BLOCKCHAIN BASED TOLL COLLECTION SYSTEM

(for the patrial fulfillment of Bachelor of Technology Degree in Computer Science & Engineering)

Submitted by

CHITRANSHU DEEP

SHAILESH KUMAR

TDM SUNDRIYAL

VIVEK KUMAR

Under the guidance of

Mrs. Richa Gupta

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GRAPHIC ERA HILL UNIVERSITY

MAY 2024

CERTIFICATE

This is to certify that the thesis titled **"BLOCKCHAIN BASED TOLL COLLECTION SYSTEM" submitted** by **CHITRANSHU DEEP, SHAILESH KUMAR, TDM SUNDRIYAL & VIVEK KUMAR** to Graphic Era Hill University for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by them under our supervision. The contents of this project in full or in parts have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Dehradun

Date:

Mrs. Richa Gupta
Assistant Professor
GEHU

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to everyone who contributed to the successful completion of this project, "Blockchain-Based Toll Collection System."

First and foremost, we are profoundly grateful to our project guide, Mrs. Richa Gupta, whose guidance, support, and invaluable feedback were instrumental throughout the entire process.

We extend our sincere thanks to our university for providing the necessary resources and a conducive environment for research and development.

Finally, we are deeply grateful to our families and friends for their unwavering support and understanding throughout this journey. Your patience and encouragement gave us the strength to persevere.

Thank you all for your contributions, without which this project would not have been possible.

CHITRANSHU DEEP	**TDM SUNDRIYAL**
2018295	**2018809**
SHAILESH KUMAR	**VIVEK KUMAR**
2018718	**2018869**

ABSTRACT

In an era marked by tax challenges, spatial constraints, inefficiencies, and widespread fraud, Ethereum-based blockchain technology emerges as a powerful solution. This research explores how blockchain can transform toll management systems by enhancing transparency and reducing fraud.

The study highlights the significant benefits of using blockchain technology to improve data integrity and decentralization. By distributing power and ensuring data cannot be altered, blockchain promotes transparency and reliability in data management. This decentralized approach allows for smooth workflow management and maintains data integrity. Real-time audits can become a practical and effective feature.

The proposed system leverages blockchain's decentralized structure to ensure flawless transparency and maintain data integrity, making it user-friendly. Ethereum provides the necessary tools and infrastructure to implement this blockchain-powered toll collection system. Beyond addressing existing issues, the expansion of Ethereum will include advancements in data analytics, smart contracts, and decentralized transaction networks.

Keywords: Blockchain, Ethereum, Truffle, Smart Contract in transportation, decentralized toll transactions, transportation system efficiency, Ganache, Vscode, web3.js, Node.js, React.js.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

• GEHU	Graphic Era Hill University
• ETH	ETHEREUM
• VS	VISUAL STUDIOS
• NPM	NODE PACKAGE MANAGER
• RFID	RADIO FREQUENCY IDENTIFICATION
• JS	JAVA SCRIPT
• WEB3 JS	WEB 3.0 JAVASCRIPT
• CSS	CASCADING STYLING SHEET
• HTML	HYPER TEXT MARKUP LANGUAGE
• GPS	Global Positioning System
• NHAI	National Highway Authority of India
• GDPR	General Data Protection Regulation
• DFD	Data Flow Diagram

CHAPTER 1
INTRODUCTION

Importance of Toll Collection Systems

1. Traffic Management:

Toll collection systems help regulate traffic flow on highways and expressways. By requiring vehicles to pay for road usage, these systems manage demand and reduce congestion. Tolls can be adjusted based on traffic volume, encouraging drivers to travel during off-peak hours, thereby distributing traffic more evenly throughout the day. This demand management is crucial for preventing bottlenecks and ensuring smoother, more efficient traffic movement.

2. Revenue Generation:

Tolls provide a vital source of revenue for governments and private road operators. This revenue is essential for maintaining and improving infrastructure, such as repaving roads, repairing bridges, and upgrading safety features. Additionally, toll revenue funds the construction of new roads and highways needed to accommodate growing traffic volumes. Without tolls, funding for these essential activities would be severely limited, leading to deteriorating road conditions and increased congestion.

3. Infrastructure Development:

Funds generated from toll collection are essential for ongoing infrastructure development. This includes routine maintenance and major projects such as highway expansions, new roads and bridges, and advanced traffic management systems. For example, toll revenues can be used to build bypasses that divert traffic from congested city centers, reducing urban congestion and improving overall traffic flow. Such developments are vital for accommodating increasing traffic volumes and enhancing transportation network efficiency.

Challenges with Traditional Toll Collection Methods

Despite their benefits, traditional toll collection methods face significant challenges in transaction transparency, efficiency, and susceptibility to fraud. These issues highlight the need for more advanced and reliable toll collection systems.

1. Transaction Transparency:

Traditional toll collection systems often lack transparency in transactions. Users have limited visibility into how toll fees are calculated and collected, leading to dissatisfaction as drivers cannot verify the accuracy of charges. The absence of a clear, immutable transaction record makes auditing difficult and complicates dispute resolution between users and toll operators. Transparency is essential for building trust and ensuring fair toll collection practices.

2. Efficiency Issues:

Traditional toll systems, particularly those involving manual collection, are often inefficient. Vehicles must slow down or stop at toll booths, leading to delays and increased travel times. This causes inconvenience for drivers and contributes to traffic congestion, especially during peak travel times. The stop-and-go nature of manual toll collection significantly reduces transportation network efficiency. During busy periods, long lines at toll booths can create substantial delays, frustrating drivers and reducing road network capacity.

3. Susceptibility to Fraud:

Manual toll collection systems are prone to fraud and errors. Cash transactions handled by toll booth operators can be mishandled or misappropriated, leading to revenue losses. Manual transaction recording is vulnerable to manipulation, making it difficult to ensure toll collection integrity. Even electronic toll collection (ETC) systems, aimed at reducing these issues, can still face system integrity and security challenges. For example, hackers can exploit system weaknesses to evade tolls, and employees might engage in fraudulent activities without robust oversight mechanisms.

4. Human Errors and Loopholes:

Human operators in toll collection introduce the possibility of errors and inconsistencies. Mistakes in recording transactions, calculating toll fees, or providing change can lead to disputes and user dissatisfaction. Loopholes in the system can be exploited, resulting in unfair practices and further revenue losses. For example, an operator might mistakenly undercharge or overcharge a vehicle, causing revenue discrepancies and user complaints.

The Need for Advanced Toll Collection Solutions:

Given these challenges, advanced toll collection solutions are needed to address the shortcomings of traditional methods. Technological innovations in automation, data security, and transparency offer promising avenues for improvement.

1. Automation:

Advanced toll collection systems can automate many aspects of the toll collection process, reducing the need for human intervention and minimizing errors and fraud. Automated systems can use technologies such as RFID, GPS, and automatic number plate recognition (ANPR) to identify vehicles and process toll payments seamlessly. These systems can operate continuously, providing service without the need for manual oversight.

2. Data Security:

Ensuring the security of toll transactions is crucial for preventing fraud and protecting user data. Advanced encryption techniques and secure communication protocols can safeguard transaction data from unauthorized access and manipulation. Robust security measures in toll collection systems protect toll revenue integrity and build user trust.

3. Transparency:

Transparency in toll transactions is essential for ensuring fair practices and building user trust. Advanced toll collection systems can provide real-time access to transaction records, allowing users to verify toll charges and monitor toll usage. Blockchain technology, with its decentralized and immutable ledger, offers a compelling solution for enhancing transparency in toll transactions.

