SPORTSHUB

Contributors:

Kotha Sai Teja Reddy

Nemani Chalapathi Kiran

Pallav Rishit

Contents

REQUIREMENTS MODEL

FUNCTIONAL REQUIREMENTS

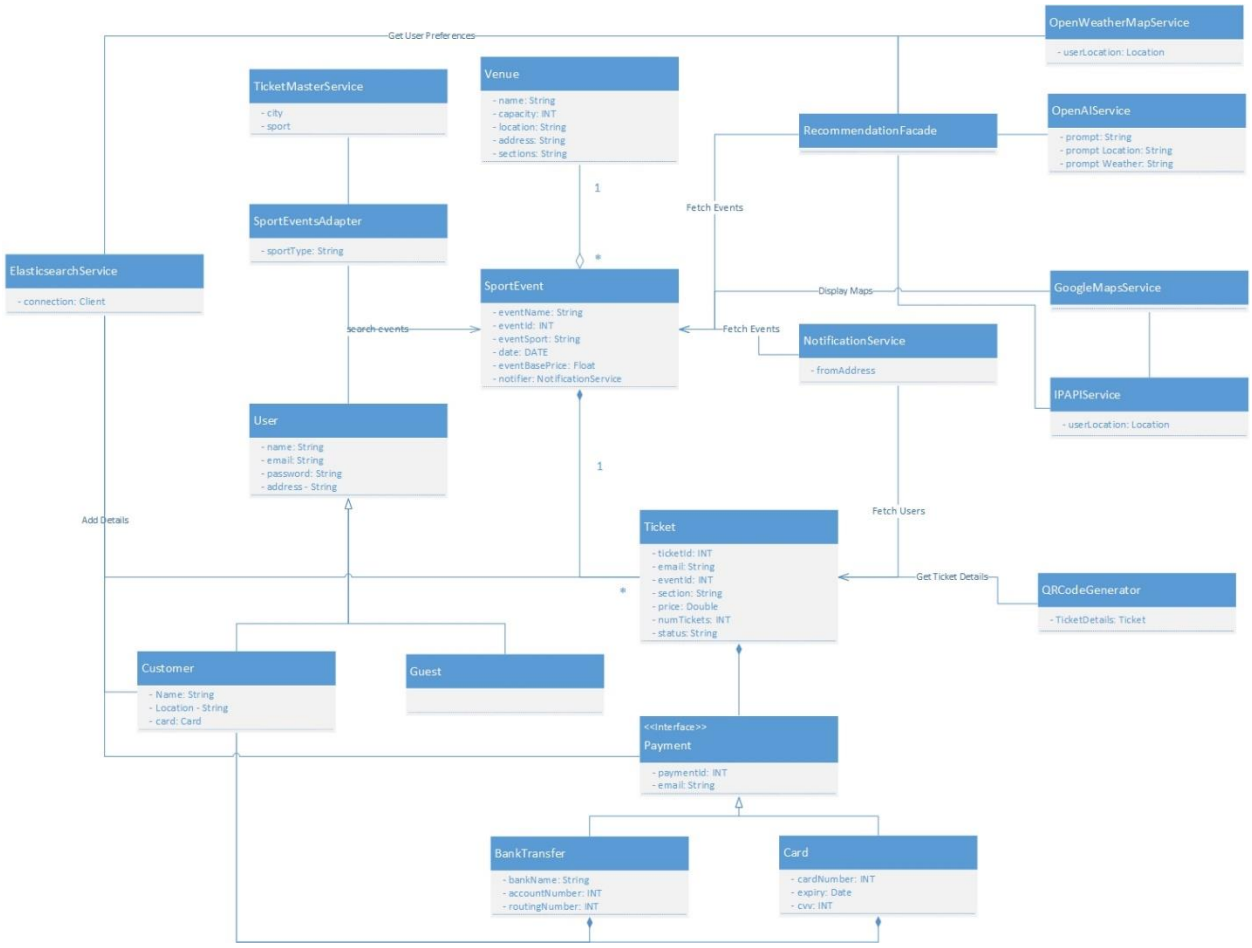
1. USER REGISTRATION
    a. User can register for an event.
2. TICKET SELECTION
    a. User can select tickets using venue sections.
3. PAYMENT PROCESSING
    a. User can purchase a ticket for an event.
    b. User can see the status of the payment.
4. SPORT EVENT BROWSING
    a. User can search for an event and browse through the list of events.
    b. Sport categories can be used to browse through the list of events.
5. ORDER MANAGEMENT
    a. Processed payment is accessible to the user via the Purchases page.
    b. User can receive status on the purchased events.
6. PROFILE MANAGEMENT
    a. User can set preferences for sport events.
    b. User can save card information for future payments.
7. USER RECOMMENDATION
    a. Recommendations are sent to user with respect to weather, location, and user preferences.
    b. User can view recommendations from OpenAI Service.
8. SPORT EVENT NOTIFICATION
    a. User can receive notifications for payments and order status.
    b. Remainders / alerts will be sent to the user after purchasing a ticket.
9. MAPS LOCATION SERVICE
    a. User can open an event and view the exact location of the venue on a map.

