Software Architecture Document

Team 1

Brooke Duvall
Robert Stenback
Edin Aleckovic

Table of Contents

Glossary…………………………………………………………………………………3

Vision……………………………………………………………………………………4

Use Cases……………………………………………………………………………….5

Domain Model………………………………………………………………………….18

SSD's……………………………………………………………………………………19

Class Diagrams…………………………………………………………………………24

Operation Contracts…………………………………………………………………….25

Glossary

Term	Definitions and Information	Format	Validation Rules	Aliases
Event	The actual thing that is happening whether it be athletic games, theater performances, and etc.			
Event ID	Numeric code that is associated with each specific event			
Inventory	A complete list of the amount of tickets available for each event			
Password	String of characters, paired with the username, that allows access into the account of the user			
Password Hash	string of digits calculated with the users passwords with a one way operation and is the actual credential saved in the system			
User ID	unique sequence of digits that is specific to each user. assigned at account creation and does not ever change			
Payment Authorization	Validation by an external payment authorization service that they will make or guarantee the payment to the seller			
Payment Authorization Request	Elements electronically sent to an authorization service. These elements include: Event ID, customer account number, amount, and timestamp.			
Receipt	A documentation (electronically or paper-format) that shows proof of payment and what was purchased.			
Seat Map	Graphical representation of the available seats at each particular event.			
Username	Identification used by the user to access the online ticketing system			
Friction	How much effort and/or decisions a user has to put into making an online purchase			
Service	Name of a class that deals with our databases			

Vision

Our vision is to build a website that users can use to buy various types of event tickets. We plan to allow users the ability to search for events, allow them to purchase tickets to that event, and then display them a confirmation of their ticket purchase. To accomplish this we plan on building a database to store users info as well as event information/ticket inventory. Once we have completed these base requirements, we plan, in the near future, on evolving our system to allow for users to select their particular seats, as well as add the event to their google calendar.

Use Cases

Search For Events

Scope: Secure Ticketing System

Level: User goal

Primary Actor: Customer interested in the purchase of tickets

Stakeholders and Interests:

- Customers
  - Wants to be able to find the specific events that they are interested in attending in the future in order to purchase tickets.

- Event organizers
  - They are invested in the amount of sales that the tickets achieve and so wish for customers to easily find their events and for this process to be accurate, fast and reliable.

- Event contributors
  - Similar to event organizers, event contributors are invested in how well the tickets for the event sell. This means they also wish for the search process to be as fast, accurate and reliable as possible.

- Company
  - The company is invested in having as little friction on the search process as possible in order to further encourage potential customers to be able to find as many events they would be interested in and to make as many ticket purchases as possible.

Preconditions: User is logged in and authenticated.

Success Guarantee: A list of events will be loaded that can be further explored. If the exact event does not exist, similar events in the area and time will be displayed.

Main Success Scenario:

➔ An actor is able to type into a search bar
➔ An actor types into a search bar the event they are looking for.
➔ The system looks up the search result in the event database

- ➔ The system returns to the actor a list of results sorted by relevance
- ➔ The actor can click on any of the results
- ➔ Once a result is clicked, the details of that event is queried to the event database
- ➔ The event details are listed to the user

Extensions
3a. If there is an error retrieving event data, skip that event.

Special Requirements

- Internet so connection between software, databases, user, and etc. may be established
- A device to use the web application on
- Web browser that may access the web application
- We want similar results to be displayed if the exact event is not found and/or does not exist
- Users are able to see results based on dates, venues, and times; not just from the exact name of the actual event.
- Technology and Data Variations List
- 2a. Event identifiers include name, unique ID, date, and venue.

Frequency of Occurrence: Continuous or near continuous.

Open issues:
Spelling mistakes from the actor
Duplicate event names

Purchasing a Ticket

Scope: Secure Ticketing System

Level: User Goal

Primary Actor: Customer interested in purchasing tickets

Stakeholders and Interests:

- Customer
    - Wants to purchase a ticket in the most convenient way as possible while entering as little information as needed. The user also wants the purchase to be as secure as possible as it is their personal information at stake.

- Event organizers
    - Organizers care about how many tickets they are able to sell on the site therefore they want each customer to have as little friction as possible because any amount of added friction will decrease the amount of sales they will get.

- Event contributors
    - Contributors often earn a percentage of the revenue earned from the sales of the tickets as compensation for their appearance at the event. This means that they are also interested in the sale of tickets to have as little friction as they possibly can because this will increase the amount of sales they will make.

- Payment processors
    - Processors want each transaction to be formatted correctly so that each transaction can be authorized.

- Venue owners
    - Venue owners are another party that directly profits off of the sale of tickets and so also have a vested interest in having each ticket transaction to be carried out as effortlessly and efficiently as possible.

Preconditions: User is logged in and has selected an event (that has ticket inventory) they would like to purchase a ticket for

Success Guarantee: The sale will be completed with a message stating the purchase was a success. The actual ticket would have been purchased and removed from available inventory so no one else may purchase the same ticket as the user. Payment authorization approvals are recorded.

Main Success Scenario:

- ➔ The user once on an event information page can select the option to purchase a ticket.
- ➔ [If time allows] the database is queried for available seats
- ➔ [If time allows] the user then will be able to select a seat
- ➔ The user is then taken to a payment processing page where they are asked to enter in their payment details.
- ➔ Once the user enters in their payment information, a verification process is conducted to make sure that the payment is legitimate
- ➔ After the verification is completed, the transaction occurs
- ➔ After the transaction, the user will be taken to a success screen
- ➔ The database will be updated with the new number of available seats/disable the seat purchased
- ➔ [If time allows] The user will be able to select a button on the success page to add the event to their google calendar.

Extensions:

*a. If a database query fails, cancel the purchase process, notify the actor, and return the actor to the purchase screen

6a. If at any point the user exits the webpage while a transaction is occurring, cancel the transaction

8a. If the database update fails, retry the operation

10a. A calendar event will be provided to the user to add to the calendar.

Special Requirements:

- Separation of website and transaction location for optimal security
- Authorization response within 30 seconds 90% of the time
- Option to cancel transaction if there is a failure in the payment process
- Ability to refund users if failure of system and card is charged incorrectly
- Payment approval process to ensure that the funds do exist prior to updating inventory
- Technology and Data Variations List (Varying I/O methods and data formats)
- [If time allows] user interface that displays a map of available seats from which a selection can be made.
- Credit card information is entered manually by user
- The verification process will work in cooperation with applicable banking systems
- Credit payment signature captured on digital receipt
- Frequency of Occurrence

- Only once the purchase of a ticket has been made.

