CS3354 Software Engineering
Final Project Deliverable 2

Tekcit

making finding, purchasing, and creating your own ticket easier than ever

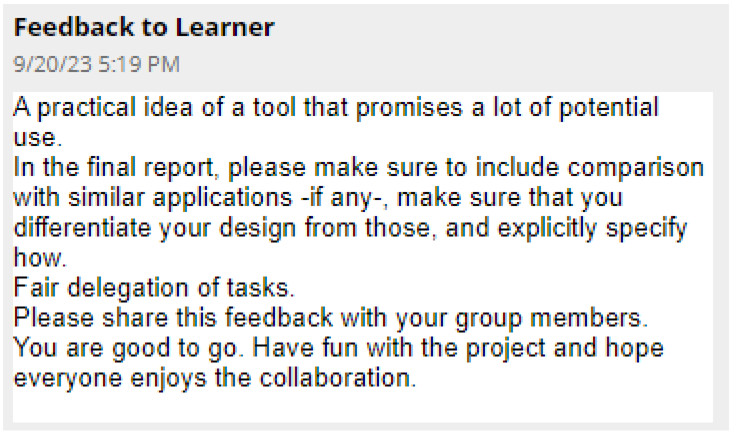
The group members:
❖ Demarcus Braclet ❖ Aryan Kulkarni ❖ Artan Vafaei ❖ Sneha Gopalakrishnan ❖ Sakina Ali
❖ Kai Castellanos ❖ Kevin Dang

1. Delegation of Tasks from the Beginning of the Project

- **Demarcus Braclet**
  - Deliverable 1 Tasks
    - 1.0 - 1.3, 2, 3
  - Deliverable 2 Tasks
    - 1, 2, 10
- **Aryan Kulkarni**
  - Deliverable 1 Tasks
    - 9
  - Deliverable 2 Tasks
    - 6
- **Artan Vafaei**
  - Deliverable 1 Tasks
    - 1.5, 8
  - Deliverable 2 Tasks
    - 3.3, 3.4
- **Sneha Gopalakrishnan**
  - Deliverable 1 Tasks
    - 5
  - Deliverable 2 Tasks
    - 5, 7
- **Sakina Ali**
  - Deliverable 1 Tasks
    - 7
  - Deliverable 2 Tasks
    - 3.1, 3.5
- **Kai Castellanos**
  - Deliverable 1 Tasks
    - 4
  - Deliverable 2 Tasks
    - 3.2, 4
- **Kevin Dang**
  - Deliverable 1 Tasks
    - 1.4, 6
  - Deliverable 2 Tasks
    - 8

2. Project Deliverable 1 Content

1.


Feedback to Learner
9/20/23 5:19 PM
A practical idea of a tool that promises a lot of potential use.
In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.
Fair delegation of tasks.
Please share this feedback with your group members.
You are good to go. Have fun with the project and hope everyone enjoys the collaboration.

- ❖ In our final report, we will include comparisons of different applications, to differentiate our design from them.
- ❖ This time, we will have a fair delegation of tasks (even points distributed between each group member)

2. https://github.com/DemarcusBraclet/3354-Avocado

3.

- ❖ Demarcus - 1.0 - 1.3, 2, 3
- ❖ Kevin - 1.4, 6
- ❖ Aryan - 9
- ❖ Artan -1.5, 8
- ❖ Sneha - 5
- ❖ Sakina - 7
- ❖ Kai - 4, 7

4.

- ❖ We believe that the spiral model is the best. We want our software to grow continuously, and with the spiral model, we can work to always update the ticketing software. The disadvantage is that it is tedious, but we want a fully operable product, with few to no bugs. It is important to note that the spiral model is often tied together with prototyping, but this will not be the case here. For the first delivery, the product should be fully-functional, and as such may be closer in function to a waterfall model, or spiral development with only alpha testing until the first delivery. The product *cannot* function unless fully completed to a well-done degree viably due to the sensitive information it handles and the events of large caliber that it may manage. However, the ticketing software is a service with many customers and users, and so it is important that there is continual progress after it is delivered. A large part of this is because the server load of the ticketing software will greatly change, but also because any continually used product will require management, though we expect the ticketing software to require a lower commitment, only needing to defer to bug and server load management. The reason that the spiral model is going to be used rather than an incremental model is because the risk management that is involved in the spiral model is integral to a service of this scale. In other words, the only important management left after the first delivery would be related to risk management (server load and bugs).

5.a.

1. User authentication:
   - The system must allow users to log in using Google, Facebook, and Outlook accounts.
   - Users must have the ability to create and log in using their own credentials on the website.
2. Ticket display and search:
   - The system must display the trending tickets on the homepage.
   - Users should be able to search for products using a search bar.
   - The system must provide categories for users to search by.
   - The homepage should display items that the user has recently viewed.
3. Secure checkout:
   - The system must ensure secure payment processing with credit cards.
   - Users should be able to pay using PayPal and other online payment methods.
4. Data encryption:
   - All user login information and data collected must be encrypted using the latest encryption standards.
   - The website must have an SSL certificate to ensure a secure connection without warnings on Google.
   - Users should have the option to save personal information like email, address, phone number, and credit card details securely.
5. Shopping cart:
   - The system must provide a shopping cart with a checkout feature.
   - Users should be able to add items to their cart.
   - Users should have the ability to delete items from their cart.
   - The system must allow users to save items for later.

5.b.

Product requirements:

1. Usability requirements: The user interface should adhere to the Web Content Accessibility Guidelines (WCAG) 2.2, ensuring accessibility for users with disabilities.

2. Efficiency requirements:

    · Performance requirements: The website should load pages and process transactions in under 2 seconds for 95% of the user interactions.

    · Space requirements: The data center hosting the system should have sufficient rack spaces for current and future server equipment, with clear labeling and organization.

3. Dependability requirements: The system should have a maximum scheduled downtime of 4 hours per month for maintenance.

4. Security requirements: All user data must be encrypted with industry-standard encryption algorithms, including TLS 1.3 for secure data transfer. Security audits and penetration testing will be conducted regularly to identify and mitigate vulnerabilities.

Organizational requirements:

1. Environmental requirements: The data center and server infrastructure should be designed for energy efficiency, including the use of energy-efficient cooling systems. Prioritize data centers that have received Energy Star certification.

2. Operational requirements: The system should be available 24/7 with a maximum of 0.1% unscheduled downtime per year. An incident response team will be in place to handle unscheduled downtime promptly.

3. Development requirements: The development team will follow an agile methodology with bi-weekly sprints, ensuring regular testing and incremental feature development.

External requirements:

1. Regulatory requirements: The system must comply with CCPA for user data privacy and consent. Legal counsel will be consulted to ensure ongoing compliance with evolving regulations.