Fig. 1.1 Toll Collection System

Blockchain Technology as a Solution:

Blockchain technology, with its decentralized and immutable nature, emerges as a particularly compelling solution. By leveraging cryptographic techniques and smart contracts, blockchain can provide a high level of security, transparency, and efficiency in toll transactions. Ethereum, a leading blockchain platform, supports these capabilities, making it an ideal choice for developing a modern, reliable toll collection system.

1. Decentralization:

Blockchain operates on a decentralized network where no single entity has control over the entire system. This decentralization ensures that transactions are transparent and tamper-proof, as each transaction is recorded on a distributed ledger accessible to all network participants. Decentralization eliminates the need for a central authority, reducing the risk of corruption and enhancing system resilience.

2. Immutability:

Once recorded on the blockchain, transactions cannot be altered or deleted. This immutability provides a clear and auditable record of all toll transactions, enhancing transparency and trust among users and authorities. Immutable records ensure that transaction data remains accurate and reliable, facilitating dispute resolution and auditing.

3. Smart Contracts:

Ethereum, a leading blockchain platform, supports smart contracts—self-executing contracts with the terms directly written into code. Smart contracts automate toll payments, ensuring that tolls are collected accurately and efficiently without the need for manual intervention. For example, a smart contract can automatically deduct the appropriate toll fee from a driver's account when their vehicle passes through a toll point, ensuring timely and accurate payments.

India's Toll Collection Evolution:

India's toll collection system has evolved significantly over the years, adopting various methods to improve efficiency and reduce congestion:

1. Manual Toll System:

- Description: Involves direct interaction between drivers and toll booth cashiers.

- Issues: Causes heavy congestion, increases the risk of accidents, and lacks transaction transparency.

- Security Concerns: Prone to fraud and errors due to human handling.

2. Closed Toll Plaza:

- Description: Vehicles pass through toll booths with minimal slowing down, facilitated by ETC systems.
- Benefits: Enhances traffic management and security, particularly on expressways.
- Technology: Uses RFID tags for vehicle identification and toll deduction.

3. E-Toll System (RFID):

- Description: Uses RFID technology to automatically deduct toll fees from a prepaid balance as vehicles pass through toll booths.
- Advantages: Reduces traffic congestion and speeds up the toll collection process.
- Current Use: Widely implemented as the FASTag system in India.

4. FASTag:

- Description: India's flagship ETC system introduced by the National Highway Authority of India (NHAI).
- Functionality: RFID tags mounted on vehicle windshields enable seamless toll transactions in designated lanes.
- Impact: Revolutionizes travel by significantly reducing stop-and-go traffic at toll booths.

Advancements Beyond Traditional Systems:

To further enhance toll collection efficiency and security, hybrid systems and futuristic technologies are being explored:

1. Hybrid Systems:

-Description: Combine traditional and modern technologies, such as GPS and blockchain.

- Purpose: Address compatibility issues and adapt to changing infrastructure needs. Hybrid systems can provide greater flexibility and resilience, ensuring that toll collection remains efficient and reliable even as technology evolves.

2. Lane Free from Toll Booth (LFTB):

- Concept: Eliminate physical toll plazas and use advanced cameras and sensors to track vehicles and automatically deduct tolls or create postpaid subscriptions.

- Potential: Though still under development, this system holds significant promise for the future of toll collection. By eliminating the need for physical toll booths, LFTB can reduce congestion, improve traffic flow, and enhance the overall user experience. Advanced technologies such as ANPR and GPS can accurately track vehicle movements and facilitate seamless toll payments.

Role of Blockchain in Toll Collection

Blockchain technology, particularly Ethereum, is emerging as a key solution for online transactions in toll collection systems. Its features of immutability, security, and transparency make it an ideal choice for ensuring reliable and efficient toll transactions. Ethereum's scalability and support for smart contracts allow for practical deployment and expansion of blockchain-based toll collection systems.

FASTTAGS:

FASTag represents a monumental shift in toll collection systems, particularly notable for its transformative impact on India's transportation infrastructure. Introduced by the National Highway Authority of India (NHAI) under the Electronic Toll Collection (ETC) program, FASTag is emblematic of the nation's transition towards digital, contactless payment solutions aimed at enhancing efficiency, reducing congestion, and improving user experience on highways and expressways.

Operatively, FASTag relies on Radio-Frequency Identification (RFID) technology, employing passive RFID stickers affixed to vehicle windshields. These RFID tags harbor unique identification information tethered to a prepaid account managed by banks or authorized service providers. As a FASTag-equipped vehicle approaches a toll plaza, overhead RFID readers automatically detect and register the tag, enabling seamless toll payment without necessitating physical cash transactions or manual intervention.

The operational simplicity and convenience inherent in FASTag have proven instrumental in reshaping India's toll collection landscape. By obviating the need for vehicles to halt or decelerate at toll booths, FASTag has substantially mitigated congestion and travel durations on highways, bolstering overall traffic flow and road safety. Moreover, FASTag's contactless transaction mechanism has assumed heightened relevance in contexts such as the COVID-19 pandemic, where minimizing physical interactions is paramount.

A cornerstone of FASTag's efficacy lies in its interoperability across all toll plazas nationwide, irrespective of the issuing bank or toll operator. This interoperability ensures seamless travel for motorists across the country, as a single FASTag suffices for toll payments on any national highway or expressway. Additionally, FASTag users enjoy ancillary benefits such as discounted toll rates and dedicated lanes at toll plazas, incentivizing widespread adoption and utilization.

The pervasive adoption of FASTag owes to concerted efforts from governmental bodies, financial institutions, and toll operators. Awareness campaigns, incentives, and regulatory mandates have collectively galvanized the rapid uptake of FASTag among vehicle owners. Consequently, millions of vehicles are now equipped with FASTag's, catalyzing the digitization and modernization of India's toll collection infrastructure.

Looking forward, FASTag is poised to assume an increasingly pivotal role in shaping the trajectory of transportation and toll collection systems. Ongoing initiatives aimed at augmenting FASTag's functionality and coverage, including the integration of supplementary services like parking payments and fuel purchases, are expected to further propel adoption and usage. Moreover, technological advancements such as GPS integration and real-time monitoring capabilities hold promise for optimizing toll collection processes and enhancing overall efficiency and user experience on India's highways and expressways.

RFID'S:

Radio-Frequency Identification (RFID) technology has emerged as a transformative force across various industries, offering unparalleled capabilities in automated data collection, asset tracking, and identification. At its core, RFID systems comprise RFID tags, RFID readers, and a backend database or system for storing and processing data. RFID tags consist of a microchip and an antenna, which enable them to transmit and receive radio frequency signals. RFID readers emit radio waves to activate RFID tags within their proximity and capture the information transmitted by these tags.

In toll collection systems, RFID technology has revolutionized the way tolls are collected, replacing traditional cash-based transactions with electronic and contactless methods. RFID tags, also known as toll tags or transponders, are affixed to vehicles, typically on the windshield or license plate. Each RFID tag is encoded with a unique identification number linked to a prepaid account or vehicle registration.

As a vehicle equipped with an RFID tag approaches a toll plaza, overhead RFID readers emit radio frequency signals to activate the tag. The RFID tag responds by transmitting its unique identification number to the RFID reader. The reader captures this information and communicates it to the toll collection system, which processes the transaction and debits the corresponding toll amount from the linked account.

One of the key advantages of RFID technology in toll collection is its speed and efficiency. RFID-based toll transactions occur in real-time and without the need for the vehicle to stop or slow down, significantly reducing congestion and travel times at toll plazas. Moreover, RFID technology offers high accuracy and reliability, ensuring seamless toll collection even in high-traffic environments.

RFID technology also enables interoperability between different toll roads and toll operators. RFID tags issued by one toll authority can be used for toll payments on multiple toll roads and highways, providing convenience for motorists and eliminating the need for multiple toll accounts or payment methods.