Open Issues:
What to do when the database update option fails? Which system do we use to ensure there are sufficient funds?

Logging In

Scope: Secure Ticketing System

Level: User goal

Primary Actor: Customer Interested in the Purchase of Tickets

Stakeholders and Interests:

- Customers
  - Users must be authenticated and authorized in order to be able to access the system so that they can purchase and search for tickets.

- Event Organizers
  - In order to list and sell tickets on the system, Event Organizers must have an account and log in prior to taking any actions surrounding the listing of an event.

- Event Contributors
  - Similar to event organizers, event contributors are invested in how well the tickets for the event sell. This means they also must need to log in and hold an authorized account with the system.

- Company
  - The company is invested in having as little friction (but maximum safety) on the log-in process as possible so that potential customers can login quickly and securely to make the purchase of tickets extremely efficient.

Preconditions: User is connected to the internet and has access to a search engine.

Success Guarantee: The user is able to login securely and quickly.

Main Success Scenario:

➔ The user enters our website name
➔ The user selects the option to login
➔ The user is prompted to enter their username and password
➔ The system cross references the username to see if that is the correct password associated with the account
➔ The user is then logged into the system and may search, purchase, or list event tickets.

Extensions

4a. If the username and password matchup, the user enters the system

4b. If the username and password do not match, the user is notified that the username and/or password is incorrect and will then have the option to try again

Special Requirements:

- Internet- allows connection between software, databases, user, and etc. to be established.
- A device that can access the internet
- A web browser on the device
- Proper error messages to be displayed if the user makes a mistake
- It is as simple as possible

Technology and Data Variations List:
2a. Each username must be unique

2a. The password of each account must follow the specific password guidelines

Frequency of Occurrence: Each time the user wishes to access the system from a recognized device or after logging out.

Open Issues:
-Spelling Mistakes
-Forgotten Passwords
-Username creation

Creating an Account

Scope: Secure Ticketing System

Level: User goal

Primary Actor: Customer Interested in the Purchase of Tickets

Stakeholders and Interests:

- Customers
    - Users must be authenticated and authorized in order to be able to access the system so that they can purchase and search for tickets.

- Event Organizers
    - In order to list and sell tickets on the system, Event Organizers must have an account and log in prior to taking any actions surrounding the listing of an event.

- Event Contributors
    - Similar to event organizers, event contributors are invested in how well the tickets for the event sell. This means they also must need to log in and hold an authorized account with the system.

- Company
    - The company is invested in having as little friction (but maximum safety) on the log-in process as possible so that potential customers can login quickly and securely to make the purchase of tickets extremely efficient.

Preconditions: User is connected to the internet and has access to a search engine.

Success Guarantee: The user is able to login securely and quickly.

Main Success Scenario:

➔ The user enters our website name
➔ The user selects the option to create an account
➔ The user is prompted to enter a username, password, and email address to be associated with the account
➔ The system checks to see if there already exists a user with the username provided
➔ The system checks the password to see if it is 15 characters and includes a lower case letter, upper case letter, number, and special character
➔ The new user is added to the user database
➔ The user is then logged into the system and may search, purchase, or list event tickets.

Extensions:
4a. If there already exists a user with that username, the user is given an error message stating that the username is already taken

5a. If the password doesn't match the requirements, the user is given an error message stating that the password does not meet requirements

Special Requirements:
- Internet- allows connection between software, databases, user, and etc. to be established.
- A device that can access the internet
- A web browser on the device
- Proper error messages to be displayed if the user makes a mistake
- It is as simple as possible

Technology and Data Variations List:
3a. Each username must be unique

3b. The password of each account must follow the specific password guidelines

Frequency of Occurrence: Once, when the user first uses the website

Open Issues:
Database access

Abuse Case

Buying all tickets for resale

Scope: College of Charleston Ticketing System

Level: Misuse of core functionality

Primary User: Scalper/ticket reseller

Stakeholders: Potential customers who want tickets, investors , system maintainers

Preconditions:

- Tickets for sale
- Tickets are limited

Guarantee:

The abuser is able to purchase all or almost all available stock for a ticket before many other people have had an opportunity to

Main Success Scenario:

After attempting to purchase tickets in a large quantity, especially towards the beginning of the tickets availability, the abuser is blocked from going through with the transaction and is informed that they are only able to purchase a limited amount of tickets and thus the translation cannot proceed

Extensions:

The user is buying many tickets for a large group of people

Special Requirements:

Tickets have to be available, recently purchasable, and in demand

Operation: startPurchase(eventId : int, model: Model) : String

Cross Reference: Abuse case: purchasing all or most of a tickets stock

Preconditions: Ticket must be purchasable

Post conditions

- The ticket is checked if its a high seller or if its brand new and if it is the user is unable to purchase requested amount
- If neither is true the requested amount is deemed fine to purchase and the user is not aware that the procedure even occurred

Misuse Case

Accidental Purchase of a Ticket

Scope
College of Charleston Secure Ticketing System Level: Misuse of a necessary function

Primary Actor:
Actual User of the Secure Ticketing System Stakeholder: The Account Owner

Preconditions

- The user has to be logged in
- The user has to clicked on (some) event's details
- Success (Failure) Guarantee
- The user is searching through the offered events at the College of Charleston. Whether it is the event they were planning to purchase tickets for (but not ready to purchase them just yet), or a completely random event, the user misclicks/slips/or something of similar action and accidentally selects 'Purchase' for an event they were not prepared to purchase a ticket for.

Main Success Scenario:
The user comes to the point where they have clicked 'Purchase' on an event they were not meaning to purchase a ticket for. Thinking they have already purchased the ticket, the system luckily has implemented a purchase confirmation step before actual purchase of the ticket. So, instead of the ticket being automatically purchased (or even added to the cart if we get to that point), the system displays a message that says, "Are you sure you want to purchase a ticket for ________?" with the options yes or no. Now, the user can simply select 'no' and not have to worry about accidentally purchasing a ticket they did not want. Special Requirement: The option to purchase the ticket has to be present (So, there has to be remaining inventory for the accidental purchase to even think about occurring)

Frequency of Occurrence
Hopefully never.

