Systems Capstone: Project Proposal

Sameer Dandekar, Nathan Kennedy, Ankita Khera, Rachel Kitchen, Niles Rogoff

Abstract: The Problem

Students are not rewarded for attending more VT sports events throughout the year, especially through the lottery system. For popular events, tickets sell out and then students resell their ticket for hundreds of dollars beyond market value, unregulated by the university since students can print tickets and send them to others. The current student ticket system requires a VT login, and gives students one of three options: 1) buy a season pass, 2) buy tickets per game at full sale price, or 3) enter the student lottery.

Abstract: The Solution

We intend to create a blockchain-based system to replace the VT student sports ticket system with a focus on the student lottery and football tickets. The EOSIO blockchain will be used to validate, store, and manage various transactions, which will be based around an EOSIO token called a “Hokie Token” with the symbol “HTK”. These transactions and actions on the blockchain will include ticket exchanges, lottery winners, and auction winners, and involvement in games with students receiving tokens for attendance. Each user will have their own EOSIO account on the blockchain corresponding to their username, and they will have the option to manage it through our application or manage it manually. As a result, students will benefit from a fair distribution system and will enjoy the ticketing and sporting event experience. Virginia Tech will benefit from subduing the ticket black market, allowing the school to fully regulate and profit from the sale of tickets, whether they be sold first-hand or second-hand. The broader community will benefit from greater enthusiasm resulting from incentivizing students who display exemplary school spirit.

Technical Description

For our minimum viable product, we will have a website front-end along with a login interface providing for unique usernames and passwords, as well as a portal for account information. These applications will provide four key transactional features -- buying and selling tickets to and from Virginia Tech, entering the ticket lottery, and receiving a ticket from the lottery. Administrators will create games and tickets, execute student lotteries, and be able to see a list of active tickets for a game. Each ticket, game, and user is stored on the EOSIO blockchain and managed through a smart contract called hokipoki. We will begin development by creating a

server to host our application, and setting up means of communicating with the EOSIO blockchain platform. After that, we will begin working in separate groups on the backend and frontend, and ultimately come back together to work on integrating the two for a seamless application. For the blockchain and back-end development we will be using a variety of tools. We will be using EOSIO for the blockchain/backend implementation, AWS for hosting the server, GitLab for version control, and Mako-Server for a web-server and templating engine. For the front-end design we will be using MarvelApp to create visual prototypes and React for the web-application framework. The user interface and experience development will start with workflows and design mock-ups before actual programming begins. The functions available on the website will be mapped out and ranked by importance. User experience with the current ticket system will be evaluated to identify potential improvements for our product's user experience. Successful ticket distribution platforms along with payment services will also be analyzed to find patterns in user experience.

Requirements

The MVP requirements are as follows:

Front-End Functionality

- Website front-end application
    - Login Interface
        - Allows for unique usernames and passwords
    - Portal containing account information

Application Functionality

- Buying and selling football tickets to and from Virginia Tech
- Entering the ticket lottery
- Withdrawing from the ticket lottery
- Receiving a ticket from the lottery

Blockchain Uses

- Usernames will correspond to users present on the EOSIO blockchain
- Tickets, games, and lottery entries will be stored on the EOSIO blockchain
    - They will be managed through a smart contract
- Hokie Tokens will be stored, transferred, and otherwise managed on the EOSIO blockchain, via the eosio.token contract.

Known Issues

Implementation and integration issues

The largest issues come from user management. The `hokipoki` smart contract needs to be able to transfer HTK on behalf of all users who want to buy or sell tickets. HTK is managed by

the `eosio.token` smart contract which requires `user@active` permission to transfer tokens. However, even with an explicit code grant on the contract, `hokipoki` can only execute inline actions to another contract with the permissions `hokipoki@eosio.code`. In order to buy or sell tickets, the user executing the buy or sell action must have explicitly granted permission to execute inline actions as `user@active` to the hokipoki contract, and that permission is stored on their user. This means that our application must manage every user's permissions explicitly, or at least set up this permission grant on account creation. Since all account creation in our MVP will be done through our application, this will not be a permanent problem, but it presents a management challenge to overcome.

User management is made more difficult by the fact that eosio only allows usernames of 1-13 characters, and uses only the characters [a-z0-5.], which are both restrictions that may not apply to student PIDs, such as the PID of one of our team members, "ankita99." For our proof-of-concept project, we chose not to address this problem. However, solutions could include coming up with an encoding form for PIDs with the digits 6-9 in them, assigning eosio usernames based on a 64-bit hash of a user's PID, or keeping a static map from pid to eosio username in a permanent data store on the server of the web application.

Integration between the blockchain smart contracts and web application may be the hardest part of the project. For example, the cleos APIs for javascript are very raw, do not support all flags that the command line interface or features that the C++ interface supports. The most major difference, from the point of view of application development, is how simple it is to select all football games on a specific day from the C++ interface, as we have a secondary index on that table. However, while the Javascript API allows us to select by that secondary index, it will select all games on that day and in the future, so we must keep requesting pages until we find one in the future, which is a series of network requests that we can't determine has terminated until we have finished the previous request, deserialized the response, and analyzed the contents.