Beyond toll collection, RFID technology finds applications in various other sectors, including logistics, retail, healthcare, and manufacturing. In logistics and supply chain management, RFID tags are used to track and trace shipments, optimize inventory management, and improve supply chain visibility. In retail, RFID technology enables inventory tracking, product authentication, and theft prevention. In healthcare, RFID tags are used for patient identification, asset tracking, and medication management. In manufacturing, RFID technology streamlines production processes, enhances product traceability, and improves quality control.

Let's break down each component and aspect of RFID technology and its application in toll collection systems:

1. RFID Tags: RFID tags are small devices consisting of a microchip and an antenna. These tags are affixed to vehicles, typically on the windshield or license plate. Each RFID tag contains a unique identification number or code, which is encoded into the microchip. RFID tags come in various forms, including passive, active, and semi-passive. In toll collection systems, passive RFID tags are commonly used due to their cost-effectiveness and simplicity.

2. RFID Readers: RFID readers are devices that emit radio frequency signals to activate RFID tags within their proximity and capture the information transmitted by these tags. RFID readers are installed at toll plazas and are equipped with antennas to detect RFID tags passing through the toll lane. When a vehicle equipped with an RFID tag approaches the toll plaza, the RFID reader emits radio waves to activate the tag, prompting it to transmit its unique identification number.

3. Backend Database or System: The data captured by RFID readers is communicated to a backend database or system for processing. This backend system is responsible for storing and managing information related to RFID tag

IDs, prepaid accounts, toll rates, and transaction records. When a toll transaction occurs, the backend system verifies the RFID tag ID, retrieves the corresponding account information, calculates the toll amount, and debits the toll from the linked account.

4. Real-time Transaction Processing: RFID-based toll transactions occur in real-time, allowing for seamless and efficient toll collection. As a vehicle equipped with an RFID tag passes through the toll plaza, the RFID reader captures the tag's unique ID and transmits it to the backend system. The backend system processes the transaction instantaneously, debiting the toll amount from the linked prepaid account. This real-time processing eliminates the need for vehicles to stop or slow down at toll booths, reducing congestion and travel times.

5. Interoperability: RFID technology enables interoperability between different toll roads and toll operators. RFID tags issued by one toll authority can be used for toll payments on multiple toll roads and highways, providing convenience for motorists. This interoperability eliminates the need for drivers to maintain multiple toll accounts or payment methods, streamlining the toll payment process and enhancing user experience.

Overall, RFID technology offers a fast, efficient, and interoperable solution for toll collection systems, revolutionizing the way tolls are collected and managed. By leveraging RFID tags, readers, and backend systems, toll authorities can enhance operational efficiency, reduce congestion, and improve the overall travel experience for motorists.

SATELLITE BASED TOLL COLLECTION SYSTEM:

Satellite-based toll collection systems represent an innovative approach to toll collection, utilizing satellite positioning technology to streamline toll payments and enhance traffic management on highways and expressways. Unlike traditional toll collection methods that rely on physical infrastructure such as toll booths and RFID readers, satellite-based systems leverage satellite positioning systems like GPS or GNSS to track vehicles and calculate tolls based on their location and distance traveled.

Here's a breakdown of how satellite-based toll collection systems function:

1. Vehicle Tracking: These systems utilize GPS or GNSS technology to track the precise location of vehicles as they travel along highways. Each vehicle is equipped with a GPS-enabled device or transponder that communicates with satellites to determine its real-time position.

2. Distance Measurement: By continuously monitoring vehicle locations, satellite-based systems can calculate the distance traveled by each vehicle between specific points on the highway. This distance measurement serves as the basis for determining the toll amount, typically charged per kilometer or mile traveled.

3. Automatic Toll Calculation: As vehicles progress along the highway, satellite-based toll collection systems automatically compute the toll amount based on the distance traveled and applicable toll rates. This calculation is performed in real-time by onboard software or centralized toll management systems linked to satellite networks.

4. Prepaid or Postpaid Accounts: Similar to RFID-based toll collection systems, satellite-based systems can be linked to prepaid or postpaid accounts maintained by vehicle owners or operators. Toll charges are automatically deducted from prepaid accounts or billed to postpaid accounts, eliminating the need for cash transactions or manual interventions.

5. Efficiency and Accuracy: Satellite-based toll collection systems offer several advantages over traditional methods. By eliminating physical toll booths and manual transactions, these systems reduce congestion and travel times on highways. Moreover, satellite positioning technology ensures accurate toll calculations, minimizing errors in toll collection.

6. Interoperability and Scalability: These systems have the potential for high interoperability and scalability, allowing seamless integration with existing toll infrastructure and interoperability across different toll roads and jurisdictions.

This interoperability enables motorists to use a single toll account or device for payments across multiple highways, enhancing user convenience.

In summary, satellite-based toll collection systems present an efficient and forward-thinking solution for toll collection, harnessing satellite positioning technology to automate payments and optimize traffic flow on highways and expressways. As technology advances, these systems are expected to play a significant role in modernizing transportation infrastructure and improving the overall travel experience for motorists.

TOLLS COLLECTION AROUND THE WORLD:

Toll collection systems vary significantly around the world, with each country implementing unique approaches tailored to its infrastructure, traffic patterns, and economic considerations. Here's an overview of toll collection systems in several countries across different continents:

1. United States: Toll roads are prevalent in the United States, particularly in regions with high population density and extensive highway networks such as the Northeast and California. Toll collection methods include traditional toll booths where motorists pay cash or use electronic toll collection (ETC) systems like E-ZPass. Many toll roads also employ open-road tolling systems, where overhead gantries equipped with RFID readers automatically deduct tolls from transponders or license plate images.

2. Canada: Canada has an extensive network of toll roads, bridges, and tunnels, primarily concentrated in urban areas such as Toronto and Vancouver. Toll collection methods vary by province and include traditional toll booths, electronic toll collection systems like Transponder, and video tolling systems that capture license plate images for billing.

3. United Kingdom: The United Kingdom utilizes a mix of toll roads, bridges, and congestion charging schemes to manage traffic and generate revenue for infrastructure maintenance. Major toll roads such as the M6 Toll employ

electronic toll collection systems, while congestion charging schemes like the London Congestion Charge use automatic number plate recognition (ANPR) cameras to enforce fees for vehicles entering designated zones.

4. France: France operates an extensive toll road network, particularly on its autoroutes (motorways). Toll collection methods include traditional toll booths, liber-t electronic tolling devices, and télépéage systems that allow vehicles to pass through toll plazas without stopping. France also utilizes tolling systems for bridges and tunnels, such as the Millau Viaduct.

5. Germany: Germany's toll collection system primarily applies to heavy commercial vehicles using its autobahn network. The country plans to introduce a satellite-based tolling system for trucks known as the German Truck Toll (LKW-Maut), which will calculate tolls based on distance traveled and vehicle emissions.

6. Australia: Australia employs toll roads in major cities like Sydney, Melbourne, and Brisbane to manage urban congestion and finance infrastructure projects. Toll collection methods include electronic tolling systems like E-TAG and video tolling systems that capture license plate images for billing.

7. China: China has one of the largest networks of toll roads in the world, spanning thousands of kilometers across the country. Toll collection methods include traditional toll booths, electronic toll collection systems like ETC and E-PASS, and license plate recognition systems for video tolling.

8. India: India has seen significant growth in toll road development in recent years, particularly with the introduction of Electronic Toll Collection (ETC) systems like FASTag. FASTag employs RFID technology for automatic toll collection, streamlining traffic flow and reducing congestion on highways and expressways.

9. Brazil: Brazil utilizes toll roads extensively, particularly in urban areas and on major highways. Toll collection methods include traditional toll booths,

electronic tolling systems like Sem Parar and Connect-Car, and ANPR-based video tolling systems.