Operation: confirmPurchase(eventID: int, cardNumber: String, expDate: String, cvv: int, zipCode: int) : String
Cross Reference: Misuse Case- Accidental Purchase of a Ticket

Preconditions: User must have already selected to purchase a ticket for some event

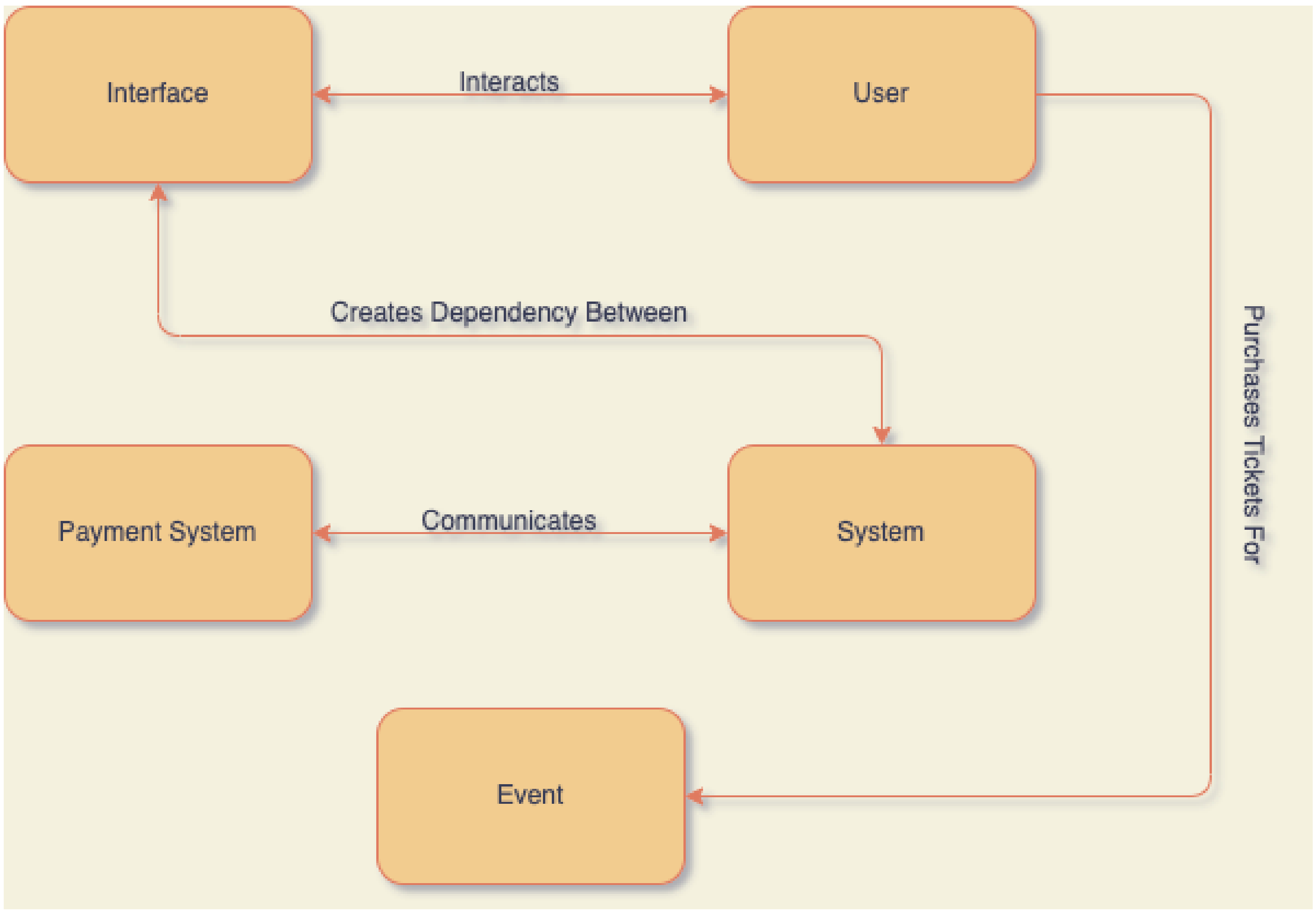
Post-Conditions:
- A confirmation option is sent to the UI of the user
- The user selects whether or not they confirm the purchase of their ticket ** If Yes: purchase(id) ** Else: cancelPurchase(eventId : int, model Model) : String

Operation: cancelPurchase(eventId : int, model Model) : String
Cross Reference: Misuse Case- Accidental Purchase of a Ticket

Preconditions: User must have already selected to purchase a ticket for some event

