SimpliEvent

https://github.com/ZiliangLi2002/OOP-Project-SimpliEvent.git

Group# 26
Yiyan (Ethan) Sha
Ziliang (Steven) Li

2023/12/16

Table of Work

(Please write x in the boxes to mention what each student achieved in this project)

	Steven Li	Ethan Sha
Project Description		x
Uses Cases Diagram(s)	x	
Sequence Diagrams	x	x
Class diagram(s)	x	
Implementation	x	x
Conclusion		x

General Description:

SimpliEvent is an event booking platform designed to streamline the process of discovering and attending events. It offers a simplified, user-centric interface for browsing and booking various events, ranging from concerts to sports games.

Key Features:

1. Simplified Event Browsing:
   - Main page showcases a list of events in a simplified format.
   - Each event preview includes a title, and brief description.
   - User can choose to display events sorted by rating.
2. Event Details:
   - Clicking on an event leads to a detailed page with specific dates, full description, and venue details.
   - Users can book event from the event detail page.
3. Search Functionality:
   - An intuitive search feature to find events by name or venue.
   - Search results display in a simplified format, like the main page.
4. View my events:
   - A dedicated notification section where users can view their saved events.
   - User can delete an event they saved previously.
5. User Profile
   - A section that displays the user information
   - Users are allowed to edit the user information.

Benefit and Advantages:

From an OOP perspective, SimpliEvent's architecture adheres to the SOLID principles, ensuring a robust and maintainable codebase:

Single Responsibility Principle: By utilizing distinct classes like EventBook and UserBook, SimpliEvent's design ensures that each module is responsible for a single aspect of the application's functionality. This separation of concerns means that changes in one part of the system have minimal impact on others, leading to easier maintenance and scalability.

Open/Closed Principle: The application's classes are open for extension but closed for modification. For instance, new features such as additional event categories or user profile options can be added without altering existing code, by extending current functionalities.

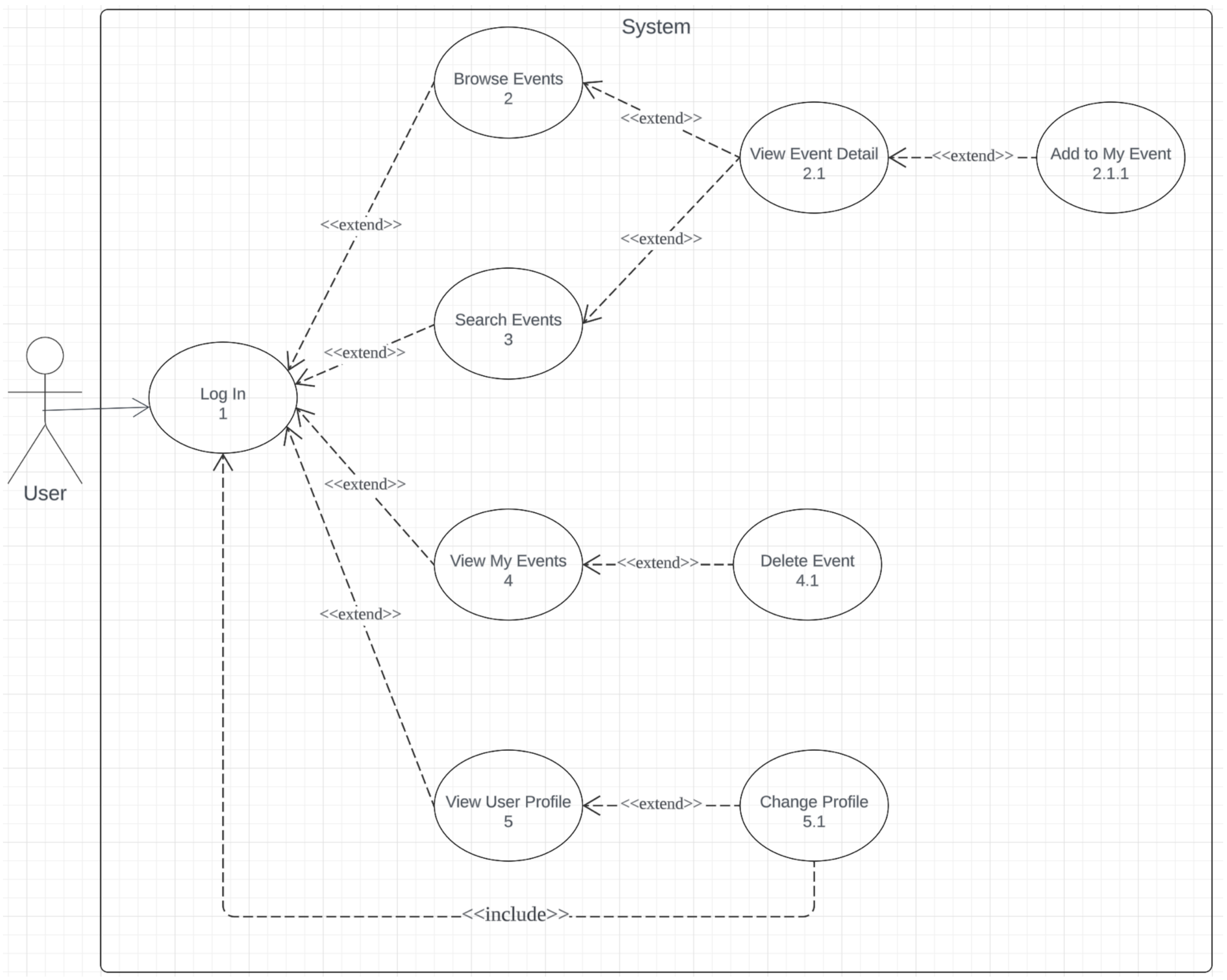
Liskov Substitution Principle: SimpliEvent's use of inheritance and interfaces ensures that objects can be replaced with instances of their subtypes without affecting the correctness of the program. This principle is particularly evident in the GUI components, where user interface elements can be seamlessly interchanged or updated.

