Software Architecture Document

Architectural Representation

Welcome to the CofC Ticketing System! Here is our overview on the project: System Sequence Diagrams (SSDs) visually represent interactions between external actors and a system, capturing high-level event sequences. Class Diagrams illustrate the static structure of a system, depicting classes, attributes, and relationships as foundational blueprints for implementation. Operation contracts define the expected behavior of methods within a class, specifying preconditions, postconditions, and invariants. Activity Diagrams provide a dynamic representation of workflow and activities within a system, illustrating the sequence of events, decision points, and flow of control. Fully Dressed Use Cases offer comprehensive documentation for system functionality, including actors, preconditions, and postconditions. Abuse Use Cases focus on intentional misuse scenarios, aiding in identifying vulnerabilities and guiding the development of security measures to protect against potential threats. UI State Machine Models showcase different states of a user interface and transitions between them, crucial for understanding and designing responsive user experiences. Together, these modeling techniques

contribute to system design, communication, security considerations, and dynamic process visualization in the software development lifecycle.

Architectural Factors

Factor	Measures and quality scenarios	Variability (current flexibility and future evolution)	Impact factor (and its variability) on stakeholders, architecture and other factors	Priority for Success	Difficulty or Risk
Recovery from remote service failure	When a remote service fails, reestablish connectivity within 1 minute of its detected availability, under normal store load in a production environment.	Current flexibility- local client-side simplified services are desirable and accepted until reconnected. Evolution- none	High impact on the scability of the design.	H	M
Recovery from remote database failure	As above	Current flexibility- current data is kept in MySQL, but if connection fails then local client-side use of cached information is acceptable until reconnected. Evolution- There will be mass storage and replication solutions.	As above	H	H

Support many third-party services (ticket inventory, sports tournaments, ticket prices) They will vary at each installation.	When a new third-part system must be integrated, it can be almost immediately via front-end or back-end developer.	Current flexibility- adding new sporting events, ticket inventory, and ticket prices. Evolution- Some period of time there should be a system put in place for administrators to input events, tickets, and prices themselves, without developers involvement.	Low design impact as the data is input into the system, but nothing about the design is changed.	H	L
Current tax rules must be applied.	When user is purchasing ticket, all government and state taxe rules should be applied to purchase.	Current flexibility- Current tax rules are not put in place yet. Evolution- Implementing tax rules once system is put out to public.	Failure to comply is a criminal offense.	H	L

Architectural Decisions

Technical Memo: Recovery from remote service failure

Solution Summary: Automate the reestablishment of connectivity to remote services within 1 minute of their detected unavailability, utilizing local client-side simplified services during the downtown.

Factors

- Robust recovery from remote service failure
- Robuse recovery from remote database failure

Solution

In order to achieve optimal service with failure there needs to be backup service available. Instead of storing data in a MySQL database solely, the college should consider storing customer and event data within a cloud-based system that allows for accessibility from multiple zones and regions. If one data center fails, then another should be accessible for the system. It is also important to not only rely on local-client server sides but to make sure the current systems are optimal enough to support the use of many users at once.

Motivation

The college wants to ensure minimal downtime and a seamless experience for users during remote service failures. Local client-side services act as a temporary workaround to maintain critical functionality.

Unresolved Issues

None are identified.

Alternatives Considered

The different tiers of service agreements, based off of cloud-based or connectivity services selected. Each tier improves reliability and efficiency, but some might be too costly.

Technical Memo: Adaptability of other services into the system

Solution Summary: Facilitate easy integration of new third-party systems via both front-end and back-end development, with plans to implement a more user-friendly system for administrators in the future.

Factors

- Integration of new third-party systems should be almost immediate via front-end or back-end development.

Solution

There should be a self automating system that allows for administrators to log onto an admin website to input new events and ticket prices. Once input and submitted, the ticketing system page should update promptly with the new events and their varying factors that come with it.

Motivation

Enable quick and flexible integration of diverse third-party services. Plan for future scalability with an administrator-friendly input system.

Unresolved Issues

Specify details for the administrator-friendly input system.

Alternatives Considered

No alternatives are considered.

Technical Memo: Legal tax rule compliance

Solution Summary: Purchase a tax calculator component.

Factors

- Current taxes must be applied

Solution

Government, state, and local tax authorities and regulations are consistently changing. These occurances happen as often as weekly, biweekly, and/or monthly. In order to keep up with these regulatory changes the college should consider purchasing a tax calculator that changes according to the laws and regulations at any given time.

Motivation

Ensure legal compliance and transparent user experience during ticket purchases. Plan for a phased implementation of tax rules to coincide with the system's public release.