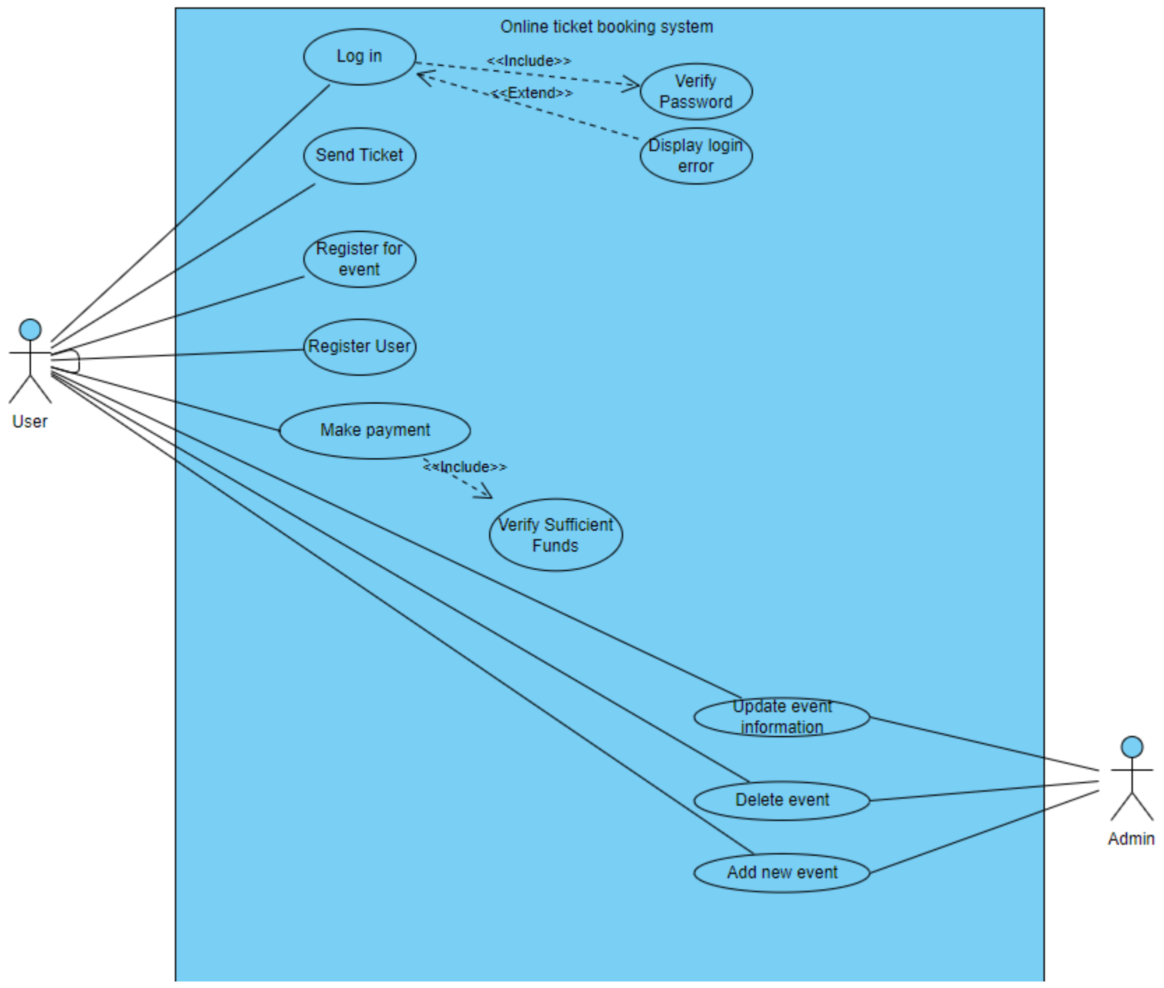
2. Ethical requirements: Content moderation will be in place to prevent the promotion of harmful, offensive, or fraudulent content on the platform. An AI-driven content moderation system will be employed to assist in real-time content filtering.

3. Legislative requirements: Assuming that the product will be released in the US, the organization will comply with US legislative requirements, including tax laws, consumer protection laws, and anti-discriminatory regulations.

- Accounting requirements: The system will maintain detailed financial transaction logs for auditing and accounting purposes.

- Safety/security requirements: Access to production servers will be restricted to authorized personnel and a robust access control system will be in place to maintain security.

