Introduction: Architectural Representation

This SAD summarizes the architecture from multiple views. These include:

- Logical View: Includes UML diagrams, class diagrams of major elements, and general commentary on the large-scale structure and functionality of major components
- Development View
- Deployment View: Includes deployment diagrams showing the nodes and allocation of processes and components as well as commentary on the networking
- Process View: Includes UML class and interaction diagrams illustrating the processes and threads of the system
- Security View: Includes an overview of the security considerations and highlights major points within the architecture where they are applied
- Use Case View: Includes a summary of the most architecturally significant use cases.

This SAD will also include the following architecture-related commentaries:

Architectural Decisions

Includes technical memos detailing the decisions made regarding architecture

Architectural Factors: Includes SSDs of major use cases and commentaries explaining the underlying processes

Architectural Decisions
This ticketing system uses Spring Boot and Maven for cleaner code and easy integration of APIs. The APIs used were chosen specifically to better suit the requirements of the client.

Motivation
Ease of Use: The application must be easy for future developers to understand and work with. It must also be intuitive and smooth for end users.

Security: A very strict set of password requirements were absolutely, undeniably required by the client.

Solution:

The application was built as simply as possible. A MongoDB database was implemented using Spring Boot.

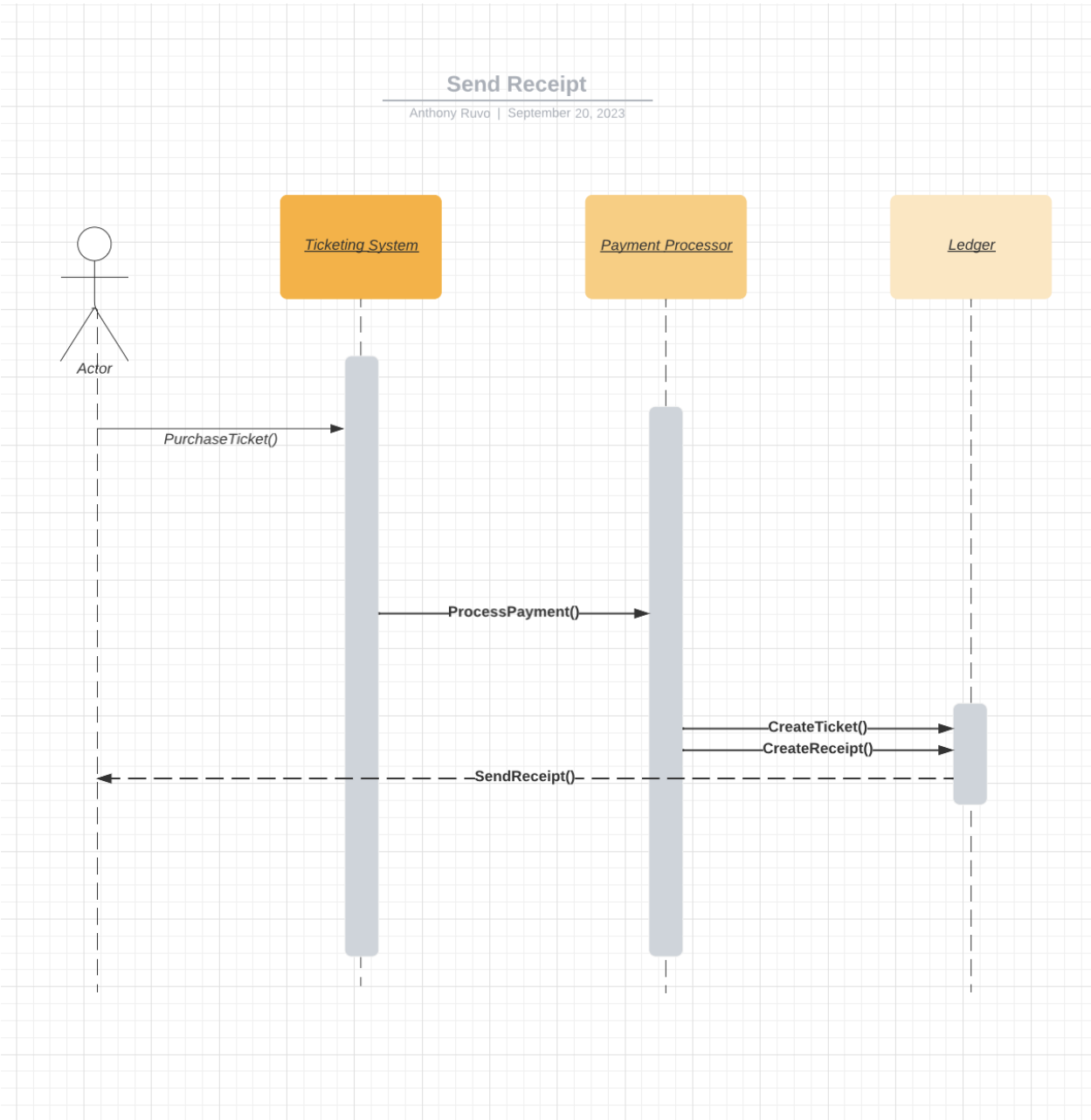
A highly complex and extremely secure password is required by the software to ensure maximum security.

