Web3 application implementing a ticketing platform

6CCS3PRJ Final Project

BSc Computer Science

Author: Henry Li
Student ID: 1923074
Supervisor: Dr Agi Kurucz

April 2023

Abstract

Web3 is new and emerging technology that plans to revolutionise the way we use the web. Traditional ticketing systems that do not use Web3 face issues of counterfeiting, third party resellers and fraud. These issues negatively impact both the ticketing platform as well as the attendees. This project aims to leverage the use of blockchain technology to alleviate these issues. Users will be able to purchase tickets using cryptocurrencies and resell directly through the platform.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated the contrary.

I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service.

I confirm this report does not exceed 25,000 words.

Henry Li

April 2023

Acknowledgements

I would like to thank my supervisor, Dr Agi Kurucz, who has provided exceptional support and communication throughout the project. The guidance and feedback provided was essential to progression of this project.

Contents

1. Introduction

1.1. Aims & Objectives

An emerging technology in recent years has been the use of blockchain and cryptocurrencies. I am passionate to explore and learn about these technologies and implement them in an application that has real-world use cases that benefit users.

As this will be my first time diving into using these technologies it will require a significant amount of research and self-learning. This project will let me significantly further my knowledge and skills in these areas.

The main requirements for this project will be:

- To have a web application that will allow users to purchase tickets for music events.
- The ticket should be in the form of a token, which is generated by a smart contract on a blockchain.
- The application should allow the user to connect their crypto wallet in order to perform transactions.
- The application should be user-friendly by having good flow and layout.

2. Background

2.1. Blockchain

A blockchain is a digital form of data storage, where transactions are logged and stored in chronological order. A block consists of many transactions which subsequently form the blockchain. These blocks must be verified by nodes on the blockchain network in order to be accepted and added to the blockchain. Consensus mechanisms such as Proof of Work (PoW) and Proof of Stake (PoS) are used in order to decide whether blocks are accepted into the blockchain or not. (1)

As blockchains are maintained by multiple nodes, it gives them a decentralised nature as no one single entity owns the blockchain. This gives them special characteristics compared to traditional data storage solutions. One of these key characteristics is the fact that data on the blockchain is immutable, this means that once the data is on the blockchain it cannot be modified.

Each block is identified by its own unique cryptographic hash value. The hashing process involves using meta-data such as the checksum of the hash from the preceding block. Therefore, if a malicious actor attempted to modify data on a block it would invalidate the hashes of future blocks, breaking the blockchain. (2)

For there to be an incentive for nodes to validate the network, a transactional fee called a gas fee is paid to the validators when they calculate the hash for the latest block to be added to the network. (3) The Ethereum blockchain is a popular blockchain that incorporates this system, requiring a gas fee to be paid for every transaction that writes data to the blockchain. The gas fee is set by the network's validators (miners), meaning that it can

fluctuate depending on supply and demand between the miners and users of the network writing to the blockchain. (4)

2.2. Web3

Web2 is the version of the internet most users are interacting with today. This version of the web is dynamic as it allows users to read and write to the web. This version of the web is dominated by tech giants such as Apple, Meta, Google and Amazon and has led to an explosion of the number of users using the web daily. (5) The data stored on Web2 platforms is centralised, meaning it is owned by a single entity. This has led to privacy and security concerns for users as they have little control over their online information.

Web3 is the next iteration of the web that aims to solve these issues by providing decentralisation of the web. This version of the web is governed by the users themselves and data is mainly stored on peer-to-peer networks such as blockchains. Web3 provides users full ownership of their data, i.e., they can "read-write-own" the web and do not have to rely on central entities such as Google to store their data. (6)

2.3. Tokens

A fungible asset is an asset that can be swapped or traded for an identical object which has the same value; for example, a dollar bill can be exchanged for another dollar bill — they are of equal value.

As opposed to fungible assets, non-fungible assets are different as they cannot easily be swapped or traded for another non-fungible asset as each non-fungible asset is considered as "one-of-a-kind", giving them a collectable nature.

A token is a blockchain-based record that represents a piece of digital media and proves ownership to its owner. Tokens can represent any type of digital asset — examples include digital artwork, music files and concert tickets. They can either be non-fungible (NFTs) or fungible.

Both types of tokens can be purchased and sold through Web3, using cryptocurrencies such as Bitcoin or Ethereum to facilitate the payment.

2.4. Smart Contracts

A smart contract is a piece of programming code that is deployed onto the blockchain. They can include functions that can be executed when specified conditions are met. As smart contracts are deployed onto the blockchain, it means the logic of the code cannot change — if you wanted to change a function in a smart contract it would require another deployment of the contract onto the blockchain. (7)

Smart contracts can be used to create tokens, this process is called “minting” and involves creating a digital asset and publishing it to the blockchain. For Ethereum, there is a widely used ERC-1155 token standard which facilitates functions including minting, transfer and burning (destruction) of tokens. (8) Smart contracts on the Ethereum network are programmed in a language called Solidity.

2.5. Web Ticketing Platforms

A web ticketing platform is usually hosted on the World Wide Web and can usually be accessed through a web browser or smartphone application. These platforms allow the user to browse and purchase tickets for any type of event or service.

Once the ticket is purchased, users will usually be able to either collect a physical ticket from a specified location, print the ticket, or show the ticket on an electronic device such as a smartphone.

2.6. Security

Purchase of tickets through traditional online web ticketing platforms is typically completed through a credit or debit card transaction, where the customer authorises the merchant to charge the card that they have provided their details for. As these transactions occur over the internet, it requires security measures such as HTTPS (9) to encrypt data being sent between the merchant and customer to prevent data theft. If the merchant does not implement sufficient security measures it poses severe security threats to the customer.

Modern browsers such as Google Chrome have implemented warnings to inform users when they are visiting websites that make them vulnerable to threats such as Man-in-the-Middle (MITM) attacks (10), however users running outdated browsers may not see such warnings and continue to visit these websites and have their information stolen.

In this project I will be using Web3 technologies to avoid these issues associated with Web2 ticketing platforms. Payment will be made through cryptocurrency transactions which have to be signed by the sender's private key to ensure the transaction is not sent maliciously. (11) This also prevents the need for any sensitive information such as credit card numbers to be sent over the internet, which risk being stolen.

2.7. Ticket Resale

A major issue concerning current ticketing platforms is the secondary ticket market. Major ticketing companies such as Ticketmaster face frustration and

backlash by customers due to users buying a large number of tickets and reselling them on the secondary ticket market at extortionate prices. This practise has been informally named as “price gouging” or “scalping” and has been a centre of criticism for many ticketing companies in recent years. In 2019 the secondary ticket market was worth $15.19 billion which has prompted the creation of legislation in Europe to help combat the issue. (12)

The use of tokens can be used to prevent scalping of tickets as the tokens cannot be traded or resold unless functionality is added in the smart contract. As resold functionality is defined in the smart contract, it can be used to limit the price at which tickets are resold for in order to prevent excessive pricing. This also means that resale can occur officially through the platform, as opposed to through third party services which lowers the risk of buyers from being victims of fraudulent behaviour.

2.8. Existing work

As blockchain technology has only come into fruition within the past few years, the main focus has been on the sale of tokens as digital assets as a commodity.

OpenSea is a well-known NFT marketplace that was launched in 2018 that allows users to browse through many categories and collections of NFTs and to auction on them. The NFTs purchased through OpenSea are just digital assets with no extra functionality.

