Java Lagged - Ticketing System SAD

Evan Luecken - - Caleb Miranda - - Sierra Puwalowski

CSCI 360-01 Fall 2023

Architectural Representation

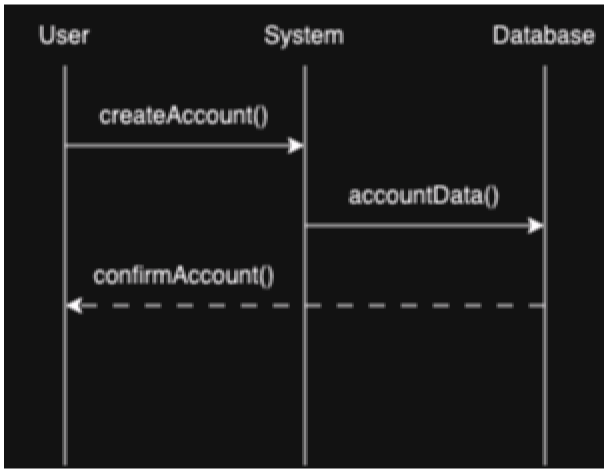
This SAD summarizes the architecture of the website from multiple views. These are:

- Architectural Factors: Contains the SSDs used for this project.
- Architectural Decisions: An overview of the technical and architectural decisions made throughout development.
- Logical View: Contains the class diagrams for this project.
- Development View: An overview of the operation contracts used within the code.
- Deployment View: Contains the Activity Diagrams for each use case.
- Process View: Contains the UI State Machine model.
- Security View: An overview of the Abuse Use Cases and diagrams to support those use cases.
- Use Case View: An overview of the Use Cases within this project.

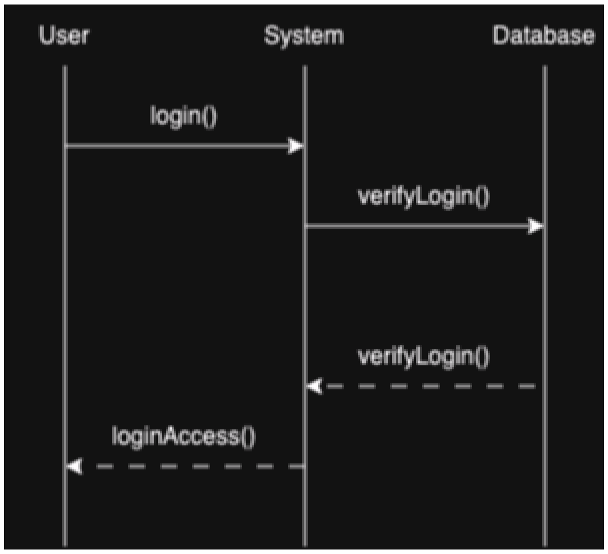
Architectural Factors

Our project is made up of two main parts. The first part is the back-end powered by spring boot in the java language. This back-end communicates with our firebase server that holds all user information. The second part is the front-end, which is written in java and run through Gradle to create the UI.

