# Software Design Specification

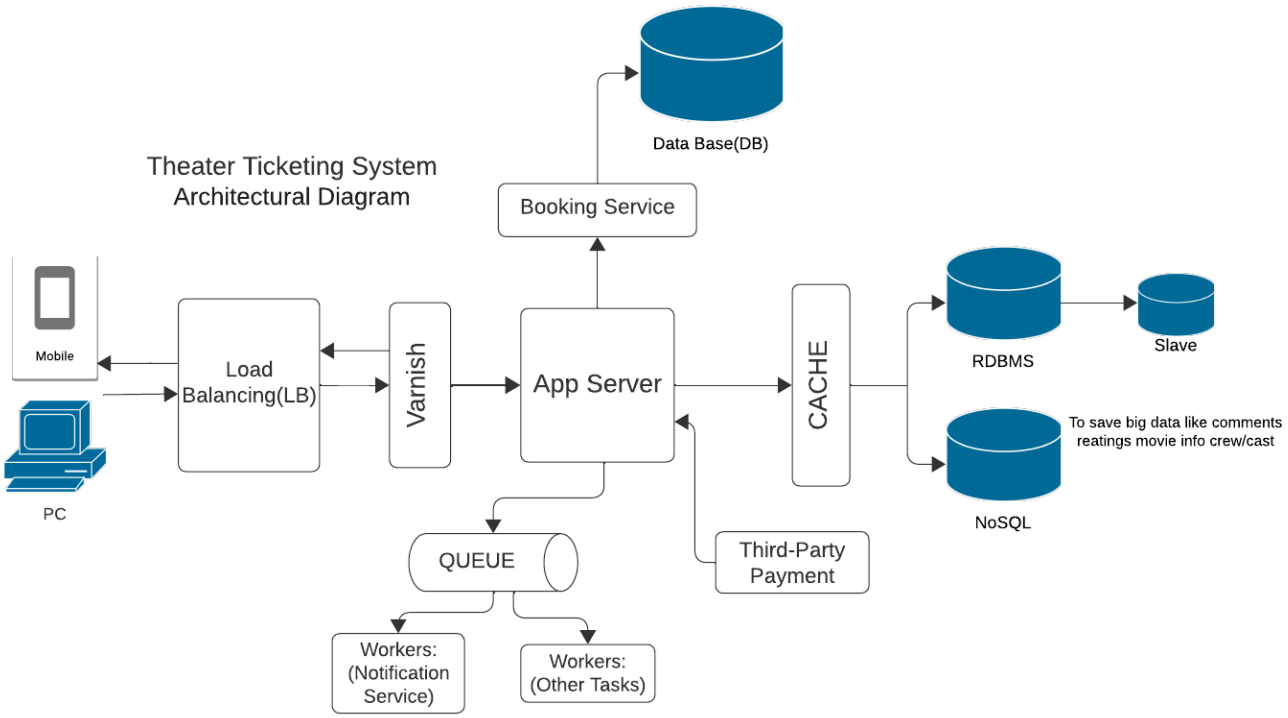
*Team Members: Diego Vasquez-Del-Mercado, and Rami Azouz*
*June, 2, 2024*

## Brief Overview of the System:

The Theater Ticketing System is an online solution aimed at streamlining the ticket purchasing process for theater performances. It offers users the ability to explore different shows, select their preferred seats, complete ticket purchases, and handle their user profiles efficiently. This document should contain all the necessary information to complete a theater ticketing system application.

Github Link: https://github.com/RamiAzouz/Theater-ticketing-system.git

# Architectural Diagram:




Software Architecture Overview:

The architectural diagram for the Theater Ticketing System shows the various components to handle that occurs when the user buys a ticket,also affecting caching, load balancing, and database interactions. The diagram is segmented into the following main components:

1. Users (Mobile and PC): These are the clients that will interact with the system. Users will order tickets by using a mobile device, a computer, or even accessing through a browser on a mobile device.
2. Load Balancing (LB): Distributes incoming requests to make sure that the servers are not overwhelmed, to ensure efficient and continuous reliability and to minimize downtime.
3. Varnish: A caching layer that helps to speed up the request processing by storing copies of frequently accessed data.