YellowHeart is another NFT marketplace that leverages NFTs to also be used in the context of ticketing. Users can purchase NFTs which represent a ticket for a particular event. Whilst this implementation has many benefits, it is not without its shortcomings. As YellowHeart uses the ERC-721 token standard (13), its tokens can only be non-fungible. This leads to high costs

and poor scalability as it requires a new smart contract to be deployed to the blockchain for each event.

This project aims to deal these shortcomings by using the ERC-1155 token standard (8) that allows us to create multiple tokens with different IDs within a single contract which suits the purpose of ticketing more appropriately. Additionally, these tokens can be made fungible, allowing for multiple quantities of a token to be created. This means we can represent the ownership of a token of a specific ID as a ticket for a specific event.

3. Requirements & Specification

3.1. Requirements

I have decided to use the Ethereum blockchain for my web application as this is one of more mature blockchains that has more development tools and documentation available. In addition, the Goerli testnet (15) will be used to develop my application so that no real Ether (ETH) cryptocurrency will need to be used for transactions.

This web application will require the installation of the MetaMask browser extension (14) to facilitate the purchase transactions to the smart contract from the user. MetaMask is available on both Firefox and Chromium based browsers. If the user does not have this extension installed it will direct them an installation page.

3.1.1. Content Requirements

- C1 — Application should be able to store music events.
- C2 — User should be able to search for music events by artist, genre or location.
- C3 — User should be able to filter events by genre, location and price.
- C4 — User should be able to select an event to view more details.
- C5 — User should be able to connect their MetaMask wallet.
- C6 — User should be able to buy a ticket for an event. This can be a new ticket or a resale ticket.
- C7 — User should be able to list a ticket they have purchased for resale.

- C8 — Application should show a confirmation prompt on sensitive actions.
- C9 — Application should provide appropriate visual feedback to the user in the case of errors or messages.
- C10 — User should be able to view the tickets they have purchased.
- C11 — Application should have an intuitive and easy to navigate Graphical User Interface (GUI).
- C12 — Admins should be able to create or modify events.
- C13 — Admins should be able to scan a ticket to mark it as used.

3.1.2. Smart Contract Requirements

- S1 — Smart contract should be deployed onto the Goerli testnet.
- S2 — Smart contract should allow the purchase of a token.
- S3 — Smart contract should allow the resale of a token.
- S4 — Smart contract should provide the remaining quantity of tokens available for an event.
- S5 — Smart contract should be able to transfer its balance of Ether to a specified wallet address.
- S6 — Smart contract should be able to mark a token as used.
- S7 — Smart contract should be able to associate metadata with a token.
- S8 — Smart contract should allow for creation and modification of events.

3.2. Specification

The priority of each requirement will be assessed; high priority requirements are considered essential for the project, whilst low priority requirements are welcome additions that are not critical. I will implement the requirements in the respective order: high, medium, low.

3.2.1. Content Specification

Requirement Code	Requirement	Specification	Priority
C1	Application should be able to store music events.	SQLite database will be used to store the events. Events fields: name, description, artist, venue, city, date & time, image URL, contract deployment status, cancelled status.	High
C2	User should be able to search for music events by artist, genre or location.	Search bar in the navigation bar of the website. Event listeners to detect search submission which is then used to query the database for events.	Medium
C3	User should be able to filter events by genre, location and price.	Filter inputs on the events page. These inputs will be used to query the database for events matching the criteria.	Low
C4	User should be able to select an event to view more details.	User should be able to click an event which takes them to a page for that specific event.	Medium

Requirement Code	Requirement	Specification	Priority
C5	User should be able to connect their MetaMask wallet.	Button which causes MetaMask to prompt the authorise their wallet with the website.	High
C6	User should be able to buy a ticket for an event. This can be a new ticket or a resale ticket.	Button which causes MetaMask prompt to authorise transaction to buy the ticket.	High
C7	User should be able to list a ticket they have purchased for resale.	Button which interacts with the smart contract to accomplish this.	High
C8	Application should show a confirmation prompt on sensitive actions.	A confirmation modal should appear where they can either confirm or cancel the action.	Medium
C9	Application should provide appropriate visual feedback to the user in the case of errors or messages.	Alerts should show on the user's webpage.	Medium
C10	User should be able to view the tickets they have purchased.	A page where they can view the purchases they have made in the past.	High
C11	Application should have an intuitive and easy to navigate Graphical User Interface (GUI).	Web elements should be easy to read and spaced well. The layout should be responsive to accommodate different screen sizes.	Medium
C12	Admins should be able to create or modify events.	A password protected page where events can be modified. Changes will be reflected in the database.	Low

Requirement Code	Requirement	Specification	Priority
C13	Admins should be able to scan a ticket to mark it as used.	A password protected page that scans a ticket using a camera. It is then marked as used on the smart contract.	Low

3.2.2. Smart Contract Specification

Requirement Code	Requirement	Specification	Priority
S1	Smart contract should be deployed onto the Goerli testnet.	Infura (16) will be used to facilitate deployment of the smart contract onto the Goerli blockchain.	High
S2	Smart contract should allow the purchase of a token.	Contract should implement the ERC-1155 token standard (8) to issue tokens. The creation of the token should require payment.	High
S3	Smart contract should allow the resale of a token.	Contract should have a function that marks it for resale and a function to buy the resale ticket.	High
S4	Smart contract should provide the remaining quantity of tokens available for an event.	Contract function to return the quantity and number of tokens supplied for an event.	Medium
S5	Smart contract should be able to transfer its balance of Ether to a specified wallet address.	Contract function to transfer balance to an address.	Low
S6	Smart contract should be able to mark a token as used.	Contract function to mark a token as used.	High

Requirement Code	Requirement	Specification	Priority
S7	Smart contract should be able to associate metadata with a token.	Contract function that returns a URI pointing to JSON metadata for the token.	Low
S8	Smart contract should allow for creation and modification of events.	Contract functions to create an event and modify events.	High

4. Design

4.1. Use Cases

In the use case diagram below, we identify the interactions between the actors and the systems, defining the main functionalities of our application.