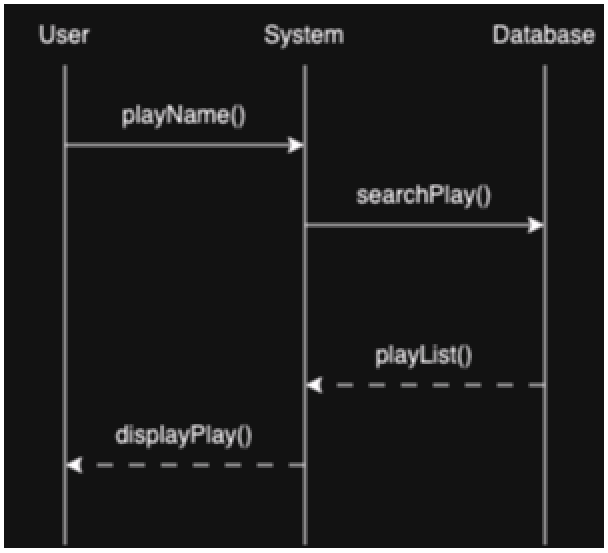
Creating an Account SSD



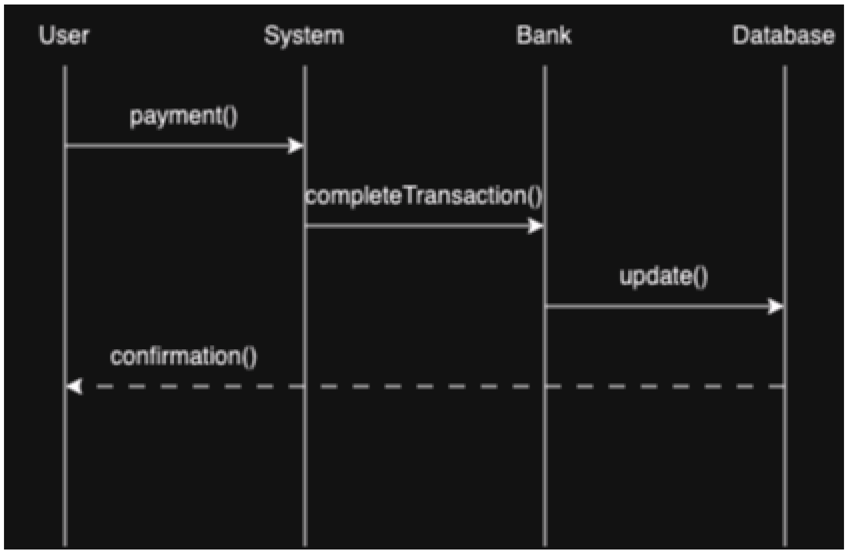
Login SSD



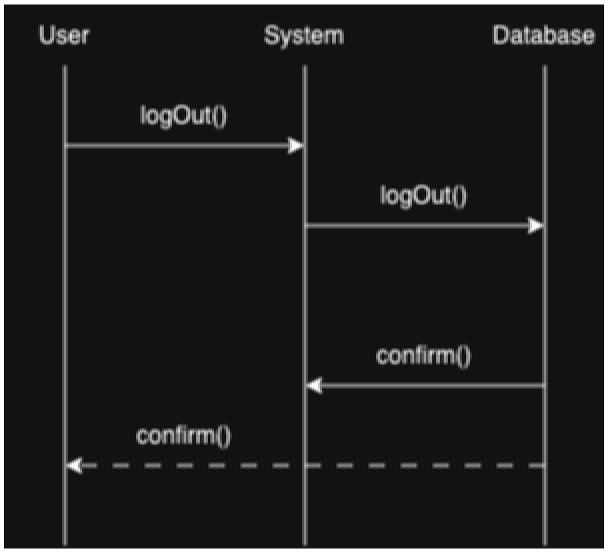
Search for a Play SSD



Purchase a Ticket SSD



Log Out SSD



Architectural Decisions

The goal of this project is to create a java app secure ticketing system for the College of Charleston that can interact with a database and allow users to search for events and purchase tickets for those events if available. Additionally, the project will contain a mock process payment and a display message of payment success as well as updating the ticket inventory after a transaction. The project will allow users to securely login to their own profiles using a unique username and a password that must be 15 characters and include a lowercase letter, upper case letter, number, and special character. If it's possible to implement, the project will also add purchased tickets, event date/time and info to the user's Google Calendar and allow users to choose their seats in a seated event.

<u>Questions</u>
1.How many events would this program be expected to track?
2.Should user profiles contain any other information, such as age?
3.Can we assume that only CofC students and/or faculty will be using this application?
4.We could implement a sorting filter to user queries to help them find events faster (i.e. 5.sort by time/location of event)
6.Should we allow one user to purchase more than one ticket?

<u>Introduction</u>
We will specify all additional requirements for our ticketing system in this document

<u>Errors</u>
Categorize all errors and logs them in a database to be review for consistent problems

<u>Security</u>

You will need 2 factor authentication from a 3rd party app authenticator to sign up for an account, or you will be charged as a guest with no information saved

Usability
All prices will be listed pre tax clearly before purchase, and all fees and other small rules will be listed in bold with different languages, like a no refund policy

Performance
Option for guest check out

Providers
Will need to be able to interact with multiple different providers, like PayPal and other 3rd party online payments.

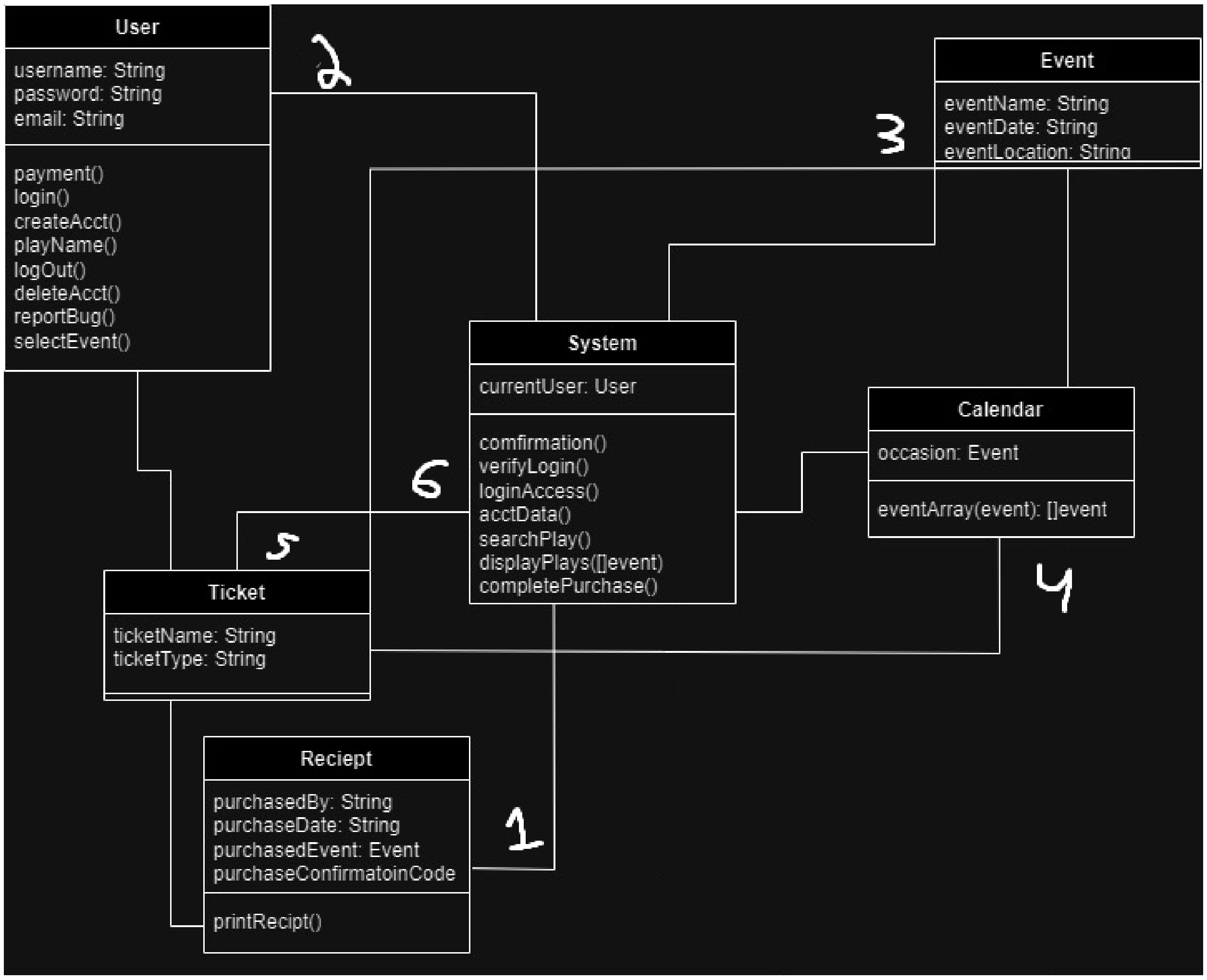
OTHER
Will need to take data from the college about location and the seating arrangements to accurately display the positions of each seat.