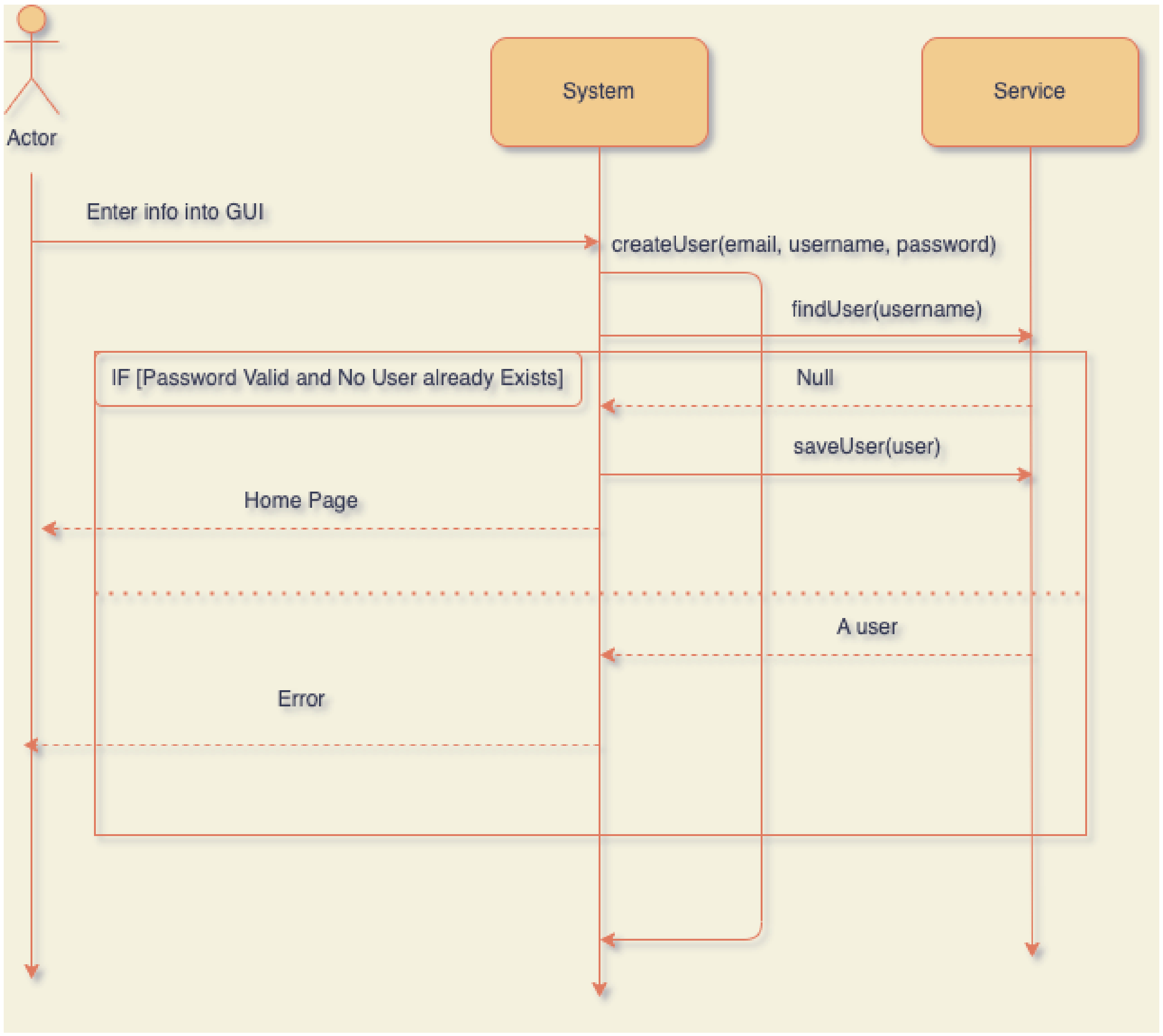
Post-Conditions:
- A confirmation option is sent to the UI of the user
- The user selects no
- The purchase of the ticket is canceled.