10. Japan: Japan operates an extensive network of toll roads, including expressways and bridges. Toll collection methods include traditional toll booths, electronic toll collection systems like ETC and ETC2.0, and smartphone-based tolling apps.

11. Spain: Spain has a well-developed toll road network, particularly on its major highways and bridges. Toll collection methods include traditional toll booths, electronic tolling systems like VIA-T and Telepeaje, and ANPR-based video tolling systems. Spain also operates toll tunnels and bridges, such as the Eix Transversal tunnel in Barcelona.

12. Italy: Italy utilizes toll roads extensively, especially on its autos trade (motorways) connecting major cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like Tele pass, and ANPR-based video tolling systems. Italy also employs tolls on bridges and tunnels, such as the Grande Raccordo Anulare in Rome.

13. South Africa: South Africa operates toll roads across its major highways and bridges, particularly in urban areas like Johannesburg and Cape Town. Toll collection methods include traditional toll booths, electronic tolling systems like e-tag, and ANPR-based video tolling systems. South Africa also employs tolling for bridges, such as the Nelson Mandela Bridge in Johannesburg.

14. Mexico: Mexico has an extensive toll road network, particularly on its major highways connecting cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like TAG and IAVE, and ANPR-based video tolling systems. Mexico also utilizes tolls for bridges and tunnels, such as the Autopista Urbana Sur in Mexico City.

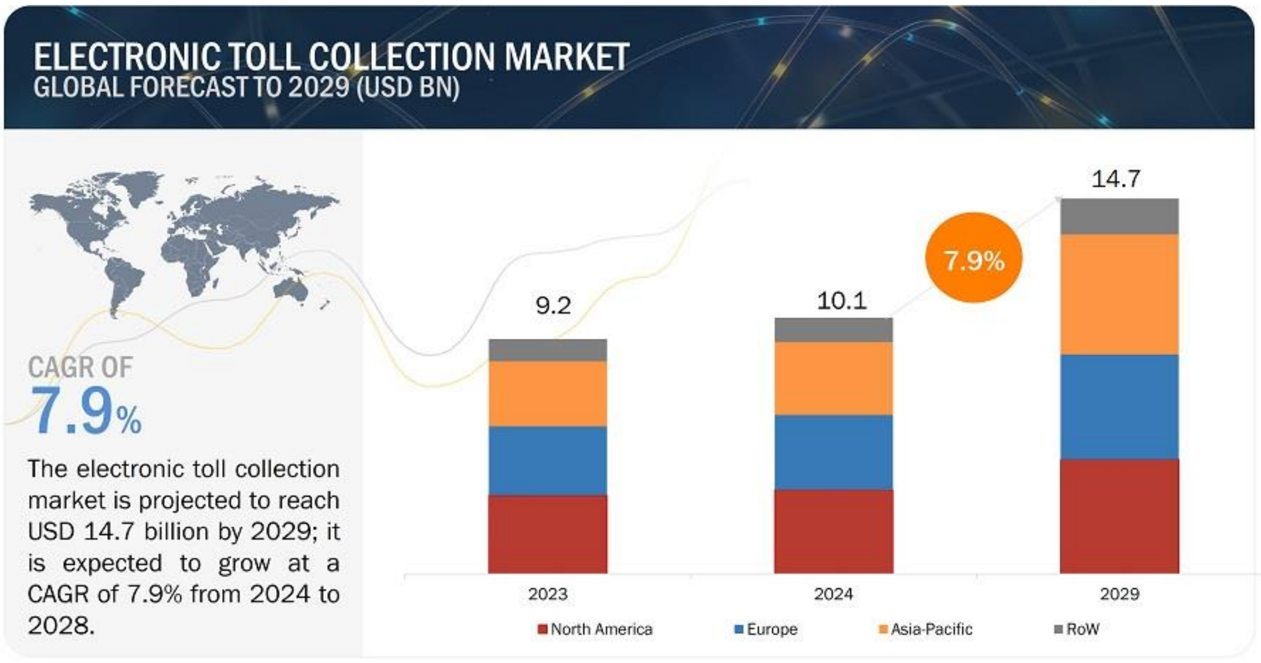


Fig. 1.2 Market Cap of Tolls around the World.

15. Norway: Norway employs tolls primarily in urban areas and on major bridges and tunnels. Toll collection methods include traditional toll booths, electronic tolling systems like Auto PASS, and ANPR-based video tolling.

systems. Norway also utilizes tolling for tunnels, such as the Oslo fjord Tunnel.

16. Sweden: Sweden operates tolls mainly in urban areas and on major bridges and tunnels. Toll collection methods include traditional toll booths, electronic tolling systems like BroBizz and Auto PASS, and ANPR-based video tolling systems. Sweden also employs tolling for bridges, such as the Oresund Bridge connecting Sweden and Denmark.

17. Singapore: Singapore is known for its Electronic Road Pricing (ERP) system, which employs gantries equipped with ERP devices to charge vehicles for road usage during peak hours and in congested areas. The ERP system uses satellite-based positioning and electronic payment systems to deduct tolls automatically from prepaid cards or linked accounts.

18. Thailand: Thailand operates toll roads primarily in urban areas and on major highways connecting cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like Easy Pass and Expressway

Easy Pass, and ANPR-based video tolling systems. Thailand also employs tolling for bridges, such as the Rama VIII Bridge in Bangkok.

19. Portugal: Portugal has an extensive network of toll roads, particularly on its major highways and bridges. Toll collection methods include traditional toll booths, electronic tolling systems like Via Verde, and ANPR-based video tolling systems. Portugal also employs tolls for bridges, such as the Vasco da Gama Bridge in Lisbon.

20. Turkey: Turkey operates toll roads across its major highways and bridges, especially in urban areas like Istanbul and Ankara. Toll collection methods include traditional toll booths, electronic tolling systems like HGS and OGS, and ANPR-based video tolling systems. Turkey also utilizes tolling for bridges, such as the Bosphorus Bridge connecting Europe and Asia.

21. Russia: Russia has toll roads primarily in urban areas and on major highways connecting cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like Platon, and ANPR-based video tolling systems. Russia also employs tolling for bridges, such as the Western High-Speed Diameter in Saint Petersburg.

22. Poland: Poland operates toll roads mainly on its major highways connecting cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like via TOLL, and ANPR-based video tolling systems. Poland also utilizes tolling for bridges, such as the Świnoujście Bridge connecting Poland and Germany.

23. Argentina: Argentina has toll roads across its major highways and bridges, particularly in urban areas like Buenos Aires and Cordoba. Toll collection methods include traditional toll booths, electronic tolling systems like TelePASE, and ANPR-based video tolling systems. Argentina also employs tolling for tunnels, such as the Tunnel of Saint Martin in Buenos Aires.

24. Chile: Chile operates toll roads primarily on its major highways connecting cities and regions. Toll collection methods include traditional toll booths, electronic tolling systems like TAG and VIA-T, and ANPR-based video tolling systems. Chile also utilizes tolling for bridges, such as the Costanera Norte Viaduct in Santiago.

25. Netherlands: The Netherlands has toll roads mainly in urban areas and on major bridges and tunnels. Toll collection methods include traditional toll booths, electronic tolling systems like E-ZPass, and ANPR-based video tolling systems. The Netherlands also employs tolling for tunnels, such as the Westerschelde Tunnel.

26. Indonesia: Indonesia has toll roads in major cities and regions, primarily aimed at reducing congestion and improving transportation infrastructure. Toll collection methods include traditional toll booths, electronic tolling systems like e-TOLL and e-Money, and ANPR-based video tolling systems. Indonesia also employs tolling for bridges, such as the Sur Amadu Bridge connecting Java and Madura islands.