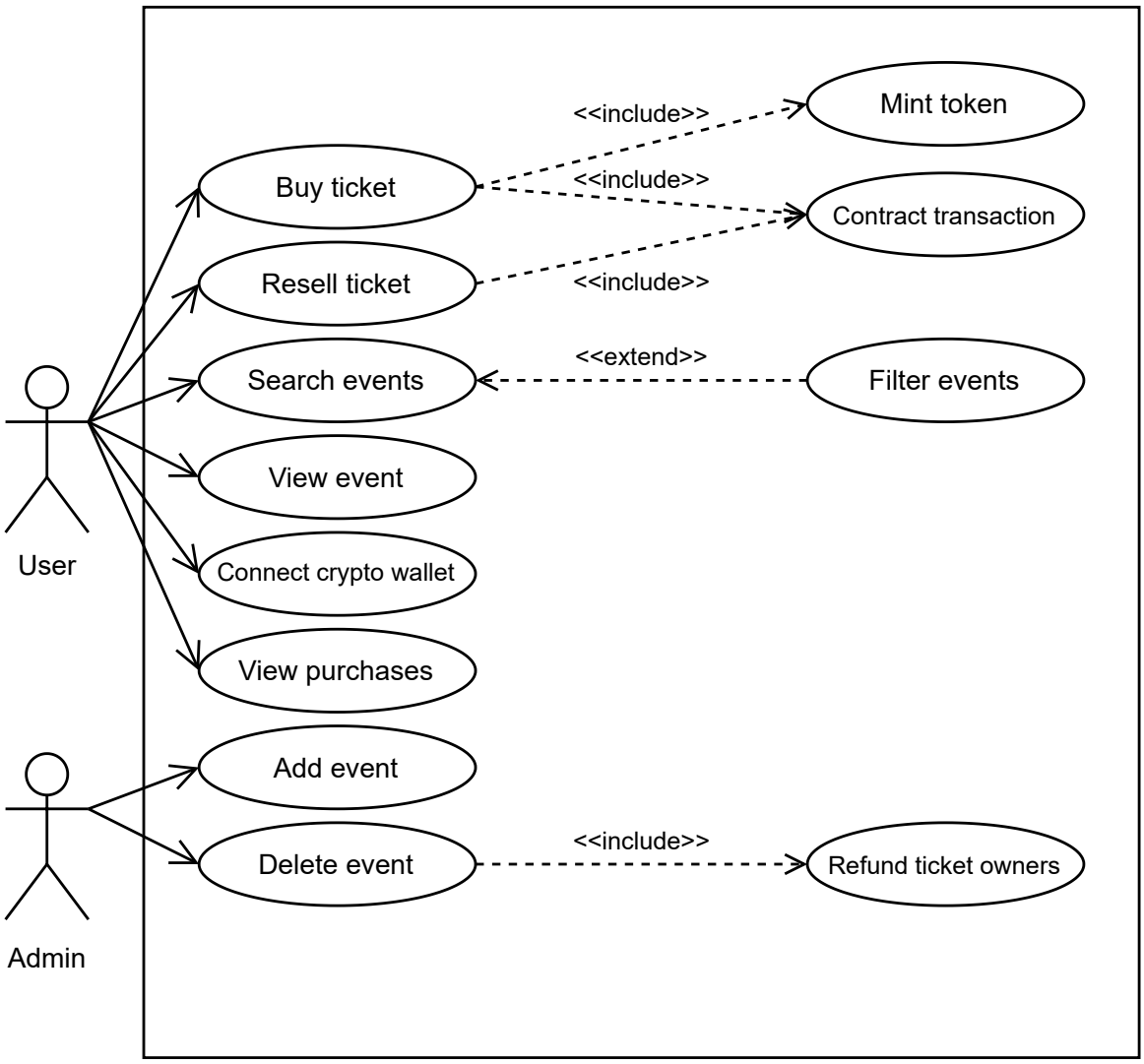

Figure 4.1: Web3 Application Use Case Diagram

Users will access the application via a URL on their web browser, where they will arrive at the home page of the application. They will be able to navigate to events and connect their crypto wallet. Once users have

connected their crypto wallet, they will be able to buy tickets and view their previous purchases, where they also can list a ticket for resale.

Admins will be able to access their interface by a specific URL, where they will need to enter a password for authentication. Once authenticated they can add and delete events.

4.2. System Architecture

The system architecture is divided into three main layers. The interactions between the different components of these subsystems can be seen in the diagram below (Figure 4.2). The arrows between the subsystems indicate dependency.

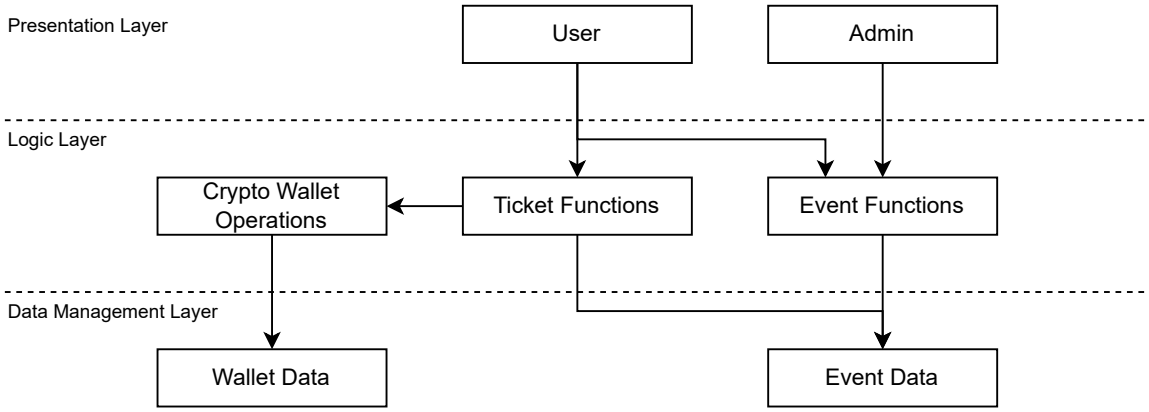

Figure 4.2: Web3 Application System Architecture Diagram

The presentation layer represents what the actor will see when they use the application. These are reliant on the logic of the subsystems in the logic layer. Users can interact with event functions (viewing) and ticket functions (buying/reselling) which require the crypto wallet operations. Meanwhile admins can interact with a different subset of event functions (viewing/adding/deleting). Each of the logic layers work on some data which will be available to interact with either on the application's database or the blockchain.

4.3. Data Flow

In the data flow diagram (Figure 4.3) below we can visualise how data moves around in the system for the main use cases.

