Design Document

Version 1.0 approved

Prepared by the staticDev team

Yoosuf Batliwala, Raymond Lo, Jafet Ortega, Alberto Perez

CECS 491A

Cal State Long Beach

Fall 2022

Objective and Background

- The objective of this application, Howeler, is to bring local entertainment such as performing arts, fairs, and artist showcases in one place, where the main focus is local entertainment details and promotion for people to easily find on a single platform.
- The background, we want to provide a cross-platform application that will be utilized by common people of Long Beach, California. We understand that by having a cross platform application, we can reach many people in the City of Long Beach.
    - This application will be created using node.js for the backend development, and react native for the front end development.

Detailed Design

- **List of Classes:**
- **Entity**
    - **Account** - a user of the application. A user has the following attributes: Name, Email, personId, Phone Number, Postal Code, Country, Street, State,City.
    - **Event** - an Event is a type of feature that is created when a creator decides to promote local entertainment such as performing arts, fairs, and artist showcases.
- **Boundary**
    - **Mainpage -**
        - This interface will allow a user to register and login on to the application
    - **Register boundary -**
        - Allows users to create an account. The User will use a verified email (gmail,outlook,yahoo, etc). A User will create a unique username for identification on the platform. In addition to the username, a User will need to set up a password.
    - **Login boundary -**
        - This U.I will allow a User to enter their login credentials (username or email, and password).
    - **User boundary -**
        - Will allow the user to communicate with the database and obtain a user's LIKES, Subscriptions, and to access the Event Page.
    - **Events Page boundary -**
        - Will allow the User to search, refresh, and view an event.
    - **Advertiser boundary -**
        - Allows advertisers to add posts and event notifications that users can view and interact with.
    - **EventList boundary -**
        - This interface allows users to search and select events that have been posted by an advertiser.
    - **Subscription List boundary** -
        - This list will account for the number of subscriptions a user subscribes to an advertiser.

    - **LikeList boundary -**
        - This interface will display all the events a User LIKES.
    - **ViewEvent boundary -**
        - This window will contain all the details of an Event such as event ID, Event Details, and username of the advertiser who posted the event.
    - **CreateEvent boundary -**
        - This boundary allows the advertiser to communicate with users by creating an event with a title, userID, date, and description.
    - **Event boundary**
        - This boundary contains the details of an event such as title, date, and description.
    - **ViewAccount**
        - This object will be to demonstrate to a user the account information of an advertiser, such as event IDs, event details, and usernames.
- **Control Objects**

    - **Account Controller -**
        - This object has methods such as returnAccount(), createAccount(), verifyAccount(), and deleteAccount
    - **User Controller** -
        - This object controller will allow a user to retunLikes, returnSubscriptions, access eventPage(), viewAccount(), viewEvent(), like().
    - **Events Page Controller -**
        - The controller will allow a user to search, refresh, viewEvents.
    - **Advertiser Controller -**
        - This controller will allow a user to create an event, delete an event, edit and event, get the event list, and eventPage.

List of methods

UML Diagram
Front End UML:

- **verifyAccount()**
    - Used to authenticate user accounts by binding email and user generated password.
- **createAccount()**
    - Creates user accounts using users' information (state, email, etc.)
- **verifyEntry()**
    -
- **returnLikes()**
    - This method will return the number of likes a user has made to advertisers.
- **returnSubscriptions()**

    - This method will return a list of advertisers that a user has subscribed to. A subscription is where a user follows an advertiser because they are interested in the events posted by the advertiser
- **deleteAccount()**
    - Allows user to remove their account and its information associated with it
- **eventPage()**
    - This method will return the page where
- **search()**
    - Enable user to search for events listed
- **refresh()**
    - Refreshes page
- **viewEvent()**
    - Provides information about an event
- **createEvent()**
    - Makes an event using information provided by user
- **getEventList()**
    - This method will retrieve the events posted onto the event page.
- **viewAccounts()**
    - This method will allow a User to view an advertisers account within the Subscription List.
- **viewAccount()**
    - This method will allow a User to view the account of an advertiser associated with an event.
- **likeEvent**
    - This method will allow a User to like an event and store it into the Like List.
- **addToCalender()**
    - This method will store the event and it's details into the Users calendar
- **edit()**
    - This method will allow an Advertiser to edit the title, date, and description of an event.
- **delete()**
    - This method will allow an Advertiser to remove
- **subscribe()**
    - This method will allow an User to subscribe to an Advertiser.
- **viewEventList()**
    - This method will allow a User to view the Advertisers

Back End UML:

Account Controller Methods

- **returnAccount()**
    - Used to retrieve account information.
- **verifyAccount(String, String)**
    - Used to authenticate user accounts by comparing user input email and password with secure DB.