27. Malaysia: Malaysia operates toll roads on major highways and expressways across the country, particularly in urban areas like Kuala Lumpur and Penang. Toll collection methods include traditional toll booths, electronic tolling systems like Touch 'n Go and SmartTag, and ANPR-based video tolling systems. Malaysia also utilizes tolling for bridges, such as the Penang Bridge connecting the island of Penang to the mainland.

28. South Korea: South Korea has toll roads primarily on its major highways and expressways, aimed at improving transportation efficiency and connectivity. Toll collection methods include traditional toll booths, electronic tolling systems like Hi-pass and U-pass, and ANPR-based video tolling systems. South Korea also employs tolling for bridges, such as the Yeong Jong Grand Bridge connecting Incheon International Airport to the mainland.

29. Greece: Greece operates toll roads on major highways and bridges, particularly in urban areas and tourist destinations. Toll collection methods

include traditional toll booths, electronic tolling systems like e-PASS and Athena, and ANPR-based video tolling systems. Greece also utilizes tolling for tunnels, such as the Rio–Antirrio bridge connecting the Peloponnese peninsula to mainland Greece.

30. Norway: Norway has toll roads in urban areas and on major bridges and tunnels, aimed at managing traffic and financing infrastructure projects. Toll collection methods include traditional toll booths, electronic tolling systems like Auto PASS, and ANPR-based video tolling systems. Norway also employs tolling for ferry crossings and bridges, such as the Svinesund Bridge connecting Norway and Sweden.

31. Denmark: Denmark operates toll roads primarily in urban areas and on major bridges and tunnels, aimed at reducing congestion and financing transportation projects. Toll collection methods include traditional toll booths, electronic tolling systems like BroBizz and Easy Go, and ANPR-based video tolling systems. Denmark also utilizes tolling for bridges, such as the Oresund Bridge connecting Denmark and Sweden.

32. New Zealand: New Zealand has toll roads on major highways and bridges, particularly in urban areas and tourist destinations. Toll collection methods include traditional toll booths, electronic tolling systems like the Northern Gateway Toll Road, and ANPR-based video tolling systems. New Zealand also employs tolling for tunnels, such as the Waterview Tunnel in Auckland.

In conclusion, toll collection systems vary widely across the globe, with each country implementing its own methods tailored to its infrastructure and transportation needs. From traditional toll booths to sophisticated electronic tolling systems and satellite-based technologies, toll collection methods continue to evolve to improve traffic management and fund infrastructure projects. Despite differences in approach, the overarching goal remains the same: to ensure efficient transportation and enhance the overall travel experience for motorists. As technology continues to advance, we can expect further innovations in toll collection systems, contributing to safer, more sustainable, and better-connected transportation networks worldwide.

ROLE OF BLOCK CHAIN:

1. Transparency: Blockchain's transparency comes from its decentralized and immutable nature. When a toll transaction occurs, such as a vehicle passing through a toll gantry, the details are recorded on the blockchain ledger. This record includes information like the time, location, and toll amount paid. Once recorded, this data cannot be altered or deleted, providing a transparent and auditable trail of transactions. This transparency builds trust among toll operators, government agencies, and motorists as they all have access to the same accurate information.

2. Security: Blockchain ensures the security of toll transaction data through cryptographic techniques and decentralization. Each transaction is cryptographically linked to the previous one, forming a secure chain of blocks. Moreover, blockchain operates on a decentralized network, where transaction data is distributed across multiple nodes. This redundancy makes it extremely difficult for anyone to alter transaction records without detection. Additionally, consensus mechanisms ensure that all nodes agree on the validity of transactions, further enhancing security and preventing unauthorized access.

3. Efficiency: Blockchain's efficiency lies in its ability to automate toll collection processes using smart contracts. Smart contracts are self-executing contracts with predefined rules encoded on the blockchain. In toll collection systems, smart contracts can automate tasks such as toll fee calculation, vehicle passage verification, and payment processing. By eliminating manual intervention, smart contracts reduce processing times, minimize errors, and lower administrative costs associated with toll collection operations. This automation enhances the overall efficiency of toll infrastructure, improving traffic flow and reducing congestion.

3. Interoperability: Blockchain enables seamless integration and communication between different tolling systems and jurisdictions. Toll operators and government agencies can establish a common platform for recording and verifying toll transactions. This standardized protocol allows motorists to use a single toll account or device across multiple toll roads, bridges, and tunnels

operated by different authorities. As a result, travelers experience greater convenience and efficiency when navigating toll infrastructure without the need to manage multiple accounts or devices.

4. Decentralization: Blockchain's decentralized architecture ensures the resilience and reliability of toll collection systems. Transaction records are distributed across a network of nodes, eliminating single points of failure. Even in the event of a network outage or cyber-attack, transaction data remains accessible and secure on the blockchain ledger. This resilience ensures continuous operation of tolling services, maintaining traffic flow, and supporting economic activity.

BLOCKCHAIN PLATFORMS:

1. Ethereum:

- Ethereum is a decentralized platform known for its smart contract capabilities.
- Toll collection systems can utilize Ethereum's smart contracts to automate toll transactions and ensure transparency.
- Ethereum's scalability makes it suitable for managing high transaction volumes in tolling networks.

2. Hyperledger Fabric:

- Hyperledger Fabric is an open source blockchain framework designed for enterprise use cases.
- Toll operators can deploy private or permissioned blockchain networks using Hyperledger Fabric to ensure data privacy and regulatory compliance.
- Features like identity management and confidentiality make Hyperledger Fabric suitable for toll collection systems.

3. Ripple:

- Ripple is a blockchain platform specializing in cross-border payments and remittances.
- While not commonly used in toll collection, Ripple's fast transaction speeds and low costs could be beneficial for international tolling scenarios.

- Ripple's consensus algorithm ensures fast transaction settlement, making it suitable for high-volume tolling environments.

4. Stellar:

- Stellar focuses on cross-border payments and asset transfers.
- Toll operators could leverage Stellar's fast transaction speeds and low fees for processing toll payments in international tolling scenarios.
- Stellar's consensus mechanism provides security and reliability for toll transactions across distributed networks.

5. Cardano:

- Cardano aims to provide a scalable, interoperable, and sustainable blockchain infrastructure.
- Toll operators could utilize Cardano for building toll collection systems with built-in flexibility and adaptability.
- Cardano's architecture enables efficient transaction processing and smart contract execution, suitable for complex tolling solutions.

6. Tezos:

- Tezos emphasizes security, decentralization, and upgradability.
- Toll operators could use Tezos for toll collection systems with built-in governance mechanisms, allowing stakeholders to propose protocol upgrades without disrupting the network.
- Tezos' consensus algorithm ensures decentralized agreement among participants, providing scalability and security for toll transactions.

7. VeChain:

- VeChain focuses on supply chain management and product traceability.
- Toll operators could apply VeChain's features to track toll payments, monitor vehicle movement, and ensure regulatory compliance in tolling operations.
- VeChain's platform provides transparency and accountability, enhancing trust in tolling systems.

Network Scalability Challenges:

In the context of toll collection systems, "networks" refer to the infrastructure and communication channels through which tolling data is transmitted and processed. These networks encompass various components, including toll booths, gantries, backend servers, payment gateways, and communication protocols. Here's an explanation of networks and the security challenges they pose:

1. Toll Collection Networks:

- Toll collection networks comprise physical toll infrastructure such as toll booths, gantries, and sensors installed along highways, bridges, and tunnels.
- These networks are responsible for detecting vehicles, capturing tolling data (e.g., vehicle identification, time-stamped transactions), and transmitting this data to backend systems for processing.
- Toll collection networks may utilize technologies like radio-frequency identification (RFID), automatic number plate recognition (ANPR), or GPS-based tracking systems to identify vehicles and record toll transactions.