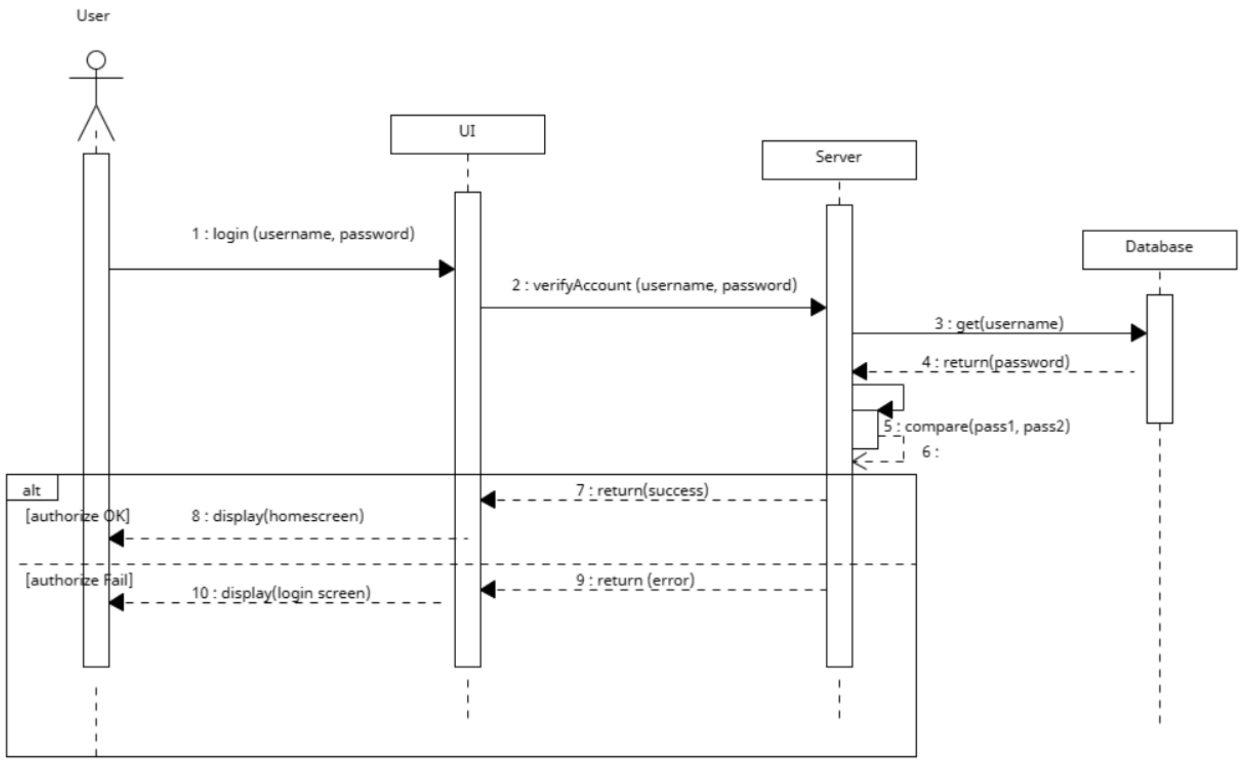
6. Use Case Diagram

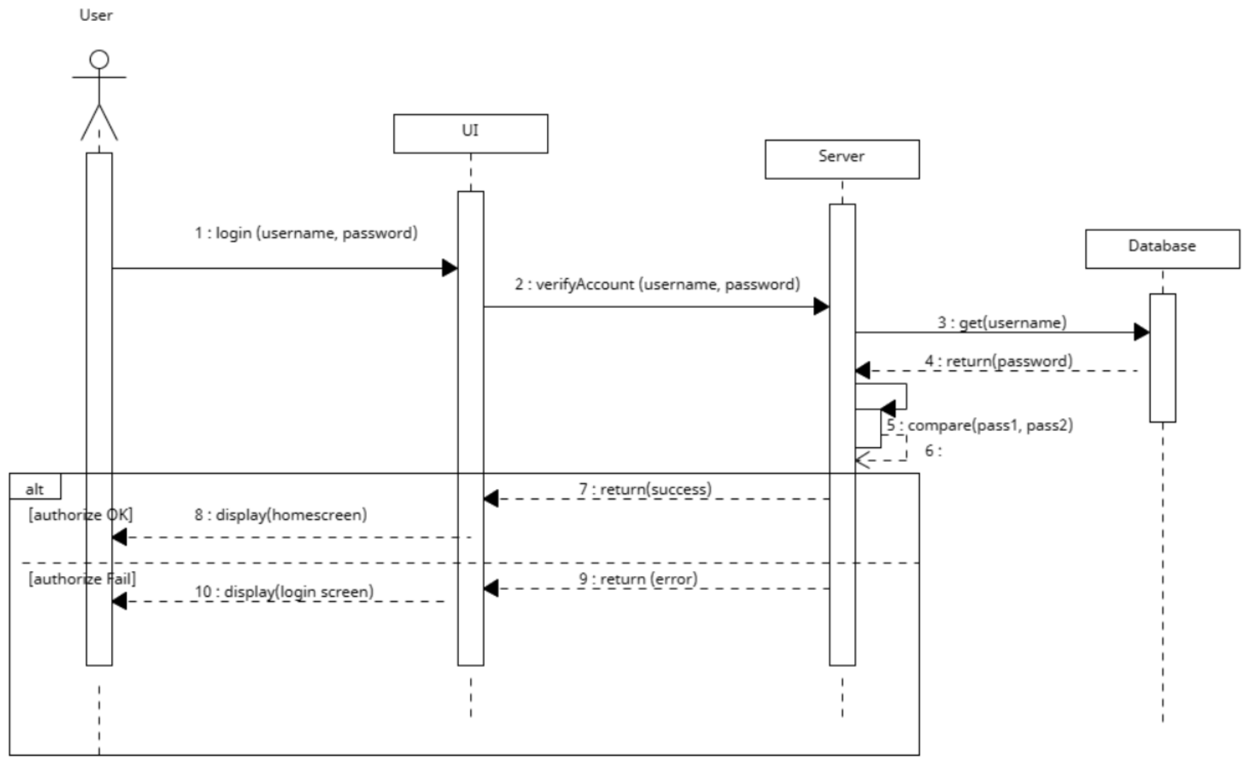


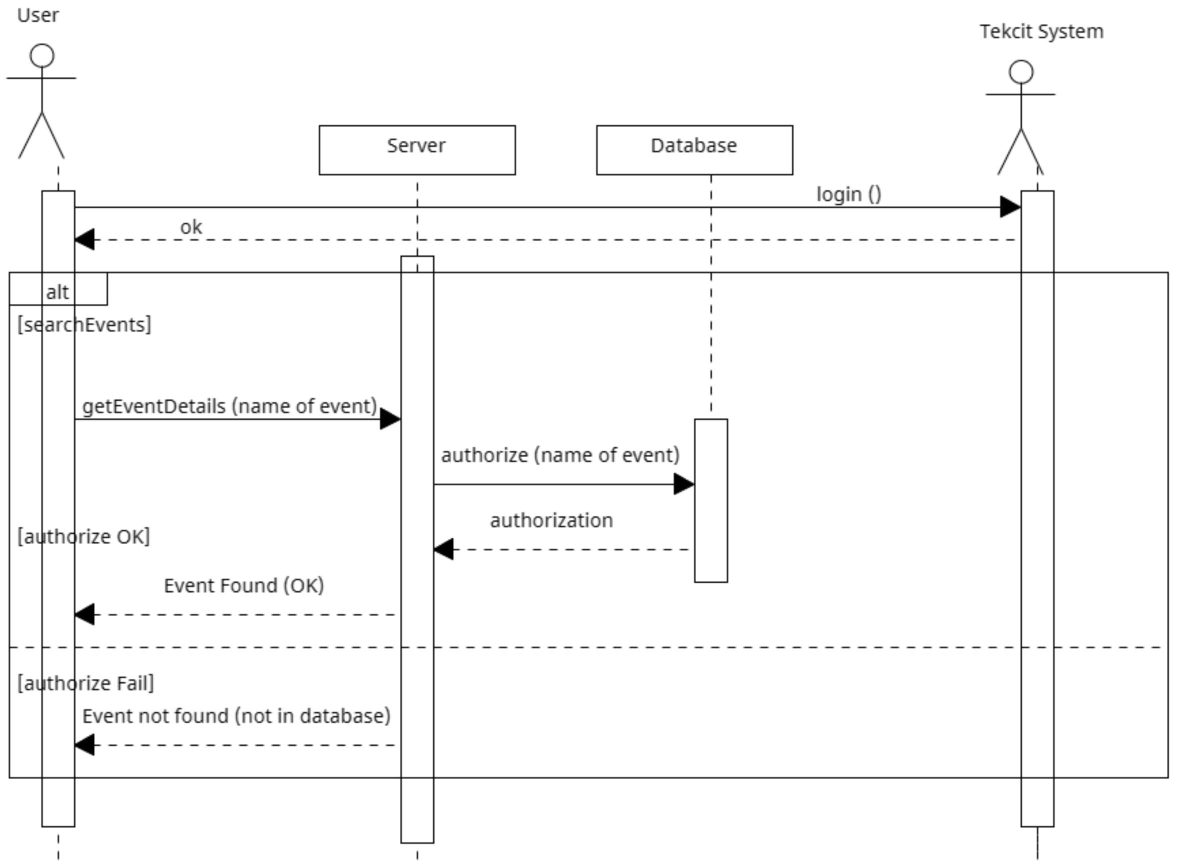
7. Sequence Diagram

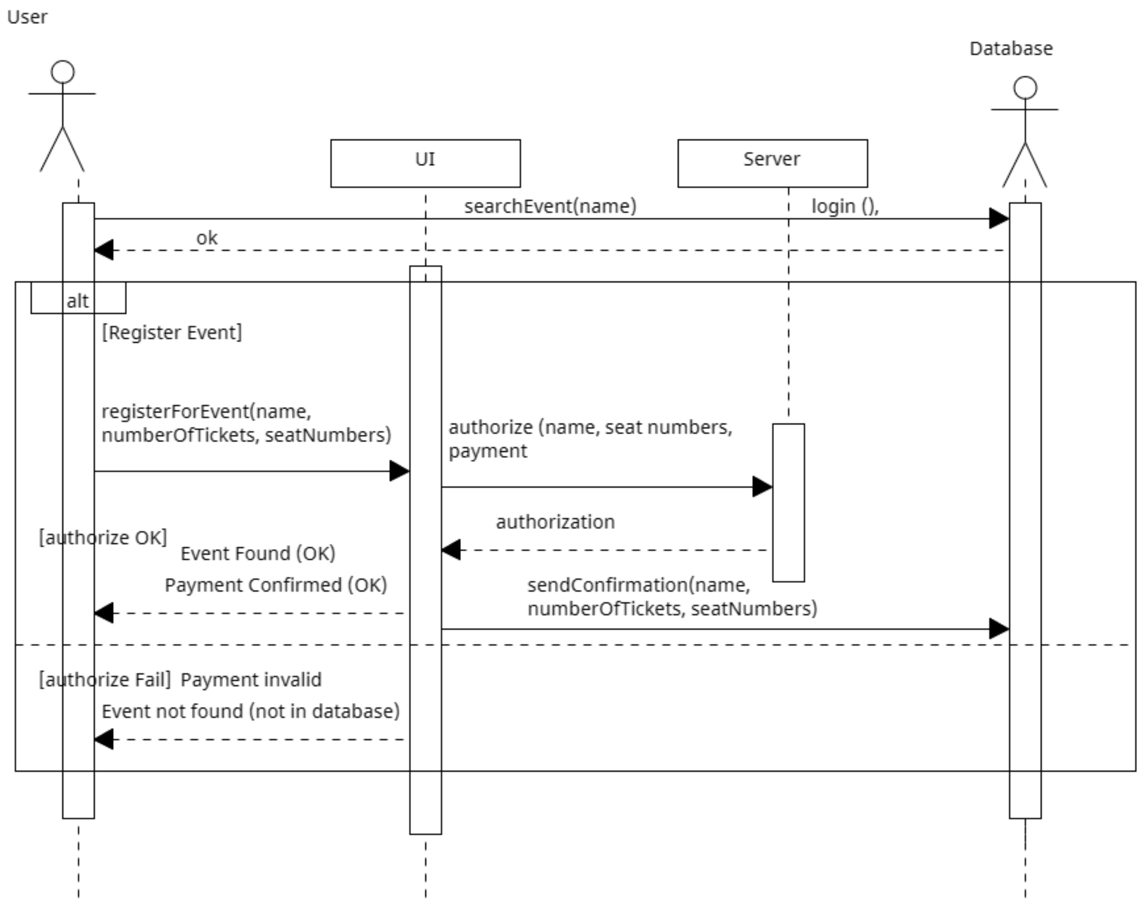