- **createAccount()**
    - Creates user accounts using users' information (state, email, etc.)
- **deleteAccount()**
    - Allows user to remove their account and the information associated with it.
- **logout ()**
    - This method would allow a user/advertiser to end a session on the application.
- **login()**
    - This method would allow a user/advertiser to begin a session on the application/

User Controller Methods

- **returnLikes()**
    - Retrieve all events liked by the user.
- **returnSubscriptions()**
    - Retrieve all advertisers subscribed to by the user.
- **eventPage()**
    - View this user's event page (includes events based on subscriptions).
- **viewEvent()**
    - View the event selected
- **like()**
    - Add event to user's like list.

Events Page Controller Methods

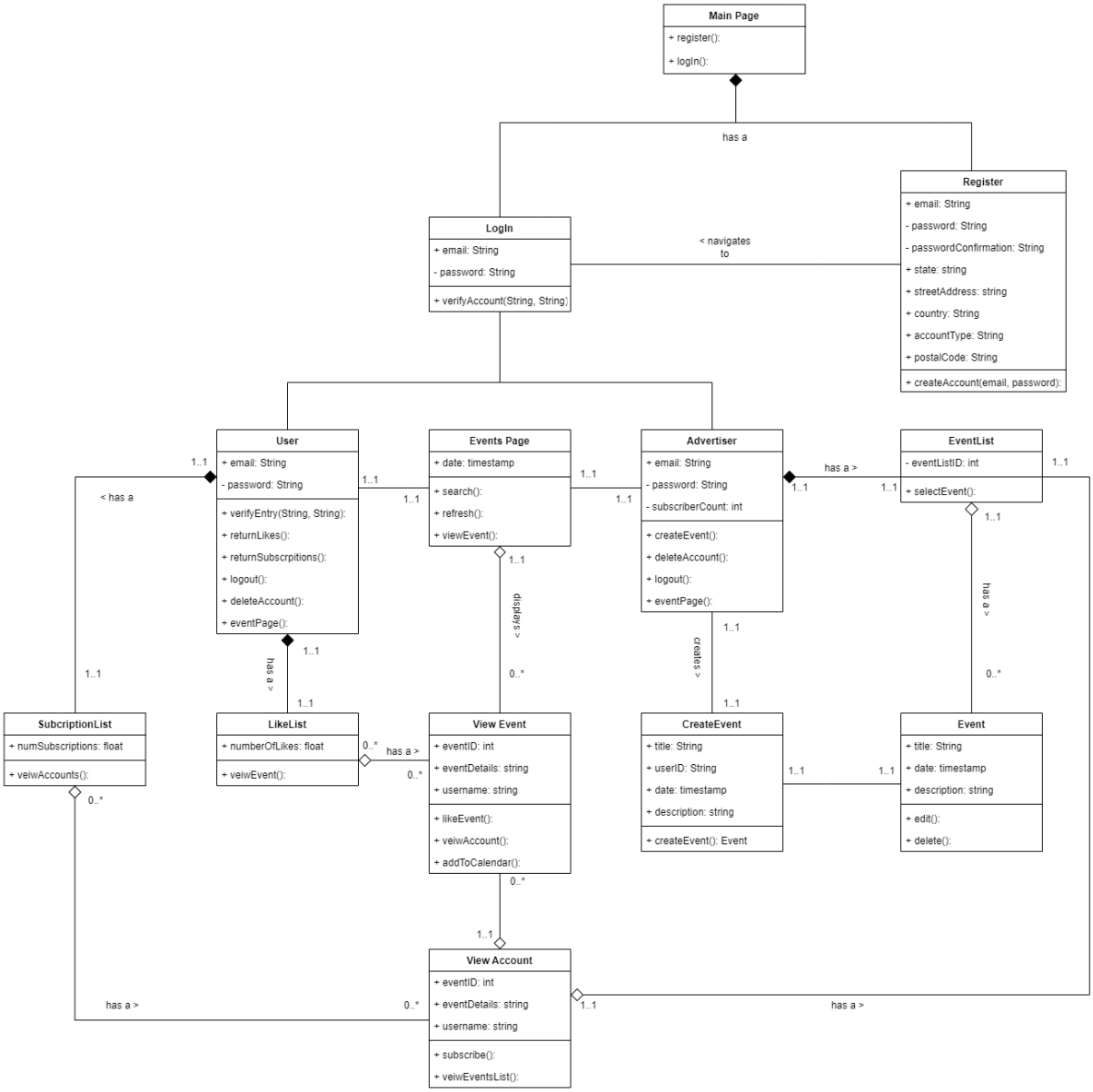
- **search()**
    - Search the page for events matching keywords.
- **refresh()**
    - Reload the page with any new events just added, or advertisers that have just been subscribed to.
- **viewEvent()**
    - Retrieve all event information including title, time and date, location, advertiser, like count, and additional details.