Will need to be able to use the colleges' student verification system to verify student pricing.

Logical View

Our project is made up of two main parts. The first part is the back-end powered by spring boot in the java language. This back-end communicates with our firebase server that holds all user information. The second part is the front-end, which is written in java and run through Gradle to create the UI.

Class Diagram



Development View

These are the Operation Contracts that were used while developing this system.

<u>Creating an Account</u>

Operation	createAccount()
Cross-Reference	Use Case: Creating an Account
Precondition	An account can be created and stored in database
Postcondition	The transaction is completed and recorded

Login

Operation	logIn()
Cross-Reference	Use Case: Login
Precondition	User has a valid account and is not already logged in
Postcondition	The user has entered login information and is awaiting verification

Operation	verifyLogIn()
Cross-Reference	Use Case: Login
Precondition	Login information has been input
Postcondition	Login information is either confirmed or denied by the database

Operation	logInAccess()
Cross-Reference	Use Case: Login
Precondition	Login information has been checked by the database
Postcondition	Login access is either granted or denied to the user

Search for a Play

Operation	searchPlay()
Cross-Reference	Use Case: Search for a Play
Precondition	The user has navigated to the search bar and entered some criteria to search by
Postcondition	Plays that match the criteria are displayed to the user

Purchase a Ticket

Operation	payment()
Cross-Reference	Use Case: Purchase a Ticket
Precondition	The user has selected some event that they wish to purchase a

	ticket for
Postcondition	Payment information is entered and ready to be processed

Operation	completeTransaction()
Cross-Reference	Use Case: Purchase a Ticket
Precondition	The user has confirmed their purchase
Postcondition	The transaction is completed and recorded

Operation	update()
Cross-Reference	Use Case: Purchase a Ticket
Precondition	The transaction has been completed and recorded
Postcondition	The ticket inventory has been updated with the new number of tickets

Operation	confirmation()
Cross-Reference	Use Case: Purchase a Ticket
Precondition	Payment is pending and ready to be processed
Postcondition	Payment is either confirmed or denied by the user

Operation	printReceipt()
Cross-Reference	Use Case: Purchase a Ticket
Precondition	purchase has been completed
Postcondition	A message is printed out to the console, giving the purchase date, buyer name, total cost, event name, and the names of the purchased tickets.

<u>Log Out</u>

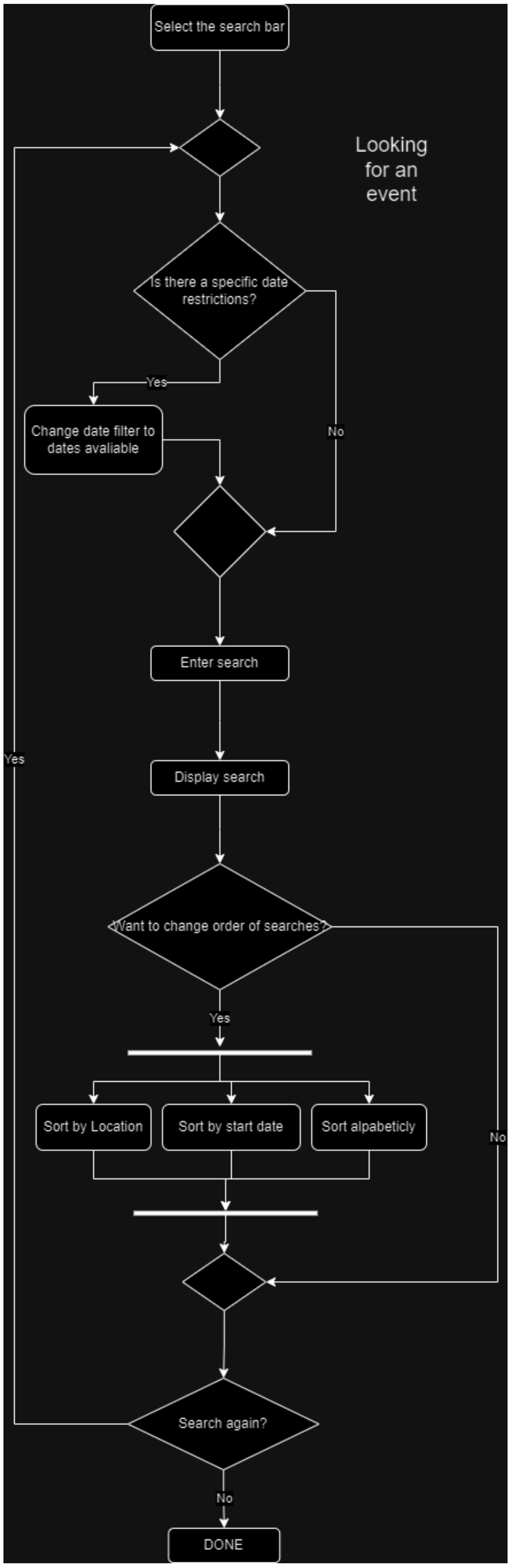
Operation	logOut()
Cross-Reference	Use Case: Log Out