Sequence Diagrams are displayed in the following order from left to right:

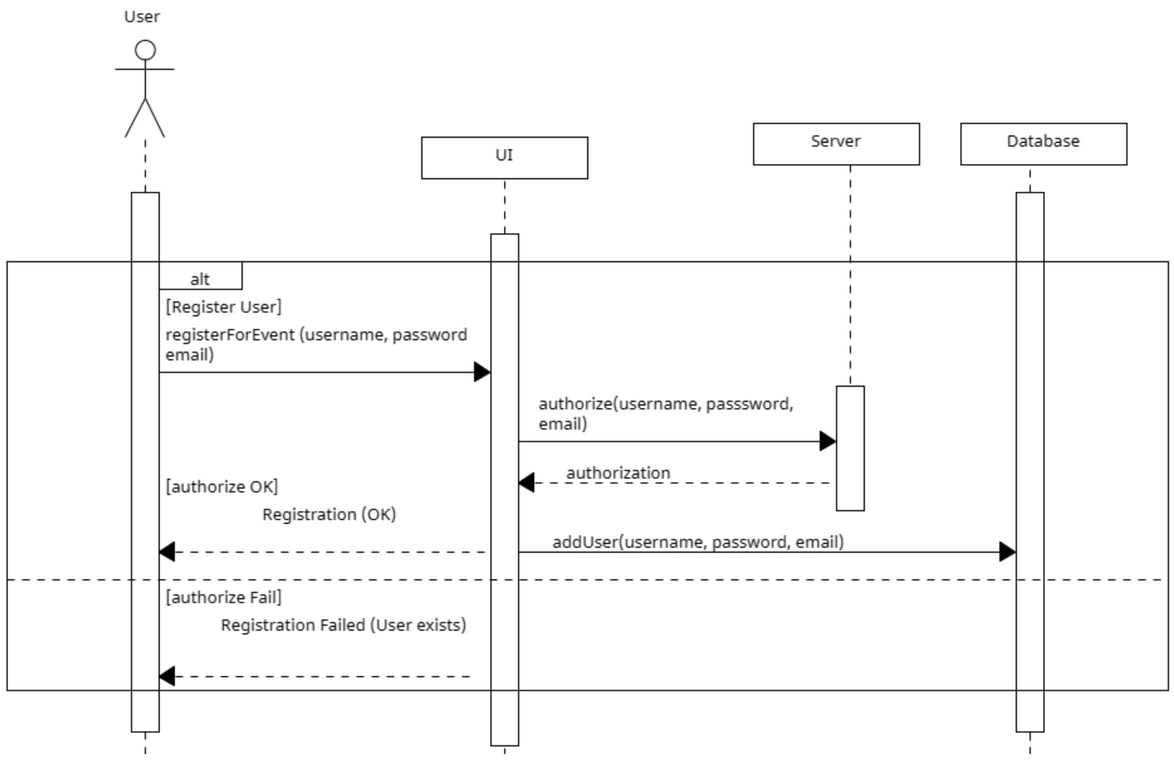
- Log In, Search Events, Register for an Event, Enter Personal details, Send Ticket, update event, add new event, delete event