Interface Segregation Principle: The application's design favors many client-specific interfaces over one general-purpose interface. Each GUI component, like those for searching or viewing events, is tailored to its specific use case, ensuring that classes only implement the behaviors they need.

Dependency Inversion Principle: High-level modules in SimpliEvent do not depend on low-level modules like the data access layers directly. Instead, they rely on abstractions that do not depend on details, but rather, details depend on abstractions. This approach is evident in the way EventBook and UserBook handle data, providing a flexible foundation that can adapt to changes in data storage or retrieval mechanisms.

By upholding these SOLID principles, SimpliEvent not only delivers a dynamic and user-focused service but also maintains a codebase that is resilient and adaptable, catering to both current user needs and future expansions. Security measures are woven into the fabric of the application, safeguarding user data and ensuring a trustworthy platform for all event booking activities.

Use Case Diagrams



UC Reference Name/Number: Log in/1

Overview: After entering the correct username and password, user will log in successfully and be able to see the home screen.

Related use cases: Browse Events, Search Events, View My Events, View User Profile

Actors: User

UC Reference Name/Number: Browse Events/2

Overview: User can see all the events listed. For each event, information of name, date, venue, and rating is displayed.

Related use cases: Log In, View Event Detail.

Actors: User

UC Reference Name/Number: View Event Detail/2.1

Overview: User can view the information of an event. Event name, date, venue, and a detailed description is displayed.

Related use cases: Browse Events, Add to My Event, Search Events

Actors: User

UC Reference Name/Number: Add to My Event/2.1.1

Overview: User can add an event to “My Events”.

Related use cases: View Event Detail

Actors: User

UC Reference Name/Number: Search Events/3

Overview: User can search for an event by name or venue.

Related use cases: Log In, View Event Detail.

Actors: User

UC Reference Name/Number: View My Events/4

Overview: User can view all the events that are added to “My Events”

Related use cases: Log In, Delete Event

Actors: User

UC Reference Name/Number: Delete Event/4.1

Overview: User can remove an event from “My Events”

Related use cases: View My Events

Actors: User

UC Reference Name/Number: View User Profile/5

Overview: User can View their information of username and email.

Related use cases: Log In, Change Profile

Actors: User

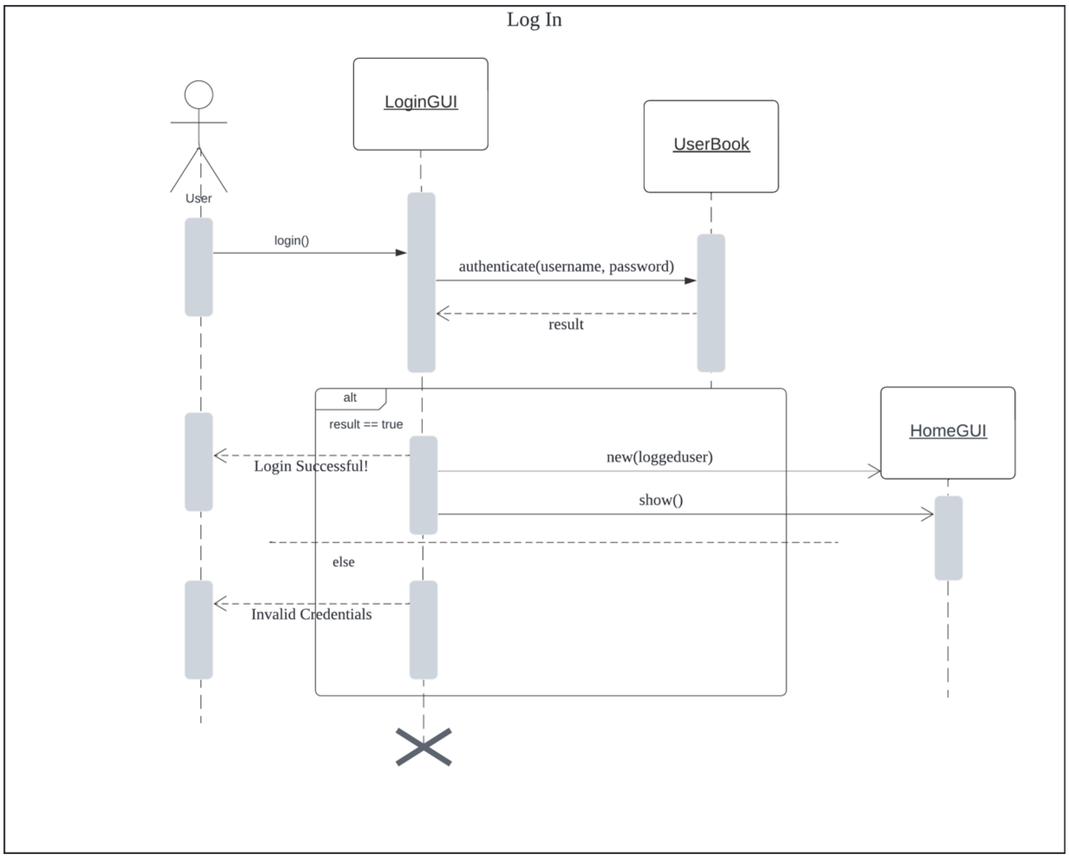
UC Reference Name/Number: Change Profile/5.1

