CSCI 4448/5448
Object Oriented Analysis and Design
Final Report

EvenTrade

Developers

Marie Hargan
Lukas Zumwalt

Final State of System Statement

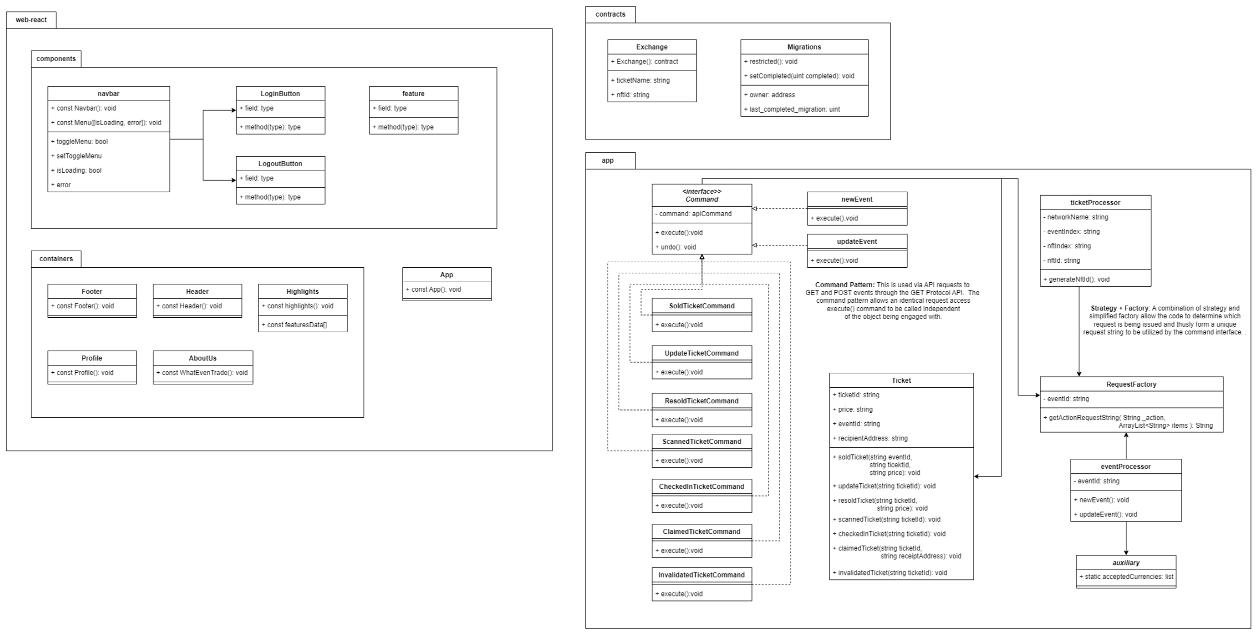
The final state of our coded system demonstrates front-end user interfaces where an individual may interact with our preliminary services, account generation and access, and investigate the basics of what we would offer as an organization. The final implementation of a streamlined smart ticket non-fungible token minting service and online exchange forum was limited, however, primarily due to availability of smart contract deployment API's.

The technology required to pull off the task of seamless smart contract deployment is not able to be written within 4 weeks, and the API we required to utilize this technology was not acquired in time. This was an oversight in project design but allowed for a lot of creative investigation into the community that is building Web 3.0 and modern blockchain technology. The biggest change that occurred in the final code implementation was the integration of Auth0 to create a user authentication system with the web application.

Connection between the users' front end interface and the back end decentralized application did not come to fruition in this project - partially due to lack of development resources for the primary smart contract API, and partially due to real-world time constraints.