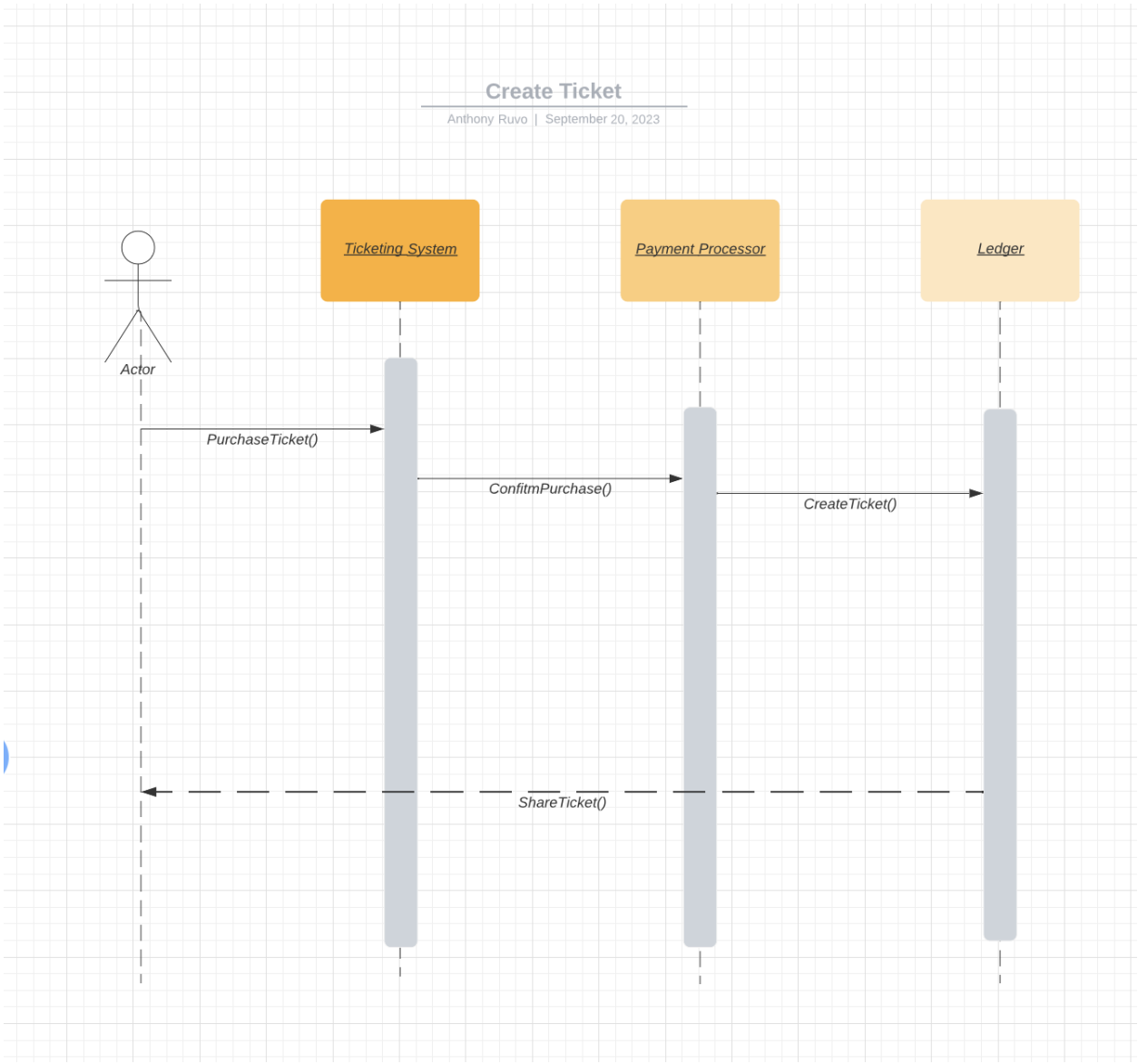
Supplemental Specifications
Account: Custom Java class that contains user information

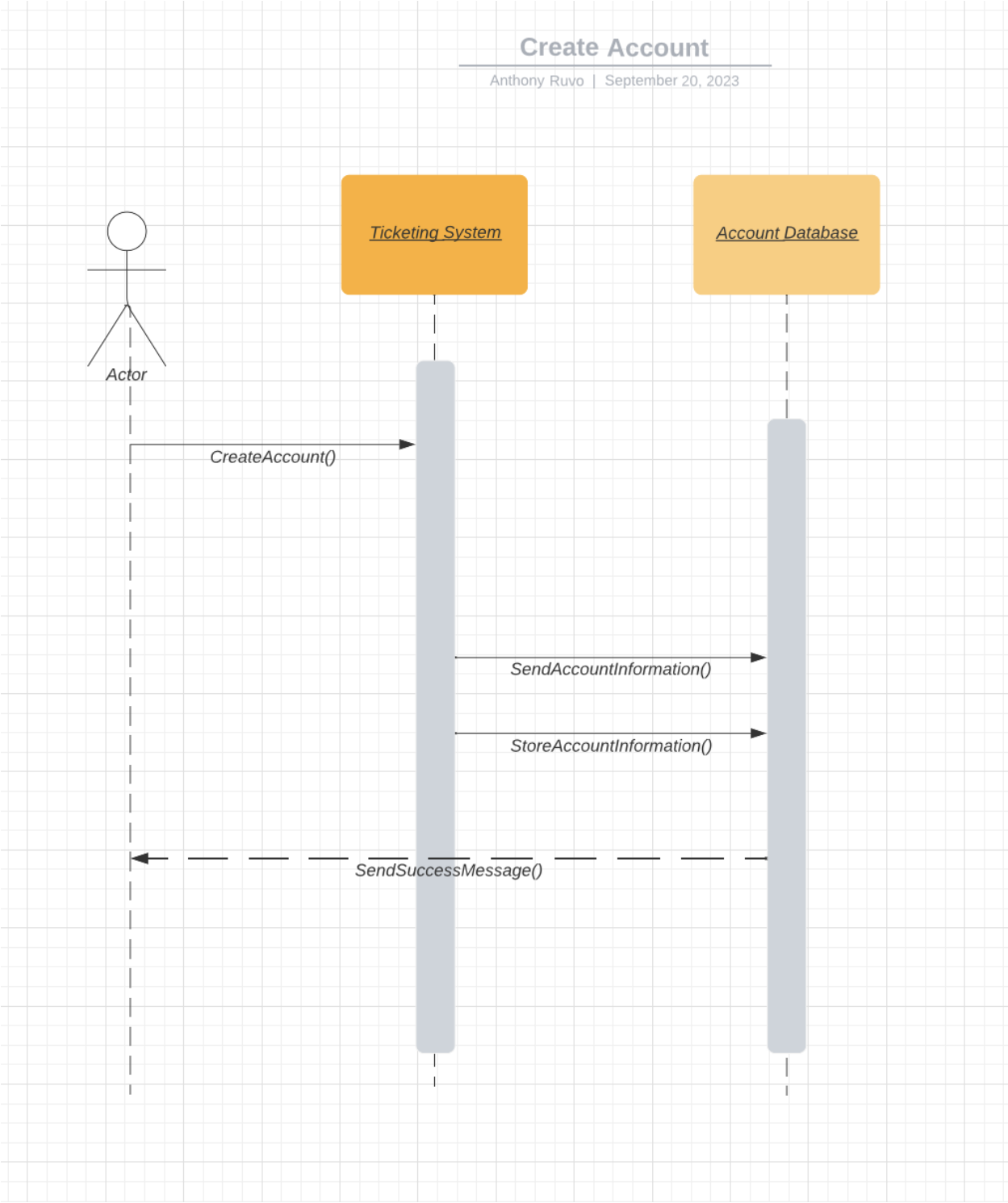
Extra Secure Password: At least 15 characters and includes a lowercase letter, an uppercase letter, a number, and a special character

