
	"column": 26,
	"row": "E"
	},
	{
	"column": 26,
	"row": "F"
	},
	{
	"column": 26,
	"row": "G"
	},
	{
	"column": 26,
	"row": "H"
	},
	{
	"column": 26,
	```

```json
            "row": "I"
        },
        {
            "column": 26,
            "row": "J"
        },
        {
            "column": 26,
            "row": "K"
        },
        {
            "column": 26,
            "row": "L"
        },
        {
            "column": 26,
            "row": "M"
        },
        {
            "column": 26,
            "row": "N"
        },
        {
            "column": 26,
            "row": "O"
        },
        {
            "column": 26,
            "row": "P"
        },
        {
            "column": 26,
            "row": "Q"
        },
        {
            "column": 26,
            "row": "R"
        },
        {
            "column": 26,
            "row": "S"
```

	```
	        },
	        {
	            "column": 26,
	            "row": "T"
	        },
	        {
	            "column": 26,
	            "row": "U"
	        }
	    ]
	}
	```

5.2.1.5 Ticket Controller

Controls the ticket operations of the users.

5.2.1.5.1 /purchase

Manages the ticket purchase operation of a user. If every validation, seat availability, payment (mock), and date and time of the movie, pass, the ticket purchase operation is considered as successful. After the purchase operation, an email is sent to the user saying that "Dear <NAME_OF_THE _USER> ticket purchase has been successful for the movie <MOVIE_NAME>, at the date <DATE_TIME>, at the theater <THEATER_NAME>, on the seat <SEAT_ROW>-<SEAT_COLUMN> for the ticket price <TICKET_PRICE>."

URL	{baseUrl}/ticket/purchase
Method	POST
Request Parameters	N/A

Request Body	
	```
	{
	    "movieId": 44,
	    "showtimeId": 103,
	    "theaterId": 2,
	    "seat": {
	        "row": "J",
	        "column": 9
	    },
	    "transaction_price": 15,
	    "credit_card": {
	        "owner_name": "John Smith",
	```

	`"number": "1234 4567 8901 2345", "cvv_code": "012" } }`
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`{ "key": "SUCCESS", "ticketId": 1992, }`

5.2.1.5.2 /information

Returns the information of the tickets of the user whose Id is given.

URL	{baseUrl}/ticket/profile
Method	GET
Request Parameters	?userId=1122
Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`[ { "movieName": "John Wick", "movieId": 44, "dateTime": "2023-04-21T22:15:00+03:00", "theatre": "Mithat Çoruh Amfi", "theatreId": 2, "seat": { "row": "D", "column": 15 },`

```
        "price": 15
    },
    {
        "movieName": "Recep İvedik 8",
        "movieId": 41,
        "dateTime": "2023-02-22T10:45:00+03:00",
        "theatre": "En güzel salon",
        "theatreId": 1,
        "seat": {
            "row": "J",
            "column": 9
        },
        "price": 20
    }
]
```

5.2.1.5.3 /cancel

Cancels the ticket whose id is given. As a mock design, the system will assume that the purchase price of the ticket will be refunded automatically. After the cancellation process, an email will be sent to the user saying that "Dear <NAME_OF_THE _USER> ticket cancellation has been successful for the movie <MOVIE_NAME>, at the date <DATE_TIME>, at the theater <THEATER_NAME>, on the seat <SEAT_ROW>-<SEAT_COLUMN>. The ticket price <TICKET_PRICE> has been refunded to your credit card. You can see the balance in a few work days based on your bank's process."

URL	{baseUrl}/ticket/cancel
Method	DELETE
Request Parameters	?ticketId=1192&userId=1122
Request Body	N/A
Request Headers	`"token": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwaG9uZSI6Iis5MDUzMjY3NDY2MDgiLCJpYXQiOjE2Nzk4MzcxMzR9.oy4L5le48d38V9tJYdr9fUrYNUuRGdXSA9Auva24-HI"`
Sample Response Body	`{ "key": "SUCCESS" }`

5.2.2 Google Cloud Functions

5.2.2.1 Movie Populator

Movie populator is a weekly scheduled Cloud Function whose code is written in Node.js to ease the maintenance of the codebase. It pursues the following procedure to accomplish its job:

1. Makes a HTTP request to the external API of The Movie Database to fetch the showing movies in Turkey.
   The API call has the following details as a sample:

URL	https://api.themoviedb.org/3/movie/now_playing
Method	GET
Request Parameters	?api_key=7f1d6c7331128cc279998bdc6c0cb873
Request Body	N/A
Request Headers	-
Sample Response Body	{ "adult": false, "backdrop_path": "/xDMIl84Qo5Tsu62c9DGWhmPI67A.jpg", "genre_ids": [28, 12, 878], "id": 505642, "original_language": "en", "original_title": "Black Panther: Wakanda Forever", "overview": "Queen Ramonda, Shuri, M’Baku, Okoye and the Dora Milaje fight to protect their nation from intervening world powers in the wake of King T’Challa’s death. As the Wakandans strive to embrace their next chapter, the heroes must band together with the help of War Dog Nakia and Everett Ross and forge a new path for the kingdom of Wakanda.", "popularity": 3952.862, "poster_path": "/sv1xJUazXeYqALzczSZ3O6nkH75.jpg", "release_date": "2022-11-09",

	"title": "Black Panther: Wakanda Forever", "video": false, "vote_average": 7.4, "vote_count": 3583 }

2. Iterates through each movie and checks if our system already has that movie in the system by connecting to the Cloud SQL instance.
