Technical Specification

Krypto Kasino

Jacques Reynolds - 18413066

Scott McDonnell - 18342656

Table Of Contents

I. Introduction

Overview of the Project

Krypto Kasino aims to develop a decentralised online poker platform, leveraging blockchain technology to address the prevalent issues in traditional online poker platforms. The application will provide an easy-to-use and responsive front-end interface for users, enabling them to enjoy an engaging and secure online poker experience. The project will utilise React for front-end development, Node.js for the back-end, and Solidity to create smart contracts on the Ethereum blockchain.

The platform will allow users to connect their crypto wallets seamlessly, streamlining the process of depositing and withdrawing funds. By using cryptocurrencies, the platform offers numerous advantages over traditional payment methods, such as lower transaction fees, faster processing times, and improved privacy.

Importance of Decentralisation in Online Poker

Decentralisation plays a critical role in enhancing the online poker experience by addressing some of the key issues that have plagued traditional platforms. The following are some of the primary benefits of implementing a decentralised online poker platform:

1. **Trust and transparency:** In a decentralised online poker platform, trust and transparency are enhanced, as game outcomes and transactions are recorded on the blockchain [1]. This ensures that every action is transparent and verifiable, preventing any form of tampering or manipulation and fostering trust among players [3].
2. **Security and privacy:** Decentralised platforms leverage cryptographic techniques to secure user data and transactions. As Tschorsch and Scheuermann [5] note that decentralised platforms leverage cryptographic techniques to secure user data and transactions. Casino et al. [2] further explain that by using cryptocurrencies, players can maintain their privacy and avoid sharing sensitive personal and financial information with third parties, reducing the risk of fraud and identity theft.
3. **Lower fees and faster transactions:** Swan [4] highlights that lower fees and faster transactions can be achieved by using cryptocurrencies, as transaction fees can be significantly reduced and processing times can be shortened.
4. **Reduced regulatory restrictions:** Decentralised online poker platforms can also help reduce regulatory restrictions, as cryptocurrencies operate outside the purview of traditional banking systems [5], promoting a more inclusive and accessible online poker environment [2].
5. **Player empowerment:** Player empowerment is another advantage of decentralised platforms, as Swan [4] explains that they eliminate the need for a central authority, giving players more control over their gaming experience. In a decentralised online poker platform, players can verify game fairness, access transparent records of their transactions, and retain control over their funds [1].

By incorporating decentralisation in Krypto Kasino, the project aims to provide a secure, transparent, and user-friendly online poker experience that addresses the drawbacks of traditional platforms and sets a new standard for the industry.

II. Motivation

Issues with Traditional Online Poker Platforms

Trust and Transparency

Traditional online poker platforms rely on centralised systems to manage their games, often leading to concerns about trust and transparency [10]. Players cannot independently verify the fairness of the game outcomes, and there have been instances of fraud and cheating on some platforms [6]. Additionally, traditional platforms usually have proprietary algorithms and closed-source software, which makes it difficult for players to ascertain whether the games are truly random and unbiased. This lack of transparency can discourage players from participating and erode trust in the platform.

Payment Processing and Regulations

Traditional online poker platforms often face challenges related to payment processing and regulations. Players must typically rely on traditional payment methods, such as credit cards or bank transfers, which can be subject to high fees [8], slow processing times [7], and potential fraud. Moreover, regulatory restrictions can limit the ability of players from certain jurisdictions to participate in online poker [9]. This is particularly true in regions where online gambling is heavily regulated or even banned, making it difficult for players to access poker platforms and engage in the activity legally.

Platform Control and Player Exploitation

In centralised poker platforms, the platform operator has significant control over the games and player experience. This control can lead to potential exploitation of players by the platform, as operators may have the power to manipulate game outcomes, freeze player accounts, or confiscate funds without any recourse for the affected players. Furthermore, centralised platforms often require players to deposit funds into their platform accounts, which can be risky, as it makes players dependent on the platform's security measures and financial stability. These risks can deter players from participating in online poker and ultimately undermine the growth and sustainability of the industry.

Advantages of Blockchain Technology in Online Poker

Decentralisation

One of the main advantages of incorporating blockchain technology into online poker is decentralisation. Decentralised platforms remove the need for a central authority or intermediary, giving players more control over their gaming experience [5]. In a decentralised online poker platform, the game logic and transactions are managed through smart contracts on the blockchain, ensuring that no single entity can manipulate or control the outcomes [5]. This fosters trust among players and encourages participation in the platform.

Security and Privacy

Blockchain technology offers significant improvements in security and privacy compared to traditional online poker platforms. Cryptographic techniques are used to secure user data and transactions, protecting against data breaches and hacking attempts [2]. Additionally, by utilising cryptocurrencies, players can maintain their privacy and avoid sharing sensitive personal and financial information with third parties [2]. This reduces the risk of fraud, identity theft, and other security concerns that have plagued traditional platforms.

Fairness and Transparency

Blockchain technology can significantly enhance fairness and transparency in online poker. With game outcomes and transactions being recorded on the blockchain, every action is transparent and verifiable [1]. This ensures that all players can independently verify the fairness of the game and the integrity of the platform. Additionally, the use of open-source smart contracts allows for greater scrutiny of the game logic, ensuring that the games are truly random and unbiased [1]. This increased level of transparency builds trust among players, encouraging them to participate in the platform and promoting the growth of the online poker industry.

III. Research

Existing blockchain-based poker platforms

In recent years, several blockchain-based poker platforms have emerged to address the limitations of traditional online poker platforms. These decentralised platforms utilise blockchain technology to improve trust, transparency, security, and fairness in online poker. In this section, we will discuss some popular decentralised and non-decentralized poker platforms and contrast their features.

1. Decentralised poker platforms:
    - **Virtue Poker:** Virtue Poker is a decentralised online poker platform built on the Ethereum blockchain. It uses smart contracts to manage game logic, betting, and payouts, ensuring a transparent and trustless environment. Players can connect their crypto wallets, maintain control over their funds, and verify the fairness of the games. Virtue Poker has established partnerships with well-known poker professionals to build credibility and trust within the poker community.
    - **CoinPoker:** CoinPoker is another decentralised poker platform that leverages blockchain technology to enhance the online poker experience. The platform uses its native cryptocurrency, CHP, for transactions and has its own blockchain for increased transparency and security. CoinPoker employs a decentralised random number generation (RNG) algorithm, which ensures that the game outcomes are fair and unbiased.

2. Non-decentralized poker platforms:
    - **PokerStars:** PokerStars is one of the most popular traditional online poker platforms. While it has a large user base and a wide variety of games, it is a centralised platform that does not utilise blockchain technology. As a result, it faces some of the limitations discussed earlier, such as a lack of trust and transparency, potential security concerns, and regulatory restrictions.
    - **888poker:** 888poker is another well-established online poker platform that operates on a centralised model. Although it offers a variety of games and promotions, it shares the same limitations as other centralised platforms in terms of trust, transparency, and regulatory challenges.

In summary, the research highlights the growing trend of decentralised poker platforms leveraging blockchain technology to address the shortcomings of traditional online poker platforms. These decentralised platforms offer a more secure, transparent, and fair gaming experience, allowing players to maintain control over their funds and verify game outcomes. While centralised platforms like PokerStars and 888poker continue to dominate the market, the emergence of blockchain-based platforms such as Virtue Poker and CoinPoker demonstrates the potential for a shift towards a more decentralised and trustless online poker ecosystem.

Technologies and Frameworks Used

To develop the web application, several modern technologies and frameworks are employed. These tools facilitate the creation of a robust, responsive, and user-friendly online poker platform. The following sections discuss the role of React, Node.js, and Solidity in the development process.

React

React is an open-source JavaScript library developed by Facebook, designed for building user interfaces. It allows developers to create reusable UI components, manage the state of the application, and render the UI efficiently. React is known for its performance, flexibility, and scalability, making it an ideal choice for developing the front-end of the Web Application [14].

Key advantages of using React:

1. **Component-based architecture:** React's component-based architecture promotes code reusability and modularity, allowing developers to build complex user interfaces with less code and greater maintainability [15].
2. **Virtual DOM:** React uses a virtual DOM (Document Object Model) to update the UI efficiently. This approach minimises the number of direct manipulations to the actual DOM, resulting in improved performance and a more responsive user experience [16].
3. **Ecosystem and community support:** React has a vast ecosystem and a strong community of developers, providing access to numerous libraries, tools, and resources that can be leveraged to build a feature-rich online poker platform [14].

Node.js

Node.js is an open-source, cross-platform runtime environment that allows developers to run JavaScript code on the server-side. It is built on the V8 JavaScript engine, which provides excellent performance and supports modern JavaScript features. Node.js is particularly well-suited for building scalable and high-performance web applications, as demonstrated by its ability to handle multiple requests simultaneously without affecting performance [17]. This makes Node.js a perfect choice for the project.

Key advantages of using Node.js:

1. **Single language for both front-end and back-end:** Node.js enables developers to use JavaScript for both front-end and back-end development, simplifying the development process and reducing the need for multiple programming languages. As highlighted by Hegazy [15], this approach improves the maintainability and consistency of the codebase.
2. **Non-blocking I/O:** Node.js features a non-blocking, event-driven architecture, which allows it to handle multiple requests simultaneously without affecting performance. Tilkov & Vinoski [17] emphasise the significance of this characteristic in building highly concurrent applications, making it particularly important for an online poker platform where numerous players interact concurrently.
3. **Large package ecosystem:** Node.js has a vast package ecosystem, facilitated by the Node Package Manager (npm). Wittern et al. [18] note that this ecosystem provides access to a wide

array of libraries and tools that can be used to enhance the functionality of the website, thus enabling faster and more efficient development.

By utilising React and Node.js, our web application can be developed with a modern, high-performance tech stack that ensures a responsive, scalable, and user-friendly online poker experience.

Solidity

Solidity is a high-level, object-oriented programming language designed specifically for writing smart contracts on the Ethereum blockchain. Smart contracts are self-executing snippets of code with the terms of the contract directly written into the code. Solidity enables developers to create complex contracts with custom logic, making it a suitable choice for implementing the buy-ins and payouts for Krypto Kasino.