Precondition	User is logged in
Postcondition	User confirms information

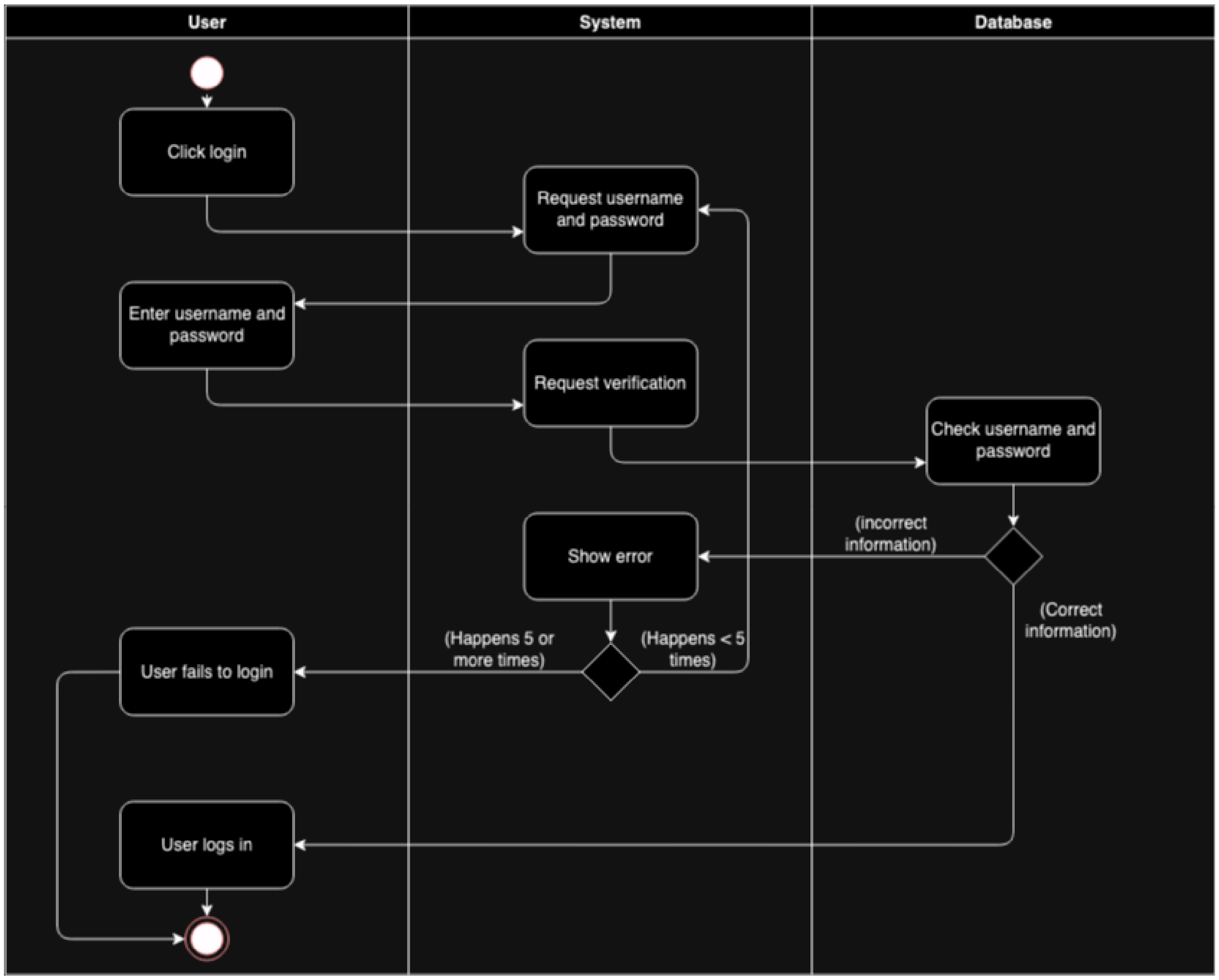
Operation	confirm()
Cross-Reference	Use Case: Log Out
Precondition	User is logged in Logging out is underway
Postcondition	User is logged out