3. If not, it inserts the movie into the Movie table.
4. It creates new showtimes in different theaters of the system again by inserting them into the Cloud SQL Instance.

5.2.2.2 Reminder Email Generator

Reminder Email Generator is a daily scheduled Cloud Function whose code is also written in Node.js to ease the maintenance of the codebase. It pursues the following procedure to accomplish its job:

1. At midnight, it is invoked.
2. Stores the current day in a variable during the runtime.
3. Connects to the Cloud SQL instance.
4. Executes a SQL query to get the user information and tickets that have movie shows in that day stored in the variable.
5. Sends an email to the user's emails who have a ticket that day to remind them.
   Email body sample:
   "Dear <NAME_OF_THE _USER>, today, you have a ticket purchased for the movie <MOVIE_NAME>, at the date <DATE_TIME>, at the theater <THEATER_NAME>, on the seat <SEAT_ROW>-<SEAT_COLUMN> for the ticket price <TICKET_PRICE>. "

5.2.3 Additional Data Components

We defined our tables, their elements and the relationship between these entities in Section 3. In Section 3, we also defined the primary, unique and foreign keys explicitly because they have great importance to provide security and maintainability of our application. In addition to these primary features, we also used triggers so that when one of the entities is updated, these triggers can change the related the entities automatically.

The trigger below provides that when a user purchase a ticket, the movie of the ticket will be added to the movie list of the user automatically.

```
DELIMITER $$
CREATE TRIGGER add_movie_to_user_movies

AFTER INSERT ON tickets
FOR EACH ROW
BEGIN
    DECLARE movie_id INT;

    SELECT ms.movieId
    INTO movie_id
    FROM TICKET as t, MovieSeance as ms
    WHERE t.id = NEW.id and ms.id = NEW.movieSeanceId


    UPDATE user
    SET movies = JSON_ARRAY_APPEND(movies, '$', movie_id)
    WHERE id = NEW.userId;
END$$
DELIMITER ;
```

Also, when users try to cancel their tickets from the application, the following trigger will check the date of the ticket and if the ticket's time passed, it won't permit users to cancel their tickets.

```
DELIMITER $$
CREATE TRIGGER check_ticket_date_on_delete
BEFORE DELETE ON Ticket
FOR EACH ROW
BEGIN
    DECLARE ticket_date DATETIME;
    SET ticket_date = OLD.datetime;

    IF (ticket_date < NOW()) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete a ticket that has already
occurred.';
    END IF;
END$$
DELIMITER ;
```

Section 6 - User Interface Design

6.1 User Interface Design Overview

6.1.1 Login Page

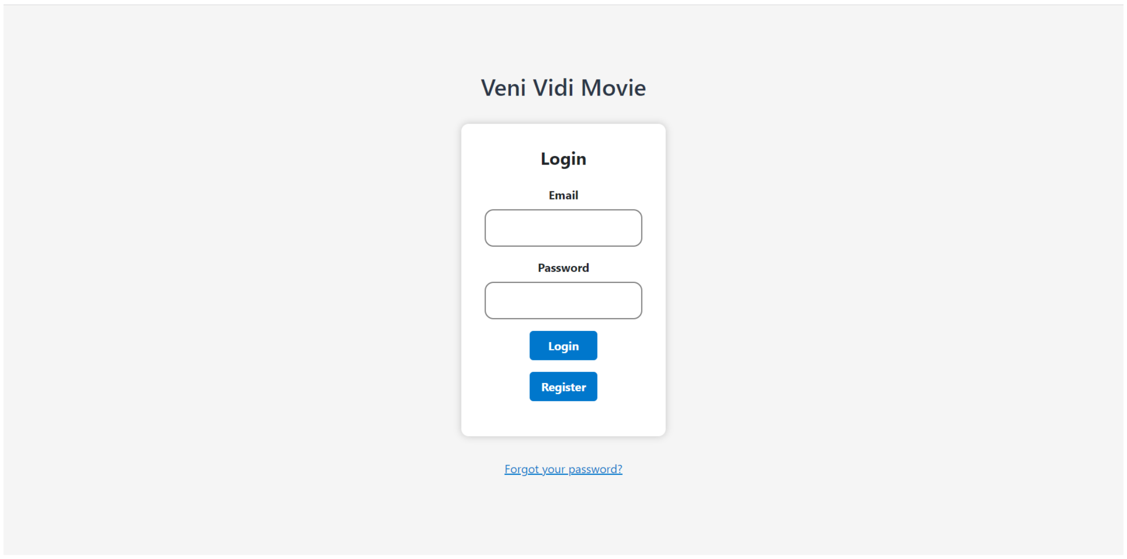


Figure: Login Page

The user can login to the application by entering his/her email address and password on this page. If not registered, the user is redirected to the registration page when the register button is clicked.