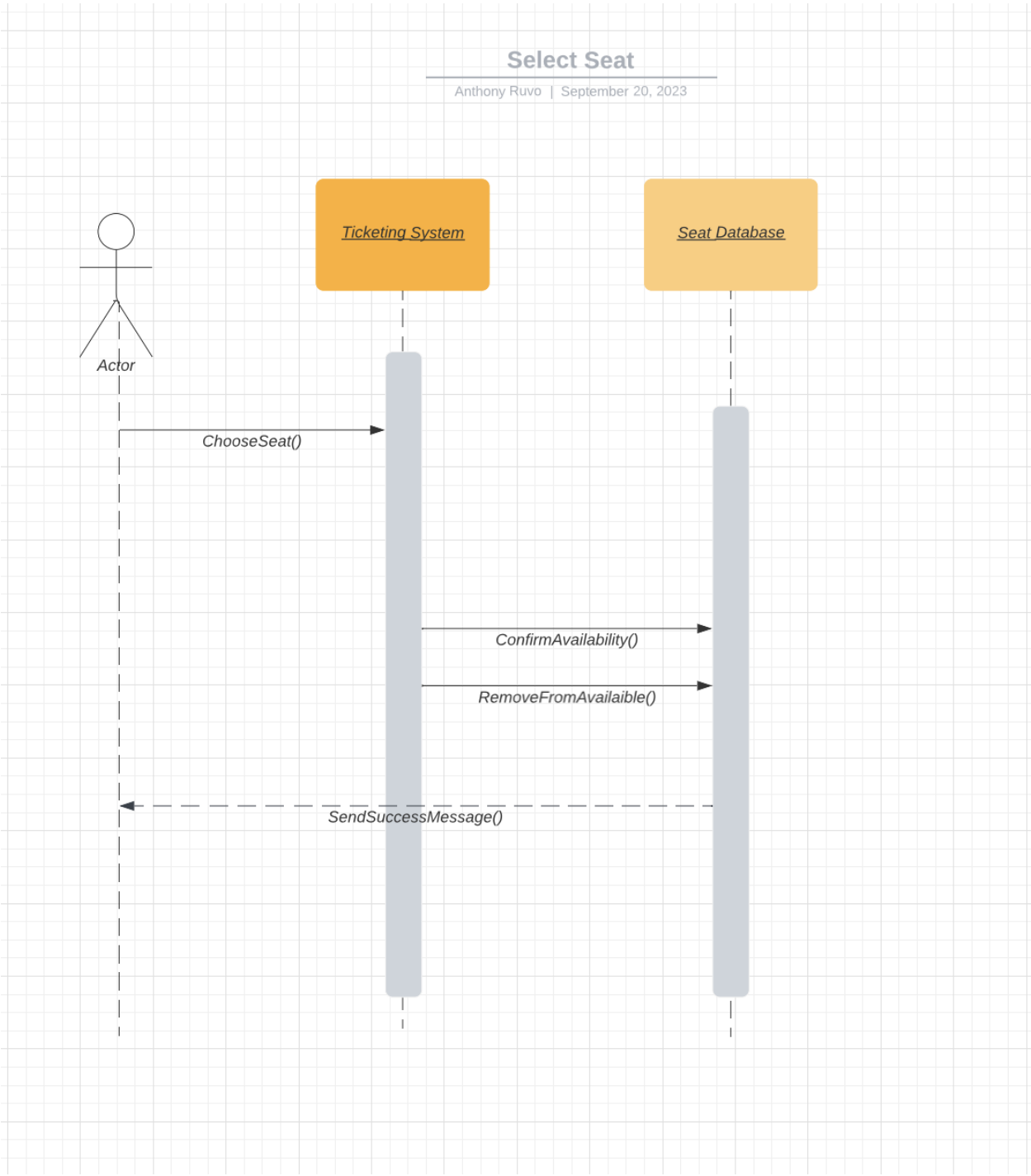
No unauthorized access in this app!
activeAccount: A logged-in account object in the TicketSystem. Used to validate various methods