Domain Model

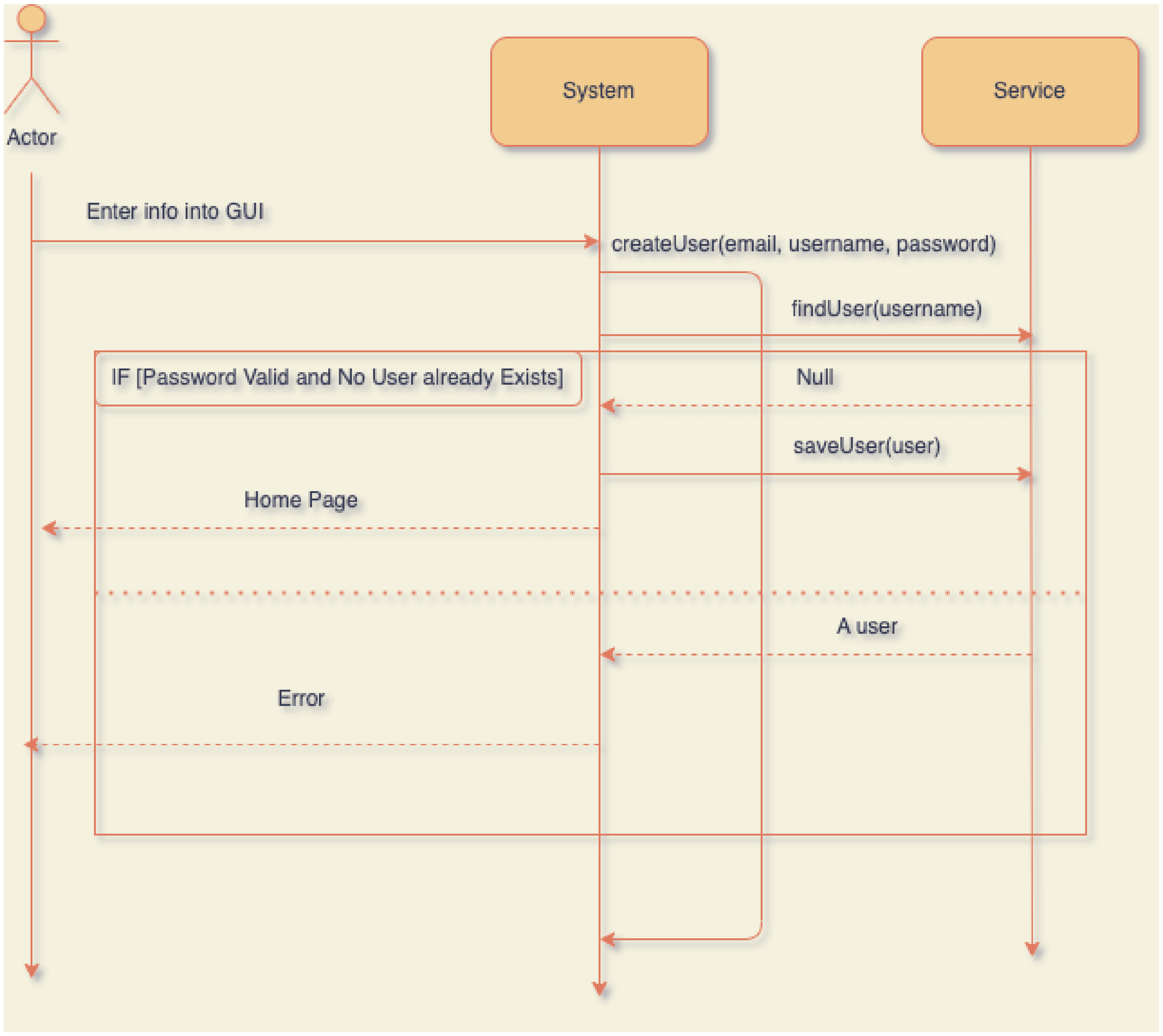


System Sequence Diagrams

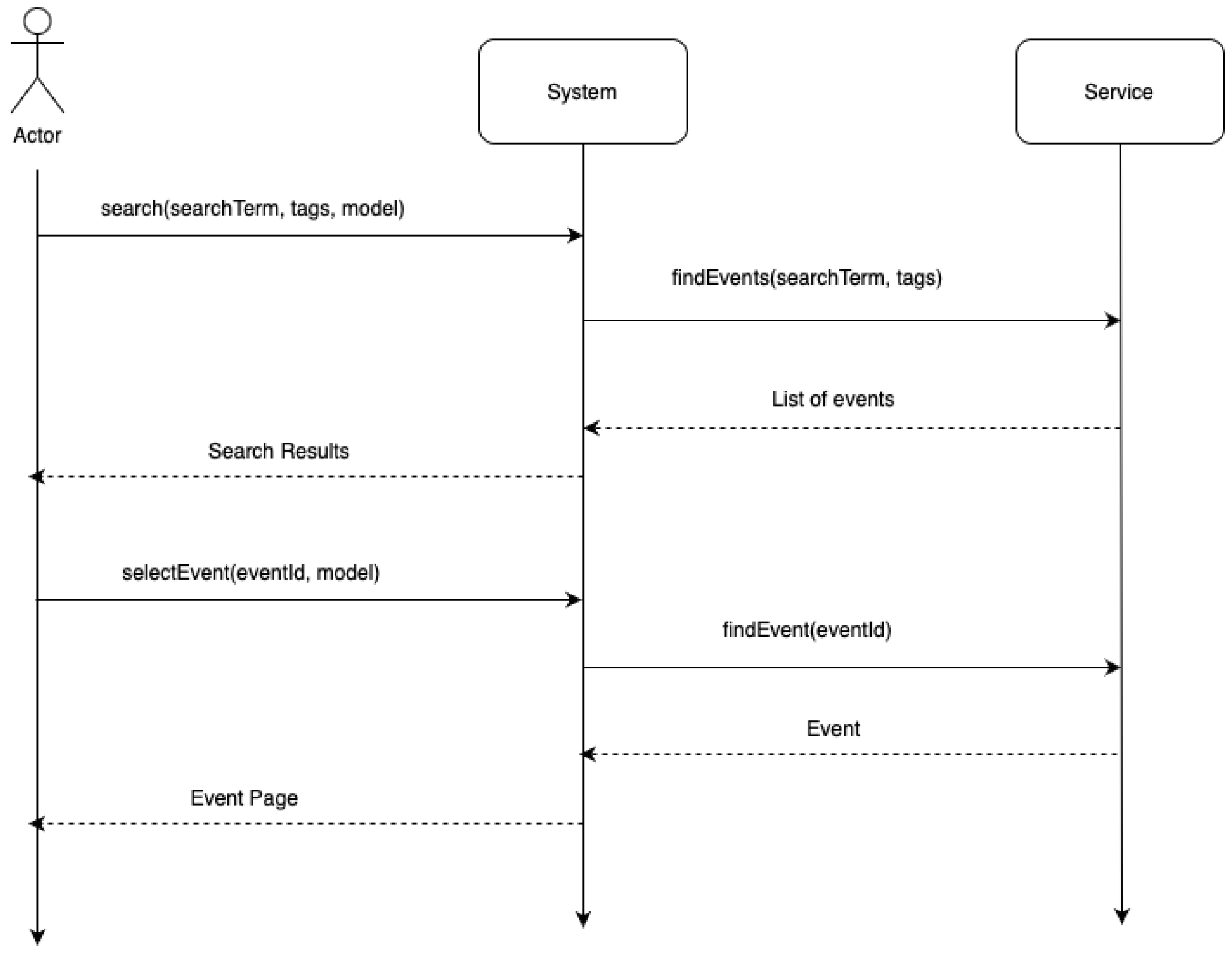
Creating an Account:



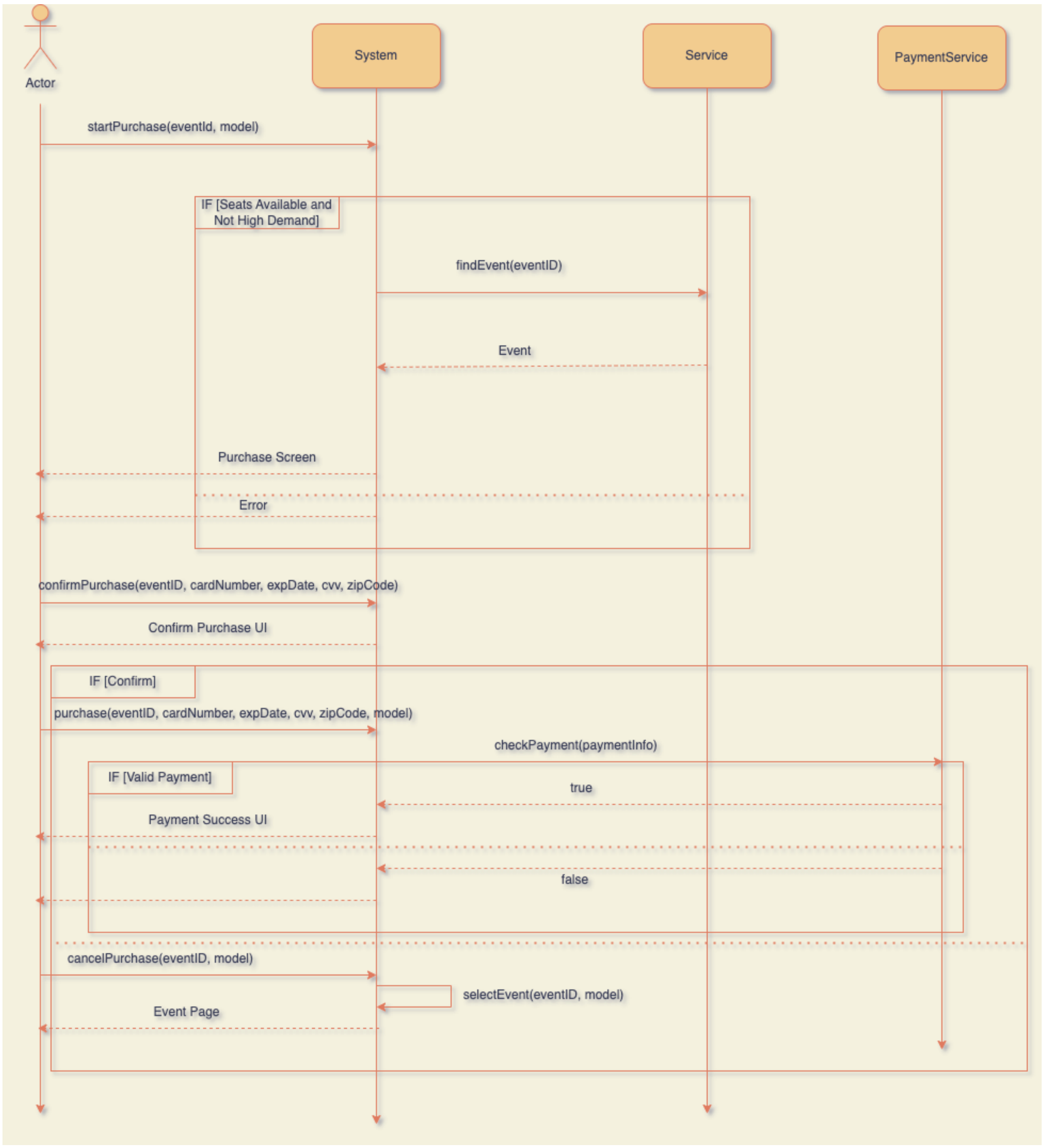
Logging In:


Actor
System
Service
Enter info into GUI
createUser(email, username, password)
findUser(username)
IF [Password Valid and No User already Exists]
Null
saveUser(user)
Home Page
A user
Error

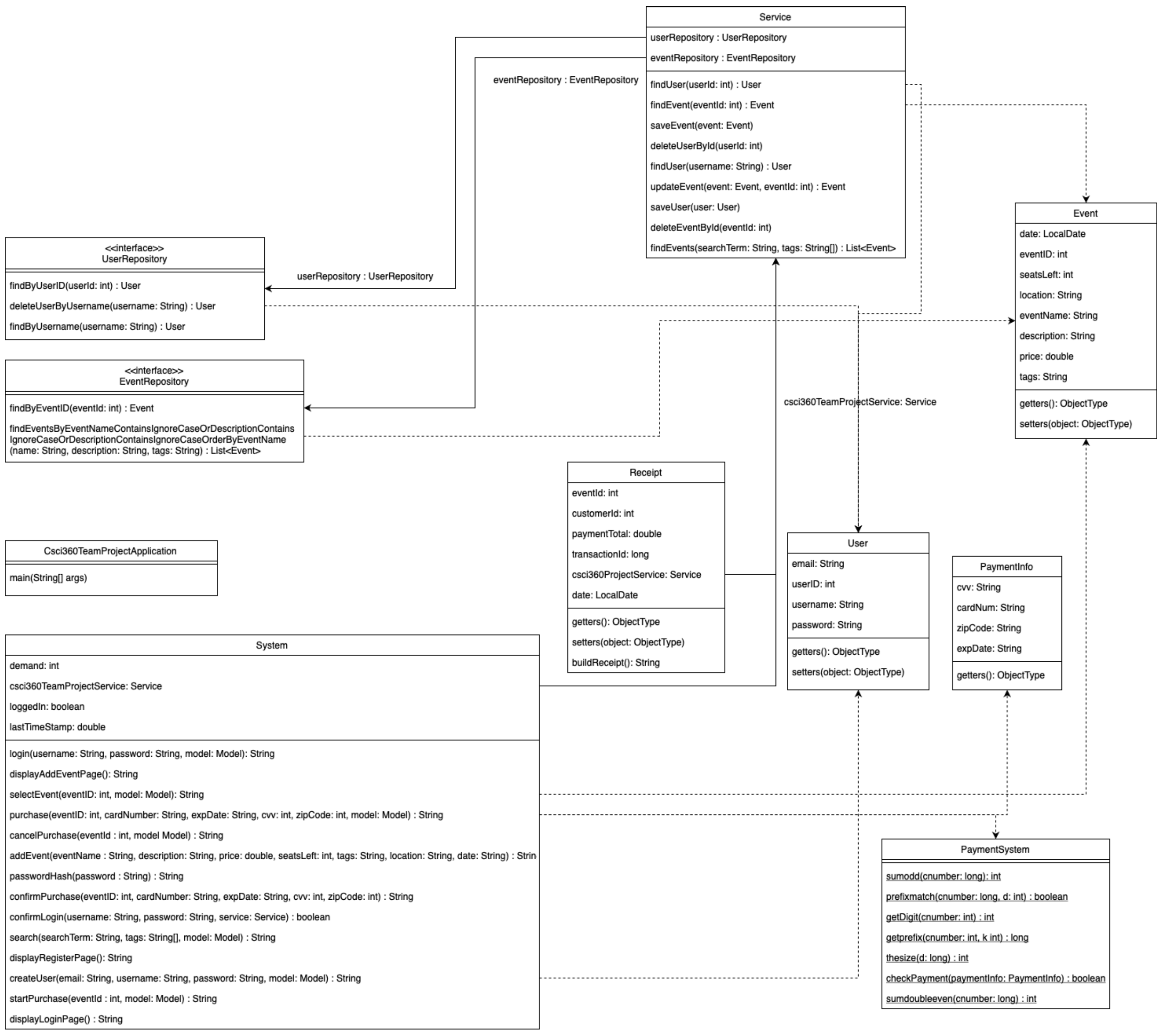
Search for Events:


Actor
System
Service
search(searchTerm, tags, model)
findEvents(searchTerm, tags)
List of events
Search Results
selectEvent(eventId, model)
findEvent(eventId)
Event
Event Page

Purchasing a Ticket (Is the same SSD for Misuse Case: [Accidental Purchase of a Ticket]):


Actor
System
Service
PaymentService
startPurchase(eventId, model)
IF [Seats Available and Not High Demand]
findEvent(eventID)
Event
Purchase Screen
Error
confirmPurchase(eventID, cardNumber, expDate, cvv, zipCode)
Confirm Purchase UI
IF [Confirm]
purchase(eventID, cardNumber, expDate, cvv, zipCode, model)
checkPayment(paymentInfo)
IF [Valid Payment]
true
Payment Success UI
false
cancelPurchase(eventID, model)
selectEvent(eventID, model)
Event Page