6.1.2 Registration Page

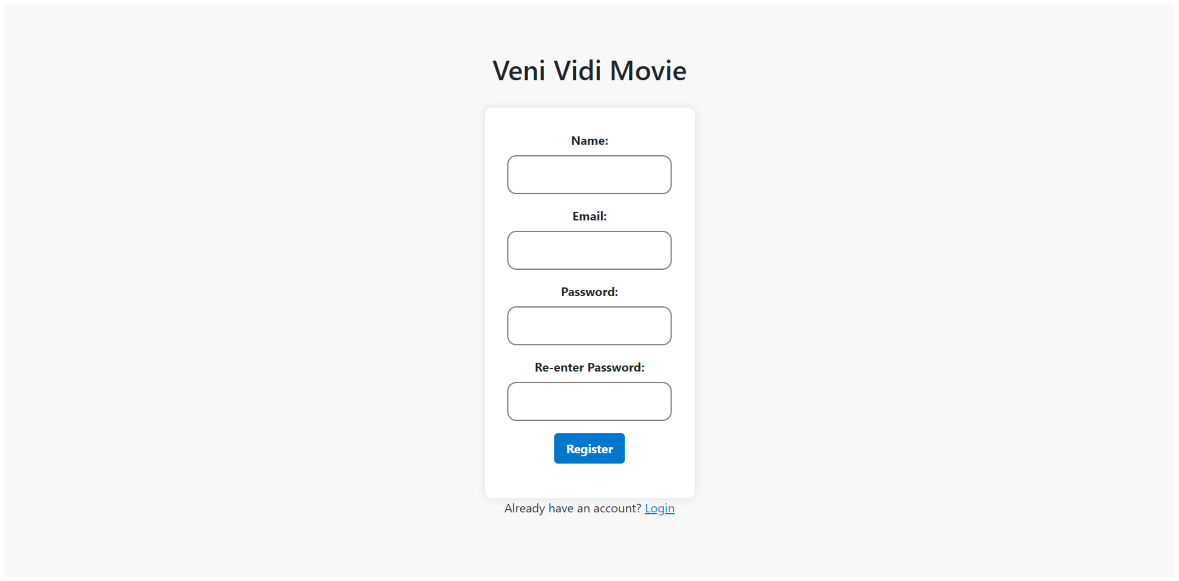


Figure: Registration Page

The user can sign up for the application by entering his/her name, email address, and password on this page. If already registered, the user is redirected to the login page when the login button is clicked.

6.1.3 Home Page

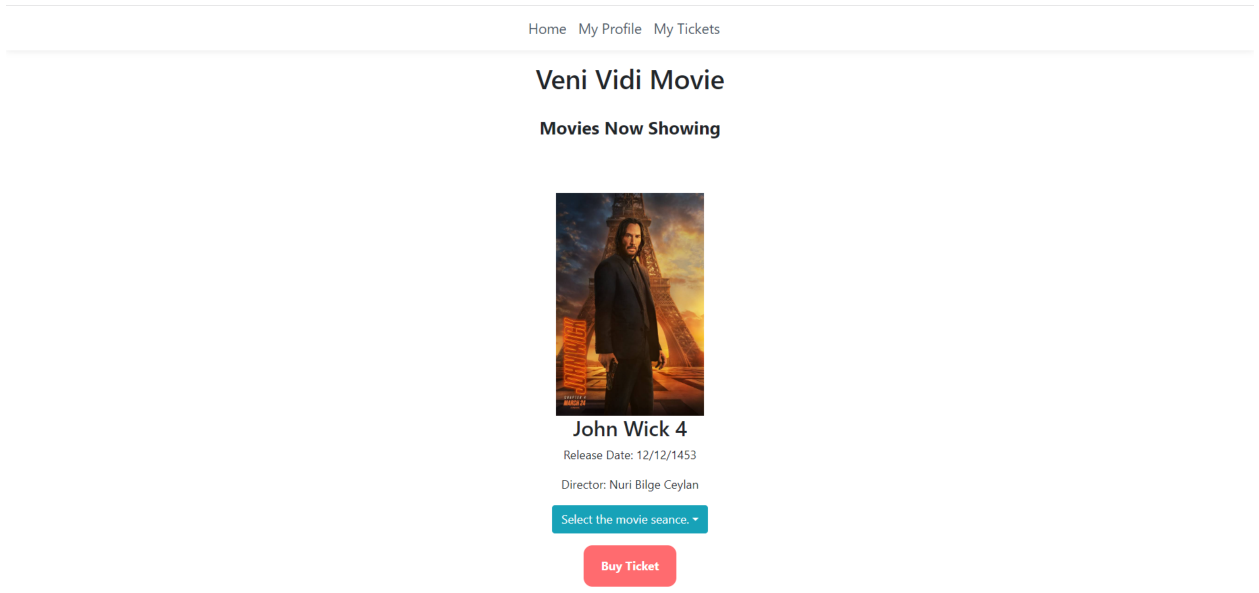
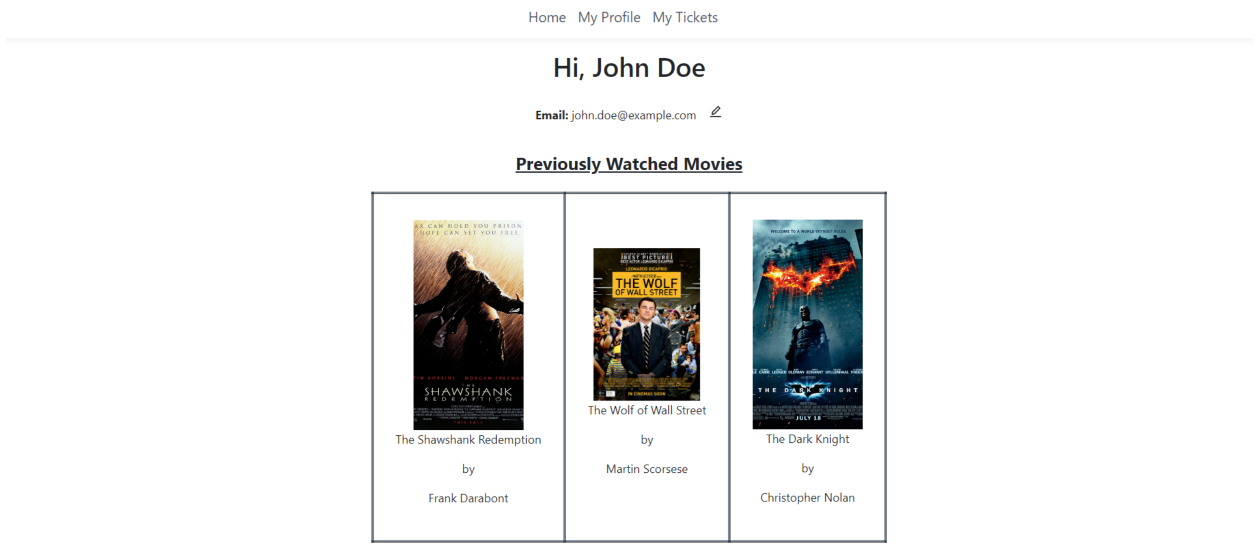


Figure: Home Page

After registering or logging into the application, the user is redirected to the home page. On this page, all movies currently showing in theaters are displayed. The movie name, release date, and director are shown for each movie. The user can select the movie seance for a particular movie. When the user clicks on the “Buy Ticket” button, he/she is redirected to the page where seat selection takes place in order to complete the movie ticket purchase. The user can also be redirected to the profile and tickets page using the navigation bar.

6.1.4 Profile Page



Hi, John Doe

Email: john.doe@example.com

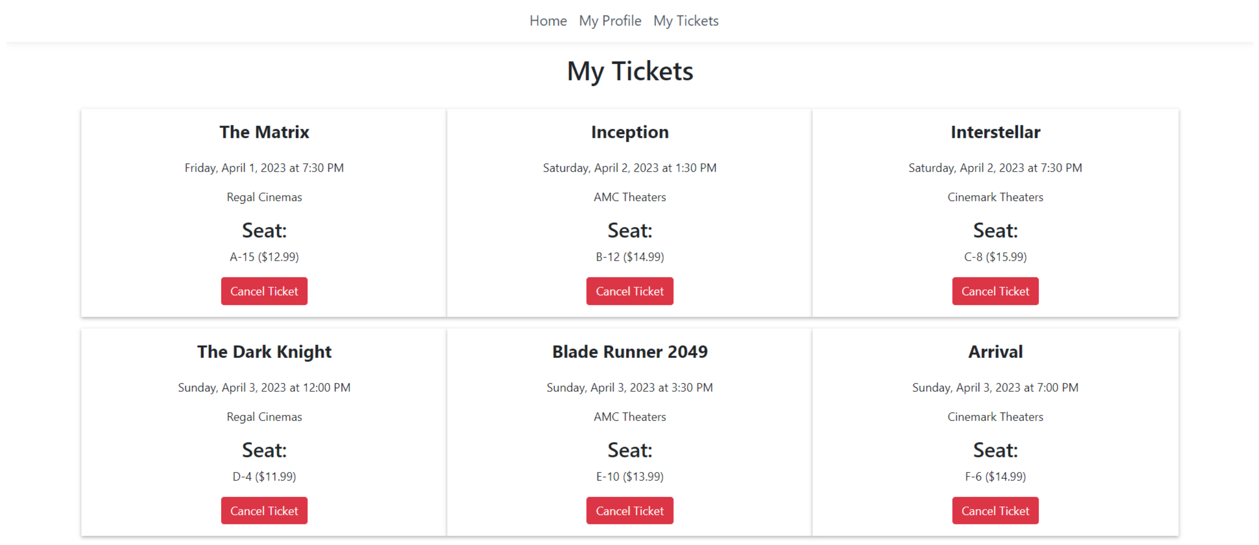
Previously Watched Movies

The Shawshank Redemption by Frank Darabont	The Wolf of Wall Street by Martin Scorsese	The Dark Knight by Christopher Nolan

Figure: Profile Page