Key advantages of using Solidity:

1. **Turing-complete language:** Solidity is a Turing-complete language, which means it can be used to develop smart contracts with complex logic and functionality. This is essential for implementing on-chain transactions for our poker platform.
2. **Compatibility with the Ethereum Virtual Machine (EVM):** Solidity is designed to be compatible with the Ethereum Virtual Machine (EVM), enabling developers to deploy smart contracts on the Ethereum blockchain with ease.
3. **Active development and community support:** Solidity is actively developed and supported by a large community of developers, providing access to resources, libraries, and tools that can be used to build, test, and deploy smart contracts for the project.

Ethereum blockchain

The Ethereum blockchain is a decentralised, open-source blockchain platform that enables the development and deployment of smart contracts. It is the second-largest blockchain platform by market capitalization and is widely used for decentralised applications (dApps) and projects in various industries, including online gaming.

Key advantages of using the Ethereum blockchain:

1. **Decentralisation and security:** By leveraging the Ethereum blockchain, the poker game can achieve a high level of decentralisation and security. The platform can benefit from the distributed nature of the blockchain, which prevents any single entity from controlling or manipulating the system.
2. **Smart contract functionality:** Ethereum's primary innovation is the introduction of smart contracts, which allows for the creation of programmable, self-executing agreements. This enables Krypto Kasino to automate essential aspects of the platform, such as buy-ins and payouts, ensuring a trustless and transparent gaming environment.
3. **Large developer community and ecosystem:** The Ethereum blockchain boasts a large developer community and a diverse ecosystem of tools, libraries, and resources. This provides a strong foundation for building blockchain related web applications and ensures that these platforms can continue to evolve and improve over time.

By incorporating Solidity and the Ethereum blockchain, Krypto Kasino can leverage the power of decentralised technology to create a secure, transparent, and fair online poker platform, addressing the limitations of traditional centralised platforms.

Smart contracts for game logic

Smart contracts play a crucial role in the development and execution of decentralised poker games on the Ethereum blockchain. By implementing self-executing contracts with predefined rules and conditions, it is possible to automate various aspects of the poker game, such as player registration, fund management, and reward distribution. Some of the key functionalities of smart contracts in the context of a poker game can include:

1. **Game Setup:** Smart contracts can be used to create and configure new poker games, including setting game parameters such as table limits, buy-in amounts, and rules.
2. **Player Registration:** Smart contracts enable secure and transparent registration of players, ensuring that each participant meets the eligibility criteria and adheres to the platform's terms and conditions.
3. **Fund Management:** By using smart contracts, the platform can securely manage the transfer of funds between players and the platform. This includes handling deposits, withdrawals, and escrow services for game pots, which helps prevent fraud and ensure fair gameplay.
4. **Reward Distribution:** Smart contracts can automate the distribution of rewards to winning players, ensuring that payouts are accurate and timely, while also reducing the potential for human error or manipulation.
5. **Game Fairness:** Smart contracts can incorporate provably fair algorithms to determine game outcomes and ensure the randomness and integrity of the poker games. This helps to build trust among players and encourages participation in the platform.

Key challenges in implementing decentralised poker

While decentralised poker platforms offer numerous advantages over traditional systems, there are several challenges that developers must overcome in order to create a seamless and engaging user experience:

1. **Scalability:** One of the primary challenges in implementing decentralised poker games is ensuring that the platform can handle a large number of simultaneous users and transactions without compromising performance. This requires optimising the underlying blockchain infrastructure, as well as the front-end and back-end systems, to support high throughput and low latency.
2. **Legal and Regulatory Compliance:** Decentralised poker platforms must navigate a complex landscape of legal and regulatory requirements, which can vary significantly between jurisdictions. This may involve obtaining licences and permits, adhering to anti-money laundering (AML) and know-your-customer (KYC) regulations, and ensuring compliance with local gambling laws.
3. **User Experience:** To attract and retain players, decentralised poker platforms must provide a user experience that rivals or surpasses that of traditional platforms. This includes creating

intuitive user interfaces, responsive designs, and seamless onboarding processes, as well as ensuring that the platform is accessible and easy to use for players of all skill levels.

4. **Security:** Decentralised poker platforms must ensure that user data, transactions, and game outcomes are secure and resistant to attacks. This involves implementing robust security measures, such as encryption, secure wallet integrations, and ongoing monitoring and testing of the platform's infrastructure.
5. **Adoption and Network Effects:** For decentralised poker platforms to succeed, they must attract a critical mass of users and create a vibrant ecosystem of players. This requires effective marketing and community-building efforts, as well as the development of partnerships and collaborations with other stakeholders in the online poker industry.

By addressing these opportunities and challenges, we decided to create a poker platform that handles buy-ins and payouts with blockchain transactions. We hope to showcase the advantages of using blockchain technology in online poker.

IV. Design

System architecture

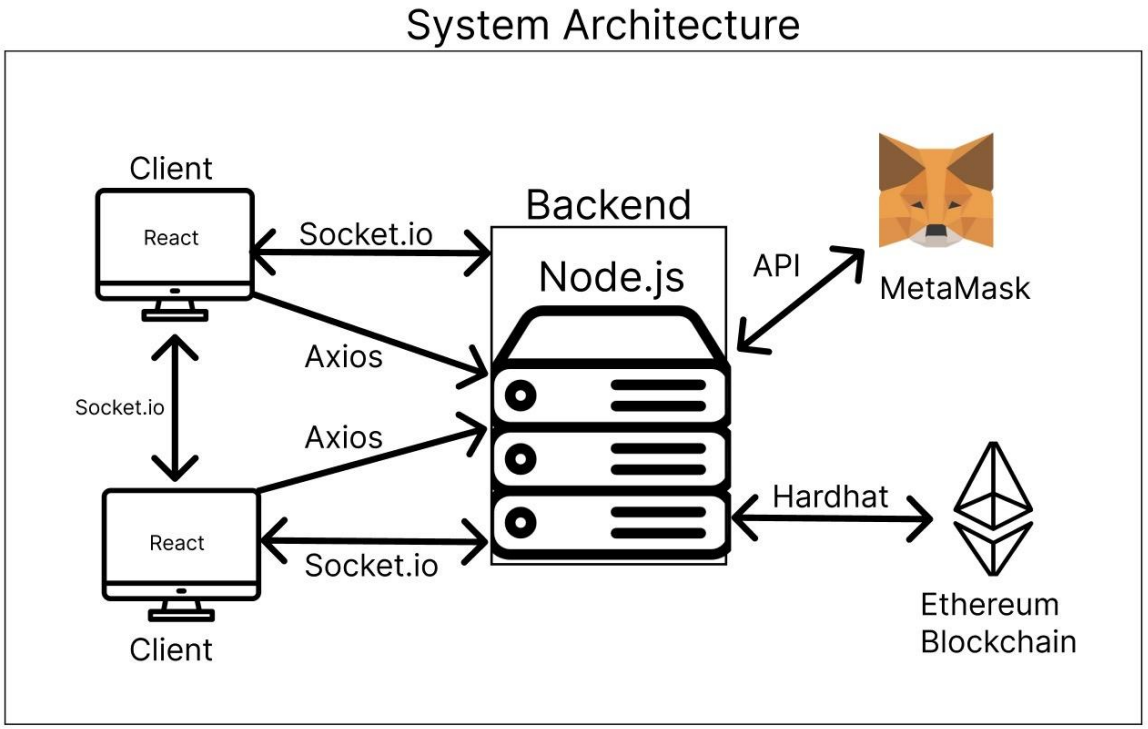

Diagram 1: System Architecture

The Krypto Kasino system architecture is designed to deliver a seamless and engaging experience for players while ensuring security, scalability, and performance. It consists of a front-end design, which

provides the user interface, and a back-end design, which handles the underlying logic for the poker game and communication with the Ethereum blockchain.

Frontend design

The front-end design of the web app focuses on delivering an engaging, user-friendly, and visually appealing experience for the players. It is developed using the React JavaScript library and TailwindCSS for styling, a popular choice for building modern, responsive, and scalable user interfaces. Key components of the front-end design include the website home page which consists of the navbar, welcome, join poker game and footer React components. The join poker game leads to the poker lobby page which is where the game is played.

Backend design

The back-end design of Krypto Kasino is responsible for managing the platform's core functionality, such as game logic, player management, and communication with the Ethereum blockchain. The back-end is developed using Node.js, a popular and efficient JavaScript runtime built on Chrome's V8 JavaScript engine. The following are the primary components of the back-end design:

1. **Server-side Logic:** The back-end handles the server-side logic, including player authentication, game state management, and the execution of poker logic. This ensures that the platform operates smoothly and consistently, providing a reliable and enjoyable experience for the players.
2. **Ethereum Blockchain Integration:** The back-end communicates with the Ethereum blockchain to manage smart contracts, process transactions, and interact with decentralised applications (dApps). This enables the platform to leverage the benefits of blockchain technology, such as trust, transparency, and security.
3. **API Development:** The back-end design includes the development of APIs (Application Programming Interfaces) that allow the front-end components to interact with the back-end services. These APIs facilitate data exchange between the user interface and the server-side logic, ensuring a seamless and responsive user experience.

By implementing a comprehensive system architecture with well-designed front-end and back-end components, Krypto Kasino aims to create an engaging and enjoyable environment for players, showcasing the benefits of using blockchain technology in online poker.

Smart contracts design

Smart contracts are an integral part of the poker game, as they enable the automation of various game aspects while providing transparency, security, and trust. The smart contract design encompasses the game setup, player registration, fund management, and reward distribution, as detailed in the essay's research section. In this section, we will focus on the design of smart contracts for buy-ins and payouts.

Payouts

Payout mechanisms in Krypto Kasino are managed through smart contracts, ensuring that funds are handled securely and fairly throughout the gaming process.

- **Payout Distribution:** Once the winner(s) have been determined, the smart contract automatically distributes the pot to the appropriate player(s). This automated payout process eliminates the need for manual intervention, reducing the potential for human error or manipulation.

By incorporating smart contracts for buy-ins and payouts, the poker game ensures that the gaming experience is secure, transparent, and fair, fostering trust among players and promoting the advantages of blockchain technology in online poker.

User Interface and User Experience Design

Design Principles

An essential aspect of the web application is creating a user interface and user experience that is engaging, intuitive, and enjoyable for players. The design should prioritise ease of use and accessibility, enabling players of all skill levels to participate in the platform. In this section, we will focus on responsive web design as a key component of the user interface and user experience design, and the use of TailwindCSS and React to achieve it.