4. App Server: This is the main application, it is a server that processes requests, manages business logic, handles interactions with other services, allows for online payment or links to a third party payment service, and allows users to leave reviews that could also link to third-party review websites like Yelp or Rotten Tomatoes, and sends out conformation receipts. All coding should be compatible to handle all of these requirements and should maintain fast loading speeds.
5. Booking Service: Manages booking-related operations and interactions with the database. Should be able to access the database to check for available tickets, check the database for which seats are taken and not taken, and either allow the user to buy tickets directly online or through a third party payment process.
6. Cache: Stores frequently accessed data to reduce database load and improve performance.This is an important component of the application, because it keeps fast loading times of the system and ensures minimum downtime and should be implemented into the code with care.
7. Third-Party Payment: Handles payment processing by integrating with external payment services. This can either be directly coded into the application or be substituted with redirecting the user to a third party payment gateway(preferred, because if you have to code the payment process yourself, then you are also are responsible with keeping a secure database with the credit card details of the users, which will require programming more security protections, which will increase labor costs.)
8. Queue: Manages background tasks and notifications. This will queue up the task needed to be done in a first come first serve basis to be fair for users that process their tasks first.
9. Workers: Processes queued tasks such as sending notifications and other asynchronous operations. This refers to the "workers" portion of the architectural diagram, which will work on sending out the notifications for the users upcoming movie dates and times, and sending out email confirmation receipts to users after their payment process is complete. This step is crucial because it gives the application better reliability, by providing users with confirmations and good communication between the program and customer.
10. Database (DB): The primary data store for the application, consisting of an RDBMS and NoSQL databases. This stores all of the necessary data for the theater ticketing system, like available tickets, available seats, ticket prices, user login information, etc…
11. Slave: A replica of the primary RDBMS for read-heavy operations.

Description of Components

1. Users (Mobile and PC)
    - Description: User interfaces for accessing the theater ticketing system.
    - Attributes: Device type, IP address.
    - Operations: Send request, receive response.
2. Load Balancing (LB)

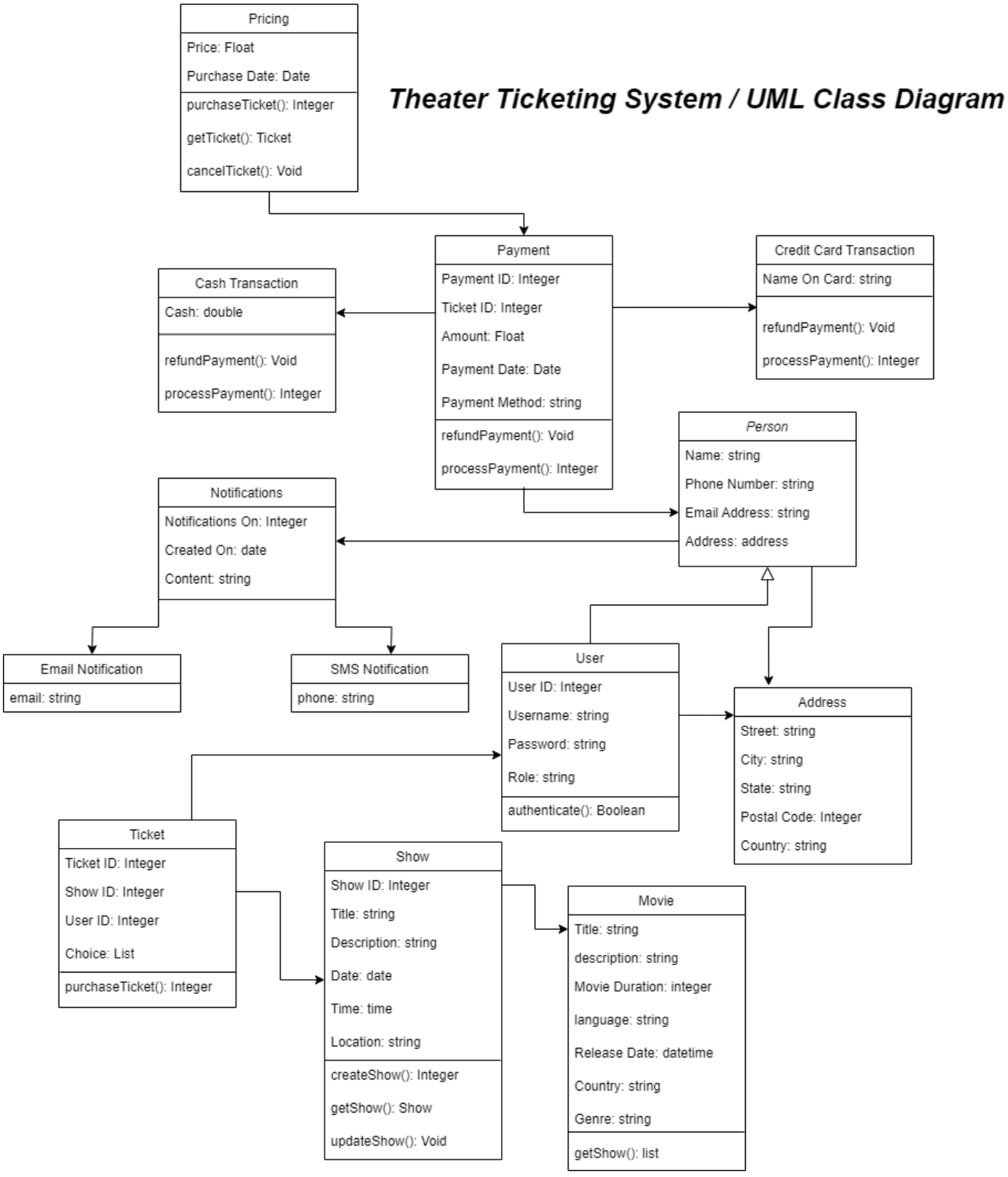
- Description: Distributes incoming traffic to multiple app servers to ensure even load distribution.
- Attributes: Load balancer ID, algorithm type.
- Operations: Distribute request, monitor server health.