Unresolved Issues

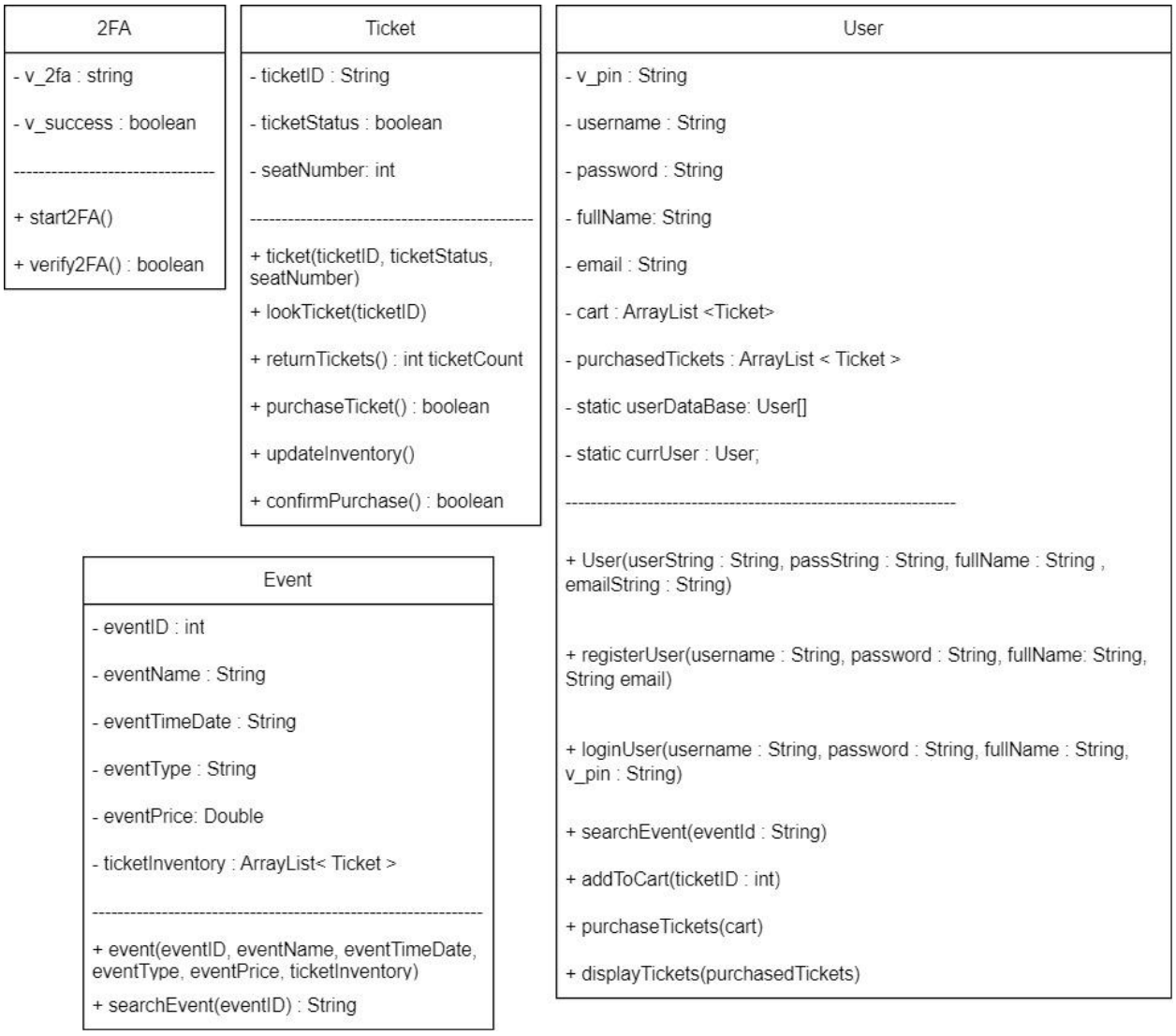
There are no identified unresolved issues.