Responsive web design refers to the practice of designing and developing a website that adapts its layout and user interface elements to different screen sizes, resolutions, and devices. Implementing responsive web design in KryptoKasino ensures that the platform provides an optimal user experience across a variety of devices, including desktop computers, laptops, tablets, and smartphones. TailwindCSS, a popular utility-first CSS framework, is used in conjunction with React to create responsive, customizable, and visually appealing UI components. Key aspects of responsive web design in the web application include:

- **Fluid Grid Layout:** The application utilises a fluid grid layout that dynamically adjusts the size and position of UI elements based on the user's screen size and resolution. TailwindCSS provides a flexible and customizable grid system, which is used in combination with React to ensure that the content is presented in a visually appealing and organised manner, regardless of the device being used.
- **Flexible Media:** Images, videos, and other media elements are designed to be flexible, automatically resizing and repositioning themselves based on the available screen space. TailwindCSS offers responsive utilities to handle media elements, ensuring that they remain clear and visible, enhancing the user experience and ensuring that players can fully engage with the platform.
- **Adaptive UI Components:** The user interface components, such as buttons, menus, and input fields, are designed to adapt to different screen sizes and devices. TailwindCSS provides a comprehensive set of utility classes, which, when used with React, ensures that the UI elements remain easy to interact with, regardless of the user's device or input method (e.g., touch screen or mouse).

- **Cross-Browser Compatibility:** KryptoKasino is developed to be compatible with a wide range of web browsers, including popular choices such as Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge. The combination of TailwindCSS and React ensures that the platform is accessible and functional for users, regardless of their preferred browser.
- **Performance Optimization:** The responsive web design is optimised to ensure fast loading times and smooth performance on various devices and network conditions. Techniques such as image optimization, code minification, and caching are employed, along with the efficient performance of TailwindCSS and React, to reduce the application's resource requirements and enhance the overall user experience.

By implementing responsive web design with the help of TailwindCSS and React in Krypto Kasino, the platform can deliver a seamless and enjoyable user experience across a wide range of devices and screen sizes. This not only makes the platform more accessible to a diverse audience of players but also showcases the versatility and adaptability of blockchain-based online poker applications.

Website Design

In designing the user interface of Krypto Kasino, the design principles and practices mentioned above have been followed to enhance the user experience and ensure a visually appealing, intuitive, and engaging platform. We will explain the design principles followed for the welcome component on the home page as well as the poker lobby component, where users play their games of poker:

- **Consistency:** To ensure that UI elements share consistent styles, such as padding, borders, and text attributes we made use of commonStyles constants in the code. This promotes a cohesive and polished appearance throughout the platform, making it easier for users to understand and interact with the interface.
- **Hierarchy and typography:** The use of different font sizes, weights, and colours in the welcome component helps establish a clear visual hierarchy, guiding users' attention to the most important information. The prominent heading "Play Poker With Crypto!" on the home page immediately captures the user's attention, while the supporting text provides additional context and details. Similarly, in the poker lobby the "Krypto Kasino" title stands out with its larger font size and bold weight, while the pot amount is displayed in a smaller and lighter text style.
- **White space:** Ample white space has been incorporated into the design, as seen in the padding and margins surrounding the text and buttons. This prevents the interface from feeling cluttered and allows users to easily focus on the content.
- **Visual cues and feedback:** The "Connect Wallet" button features a hover effect that changes its background colour when the cursor hovers over it, providing visual feedback and indicating interactivity to users. This encourages users to engage with the platform and complete the desired action. Additionally, conditional rendering is used to show or hide certain elements based on the game state and the user's actions, ensuring that users are only presented with relevant and actionable options, ie. check, raise and fold.
- **Responsiveness:** The use of the TailwindCSS utility classes, such as sm:, md:, and w-, ensures that the welcome component adapts its layout and appearance based on the user's screen size and device. This ensures an optimal user experience across different devices and screen sizes.

- **Layout and organisation:** The welcome component makes use of a grid layout to organise the features and benefits of the platform, such as "Secure," "Reliable," and "Easy." This layout presents the information in a visually appealing and easily digestible format, allowing users to quickly understand the platform's advantages. Furthermore, the lobby component is well-organised, with distinct sections for the chair, community cards, and player actions. This clear separation of elements enables users to quickly understand the interface and focus on their gameplay.

By adhering to these design principles and practices, the Krypto Kasino's home page effectively conveys the platform's purpose and value proposition while providing an engaging and visually appealing user experience. This sets the foundation for a positive user experience throughout the rest of the platform.

V. Implementation

Front-end

React Components

In this section, we will explore the various components of the React application. The app consists of several key components:

- App
- Card
- Chair
- CommunityCard
- Lobby
- Navbar
- Player
- Poker
- Welcome

We will examine some of these components, detailing their functionalities, states, event listeners, and interactions with other components.

Additionally, we will delve into how the application connects to the Ethereum network and smart contracts, which enable cryptocurrency transactions for the poker game. This comprehensive overview will provide insight into the inner workings of Krypto Kasino with React and blockchain technology.

App

This code defines the App component, which is the main component of the React application. The component imports various components such as Welcome, Poker, Navbar, Footer, Lobby, Room, Play,

Learn, PokerTable, and Chair. It also imports React, ReactDOM, and components from the react-router-dom package, which allows for client-side routing. The Home component is defined as a functional component that returns the Welcome and Poker components wrapped in a div element with a gradient-bg-welcome CSS class.

The App component returns a div element with a h-full CSS class that contains the Navbar, Routes, and Footer components wrapped in a div element with an app-container gradient-bg-welcome CSS class.

```
18  const App = () => {
19
20    return (
21      <div className="h-full">
22        <div className='app-container gradient-bg-welcome'>
23          <Navbar />
24          <Routes>
25            <Route path="/" element={< Home />} />
26            <Route path="/game" element={<Lobby />} />
27            <Route path="/play" element={< Poker />} />
28            <Route path="/learn" element={< Learn />} />
29            <Route path="/login" element={< Login />} />
30          </Routes>
31          <Footer />
32        </div>
33
34      </div>
35    );
36  };
```

The Routes component defines the different routes for the application using the Route components, which are defined by a path and an element to render when the path matches. The available routes are the Home, Lobby, Poker and Learn components.

Card

This is the component that models a playing card. It takes two props: id and card. id is a string representing the unique identifier of the card, and card is an object representing the rank and suit of the card to be rendered. The component first checks if a card object is passed as a prop. If it is, the component renders the card's rank and suit using JSX. The rank is rendered with a text element that is styled based on the colour of the suit. The suit is also rendered as text, and its colour is also based on the colour of the suit. If a card object is not passed as a prop, the component renders the id prop instead.

The component returns a div element that renders the card content, with styling based on its position and the content it holds. The component also has two div elements that are positioned absolutely and rotated to give the appearance of a tilted border. The Card component is then exported for use in other React components.

Chair

This component accepts two props: id and card. The component checks if the card prop is defined, and if so, it renders the rank and suit of the card inside a div. The colour of the rank and suit text changes dependently. if the card prop is not defined, the component just renders the id prop.

```
78  function renderChair(seatNum) {
79    if (game && game.players) {
80      const player = game.players.find((p) => p.seatNum === seatNum);
81      const chairStyles = {
82        1: "row-start-1 row-end-2 col-start-2 col-end-2",
83        2: "row-start-1 row-end-2 col-start-5 col-end-5",
84        3: "row-start-2 row-end-3 col-start-6 col-end-6",
85        4: "row-start-5 row-end-5 col-start-6 col-end-6",
86        5: "row-start-5 row-end-5 col-start-1 col-end-1",
87        6: "row-start-2 row-end-3 col-start-1 col-end-1",
88      };
```

The component returns a div element with some styles, which contains the cardContent and two child div elements with border styles to create a border effect around the card. The Card component is then exported as the default export of this module. Each chair contains its own individual positioning to make sure they fit around the poker table evenly.

CommunityCard

CommunityCard is a component that renders a community card used in a poker game. It takes two props, the id of the card, and the card object, which contains the rank and suit of the card. The component first checks if the card prop exists.. If it does, it renders the cardContent that displays the rank and suit of the card. If not, it returns an empty string. The cardContent is a div that contains two elements: a paragraph with the rank of the card and another paragraph with the suit of the card. The text-3xl and text-4xl classes are used to style the rank and suit text to be a certain size, while the text-black and text-red-600 classes are used to set the text colour to black or red depending on the suit.

The component returns a div element that contains the cardContent. It also contains two div elements that are positioned absolutely to create a border around the card that is rotated at a 15-degree angle. The style prop is used to set the width, height, and border radius of the card and the transform property is used to rotate the border.

Lobby

The Lobby component is a React functional component that renders a poker game lobby. The component is responsible for setting up various states and event listeners for the poker game lobby. It uses the useEffect hook to manage side-effects like setting up a socket.io client and attaching event listeners to various socket events. The component also defines various functions
which are called to update the game state or handle user actions.

The component also renders various child components like **CommunityCard**, **Player**, and **Chair**, which are used to display game information like community cards, player details, and seat information respectively.

The Lobby component uses a number of state variables to keep track of the game state, including:

- **game:** A state variable that stores the game object received from the server. This object contains information about the game, including the players, their chips, and the community cards.
- **pot:** A state variable that stores the current size of the pot.
- **handStarted:** A state variable that stores whether the current hand has started.
- **communityCards:** A state variable that stores the current community cards.
- **actionsTaken:** A state variable that stores the number of actions taken in the current round.
- **betMade:** A state variable that stores whether a bet has been made in the current round.
- **currentPlayerIndex**: A state variable that stores the index of the current player in the game.players array.
- **playerData:** A state variable that stores information about the current player, including their name, chips, and socket ID.
- **loading**: A state variable that stores whether the component is currently loading.

The component also uses a number of socket events to update the game state and handle user actions. These events include:

- **game_updated:** This event is triggered when the game state is updated on the server. The component updates its game state variable with the new game object.
- **bet_placed**: This event is triggered when a player places a bet. The component updates the pot state variable with the new pot size, sets actionsTaken to 0, and sets betMade to true.
- **check_made:** This event is triggered when a player checks. The component sets actionsTaken to 0 and sets betMade to false.
- **player_joined:** This event is triggered when a player joins the game. The component adds the new player to the game.players array.
- **player_left:** This event is triggered when a player leaves the game. The component removes the player from the game.players array.
- **hand_started:** This event is triggered when a new hand is started. The component sets handStarted to true, sets actionsTaken to 0, and sets betMade to false.
- **community_cards_dealt:** This event is triggered when the community cards are dealt. The component updates the communityCards state variable with the new community cards.
- **player_turn**: This event is triggered when it is a player's turn to act. The component updates the currentPlayerIndex state variable with the index of the current player.