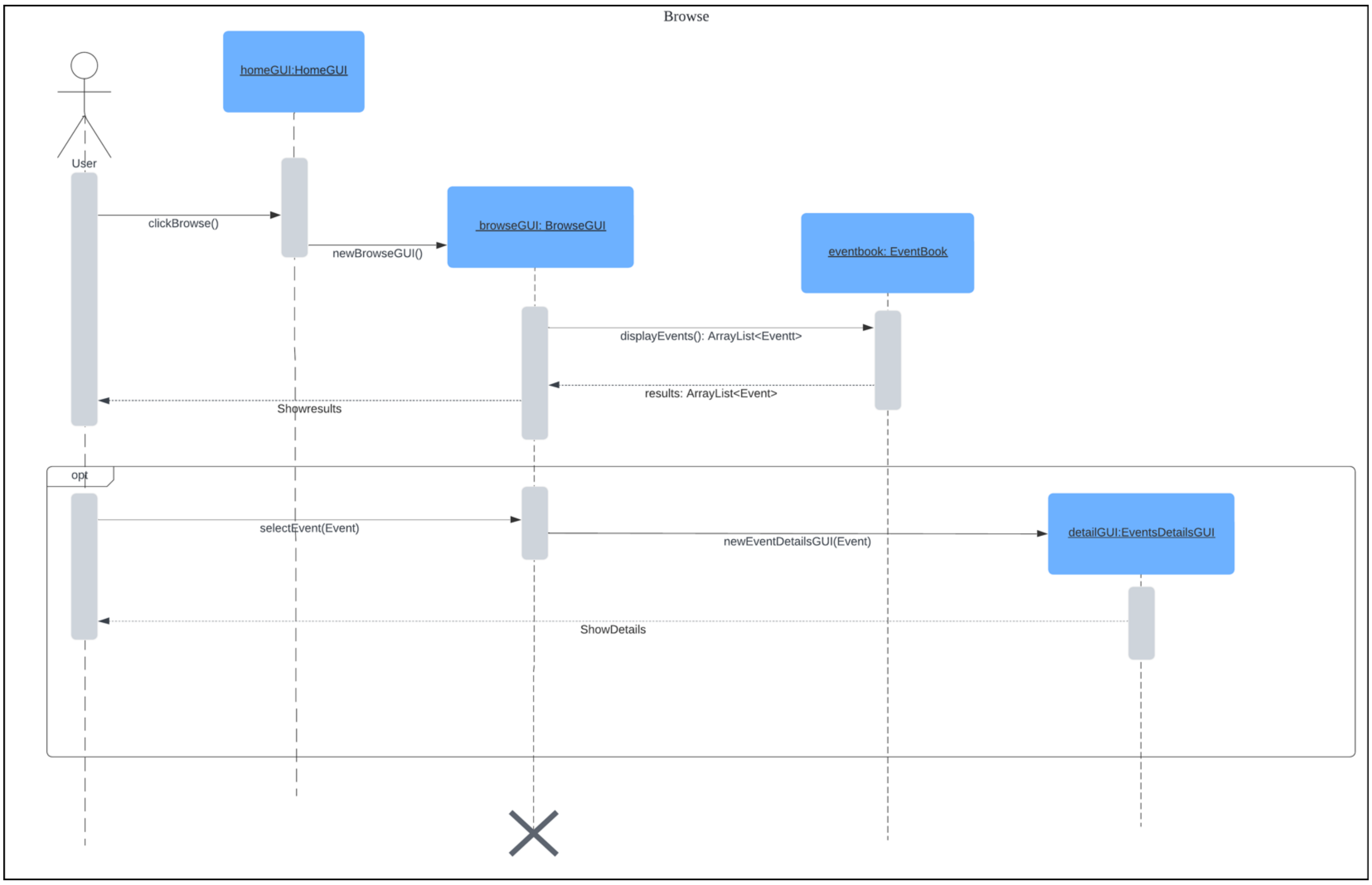
Overview: User can change the username, email, or password he uses for log in. After changing profile, the system will log out automatically, and the user will need to log in again.