NON-FUNCTIONAL REQUIREMENTS

1. PERFORMANCE
    a. Handle large number of simultaneous users without significant slowdowns.
    b. System should provide fast responses for interactions such as ticket selection, payment processing, and event browsing.
    c. Pages should be optimized to ensure smooth load time across devices and network conditions.
2. USABILITY
    a. User Interface should be easy to navigate, with clean instructions, and alerts at each step.
    b. Proper User Experience methods must be used to ensure easy understanding.
    c. User should be able to navigate to the home screen from any page.
3. RELIABILITY
    a. The system should have high availability with minimal downtime for maintenance.

    b. Should handle large number of concurrent transactions without huge performance delegation.
    c. Data Integrity must be maintained through regular backups and recovery.
4. MONITORING AND LOGGING:
    a. Alerts should be generated for abnormal system behavior or security incidents.
    b. Proper monitoring mechanisms should be used to track system performance, downtime, and security incidents.
5. SCALABILITY
    a. The architecture must be designed to scale to accommodate increased traffic during peak hours.
    b. Load balancing mechanisms must be properly implemented to distribute workloads across servers.
6. SECURITY
    a. Proper protocols must be used to maintain secure communication.
    b. User Data such as credentials and payment information must be stored in encrypted format.
    c. Must follow local laws for storage and recommendation of events.

ANALYSIS MODEL



An analysis model is a simplified version of an activity diagram used for high-level modeling of business processes or system behavior.

COMPONENTS AND RELATIONSHIPS:

User: an interface that has User properties: name, email, password, and address. This is inherited by Customer and Guest Entities.

Customer: customer has name, location, and payment method. Inherits the properties of User.

Guest: An unidentified user accessing the site. Inherits the properties of a User.

SportEvent: Has detailed information on sports events and is connected to Ticket (composition) and Venue (aggregation). The Ticket cannot exist without a sport event but a Venue can exist without a sport event.

TicketMasterService: Captures data from Ticketmaster API and is connected to SportEventAdapter to organize information according to different sport events.

SportEventAdapter: Has necessary functions to organize various sport event type and send it to front end.

Venue: One venue is associated with a particular event. The connection is Aggregation.

Ticket: Purchasing ticket information is present in this model. Is connected to Payment to store payments and is connected to notification service and QR Code generator.

Payment: Is inherited by Card and BankTransfer. Has necessary details about payment information.

Card: is inherited from Payment. This is used in Customer class to store payment information for future.

BankTransfer: is inherited from Payment. This is used in Customer class to store payment information for future.

RecommendationFacade: Manages recommendations from across various ServiceAPIs. Is connected to OpenAIService, IPAPIService, and OpenWeatherMapService.

OpenAIService: Stores necessary information to connect to the client and provide queries and exchange information.