The user's name and email address are shown on this page. The user is allowed to edit his/her email address by clicking the edit icon next to the email address information. The movies that the user has previously purchased a ticket for and watched are displayed. The user can also be redirected to the home and tickets page using the navigation bar.

6.1.5 Tickets Page



Home My Profile My Tickets

My Tickets

The Matrix Friday, April 1, 2023 at 7:30 PM Regal Cinemas Seat: A-15 ($12.99) Cancel Ticket	**Inception** Saturday, April 2, 2023 at 1:30 PM AMC Theaters Seat: B-12 ($14.99) Cancel Ticket	**Interstellar** Saturday, April 2, 2023 at 7:30 PM Cinemark Theaters Seat: C-8 ($15.99) Cancel Ticket
The Dark Knight Sunday, April 3, 2023 at 12:00 PM Regal Cinemas Seat: D-4 ($11.99) Cancel Ticket	**Blade Runner 2049** Sunday, April 3, 2023 at 3:30 PM AMC Theaters Seat: E-10 ($13.99) Cancel Ticket	**Arrival** Sunday, April 3, 2023 at 7:00 PM Cinemark Theaters Seat: F-6 ($14.99) Cancel Ticket

Figure: Tickets Page

The user can view his/her tickets on this page. The ticket contains the movie name, theater name, date and session of the movie, and the selected seat. The user is allowed to cancel a ticket before the movie date and time arrive. The user can also be redirected to the home and profile page using the navigation bar.