The Lobby component also defines a number of helper functions that are used to update the game state or handle user actions. These functions include:

- **activePlayers**: This function takes a game object as input and returns an array of active players (i.e., players who have not folded or gone all-in).
- **countActivePlayers**: This function takes a game object as input and returns the number of active players.
- **countAllIn**: This function takes an array of players as input and returns the number of players who have gone all-in.
- **getPlayerAddress:** This function takes a player object as input and returns the player's Ethereum address (which is used to send/receive cryptocurrency in the game).

- **handleLeaveGame:** This function is called when a player leaves the game. It updates the game state (i.e., removes the player from the game) and emits an event to update the game state on the server.
- **handleLeaveGameWithLoader**: This function is similar to handleLeaveGame, but it displays a loading screen while the game state is being updated.
- **handleRaise**: This function is called when a player raises the bet. It updates the game state (i.e., increases the pot and sets the betMade flag to true) and emits an event to update the game state on the server.
- **handleCall:** This function is called when a player calls a bet. It updates the game state (i.e., increases the pot) and emits an event to update the game state on the server.
- **handleFold:** This function is called when a player folds their hand. It updates the game state (i.e., sets the player's hasFolded flag to true) and emits an event to update the game state on the server.
- **handleCheck:** This function is called when a player checks their hand. It updates the game state (i.e., sets the betMade flag to true) and emits an event to update the game state on the server.
- **handleAllIn:** This function is called when a player goes all-in. It updates the game state (i.e., sets the player's isAllIn flag to true) and emits an event to update the game state on the server.
- **resetHand:** This function is called when a new hand is dealt. It updates the game state (i.e., resets all player properties) and emits an event to update the game state on the server.
- **updateGame:** This function is called to update the game state (i.e., sets the game state using the setGame hook).

Navbar

This component defines the navigation bar for the website. It imports a number of icons and images and uses them to create the navigation bar. The component defines a **NavbarItem** component that takes two props: **title** and **classProps**. This component is used to create the navigation items in the navigation bar. The main **Navbar** component defines a state variable **toggleMenu** that is used to toggle the visibility of the mobile navigation menu.

The component returns a **nav** element that contains the logo of the website and a **ul** element that contains the navigation items. The **ul** element is hidden on mobile devices and can be toggled using the **toggleMenu** state variable. When the mobile menu is opened, a new **ul** element is created and displayed that contains the navigation items in a vertical layout. It defines styles for the navigation bar using Tailwind CSS classes. Overall, the **Navbar** component provides a responsive and user-friendly navigation experience for the website.

Player

This defines a simple React functional component named Player. This component is used to display a player's name and their current hand of cards. It takes two props as input: **playerName**, which is a string representing the player's name, and playerHand, which is an array of objects representing the player's current hand of cards.

The component first renders the player's name as an **<h3>** tag, and then renders the player's hand of cards as a set of ** **s with the class name "card". If the player's hand is empty, the component instead displays the message "No cards". Inside the **playerHand.map()** function, the component

iterates over each card object in the **playerHand** array, and renders the card's rank and suit as plain text inside the corresponding ** ** with the class name "card". Overall, the Player component is a simple and reusable component that can be used to display a player's information in a poker game or any other card game.

Poker

This is a React component that serves as the main page for the Poker game. It displays an Ethereum wallet address and a button that, when clicked, opens a modal that allows the user to join the game by entering their name and the amount of cryptocurrency they want to buy in with. The component initialises the Ethereum provider using Web3 Provider from ethers.js, and connects to the smart contract using the contract address and ABI.

The component also has a state that manages whether the modal is open or not, whether the page is loading or not, and whether the user has joined the game or not. When the user successfully joins the game, the component updates the state to reflect that, and the player is redirected to the game page. The **joinGame** function is used to execute the buyIn function in the smart contract. It takes the user's name and crypto amount as inputs, converts the amount to Wei, and sends it to the smart contract as part of the transaction. Once the transaction is confirmed, the player's data is updated in the state and the gameJoined state is set to true.

The **handleNameChange** and **handleCryptoAmountChange** functions update the name and cryptoAmount state variables respectively based on the user's input in the modal. The **handlePlayButtonClick** function sets the **showModal** state variable to true, which opens the modal when the user clicks the "Play Game" button.

The **fetchPlayerData** function fetches the player data from the smart contract using the provider and contract instance. It retrieves the player's joined status, balance, and name, and converts the balance from Wei to Ether.

This component serves as the entry point for the Poker game, allowing users to join the game and initialising the necessary connections to the Ethereum network and the smart contract.

React Router

We use React Router to manage client-side routing. React Router is a powerful and flexible library that allows us to create a complex single-page application with ease. It enables us to map URLs to different components in our application, ensuring that your users can navigate through your application seamlessly.

In **App.jsx**, React Router is imported from 'react-router-dom' and used in the **App** component. The **BrowserRouter** alias, **Router**, is used to wrap the application, allowing you to define routes using the **Routes** and **Route** components.

A few routes are defined within the **Routes** component, and each route maps to a specific URL path. For instance, the root path ("/") maps to the **Home** component, while "/game" maps to the **Lobby**

component. As users navigate through the application, the corresponding components are rendered based on the URL path.

The **App** component includes a **Navbar** component, which typically contains navigation links to different parts of the application. Below the **Navbar** component, the **Routes** component is used to define different routes for the application. When a user clicks on a navigation link, React Router updates the URL, and the appropriate component associated with the route is rendered.
By defining the routes and associating them with their respective components, we can ensure that our users can easily navigate our single-page application without the need for full-page refreshes.

State management

React state management is a critical aspect of developing a React application. State management allows us to manage and share data across components, making our application more dynamic and responsive to user interactions.

We will use the functional component Lobby as our example of how we use state management. The **useState** and **useEffect** hooks are used to manage the state of the application and the side effects that depend on the state.

The **useState** hook is used to declare state variables and their associated setters. In this case, various state variables such as **game**, **currentPlayerIndex**, **timer**, **betAmount**, **startIndex**, **raises**, **actionsTaken**, **gameStarted**, **handStarted**, **communityCards**, **currentActionIndex**, **pot**, **betMade**, and **loading** are declared.

The **useEffect** hook is employed to handle side effects, such as subscribing to events and performing actions when certain dependencies change. In this example, **useEffect** is used to set up event listeners for the Socket.IO connection and handle incoming messages from the server, such as "game_data", "new_player", "winning_hand", "hand_reset", and many others.

When these events are received, the component's state is updated accordingly, using the respective setter functions provided by the **useState** hook. For example, when the "game_data" event is received, the **setGame** function is called to update the **game** state variable with the new data.

Additionally, the **useEffect** hook takes care of cleaning up resources when the component is unmounted or when the dependencies change. The cleanup function is defined to disconnect the Socket.IO connection, ensuring no memory leaks occur.

The **Lobby** component also contains several event handler functions such as **handleRaise**, **handleCall**, **handleFold**, **handleCheck**, and **handleAllIn**. These functions are responsible for responding to user interactions, updating the component's state, and emitting events to the server using the Socket.IO connection.

Back-end

Poker Game Logic

The gameLogic.js file contains the main logic and functions used in the poker application. This file exports several The **gameLogic.js** file contains the main logic and functions used in the poker application. This file exports several functions that handle game creation, player management, and game progression. The main exported functions are:

1. **newGame**: Initialises a new game object with the appropriate data structure, properties, and a shuffled deck.
2. **addPlayer**: Adds a new player to the game.
3. **removePlayer**: Removes a player from the game.
4. **dealHoleCards**: Deals two hole cards to each player in the game.
5. **addCommunityCards**, **dealFlop**, **dealTurn**, and **dealRiver**: These functions deal community cards (flop, turn, and river) and update the game state accordingly.
6. **fold**, **call**, **check**, and other player action functions: Handle various player actions, such as folding or placing bets, and update the game state and player properties accordingly.
7. **determineWinningHand**: Determines the winning player(s) based on the players' hands.
8. **checkForWinner**: Determines the winner(s) of the current hand and distributes the pot accordingly.
9. **compareHands**: Compares two poker hands and determines which one is the winner.
10. **getCardCounts**, **getKickers**, and **getMostFrequentRank**: These helper functions are used for evaluating poker hands and determining the frequency of each card rank, the kicker cards, and the most frequent rank in a hand, respectively.
11. **isGameOver**: Determines if the game is over based on the game state and player properties.
12. **placeBets**, **allIn**, and **handleBet**: Handle the betting process and update the game state and player properties accordingly.
13. **endOfHand**: Handles the end of a hand, updating the game state and preparing for the next hand.

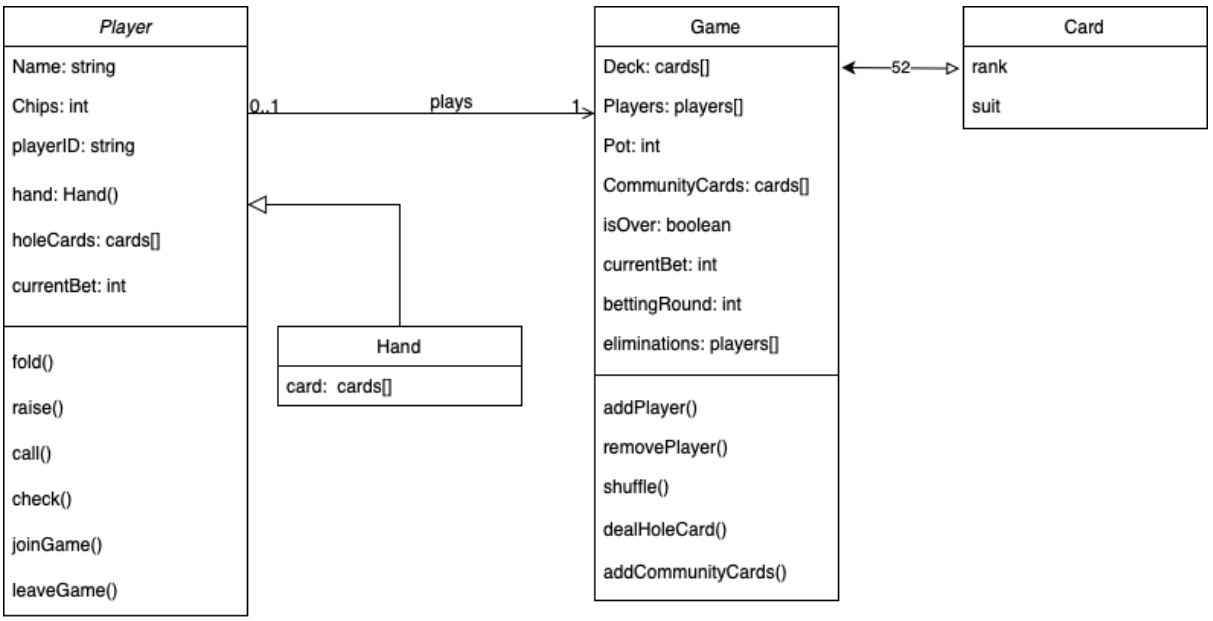

Diagram 2: Class Diagram

WebSockets

The **socketUtils.js** file contains the implementation of the back-end logic for the poker game. It is responsible for handling various events emitted by the front-end and updating the game state accordingly. This file uses Node.js with the Socket.IO library for real-time communication between the server and clients. It also interacts with a smart contract on the Ethereum blockchain.