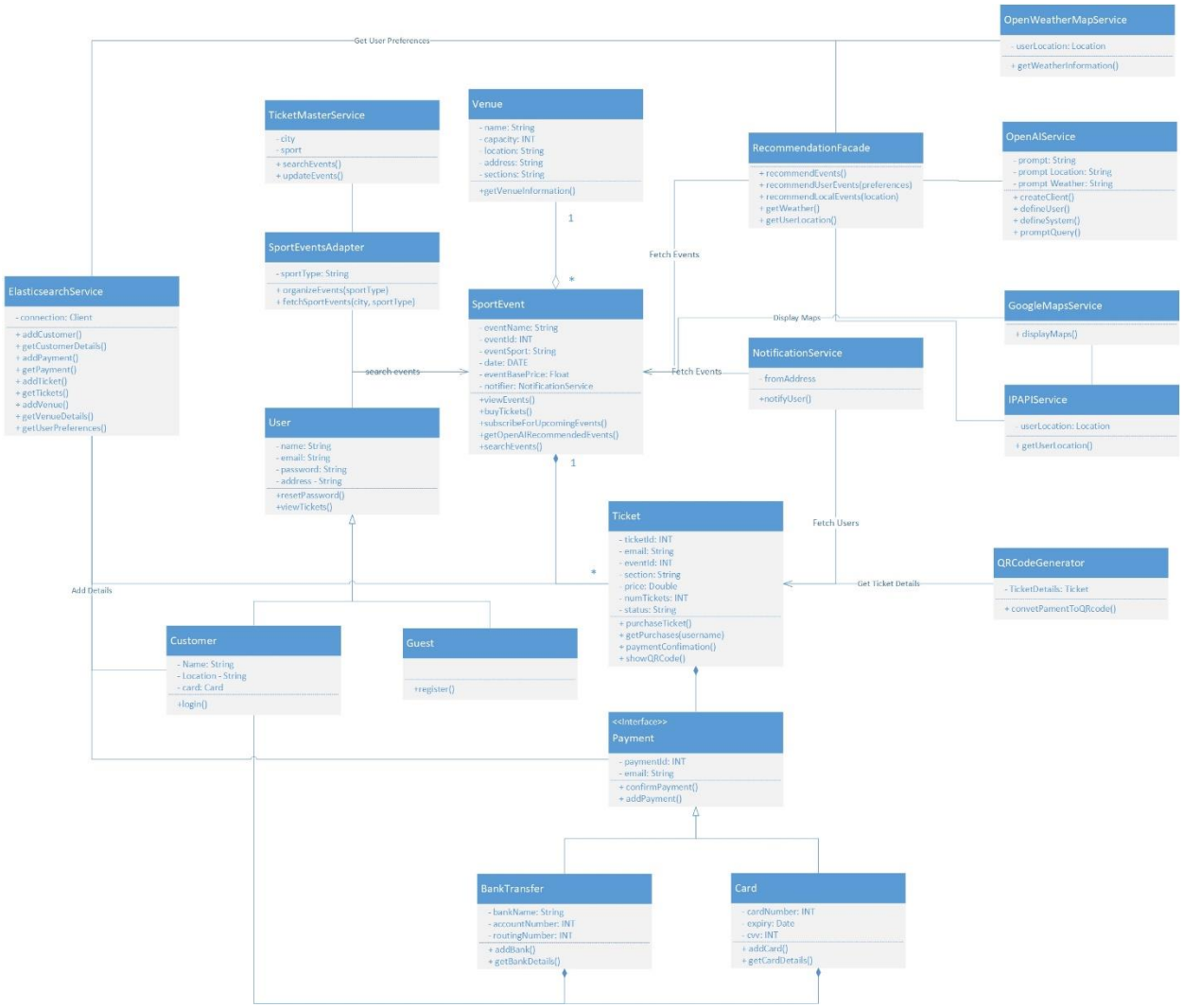
IPAPIService: gets user API and location information and this data is fetched by RecommendationFacade to process recommendations by OpenAIService.

OpenWeatherMapService: has necessary information to connect to openweathermap Api and fetch current weather information using coordinates.

NotificationService: is connected using Observer Design pattern and sends notifications to users for change in dates and ticket information.

ElasticsearchService: DataStore to store all the event and profile information. It also stores user preferences used by OpenAIService to recommend events with respect to user preferences.

DESIGN CLASS DIAGRAM



Design class diagram provides in depth information about methods and specific information about connection to each Entities and Datastores.

User: is connected to sport events to search tickets and get purchased ticket information.

Customer: has Payment information connected. Can addCard and addbank. Has an option to login.

Guest: An unidentified user accessing the site. As the user has no prior information, the user can register and enter their information.

SportEvent: Has detailed information on sports events and is connected to Ticket (composition) and Venue (aggregation). Connects to RecommendationFacade to trigger OpenAIRecommendations. Connected to TicketMasterAdapter to fetch ticket information from TicketMasterService. Is connected to elasticsearchService Datastore to store purchases and order information.