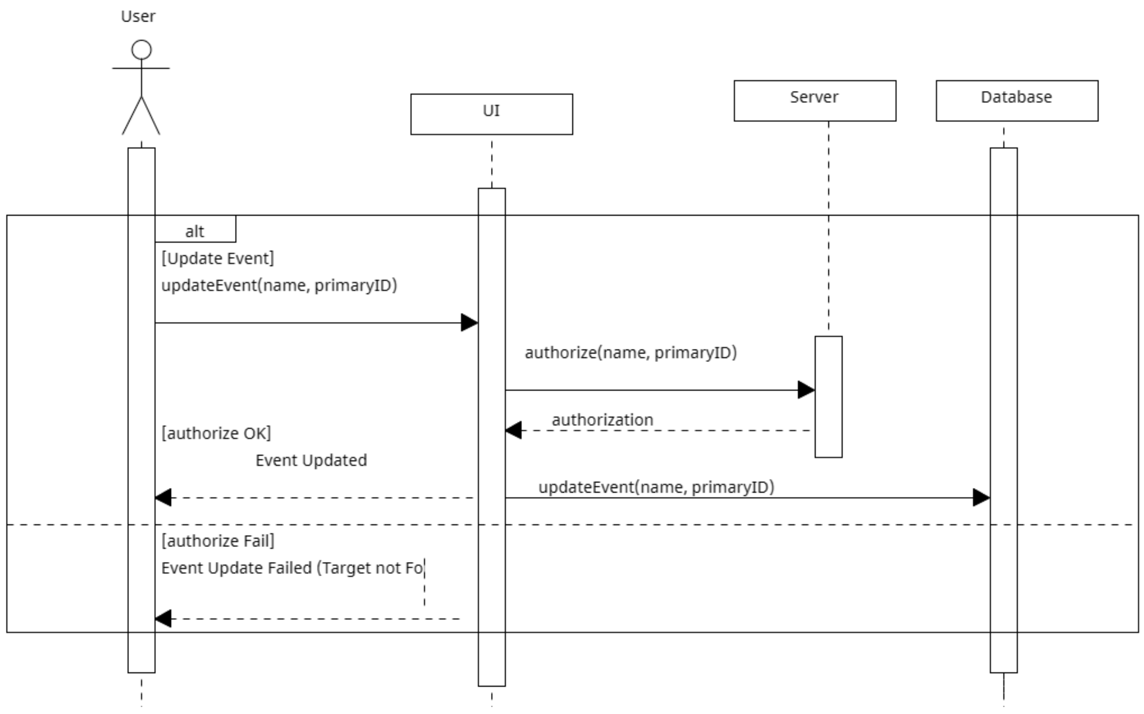


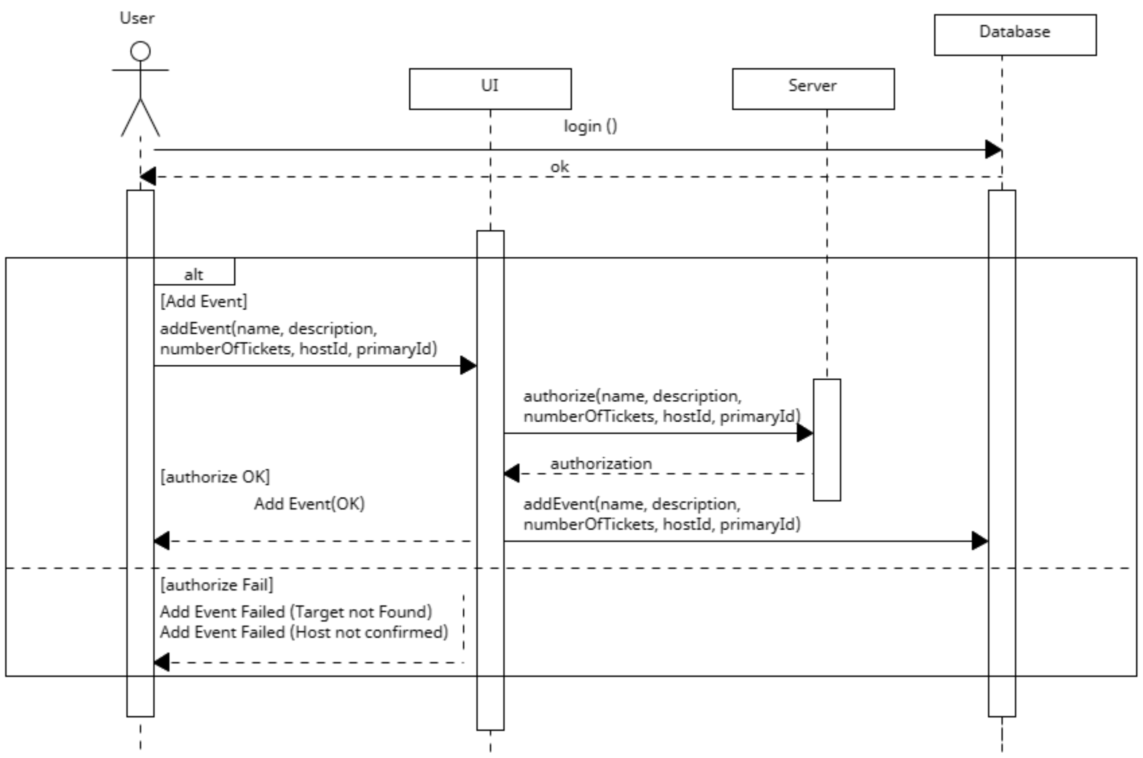
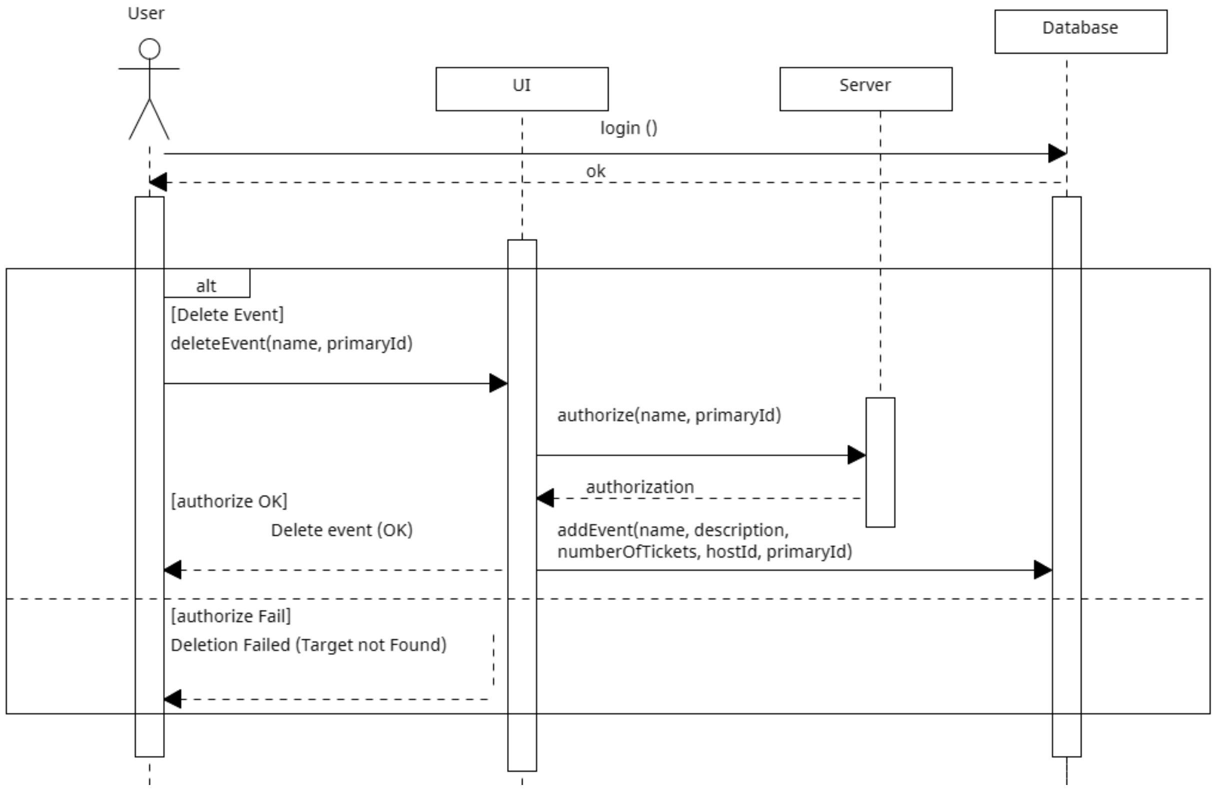

User
UI
Server
Database
1 : login (username, password)
2 : verifyAccount (username, password)
3 : get(username)
4 : return(password)
5 : compare(pass1, pass2)
6 :
alt
[authorize OK]
7 : return(success)
8 : display(homescreen)
[authorize Fail]
9 : return (error)
10 : display(login screen)


User
Tekcit System
Server
Database
login ()
ok
alt
[searchEvents]
getEventDetails (name of event)
authorize (name of event)
authorization
[authorize OK]
Event Found (OK)
[authorize Fail]
Event not found (not in database)