The **socketUtils.js** file exports two main functions:

1. **exports.sio**: This function initialises a new Socket.IO server with the given options, including transport and CORS settings.
2. **exports.connection**: This function sets up the event listeners for the Socket.IO server.

The **exports.connection** function is responsible for managing the main socket connection events and game events in the poker application. This function is exported from the **socketUtils.js** file and is invoked by the main server. The function takes **io** as an argument, which is an instance of the **socket.io** library.
Inside the **exports.connection** function, the following events and actions are defined:

1. **Connection:** When a new user connects to the server, a "connection" event is emitted. The function listens for this event and sets up various event listeners for the connected user.
2. **Message:** This event is emitted when a user sends a message through the socket connection. The server listens for this event and logs the received message.
3. **Join_room:** This event is emitted when a user wants to join a specific room. The server listens for this event and adds the user to the specified room.
4. **New_game:** This event is emitted when a new game is created. The server listens for this event and initialises a new game using the **gameLogic.newGame()** function.
5. **Join_game:** This event is emitted when a user wants to join an existing game. The server listens for this event and adds the player to the game using the **gameLogic.addPlayer()** function.
6. **Deal_cards**, **deal_flop**, **deal_turn**, and **deal_river** events: These events are emitted when the game progresses through different stages. The server listens for these events and updates the game state using the appropriate **gameLogic** functions.
7. **Get_winning_hand**: This event is emitted when the winning hand must be determined. The server listens for this event and calculates the winner using the **gameLogic.checkForWinner()** function.
8. **Reset_game**: This event is emitted when the current hand is finished, and the game state must be reset. The server listens for this event and resets the game using the **gameLogic.endOfHand()** function.
9. **Fold**, **place_bets**, **call**, **check**, **all_in**, and **call_all_in** events: These events are emitted when a player performs an action in the game, such as folding or placing bets. The server listens for these events and updates the game state using the appropriate **gameLogic** functions.
10. **Leave_game**: This event is emitted when a player wants to leave the game. The server listens for this event and processes the player's exit, including updating their balance in the smart contract and removing them from the game using the **gameLogic.removePlayer()** function.

11. **Disconnect**: This event is emitted when a player disconnects from the server. The server listens for this event, removes the player from the game using the **gameLogic.removePlayer()** function, and updates the game state.

These events and actions ensure that the poker game runs smoothly and that the game state is updated correctly based on the actions of the connected players.

Solidity smart contract deployment on the Ethereum network

Deployment

Prerequisites

- Node.js runtime environment
- ethers.js library installed
- Configured Ethereum network provider

Deployment Script

The deployment script, **deploy.js**, follows these steps:

1. Retrieve the deployer's Ethereum account using the **ethers.getSigners()** function.
2. Log the deployer's address and balance to the console.
3. Get the contract factory for the "PokerGame" smart contract using **ethers.getContractFactory()**.
4. Deploy the contract to the Ethereum network using **pokerGame.deploy()**.
5. Wait for the deployment transaction to be confirmed and the contract to be deployed using **pokerGame.deployed()**.
6. Log the deployed contract's address to the console.
7. Handle any errors that may occur during the deployment process.

Execution

To deploy the "PokerGame" smart contract on the Ethereum network, follow these steps:

1. Ensure that Node.js and the ethers.js library are installed.
2. Configure the Ethereum network provider in the deployment script.
3. Open a terminal or command prompt and navigate to the directory containing the **deploy.js** script.
4. Execute the script by running the command: **node deploy.js**.
5. Monitor the console output for the deployer's address, balance, and the deployed contract's address.
6. Once the script finishes executing without errors, the "PokerGame" contract is successfully deployed on the Ethereum network.

Overview

By following the steps outlined in this technical specification and executing the **deploy.js** script, we deployed the "PokerGame" smart contract on the Ethereum network using the ethers.js library.

Smart Contract

The contract is named "PokerGame" and is defined with the pragma directive **pragma solidity ^0.8.0** to specify the compiler version and licensing information.

Player Struct

The contract includes a struct named "Player" to represent a player in the game. It has the following properties:

- **joined**: A boolean value indicating whether the player has joined the game.
- **balance**: A uint256 value representing the player's balance (in Wei).
- **name**: A string value representing the player's name.

State Variables

The contract declares the following state variables:

- **players**: A mapping that associates player addresses with their corresponding Player struct.
- **playerList**: An array of player addresses, used for tracking the players who have joined the game.

buyIn Function

The **buyIn** function allows players to join the game by sending a positive amount of Ether as a buy-in. It takes a string parameter **_name** to set the player's name. The function performs the following steps:

- Checks that a positive amount of Ether has been sent.
- Updates the player's balance and join status in the **players** mapping.
- Adds the player's address to the **playerList** array.

exitGame Function

The **exitGame** function allows players to exit the game and withdraw their winnings or cover their losses. It takes an **int256** parameter **winnings**, which can be positive (winnings), negative (losses), or zero (no change in balance). The function performs the following steps:

- Handles positive winnings by transferring the corresponding amount of Ether from the contract to the player.
- Handles negative winnings (losses) by deducting the amount from the player's balance and sending it back to the player.
- Handles zero winnings by simply returning the player's balance to them.
- Removes the player from the **playerList** array and updates their join status in the **players** mapping.

removePlayer Function

The **removePlayer** function is a private helper function used by **exitGame** to remove a player from the **playerList** array and update their join status in the **players** mapping.

Overview

The "PokerGame.sol" Solidity smart contract provides the functionality to manage players in a poker game, allowing them to join the game, exit the game with their winnings or losses, and handle player

balances. It serves as the backend logic for a decentralised poker application on the Ethereum network.

Integration of front-end and back-end components

The front-end integrates with the back-end through socket communication. Here's an overview of how the integration works:

1. **Front-end:** The front-end establishes a socket connection with the back-end using Socket.IO. It sends and receives events to communicate with the server.
2. **Back-end**: The back-end handles the socket events emitted by the front-end and performs the necessary logic and operations.
3. **Socket events:** The front-end emits socket events to the back-end to perform actions or request information. Examples of socket events in your code include "new_game", "join_game", "deal_cards", "fold", "place_bets", etc.
4. **Event handling:** The back-end listens for socket events emitted by the front-end and handles them using event handlers. Each event handler performs the corresponding game logic or operations.
5. **Game logic:** The back-end contains the game logic implementation. It includes functions for starting a new game, joining a game, dealing cards, handling player actions (fold, bet, call, etc.), determining winners, and updating the game state.
6. **Updating clients:** After processing a socket event, the back-end sends the updated game state or relevant data back to the front-end using socket communication. The front-end receives the updated data and reflects it in the user interface.
7. **Broadcasting**: The back-end broadcasts the updated game state to all connected clients using the socket.io io.emit() method. This ensures that all clients receive the same game state and stay in sync.
8. **Disconnection handling:** When a client disconnects, the back-end removes the corresponding player from the game and updates the game state. It then broadcasts the updated game state to all connected clients.
9. **Integration with Smart Contract**: The back-end interacts with the PokerGame smart contract deployed on the Ethereum network. It includes functions for calling smart contract methods, such as exitGame and removePlayer, to handle player exits and other related operations.

The front-end and back-end work together through socket communication to create a real-time multiplayer poker game. The back-end handles the game logic and operations, while the front-end provides the user interface and communicates with the server using socket events.