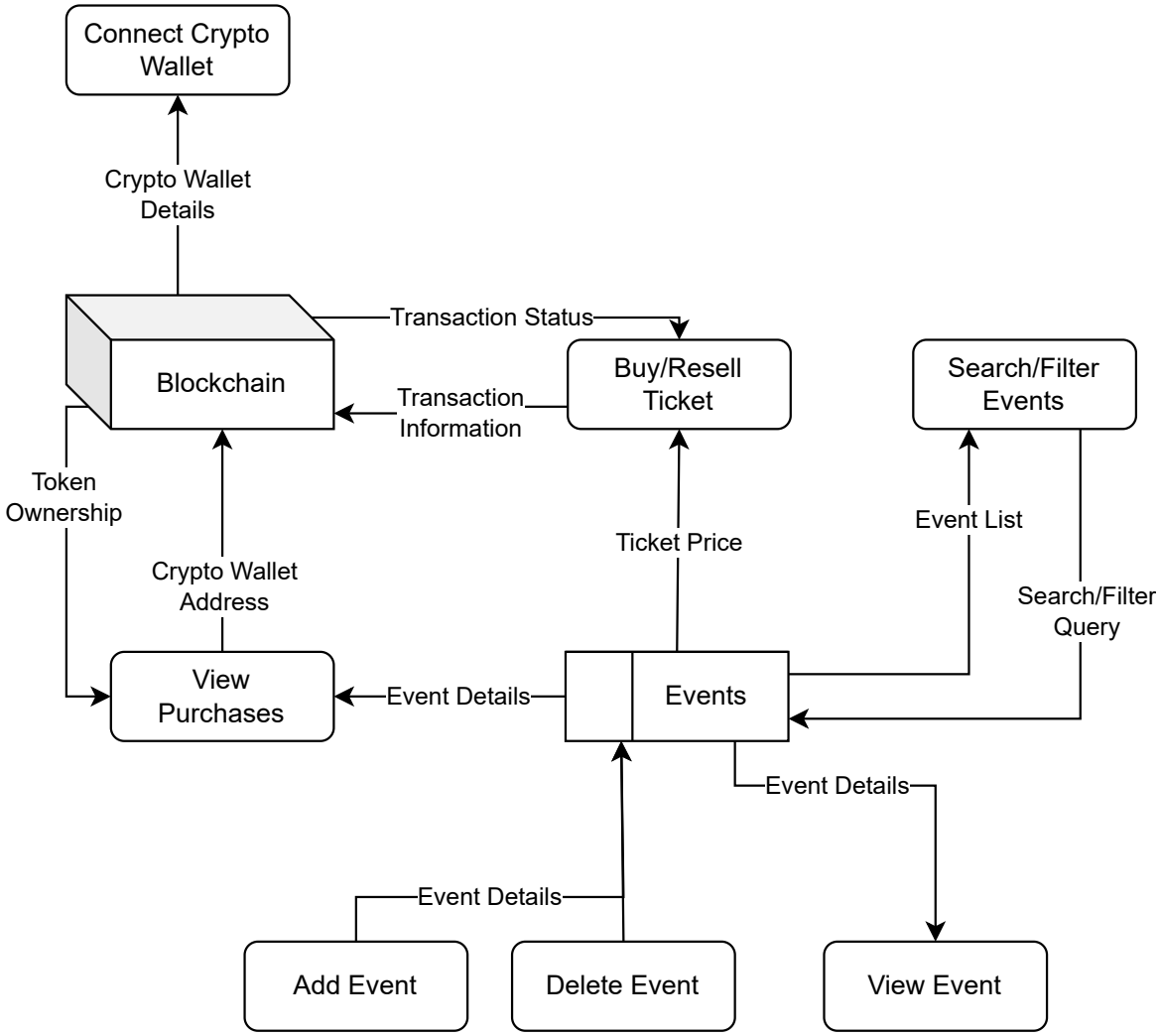

Figure 4.3: Web3 Application Data Flow Diagram

The central point of data in this application will be our events, essential event information will be published onto the blockchain to facilitate the purchasing of tickets. Although we could theoretically store all the data on the blockchain, it helps to have key information available within our system itself for quick access. However, we will have to think carefully during implementation about how to deal with potential data inconsistencies

between the data on our system and the corresponding data on the blockchain.

The ‘user’ in this system will be identified by their crypto wallet address as this will be unique on the blockchain. Therefore, there is no requirement to store user data on our database.

The logic behind data retrieval and manipulation in this system will be handled by functions on the smart contract in the case of the blockchain, or through a RESTful API on our backend server in the case of our own database.

4.4. Graphical User Interface

The user interface for the application is an essential part of the system and is primarily what the end users will interact with. There should be a balance between simplicity, flow, ease of use and functionality. Whilst the interface should be friendly for less experienced users, it should avoid restricting functionality for more advanced users. The aim is to have all round good user experience.

As our system is a web-based application, I have designed wireframes for the different views of our website, these can be found in appendix A1. As we are using MetaMask extension to initiate transactions with the smart contract, some of user interaction will be completed through the extension’s interface (see appendix A2 for an example). Our website should ensure good flow and handoff between the two.

4.4.1. Home Page

This is the first page users will see when visiting the website. Here they can view general information and navigate to other pages.

4.4.2. Navigation Bar

The navigation bar will be present on all non-admin facing pages. There will be a search bar for events, links to other pages and a button to connect their crypto wallet via MetaMask. When connecting their wallet, a prompt will open in MetaMask asking the user to confirm the action. If they have already connected their wallet, the text of the button will reflect this and instead the button will act as a disconnect button. Different links will show depending on whether users have connected their wallet or not — for example, a link to the purchases page will not show if they have not connected their wallet.

4.4.3. Browse Events Page

Here users will be able to browse the upcoming events they can purchase tickets for. They will have the ability to filter by genre, location and price. When users search for events from the navigation bar, it will take them to this page with the relevant events that match their search query. They will be able to use the filters with their search query simultaneously.

4.4.4. Event Page

This will show key information for the event — its name, date, venue, location, price, description and genres. There will be a purchase button to allow users to buy the ticket. If the event has sold out, the button will be disabled with text reflecting it is sold out. Additionally, if resale tickets are available, there will be a button to view all the listings. If users have not connected their wallet, the identical connect wallet button found in the navigation bar will be shown in place of the purchase button instead.

The genres for the events will have their own dedicated pill shaped buttons, when these are clicked it will take them to the browse events page with the respective genre applied in the filters.

4.4.5. Purchases Page

This page can only be accessed if the user has connected their wallet, otherwise they will be redirected to the home page.

On this page a list of events and their details the user has purchased will appear. These purchases will have a button to allow users to list or unlist tickets for resale if they are not expired or cancelled. Users will be easily to switch between viewing upcoming, selling, cancelled and expired tickets by clicking on the respective tabs near the top of the page.

4.4.6. Admin Page

Admins will have the ability to edit events by clicking on the event to update their details. Admins will be able to cancel events from this page. A button on the top right of the page will let admins add an event to the application, in addition to a button to navigate them to a page to scan tickets.

4.4.7. Admin Add/Edit Event Page

This page will be used for both adding and editing events. Page text and fields will dynamically update depending on which action is taken. Admins can enter and submit the required data which will update the information for the event in the database. The data entered in these fields will be validated before the data is submitted — if the data for a field is invalid it will show an error message below the field indicating them to amend the issue.

4.4.8. Admin Scan Ticket Page

On this page admins will be able to use a camera on their device to scan tickets and subsequently mark them as used.

4.4.9. Confirmation Modal

This is a simple dialog box that will show for certain actions that require confirmation. It will display text about the action they are intending to perform. They will have the option to either continue or cancel the action by clicking the respective buttons. This will be used on important actions where making a mistake could cause detrimental effects e.g., accidentally cancelling an event.

5. Implementation & Testing

Please refer to appendix D for the source code of the implemented system.

5.1. Development Approach

For most of this project, a waterfall development approach was taken. This means that the specification and design of the core features in the application were planned before implementation. Although this approach requires more planning time upfront, it provides a clear structure for implementation and can lead to higher quality assurance. A small number of features were implemented using an iterative approach due to the flexibility it provides, allowing features to adapted easily.

Throughout implementation, clean code principles (17) were followed to ensure code cleanliness. This was an important factor as it ensured readability, maintainability, scalability and easier debugging. This reduces technical debt in the long run and promotes better software quality.

Below is the order of implementation taken throughout the project, with references to the requirement codes next to them:

1. Smart contract ability to store core event information. (S2, S3, S4, S8)
2. Smart contract ability to purchase a ticket for an event. (S2)
3. Smart contract resale functionality of tickets, allowing tickets to be listed for resale. These resale tickets can be purchased by other users. (S3)
4. Smart contract ability to mark a ticket as used. (S8)
5. Frontend web application ability to connect MetaMask wallet. (C5)

6. Database schema to store event information. (C1, C2, C3, C4)
7. Frontend web application ability to purchase a ticket for an event. (C4, C6)
8. Frontend web application ability to view purchases. (C7, C10)
9. Frontend web application ticket resale functionality. (C6, C7)
10. Frontend web application event searching and filtering. (C2, C3)
11. Add confirmation modals to ticket resale listing and event cancellation. (C8, C11)
12. Show success and error messages when performing actions. (C9, C11)
13. Frontend web application admin interface to create and modify events. (C12)
14. Frontend web application admin interface ticket scanning functionality. (C13)
15. Smart contract ability to transfer balance to a wallet address. (S5)
16. URI for metadata associated with each token. (S7)

5.2. System Functionalities

5.2.1. Smart Contract Functionality

In order to streamline the development of the smart contract used in the application, the Truffle Suite (18) framework was used. Truffle Suite provides tools to compile, test and deploy smart contracts. The contract builds upon the ERC-1155 token standard (8) as well as the OpenZeppelin `Ownable` contract (19) to handle function access control, allowing for certain contract

functions to be restricted to the contract owner. The functionality included in the smart contract is as follows:

1. Key value mappings: these allow for information to be fetched from the smart contract in a single call. There are four mappings in the implemented smart contract:

    - `resaleTokenEntries`: this maps a wallet address to an array containing which tickets they have for resale. This is used to retrieve the resale status of purchases for a given user.