Alternatives Considered

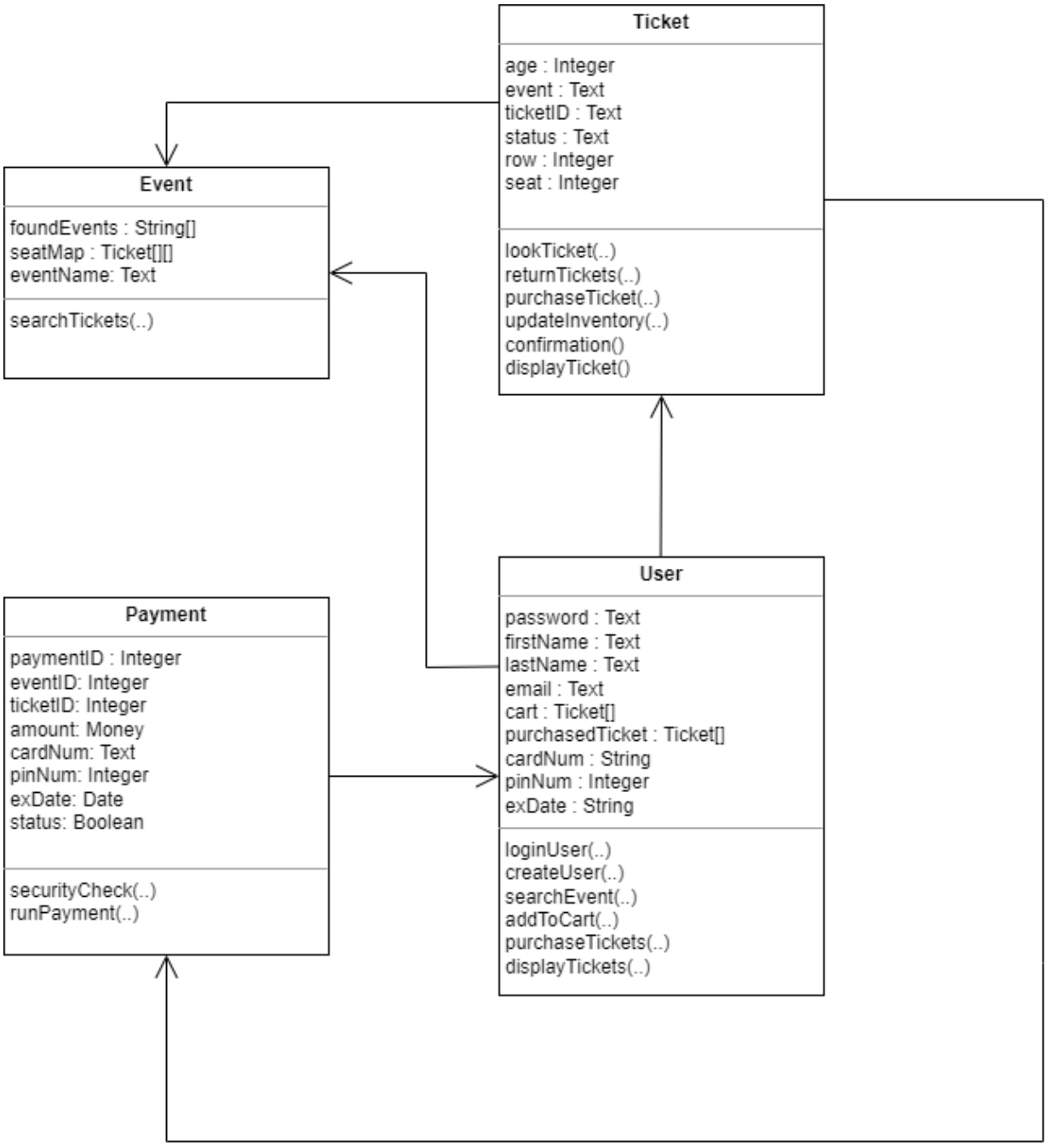
No alternatives are considered

Logical Views

UML Class Diagram



Order of Class Implementation Diagram



Development View

Operations Contract: Check Ticket Availability

Operation	searchEvent(eventName: String)
Cross Reference	Check ticket availability
Pre-conditions	The user must be logged in and on the homepage of the application. The user must have either selected or typed in the specific event they are looking for.
Post-conditions	If they have fulfilled the necessary requirements for searching for the event, then the user will be able to select the event on the application.

Operation	searchTickets(ticketID: String)
Cross Reference	Check ticket availability
Pre-conditions	There is a user, user is logged in, user searched for an event, and has selected the event of their choosing
Post-conditions	Ticket info is updated and returned if there are tickets available.

Operation	returnTickets()
Cross Reference	Check ticket availability
Pre-conditions	The ticket info
Post-conditions	The ticket info and availability is sent and displayed to the user

Operations Contract: Login Accessibility

Operation	registerUser()
Cross Reference	Creating a login for a user (login accessibility)
Pre-conditions	The person must have a unique username and a password that must be 15 characters and include lower case letter, upper case letter, number, and special character
Post-conditions	If they have fulfilled the necessary requirements for a unique username and a password with all of the following conditions, then they will be accepted into the system

Operation	loginUser()
Cross Reference	The system has accepted the user into the system (login accessibility)
Pre-conditions	The person must have entered their unique username and password into the system.
Post-conditions	If they have fulfilled the necessary requirements for their own unique username and a password with all of the following conditions, then they will be accepted into the application.