TicketMasterService: has internal functions to create client with API Key and searchEvents.

SportEventAdapter: Has necessary functions to organizeSportEvent with event type and send it to front end.

Venue: One venue is associated with a particular event. The connection is Aggregation.

Ticket: Purchasing ticket information is present in this model. Is connected to Payment to store payments and is connected to notification service and QR Code generator.

Payment: Is inherited by Card and BankTransfer. Has necessary details about payment information.

Card: is inherited from Payment. This is used in Customer class to store payment information for future.

BankTransfer: is inherited from Payment. This is used in Customer class to store payment information for future.

RecommendationFacade: Manages recommendations from across various ServiceAPIs. Is connected to OpenAIService, IPAPIService, and OpenWeatherMapService.

OpenAIService: Stores necessary information to connect to the client and provide queries and exchange information.

IPAPIService: gets user API and location information and this data is fetched by RecommendationFacade to process recommendations by OpenAIService.

OpenWeatherMapService: has necessary information to connect to openweathermap Api and fetch current weather information using coordinates.

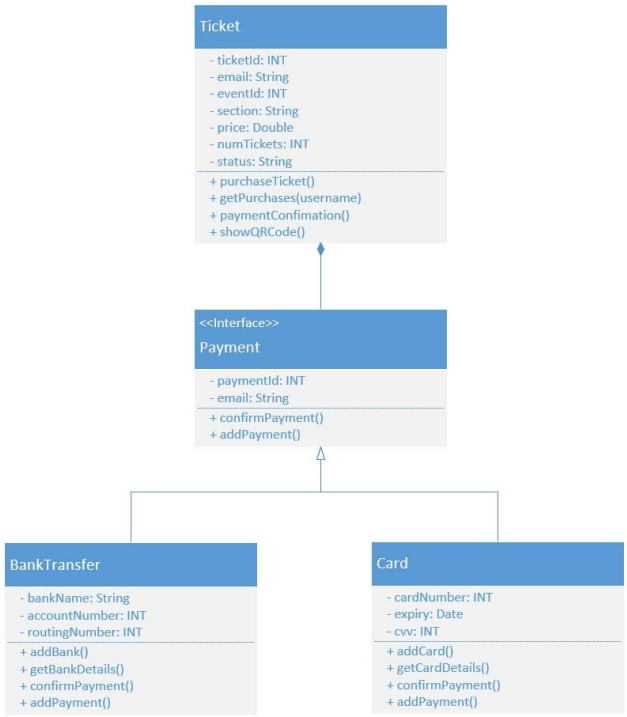
GoogleMapService: has necessary functions to trigger a map placement on the frontend with appropriate Pins. Is connected to RecommendationFacade to show recommended events on map.

NotificationService: is connected using Observer Design pattern and sends notifications to users for change in dates and ticket information.

ElasticsearchService: DataStore to store all the event and profile information. It also stores user preferences used by OpenAIService to recommend events with respect to user preferences.