6.1.6 Seat Selection Page

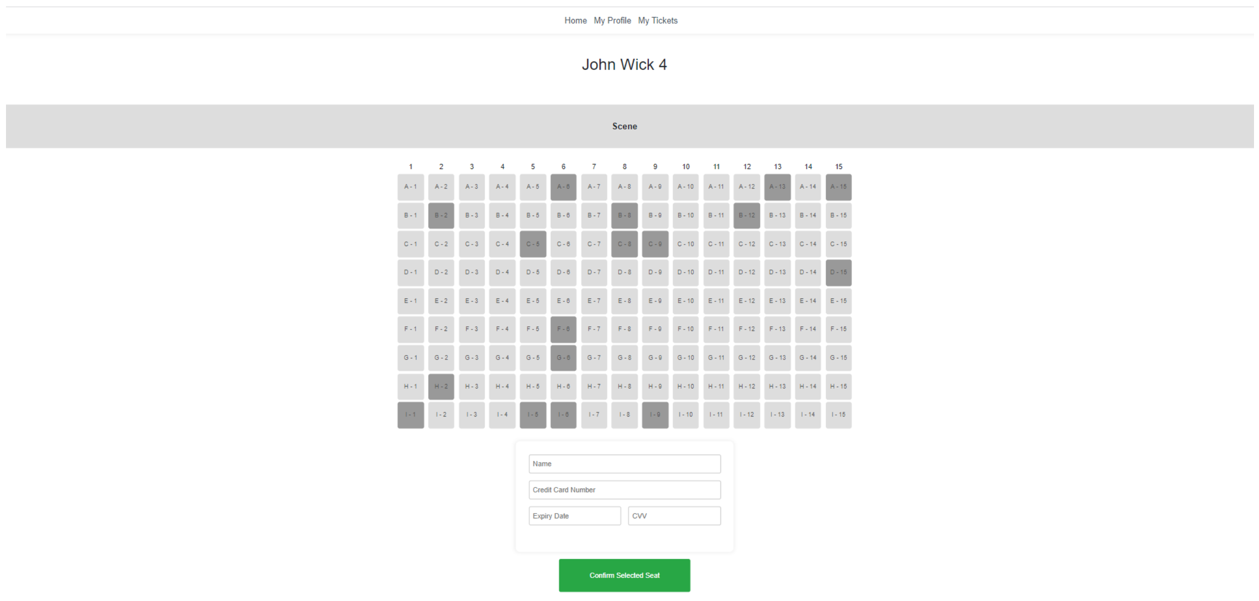


Figure: Seat Selection Page

The user selects a seat to watch the movie on this page. The user is prompted to enter the credit card information. When the “Confirm Selected Seat” button is clicked after selecting a seat and entering credit card information, the process of movie ticket purchase is successfully completed. The user can also be redirected to the home, profile and tickets page using the navigation bar.

6.2 User Interface Navigation Flow

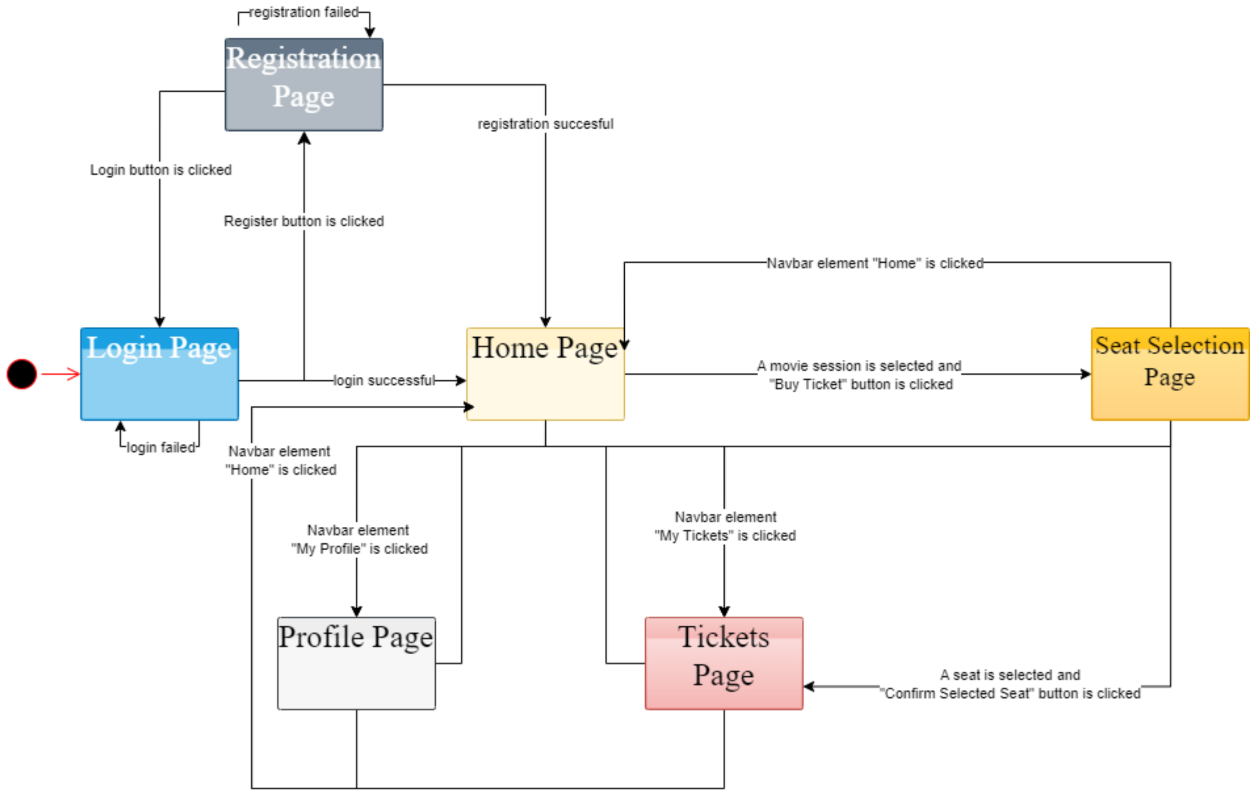

registration failed
Registration Page
registration succesful
Login button is clicked
Register button is clicked
Navbar element "Home" is clicked
Login Page
login successful
Home Page
A movie session is selected and "Buy Ticket" button is clicked
Seat Selection Page
login failed
Navbar element "Home" is clicked
Navbar element "My Profile" is clicked
Navbar element "My Tickets" is clicked
Profile Page
Tickets Page
A seat is selected and "Confirm Selected Seat" button is clicked

Figure: User Interface Navigation Flow

6.3 Use Cases / User Function Description

When we present our User Interface Design in section 6.1., we also explained their use cases, and functionalities for users.