Deployment View

Search for a Play Activity Diagram

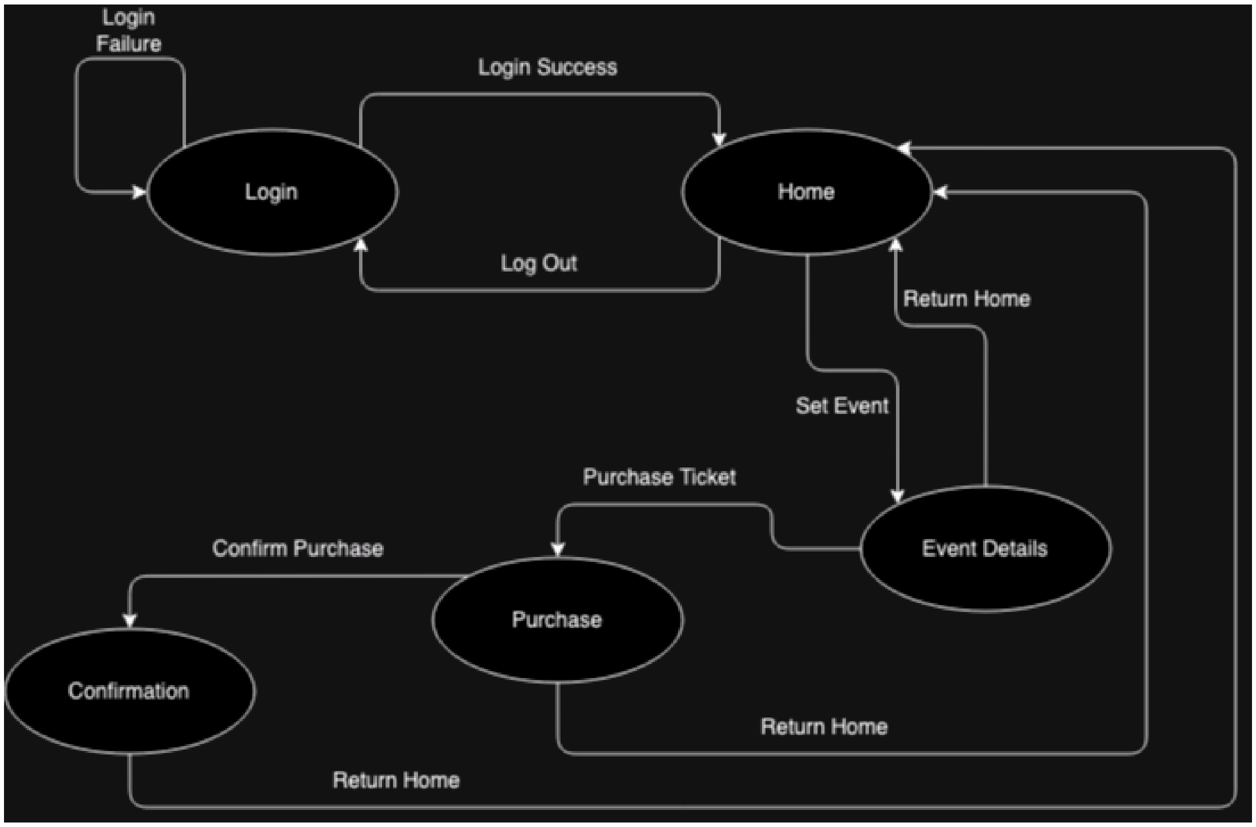

Select the search bar
Looking for an event
Is there a specific date restrictions?
Yes
No
Change date filter to dates avaliable
Enter search
Display search
Want to change order of searches?
Yes
No
Sort by Location
Sort by start date
Sort alpabeticly
Search again?
Yes
No
DONE