    - `resaleTokens`: this maps an event ID to an array of all resale tickets for the given event. This is used to display the resale tickets for an event.

    - `events`: this maps an event ID to its corresponding event data. This includes time, price, total quantity, created status, cancelled status.

    - `usedTokens`: this maps a wallet address to an array of event IDs for which tickets have been used. One entry in the array represents one used ticket for that event.

    Mappings in Solidity have a default value for every single key possible. Therefore, in some cases, we need to include data to distinguish whether the value for a key has been initialised or if it is just a default value. For example, this is the reason why the event data in the `events` mapping has a “created” status.

2. Ticket purchasing: when the `buyToken` function is called, the contract checks that the event is valid; it should be created and not be cancelled. The event should have available quantity and not be in the past. Additionally, the payment provided should be of sufficient value.

Once these checks pass, the `supplied` attribute of the event is updated, and the token(s) are minted to the buyer.

3. Marking tickets as used: when the `markTokenAsUsed` function is called, the event is checked to be valid, and the user is checked to have enough tickets remaining to mark as used. The tickets are then marked as used by adding entries to the `usedTokens` mapping.

4. Listing tickets for resale: the `listTokenForResale` function checks whether the event is valid and if the user has enough tickets to list for resale. The tickets are then listed by adding entries to the `resaleTokens` and `resaleTokenEntries` mappings.

5. Unlisting tickets for resale: the `unlistTokenForResale` function checks the event is valid and checks the quantity to unlist is at least what the user has already listed. It then updates the entries on the `resaleTokens` and `senderResaleTokenEntries` mappings to mark them as sold.

6. Buying a resale ticket: when the `buyResaleToken` function is called the event is checked to be valid and the payment is checked to be sufficient. It then attempts to find the resale token by looping through the `resaleTokenEntries` for the specified address. If it is found, this entry is marked as sold and the token is transferred to the buyer. The payment is then transferred to the seller.

7. Event creation: the `createEvent` function allows for an event to added to the contract. It checks the event with the supplied ID has not already been created, the quantity is greater than zero and the time is in the future. It then adds the event to the `events` mapping.

8. Event updating: the `updateEvent` function checks the event is valid. It ensures the quantity is greater than zero and greater or equal to the number already supplied. The time must be in the future. The corresponding event is then updated in the `events` mapping.

9. Event cancellation: the `cancelEvent` function checks the event is valid, and whether the supplied arrays of owner addresses and corresponding ticket quantities are valid. These arrays are used to supply refunds to the ticket holders and are supplied to the function as parameters as they cannot be obtained from the contract itself — instead the blockchain must be queried externally. If there is enough balance in the contract to supply the refunds it will mark the event as cancelled in the `events` mapping and refund the ticket holders.

10. Transferring balance: the `transferBalance` function simply transfers the balance of the smart contract to a specified wallet address.

5.2.2. Frontend Web Application Functionality

The frontend was built using the React library (20) to handle rendering of UI components. React handles rendering efficiently by only updating parts of the UI that need to be changed depending on the state of the application. Tailwind CSS (21) and Bootstrap Icons (22) were used to assist in styling the web application. Throughout the development process page responsiveness was kept in mind, adjusting the content to fit screens of all sizes. The functionality of the frontend application are as follows:

1. Routing: React Router (23) was used to map URL paths to specific views within the application. It also manages navigation between views and can provide contexts across the application to hold global states.

2. `Wallet` component: provides a context to access the wallet address. It will automatically connect to the wallet at application start if it has been connected previously and handles switching the wallet to the Ethereum test network. It listens to events to update the wallet address if accounts are switched by the user and disconnects the wallet if it is on the incorrect Ethereum network. The interaction with the wallet is achieved through the Web3.js library. (24) The user can also choose to disconnect their wallet; in which case this state is stored on the browser's local storage using the useHooks library (25) to prevent automatically reconnecting on the next visit to the application.

3. `RequireWallet` component: allows views to be restricted to users that have their wallet connected. If the user does not have their wallet connected, they will be redirected.

4. `Admin` component: provides a context to access a state determining whether the user is an admin or not.

5. `RequireAdmin` component: restricts views to admins only. Other users will be redirected.

6. `api` module: uses the Axios library (26) to send HTTP requests to the backend server API.

7. `NavBar` component: provides the navigation for non-admin facing views, allowing users to easily navigate the application. It dynamically updates based on the status of the user e.g., the purchases tab only shows if the user has connected their wallet. The navigation bar also includes a search bar for events and a button for users to connect their MetaMask wallet.

8. `HomeView` component: this is the first view the user is greeted with when visiting the application. Users can easily browse events from here.

9. `EventsView` component: this view allows the user to browse all upcoming events. These events are fetched from the database through HTTP requests, supporting pagination. Filters for genre, location and price can be applied, with the results updating immediately. Searching events can be achieved from the navigation bar. The state of the search term, filters and page number is preserved in the URL query parameters so that they are not reset when the user refreshes the page or navigates back/forward to the page leading to an improved user experience.

10. `SingleEventView` component: from this view, more detailed information about the event can be viewed. There is an individual pill shaped button for each genre, when clicked will take the user to the `EventsView` with the corresponding genre applied in the filters.

    A connect wallet button will show if the user has not connected their wallet, otherwise, a purchase button will be shown. The user can choose to purchase up to six tickets at once. Clicking the purchase button triggers a MetaMask prompt to open in order to confirm the transaction. If there are no more tickets available, the purchase button will be disabled. In the case there are resale tickets available from other sellers, a button to view resale tickets will be shown next to the purchase button.

11. `SingleEventResaleView` component: this view shows a list of resale tickets with their corresponding seller address. Users can purchase any

of these tickets, prompting them to confirm the transaction in MetaMask.

12. `PurchasesView` component: here users can view all the purchases they have made. Purchases are separated by tabs for upcoming, selling, cancelled or expired tickets.

    For upcoming tickets, a QR code can be generated to allow admins to scan their ticket and allow them entry. This QR code is generated using the react-qr-code library (27), and is digitally signed using their MetaMask wallet, making it unique to the user. If the ticket has not expired, users can list or unlist their tickets for resale here.

13. `AdminLoginView` component: this view allows a user to enter a password to access the admin interface. This view can only be navigated to using "/admin" path e.g., http://localhost:3000/admin. The admin password has been set to "kcladmin".

14. `AdminEventsView` component: here admins will be able to browse and search upcoming events. Similarly to the `EventsView`, the page number status is preserved in the URL query parameters. Clicking on an event will allow them to edit it. Admins also have the option to cancel a given event. In the top right there are buttons to navigate to pages to scan a QR ticket or create a new event.