2. Backend Systems:

- Backend systems consist of servers, databases, and software applications responsible for processing toll transactions, managing user accounts, and generating reports.
- These systems receive tolling data from toll collection networks, validate transactions, calculate toll fees, and update user accounts accordingly.
- Backend systems also handle payment processing, including authorization, settlement, and reconciliation of toll payments made by motorists.

3. Communication Channels:

- Communication channels facilitate the transmission of data between toll collection networks and backend systems.
- These channels may include wired or wireless connections, such as Ethernet, Wi-Fi, cellular networks, or dedicated communication links (e.g., fiber optic cables).
- Secure communication protocols like HTTPS, TLS/SSL, or VPNs are often employed to encrypt data transmission and protect against interception or tampering.

Security Challenges in Toll Collection Networks:

1. Data Privacy: Toll collection systems process sensitive information, including vehicle identification, license plate numbers, and payment details. Unauthorized access to this data could lead to privacy breaches and identity theft.

2. Data Integrity: Ensuring the integrity of tolling data is crucial to prevent tampering or manipulation of transaction records. Any unauthorized changes to toll transactions could result in revenue loss or disputes between toll operators and motorists.

3. Cybersecurity Threats: Toll collection networks are vulnerable to cyber-attacks, including malware infections, ransomware attacks, and distributed denial-of-service (DDoS) attacks. These threats can disrupt tolling operations, compromise data integrity, and lead to financial losses.

4. Fraud and Theft: Toll collection systems are susceptible to fraud and theft, including toll evasion, payment card fraud, and skimming attacks. Toll operators must implement robust security measures to detect and prevent fraudulent activities.

5. Physical Security: Physical security risks, such as vandalism, theft of tolling equipment, or unauthorized access to toll facilities, pose significant challenges to toll collection networks. Implementing surveillance cameras, access controls, and perimeter security measures can mitigate these risks.

6. Compliance and Regulation: Toll collection systems must comply with regulatory requirements and industry standards for data protection, privacy, and payment security. Non-compliance can result in legal penalties, fines, or damage to the reputation of toll operators.

To address these security challenges, toll operators can implement a range of measures, including encryption, authentication mechanisms, intrusion detection systems, security audits, and employee training programs. Collaborating with cybersecurity experts and adopting industry best practices can help enhance the security posture of toll collection networks and safeguard against emerging threats.

SECURITY CHALLENGES

1. Data Privacy: Toll collection systems handle sensitive information, including personally identifiable information (PII) such as vehicle registration details, license plate numbers, and payment card information. To protect data privacy, toll operators must implement robust data encryption techniques, access controls, and data anonymization practices. Encrypting data both at rest (stored on servers or databases) and in transit (during transmission between toll booths and backend systems) ensures that sensitive information remains secure and confidential. Additionally, implementing strict access controls, role-based permissions, and regular data audits can help prevent unauthorized access to sensitive data and mitigate the risk of data breaches.

2. Data Integrity: Maintaining the integrity of tolling data is essential to ensure the accuracy and reliability of transaction records. Any unauthorized modifications or tampering of toll transactions can lead to revenue loss, disputes, and legal liabilities. Toll operators can employ cryptographic hash functions and digital signatures to verify the integrity of tolling data. By generating unique hashes for each toll transaction and digitally signing transaction records using private keys, toll operators can detect any unauthorized changes to transaction data. Implementing blockchain technology, which provides an immutable and tamper-proof ledger of transactions, can further enhance data integrity in toll collection systems.

3. Cybersecurity Threats: Toll collection networks are susceptible to various cybersecurity threats, including malware infections, ransomware attacks, and distributed denial-of-service (DDoS) attacks. Toll operators must implement robust cybersecurity measures to protect against these threats. This includes deploying antivirus software, intrusion detection and prevention systems (IDS/IPS), firewalls, and security patches to safeguard against malware and cyber-attacks. Conducting regular security assessments, penetration testing, and

security awareness training for employees can help identify vulnerabilities and improve the overall security posture of toll collection networks.

4. Fraud and Theft: Toll collection systems are vulnerable to fraud and theft, including toll evasion, payment card fraud, and skimming attacks. Toll operators can mitigate these risks by implementing multi-factor authentication (MFA) mechanisms, tokenization for payment card transactions, and real-time transaction monitoring systems. MFA requires users to provide multiple forms of authentication, such as passwords, biometrics, or one-time passcodes, before accessing tolling services, reducing the risk of unauthorized access and fraud. Tokenization replaces sensitive payment card information with unique tokens, preventing unauthorized access to card data and reducing the risk of payment card fraud. Real-time transaction monitoring systems analyze transaction patterns and detect anomalies or suspicious activities, allowing toll operators to respond promptly to potential fraud incidents.

5. Physical Security: Physical security risks, such as vandalism, theft of tolling equipment, or unauthorized access to toll facilities, pose significant challenges to toll collection networks. Toll operators can enhance physical security measures by installing surveillance cameras, access controls, and perimeter security fencing at toll facilities. Access controls, such as biometric authentication, keycard systems, or security guards, restrict unauthorized entry to tolling infrastructure and deter potential intruders. Surveillance cameras provide continuous monitoring of tolling facilities and can capture evidence of security incidents or suspicious activities, aiding in investigations and law enforcement efforts. Additionally, implementing alarm systems, motion sensors, and regular security patrols can further enhance the physical security of toll collection networks.

6. Compliance and Regulation: Toll collection systems must comply with regulatory requirements and industry standards for data protection, privacy, and payment security. This includes adhering to regulations such as the General Data Protection Regulation (GDPR), Payment Card Industry Data Security Standard (PCI DSS), and industry-specific regulations for tolling operations. Toll operators must conduct regular security audits, risk assessments, and compliance checks to ensure adherence to relevant regulations and standards. Partnering with cybersecurity experts, legal advisors, and industry associations can help toll

operators stay informed about regulatory changes and best practices for maintaining compliance in toll collection networks.

By addressing these security challenges and implementing appropriate security measures, toll operators can enhance the resilience, reliability, and trustworthiness of toll collection networks, ensuring the secure and efficient operation of tolling services for motorists and stakeholders alike.

Overview:

The introduction provides an overview of toll collection systems, emphasizing their role in managing traffic and infrastructure costs. It acknowledges the challenges faced by traditional tolling methods, including transparency issues and susceptibility to fraud. The introduction introduces blockchain technology as a potential solution, citing its benefits in enhancing transparency, security, and efficiency.

Additionally, the introduction discusses the evolution of toll collection systems in India, highlighting various methods used and their shortcomings. It underscores the importance of technological advancements to improve user experience and operational efficiency.

In conclusion, the introduction sets the context for exploring blockchain-based toll collection systems and emphasizes the project's significance in addressing the limitations of traditional tolling methods. Through the adoption of blockchain technology, the project aims to redefine toll collection systems, offering a more transparent, secure, and efficient approach to managing transportation infrastructure.

CHAPTER 2

Requirement Analysis for a Blockchain-based Toll Collection System

Given the discussion on traditional toll collection systems, Electronic Toll Collection (ETC), and the functionalities of blockchain, here's a breakdown of the requirement analysis for a blockchain-based toll collection system:

1. **Functional Requirements:**

- **User Roles:** Define user roles like drivers, toll road authorities, and potentially government agencies.
- **Account Management:** Users (drivers) need to create accounts, link payment methods (cryptocurrency or traditional), and top up their accounts.
- **Toll Management:** The system should automatically detect vehicles on the toll road (via license plate recognition or on-board units).
- **Transaction Processing:** Secure and automatic toll deduction should occur based on vehicle class, distance travelled, and predefined toll rates.
- **Data Recording:** All transactions (vehicle ID, toll amount, timestamp) should be immutably recorded on the blockchain.
- **Dispute Resolution:** A mechanism for handling disputes (incorrect charges, technical issues) needs to be defined.