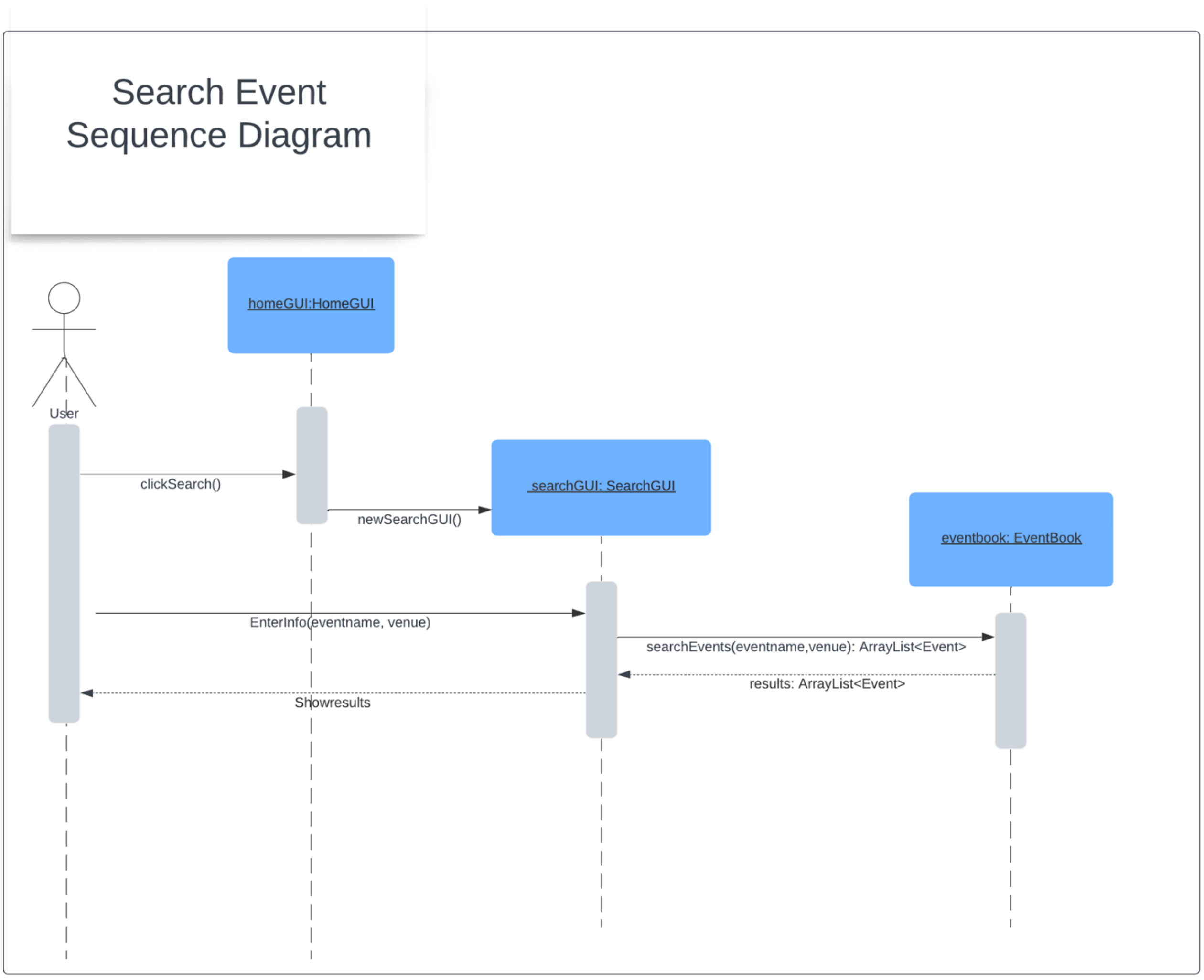
Related use cases: View User Profile, Log In