Login Activity Diagram



Process View

Ui State Machine Model

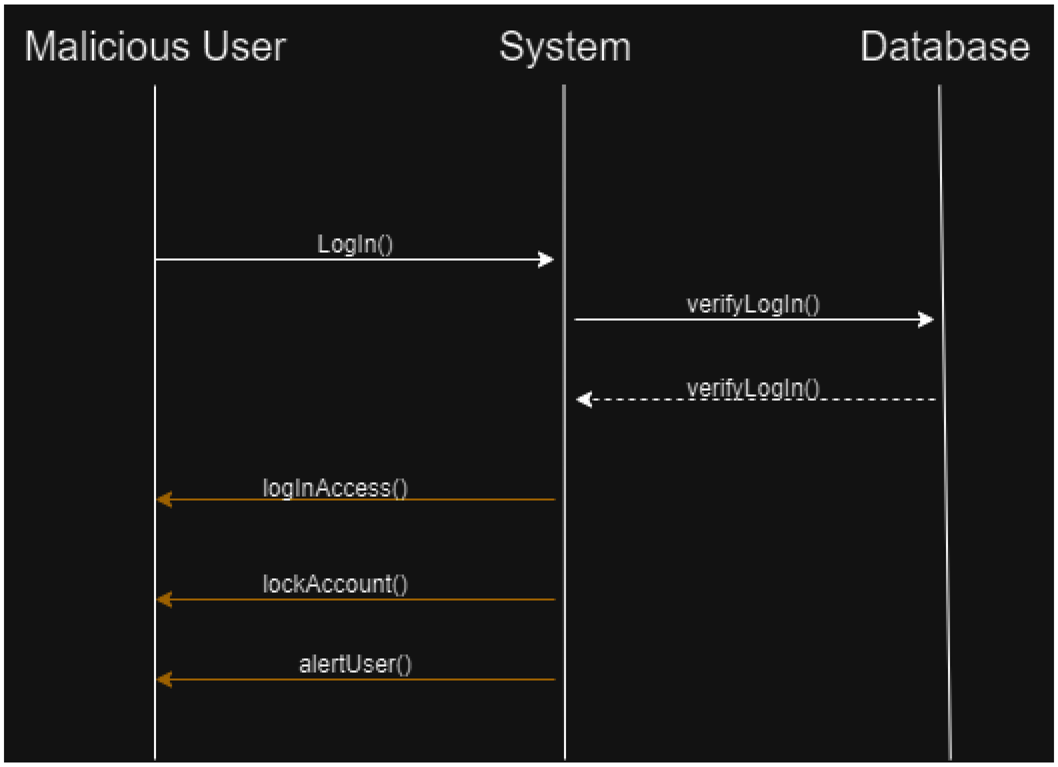


Security View

<u>Brute Force Attack</u>

In this scenario, we’re creating a way to prevent a malicious actor from using brute force algorithms or online password dictionaries to forcefully guess a user’s password. This is done by adding a verification system to the login operation, as well as setting a limit to how many times logging in may be attempted.

SSD



Operation Contracts

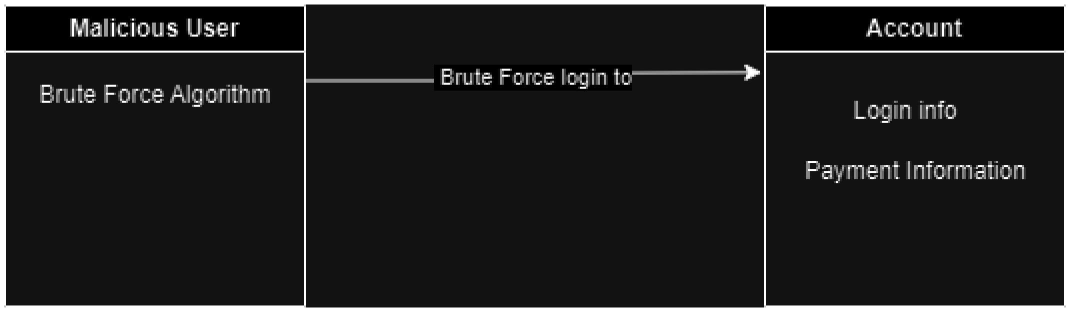
Operation	loginAccess()
Cross-Reference	Brute Force/Dictionary Attack
Precondition	User login info has been verified as either valid or invalid
Postcondition	User is granted or denied access to the account, if denied, the number of failed login attempts is incremented

Operation	lockAccount()
Cross-Reference	Brute Force/Dictionary Attack
Precondition	A fifth login attempt has failed
Postcondition	All further attempts are temporarily blocked

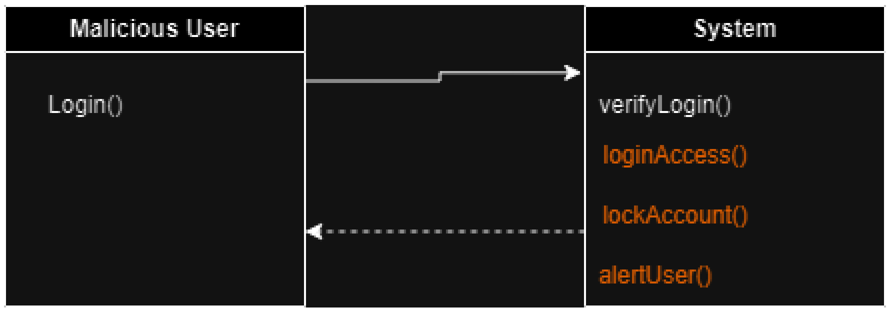
Operation	alertUser()
Cross-Reference	Brute Force/Dictionary Attack
Precondition	User account has been locked

Postcondition	An email is sent to the user informing them that their account has been locked.