2. **Non-Functional Requirements:**

- **Security:** The system should be highly secure to prevent unauthorized access, data manipulation, and fraud.
- **Scalability:** The blockchain needs to handle a high volume of transactions efficiently to avoid congestion.
- **Performance:** Transaction processing and data retrieval should happen with minimal latency for a smooth user experience.

- **Privacy:** While transactions are recorded, user identities (driver information) might need to be anonymized depending on regulations.
- **Interoperability:** The system should ideally integrate with existing ETC infrastructures (OBUs) for wider adoption.
- **Regulatory Compliance:** The system should adhere to relevant government regulations regarding data privacy and toll collection practices.

3. Blockchain-Specific Requirements:

- **Consensus Mechanism:** Choose a suitable consensus mechanism (Proof of Stake, Proof of Authority) considering factors like transaction speed, energy consumption, and network governance.
- **Smart Contracts:** Develop smart contracts to automate toll calculations, deductions, and dispute resolution processes.
- **Data Storage:** Determine how much data will be stored on-chain (essential information) and off-chain (detailed transaction records) for cost optimization.

4. Additional Considerations:

- **Cost Analysis:** Evaluate the cost of setting up and maintaining the blockchain network compared to traditional ETC systems.
- **Pilot Testing:** Consider implementing a pilot program on a limited scale to test the system's functionality and identify potential issues before full deployment.
- **User Interface:** Design user-friendly interfaces for drivers, toll authorities, and other stakeholders to interact with the system.

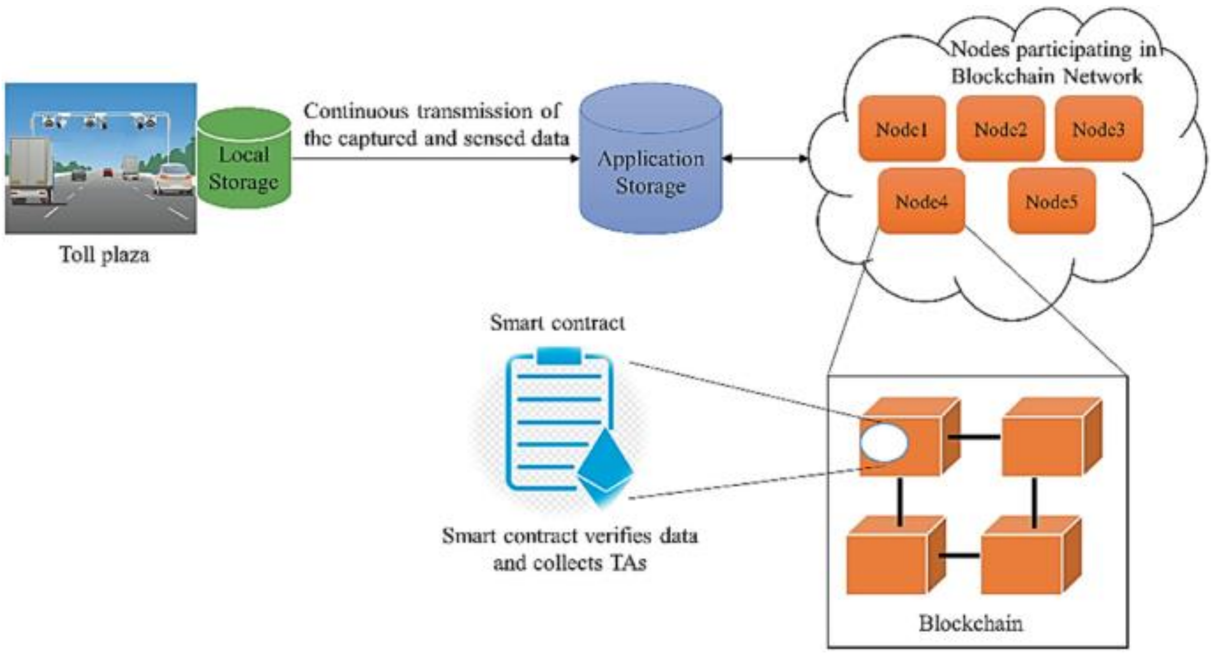

Toll plaza
Local Storage
Continuous transmission of the captured and sensed data
Application Storage
Nodes participating in Blockchain Network
Node1
Node2
Node3
Node4
Node5
Smart contract
Smart contract verifies data and collects TAs
Blockchain

Fig. 2.1 Blockchain Architecture

1. Functional Requirements:

- **User Roles:**
  - **Drivers:** Users who own vehicles and will be charged tolls for using the road. They need functionalities for account creation, payment linking (crypto or traditional), account top-up, and potentially managing trip history.
  - **Toll Road Authorities:** Entities responsible for managing the toll roads. They need functionalities for system administration, setting toll rates, monitoring traffic flow, managing disputes, and potentially accessing anonymized usage data for planning purposes.
  - **Government Agencies (Optional):** Depending on the regulatory landscape, government agencies might be involved. They might need access to anonymized data for auditing purposes or have functionalities for setting compliance guidelines.

- **Account Management:**
  - Drivers should be able to create secure accounts with username and password or integrate existing digital identity solutions.

    - The system should allow linking various payment methods, including cryptocurrency wallets and traditional credit/debit cards.
    - A top-up functionality needs to be in place for drivers to add funds to their accounts before or after using the toll road.

- **Toll Management:**
    - The system needs to automatically detect vehicles using the toll road. This can be achieved through:
        - **License Plate Recognition (LPR):** Cameras capture license plates, and the system identifies the vehicle and associated account.
        - **On-Board Units (OBUs):** Vehicles carry transponders that communicate with roadside readers, similar to existing ETC systems.

- **Transaction Processing:**
    - Tolls should be automatically deducted from the driver's account based on:
        - **Vehicle Class:** Different vehicle classes (cars, trucks, motorcycles) might have different toll rates.
        - **Distance Travelled:** The system might calculate tolls based on the distance travelled on the toll road.
        - **Predefined Toll Rates:** Toll authorities will define the toll rates for different vehicle classes and potentially different times of day (peak vs. off-peak hours).

- **Data Recording:**
    - All toll transactions, including vehicle identification (license plate or OBU ID), toll amount, timestamp, and potentially location data, should be immutably recorded on the blockchain. This ensures transparency and auditability.

- **Dispute Resolution:**
    - A mechanism needs to be established to handle potential disputes, such as:
        - Incorrect toll charges due to misidentification of the vehicle class or technical issues.
        - Double charges for the same trip.
        - Account hacking or unauthorized transactions.
    - The system might involve a process for drivers to flag disputes, and toll authorities or a designated third party would investigate and resolve them.

2. Non-Functional Requirements:

- **Security:**
    - The system should be highly secure against various threats, including:
        - **Unauthorized Access:** Prevent hackers from gaining access to user accounts or manipulating system data.
        - **Data Manipulation:** Ensure the immutability of the blockchain ledger to prevent altering transaction records.
        - **Fraudulent Activities:** Mitigate the risk of drivers evading tolls or manipulating the system for unauthorized use.
    - Implementing robust encryption techniques, access controls, and regular security audits are crucial.

- **Scalability:**
    - The blockchain network needs to handle a high volume of toll transactions efficiently, especially during peak hours. This can be achieved through:
        - Choosing a scalable blockchain platform that can handle increasing transaction loads.
        - Implementing techniques like sharding or off-chain data storage to optimize blockchain usage.