Operations Contract: Purchase tickets

Operation	lookUpTicket(event tickets : string)
Cross Reference	Purchase tickets
Pre-conditions	The user has logged in, the system is running, and the ticket provided is valid and corresponds to an existing ticket.
Post-conditions	If the operation is successful and a valid ticket is found. The ticket details contain detailed information about the ticket that will be returned.

Operation	displayTickets (list of Tickets: string) Ticket list (Ticket ID, Event Name, Ticket Price, Availability)
Cross Reference	Purchase tickets
Pre-conditions	the system is running and accessible,the user looks up for tickets with valid event names.
Post-conditions	If the operation is successful, the tickets list contains information about available tickets which is Ticket list (Ticket ID, Event Name, Ticket Price, Availability). But if the operation is not successful it will display a message stating no availability, try another event name or date for example.

Operation	AddtoCart (ticketID: int)
Cross Reference	Purchase tickets
Pre-conditions	The TicketID provided exists in the database of available tickets. The user is authorized to modify the specified shopping cart. - The ticket is not already in the user's cart (unless the system supports multiple quantities of the same ticket in a cart).
Post-conditions	If the operation is successful, the specified ticket is added to the user's shopping cart.

Operation	PurchaseTicket(ticketID: int)
Cross Reference	Purchase ticket
Pre-conditions	The user has logged in, added tickets to their cart, and then selected “Purchase Tickets” then the ticket will be saved in the system as ready for purchase.
Post-conditions	The tickets added to cart will be ready for purchase

Operation	confirmation () return (confirmationID: int)
Cross Reference	Purchase tickets
Pre-conditions	The system is running. The Transaction provided is valid and corresponds to an existing, pending transaction, order, or action.
Post-conditions	If the operation is successful, the specified payment transaction is marked as completed.

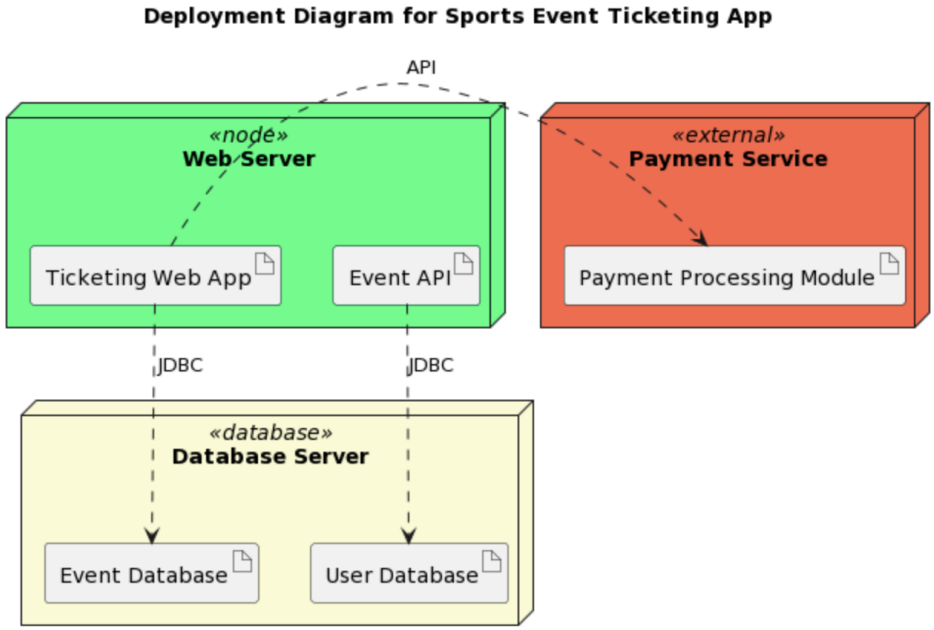
Operation	UpdateInventory(ticketID: int)
Cross Reference	Purchase tickets
Pre-conditions	The user has purchased Tickets, and the ticket inventories need to be updated.
Post-conditions	The ticket inventories are updated and are confirmed.