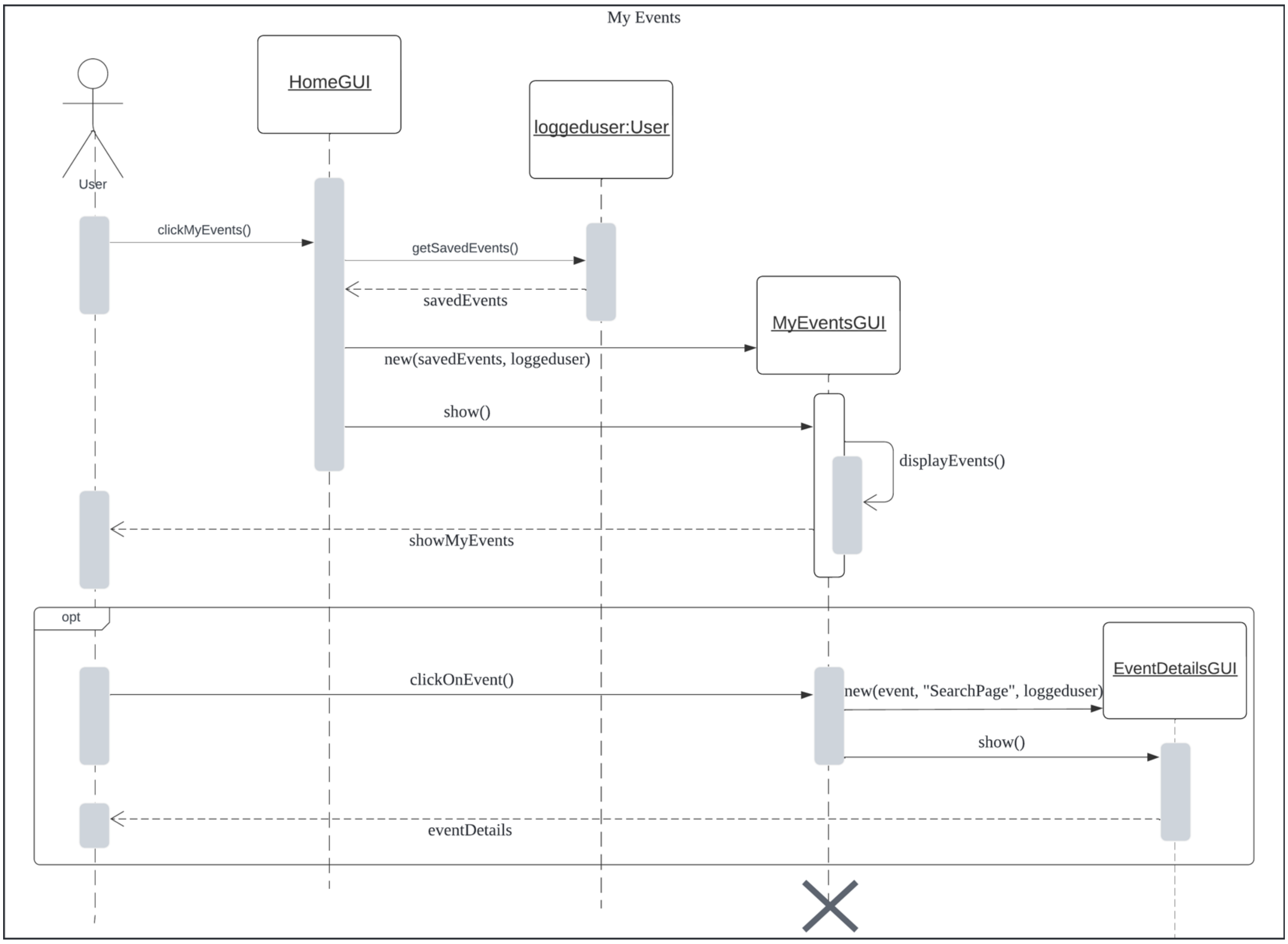
Actors: User

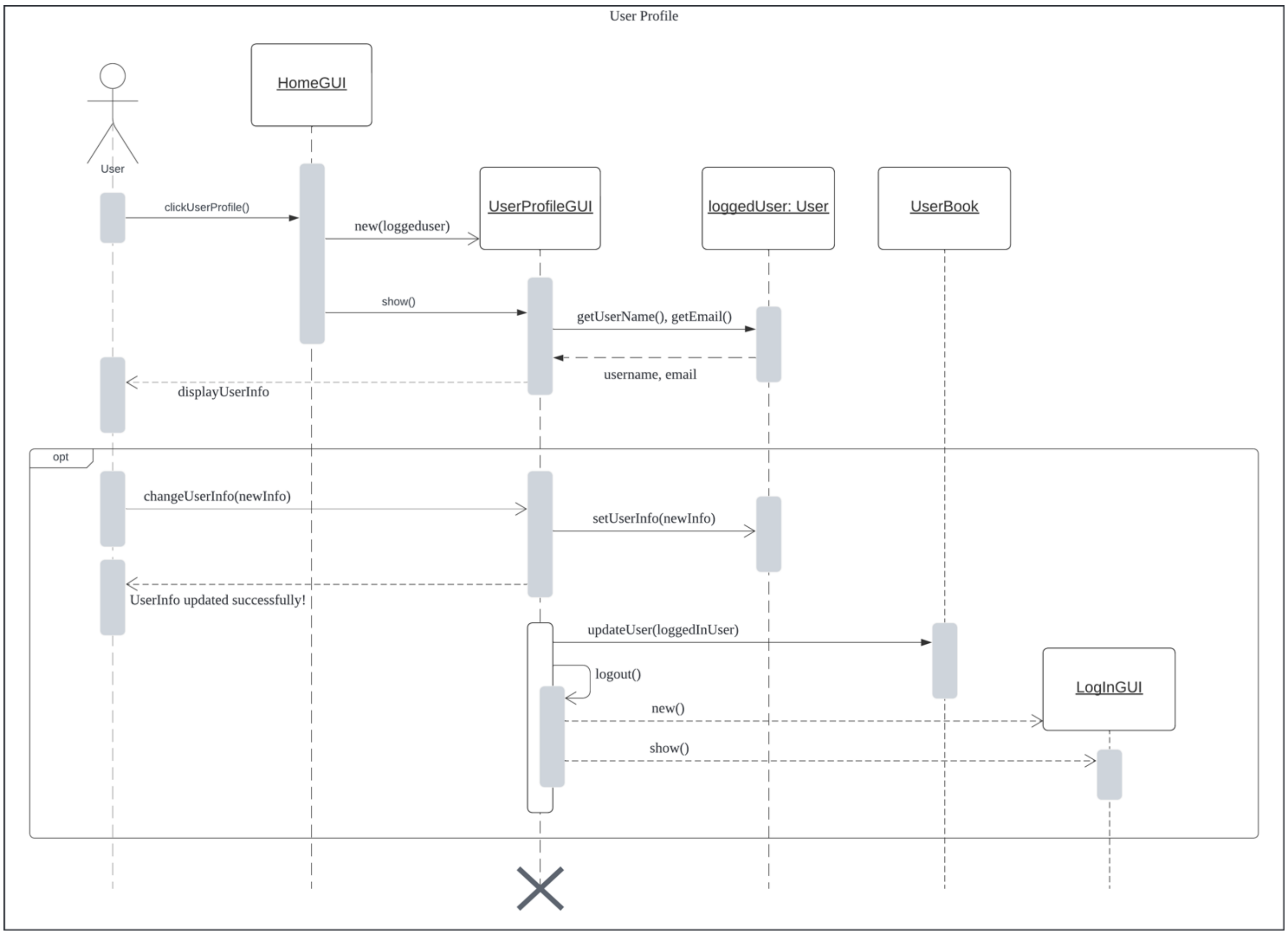
Sequence Diagrams

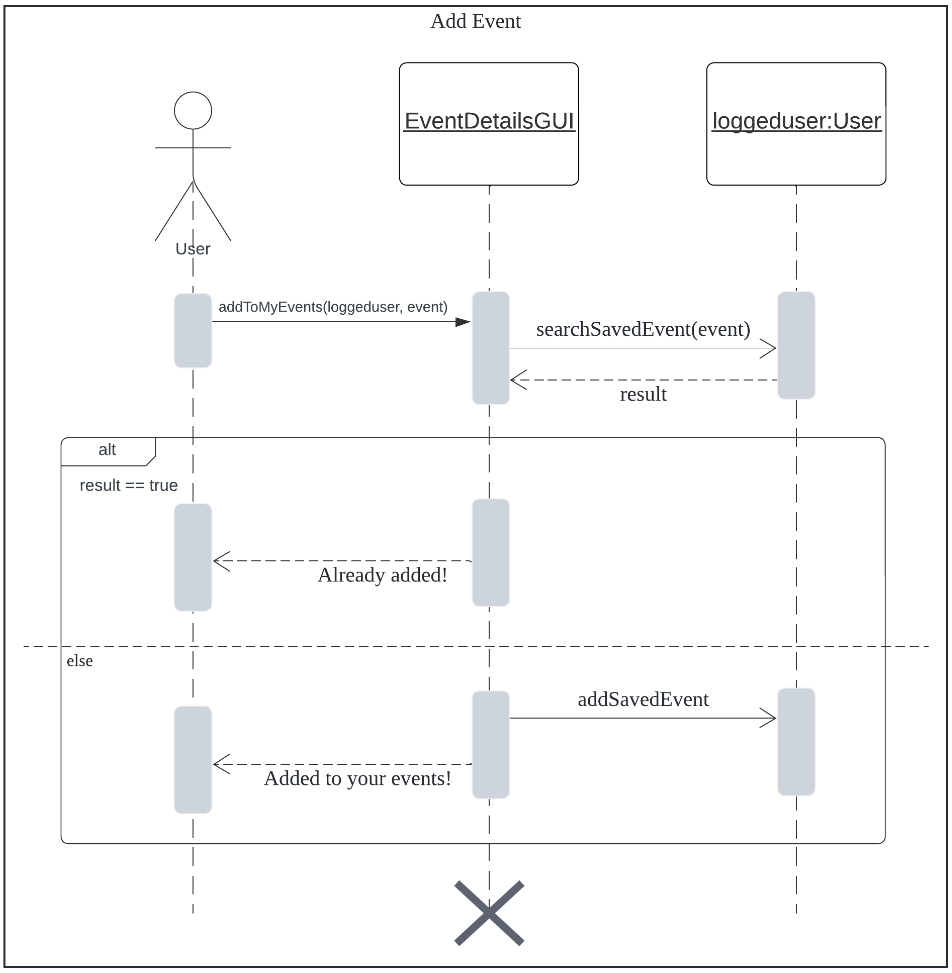


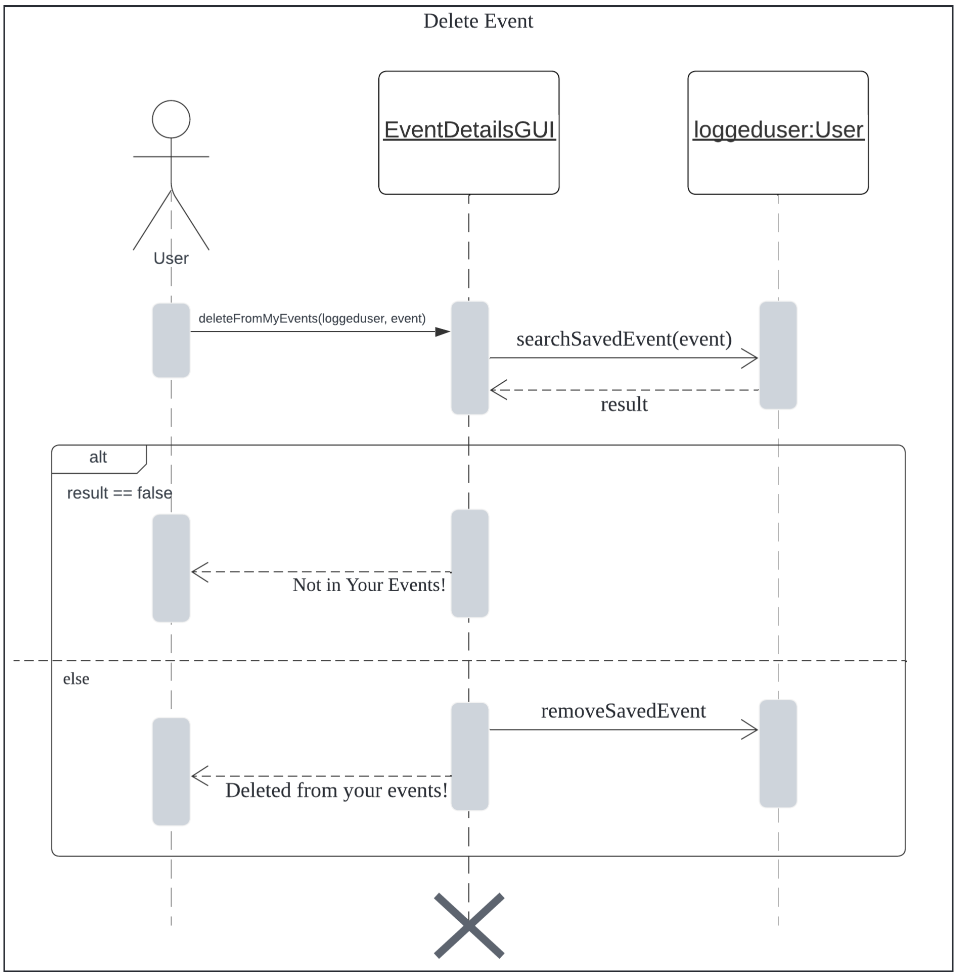

Browse
User
homeGUI:HomeGUI
browseGUI: BrowseGUI
eventbook: EventBook
clickBrowse()
newBrowseGUI()
displayEvents(): ArrayList<Eventt>
results: ArrayList<Event>
Showresults
opt
selectEvent(Event)
newEventDetailsGUI(Event)
detailGUI:EventsDetailsGUI
ShowDetails


Search Event Sequence Diagram
User
homeGUI:HomeGUI
searchGUI: SearchGUI
eventbook: EventBook
clickSearch()
newSearchGUI()
EnterInfo(eventname, venue)
searchEvents(eventname,venue): ArrayList<Event>
results: ArrayList<Event>
Showresults