Other integration challenges could arise depending on how we decide to move forward with the web front-end. If we choose to switch to React or Angular or another frontend framework that does not play nice with embedded server side code like php or mako templates, we will have to create a separate web api, perhaps even served by a different web service (php, or a custom written web service in a different language) and passed through from the front line http server. This introduces additional complexity, and would require two additional layers of wrappers over cleos (js client for making REST calls, and REST server over an existing wrapper over cleos). It would also reduce performance, for example, making an API call to get the balance for a particular user would incur an entire HTTP request which would have to be parsed twice on the server, and could not be made until the page has loaded, whereas a templating language such as php or mako could retrieve the balance while generating the page, and send a page with the balance already available to the requesting client.

Security Issues

In addition to implementation and integration issues, there are several security problems that could arise during the project.

Some security issues are inherent to using the EOSIO platform itself. For example, when you create an EOSIO server, the default key used for the system account (eosio) is a development key that corresponds to a publicly known private key. The only way to change this is to generate a new public and private key, insert them as the key provider in the local nodeos configuration file, and specify and pass a customized genesis json file. All of these steps must be done before nodeos is started for the first time, or the entire blockchain must be wiped and recreated from scratch. We had to take several attempts at this, but the issue has been addressed.

Of course, the system will be susceptible to human error. For example, an actor could trick an administrator into creating many accounts for them, then use their mass of accounts to rig auctions, or to purchase tickets or receive lottery tickets then resell them to other students for cash, or attempt to farm hokie tokens by checking in to events or performing other actions.

Additionally, each student has an account on the blockchain, created and managed by our application. The user has the option to upload a public key to use as their active key, which would allow them to authenticate directly by directing their local cleos client to connect to our nodeos server. Since they have active access to an account on the blockchain with no bandwidth or ram restrictions, they could deploy their own smart contract that stores an incredible amount of data in a table, and potentially perform a denial of service attack against the server by running it out of disk space or available memory. This could be negated by applying reasonable ram and network limits to each user on account creation, but we do not believe it will be a problem until such a system is deployed in the real world.

Finally, the private keys for all students who choose to use automatic key management are stored in the same wallet, which is left unlocked on the server so that the web application can buy and sell tickets and push actions on behalf of the user. If this wallet were to be compromised, not only could an attacker transfer HTK or buy or sell tickets on behalf of most users, they could redeploy the `hokipoki` smart contract with different code. It is important to make sure that this wallet is not compromised, and that arbitrary actions cannot be performed using the keys in the wallet via the web interface.

In a similar vein, the keosd wallet must be unlocked on server startup before any student with automatic key management can buy or sell tickets. This means that either a team member must manually log in and enter the password on server startup, or the password must be kept on the server. For this project, we have opted for the first solution, however it could be stored in a systemd service only readable by root that specifies commands for entering the password to unlock on boot.

Workflows

1. User opens the web application through a browser
2. Navigates to a login page and uses provided credentials
3. User can access unique pages
    a. Account information: displays account balance, transaction history, student information
    b. Buy: page to purchase ticket for upcoming game
    c. Sell: page to sell ticket back to school
    d. Lottery: allows user to enter ticket lottery
    e. My Tickets: shows user's active and tickets

(admins can create games & execute lotteries)

Resources

Hardware

1. Computers
2. Phones

Software

1. EOSIO
2. Linux
3. GitLab
4. Python
5. Mako-Server
6. ReactJS

Current Competition

There are many existing services available for the purpose of purchasing tickets. There are large third-party products such as StubHub and Ticketmaster, as well as ticket systems specific to a group, such as schools, teams, or bands.

UVA SHOTS System

There is an existing ticket system that has a points-based lottery, the SHOTS system at UVA. This system rewards students with points for attending sports events, and allocates lottery tickets based on point order. We will be making certain changes, such as QR code identification, and allowing students to buy/sell tickets with tokens. However, there is no direct competition between UVA tickets and VT tickets, so we are focusing more on the next category.

Third-Party Products (StubHub, Ticketmaster)

VT sports tickets are often resold on third-party websites for higher than face value. We aim to minimize this for student tickets by requiring ticket validation through the app, which is linked to student ID. Our software also creates a more fair system for the lottery, so that students can receive tickets from merit (attending more games) rather than random selection, leaving them to pay hundreds of dollars for big games.

Peer Contract

Each group member will attempt to the best of their ability to fulfill the duties and responsibilities listed in the Individual Expectations section. Group members will be generous with their time spent on the project and meet regularly as outlined in the Meeting Times section.

Individual Expectations

Individual Responsibilities

Each individual has specific expectations, but as the project develops roles may shift and adjust as necessary.