Class Diagram


Service
userRepository : UserRepository
eventRepository : EventRepository
findUser(userId: int) : User
findEvent(eventId: int) : Event
saveEvent(event: Event)
deleteUserById(userId: int)
findUser(username: String) : User
updateEvent(event: Event, eventId: int) : Event
saveUser(user: User)
deleteEventById(eventId: int)
findEvents(searchTerm: String, tags: String[]) : List<Event>
eventRepository : EventRepository
userRepository : UserRepository
<<interface>>
UserRepository
findByUserID(userId: int) : User
deleteUserByUsername(username: String) : User
findByUsername(username: String) : User
<<interface>>
EventRepository
findByEventID(eventId: int) : Event
findEventsByEventNameContainsIgnoreCaseOrDescriptionContainsIgnoreCaseOrDescriptionContainsIgnoreCaseOrderByEventName(name: String, description: String, tags: String) : List<Event>
Event
date: LocalDate
eventID: int
seatsLeft: int
location: String
eventName: String
description: String
price: double
tags: String
getters(): ObjectType
setters(object: ObjectType)
csci360TeamProjectService: Service
Receipt
eventId: int
customerId: int
paymentTotal: double
transactionId: long
csci360ProjectService: Service
date: LocalDate
getters(): ObjectType
setters(object: ObjectType)
buildReceipt(): String
User
email: String
userID: int
username: String
password: String
getters(): ObjectType
setters(object: ObjectType)
PaymentInfo
cvv: String
cardNum: String
zipCode: String
expDate: String
getters(): ObjectType
Csi360TeamProjectApplication
main(String[] args)
System
demand: int
csci360TeamProjectService: Service
loggedIn: boolean
lastTimeStamp: double
login(username: String, password: String, model: Model): String
displayAddEventPage(): String
selectEvent(eventID: int, model: Model): String
purchase(eventID: int, cardNumber: String, expDate: String, cvv: int, zipCode: int, model: Model) : String
cancelPurchase(eventId : int, model Model) : String
addEvent(eventName : String, description: String, price: double, seatsLeft: int, tags: String, location: String, date: String) : Strin
passwordHash(password : String) : String
confirmPurchase(eventID: int, cardNumber: String, expDate: String, cvv: int, zipCode: int) : String
confirmLogin(username: String, password: String, service: Service) : boolean
search(searchTerm: String, tags: String[], model: Model) : String
displayRegisterPage(): String
createUser(email: String, username: String, password: String, model: Model) : String
startPurchase(eventId : int, model: Model) : String
displayLoginPage() : String
PaymentSystem
sumodd(cnumber: long): int
prefixmatch(cnumber: long, d: int) : boolean
getDigit(cnumber: int) : int
getprefix(cnumber: int, k int) : long
thesize(d: long) : int
checkPayment(paymentInfo: PaymentInfo) : boolean
sumdoubleeven(cnumber: long) : int

Operation Contracts

System

Contract C01:	Login to System
Operation:	login(username : String, password : String, model: Model) : String
Cross Reference	Use Cases: Logging In
Preconditions:	User has entered in login info to respective text areas
Postconditions:	The user is logged into the site or is given a failure message if their login info is not valid

Contract C02:	Display Add Event Page
Operation:	displayAddEventPage() : String
Cross Reference:	GUI way to add events. Not necessarily a part of the website
Preconditions:	None
Postconditions:	GUI with a way to add an event to the database displayed

Contract C03:	Select Event
Operation:	selectEvent(eventID: int, model: Model) : String
Cross Reference:	Use Cases: Searching for an Event
Preconditions:	User has clicked on an event on the search page.
Postconditions:	The user was directed to the event page for the event specified by event ID. The product details page was populated with the event data of the event specified by event id.

Contract C04:	Purchase
Operation:	purchase(eventID: int, cardNumber: String, expDate: String, cvv: int, zipCode: int, model: Model) : String
Cross Reference:	Use Cases: Purchasing a ticket
Preconditions:	User has confirmed the purchase
Postconditions:	If the payment info was valid, go to the purchase success screen and decrement seats left for the event, otherwise go to an error screen

Contract C05:	Cancel Purchase
Operation:	cancelPurchase(eventId : int, model Model) : String
Cross Reference:	Misuse Case: Accidental Purchase of a Ticket
Preconditions:	User selected the cancel purchase button on the confirm purchase screen
Postconditions:	The user was redirected back to the product details screen through a call of the selectEvent method

Contract C06:	AddEvent
Operation:	addEvent(eventName : String, description: String, price: double, seatsLeft: int, tags: String, location: String, date: String) : String
Cross Reference:	GUI way to add events. Not necessarily a part of the website
Preconditions:	User has pressed the add event button on the add event gui page
Postconditions:	A new event object was added to the database with the following above parameters

Contract C07:	Password Hash
Operation:	passwordHash(password : String) : String
Cross Reference:	Use Cases: Logging In, Creating an Account
Preconditions:	The users un-hashed password was provided to the back end when the user creates an account or is logging in
Postconditions:	The password was hashed using SHA-256 and returned by the method

Contract C08:	Confirm Purchase
Operation:	confirmPurchase(eventID: int, cardNumber: String, expDate: String, cvv: int, zipCode: int) : String
Cross Reference:	Misuse Case: Accidental Purchase of a Ticket
Preconditions:	User must have already selected to purchase a ticket for some event and filled out their payment details
Postconditions:	A confirmation option was sent to the UI of the user. The user can select whether or not they confirm the purchase of their ticket. If Yes: purchase(id). Else: cancel purchase(id).

Contract C09:	Confirm Login
Operation:	confirmLogin(username: String, password: String, service: Service) : boolean
Cross Reference:	Use Case: Logging In
Preconditions:	User entered in their login details and hit the login button in the GUI
Postconditions:	The username was checked to see if there exists a user in the database. If not, return false. If true, the password was hashed and checked to see if it matches the hashed password listed in the database. If they match, true is returned. Otherwise, false was returned

Contract C10:	Search
Operation:	search(searchTerm: String, tags: String[], model: Model) : String
Cross Reference:	Use Case: Searching for an Event

Preconditions:	User entered information into the search bar and pressed the search button
Postconditions:	The UI shifted to the search results page and events that match the search term were displayed on the page.