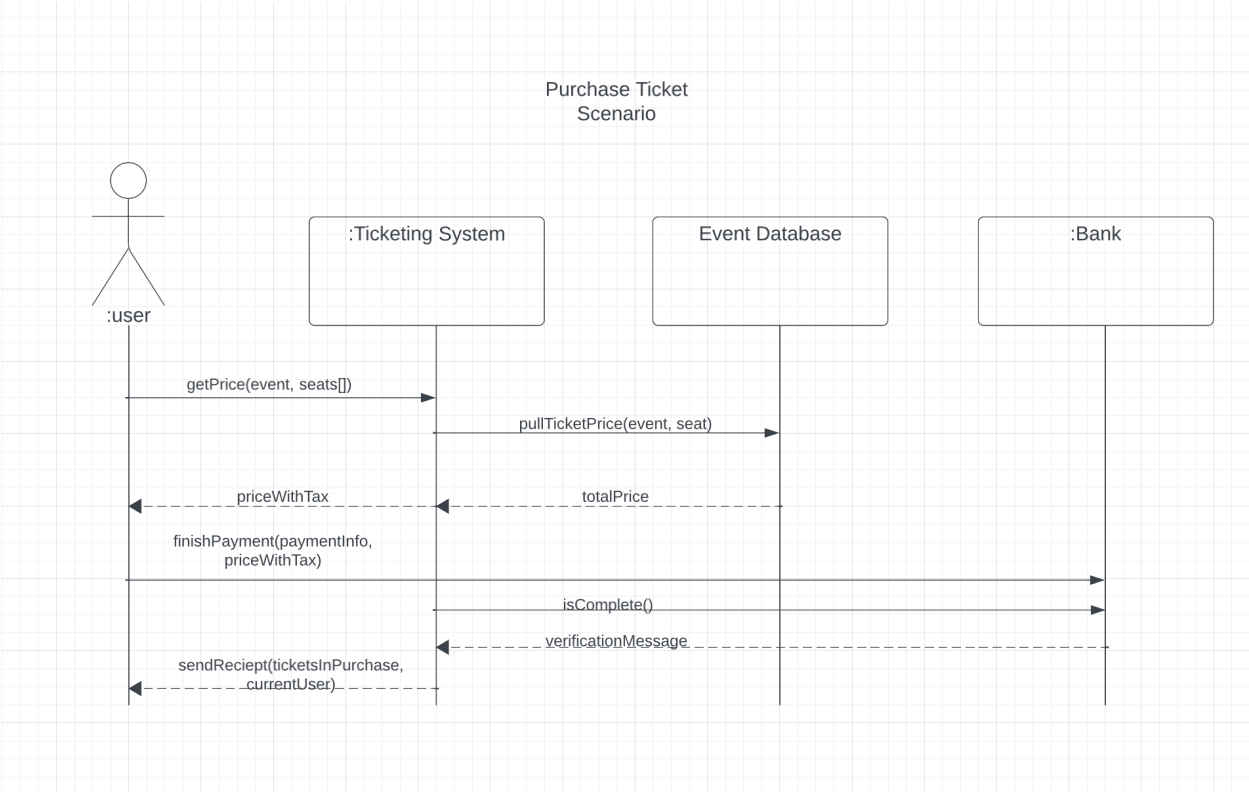
Architectural Factors

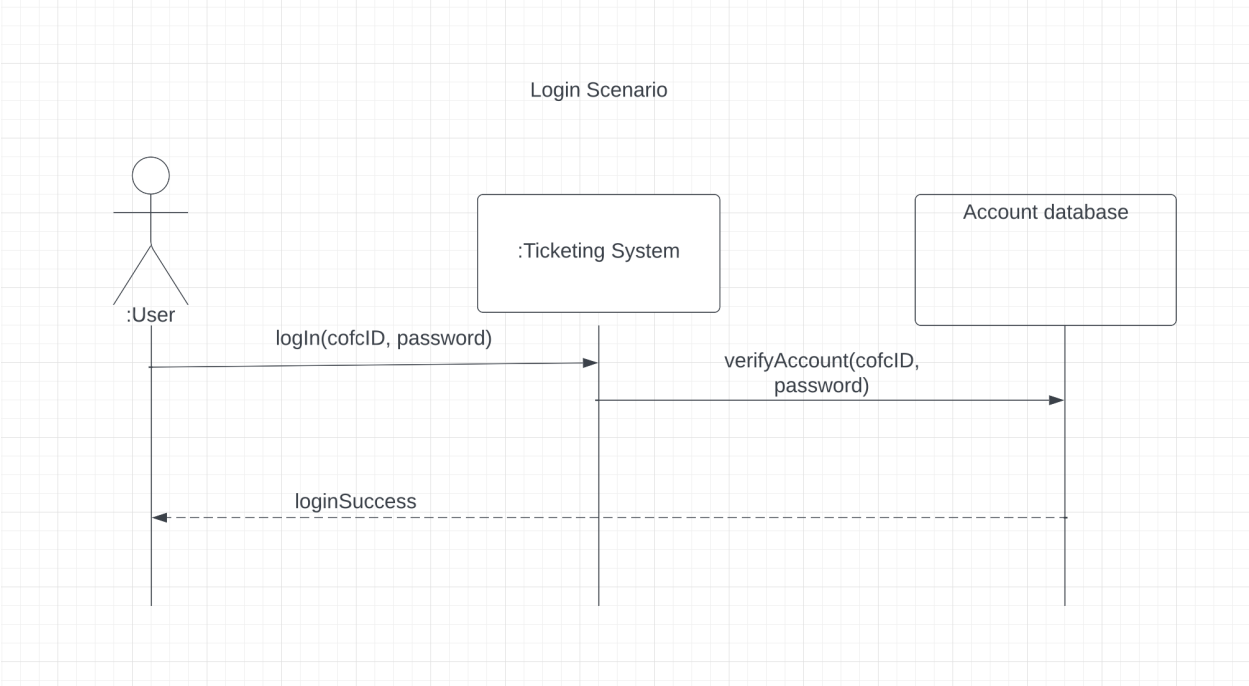

Send Receipt
Anthony Ruvo | September 20, 2023
Ticketing System
Payment Processor
Ledger
Actor
PurchaseTicket()
ProcessPayment()
CreateTicket()
CreateReceipt()
SendReceipt()

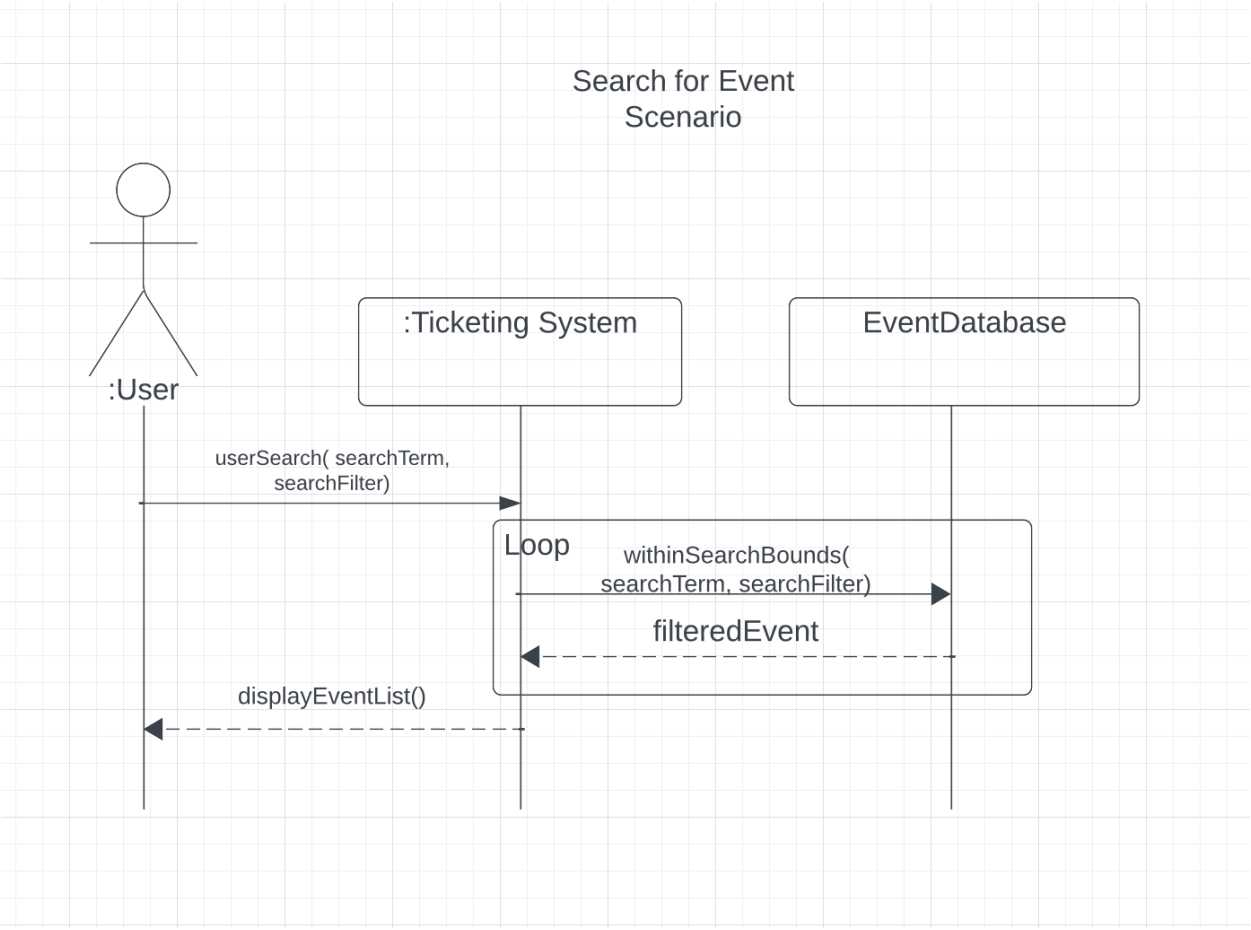

Create Ticket
Anthony Ruvo | September 20, 2023
Ticketing System
Payment Processor
Ledger
Actor
PurchaseTicket()
ConfitmPurchase()
CreateTicket()
ShareTicket()