Section 7 - Other Interfaces (Optional, only if you have something relevant)

7.1 Database Interface

- Technology: MySQL database
- Interaction: CRUD operations (create, read, update, delete) to store and retrieve data
- Protocol: SQL protocols (MySQL Driver)
- Message format: SQL queries
- Failure conditions: network errors, database errors, concurrency issues

- Handshaking: database connection and authentication

7.2 API Interface

- Technology: RESTful API provided by Express framework.
- Interaction: HTTP requests (GET, POST, PUT, DELETE) to access server-side resources
- Protocol: HTTP protocol
- Message format: JSON data
- Failure conditions: network errors, server errors, client errors (e.g., incorrect input data)
- Handshaking: API endpoint URLs and authentication

Section 8 - Alternatives Considered

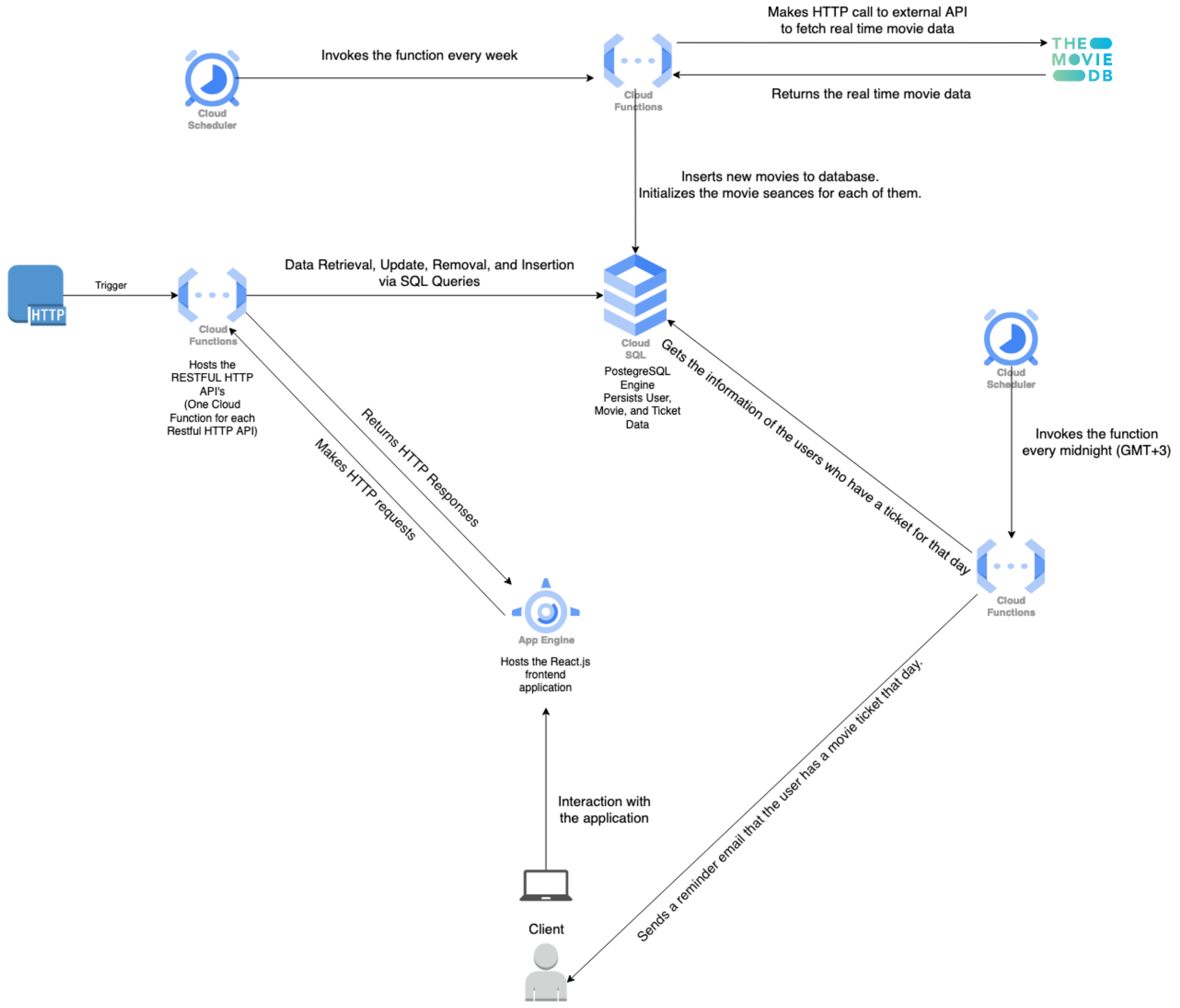

Cloud Scheduler
Invokes the function every week
Cloud Functions
Makes HTTP call to external API to fetch real time movie data
THE MOVIE DB
Returns the real time movie data
Inserts new movies to database. Initializes the movie seances for each of them.