15. `AdminCreateEditView` component: this view dynamically updates depending on whether the admin wants to create or edit an event. The Formik library (28) is used to manage the form state and the Yup library (29) is used in conjunction with Formik to validate the user input. Image upload is handled by the React Drag and Drop files library. (30)

16. `AdminQrScanView` component: admins will be presented with a viewfinder, displaying the image from a camera attached to their device. This is used to scan the QR tickets from users. Once a valid QR code has been detected, the admin will be prompted to confirm the quantity of tickets they want to mark as used. The QR reader functionality is provided by the React QR Reader library. (31)

5.2.3. Backend Server Functionalities

The backend server is powered by the Node.js runtime environment. (32) Node.js provides built-in tools to interact with the host system, as well as supporting third party packages, namely Web3.js (24) which is essential to interact with the smart contract, making it a suitable choice as a backend technology. The functionalities of the backend server are as follows:

1. RESTful API: the Restify library (33) is used to provide these endpoints. Various endpoints provide CRUD operations on both the SQLite database and smart contract data. An SQLite database is used as this is a lightweight, high performing relational database that can handle the project's use cases well. The Knex.js library (34) is used as a query builder, allowing SQL queries to be generated in a programmatic fashion. It also deals with query parameterisation to prevent SQL injection.

   Additionally, there are endpoints to fetch smart contract information required by the frontend web application.

   Input validation for endpoints is assisted by the jsonschema library (35), allowing us to define a JSON schema that needs to be satisfied for the request data.

2. Session management: admin sessions are managed by cookies, using the restify-cookies library (36) to serve the cookies to the client and the uuid library (37) to generate unique session identifiers. This enables access control on our endpoints, allowing us to secure certain endpoints to only be accessed by authorised admins.

3. Local blockchain: during local development, a locally run blockchain is provided by the Ganache library, which is a part of the Truffle Suite (18). This provides unlimited test currency and can process transactions almost instantly, making development more efficient.

4. Database seeding: test data can be seeded to the database to assist with testing the application. The Faker library (38) is used to generate fake data.

5. Serving static files: this can include a production build of the frontend web application, as well as images for created events.

5.3. Implementation Challenges

During development of software, it is inevitable that issues will arise which requires changes to the design of the system or development approach. These issues should optimally be handled in a way that is not detrimental to the maintainability and flexibility of the system.

5.3.1. Ethereum Testnet Limitations

Initially, it had been planned that the Goerli testnet (15) would be used during the development process. However, this proved to be limiting as each transaction takes around 30 seconds to be processed on the blockchain, slowing down development. Additionally, only 0.2 ETH of test currency could

be obtained per day which would be especially troublesome when there were higher transaction costs due to the network being busier.

To mitigate these issues, a local blockchain called Ganache, provided by the Truffle Suite (18) was used when the application was run in a development environment. This allowed for essentially unlimited test currency as well as significantly faster transaction times. In a production environment i.e., deployment, the Goerli testnet is still used. The application dynamically chooses which testnet to use based on supplied environment variables.

5.3.2. Fetching Purchases

The ERC-1155 token standard (8) only supports fetching the balance of a specific token type using the `balanceOf` function. In order to fetch the purchases for all events it would require the function to be called for each of the individual events. This is not a scalable approach.

To work around this, purchases are retrieved by directly querying the blockchain. In the ERC-1155 token standard, every time a token is transferred to an account a `TransferSingle` event is emitted on the blockchain. Therefore, to retrieve the purchases for a user, the `TransferSingle` events are queried from the blockchain and iterated through to filter those relevant to the user's wallet address:

```
const getTokens = async (address) ⇒ {
    const tokens = new Map();
    const events = await instance.getPastEvents(
        'TransferSingle',
        { fromBlock: 0, toBlock: 'latest' }
    );

    for (const event of events) {
        const eventId = Number(event.returnValues.id);
        const quantity = Number(event.returnValues.value);
        let diff;
```

```
            if (caseInsensitiveMatch(event.returnValues.to, address)) {
                  diff = quantity;
            } else if (caseInsensitiveMatch(event.returnValues.from,
      address)) {
                  diff = -quantity;
            } else {
                  continue;
            }

            tokens.set(
                  eventId,
                  (tokens.has(eventId)) ? (tokens.get(eventId) + diff) : diff
            );
      }

      return tokens;
};
```

5.3.3. Session Management

In order to allow for admin authentication of the application, sessions were required to keep track of the user state between HTTP requests. Unfortunately, the Restify library (33) used for the backend server does not natively support this and therefore this functionality had to be implemented manually.

This was achieved through setting a cookie containing a randomly generated session ID `sid` that would be stored by the user's browser. The state of the user is then held in a `sessions` map on the backend server with the session ID as the key. As the user's browser sends this cookie with each HTTP request, we can intercept the request and identify the user by referring to the `sessions` map to then set the request object's session attribute `req.session` to the user's session object before passing the request to the next handler:

```
const sessions = new Map();

server.use((req, res, next) ⇒ {
    const sid = req.cookies.sid ||
    crypto.randomBytes(32).toString('base64');
```

```
        res.setCookie("sid", sid);

        if (!sessions.has(sid)) {
            sessions.set(sid, {});
        }

        req.session = sessions.get(sid);

        return next();
    });
```

5.4. Testing

Testing is a critical process that ensures the system is robust, defect free and compliant to requirements.

5.4.1. Unit Testing

Unit testing of the smart contract allowed for every function and branch in the code to be tested to ensure correct functionality. Edge cases were also tested in these unit tests to detect unexpected behaviour. Testing of the smart contract is especially important as once the contract has been deployed onto a blockchain, it cannot be changed. This leaves it vulnerable to malicious actors if unintended behaviour occurs. The output of running the unit tests can be found in appendix B1.

High priority requirements for the smart contract were tested after implementation, allowing problems to be fixed before developing the user interface of the system. This led to faster development as less time was spent troubleshooting when key features of the application were implemented. Medium and low priority requirements were iteratively implemented and tested later as they did not affect the functionality of the higher priority requirements.

5.4.2. Non-functional Testing

Non-functional testing was performed across the whole application in order to maintain quality of the software produced. The following quality attributes were tested:

1. Performance: the frontend web application performs efficiently, with care taken to state management to ensure there is minimal re-rendering, speeding up the interaction with the application. Network requests were also monitored using Chrome DevTools to assure data was only being fetched when needed.

2. Compatibility: the React library (20) used to build the frontend web application supports all modern browsers, which can be used on a plethora of devices. The MetaMask browser extension (14) is also available on the majority of modern browsers and provides standalone applications on mobile devices.

3. Reliability: the application was tested using purposely failing inputs to test error handling. The application would fail gracefully and provide user feedback when relevant; it would not crash or freeze.

4. Security: interaction with the blockchain is securely handled by the Web3.js (24) library. Unauthorised users cannot access admin protected API endpoints. Cross-Origin Resource Sharing (CORS) is setup appropriately so that only the frontend web application can access the resources of the backend server.

5. Usability: the application is intuitive and easy to use. Labels, icons and styling are used appropriately. It provides a seamless user experience throughout. The user interface is responsive to accommodate for different screen sizes.

6. Modularity: the system is divided into separate components with loose coupling to allow for easy reuse. The structure of code makes it easy to add or change functionality.

5.4.3. Requirement Based Testing

Here, the functionality of the system is directly tested against the requirements outlined in chapter 3. Where possible, screenshots supporting these tests are shown in appendix B2.

Content Requirements

- C1 — Application should be able to store music events: the SQLite database contains information regarding music events.
- C2 — User should be able to search for music events by artist, genre or location: events can be searched using the search bar found in the navigation bar.
- C3 — User should be able to filter events by genre, location and price: the events page provides a list of genres and locations that can be checked or unchecked, and a maximum price can be entered. When these are changed the events are filtered accordingly.
- C4 — User should be able to select an event to view more details: from the events and purchases page, a specific event can be clicked on to bring the user to a dedicated page for the given event.
- C5 — User should be able to connect their MetaMask wallet: a connect wallet button can be found in the navigation bar, clicking on it causes a MetaMask prompt to open to allow the user to connect their wallet.