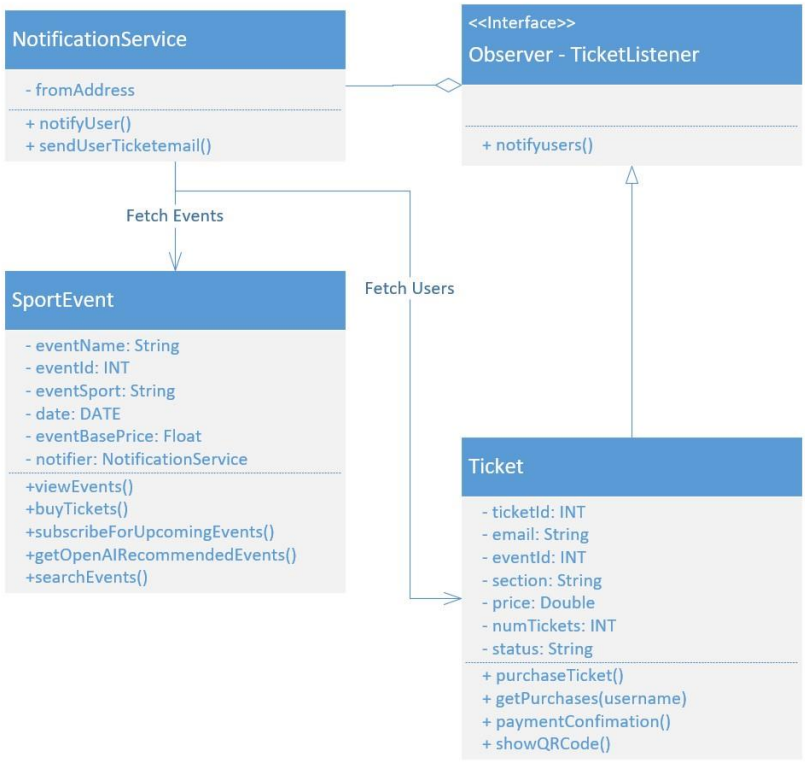
DESIGN PATTERNS USED IN PROJECT

FACTORY DESIGN PATTERN



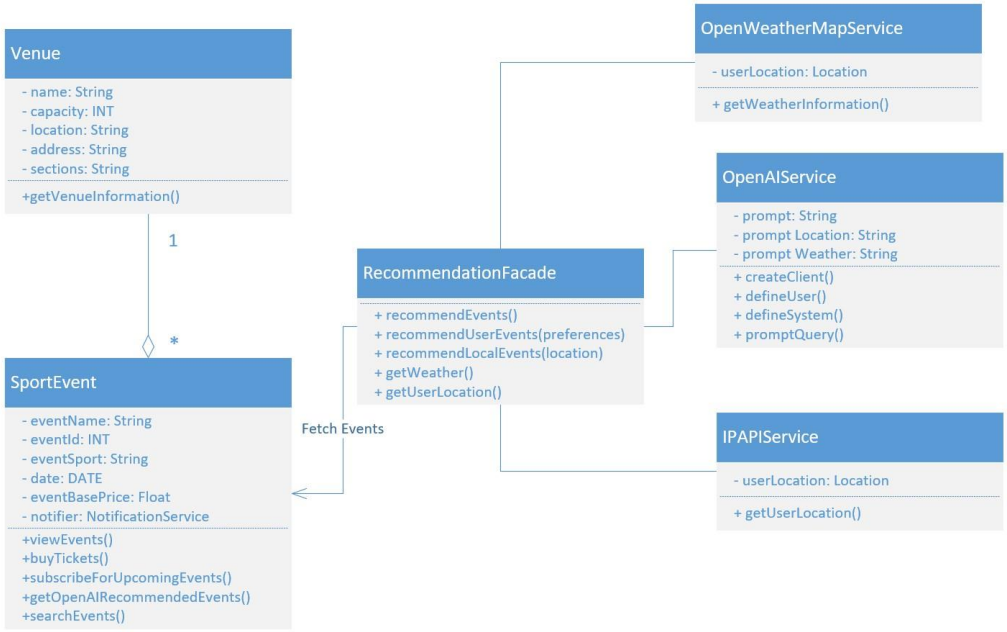
Here, Factory Design pattern is used for payment logistics of events i.e. tickets. There are many events for different sports which involve different stadiums, stadium layouts, different price rates etc. Payment for each sport event for example Football and basketball should be same and can be done by via Bank transfer or Card. The Payment interface makes the payment gateway easier by facilitating and making the bank transfer and card payment done as the same through which ticket is purchased securely and an QR code is generated.

OBSERVER DESIGN PATTERN



Here, we are using observer design pattern for out email notification about upcoming to the customers. Whenever the date of an event is approaching close i.e. when the ticket is purchased for an event, the day before the start of the event a notification is sent to the customers so that they can vary of the event happening. Here the ticketListener acts as an observer constantly fetching the event information from the database and compares it with the current data. Whenever an event is near it automatically notifies all users about the event who are having the tickets purchased with the eventide stored in the database.

FACADE DESIGN PATTERN



A Façade Design Pattern simplifies communication with the necessary services without overloading a lot of information in the calling methods. Recommendation Façade is designed to recommend events by connecting to a lot of different APIs and fetch sport events relevant to customer. In our application, RecommendationFacade is connected to IPAPI, OpenWeatherMap, and OpenAIService. SportEvent has an option to fetch getOpenAIRecommendedEvents(), the request is made to RecommendationFacade.recommendUserEvents(preferences) which is sent to OpenAIService.promptQuery() and gets all the relevant sport events according to user preferences.