HTTP
Trigger
Cloud Functions
Hosts the RESTFUL HTTP API's (One Cloud Function for each Restful HTTP API)
Data Retrieval, Update, Removal, and Insertion via SQL Queries
Cloud SQL
PostegreSQL Engine Persists User, Movie, and Ticket Data
Gets the information of the users who have a ticket for that day
Cloud Scheduler
Invokes the function every midnight (GMT+3)
Cloud Functions
Returns HTTP Responses
Makes HTTP requests
App Engine
Hosts the React.js frontend application
Interaction with the application
Client
Sends a reminder email that the user has a movie ticket that day.

Figure: Alternative System Architecture

As an alternative design, we could have the system diagram above. There are 2 main differences from the one that is shown in section 2:

1. For the backend application, instead of a Compute Engine instance, we could deploy multiple Cloud Functions that can be triggered by HTTP requests [10]. In the end, we could have the same RESTful API endpoints developed. However, there are some advantages and disadvantages:

   Advantages:
   - We could develop each endpoint in a programming language that we would like, such as Java, Python, Node.js, Go, Ruby, C#, and PHP [11].
   - They can scale up automatically based on the traffic. On a Compute Engine instance, we have to stick with the CPU, memory, or storage options unless we do not enable Autoscaling.

   Disadvantages:
   - When a Cloud Function has not been invoked for a certain amount of time (generally in 15 minutes), the first invocation will take much more time. This concept is known as Cold Start in Serverless architecture [12].
   - We do not have any solid experience with Serverless Framework. We want to deliver this project in a working state.
2. For the database storage option, we could select PostgreSQL as our database engine, which is also a popular engine. Instead, we went with MySQL. This was simply because we are all experienced in MySQL.

Other than that, we could select a NoSQL database like Firebase Firestore. However, our project is a good example of relational data. Therefore, a relational database was a must for us. Also, we could go with Cloud Spanner as a relational database service. We did not select it because we know that our traffic will not be high and our project will not be used other than the demo or testing purposes. Furthermore, we do not need a globally available database service, Cloud Spanner, due to the same reason [13].

Section 9 - Testing Plan

Test Cases

Manual testing is the way of testing software without using any automated testing tools [14]. We will be using manual testing to reveal the bugs throughout the implementation process. In other words, the test cases will be carried out manually by the group members.

Test ID: TC_01	**Test Category:** Functional
Test Title: Matching password and password verification fields	
Test Summary: In the registration page, the inputs entered into password and re-enter password fields should match.	

Test Steps: ● Open the Veni Vidi Movie web page. ● Navigate to the Registration page. ● Enter different inputs for password and password verification.
Expected Result: When the user enters unmatching inputs into password and re-enter password fields, an error message of "Passwords do not match" is shown on top of the input area.
Test Priority: Medium

Test ID: TC_02	**Test Category:** Functional
Test Title: Updating the movie list periodically	
Test Summary: The Cloud function triggered periodically should update the movie list.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Note the movies showing. ● Log out of the application. ● After the time period when the movies are updated periodically, open the Veni Vidi Movie web page again. ● Enter the credentials to sign in to the application (register if not registered). ● Note the movies showing. ● Compare the list of movies shown previously and now.	
Expected Result: The list of movies shown previously and now are different.	
Test Priority: Medium	

Test ID: TC_03	**Test Category:** Functional
Test Title: Sending confirmation email when a ticket is purchased.	
Test Summary: When the user purchases a ticket for a movie, an auto-generated email should be sent thanks to the Google Cloud function.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Choose a movie to watch, select the movie session, and click on the "Buy Ticket"	

button. ● Select an empty seat and click on the “Confirm Selected Seat” button. ● Check your email inbox.
Expected Result: The user receives an email about the ticket purchased for a particular movie.
Test Priority: Medium

Test ID: TC_04	**Test Category:** Functional
Test Title: Sending a reminder the day before the movie is shown.	
Test Summary: When the user has a ticket for a movie that will be shown in the next day, an auto-generated email should be sent thanks to the Google Cloud function.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Choose a movie to watch, select a movie session for tomorrow, and click on the “Buy Ticket” button. ● Select an empty seat and click on the “Confirm Selected Seat” button. ● Check your email inbox during the day	
Expected Result: The user receives a reminder email about the movie he/she will watch the next day.	