Domain Model



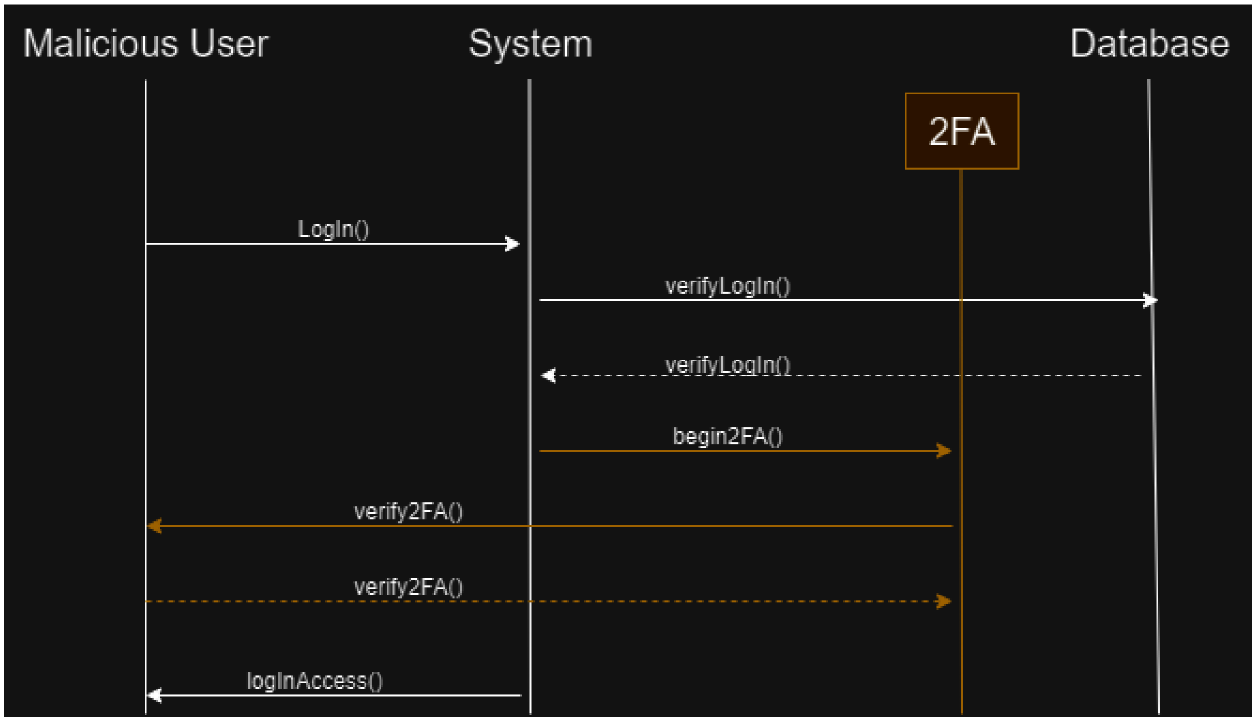
UML Diagram



<u>Stolen Login Information</u>

In this scenario, there is a malicious actor that has obtained a user's password and is attempting to gain access to their account to steal sensitive information. For this, we've implemented a 2-factor authentication verification.

SSD

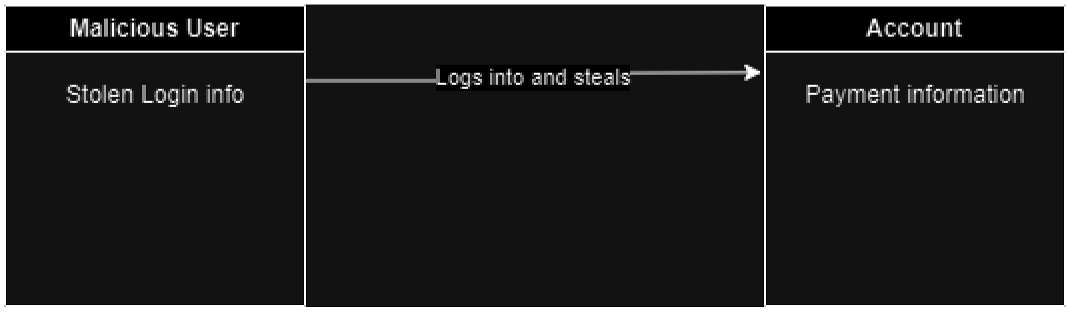


Operation Contracts

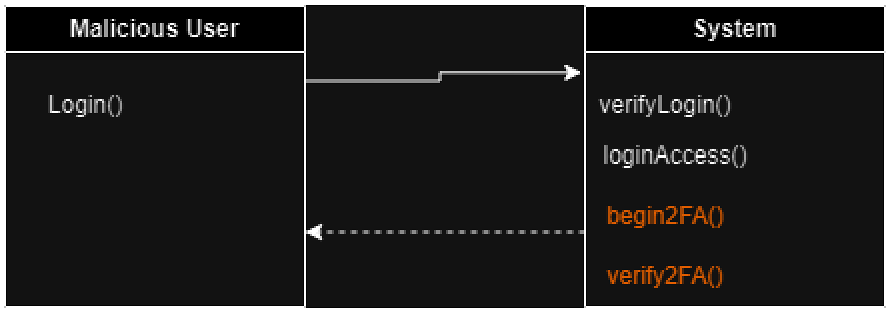
Operation	begin2FA()
Cross-Reference	Stolen Login Information
Precondition	Login info has been successfully verified
Postcondition	A 2FA code is generated and emailed to the user

Operation	verify2FA()
Cross-Reference	Stolen Login Information
Precondition	2FA code has been received by the user and input into the system
Postcondition	2FA is verified as valid or invalid

Domain Model



UML Diagram



Use Case View

The most architecturally significant use cases are Creating an Account, which is used as a basis for other use cases such as Login and Log Out, and Purchase a Ticket, which is the main purpose of the website.

Creating an Account

Use Case Section	Comment
Use Case Name	Creating an Account
Scope	Java-lagged team designed event ticketing software
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: wants to create an account to use this software College of Charleston: wants to track the number of people using this software

Preconditions	The ticketing software is running and executing normally There is space in the database to store the login information of another account
Success Guarantee	A new account is successfully created
Main Success Scenario	1. User navigates to the “Create Account” button 2. User enters a valid username and password that has at least one Uppercase letter, one lowercase letter, a number, special character, and a minimum of 10 characters 3. The user will be prompted to verify their identity through a 3rd-party authenticator 4. The system creates a new account and automatically logs the user into it
Extensions	2a. The user enters an invalid username or password - The user will be informed that their username/password is invalid and prompted to enter a new one 3a. Two-factor authentication fails to validate the identity of the user - The system will display an error to the user - The user will be prompted to reattempt two-factor authentication 4a. The system fails to create a new account - An error will be reported to the user and to the System Administrators - The user will be directed to try creating another account
Special Requirements	Java Two-Factor Authentication
Technology and Data Variations List	2a. A password verifier that will ensure all passwords meet the minimum password requirements 3a. A 3rd-party authenticator app that can be used for two-factor authentication 4a. Database that can store account information for users
Frequency of Occurrence	Almost continuously
Miscellaneous	What application should be used to facilitate two-factor authentication?