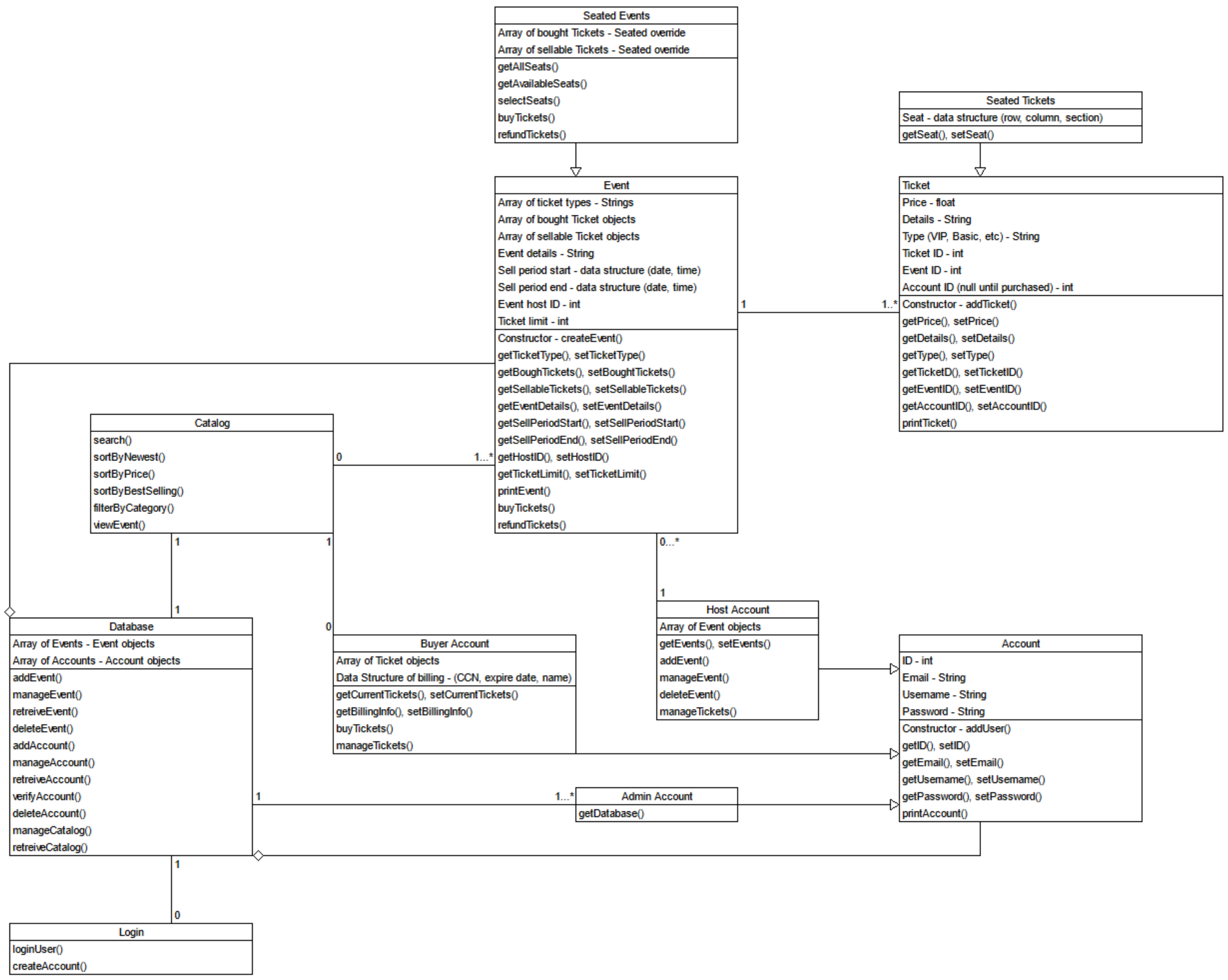

User
Database
UI
Server
searchEvent(name)
login (),
ok
alt
[Register Event]
registerForEvent(name, numberOfTickets, seatNumbers)
authorize (name, seat numbers, payment
authorization
[authorize OK]
Event Found (OK)
Payment Confirmed (OK)
sendConfirmation(name, numberOfTickets, seatNumbers)
[authorize Fail]
Payment invalid
Event not found (not in database)


User
UI
Server
Database
alt
[Register User]
registerForEvent (username, password
email)
authorize(username, passsword,
email)
authorization
[authorize OK]
Registration (OK)
addUser(username, password, email)
[authorize Fail]
Registration Failed (User exists)


User
UI
Server
Database
alt
[Update Event]
updateEvent(name, primaryID)
authorize(name, primaryID)
authorization
[authorize OK]
Event Updated
updateEvent(name, primaryID)
[authorize Fail]
Event Update Failed (Target not Fo


User
UI
Server
Database
login ()
ok
alt
[Add Event]
addEvent(name, description, numberOfTickets, hostId, primaryId)
authorize(name, description, numberOfTickets, hostId, primaryId)
authorization
[authorize OK]
Add Event(OK)
addEvent(name, description, numberOfTickets, hostId, primaryId)
[authorize Fail]
Add Event Failed (Target not Found)
Add Event Failed (Host not confirmed)

User
UI
Server
Database
login ()
ok
alt
[Delete Event]
deleteEvent(name, primaryId)
authorize(name, primaryId)
authorization
[authorize OK]
Delete event (OK)
addEvent(name, description, numberOfTickets, hostId, primaryId)
[authorize Fail]
Deletion Failed (Target not Found)

8. Class diagram


Seated Events
Array of bought Tickets - Seated override
Array of sellable Tickets - Seated override
getAllSeats()
getAvailableSeats()
selectSeats()
buyTickets()
refundTickets()
Seated Tickets
Seat - data structure (row, column, section)
getSeat(), setSeat()
Event
Array of ticket types - Strings
Array of bought Ticket objects
Array of sellable Ticket objects
Event details - String
Sell period start - data structure (date, time)
Sell period end - data structure (date, time)
Event host ID - int
Ticket limit - int
Constructor - createEvent()
getTicketType(), setTicketType()
getBoughTickets(), setBoughtTickets()
getSellableTickets(), setSellableTickets()
getEventDetails(), setEventDetails()
getSellPeriodStart(), setSellPeriodStart()
getSellPeriodEnd(), setSellPeriodEnd()
getHostID(), setHostID()
getTicketLimit(), setTicketLimit()
printEvent()
buyTickets()
refundTickets()
Ticket
Price - float
Details - String
Type (VIP, Basic, etc) - String
Ticket ID - int
Event ID - int
Account ID (null until purchased) - int
Constructor - addTicket()
getPrice(), setPrice()
getDetails(), setDetails()
getType(), setType()
getTicketD(), setTicketID()
getEventID(), setEventID()
getAccountID(), setAccountID()
printTicket()
1
1..*
Catalog
search()
sortByNewest()
sortByPrice()
sortByBestSelling()
filterByCategory()
viewEvent()
0
1...*
1
1
0...*
1
1
0
Database
Array of Events - Event objects
Array of Accounts - Account objects
addEvent()
manageEvent()
retreiveEvent()
deleteEvent()
addAccount()
manageAccount()
retreiveAccount()
verifyAccount()
deleteAccount()
manageCatalog()
retreiveCatalog()
Buyer Account
Array of Ticket objects
Data Structure of billing - (CCN, expire date, name)
getCurrentTickets(), setCurrentTickets()
getBillingInfo(), setBillingInfo()
buyTickets()
manageTickets()
Host Account
Array of Event objects
getEvents(), setEvents()
addEvent()
manageEvent()
deleteEvent()
manageTickets()
Account
ID - int
Email - String
Username - String
Password - String
Constructor - addUser()
getID(), setID()
getEmail(), setEmail()
getUsername(), setUsername()
getPassword(), setPassword()
printAccount()
1
1...*
Admin Account
getDatabase()
1
0
Login
loginUser()
createAccount()