Final Class Diagram and Comparison Statement

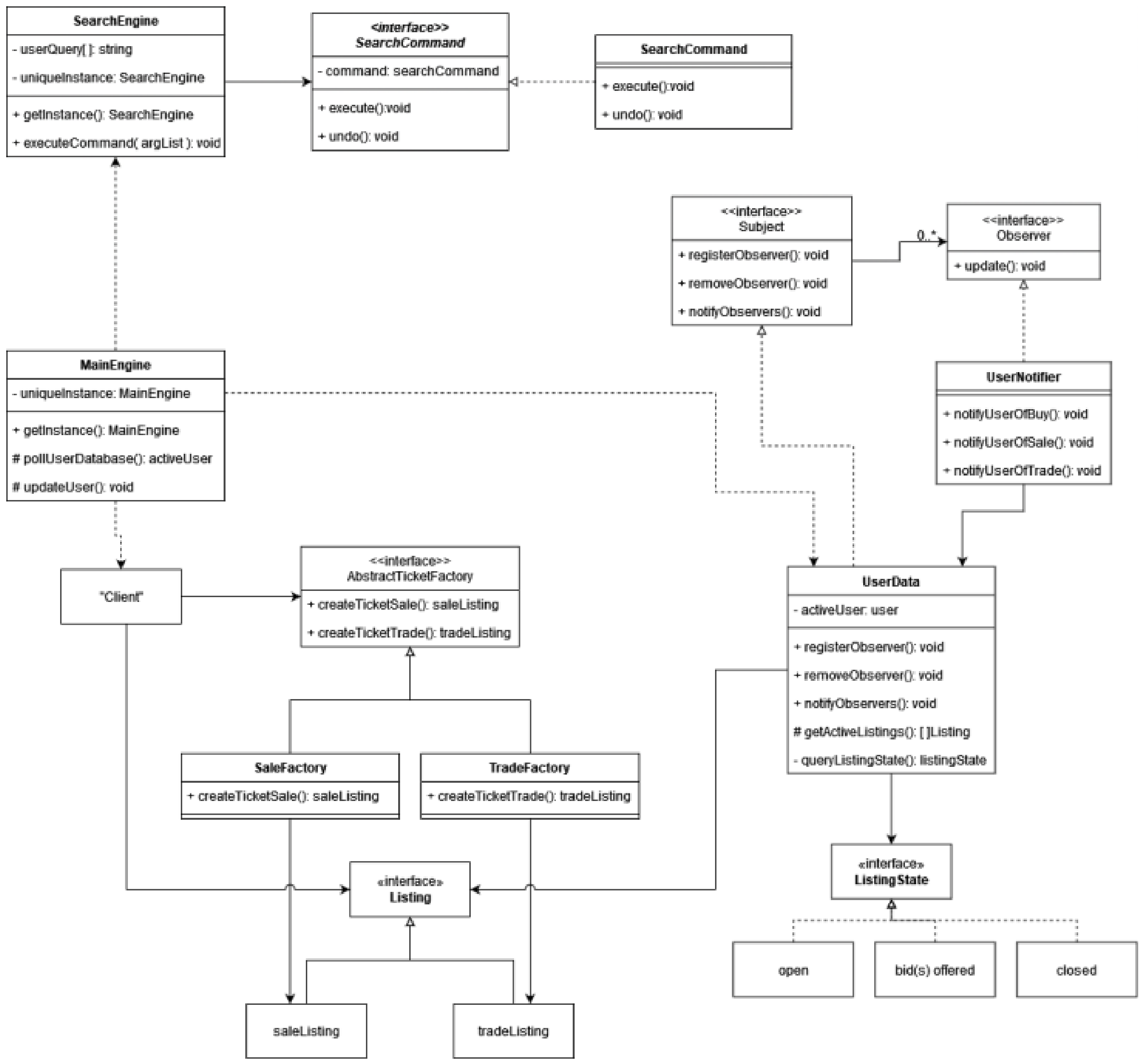
Project 7 Final UML Class Diagram:



Command Pattern: This is used via API requests to GET and POST events through the GET Protocol API. The command pattern allows an identical request access execute() command to be called independent of the object being engaged with.

Strategy + Factory: A combination of strategy and simplified factory allow the code to determine which request is being issued and thusly form a unique request string to be utilized by the command interface. .

Project 5 UML Class Diagram:


SearchEngine
- userQuery[ ]: string
- uniqueInstance: SearchEngine
+ getInstance(): SearchEngine
+ executeCommand(argList): void
<<interface>>
SearchCommand
- command: searchCommand
+ execute():void
+ undo(): void
SearchCommand
+ execute():void
+ undo(): void
<<interface>>
Subject
+ registerObserver(): void
+ removeObserver(): void
+ notifyObservers(): void
0..*
<<interface>>
Observer
+ update(): void
MainEngine
- uniqueInstance: MainEngine
+ getInstance(): MainEngine
# pollUserDatabase(): activeUser
# updateUser(): void
UserNotifier
+ notifyUserOfBuy(): void
+ notifyUserOfSale(): void
+ notifyUserOfTrade(): void
"Client"
<<interface>>
AbstractTicketFactory
+ createTicketSale(): saleListing
+ createTicketTrade(): tradeListing
UserData
- activeUser: user
+ registerObserver(): void
+ removeObserver(): void
+ notifyObservers(): void
# getActiveListings(): [ ]Listing
- queryListingState(): listingState
SaleFactory
+ createTicketSale(): saleListing
TradeFactory
+ createTicketTrade(): tradeListing
«interface»
Listing
«interface»
ListingState
open
bid(s) offered
closed
saleListing
tradeListing

Comparison Statement:

The design prepared for initial project conceptualization evolved in many ways over time, turning into what it is today. A means of delineating effort was to split development into frontend and backend work sectors. As a result, they developed independently, and utilized isolated testing and debugging environments while understanding the fundamental technology and building upon it.