Motivation

Operation contracts bring an exciting level of precision and clarity to the dynamic world of software development. These contracts serve as a beacon, lighting up the expected behavior of methods within a system, ensuring that developers, testers, and project managers are all on the same page. It's more than just documentation; it's a language that prevents errors, guides testing

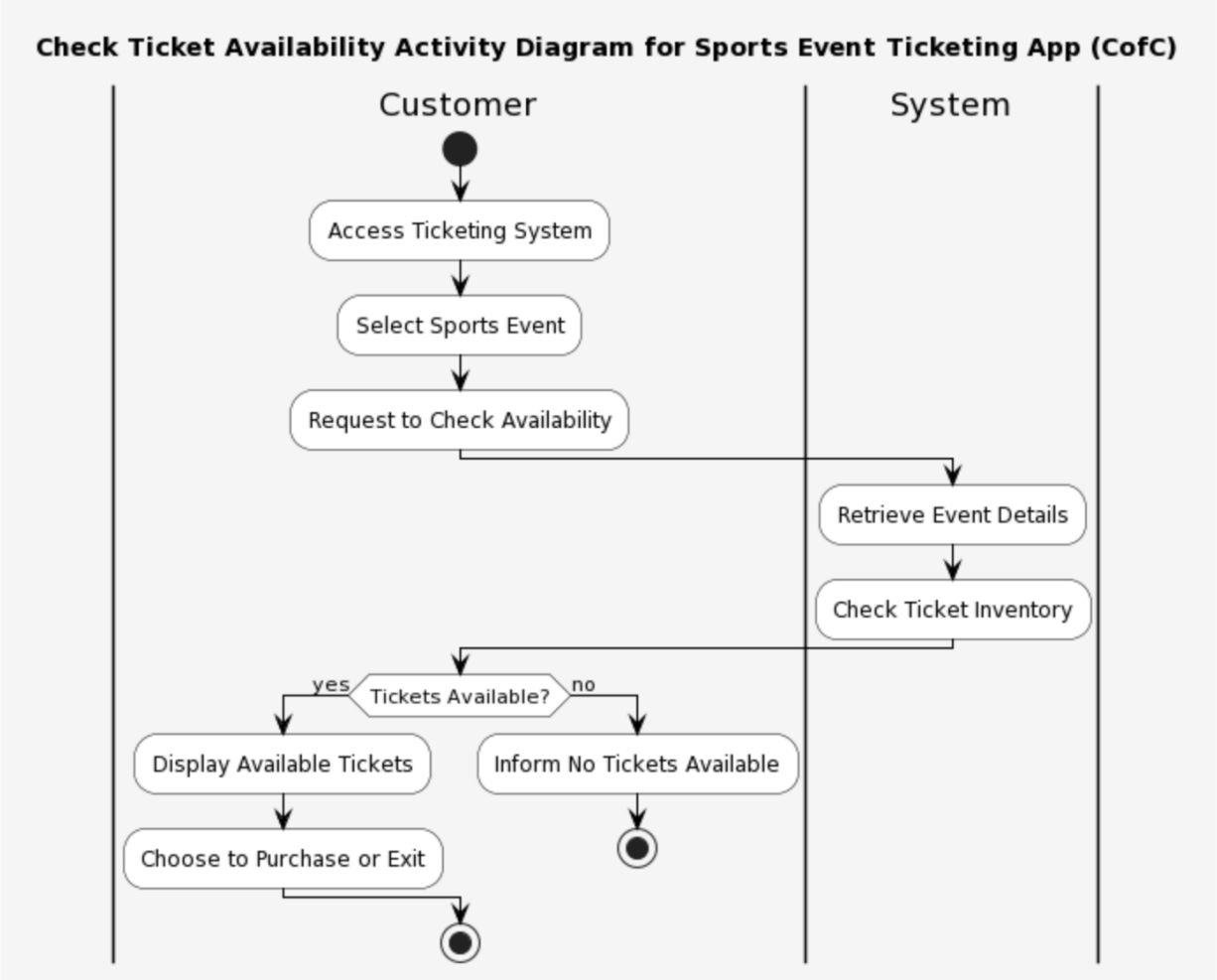
strategies, and fosters a shared understanding of how each method should dance with the rest of the system. The thrill lies in the early detection of errors, the prevention of pitfalls, and the modularity they bring to the design, creating a symphony of code that's not just robust but also comprehensively documented. Operation contracts inject excitement into debugging and maintenance phases, offering a guiding star for developers to navigate through the intricacies of the system. They're not just requirements; they're the promises and guarantees that make software development a thrilling journey of precision, reliability, and elegance.