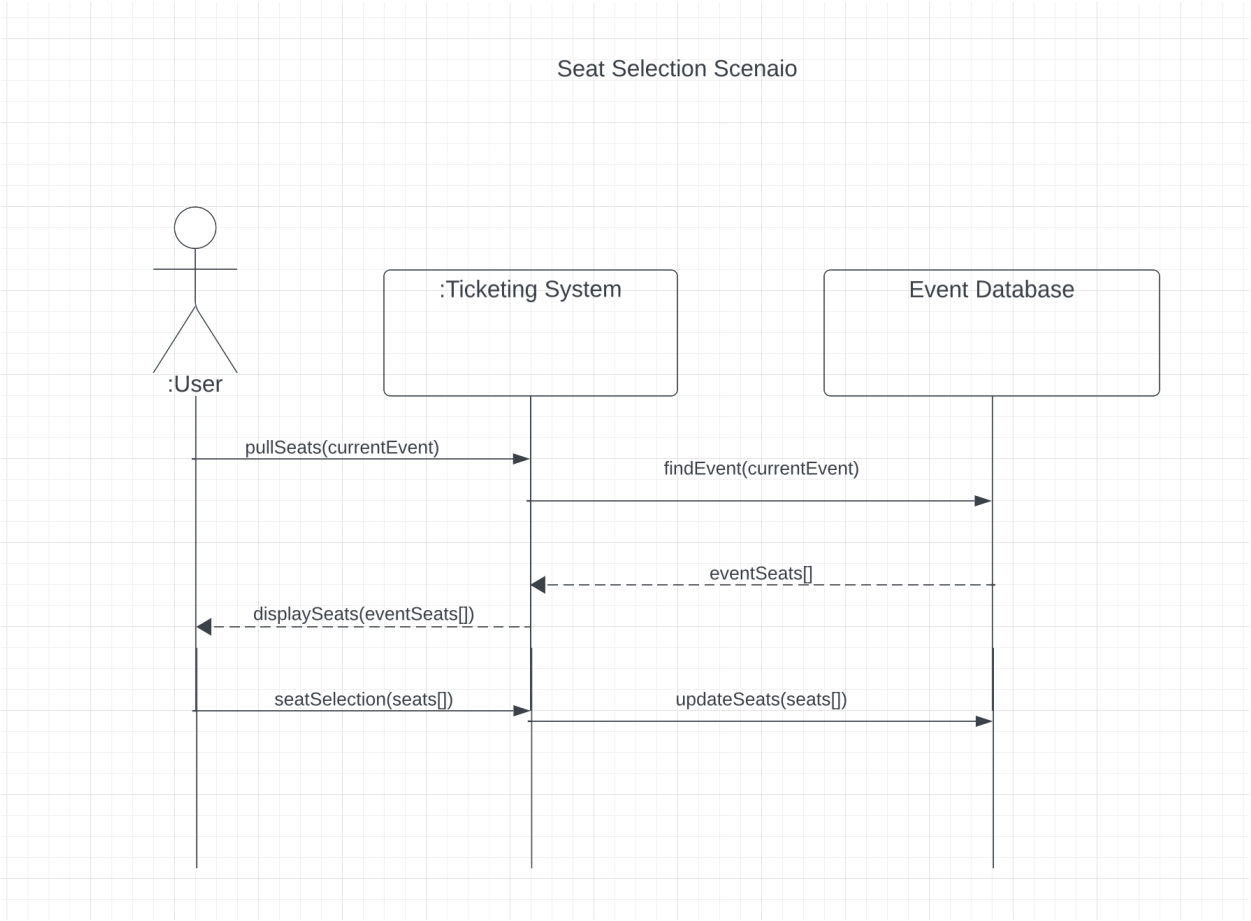

Create Account
Anthony Ruvo | September 20, 2023
Ticketing System
Account Database
Actor
CreateAccount()
SendAccountInformation()
StoreAccountInformation()
SendSuccessMessage()


Select Seat
Anthony Ruvo | September 20, 2023
Ticketing System
Seat Database
Actor
ChooseSeat()
ConfirmAvailability()
RemoveFromAvailaible()
SendSuccessMessage()


Purchase Ticket
Scenario
:user
:Ticketing System
Event Database
:Bank
getPrice(event, seats[])
pullTicketPrice(event, seat)
priceWithTax
totalPrice
finishPayment(paymentInfo,
priceWithTax)
isComplete()
verificationMessage
sendReciept(ticketsInPurchase,
currentUser)


Login Scenario
:User
:Ticketing System
Account database
logIn(cofcID, password)
verifyAccount(cofcID,
password)
loginSuccess


Search for Event
Scenario
:User
:Ticketing System
EventDatabase
userSearch(searchTerm,
searchFilter)
Loop
withinSearchBounds(
searchTerm, searchFilter)
filteredEvent
displayEventList()



Logical View

This includes expected contents along with all updated and complete class diagrams with consistent names, variables, and method names

Discussion and Motivation

These documents attempt to show the code in order of importance.

Classes

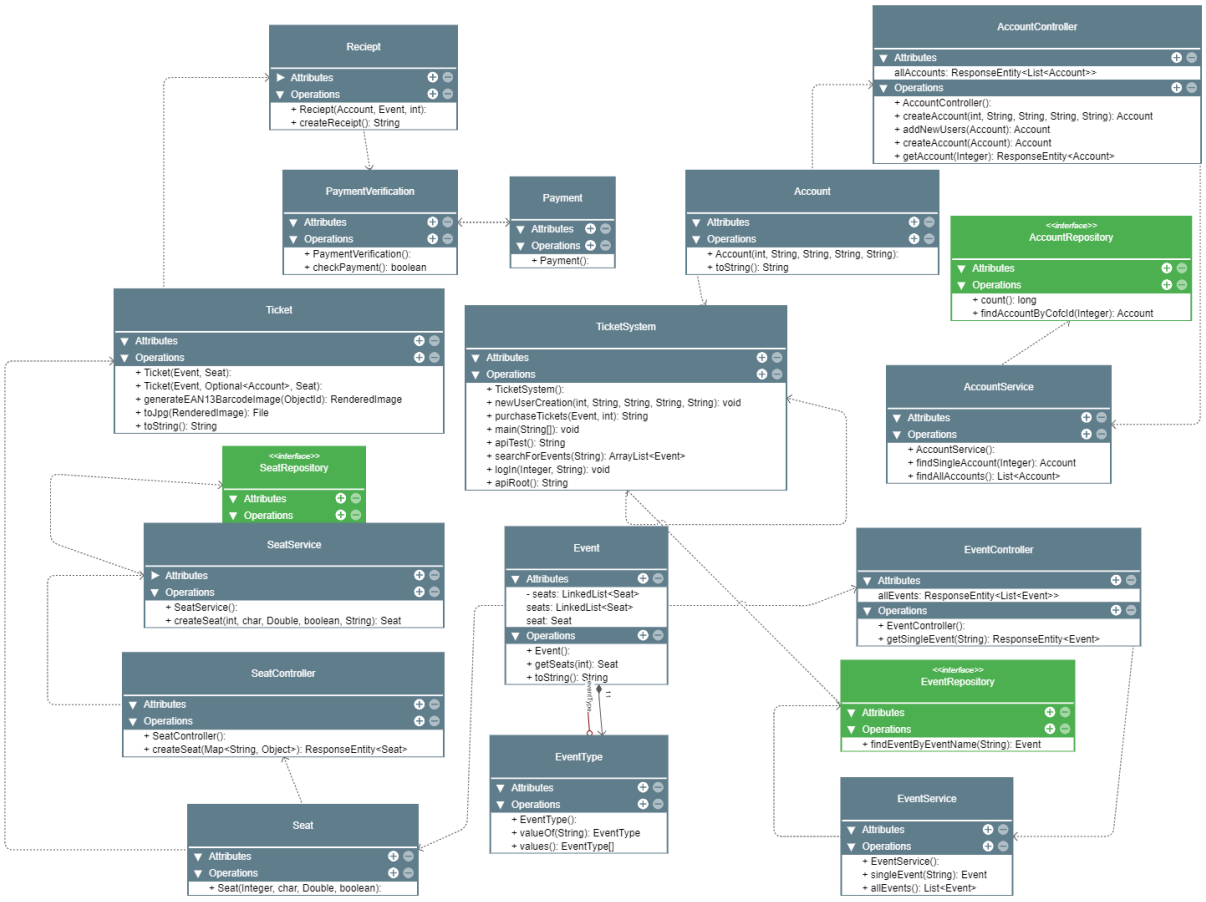
Account: Contains user information such as username, password, CofCID, email, and tickets