How the User Interacts with Krypto Kasino

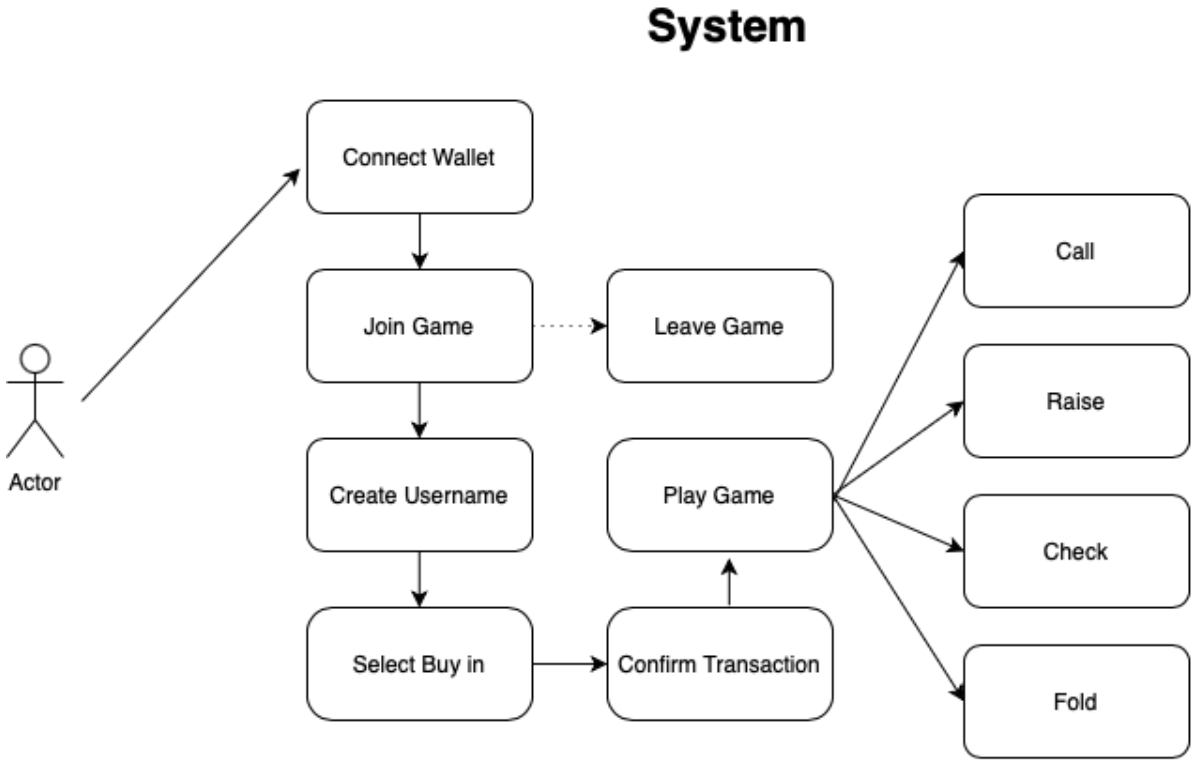

Diagram 3: Use Case

Our decentralised poker platform has been designed to offer a seamless and user-friendly experience for players. In this section, we will outline the process a user goes through while interacting with the platform, starting from connecting their wallet to eventually playing the game and exiting.

Connecting the Wallet and Launching the Game

Upon landing on the home page, users are presented with a "Connect Wallet" button. Clicking this button allows them to connect their crypto wallet to the platform, ensuring a secure and seamless integration. Once the wallet is successfully connected, the platform conditionally renders the "Play Game" button, which the user can click to proceed to the next step.

Joining the Game

After clicking the "Play Game" button, a modal appears on the screen, prompting the user to enter their desired game name and the crypto amount they wish to use as their buy-in. By clicking the "Join

Game" button, the user triggers the smart contract, which calls a function that handles players buying into the game. The smart contract processes the user's request, and once it is completed, the user is directed to the game lobby page.

Selecting a Seat and Starting the Game

In the game lobby, the user can choose a chair at the poker table by clicking on it. The game begins when at least two players are seated at the table. If there are not enough players, the user must wait for more players to join, as a minimum of two players is required to start a game.

Playing the Game and Exiting

Players engage in the poker game until only one player remains. During the game, a player can choose to leave by clicking the "Leave Game" button. This action triggers the smart contract again, but this time it calls a function that handles a player exiting the game. Additionally, if a player is eliminated from the game, the smart contract is triggered to manage the player's exit. In both scenarios, the platform ensures a smooth and secure process for players to enter and exit the game.

In conclusion, our decentralised poker platform offers a user-friendly and engaging experience for players, from connecting their wallet to playing and exiting the game. By leveraging blockchain technology and smart contracts, the platform ensures a secure, transparent, and trustworthy environment for users to enjoy the game of poker.

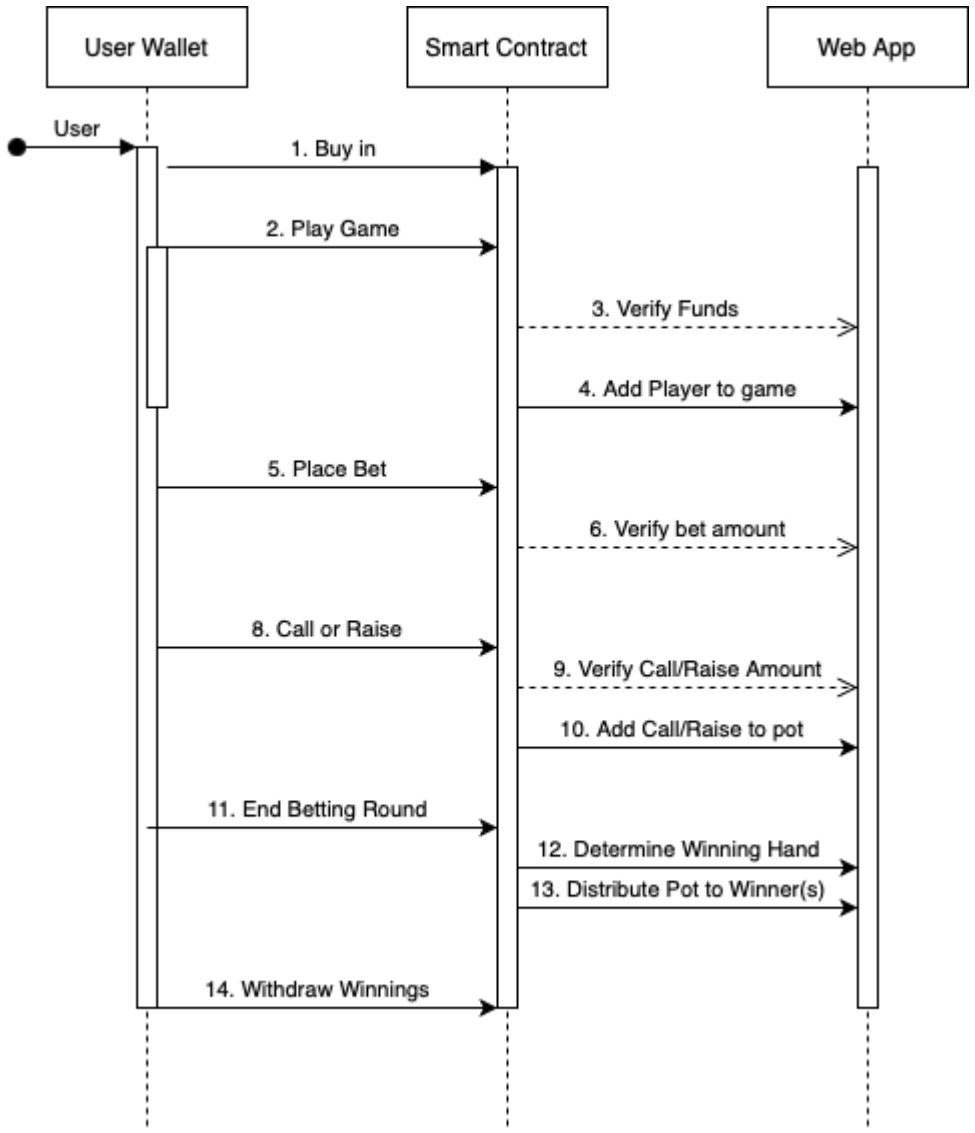

Diagram 4: Sequence Diagram

VI. Sample Code

Key snippets from React components

In this section, we will provide sample code snippets from various parts of our decentralised poker platform. These snippets showcase the key aspects of our implementation, including the React components, the Node.js server code for handling requests, and the Solidity smart contract code buy-ins and payouts.

1. Connect Wallet Button

```
{!connectedAccount && (
    <button
        type="button"
        onClick={connectWallet}
        className="flex flex-row justify-center items-center my-5 bg-[#2952e3] p-8 rounded-full cursor-pointer hover:bg-[#2546bd]"
    >
        <p className="text-white text-3xl font-semibold">Connect Wallet</p>
    </button>
)}
```

We import the TransactionContext module, which is used to manage the wallet connection state and functionality. We also import useContext from React to allow us to access the context values. We use the useContext hook to access the connectWallet function and connectedAccount state from the TransactionContext. The connectWallet function is responsible for connecting the user's wallet, while the connectedAccount state stores the wallet address if the wallet is connected.

The 'Connect Wallet' button is conditionally rendered based on the connectedAccount state. If the wallet is not connected '(!connectedAccount)', the button will be displayed. When the user clicks the button, the onClick event triggers the connectWallet function, initiating the wallet connection process.

The className attribute contains the CSS classes for styling the button, using the Tailwind CSS framework. The styling includes the background colour, padding, border radius, cursor style, and hover effect.

2. Play Game Button

```
<button
    type="button"
    onClick={handlePlayButtonClick}
    className="w-2/3 flex flex-row justify-center items-center my-5 bg-[#2952e3] p-8 rounded-full cursor-pointer hover:bg-[#2546bd]">
    <p className="text-white text-3xl text-bold">Play Game</p>
</button>
```

Here, the 'Play Game' button is rendered with an onClick event that triggers the handlePlayButtonClick function. The className attribute contains the CSS classes for styling the button using the Tailwind CSS framework as well.

When the 'Play Game' button is clicked, the handlePlayButtonClick function sets the showModal state to true, which causes a modal to be displayed for the user to enter their game name and the crypto amount they want to enter the game with.

```
onClick={async () => {
    setShowModal(false); // Close the modal
    try {
        await joinGame(name, cryptoAmount);
        navigate('/game', { state: { playerData: { name, cryptoAmount } } });
    } catch (error) {
        console.error("Transaction unsuccessful..");
    }
}}
```

The joinGame function handles the process of joining the game by interacting with the Ethereum smart contract. It checks if a wallet is connected (using the provider state), and then it calls the buyIn function in the smart contract, passing the user's entered name and the crypto amount they want to buy into the poker game with.

When the user clicks the 'Join Game' button in the modal, the modal is closed, the joinGame function is called with the user's entered name and crypto amount, and, if successful, the user is navigated to the '/game' route with the playerData object as state.

Node.js server code for handling requests

In this section, we will provide sample code for a Node.js server that handles HTTP requests. This server will be responsible for fetching player data from the API.