Test Priority: Medium	

Test ID: TC_05	**Test Category:** Functional
Test Title: Fetching previously watched movies correctly	
Test Summary: On the profile page of the application, all movies the user purchased a ticket for should be shown.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Click on the “My Profile” item of the navigation bar. ● Check the movies shown.	
Expected Result: All movies the user purchased a ticket for are shown without any missing one.	

Test Priority: Low

Test ID: TC_06	**Test Category:** Functional
Test Title: Fetching purchased tickets correctly	
Test Summary: On the "My Tickets" page of the application, all tickets purchased by the user should be shown.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Click on the "My Tickets" item of the navigation bar. ● Check the ticket shown.	
Expected Result: All tickets purchased by the user are shown without any missing one.	
Test Priority: Medium	

Test ID: TC_07	**Test Category:** Functional
Test Title: Disabling the selection of previously taken seats.	
Test Summary: The user should not be able to select a seat taken by another user.	
Test Steps: ● Open the Veni Vidi Movie web page. ● Enter the credentials to sign in to the application (register if not registered). ● Choose a movie to watch, select a movie session, and click on the "Buy Ticket" button. ● Try to select a previously taken seat.	
Expected Result: The seats taken are shown in a darker color to indicate that the seat has been taken before by another user and the user cannot select it again.	
Test Priority: High	

Section 10 - Demo Plan

In the final demo, we will first create a new user from the registration page and log in with the newly created user. Currently showing, movies will be displayed on the home page. We will choose a particular movie and select a movie session. The application will navigate the user to the seating selection page. We will select an empty seat, and we will enter some credit card information to complete the process of purchasing a movie ticket. The application will navigate the user to the tickets page. The ticket we just purchased will be seen on the screen. Also, when the date and session of the movie arrive, the movie will be shown on the profile page as a previously watched movie. We will simulate it by selecting a movie session within the demo duration and assuming that the user watched the movie. The user will log out after completing all these steps. Next, we will create another user from the registration page and log in using that user's credentials. We will run the Google Cloud function by triggering manually so that the movies will be updated. We might simulate the movie ticket purchase process once again.

Section 11 – References

[1] “forever”, NPM. www.npmjs.com. [Online]. Available: https://www.npmjs.com/package/forever. Accessed: March 29, 2023.
[2] “5 Step Guide To Deploy Your React App On Google’s App Engine”, Roberto Herman. medium.com. [Online]. Available: https://javascript.plainenglish.io/quickly-deploy-your-react-app-on-googles-app-engine-6bb97480cc9c. Accessed: March 29, 2023.
[3] “What is Three-Tier Architecture”, IBM, www.ibm.com. [Online]. Available: https://www.ibm.com/topics/three-tier-architecture. Accessed: March 30, 2023.
[4] “8 big advantages of using mysql,” K. Vyas. Datamation, 03-Feb-2023. [Online]. Available: https://www.datamation.com/storage/8-major-advantages-of-using-mysql/. [Accessed: 29-Mar-2023].
[5] “The Movie Database”, themoviedb.org. [Online]. Available: https://developers.themoviedb.org/3/movies/get-now-playing. Accessed: March 29, 2023.
[6] “Memory Management”, Mozilla Developer Network. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management. Accessed: March 29, 2023.
[7] “What Is Node.js and Why You Should Use It”, kinsta.com. [Online]. Available: https://kinsta.com/knowledgebase/what-is-node-js. Accessed: March 29, 2023.
[8] “Introduction to alerting”, Google Cloud Documentation. www.cloud.google.com. [Online] Available: https://cloud.google.com/monitoring/alerts. Accessed: March 29, 2023.
[9] “How to implement User Authentication using JWT (JSON Web Token) in NodeJS and maintain user sessions using Session storage in React?”, Mathursan Balatas. medium.com. [Online]. Available: https://medium.com/geekculture/how-to-implement-user-authentication-using-jwt-json-web-token-in-nodejs-and-maintain-user-c5850aed8839. Accessed: March 29, 2023.
[10] “Write HTTP functions”, Google Cloud Documentation. www.cloud.google.com. [Online]. Available: https://cloud.google.com/functions/docs/writing/write-http-functions. Accessed: March 29, 2023.
[11] “Cold Starts in Google Cloud Functions”, Mikhail Shilkov. www.mikhail.io. [Online]. Available: https://mikhail.io/serverless/coldstarts/gcp/. Accessed: March 29, 2023.
[12] “Cloud Spanner Documentation”, Google Cloud Documentation. www.cloud.google.com. [Online]. Available: https://cloud.google.com/spanner/docs. Accessed: March 29, 2023.
[13] “8 big advantages of using mysql,” K. Vyas. Datamation, 03-Feb-2023. [Online]. Available: https://www.datamation.com/storage/8-major-advantages-of-using-mysql/. Accessed: March 29, 2023.
[14] “Manual Testing Tutorial: What is, Types, Concepts”, guru99, www.guru99.com. [Online]. Available: https://www.guru99.com/manual-testing.html. Accessed: April 2, 2023.

Section 12 – Glossary

CI/CD: Continuous Integration/Continuous Delivery
REST: Representational State Transfer
ER Diagram: Entity Relationship Diagram
JWT: JSON Web Token
HTTP: Hyper-Text Transfer Protocol
JSON: JavaScript Object Notation
CRUD: Create, Read, Update, and Delete
API: Application Programming Interface
SQL: Structured Query Language
URL: Uniform Resource Locator
TC: Test Case
NPM: Node Package Manager
SSH: Secure Socket Shell