Event: Contains information about event objects such as event type, name, location, and date, and inventory

Ticket: Contains information about a specific instance of event attendance such as the event it is for and the owner

Receipt: Object containing information about a user's ticket purchase

Seat: Will be utilized in future iterations when users can select specific seats to purchase



Development View

Discussion and Motivation

The project is divided into a front end and a back end. The front end is written using JavaScript and React. The back end is written using Java with Spring Boot for assistance.

Use Cases

Account Registration

Name	Content

Operation	`newUserCreation(int cofcId, String firstName, String lastName, String password, String email )`
Cross References	**• cofcId: The cofcId provided by the user.** **• password: Extra secure password provided by the user.** **• logIn()**
Preconditions	**User is on the application and no account with the same username exists**
Postconditions	**If the creation is successful, the user is added to the list of users and logged in. If it is not successful, an error message is displayed**

Log In

Name	Content
Operation	`logIn(Integer cofcID, String password)`
Cross References	**• cofcId: The cofcId provided by the user.** **• password: Extra secure password provided by the user.** **• activeAccount: the active account in the TicketSystem**
Preconditions	**User is on the application and has an existing account**

Postconditions	**The user is logged in and the user's account becomes the activeAccount**

System Features

Purchase Ticket

Name	Content
Operation	`purchaseTickets(Event event, int num)`
Cross References	**• event: The event the user wants to purchase tickets for.** **• paymentValidator(): Mock process payment with a low chance of failure**
Preconditions	**The user is logged in and the requested number of tickets is available**
Postconditions	**If the purchase is successful, the tickets are added to the user's LinkedList of tickets and the number of purchased tickets is deducted from the events inventory**

Search For Events

Name	Content
Operation	`searchForEvents(String string)`
Cross References	• None
Preconditions	User is on the application
Postconditions	Any events matching the user's query are displayed on the screen

---

Generate Receipt

Name	Content
Operation	`createReceipt()`
Cross References	`Reciept(Account account, Event event, int num)`
Preconditions	User is the activeAccount on the TicketSystem and has made a successful purchase

Postconditions	**A receipt describing the user's purchase is displayed on the screen. Information is provided to the `createReceipt()` by the Receipt object.**

Operations Contracts

Operation	**newUserCreation(int cofcId, String firstName, String lastName, String password, String email)**
Cross References	**logIn()**
Precondition	**User does not have an account**
Postcondition	**A new account object is created and is added to the database. The user is then logged in and made the activeAccount**

Operation	**logIn(Integer cofcID, String password)**
Cross References	**None**
Precondition	**User has an account**
Postcondition	**If the username and password match, the user is switched to logged in and made the activeAccount**

Operation	purchaseTickets(Event event, int num)
Cross References	• isLoggedIn() • setInventory() • createReceipt()
Precondition	User is logged in and is attempting to make a purchase
Postcondition	If the purchase is successful, the tickets are added to the user's LinkedList of Tickets and the number purchased are deducted from inventory. A receipt is sent to the user

Operation	searchForEvents(String string)
Cross References	None
Precondition	User is logged in and is searching for events
Postcondition	The results of the search are displayed on the screen

Operation	createReceipt()
Cross References	purchaseTickets()