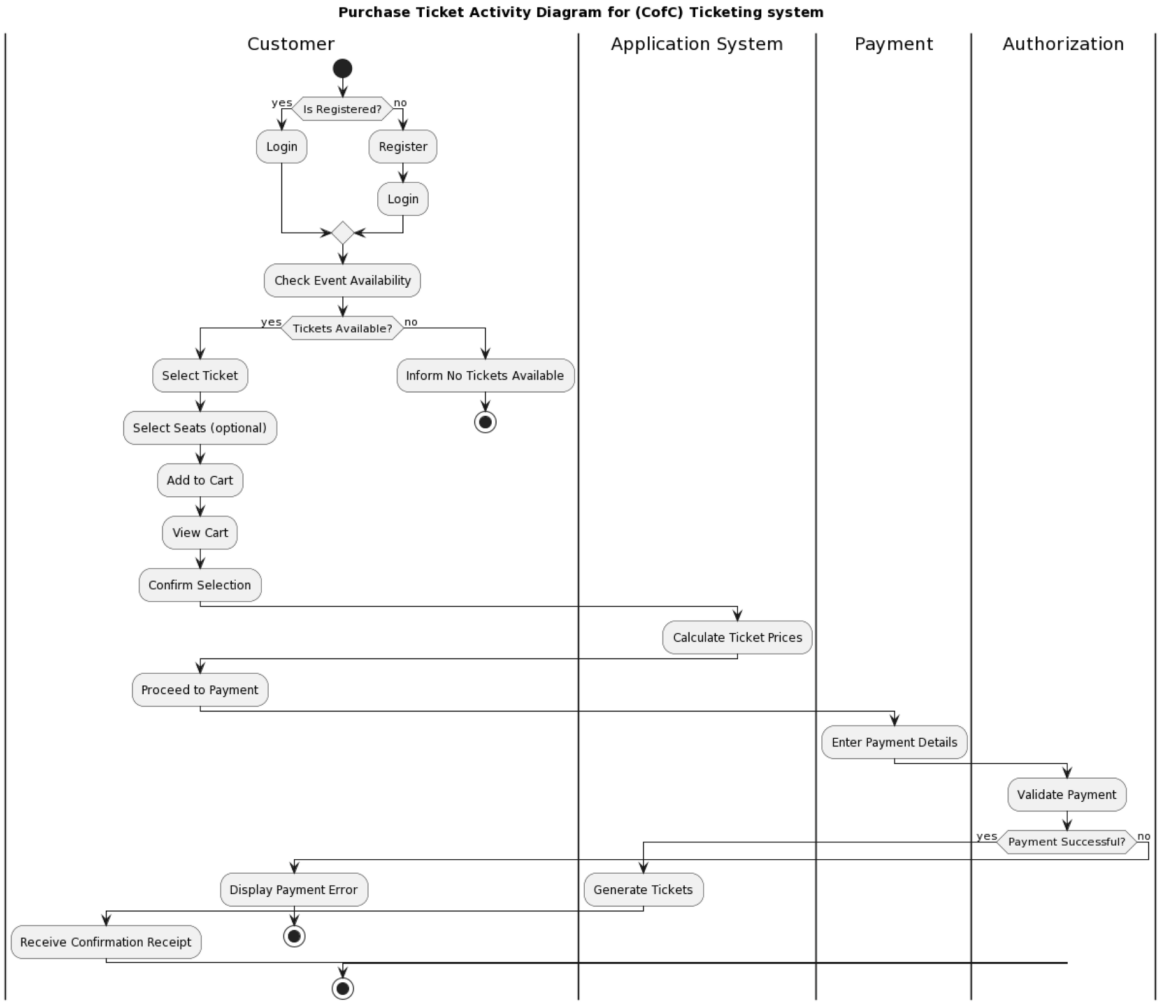
Deployment View



If you like it use it if not i don't know i wish i can have time to fix it but don't have time

Process View





Discussion:
In the Process View, we have detailed activity diagrams that capture the dynamic flow of actions within our Sports Event Ticketing App. These diagrams are tailored to represent two main functionalities: ticket purchasing and availability checking. The depicted processes ensure that the system's logic is both transparent and traceable from the user's action to the system's response.

Motivation:
The motivation behind the Process View is to provide a clear pathway for understanding how users interact with our system and how it reacts to various scenarios. By mapping out each step, we ensure that:

The flow is intuitive and user-friendly, enhancing the overall user experience.
Potential bottlenecks can be identified and optimized.

The system remains flexible to accommodate future enhancements such as new payment options or event types.

Security View

Abuse Case #1 - A malicious user gains access to the credentials of another user's account and attempts to log into the system as if they were said user. This can be done in order to change or steal information of the account owner.

Abuse Case #2 - If a malicious user gains access to another user's account and wishes to purchase tickets using the credit card already used on the account, the malicious user is able to do so.

Motivation:

We want everyone to have a peaceful and protective experience when looking at the CofC Ticketing System! We hope that you are very careful with how you display your information and possibly even delete your credit card number each time that you log on. This can stop you having a very harsh experience and keep coming back to CofC events!