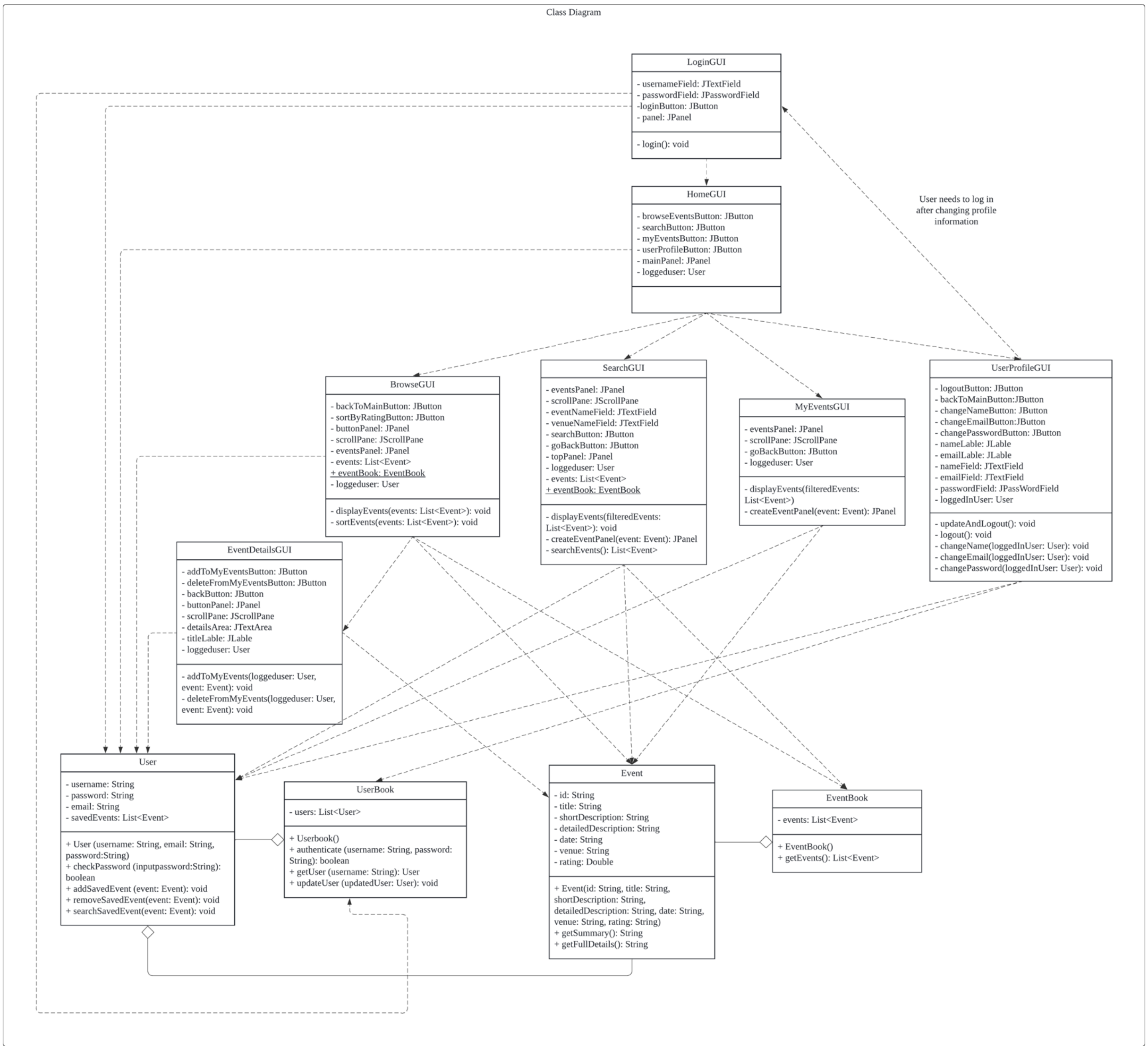

My Events
User
HomeGUI
loggeduser:User
clickMyEvents()
getSavedEvents()
savedEvents
MyEventsGUI
new(savedEvents, loggeduser)
show()
displayEvents()
showMyEvents
opt
EventDetailsGUI
clickOnEvent()
new(event, "SearchPage", loggeduser)
show()
eventDetails


User Profile
User
HomeGUI
UserProfileGUI
loggedUser: User
UserBook
clickUserProfile()
new(loggeduser)
show()
getUserName(), getEmail()
username, email
displayUserInfo
opt
changeUserInfo(newInfo)
setUserInfo(newInfo)
UserInfo updated successfully!
updateUser(loggedInUser)
logout()
LogInGUI
new()
show()


Add Event
EventDetailsGUI
loggeduser:User
User
addToMyEvents(loggeduser, event)
searchSavedEvent(event)
result
alt
result == true
Already added!
else
addSavedEvent
Added to your events!


Delete Event
EventDetailsGUI
loggeduser:User
User
deleteFromMyEvents(loggeduser, event)
searchSavedEvent(event)
result
alt
result == false
Not in Your Events!
else
removeSavedEvent
Deleted from your events!

Class Diagram


Class Diagram
LoginGUI
- usernameField: JTextField
- passwordField: JPasswordField
-loginButton: JButton
- panel: JPanel
- login(): void
HomeGUI
- browseEventsButton: JButton
- searchButton: JButton
- myEventsButton: JButton
- userProfileButton: JButton
- mainPanel: JPanel
- loggeduser: User
User needs to log in after changing profile information
BrowseGUI
- backToMainButton: JButton
- sortByRatingButton: JButton
- buttonPanel: JPanel
- scrollPane: JScrollPane
- eventsPanel: JPanel
- events: List<Event>
+ eventBook: EventBook
- loggeduser: User
- displayEvents(events: List<Event>): void
- sortEvents(events: List<Event>): void
SearchGUI
- eventsPanel: JPanel
- scrollPane: JScrollPane
- eventNameField: JTextField
- venueNameField: JTextField
- searchButton: JButton
- goBackButton: JButton
- topPanel: JPanel
- loggeduser: User
- events: List<Event>
+ eventBook: EventBook
- displayEvents(filteredEvents: List<Event>): void
- createEventPanel(event: Event): JPanel
- searchEvents(): List<Event>
MyEventsGUI
- eventsPanel: JPanel
- scrollPane: JScrollPane
- goBackButton: JButton
- loggeduser: User
- displayEvents(filteredEvents: List<Event>)
- createEventPanel(event: Event): JPanel
UserProfileGUI
- logoutButton: JButton
- backToMainButton:JButton
- changeNameButton: JButton
- changeEmailButton:JButton
- changePasswordButton: JButton
- nameLable: JLable
- emailLable: JLable
- nameField: JTextField
- emailField: JTextField
- passwordField: JPassWordField
- loggedInUser: User
- updateAndLogout(): void
- logout(): void
- changeName(loggedInUser: User): void
- changeEmail(loggedInUser: User): void
- changePassword(loggedInUser: User): void
EventDetailsGUI
- addToMyEventsButton: JButton
- deleteFromMyEventsButton: JButton
- backButton: JButton
- buttonPanel: JPanel
- scrollPane: JScrollPane
- detailsArea: JTextArea
- titleLable: JLable
- loggeduser: User
- addToMyEvents(loggeduser: User, event: Event): void
- deleteFromMyEvents(loggeduser: User, event: Event): void
User
- username: String
- password: String
- email: String
- savedEvents: List<Event>
+ User (username: String, email: String, password:String)
+ checkPassword (inputpassword:String): boolean
+ addSavedEvent (event: Event): void
+ removeSavedEvent(event: Event): void
+ searchSavedEvent(event: Event): void
UserBook
- users: List<User>
+ Userbook()
+ authenticate (username: String, password: String): boolean
+ getUser (username: String): User
+ updateUser (updatedUser: User): void
Event
- id: String
- title: String
- shortDescription: String
- detailedDescription: String
- date: String
- venue: String
- rating: Double
+ Event(id: String, title: String, shortDescription: String, detailedDescription: String, date: String, venue: String, rating: String)
+ getSummary(): String
+ getFullDetails(): String
EventBook
- events: List<Event>
+ EventBook()
+ getEvents(): List<Event>

Conclusion

This project has trained our abilities of system design, analysis, and implementation. We trained our skills in creating GUIs and writing codes that adheres to principles of Object Oriented Programming. One of the difficulties we faced was the complexity arising from numerous interrelated classes. However, through meticulous planning and rigorous analysis, we managed to generate a clear class diagram that ensures the smooth completion of our project.