Login

Use Case Section	Comment
Use Case Name	Login

Scope	Java-lagged team designed event ticketing software
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: wants to log into their account
Preconditions	The ticketing software is running and executing normally The user already possesses a valid account
Success Guarantee	The User enters their login information accurately The system successfully validates the login information and grants the user access to their account
Main Success Scenario	1. User navigates to the login button 2. On the login screen, the user enters their username and password 3. The system checks the username and password and recognizes them as valid 4. The user is granted access to their account
Extensions	2a. The user attempts to login with incorrect login information - An error is reported to the user - The user is allowed to try and input their login information up to 5 times - If 5 attempts are exceeded, the user will be directed to reset their login information 3a. The system fails to recognize the username and password as valid (even though they are) - An error is reported to the user - The user will be prompted to re-enter their login information - If the error persists, the user will be directed to reset their login information or submit a bug report to the system 4a. The user is not granted access to their account - The system will log the error and direct the user to try and login again - Repeated failed attempts will signal the System to contact the Administrators about the error
Special Requirements	Java
Technology and Data Variations List	2a. A database that can be used to store and verify login information
Frequency of Occurrence	Almost continuously
Miscellaneous	Will login need to use the same 2-factor authentication as creating an account?

Search for a Play

Use Case Section	Comment
Use Case Name	Search for a Play
Scope	Java-lagged team designed event ticketing software
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: wants to find a specific play College of Charleston: wants to sell tickets for that specific play
Preconditions	The ticketing software is running and executing normally There exists some play in the System that the user wants to find
Success Guarantee	The user's desired play is successfully found
Main Success Scenario	1. The user navigates to the search bar 2. The user searches for the desired play using some criteria (i.e. name, date, etc.) 3. The system returns the correct play that the user desired
Extensions	3a. The system fails to return the correct play that the user desired - The system will report to the user that 0 results were found for their query - The user will be prompted to attempt a new query or else search for a new play entirely
Special Requirements	Java
Technology and Data Variations List	1a. A search bar where the user can enter in some criteria and have the database return all plays matching that criteria
Frequency of Occurrence	Almost continuously
Miscellaneous	How should the returned plays be sorted and displayed to the user? How many plays should be displayed to the user at once?

Purchase a Ticket

Use Case Section	Comment

Use Case Name	Purchase a Ticket
Scope	Java-lagged team designed event ticketing software
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: wants to purchase a ticket College of Charleston: wants to receive money from the transaction and ensure an accurate inventory of tickets
Preconditions	The ticketing software is running and execution normally The user has found whatever play being searched for and proceeded to checkout There exists some ticket in the inventory that is able to be purchased
Success Guarantee	Transaction is recorded and performed accurately The ticket is provided to the user without issue Receipt is also created and distributed to the reader Ticket inventory is updated accordingly
Main Success Scenario	1. The user enters in their payment information 2. The user confirms the purchase 3. The transaction completes successfully and the ticket is digitally sent to the user 4. The system updates the ticket inventory to account for the purchased ticket
Extensions	2a. The user attempts to purchase a ticket but there are no tickets available - A warning will be displayed to the User that no tickets remain for this event - They will be notified of any similar events on other days that have available tickets 3a. The transaction fails to complete - An error will be displayed to the user and an admin will be alerted to the failure - The ticket that was attempted to be purchased will be held in reserve for 24 hours or until an admin can resolve the error and allow the user to purchase the desired ticket 3b. The ticket is never sent - An error message will be sent to both the User and an admin - The system will allow the user to choose between receiving a full refund or waiting until an admin is able to resolve the issue and send them a ticket 4a. The ticket inventory is not updated/updated incorrectly - The system will attempt to check for discrepancies in ticket inventory and alert an admin - An admin with appropriate authority will enter the database

	manually and correct the ticket amounts - The system will be reviewed to figure out what caused the discrepancy and how to fix it
Special Requirements	Java Compatibility with different payment methods
Technology and Data Variations List	3a. Payment system that can interface with different payment providers and securely complete transactions 4a. Database that can store ticket information
Frequency of Occurrence	Almost continuously
Miscellaneous	Should users be allowed to store payment information on their account so that it can be auto filled in the checkout screen?

Log Out

Use Case Section	Comment
Use Case Name	Log Out
Scope	Java-lagged team designed event ticketing software
Level	User Goal
Primary Actor	User
Stakeholders and Interests	User: wants to log out of their account
Preconditions	The ticketing software is running and executing normally The user already has a valid account The user is already logged into their account
Success Guarantee	The user is successfully logged out
Main Success Scenario	1. The user navigates to the “settings” option 2. The user selects the “log out” option 3. The user confirms their selection and is logged out
Extensions	3a. The user confirms their selection but, is not logged out - The system will display an error to the user - The user will be prompted to attempt logging out again
Special Requirements	Java Settings screen Log out functionality

Technology and Data Variations List	1a. A page containing various miscellaneous settings for the user
Frequency of Occurrence	Almost continuously
Miscellaneous	Will the “Log Out” button only be available within the settings page?