- C6 — User should be able to buy a ticket for an event. This can be a new ticket or a resale ticket: when the user has connected their wallet, they will have the option to purchase tickets from the event page. This initiates a transaction with the smart contract to buy the ticket.

- C7 — User should be able to list a ticket they have purchased for resale: users can view their purchases, where they have an option to list any unused tickets for resale. This initiates a transaction with the smart contract.

- C8 — Application should show a confirmation prompt on sensitive actions: confirmation prompts are shown when users are listing or unlisting tickets for resale, when admins are marking tickets as used and when admins are cancelling an event. These prompts allow the user to cancel or proceed with the action.

- C9 — Application should provide appropriate visual feedback to the user in the case of errors or messages: when an error occurs, either an error message is shown on the page, or an error prompt appears.

- C10 — User should be able to view the tickets they have purchased: once the user has purchased tickets, they will appear on the purchases page.

- C11 — Application should have an intuitive and easy to navigate Graphical User Interface (GUI): the application uses a simple and elegant design that is clear and easy to follow.

- C12 — Admins should be able to create or modify events: the admin interface provides forms to create or modify events.

- C13 — Admins should be able to scan a ticket to mark it as used: the admin interface has functionality to scan a QR code for a ticket, which then shows a prompt asking the admin how many tickets they want to mark as used.

Smart Contract Requirements

- S1 — Smart contract should be deployed onto the Goerli testnet: the deployed version (see appendix C) of the application uses a smart contract that has been deployed onto the Goerli network.

- S2 — Smart contract should allow the purchase of a token: this is facilitated by the `buyToken` function.

- S3 — Smart contract should allow the resale of a token: this is facilitated by the `listTokenForResale`, `unlistTokenForResale` and `buyResaleToken` functions.

- S4 — Smart contract should provide the remaining quantity of tokens available for an event: the `Event` struct used in the contract to hold event data has the field `quantity` that can be subtracted from the field `supplied` to calculate the remaining quantity.

- S5 — Smart contract should be able to transfer its balance of Ether to a specified wallet address: this is facilitated by the `transferBalance` function.

- S6 — Smart contract should be able to mark a token as used: this is facilitated by the `markTokenAsUsed` function.

- S7 — Smart contract should be able to associate metadata with a token: the contract sets a metadata URI using the `ERC1155` contract (8)

constructor which points to a backend server URI providing a JSON object satisfying the ERC-1155 metadata URI JSON schema (39).

- S8 — Smart contract should allow for creation and modification of events: facilitated by the `createEvent`, `updateEvent` and `cancelEvent` functions.

6. Evaluation

6.1. Software Evaluation

The software implementation was evaluated against its requirements, in addition to non-functional requirements. This was analysed in chapter 5.

6.1.1. Limitations

This project is not perfect. The implemented system has limitations which could be improved with further development. Improvements will be discussed in further detail during the conclusion of the report. The limitations of this project include:

- As described in section 5.3.2, purchases are retrieved by querying the blockchain. Whilst this is a reliable method of retrieving purchases, it is not particularly scalable. The time complexity of the function to retrieve purchases grows linearly with $O(n)$; once there are many purchases, it quickly becomes inefficient.

- Once an event is created, it's price is fixed and cannot be changed. This decision was taken to simplify the logic of the smart contract as making the event price variable would add significant complexity to event cancellations, as the refunding of ticket owners would have to consider the possibility of users paying different prices for the tickets.

- Similarly, the price of resale tickets is fixed to the price of the event.

- Ethereum based blockchains can be slow. Due to the decentralised nature of blockchains and their advanced verification and consensus mechanisms, transactions can take longer to process compared to a

traditional centralised system. This problem is exacerbated when the network becomes busier than usual.

- As smart contracts are immutable, it can lead to less flexibility and higher costs; if changes were to be made to a smart contract, a new version would have to be deployed to the blockchain.

- The password for the admin interface has been hardcoded in the backend server code. This is not a secure or flexible approach; the password can be viewed by anybody with access to the source code and changing the password would require all admins to use the new password.

- The session management described in section 5.3.3 shows that sessions are stored using a map object on the backend server. This is stored in memory and does not persist if the server is shut down, meaning admins would have to log in again if the server was restarted. Additionally, the space complexity of this implementation grows with $O(n)$; this could be exploited by a malicious actor by creating a large number of sessions to the server.

- The backend server code is written in JavaScript, which is a dynamically typed language. This can become difficult to maintain as the language does not provide type safety or advanced static analysis features.

- Where digital signing through the MetaMask wallet is utilised, the `personal_sign` method is used. This method has better compatibility than other signing methods but is less efficient to process on the smart contract. (40)

- The frontend web application has not implemented many accessibility features, which could hinder the user experience for certain users.

- Testing is limited to the smart contract. There may be issues with the system's functionality that stem from the backend server or frontend application.

6.1.2. Web3

Web3 is new technology that has not yet reached widespread adoption. It is still uncertain whether it will be a viable replacement for Web2.

Web3 is an ever-evolving technology, with new improvements and features constantly being added. As Web3 is almost entirely open-source, contributions can easily be proposed and added to the technology. With the given time period for this project and a single developer, it was difficult to implement more advanced features offered by Web3. Contributions to Web3 were also outside of the scope of this project.

6.2. Overall

The implemented application meets the aim of the project, and successfully satisfies the requirements outlined.

The background research conducted at the start of the project provided a solid understanding of the technology used to implement the solution and in addition with the early planning of system architecture and design, it helped greatly reduce the number of issues faced during implementation and dramatically cut down the time spent debugging.

7. Professional and Ethical Issues

7.1. Ethical Concerns

Proof-of-work (PoW) based blockchains use consensus algorithms that require a significant amount of energy to operate. This is not environmentally friendly and is a major cause of concern given the current climate crisis. Although a proof-of-work blockchain was not used in this project, it is still an issue that exists within the realm of Web3.

Blockchains provide an immutable public ledger of transactions. This ultimately leads to the loss of privacy to users and could result in abuse of personal information. This also conflicts with the EU General Data Protection Regulation (GDPR) 'right to be forgotten', which states that individuals should have the right to request for their personal data to be deleted. (41)

As Web3 operates on a decentralised structure, it is not governed by law enforcement or a government, making it hard to establish accountability. This makes it increasingly difficult to investigate and prosecute criminal activity.

7.2. BCS Code of Conduct & Code of Good Practice

The British Computing Society (BCS) provides a Code of Conduct and Code of Good Practice describing a set of rules and standards that should be considered when working in computing. This was adhered to during the entirety of the project. Intellectual property has been respected; it has been explicitly stated when open-source work has been used. All processes completed within this project have been carried out lawfully and ethically.

8. Conclusion & Future Work

8.1. Conclusion

This project focused on exploring the possibilities of how Web3 and blockchain technology can revolutionise the way we interact with the web. There is still more potential to be discovered with further development.

The decentralised nature of Web3 provides a secure and transparent method for users to purchase and resell tickets. The immutability of smart contracts provides a mechanism to enforce rules and conditions regarding functionality, eliminating the risk of fraud and manipulation. The blockchain proves ownership of tickets to users and is maintained by a distributed network of nodes to provide a tamper proof ledger of transactions. These features ultimately lead to increased confidence in both the users and administrators.