We start by initialising an Express app that uses Socket.IO for real-time communication, which is a great choice for a multiplayer poker game. The server is set up to listen for incoming HTTP requests, as well as WebSocket connections. Here's a brief explanation of the code:

We import the necessary modules, including http, dotenv, cors, express, and the custom socketUtils module that handles socket communication events. We use dotenv to configure environment variables and store them in a .env file. We then create an instance of the Express app and import the Socket.IO module with the custom socketUtils module. We create a HTTP server using the http module and pass in the Express app instance. We initialise Socket.IO with the server instance and set up event handlers using the socketUtils.connection(io) function.

```
2   const http = require('http');
3   const dotenv = require('dotenv');
4   const cors = require('cors');
5
6   dotenv.config({
7       path: './config.env'
8   })
9
10  const express = require('express');
11  const app = express();
12  const socketUtils = require('./utils/socketUtils');
13
14  const server = http.createServer(app);
15  const io = socketUtils.sio(server);
16  socketUtils.connection(io);
```

We create a middleware function called socketIOMiddleware to attach the Socket.IO instance to incoming requests. This allows us to access the Socket.IO instance inside the route handlers using req.io. We then enable CORS (Cross-Origin Resource Sharing) for all routes, allowing requests from different domains or ports. Start the server: Finally, the server listens for incoming connections on the specified port, from the environment variables (see below).

```
const socketIOMiddleware = (req, res, next) => {
    req.io = io;
    next();
};

// CORS
app.use(cors());

// LISTEN
const port = process.env.PORT;
server.listen(port, 'localhost', () => {
    console.log(`App running on port ${port}...`)
})
```

Solidity smart contract code buying in and exiting

In this section, we will discuss the Solidity smart contract code for buying in and exiting a poker game.We first define our smart contract. The smart contract is named PokerGame, and it is written in Solidity version 0.8.0. The player struct is used to store each player's data, including whether they have joined the game, their current balance, and their name. The players mapping maps each player's Ethereum address to their Player struct, and the playerList array stores the addresses of all players.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

contract PokerGame {
    struct Player {
        bool joined;
        uint256 balance;
        string name;
    }

    mapping(address => Player) public players;
    address[] public playerList;
```

The buyIn function allows a player to join the game by sending Ether. It requires a positive amount of Ether to be sent and updates the player's data in the players mapping. The player's address is also added to the playerList.

```
14      function buyIn(string memory _name) public payable {
15          require(
16              msg.value > 0,
17              "Must send a positive amount of Ether to buy in."
18          );
19          //require(!players[msg.sender].joined, "Player already joined the game.");
20
21          players[msg.sender] = Player({
22              balance: msg.value,
23              joined: true,
24              name: _name
25          });
26          playerList.push(msg.sender);
27      }
```

The exitGame function allows a player to exit the game and receive their balance. It takes an integer value winnings, which can be positive, negative, or zero. Positive winnings are added to the player's balance, while negative winnings are deducted from their balance. The player's remaining balance is then transferred back to their Ethereum address.

```
29      function exitGame(int256 winnings) public {
30          require(players[msg.sender].joined, "Player not in the game.");
31
32          if (winnings > 0) {
33              uint256 winningsInWei = uint256(winnings);
34              require(
35                  address(this).balance >= winningsInWei,
36                  "Not enough contract balance to pay winnings."
37              );
38
39              // Send winnings directly to the player
40              payable(msg.sender).transfer(winningsInWei);
41          } else if (winnings < 0) {
42              uint256 lossesInWei = uint256(-winnings);
43              require(
44                  players[msg.sender].balance >= lossesInWei,
45                  "Player doesn't have enough balance to cover losses."
46              );
47
48              // Deduct losses from the player's balance and send it back
49              players[msg.sender].balance -= lossesInWei;
50              payable(msg.sender).transfer(players[msg.sender].balance);
51          } else if (winnings == 0) {
52              // If there are no winnings or losses, just return the player's balance
53              payable(msg.sender).transfer(players[msg.sender].balance);
54          }
55
56          removePlayer(msg.sender);
57      }
```

The removePlayer private function removes a player from the game by deleting their data from the players mapping and removing their address from the playerList. It is called by the exitGame function.

```
59      function removePlayer(address playerToRemove) private {
60          for (uint256 i = 0; i < playerList.length; i++) {
61              if (playerList[i] == playerToRemove) {
62                  playerList[i] = playerList[playerList.length - 1];
63                  playerList.pop();
64                  players[playerToRemove].joined = false;
65                  delete players[playerToRemove]; // Reset the player's struct to its default state
66                  break;
67              }
68          }
69      }
```

VII. Problems Solved

Trust and transparency in online poker

One of the key problems that our decentralised online poker platform addresses is the lack of trust and transparency often associated with traditional online poker platforms. By utilising blockchain technology and smart contracts, our platform ensures that every game outcome and transaction is transparent, verifiable, and tamper-proof.

Players can independently verify the fairness of game outcomes, reducing the chances of fraud and cheating. Our web app utilises blockchain transactions for buying in and exiting the game. This means that every transaction related to the game, including player deposits and withdrawals, is recorded on the blockchain. As blockchain networks are immutable and publicly accessible, players can verify the accuracy of transactions and ensure that payouts are correctly calculated and distributed. This level of transparency discourages any attempts at fraudulent activities and provides players with the confidence that their funds are being managed fairly.

The enhanced trust and transparency provided by our decentralised poker web app attracts a broader range of players, including those who may have been hesitant to join traditional online poker platforms due to concerns about fairness and security. As more players join the platform and engage in the community, the poker ecosystem becomes more vibrant and diverse, offering a rich gaming experience for both casual and professional players alike.

Our decentralised online poker platform leverages blockchain technology and smart contracts to address the trust and transparency issues commonly associated with traditional online poker platforms. By ensuring fair game outcomes, transparent transactions, and secure data management, our platform offers an engaging and trustworthy environment for players to enjoy the game of poker.

Decentralised control and game fairness

Another significant problem solved by our decentralised poker platform is the issue of centralised control and potential player exploitation. Traditional online poker platforms are controlled by a central

authority, which can lead to manipulation of game outcomes, freezing of player accounts, or confiscation of funds without any recourse for the affected players.

Our decentralised platform eliminates the need for a central authority, ensuring that no single entity has the power to manipulate or control the gaming experience. Smart contracts are used to automate various aspects of the game, such as buy-ins and payouts, guaranteeing that the transactions are consistently and fairly enforced.

Moreover, players maintain control over their funds, as they are not required to deposit their cryptocurrencies into a platform account. Instead, their funds remain in their personal wallets, reducing dependency on the platform's security measures and financial stability. This decentralised control empowers players and fosters a fair and trustworthy environment, ultimately improving the overall online poker experience.

Secure and efficient payment processing

Traditional online poker platforms often struggle with payment processing and regulatory issues, relying on conventional methods like credit cards or bank transfers. These methods can be associated with high fees, slow processing times, and potential fraud. Our decentralised poker platform addresses these challenges by utilising cryptocurrencies for secure and efficient payment processing.

Cryptocurrencies, such as Ethereum, allow for faster transaction times and significantly lower fees compared to traditional payment methods. Additionally, cryptocurrencies provide a secure and private means of payment, reducing the risk of fraud and identity theft. Players can maintain their privacy by avoiding the need to share sensitive personal and financial information with third parties. This secure and efficient payment processing system enhances the overall poker experience and encourages more players to join the platform.

Improved user experience and player retention

Our decentralised online poker platform places a strong emphasis on user experience (UX) design, aiming to create an enjoyable and engaging environment for players. By following good design principles and practices, we have developed a platform that is visually appealing, easy to navigate, and responsive across various devices.

The use of modern web technologies like React and Tailwind CSS ensures a smooth and responsive experience for players, while the intuitive layout and user interface design make it easy for both new and experienced players to participate in games. These design choices contribute to an enhanced user experience, which is crucial for retaining players and fostering a loyal and active community.

By addressing the problems of trust and transparency, decentralisation, secure and efficient payment processing, and user experience, our platform offers a compelling alternative to traditional online poker platforms. As a result, it has the potential to attract and retain a growing number of players, driving the success and sustainability of the decentralised online poker industry.

VIII. Results

Performance evaluation

We compared the proposed solution with traditional online poker platforms. To ensure the accuracy and reliability of our poker game, we wrote Jest scripts to test the game logic, such as determining the correct winning hand between two hands. The Jest scripts allowed us to validate the various functionalities of our application, such as game initialization, hand rankings, determining winning hands, and hand comparison.

For example, we used the Jest script below to test the game initialization:

```
6   // 1. GAME INITIALIZATION
7   describe('1. Game Initialization', () => {
8     test('new game should create a deck of 52 shuffled cards', () => {
9       // Arrange
10      const game = gameLogic.newGame();
11      const expectedDeckLength = 52;
12      const possibleRanks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'];
13      const possibleSuits = ['♠', '♣', '♦', '♥'];
14      const expectedCards = [];
15      for (const rank of possibleRanks) {
16        for (const suit of possibleSuits) {
17          expectedCards.push({ rank, suit });
18        }
19      }
20    });
```

This script helped us confirm that a new game creates a deck of 52 shuffled cards and properly adds and removes players from the game.

Additionally, we used Jest scripts to test hand rankings and confirm that the game logic correctly identifies various poker hands, such as royal flushes and four-of-a-kind. These scripts also ensured that the game logic accurately determines the winning hands in different scenarios, including cases with one pair, straight flushes, and identical hands.

For example, the following test case is designed to check the functionality of the determineWinningHand function in the game logic for a poker game. The test is focused on evaluating whether the function correctly determines the winning hand when comparing two players' hands.