Use Case View

Fully Dressed Use Case #1

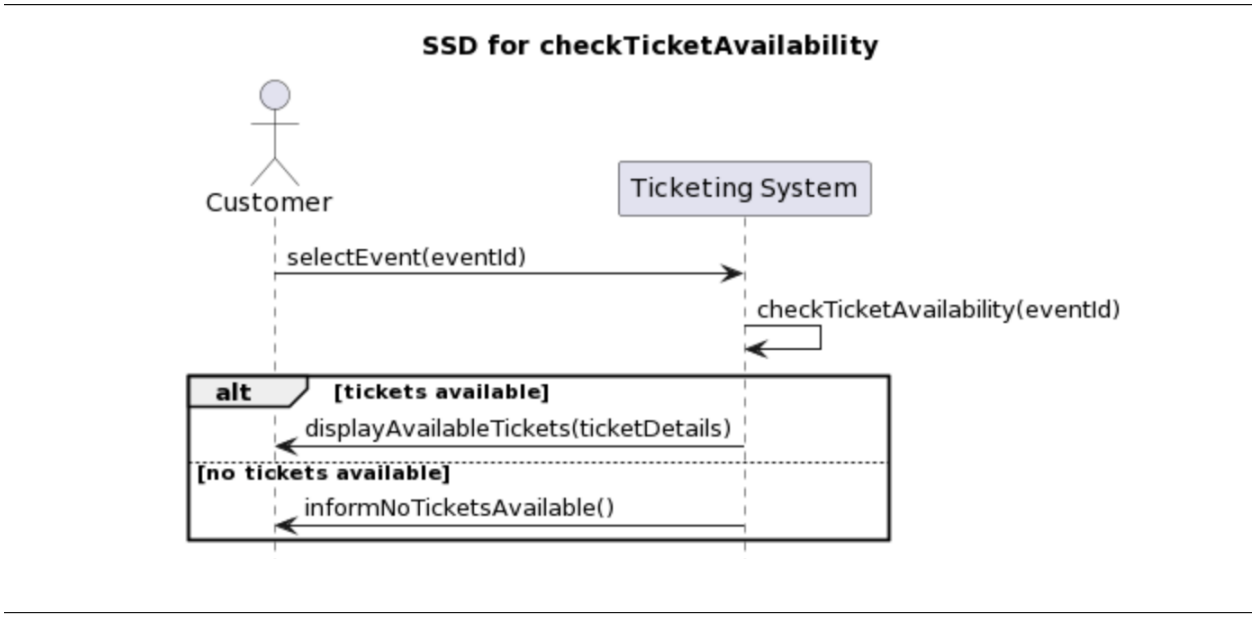
Use Case Name:	Checking Availability
Scope:	Ticketing System
Level:	“User - Goal” or subfunction
Stakeholders and Interest:	Attendee: wants to quickly find out if tickets are available for an event. Event Organizer: needs to monitor ticket sales and availability
Primary Actor:	Any person trying to get tickets for a CofC event
Preconditions:	There must be tickets available for people to buy for the event, and the ticketing system is operational
Success Guarantee:	If they have bought a ticket, then the counter of how many available tickets must go down
Main Success Guarantee:	1. Attendee selects an event to view 2. System displays the number of remaining tickets 3. If ticket are available, the system enables the option to purchase
Extensions:	A failure would involve when people are buying the ticket, the availability counter is not updating No tickets are available
Special Requirements:	Adding a timer for sales, showing how many tickets are left without clicking on the event, and pictures for the background of the event so that it has more appeal to the users.
Technology and Data Variations List:	Use PayPal to buy the tickets, Using Apple Pay instead of manually typing the numbers on the user’s card
Frequency of Occurrence:	Could be nearly continuous