In this project, Web3 has shown its capabilities in handling ticket-based transactions and provides a proof of concept to what can potentially be achieved in other domains.

8.2. Future Work

This project is not perfect and has room for improvements and new features that could be added in the future. The application has been designed with modularity in mind to allow for changes to be easily implemented to the software. Ideas for future work include:

- Using blockchain indexing services to fetch transactions. Instead of directly querying the blockchain for transaction information, which can encounter scalability limitations, an indexing service such as TheGraph (42) can be used instead. These services create indexed databases of

blockchain information that is optimised for querying to quickly retrieve information.

- Flexible pricing of tickets. This would allow event organisers to change the price of tickets as they wish. Although this would add complexity to refunds and event cancellation, the use of blockchain indexing services mentioned above would allow for easier handling of business logic involved in these processes.

- Using different distributed ledgers. In this project an Ethereum based blockchain was used, however, other new and emerging distributed ledger technologies could be experimented with. These alternatives can provider faster transaction speeds and throughput than traditional blockchain technology. A notable example of this includes Hedera Hashgraph, which can process up to 10,000 transactions per second using parallel transaction processing. (43)

- An administrator management system. A table could be added to the backend database to store multiple administrators with their login credentials. A frontend user interface could then be created to manage these users. This would allow for each administrator to have their own individual login, and allow for administrators to be easily added, modified and deleted.

- Using a session storage engine. Instead of simply storing the user sessions in the memory of our backend server, a dedicated session storage engine such as Redis (44) can be used to cache this data. This would reduce memory used to manage sessions and allow sessions to persist across server restarts.

- For data signing, the `eth_signTypedData_v4` could be used instead of the currently implemented `personal_sign` method. (40) This would improve the efficiency of processing the signature on the blockchain/ smart contract.

- Implementation of website accessibility features. W3C provides the Web Accessibility Initiative (WAI) which stipulates accessibility principles and practices to follow. (45) This would make the website accessible to as many users as possible.

- A shopping cart for users. This feature would allow for multiple tickets from different events to be added to a cart and allow for the purchase to occur in a single transaction. This would improve efficiency and reduce transaction costs.

- Event recommendations. AI can be used to take the purchases a user has made in the past and recommend events that the user may be interested in. This could increase user interaction and ticket sales.

- Customer support service. This would allow the user to contact support when they have any issues or queries, improving their user experience. Additionally, AI could also be used to provide a chatbot experience for common, simpler queries.

- A loyalty programme. This could be introduced to reward customers for their purchases. There could be different tiers, depending on the number of purchases the user has made, which determine a discount on future purchases.

- Event venue seating/standing selection. Venues often have different tiers of seating and standing areas. A feature could be added to allow

the user to select which area they would like to purchase tickets for, and the price of the ticket could vary given the area they have chosen.

- Social media integration. Buttons can be added to events pages to easily share them to social media, increasing user interaction.

- End-to-end testing of the entire system. This involves testing to validate all components of the software have been integrated correctly and meet requirements. The testing reflects real-world user scenarios, and by detecting issues in the system, they can be resolved and improve user experience.

9. References

1. Cryptopedia. Gemini. [Online] 28 June 2022. https://www.gemini.com/cryptopedia/blockchain-technology-explained/.
2. Sahu, Mayank. upGrad. [Online] 22 November 2022. https://www.upgrad.com/blog/what-makes-a-blockchain-network-immutable/.
3. Frankenfield, Jake. Investopedia. [Online] 27 September 2022. https://www.investopedia.com/terms/g/gas-ethereum.asp/.
4. Development Documentation. Ethereum. [Online] 4 October 2022. https://ethereum.org/en/developers/docs/gas/.
5. Our World in Data. [Online] 2015. https://ourworldindata.org/internet/.
6. Development Documentation. Ethereum. [Online] 26 September 2022. https://ethereum.org/en/developers/docs/web2-vs-web3/.
7. IBM. [Online] https://www.ibm.com/uk-en/topics/smart-contracts/.
8. Ethereum Improvement Proposals. Ethereum. [Online] https://eips.ethereum.org/EIPS/eip-1155/.
9. Cloudflare. [Online] https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/.
10. Hashed Out. The SSL Store. [Online] 14 September 2017. https://www.thesslstore.com/blog/google-chrome-63-tls-interception-warning/.
11. Development Documentation. Ethereum. [Online] 22 November 2022. https://ethereum.org/en/developers/docs/transactions/.

12. Lunny, Oisin. Forbes. [Online] 24 June 2019. https://www.forbes.com/sites/oisinlunny/2019/06/24/the-battle-for-15-19b-secondary-ticket-market-heats-up-with-first-europe-wide-anti-touting-law/.

13. Ethereum Improvement Proposals. Ethereum. [Online] https://eips.ethereum.org/EIPS/eip-721/.

14. MetaMask. [Online] https://metamask.io/.

15. Goerli Testnet. [Online] https://goerli.net/.

16. Infura. [Online] https://www.infura.io/.

17. C. Martin, Robert. Prentice Hall. 1 August 2008. Clean Code: A Handbook of Agile Software Craftsmanship.

18. Truffle Suite. [Online] https://trufflesuite.com/.

19. Documentation. OpenZeppelin. [Online] https://docs.openzeppelin.com/contracts/2.x/access-control/.

20. React. [Online] https://reactjs.org/.

21. Tailwind CSS. [Online] https://tailwindcss.com/.

22. Bootstrap Icons. [Online] https://icons.getbootstrap.com/.

23. React Router. [Online] https://reactrouter.com/en/main/.

24. Web3.js. [Online] https://web3js.org/.

25. useHooks. [Online] https://usehooks.com/.

26. Axios. [Online] https://axios-http.com/.

27. rosskhanas. react-qr-code. GitHub. [Online] https://github.com/rosskhanas/react-qr-code/.

28. Formik. [Online] https://formik.org/.

29. jquense. Yup. GitHub. [Online] https://github.com/jquense/yup/.

30. KarimMokhtar. React Drag and Drop Files. GitHub. [Online] https://github.com/KarimMokhtar/react-drag-drop-files/.

31. JodusNodus. React QR Reader. GitHub. [Online] https://github.com/JodusNodus/react-qr-reader/.

32. Node.js. [Online] https://nodejs.org/en/.

33. Restify. [Online] http://restify.com/.

34. Knex.js. [Online] https://knexjs.org/.

35. tdegrunt. jsonschema. GitHub. [Online] https://github.com/tdegrunt/jsonschema/.

36. nathschmidt. restify-cookies. GitHub. [Online] https://github.com/nathschmidt/restify-cookies/.

37. uuidjs. uuid. GitHub. [Online] https://github.com/uuidjs/uuid/.

38. Faker. [Online] https://fakerjs.dev/.

39. ethereum. GitHub. [Online] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1155.md#erc-1155-metadata-uri-json-schema.

40. Documentation. MetaMask. [Online] https://docs.metamask.io/guide/signing-data.html.

41. Right to erasure (‘right to be forgotten’). GDPR. [Online] https://gdpr.eu/article-17-right-to-be-forgotten/.

42. TheGraph. [Online] https://thegraph.com/en/.

43. Altcoins. Bybit Learn. [Online] 7 July 2022. https://learn.bybit.com/altcoins/what-is-hedera-hashgraph-hbar/.

44. Redis. [Online] https://redis.io/.

45. Web Accessibility Initiative. W3C. [Online] https://www.w3.org/WAI/.