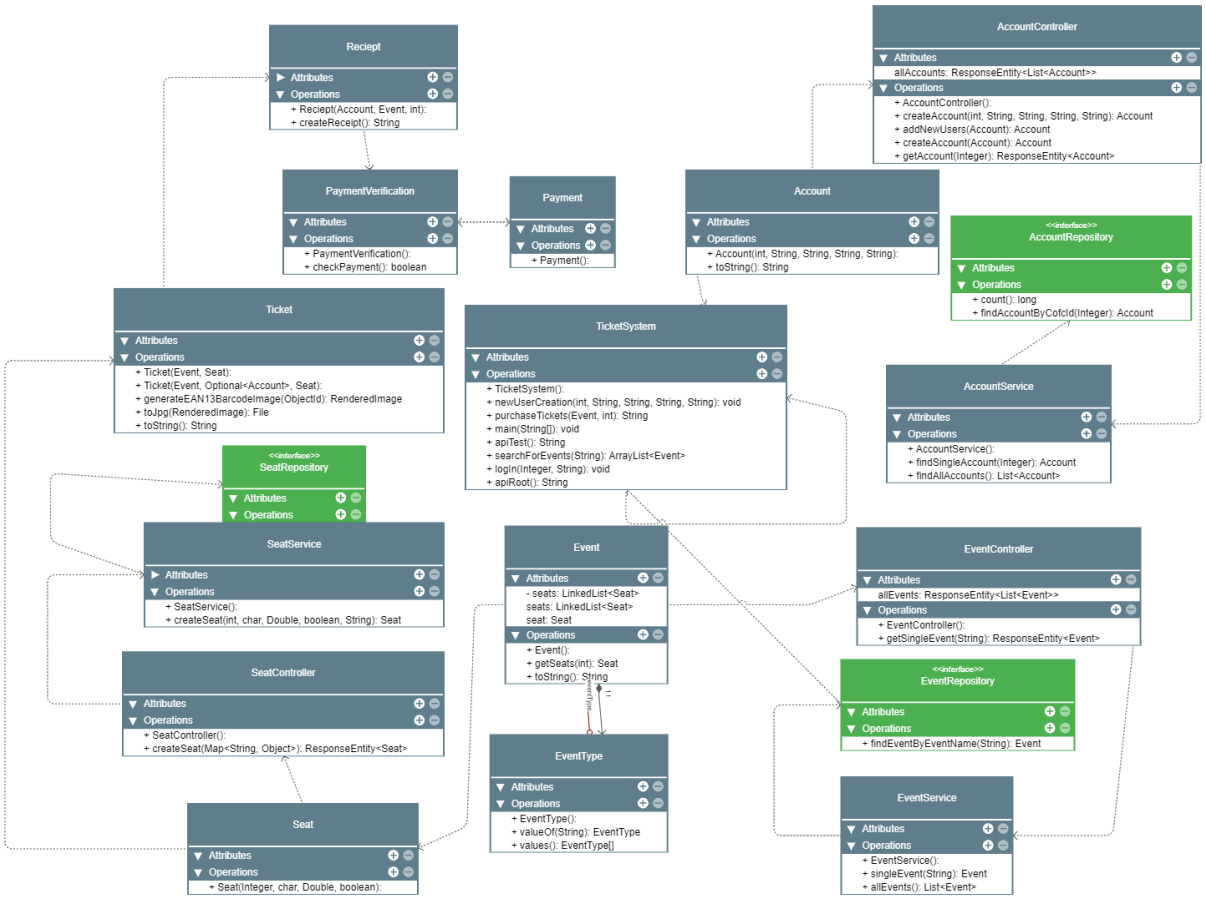
Precondition	**User has made a successful payment**
Postcondition	**A receipt is displayed to the user detailing the purchase**

Deployment View

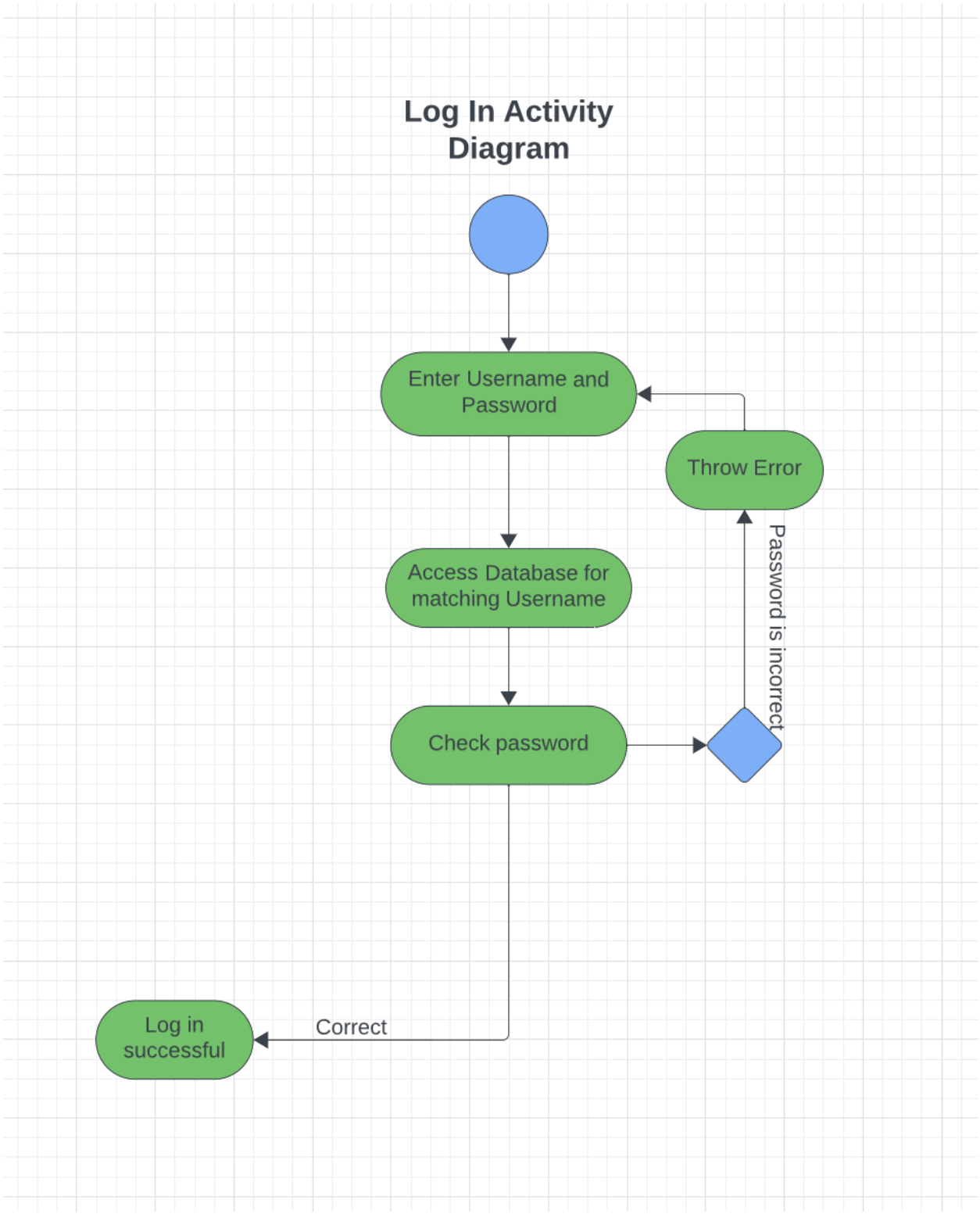
Discussion and Motivation

Diagram showing the structure of the code and the process of logging in. logIn() is arguably the most important function in the code. It ensures that only validated users are allowed to purchase tickets.