Contract C11:	Display Register Page
Operation:	displayRegisterPage() : String
Cross Reference:	Use Case: Creating an Account
Preconditions:	User hit the register button on any screen
Postconditions:	The register UI was displayed

Contract C12:	Create User
Operation:	createUser(email: String, username: String, password: String, model: Model) : String
Cross Reference:	Use Case: Creating an Account
Preconditions:	User entered in account info on the register UI page and pressed the register button
Postconditions:	The username is checked to see if it already exists in the database and the password is checked to see if it matches MVP requirements (15 letters, lowercase & uppercase letters, at least 1 number, at least one special character). If a user already exists in the database with the username provided or the password did not comply with requirements, the user was taken to an error screen. Otherwise, the password was hashed, the user was added to the database, and the user is redirected back to the home screen

Contract C13:	Start Purchase
Operation:	startPurchase(eventId : int, model: Model) : String
Cross Reference:	Use Case: Purchasing a Ticket, Abuse Case: Buying all of a tickets stock
(Scalping)	
Preconditions:	User clicked the buy now on the product page
Postconditions:	If the buy now button was clicked 5 times in the last 5 seconds (can be adjusted), return an error. If the user is not logged in, display the login screen. If the event has no seats left, return an error. Otherwise, go to the purchase UI screen and add the event details to the page

Contract C14:	Display Login Page
Operation:	displayLoginPage() : String
Cross Reference:	Use Case: Logging In
Preconditions:	User pressed the login button on any page
Postconditions:	The login UI page was displayed

User, Event, Receipt, and PaymentInfo

Contract C15:	Get Attribute
Operation:	getAttribute()
Cross Reference:	Use Case: All
Preconditions:	Object was instantiated
Postconditions:	Attribute was returned

User, Event, and Receipt

Contract C16:	Set Attribute
Operation:	setAttribute(attribute: ObjectType)
Cross Reference:	Use Case: All
Preconditions:	Object was instantiated
Postconditions:	The old attribute was replaced with attribute

Receipt

Contract C17:	Build Receipt (Unused)
Operation:	buildReceipt() : String
Cross Reference:	Use Cases: Purchasing a Ticket
Preconditions:	Receipt object created and has all of its fields filled with info.
Postconditions:	A string is created with all of the fields needed for a receipt

Service

Contract C18:	Find User
Operation:	findUser(userId: int) : User
Cross Reference:	Use Cases: All
Preconditions:	User with specified id is in the database
Postconditions:	User with specified id was returned

Contract C19:	Find Event
Operation:	findEvent(eventId: int) : Event
Cross Reference:	Use Cases: All
Preconditions:	Event with specified id is in the database
Postconditions:	Event with specified id was returned

Contract C20:	Save Event

Operation:	saveEvent(event: Event)
Cross Reference:	Use Cases: All
Preconditions:	event is not null
Postconditions:	Event with specified id was saved to the database

Contract C21:	Delete By User ID
Operation:	deleteUserById(userId: int)
Cross Reference:	Test cases
Preconditions:	User with specified id is in the database
Postconditions:	User with specified id was removed from the database

Contract C22:	Find User by Username
Operation:	findUser(username: String) : User
Cross Reference:	Use Cases: All
Preconditions:	User with specified username is in the database
Postconditions:	User with specified username was returned

Contract C23:	Update Event
Operation:update	Event(event: Event, eventId: int)
Cross Reference:	Use Cases: Purchasing a Ticket
Preconditions:	Event with specified id is in the database
Postconditions:	Event with specified id was replaced with event in the database

Contract C24:	Save User
Operation:	saveUser(user: User)
Cross Reference:	Use Cases: All
Preconditions:	user is not null
Postconditions:	User with specified id was saved to the database

Contract C25:	Delete Event By Id
Operation:	deleteEventById(eventId: int)
Cross Reference:	Test cases
Preconditions:	Event with specified id is in the database
Postconditions:	Event with specified id was removed from the database

Contract C26:	Find Events
Operation:	findEvents(searchTerm: String, tags: String[]) : List
Cross Reference:	Searching for an event
Preconditions:	Search term is not null
Postconditions:	A list of events that are related to the search terms and tags were returned

UserRepository

Contract C27:	Find By User ID (Unused)
Operation:	findByUserID(userId: int) : User
Cross Reference:	None
Preconditions:	User with specified id is in the database
Postconditions:	User with specified id was returned

Contract C28:	Delete By Username (Unused)
Operation:	deleteUserByUsername(username: String) : User
Cross Reference:	None
Preconditions:	User with specified username is in the database
Postconditions:	User with specified user was removed from the database

Contract C29:	Find By Username
Operation:	findUser(username: String) : User
Cross Reference:	Use Cases: All
Preconditions:	User with specified username is in the database
Postconditions:	User with specified username was returned

EventRepository

Contract C30:	Find By Event ID (Unused)
Operation:	findEvent(eventId: int) : Event
Cross Reference:	Use Cases: All
Preconditions:	Event with specified id is in the database
Postconditions:	Event with specified id was returned

Contract C31:	Search for Events
Operation:	findEventsByEventNameContainsIgnoreCaseOrDescriptionContainsIgnoreCaseOrDescriptionContainsIgnoreCaseOrderByEventName(name: String, description: String, tags: String) : List<Event>
Cross Reference:	Searching for an event
Preconditions:	None
Postconditions:	A list of events that are related to the name, description, and tags were returned

PaymentSystem

Contract C32:	Sum Odd
Operation:	sumodd(cnumber: long) : int
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return sum of odd-place digits in cnumber

Contract C33:	Prefix Match
Operation:	prefixmatch(cnumber: long, d int) : boolean
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return true if the digit d is a prefix for cnumber. Otherwise return false

Contract C34:	Get Digit
Operation:	getDigit(cnumber: int) : int
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return this cnumber if it is a single digit, otherwise, return the sum of the two digits

Contract C35:	Get Prefix
Operation:	getprefix(cnumber: int, k int) : long
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return the first k number of digits from cnumber. If the number of digits in number is less than k, return cnumber.

Contract C36:	Number of Digits
Operation:	thesize(d: long) : int
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return the number of digits in d

Contract C37:	Check Payment
Operation:	checkPayment(paymentInfo: PaymentInfo) : boolean
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	paymentInfo is instantiated
Postconditions:	Return true if the fields of paymentInfo are set to valid values. Otherwise, return false

Contract C38:	Sum Double Even
Operation:	sumdoubleeven(cnumber: long) : int
Cross Reference:	Use Cases: Purchase a Ticket
Preconditions:	none
Postconditions:	Return the sum of all even digits in cnumber doubled

Csci360TeamProjectApplication

Contract C39:	Main
Operation:	main(args: String[])
Cross Reference:	Use Cases: All
Preconditions:	None
Postconditions:	The project was launched