Advertiser Controller Methods

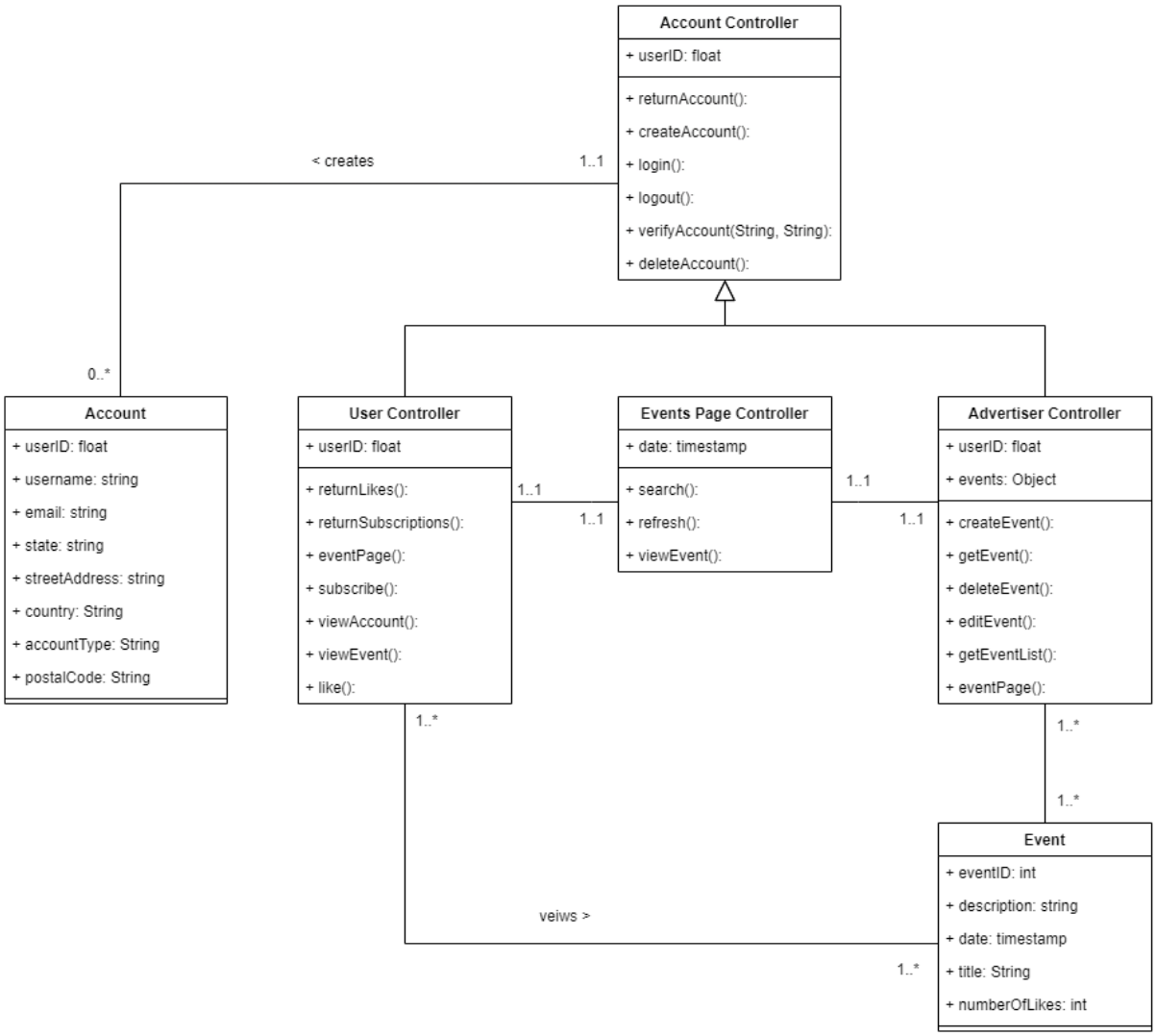
- **viewAccount()**
    - View this advertiser' account information.
- **createEvent()**
    - Create an event with user-inputted information
- **deleteEvent()**
    - Deletes the selected event from the DB.
- **editEvent()**
    - Edit the details of an event.
- **getEventList()**
    - Retrieve a list of events posted by this advertiser.
- **eventPage()**
    - Retrieves the page of events (put on by other people) the advertiser sees when logged in.