Order of Class Implementation



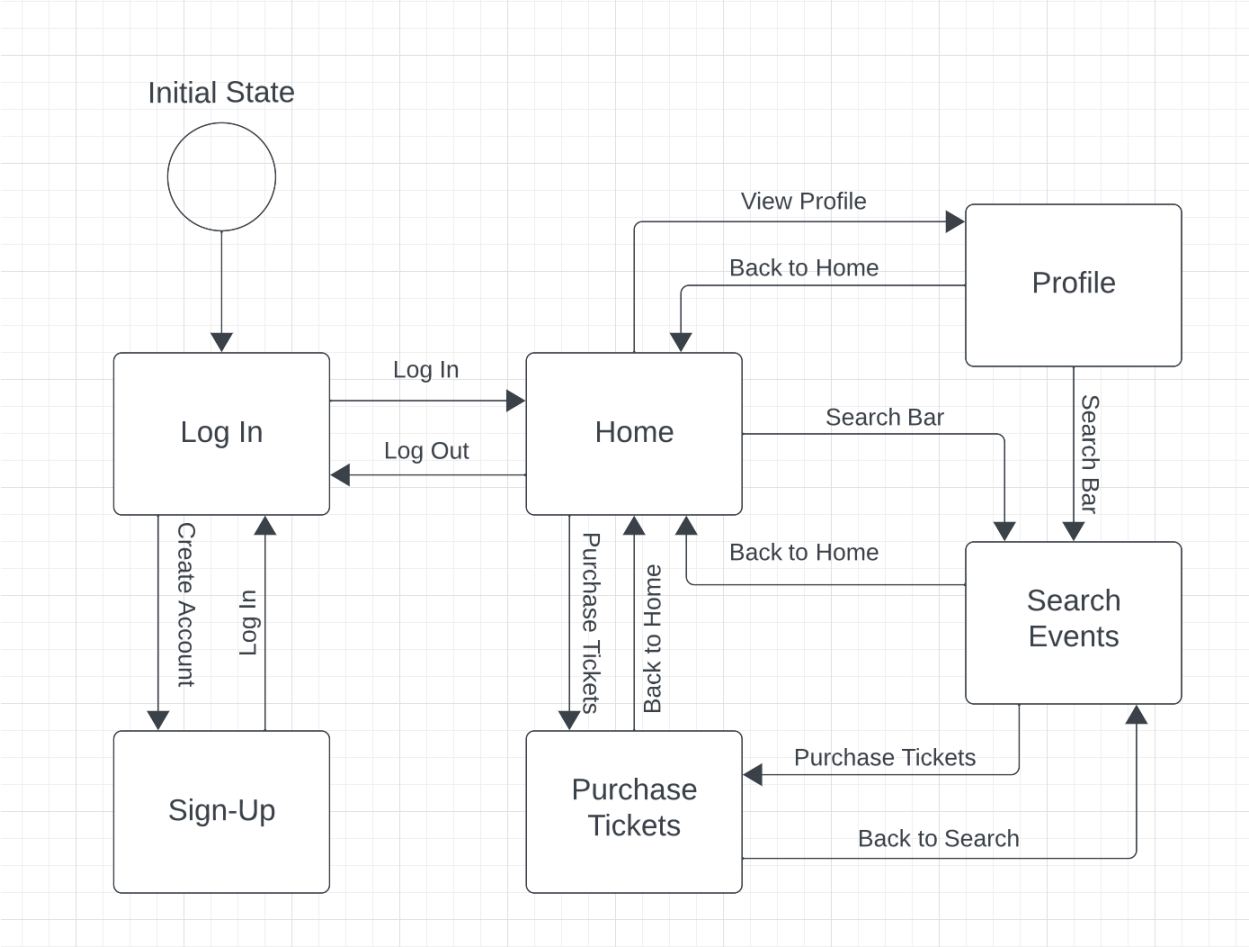
Log in Activity Diagram


Log In Activity Diagram
Enter Username and Password
Throw Error
Access Database for matching Username
Password is incorrect
Check password
Log in successful
Correct

Process View

Motivation: To show developers how different processes are broken up.

Class Diagrams: The pages of our system include Log In, Sign Up, Home, Profile, Search Events, and Purchase Tickets, which interact through a backend written in Java.



1. Starting at the login screen, users can either log in or register, upon registering they are brought back to the login screen, now with ownership of an account.
2. Next they are taken to the homepage, with a search bar, profile button, and shortcuts to buy tickets directly.

3. When a user clicks on their profile they are brought to a screen showing their profile.
4. When a user clicks to purchase a ticket for an event, they are brought to a checkout screen which allows them to input credit card information.
5. After clicking purchase, users can press the back button to return to the previous page at any time.
6. From the home screen users can also log out.

Security View

Discussion and Motivation

Prevent unauthorized access and keep user data secure from malicious actors

Abuse Cases

Description

A malicious actor attempts to use a stolen or compromised credit card to make a purchase

Countermeasures

1. Payment Processor Validates all Information: The payment processor will validate all information and reject the payment if any information does not match.
2. Future Implementation of Secure APIs: APIs can provide more security and shift the responsibility from the client to the publisher of the API.

Description

A malicious user attempts to log into another user's account

Countermeasures

1. Strong Password Requirements: The application has strict password requirements designed to prevent brute-force attacks.
2. Future Implementation of Google OAuth: Google OAuth provides a significantly more secure environment. It would also make integration into Google Calendar and Reminders smoother.

Use Case View

Discussion and Motivation The most architecturally significant use cases are creating a new account and purchasing a ticket.

Use Case Section	Comment
Use Case Name	Creating a new account
Scope	CofC Secure Ticketing System
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: Wants to create an account and potentially purchase tickets CofC: Wants to track attendance to events
Preconditions	The application is running and the CofCID is not associated with any previously created accounts
Success Guarantee	A new account is created, added to the database, and made the active account

Main Success Scenario	1. User attempts to create an account 2. User enters valid credentials and a super secure password 3. A new account is created and the user is logged in and made the activeAccount
Extensions	2a. Invalid credentials • User is shown an error and the account is not created 3a. Code fails to create a new account (would never happen) • An error will be reported to the user and the user will be directed to try creating another account
Special Requirements	Java
Technology and Data Variations List	2a. Verification that the password meets the requirements 3a. Account storage
Frequency of Occurrence	Frequent
Miscellaneous	Is Google OAuth a better option?

Use Case Section	**Comment**
Use Case Name	Purchase Tickets
Scope	CofC Secure Ticketing System

Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: Wants to see the Cougs Win! CofC: Wants to sell tickets to events and make money
Preconditions	The user is logged in, and tickets are available
Success Guarantee	Tickets are available, payment is successful, ticket(s) added to user's LinkedList, receipt displayed to user, inventory is updated
Main Success Scenario	1. User attempts to purchase ticket(s) 2. User attempts purchase 3. Payment is successful, tickets are transferred to user, inventory is updated
Extensions	2a.Tickets not available • User is shown an error and the purchase is not made 3a. Payment fails • User is prompted to try again
Special Requirements	Java
Technology and Data Variations List	2a. A secure payment processor
Frequency of Occurrence	Frequent
Miscellaneous	3rd party payment processor (e.g. Stripe) might be a better option

Architectural Factors

- Must be built using the Java (or equivalent) object oriented language.
- Must include unit tests for each applicable methods.

Must Have's (MVP):

- Secure login
- Unique username
- Password must be 15 characters and include lower case letter, upper case letter, number, and special character

Functions of system

- Allow users to search for events
- Allow users to purchase tickets for events, if they are available
- Mock process payment
- Display message of payment success
- Update ticket inventory

Stretch goals:

- Add purchased tickets event date/time and info to the user's Google Calendar
- Allow users to choose their seats (if seated event)
- Update seat map