3. Varnish
    - Description: A caching layer that stores copies of frequently accessed resources.
    - Attributes: Cache ID, size, TTL (Time To Live).
    - Operations: Store cache, retrieve cache, invalidate cache.
4. App Server
    - Description: The main server that handles business logic and processes requests.
    - Attributes: Server ID, CPU usage, memory usage.
    - Operations: Process request, interact with database, call external services.
5. Booking Service
    - Description: Manages ticket booking operations.
    - Attributes: Booking ID, user ID, show ID, seat number.
    - Operations: Create booking, update booking, cancel booking, retrieve booking details.
6. Cache
    - Description: Stores frequently accessed data to reduce load on the primary database.
    - Attributes: Cache key, value, expiration time.
    - Operations: Store data, retrieve data, clear cache.
7. Third-Party Payment
    - Description: Handles payment processing through external payment services.
    - Attributes: Payment ID, amount, payment method.
    - Operations: Process payment, verify payment status, refund payment.
8. Queue
    - Description: Manages background tasks and ensures they are processed asynchronously.
    - Attributes: Queue ID, task ID, priority.
    - Operations: Enqueue task, dequeue task, monitor queue.
9. Workers
    - Description: Processes tasks in the queue such as sending notifications and other background operations.
    - Attributes: Worker ID, task type, status.
    - Operations: Execute task, update task status, retry task.
10. Database (DB)
    - Description: The primary data store, consisting of both RDBMS and NoSQL databases.
    - Attributes: Database ID, type, size, connection string.

- Operations: Insert data, update data, delete data, query data.

11. Slave
    - Description: A read replica of the primary RDBMS to handle read-heavy operations.
    - Attributes: Replica ID, lag time, read-only status.
    - Operations: Synchronize with primary, handle read queries.

# *UML Class Diagram*



# Description of UML Class Diagram for Theater Ticketing System

The UML Class Diagram for the Theater Ticketing System illustrates the system's structure, showcasing various classes, their attributes, operations, and the relationships among them. Below is a summarized description of the key components:

**Class: Person**

Attributes:

- Name: String - Captures the person's name, is string to capture the users name in string format..
- Phone Number: String - Stores the person's phone number for contact purposes.
- Email Address: String - Stores the person's email address for communication.
- Address: Address - Links to the Address class to store detailed address information.

**Class: User**

Attributes:

- User ID: Integer - A unique identifier for each user, ensuring distinct user records.
- Username: String - The users login name, essential for authentication.
- Password: String - The users password, Stored securely for authentication.
- Role: String- Defines the users role, for example the user could be a customer, an employee, etc… Its important for access control or permission.

Operations:

- Authenticate(): Boolean - Verifies user credentials to allow or deny access.

**Class: Ticket**

Attributes:

- Ticket ID: Integer - A unique identifier for each ticket, ensuring distinct records.
- Show ID: Integer - Links the ticket to a specific show.
- User ID: Integer - Links the ticket to the user who purchased it.
- Choice: List - Stores seat choices or preferences.

Operations:

- purchaseTicket(): Integer - Facilitates the ticket purchasing process.

**Class: Show**

Attributes:

- Show ID: Integer - A unique identifier for each show, ensuring distinct records.
- Title: String - The title of the show, important for identification.
- Description: String - A brief overview of the show.
- Date: Date - The date of the performance.
- Time: Time - The time of the performance.
- Location: String - The venue where the show is held.

Operations:

- createShow(): Integer - Facilitates the creation of a new show.
- getShow(): Show - Retrieves details of a specific show.
- updateShow(): Void - Updates the details of an existing show.

**Class: Movie**

Attributes:

- Title: String - The title of the movie.
- Description: String - A brief overview of the movie.
- Movie Duration: Integer - The length of the movie.
- Language: String - The language of the movie.
- Release Date: DateTime - The release date of the movie.
- Country: String - The country of origin.
- Genre: String - The genre of the movie.

Operations:

- getShow(): List - Retrieves a list of shows for the movie.

Overall, these attributes and operations are designed to encapsulate the essential data and functionalities needed to manage the theater ticketing process efficiently, ensuring that all relevant aspects are covered and interactions are handled seamlessly.

### *Development plan and timeline:*

Diego was responsible for creating the architectural diagram of all major components.
Rami was responsible for developing the UML Class Diagram, including the description of classes, attributes, and operations.

1. UML Class Diagram: Will be created by Rami, estimated timeline will be 2 hours.
2. UML Description of classes, attributes and operations: Will be done by Rami, estimated timeline will be 3 hours.
3. ARCH Diagram: Will be Created by Diego, estimated timeline will be 2 hours.
4. ARCH Description: Will be done by Diego, estimated timeline will be 2 hours.
5. Document overview and Description: Will be done by both team members Rami & Diego, estimated timeline will be 2 hours.