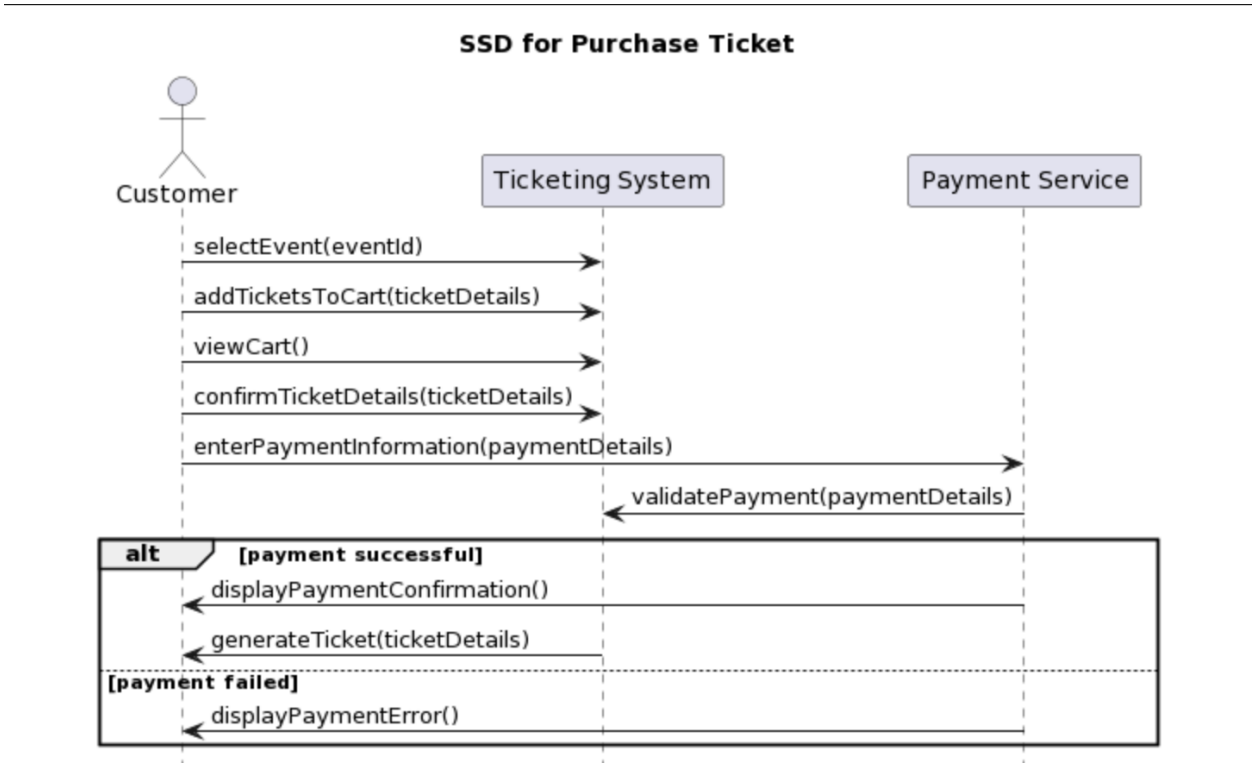
Miscellaneous:	Open Issues (None at the moment, will update per project)

Fully Dressed Use Case #2

Use Case Name:	Purchase ticket
Scope:	Ticketing System
Level:	“User - Goal” or subfunction
Stakeholders and Interest:	Attendee: Wants to secure a ticket for an event at CofC. Payment Processor: Processes payment transactions securely. System Administrator: Ensures that the ticket purchasing process is smooth and handles any issues with ticket inventory or event information.
Primary Actor:	Any person trying to purchase tickets for a CofC event
Preconditions:	Attendee is registered and has selected a ticket to purchase.
Postconditions:	Attendee receives a confirmation of the purchase.
Success Guarantee:	If they have bought the ticket then the screen will display a receipt
Main Success Guarantee:	Attendee chooses to purchase a ticket and proceeds to payment. Attendee enters payment details. System validates payment and confirms the transaction. Attendee receives a ticket receipt.
Extensions:	1. Payment validation fails. 2.System notifies the attendee of the failure and suggests corrective action. 3.Attendees are prompted to re-enter payment details or choose a different payment method.
Special Requirements:	Adding a timer for sales, having the user select the number of tickets they are purchasing, and having the user enter their email and phone number while purchasing
Technology and Data Variations List:	Use PayPal to buy the tickets, Using Apple Pay instead of manually typing the numbers on the user’s card, and using CofC ID to get access to free tickets
Frequency of Occurrence:	Could be nearly continuous (As often as tickets are purchased.

	)
Miscellaneous:	Open Issues (None at the moment, will update per project)





Discussion:
The Use Case View focuses on the "Checking Availability" and "Purchase Ticket" functionalities, which are pivotal to our application's operation. These use cases have been elaborated to cover

all the necessary steps and alternatives, offering a comprehensive perspective of the system's capabilities and interactions with its users.

Motivation:
Our Use Case View is driven by the need to provide a system that is:

Reliable, ensuring that ticket availability is accurately reflected in real-time to avoid overbooking.
Secure, guaranteeing that transactions are processed safely and users' data is protected.
Adaptable, allowing for easy updates and changes to the event catalog or ticketing policies without disrupting the overall system performance.
By providing a detailed discussion and strong motivation for our Process and Use Case Views, we aim to build a robust foundation for the development, scalability, and maintenance of the Sports Event Ticketing App. This approach aligns with our commitment to delivering a quality service that meets the needs of both event organizers and attendees, ensuring a seamless ticketing experience.