Note: the account class splits users into the host and buyer for programmer clarity.
Tabular data for the some classes within the class diagram for clarity:

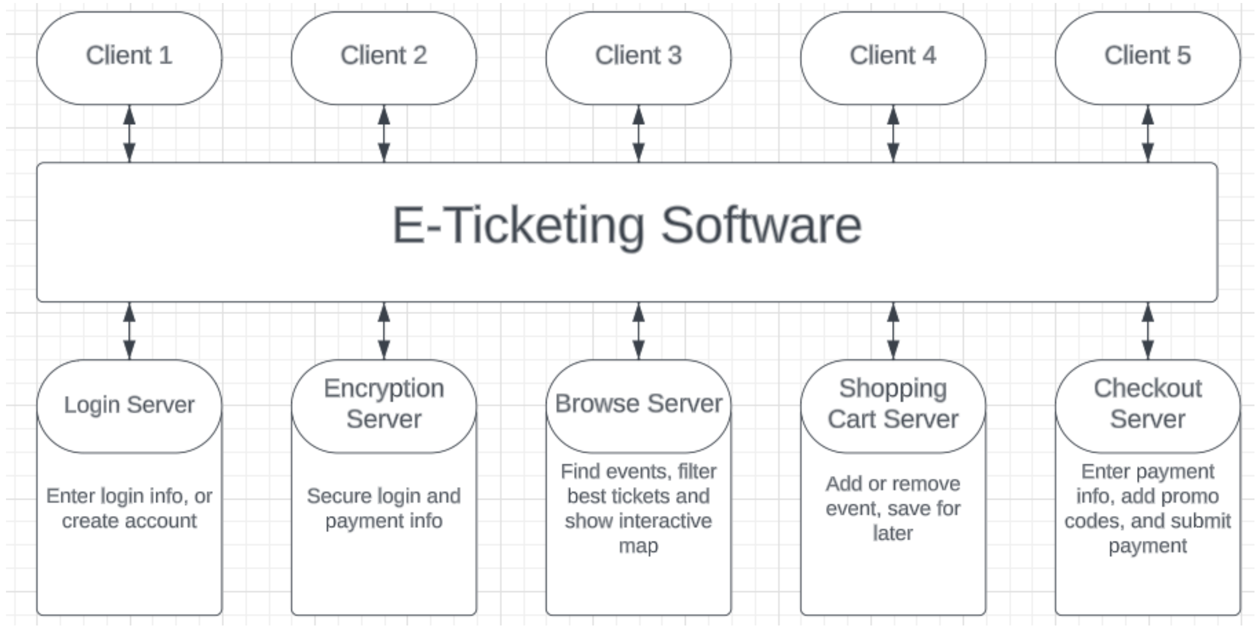
Catalog
The catalog is the class meant for the main page with lists of events to search through. It has access to the database so that it can return the array of events for display. There should only be one database to any one catalog. It needs access to the event class in order to effectively display these events. A catalog will know several events via the array, but the event does not need to know it is part of the catalog. The buyer account has access so they can view and use the catalog, given that buyers find events via the catalog.
search() - this is the search bar basically. Enter a string in order to get a list of events that are relevant
sortByNewest() - sorts the array by when the event was posted (descending or ascending). Returns the new array.
sortByPrice() - sorts the array by price (descending or ascending). Returns the new array.
sortByBestSelling() - sorts the array by a combination of recent ticket purchases and overall popularity. Returns the new array.
filterByCategory() - Reduces the array to only events that contain the given category. Returns the array.
viewEvent() - This is what will change the display from a list of events to a singular event. Will mean that more data will be accessed from a particular event rather than surface level information.

Database
The database is not the *actual* database. This class is the intermediary or safety lever for making sure that data is being abstracted/protected and information transfers must go through this class in order to access the database. Effectively aggregates the events and accounts, though the data of these classes will be held elsewhere for data security. The catalog, the admin account, and the login class all access the database. Each admin has access to the one instance of the database, databases will know of all current admins (must know at least one). In order to login, access to the database is necessary. And the catalog uses the database in order to help create the display
All the functions under the database are straightforward, but note the difference of “retreiveEvent()” and company that are notably different from a “getEvent()” function. This is to make clear that access to the database will require an encryption/decryption process through another system.

Account
Note that the account class is the parent class of the buyer, host, and admin. The parent account class is not to be used (and therefore is connected to nothing), the children classes determine what the account has access to.
Buyer - the buyer accesses the one and only catalog. The catalog gives the account access to purchaseable tickets for buying, as well as a personal attribute for tickets they already have.
Host - the host accesses the event class to create events, so direct access to event class is necessary. Every event has one host account.

Admin - admins directly access the database. This gives them the ability to manage everything.

9. For our E-Ticketing Software, we decided to use-client server architectural design pattern since it would be easier to split up each function of the application into a separate server which would be accessible to the clients. The first server that the client would use is the login server where they would be able to login using username/email and password. If they do not have an account then they will be able to create an account. Another server that is important for security reasons is the encryption server which is going to be utilizing the latest encryption software with SSL certificate ensuring that personal information such as name, phone number, email, address, password, and credit card info is secured. The shopping cart server is where clients will be able to add or save tickets that they potentially want to buy. They also have the function to delete tickets they decide not to buy or accidentally added. At last, the checkout server is where you would confirm your payment by adding your payment information, promo codes will provide discounts, and then submit payment which will lead to a confirmation.


Client 1
Client 2
Client 3
Client 4
Client 5
E-Ticketing Software
Login Server
Enter login info, or create account
Encryption Server
Secure login and payment info
Browse Server
Find events, filter best tickets and show interactive map
Shopping Cart Server
Add or remove event, save for later
Checkout Server
Enter payment info, add promo codes, and submit payment

3.1 Project Scheduling

- The basic tasks that need to be set up include the five servers, the client connections to the servers, the classes (mentioned above), the operations that the user needs to go through (use case diagrams), and the entire frontend website design.
- The separation of tasks are client/server, frontend, design, and the rest of the team as a backend/dev.
    - There are five servers and their client connections that need to be set up. If a person sets up one server a day, they should be done within one work week(Mon-Fri).
    - The website design should be done within a week and can be handled alone.
    - Creating the website is a complex task and it must occur after the design has taken place. So it must start after the week of design. It will probably take two people two weeks to develop the website.