A notable difference between the initial UML and the current one is that we are not utilizing an abstract factory to "generate listings". Rather, a rudimentary simple factory is being used for generating request strings necessary for API POST requests. We are also not incorporating an observer-pattern notification system for users, as since we could not implement NFT minting there was no platform to build media ownership nor list and mediate exchanges.

We are still utilizing a command pattern in order to orchestrate API requests in our system. A real-world constraint to our design, the big blocker, was that our primary protocol GET for using smart contract ETH deployment required a privately-accessed key - which we did not receive by the time of writing this report. This company is based in the Netherlands, and the technology is so new we needed to have firsthand communication with their development engineers to be truly productive. This timeline interference is a side effect of changing our design later in development as we did not properly budget time for integrating a novel project of this scale.

The concepts presented in the original diagram are still valuable and worth following as guidance, in our opinion, as they lay out the fundamental elements required in our ideal system.

Third-Part Code vs OG Code Statement

Frontend

https://www.youtube.com/watch?v=LMagNcngvcU
https://www.youtube.com/watch?v=wr3VmbZdVA4
https://auth0.com/

The frontend code was developed using the framework found in the first youtube tutorial link. Most of the design was from following that video. However, the extension of that framework is original work, such as found in the Profile.js and profile.css files. Additionally, how the app is built and developed in App.js and Index.js was mostly original work with some serious roadblocks that were solved by following the second linked youtube tutorial. Finally, Auth0 was the major third-party extension that was used for our user login authentication. This is not commented on in the code since the library is clearly imported and used in App.js. A lot of the third-party code and resources were adapted to suit the needs of the application. React JS was a new language for me and given the timeframe of the project, I followed closely with these tutorials in order to efficiently and effectively achieve the targets of this project.

Backend

How to Build Ethereum Dapp (Decentralized Application Development Tutorial)
Ganache - Truffle Suite
GET Protocol

The backend application code was developed ad-hoc in an attempt to integrate external smart contract APIs. The majority of the back end app code was written in-house, though, in Java. The primary workhorse for deploying smart contracts representative of dynamic events, as well as the keystone functionality of minting smart ticket NFTs, could not be written by us two developers in four weeks. This code is third-party, developed by the engineers at GET Protocol based in the Netherlands. For testing smart contracts and their deployment for events - which NFT tickets are “tied” to in their own existence, we utilized Ganache, the third party blockchain deployment and development tool. It provides developers with unexchangeable Ethereum tokens to prototype deployment without having to pay for it, which was very helpful in initial sandbox stages.

Statement on OOAD Process for the Overall Project

One of the most helpful design methods we opted for was to engage real-time tracking of tasks in a Kanban board we established through Atlassian software. This is something we carried over from our employment, and it was surprisingly helpful to support tracking of effort, as we were familiar with the UI and the benefits it can bring to a development team. This benefits us in showing project development and representation of tasks being completed over time which boosted confidence and showed a graph of work completing and what needed to be done.

We lacked proper scope in the initial project design pitch, as a fundamental part of our initial design - ticket information loaded *directly* into on-chain smart contracts - held a major hole in security as ticket information would then be publicly available to anyone with the aptitude to inspect the blockchain. This was resolved in concept with NFTs, as they may represent a *public* media ledger while *protecting ownership* of said item. This crept very quickly into a bleeding edge project, as the only real solution we could find is an organization called GET Protocol which is very new and not very mature in solutions rollout.

We developed a rapid dependency on this organization and had to get all information from their development team. With the company being based in the Netherlands, this stifled progress immensely and with us being a student project (not generating direct revenue) it is assumed we took lower priority from their support engineers versus project-generating organizations also implementing their API. The primary lesson here is to pull in diverse stakeholders or poll connections in the space new to you in order to build context, and ensure communication is opened early and driven often, *especially* in novel and bleeding edge projects.

As software engineers, we were also not familiar with the mechanisms of front end code development for web applications. As such, we were conceptualizing ways to implement OO patterns, primarily observer and command patterns, *intrinsic* to the front end codebase through user login, authentication, and account management. After working through it, however, it was quickly revealed that front end code is not object oriented, and therefore did not allow these patterns to manifest in ways we are familiar with through this course. We did not want to opt to shoehorn in patterns without a real justification behind it, either, so some of our final project was not as recognizable of an object-oriented product as we initially dreamed.