UML

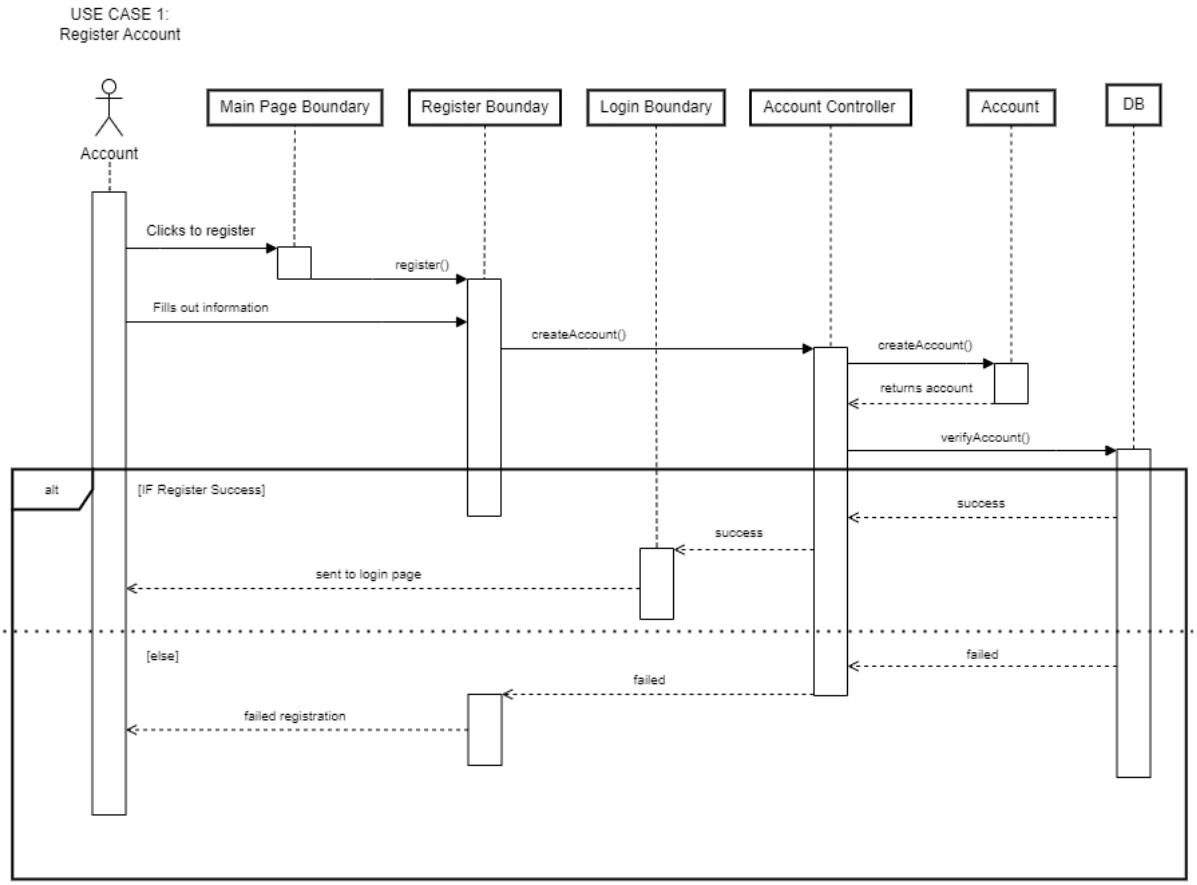
Front-End



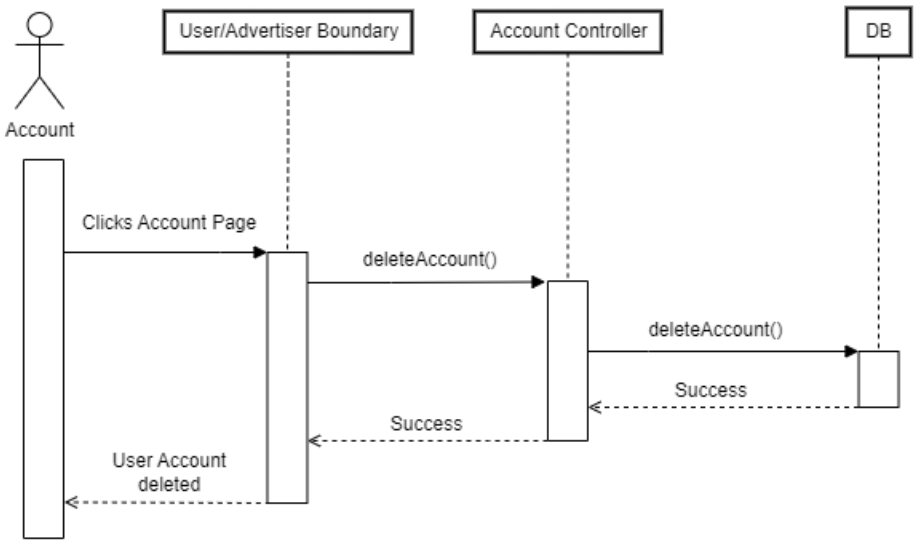
Back-End


Account Controller
+ userID: float
+ returnAccount():
+ createAccount():
+ login():
+ logout():
+ verifyAccount(String, String):
+ deleteAccount():
< creates
1..1
0..*
Account
+ userID: float
+ username: string
+ email: string
+ state: string
+ streetAddress: string
+ country: String
+ accountType: String
+ postalCode: String
User Controller
+ userID: float
+ returnLikes():
+ returnSubscriptions():
+ eventPage():
+ subscribe():
+ viewAccount():
+ viewEvent():
+ like():
Events Page Controller
+ date: timestamp
+ search():
+ refresh():
+ viewEvent():
Advertiser Controller
+ userID: float
+ events: Object
+ createEvent():
+ getEvent():
+ deleteEvent():
+ editEvent():
+ getEventList():
+ eventPage():
1..1
1..1
1..1
1..1
1..*
1..*
1..*
veiws >
1..*
Event
+ eventID: int
+ description: string
+ date: timestamp
+ title: String
+ numberOfLikes: int