```
87  // 3. DETERMINING WINNING HANDS
88  describe('3. Determining Winning Hands', () => {
89    test('Player B wins with a higher pair', () => {
90      const playerA = { name: 'A', wholeCards: [ { rank: '5', suit: '♠' }, { rank: '6', suit: '♠' } ], cards: [{ rank: '5', suit: '♠' }, { rank: '6', suit: '♠' },{ ran
91      const playerB = { name: 'B', wholeCards: [ { rank: 'J', suit: '♦' }, { rank: 'Q', suit: '♦' } ], cards: [{ rank: 'J', suit: '♦' }, { rank: 'Q', suit: '♦' },{ ran
92      const communityCards = [ { rank: 'Q', suit: '♠' }, { rank: '9', suit: '♠' }, { rank: '10', suit: '♠' }, { rank: '5', suit: '♦' }, { rank: '4', suit: '♦' } ];
93
94      const game = {
95      players: [playerA, playerB],
96      communityCards
97      }
```

In this specific case, the test checks if Player B wins when they have a higher pair than Player A. The test case sets up the following game scenario:

1. Player A's whole cards: 5♣ and 6♠
2. Player B's whole cards: J♦ and Q♦
3. Community cards: Q♠, 9♠, 10♠, 5♦, 4♦
4. Both players' hands are then combined with the community cards:
    - Player A's cards: 5♣, 6♠, Q♠, 9♠, 10♠, 5♦, 4♦
    - Player B's cards: J♦, Q♦, Q♠, 9♠, 10♠, 5♦, 4♦

In this scenario, Player A has a pair of 5s (5♣ and 5♦), while Player B has a pair of Queens (Q♦ and Q♠). Player B's pair is higher, so they should win the game. The test case initialises a game object with the two players and the community cards. It then calls the determineWinningHand function from the game-logic.js file, passing the game object as an argument. The function is expected to return an array containing the winning players.

The test case then checks if the first element of the returned winningPlayers array is equal to playerB. If it is, the test passes, indicating that the determineWinningHand function correctly determines the winner with a higher pair. If not, the test fails, indicating there might be an issue with the function's implementation. By automating the testing process, we were able to efficiently identify and resolve any issues in the game logic, ensuring a fair and enjoyable gaming experience for all players.

In addition to the automated testing, we also conducted extensive manual testing by running the server and multiple clients, simulating real-world scenarios in which players engage in the game. This allowed us to observe the game's behaviour, identify potential bugs, and assess the overall user experience. By combining automated and manual testing, we were able to refine our poker game and create a polished, well-functioning platform.

Comparison with traditional online poker platforms

When contrasting our decentralised poker game with conventional online poker platforms, a number of crucial aspects stand out. Firstly, employing blockchain technology and smart contracts provides enhanced security and transparency, as all transactions are recorded on the Ethereum blockchain. This reduces the risk of fraud and manipulation, which are potential issues in traditional, centralised poker platforms.

Our poker game also offers opportunities for further integration with blockchain technology. This can be achieved by incorporating the poker game logic into the smart contract, allowing the recording of game events on the blockchain, as well as streamlining buying in and managing payouts in a secure and transparent manner.

Furthermore, our decentralised poker game is not subject to the constraints of traditional financial systems and regulations, allowing for greater accessibility and inclusivity for players worldwide. This enables a truly easy-to play poker platform where players can compete on a level playing field, regardless of their geographic location or financial background.

By incorporating blockchain technology, smart contracts, and sound testing methodologies, we have developed a secure, transparent, and engaging poker experience.

IX. Future Work

Scaling and performance optimization

A fully decentralised poker game built on a smart contract would inevitably face challenges related to scalability and performance. As the number of players and transactions increases, the load on the underlying blockchain network could lead to increased latency, slower game progression, and higher transaction costs. To address these issues, future work should focus on optimising the smart contract design and exploring off-chain solutions, such as state channels or sidechains, to improve the overall performance of the poker game.

By employing state channels, for example, players could execute multiple actions off-chain while only committing the final state to the blockchain, thus reducing the number of transactions and associated costs. Investigating various scalability solutions would be essential for creating a seamless and enjoyable gaming experience that can accommodate a larger player base.

Further Integration of blockchain

While our current poker game utilises a specific blockchain platform, expanding the range of supported platforms could attract more users and increase overall adoption. As Mavridou and Laszka [12] discuss, Ethereum smart contracts have demonstrated the potential for designing secure applications. In future work, we could investigate the integration of other blockchain platforms to offer more options for users and create a more versatile gaming experience.

Decentralisation of Poker Game Logic

In the current version of our poker web app, the integration of blockchain technology is primarily limited to handling transactions for buying in and exiting the poker game. The game logic itself, however, remains centralised and is not embedded within the smart contract. A possible direction for future work is to fully decentralise the poker game by incorporating the game logic into the smart contract. This would not only ensure greater security and transparency for the users but also reduce the risk of manipulation or fraud by a centralised authority.

Decentralising the game logic would require extensive research into the most efficient and secure algorithms for implementing poker rules, shuffling, and card distribution within a smart contract. Moreover, incorporating mechanisms such as provably fair random number generation, and secure multi-party computation would be crucial to maintain the integrity of the game while preserving privacy.

Cross-Chain Interoperability

Another interesting avenue for future work is to explore cross-chain interoperability, allowing players to participate in the poker game using different blockchain platforms and cryptocurrencies. As Mavridou and Laszka [12] discuss, Ethereum smart contracts have demonstrated the potential for designing secure applications. This would not only widen the range of users but also provide a more inclusive and versatile gaming environment.

Achieving cross-chain interoperability would require the development of bridge protocols that can securely and efficiently transfer information and assets between different blockchain networks. In conclusion, the potential to improve and expand a crypto poker game is vast. By further decentralising the game logic, optimising performance, and exploring cross-chain interoperability, we can create a more secure, transparent, and engaging poker experience for users across various blockchain platforms.

Development of mobile applications

With the increasing prevalence of mobile devices, developing a mobile application for the web app is a potential next step. By offering a mobile application, we can reach a wider audience, increase user retention, and provide a more convenient gaming experience. As Maurer et al. [10] point out, the practical materiality of cryptocurrencies makes them well-suited for mobile applications.

To achieve this, we can look into leveraging the latest advancements in mobile app development, such as cross-platform frameworks and responsive designs, to create a mobile app that offers a seamless experience across various devices. Moreover, future work could also include exploring the integration of novel consensus mechanisms like the one proposed by Yin et al. [13] in their HotStuff protocol, which could enhance the security and efficiency of our poker platform. This could potentially solidify a poker platform's position as a secure and reliable choice in the ecosystem.

X. Conclusion

In this project, we aimed to create a decentralised online poker game, leveraging blockchain technology and smart contracts to ensure fairness, transparency, and security for the players. Through the development of a React front-end, a Node.js back-end server, and a Solidity smart contract, we were able to achieve this goal, providing players with a seamless, decentralised gaming experience. Our platform allows players to buy in and exit the game using Ethereum, while the smart contract ensures proper handling of players' funds, accurate winnings and losses calculations, and secure fund transfers. The decentralised nature of the application ensures that there is no central authority controlling the game, mitigating the risk of fraud and manipulation.

The impact of decentralisation on the future of online poker is undeniable. As blockchain technology continues to evolve, the benefits of decentralised applications in the gaming industry are becoming increasingly evident. Decentralised poker games can offer improved security, trust, and transparency, which are crucial for a fair and enjoyable gaming experience. Furthermore, decentralisation has the potential to reshape the online poker industry by allowing for the creation of truly global platforms, free from the constraints of traditional financial systems and regulations. This can result in increased accessibility and inclusivity, allowing players from all around the world to participate and compete on a level playing field.

In conclusion, our decentralised online poker game showcases the power and potential of blockchain technology in the gaming industry. By leveraging cutting-edge technologies and smart contract capabilities, we have created a secure, transparent, and engaging poker experience for players. As the

world continues to embrace decentralisation and blockchain technology, we can expect to see an even greater impact on the future of online poker and the gaming industry as a whole.

XI. References

1. Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016). MedRec: Using blockchain for medical data access and permission management. In 2016 2nd International Conference on Open and Big Data (OBD). IEEE. URL: https://doi.org/10.1109/OBD.2016.11
2. Casino, F., Dasaklis, T. K., & Patsakis, C. (2019). A systematic literature review of blockchain-based applications: Current status, classification and open issues. Telematics and Informatics. URL: https://doi.org/10.1016/j.tele.2018.11.006
3. Zohar, A. (2015). Bitcoin: under the hood. Communications of the ACM. URL: https://doi.org/10.1145/2701411
4. Swan, M. (2015). Blockchain: Blueprint for a new economy. O'Reilly Media, Inc. URL: http://www.oreilly.com/webops-perf/free/blockchain.csp
5. Tschorsch, F., & Scheuermann, B. (2016). Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials. URL: https://doi.org/10.1109/COMST.2016.2535718
6. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., & Felten, E. W. (2015). SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In 2015 IEEE Symposium on Security and Privacy. IEEE. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7163021
7. Gainsbury, S. M., & Wood, R. T. (2011). Internet gambling policy in critical comparative perspective: The effectiveness of existing regulatory frameworks. International Gambling Studies. URL: https://www.tandfonline.com/doi/full/10.1080/14459795.2011.619553
8. Gainsbury, S. M., Abarbanel, B. L., & Blaszczynski, A. (2017). Game on: Comparison of demographic profiles, consumption behaviors, and gambling site selection criteria of esports and sports bettors. Gaming Law Review. URL: https://www.liebertpub.com/doi/abs/10.1089/glr2.2017.21813?journalCode=glr2
9. Humphreys, B. R., & Pérez, L. (2012). Participation in Internet Gambling Markets: An International Comparison of Online Gamblers' Profiles. URL: https://www.tandfonline.com/doi/full/10.1080/15332861.2012.650987
10. Maurer, B., Nelms, T. C., & Swartz, L. (2013). “When perhaps the real problem is money itself!”: The practical materiality of Bitcoin. Social Semiotics. URL: https://www.tandfonline.com/doi/full/10.1080/10350330.2013.777594
11. Kwon, J., & Bünz, B. (2019). Cosmos: A network of distributed ledgers. URL: https://cosmos.network/resources/whitepaper
12. Mavridou, A., & Laszka, A. (2019). Designing secure Ethereum smart contracts: A finite state machine-based approach. In International Conference on Financial Cryptography and Data Security. URL: https://doi.org/10.1007/978-3-662-58820-8_34
13. Yin, K., Chen, M., & Sirer, E. G. (2020). HotStuff: BFT consensus in the lens of blockchain. In Proceedings of the 2020 ACM Symposium on Principles of Distributed Computing. URL: https://doi.org/10.1145/3382734.3405724
14. Cimpanu, C. (2015). React: Facebook's new JavaScript library for building user interfaces. Softpedia News. URL:

https://news.softpedia.com/news/React-Facebook-s-New-JavaScript-Library-for-Building-User-Interfaces-475679.shtml
15. Hegazy, U., Soliman, O., & El-Ramly, M. (2017). Analysing the usability of the top ten open-source front-end web development libraries and frameworks. In Proceedings of the 9th International Conference on Computer and Automation Engineering. URL: https://doi.org/10.1145/3057039.3057041
16. Meier, P. (2018). The impact of the Virtual DOM in ReactJS on application performance. In Proceedings of the 2nd Workshop on Programming Language Evolution. URL: https://doi.org/10.1145/3236950.3236956
17. Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing. URL: https://doi.org/10.1109/MIC.2010.145
18. Wittern, E., Mosser, A., & Laredo, J. A. (2018). A look at the dynamics of the JavaScript package ecosystem. In Proceedings of the 14th International Conference on Mining Software Repositories (MSR). URL: https://doi.org/10.1145/3196398.3196406