Sameer Dandekar:	Frontend UX design, mobile dev (iOS)
Nathan Kennedy:	Backend development / Frontend-Backend integration
Ankita Khera:	Backend development / Frontend-Backend integration
Rachel Kitchen:	Frontend UX design, web dev
Niles Rogoff:	Backend development, blockchain specialist

General Expectations

These are expectations for all team members. These should be true throughout the semester with the exception of spring break and emergency situations.

- Attend the regular meetings (two times per week)
- Not miss more than 3 meetings
    - If you miss 2 meetings in a row, you must cater food at the next meeting
- Spend 8-10 hours on the project each week
- Respond to project-related messages within 24 hours

Spring Break

Over spring break, no major work is expected. When creating the Gannt chart, major work expectations near spring break will be pushed before the start of break (if possible) or pushed to the week after.

Meeting times

Weekly Meetings
Tuesday 1pm, Thursday 2pm in the Systems Lounge.

Key Meetings
Key meetings will be attended by all members before each key checkpoint. These meetings are used to demonstrate work completed and receive feedback for final tweaks before progressing to the next step.

Contact Policy
Message group members on Discord as primary method of contact.

- Standard hours: 10am-10pm on weekdays.
- Group members will respond within a few hours during standard hours.
- Phone calls must be scheduled.

---

1. **Rachel**: Message me any time, on weekdays I'll respond within 2 hours but on weekends it may take more time. I probably won't answer calls unless we're already messaging and we agree to call.
2. **Ankita**: Message me on discord, I will respond within one hour most times. I will not answer phone calls unless it was pre-decided to have a phone call.
3. **Niles**: Message me on discord.
4. **Nathan**: Message me on discord between 6:00am-10:30pm. No promise I will answer outside of those hours.
5. **Sameer**: Message me on Discord and I'll usually answer within an hour or so. Phone cal

Cost Expectations

Between now and our due date of April 24, we have a period of six weeks for development. Each team member should strive to push for at least 10 hours of work per week until then. Leaving a week for spring break during the business week from March 9th-March 13th as a break, this gives us approximately 90 hours per team member across the entire term, or ~450 hours of work across the entire team.

Backend (40 hours)
- **EOSIO: (3 hours)**
    - **Smart Contract:** Setup of EOSIO smart contract
    - **Economy:** creation of EOSIO based token
- **Web Server: (2 hours)**
    - **AWS:** Creation of http server via AWS to host everything on

- **Buy/Sell/Lottery: (35 hours)**
    - **Buy:** creation of a way to handle buying tickets
    - **Sell:** creation of a way to handle either selling tickets between users, or selling back to Virginia Tech.
    - **Lottery:** Handle way that lottery applicants and season ticket holders are distributed tickets
- Testing

Frontend (100 hours)

- **Desktop Webapp (100 hours)**
    - **UI:** Create a basic website that allows a user to interact with our backend. This should include the following interfaces:
        - Account information page
        - Buy page
        - Sell page
        - Lottery page
        - Tickets page

Integration (80 hours)

- **Frontend/Backend Integration: (60 hours)**
    - **(20 hours)** Connecting the back end via the AWS web server to both the desktop webapp and the mobile app
    - **(40 hours)** Connecting a web server on the AWS server to the eosio blockchain and allowing it to query data and push actions
- **Testing (20 hours)**
    - Ensuring that the transitions between the frontend and the backend has no errors

All other time will be allocated towards other improvements outside of the MVP. Other improvements include mobile functionality, issuing tickets via qr codes, auctioning, and incentive-based rewards/penalties

Evaluation

What does success look like?

Back-End Success

- Tickets, games, and lottery entries will be stored on the EOSIO blockchain
    - They will be managed through a smart contract
- Hokie Tokens will be stored, transferred, and otherwise managed on the EOSIO blockchain, via the eosio.token contract
- Integration between the front-end and back-end is seamless

Front-End Success

- Website front-end application
    - Login Interface
        - Allows for unique usernames and passwords
    - Portal containing account information
- Well-designed and visually appealing
    - Separate locations for the different aspects of the application
        - Easy to see where everything is/easy to use all aspects

Application Success

- Buying and selling football tickets to and from Virginia Tech
- Entering the ticket lottery
- Withdrawing from the ticket lottery
- Receiving a ticket from the lottery
- Reach goals:
    - Having an auction system for the tickets
    - Attending other sporting events adds to your existing pool of Hokie Tokens

Video walkthrough with a specific storyline use case

- “Nathan wants to enter the lottery for the upcoming football game…”
    - Starting from the logging in
    - Moves onto entering the lottery system
    - Show a simulation of tickets being distributed
    - “Nathan realizes he can’t go to the game, but wants to earn some HokieTokens”
        - Scroll to a page containing how to sell tickets back to the university
    - Talk about the use of HokieTokens and how attending games and scanning a QR code will add to your tokens, which can allow you to buy tickets

Contact Information

Name	Phone #	Email
Rachel Kitchen		rachelk4@vt.edu
Ankita Khera		ankita99@vt.edu
Sameer Dandekar		sameerd@vt.edu
Niles Rogoff		nilesr@vt.edu
Nathan Kennedy		nathanmk@vt.edu