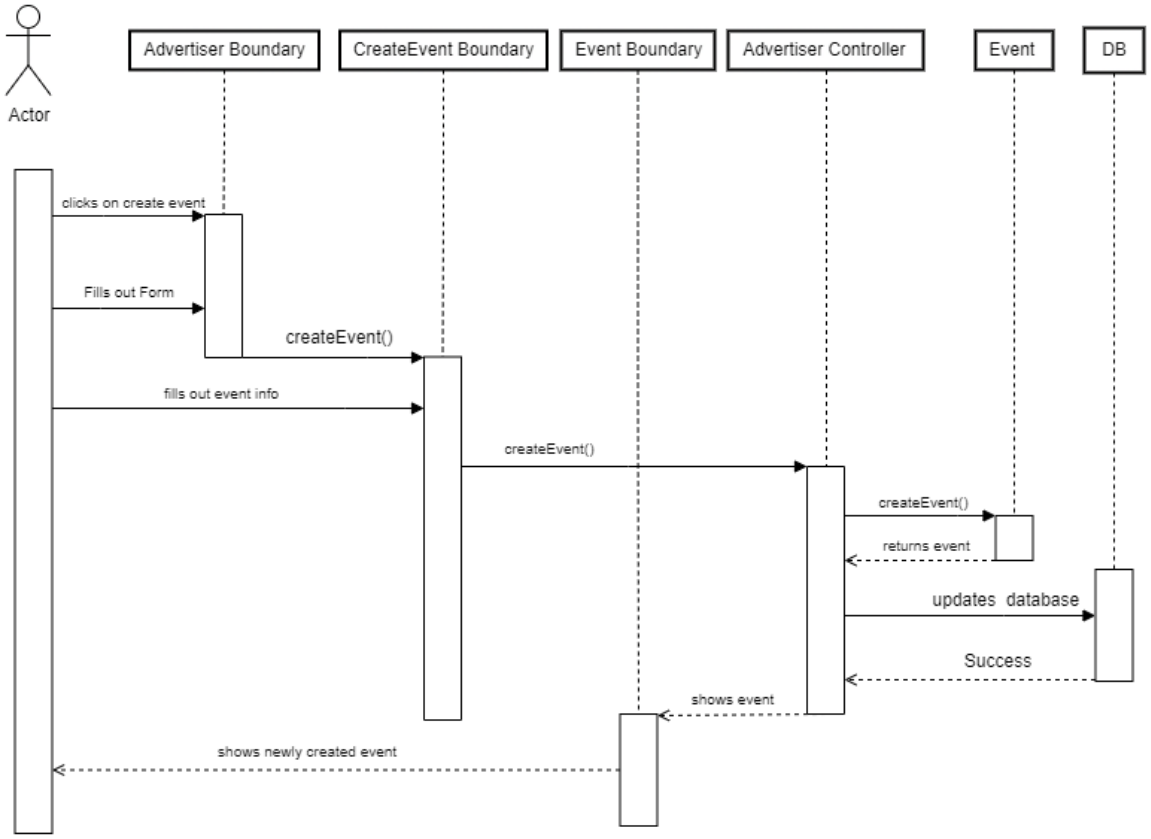
Sequence Diagram



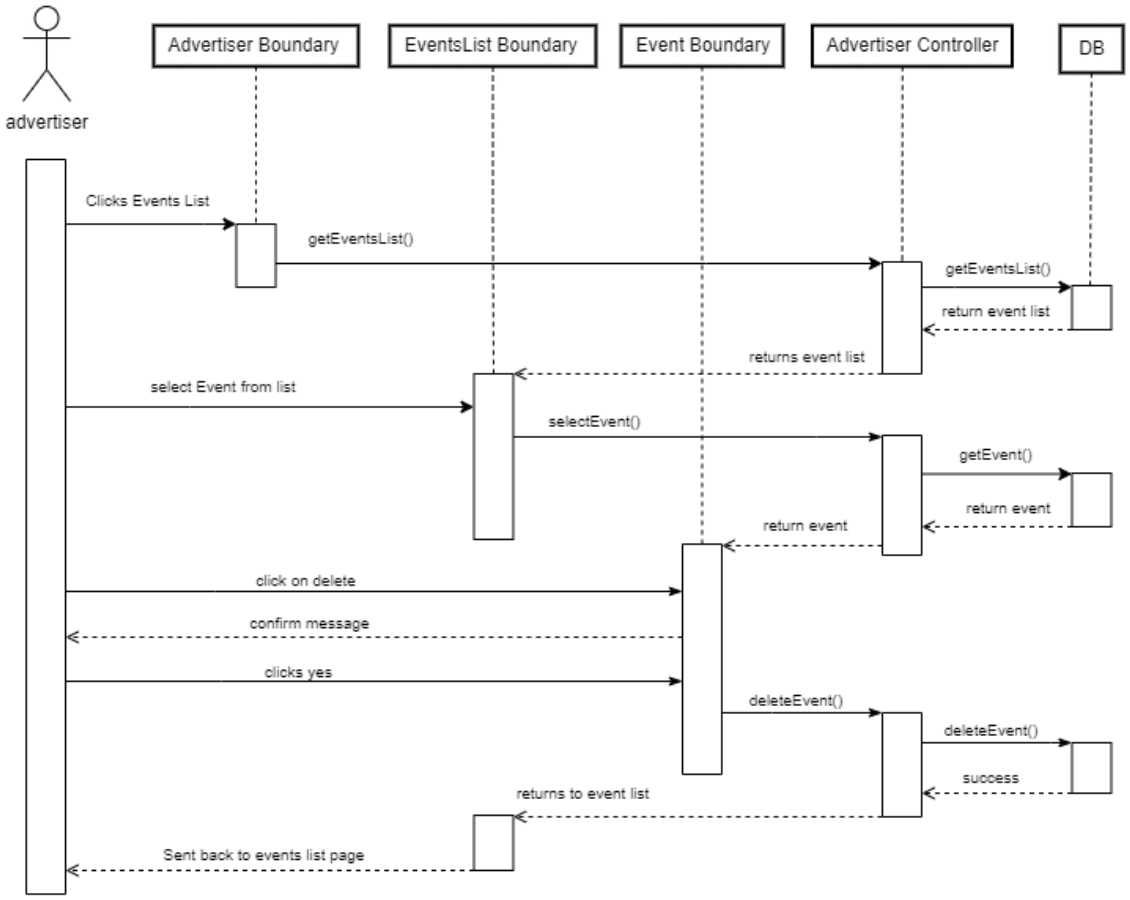
USE CASE 2:
Delete Account



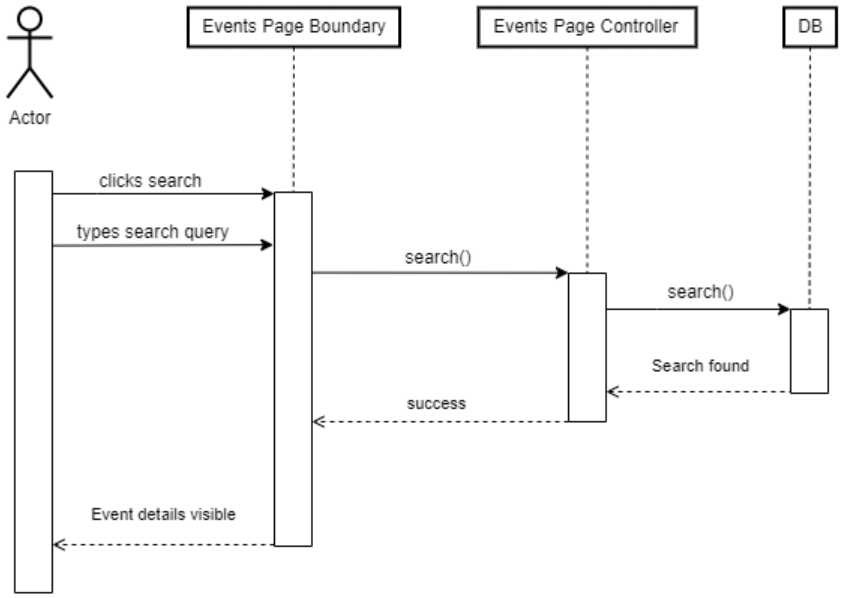
USE CASE 3:
create an event



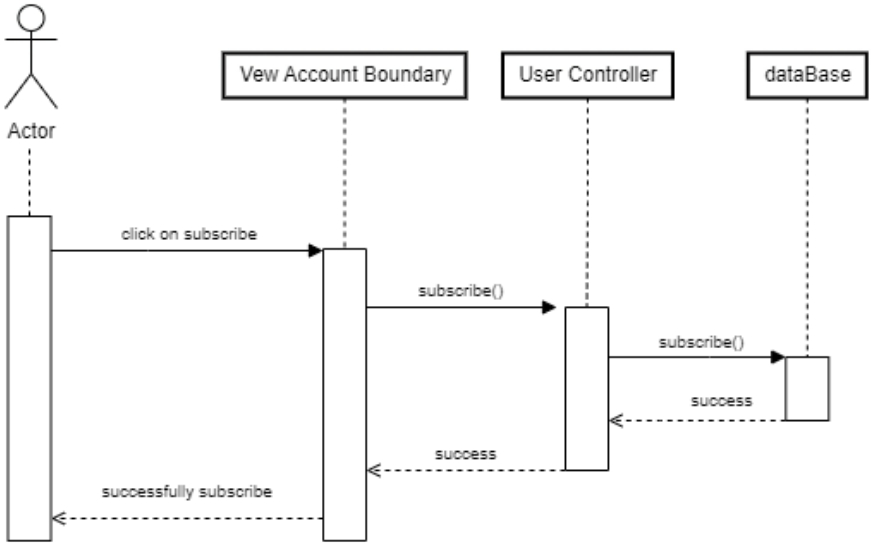
Use Case 4: delete an event

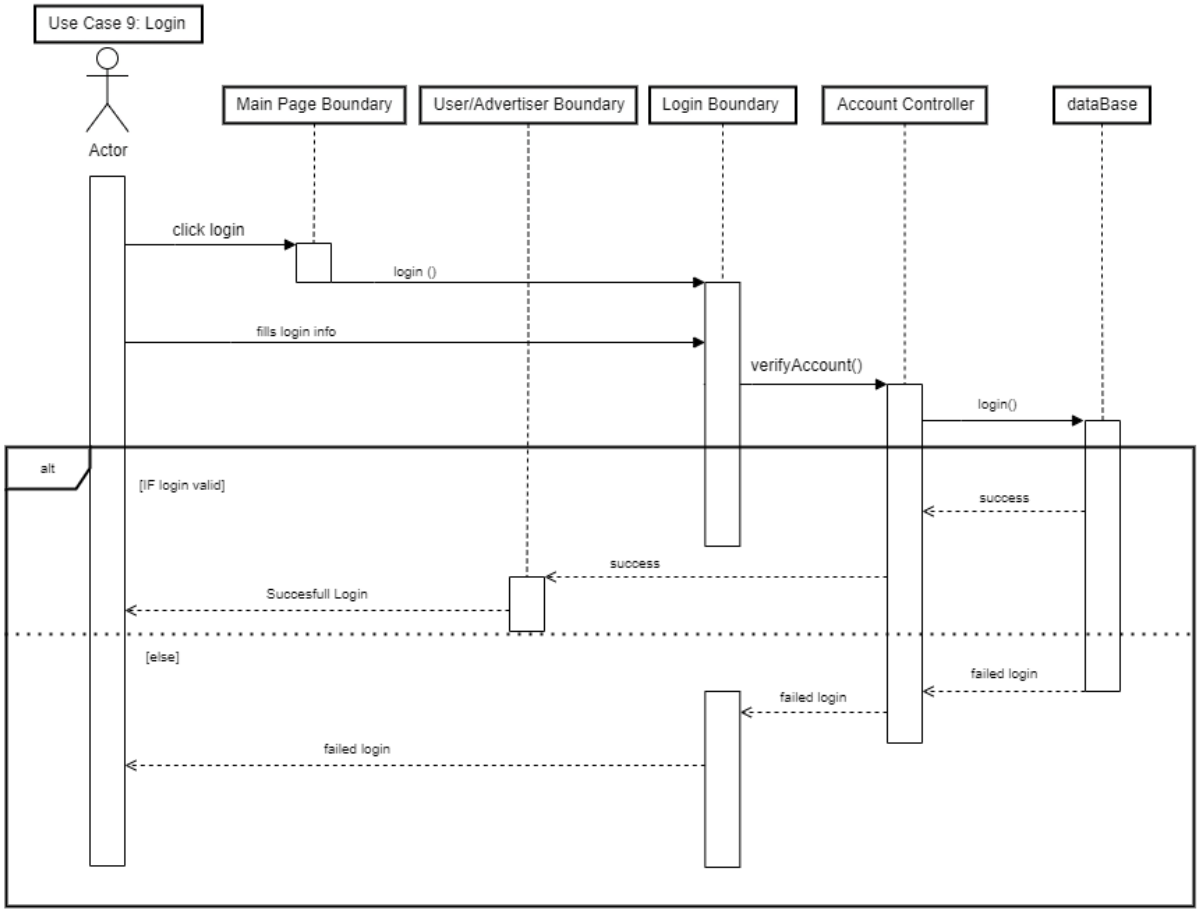


Use case 6 : Search an Event



Use Case 8: User Subscribes to Advertiser




Use Case 9: Login
Actor
Main Page Boundary
User/Advertiser Boundary
Login Boundary
Account Controller
dataBase
click login
login ()
fills login info
verifyAccount()
login()
alt
[IF login valid]
success