    - The rest of the tasks (classes and operations) make up most of the backend development, which will likely take two people 2-3 weeks.
- Given an 8-hour workday and a 4 person team, it will take 3 weeks, not including weekends. So starting April 1st, and ending April 19th.
    - Client/Server - April 1-April 5 - 1 person
    - Website Design - April 1 - April 5 - 1 person
    - Website Creation - April 8 - April 19 - 2 people
    - Backend Development - April 1 - April 12 - 2 people
    - Contingency week - April 12 - April 19

3.2 Cost Effort and Pricing Estimation

1)

	Function Category	Count	Simple	Average	Complex	Count x Complexity
1	Number of user input	10	**3**	4	6	30
2	Number of user output	3	4	**5**	7	15
3	Number of user queries	6	3	**4**	6	24
4	Number of data files and relational tables	14	7	**10**	15	140
5	Number of external interfaces	0	5	**7**	10	0
					GFP	209

1. Login, Signup, Create Ticket, Edit Ticket, Delete/Refund Ticket, Create Event, Delete Event, Buy Ticket, Choose Seats, Payment Info
2. Receipt, Successfully Edit, Display Current Page
3. Lookup by Tickets, Lookup by Event, Lookup by artist, Look up Date & Time of event, Look up Location
4. Seated Events, Event, Catalog, Seated Tickets, Ticket, Account, Host Account, Admin Account, Buyer Account, Login, Database (data files, username, password, id's)
5. Not needed

2. Determine complexity.

1) 4
2) 4
3) 5
4) 4
5) 3
6) 5
7) 5
8) 4
9) 2
10) 2
11) 1
12) 0
13) 0
14) 5

3. Compute gross function point (GFP).

- GFP = (10*3) + (3*5) + (6*4) + (14*10) + (7 * 0) = 209FP

4. Determine processing complexity (PC).

- E = FP / productivity = 209 / 60 = 3.48 ≈ 4 person - weeks

5. Compute processing complexity adjustment (PCA).

- PCA = 0.65 + 0.01 x ((4 * 4) + (4 * 5) + (2 * 0) + (1 * 3) + (1 * 1) + (2 * 2)) = 0.65 + 0.01 x 44 = 1.09

6. Compute function point (FP) using the formula: FP = GFP × PCA

- FP = 209* 1.09 = 227.81 FP

3.3 Estimated cost of hardware products

Amazon CloudFront

- a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds
- serves as front end infrastructure
- assuming:
  - Data transfer out to internet: 5,000 GB per month
  - Data transfer out to origin: 25 GB per month
  - Number of requests (HTTPS): 350,000

- Monthly cost: $425.85

Auth0

- a flexible solution to add authentication and authorization services to our applications
- serves as Encryption Server
- assuming:
  - ~20,000 monthly active users
- Monthly cost: $1,400.00

Amazon EC2

- instance types comprised of varying combinations of CPU, memory, storage, and networking capacity
- serves as back end infrastructure
- assuming:
  - Tenancy: Shared Instances
  - Operating System: Linux
  - Baseline: 2 instances
  - Peak: 5 instances
  - Instance: t4g.2xlarge (8 vCPUs, 32 GiB Memory, Up to 5 Gigabit Network Performance)
- used for Login Server, Browse Server, Shopping Cart Server, Checkout Server
- Monthly cost: $490.56

Total Monthly cost: $425.85 + $1,400.00 + 4 × $490.56 = $3,788.09

Total Yearly cost: 12 × $3,788.09 = $45,457.08

3.4 Estimated cost of software products

- cost of authentication server already included in calculation of estimated cost of hardware products
- suggested retail price for a standard Windows Server 2022 is $1069
- Linux OS is usually free of charge - there are no license fees
- Using our own code/software means that there are no additional software license fees.
- Thus, using Linux servers: cost of software products = 4 servers × $0 per server = $0.00
- Total Monthly cost: $0.00
- Total Yearly cost: $0.00

3.5 Estimated Cost of Personnel

- From calculations in 3.1 and 3.2, four people is the estimated number of team members to develop this project. Since there are no custom or special technologies used, very minimal training is required if any. The employees should be hired if they have the skills and experience needed thus removing the need for training.

4. A test plan for the software:

For one unit of our test, we decided that we would go with testing the register new user unit. Registering a new user is involved with the database, since any registered user has to be protected, utilizing a hash map for the users, and a string array for the passwords. It is important to utilize a hash map for users as it is often going to be accessed while the database runs, and constant time accesses are vital to ensure that the server runs at a proper speed. In order to do so we provided 4 main test cases that are set out to prove a few of the possible outcomes that could happen while a user is attempting to register a new account.

1. The first test case is if the user is valid. If the user is valid, the system should automatically register the user. Valid usernames would typically run through a slew of tests including a minimum length, avoiding special characters (such as spaces), and not being the same as the password. Here we test the length of the username. In a more nuanced set of code, we would ensure that the username length is within a set bound. Some websites go from about 5-20 characters.
2. The second test case is another requirement needed for usernames, such that they should all be unique. First we test if the user's username is already taken, by quickly attempting to access such a user. If it is, it displays a message that the username is already taken. This is a vital step for any user registration, as those who are attempting to create a new account need to know why their attempted registration isn't working.
3. The third test case is checking to see if the username or password field is empty, if either are empty, then this should return an error "Username or password field is empty". This one is clear cut: a username or password without any characters, while theoretically doable, is a major security issue and must be accounted for.
4. The fourth and final test case is for security, if the username or password is below three characters, return an error saying "the username or password must be greater than three characters". This test case gets us closer to the realistic expectations one might have for all the requirements that are involved in having an allowable username and password. We make sure that both the username and password are within the minimum length before we register them.

Our test case code is attached in the document.

5 Comparison of your work with similar designs

Tickermaster challenges (November 17, 2022):

- Taylor Swift pre-sale led to a surge in user, causing Ticketmaster sit crashes.
- Widespread discontent among fans prompted inquiry bu Tennesse Attorney General Jonathon Skrmettian [1]

Tekcit's fundamental approach:

- Preserves core ticketing features
- Directly addresses challenges faced by Ticketmaster.

Tekcit's scalability:

- Purposefully designed to support multiple users concurrently.
- Ensures optimal performance and speed even during peak demand periods.

Dynamic queuing system:

- Addresses stress and equity concerns during high-demand events.
- Prioritizes user experience by reducing traffic and potential server overloads.

User centric customer support:

- Recognizes the importance of responsive customer assistance.
- Support staff equipped with necessary tools for timely and courteous issue resolution.

Anti-scalping measures:

- Actively combats ticket scalping, a common issue faced by Ticketmaster users.
- Implements strict anti-scalping methods to ensure fair-pricing for tickets.

6. Conclusion

Overall, our team was successfully able to implement our software design for “Teckit”. We incorporated certain aspects of the software from existing software such as TicketMaster but devised a plan to optimize certain functionalities, especially performance under heavy loads of usage. To reduce complexity we decided to utilize the client-server architectural design system since it helped us separate functions into different servers and get a better understanding of what each function was going to accomplish. We also have provided sequence and use-case diagrams on how the software would work during a demo run. Each sequence case diagram depicts a certain scenario that the user would run across while utilizing the app such as login/creating an account, getting details about an event, and adding/deleting an event. Our class diagrams further broke down the classes, methods, and attributes that would be needed to store data and operate. We did not have to make any major changes to our software design since we did thorough research and have a well-detailed diagrams and logistics that need to be taken into consideration. In general, following the software engineering design process helped tremendously since it helped reduce complexity by breaking down the process into smaller tasks and also give a well-rounded review on the constraints that come with designing.

7. References

[1] M. Cerullo, “Taylor Swift fans’ complaints over Ticketmaster site prompts investigation,” CBS News,
https://www.cbsnews.com/news/taylor-swift-concert-ticketmaster-fans-angered-after-site-crashes-tennessee-ag-investigating/ (accessed Nov. 14, 2023).

8. Presentation Slides: Linked with the document

CS 3345 Final Project

10. GitHub requirement:

https://github.com/DemarcusBraclet/3354-Tekcit