success
Succesfull Login
[else]
failed login
failed login
failed login

Justification and Alternatives

- Being local for Long Beach area
    - Alternatives:
        - Being local around Southern California
        - Local areas near coastline area
    - Justification:
        - Closest city around our immediate area
        - Area includes many outside events and revenues
- Usable on only mobile devices
    - Alternatives:
        - Mobile and desktop devices
        - Mini outlets
    - Justification:
        - Many users/people would most likely have some type of mobile device
        - Out of the two (mobile vs desktop), mobile use will be more useful for finding events around the area they are currently at
- Helping local musicians and entertainers
    - Alternatives:
        - Sports events
        - Major events from significant figures within the area
    - Justification:
        - Already difficult for newer musicians and of such to advertise performances
            - Popular stars already have influence to advertise themselves easily

Security

- How secure is your system?
    - User side:
        - Username and Password
        - Two factor Authentication
    - System side:
        - Obfuscating source code
        - Encryption of user data within database (if applicable)

Privacy

- How do you treat user data?
    - Hiding sensitive user data from public users
        - Phone number
        - Age
    - Encryption of user data within database (if applicable)

Reliability

- What is the chance of your system being down?
    - System required to be down during any maintenance and/or updates
    - Overwhelming amount of traffic overflows system/software

Scalability

- What does it take to scale your system?
    - Scalable database
    - Include backups of server data