- **Performance:**
    - Transaction processing and data retrieval should happen with minimal latency. Drivers shouldn't experience delays when passing through toll plazas. Optimizing the system architecture and network infrastructure is essential for smooth operation.

- **Privacy:**
    - While transaction data is recorded on the blockchain for transparency, user privacy needs to be considered.
        - Driver identities (names, addresses) might be anonymized and only linked to unique identifiers on the blockchain.
        - Data privacy regulations like GDPR (Europe) and CCPA (California) need to be adhered to when handling user information.

- **Interoperability:**
    - Ideally, the system should integrate with existing Electronic Toll Collection (ETC) infrastructure, particularly On-Board Units (OBUs) already used by drivers. This would facilitate wider adoption without requiring users to switch to entirely new technology.

- **Regulatory Compliance:**
    - The system design and operation should comply with relevant government regulations regarding:
        - Data privacy laws governing how user information is collected, stored, and used.
        - Toll collection practices established by government agencies responsible for transportation infrastructure.

Blockchain-Specific Requirements:

Expanding on Blockchain Requirements for Toll Collection:

Building a robust blockchain-based toll collection system requires careful consideration beyond the core functionalities. Here's a further exploration of some key areas:

1. Advanced Dispute Resolution:

While smart contracts can automate basic disputes, a more comprehensive mechanism might be needed:

- **Multi-level Dispute Resolution:**
    - Level 1: Automated processes within the smart contract handle basic issues like toll recalculations for minor distance discrepancies.
    - Level 2: Toll authorities or a designated third-party can review disputes escalated by drivers. This might involve reviewing camera footage or other evidence.
    - Level 3: Complex disputes could be escalated to an arbitration panel for final resolution.
- **Dispute Evidence Management:** The system could allow drivers to upload relevant evidence (photos, receipts) to support their claims during the dispute resolution process. Secure storage of this evidence, potentially off-chain with proper access controls, is important.
- **Dispute Incentive Design:** Mechanisms can be implemented to discourage frivolous disputes. For example, a small deposit could be required when initiating a dispute, which would be refunded only if the dispute is resolved in the driver's favour.

2. Integration with Existing Infrastructure:

For wider adoption, integrating the blockchain system with existing toll collection infrastructure is crucial:

- **On-Board Unit (OBU) Integration:** The system should ideally work with existing OBUs used by drivers. This might involve developing adapters or interfaces to enable communication between OBUs and the blockchain network.

- **Legacy Toll Booth Integration:** While the goal might be to eventually phase out physical toll booths, allowing them to temporarily interact with the blockchain system could ease the transition. This might involve a system where booth operators collect cash and manually credit driver accounts on the blockchain later.

3. Privacy-enhancing Techniques:

Beyond basic anonymization, additional techniques can enhance user privacy:

- **Zero-knowledge proofs:** These cryptographic techniques allow drivers to prove they own a valid account or belong to a specific vehicle class without revealing the underlying data itself.
- **Differential privacy:** This technique adds noise to transaction data, making it difficult to identify individual users while preserving overall trends in the data for toll authorities.

4. Incentive Programs and Gamification:

Blockchain can be used to implement incentive programs to encourage desired behaviour:

- **Loyalty Programs:** Drivers who use the toll road frequently could earn rewards points stored on the blockchain, redeemable for discounts or other benefits.

- **Congestion Pricing:** The system could dynamically adjust tolls based on real-time traffic data, incentivizing drivers to use the toll road during off-peak hours.

5. Governance and Regulatory Considerations:

- **Governance Model:** A clear governance model needs to be established, especially for permissioned blockchain networks. This would define how toll authorities, potentially government agencies, and other stakeholders participate in decision-making processes related to the system's operation and evolution.
- **Regulatory Landscape:** Continuously monitoring and adapting to evolving regulations around blockchain technology, data privacy, and toll collection practices is essential for long-term success.

ANALYTICS:

1. Pilot Testing and Phased Rollout:

Given the complexities involved, a phased approach is recommended:

- **Proof-of-Concept (POC):** Develop a small-scale prototype to test core functionalities like toll calculation, smart contract execution, and data storage on a blockchain platform.
- **Pilot Program:** Implement the system on a limited section of a toll road with a controlled number of users. This allows for real-world testing, identifying, and addressing issues before a full-scale deployment.
- **Phased Rollout:** Gradually expand the system to cover more toll roads and users based on the success of the pilot program. This allows for iterative improvements and minimizes potential disruption to a large user base.

2. Cost-Benefit Analysis:

A thorough cost-benefit analysis is essential to determine the system's feasibility:

- **Cost Factors:**
  - Setting up and maintaining the blockchain network infrastructure.
  - Development and ongoing maintenance of smart contracts.
  - Integration with existing toll collection infrastructure (OBUs, toll booths).
  - Regulatory compliance costs.
- **Benefit Factors:**
  - Reduced operational costs from eliminating manual toll collection.
  - Increased revenue collection efficiency due to automation and reduced fraud.
  - Improved traffic flow through faster toll processing.
  - Potential for implementing dynamic pricing strategies for congestion management.

The analysis should compare the total cost of ownership for the blockchain system with the projected benefits to determine if it offers a compelling return on investment.

3. Environmental Impact Assessment:

The environmental impact of the chosen consensus mechanism needs to be considered:

- **Proof of Work (PoW) consumes a significant amount of energy.** If chosen, exploring renewable energy sources to power the blockchain network could be crucial.
- **Proof of Stake (PoS) is a more environmentally friendly alternative.** However, the energy consumption might still be a factor depending on the number of transactions processed.

4. User Adoption and Education:

Encouraging user adoption is essential for the system's success:

- **Public Awareness Campaigns:** Educate drivers about the benefits of the new system, including faster toll processing and potential loyalty programs.
- **User-friendly Interface:** Design a user-friendly mobile app or web interface for drivers to manage their accounts, top up balances, and track trip history.
- **Customer Support:** Provide robust customer support channels to address user inquiries and troubleshoot any issues that might arise.

5. Future-proofing the System:

The system should be designed with scalability and future advancements in mind:

- **Modular Design:** Develop the system with modular components to allow for easier integration of new features and functionalities in the future.
- **Interoperability Standards:** Adhere to emerging interoperability standards for blockchain technology to facilitate potential future connections with other transportation infrastructure systems.

- **Continuous Monitoring and Improvement:** Continuously monitor system performance, identify areas for improvement, and adapt to evolving technologies and user needs.

DATA INTEGRITY SECTION:

1. Security Enhancements:

- **Hardware Security Modules (HSMs):** These tamper-resistant devices can be used to store cryptographic keys securely, further protecting the system from unauthorized access and manipulation.
- **Multi-signature wallets:** For high-value toll road authorities, implementing multi-signature wallets for managing toll revenue can require multiple authorized parties to approve transactions, adding an extra layer of security.

- **Penetration Testing and Vulnerability Assessments:** Regularly conducting penetration testing and vulnerability assessments helps identify and address potential security weaknesses in the system before they can be exploited.

2. **Scalability Solutions:**

- **Sharding:** This technique partitions the blockchain into smaller shards, allowing for parallel processing of transactions and improving scalability for high-volume toll collection systems.

- **Layer 2 solutions:** These solutions process transactions off-chain and then commit them to the main blockchain periodically. This can significantly improve transaction throughput without compromising security.

3. **Privacy-preserving Techniques - Deep Dive:**

- **Zero-knowledge proofs with zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge):** This advanced cryptographic technique allows drivers to prove they meet specific criteria (e.g., owning a valid account, belonging to a specific vehicle class) without revealing any underlying data. This enhances user privacy while ensuring compliance with toll regulations.

- **Homomorphic encryption:** This technique allows computations to be performed on encrypted data without decrypting it first. This could be used to enable toll authorities to analyse anonymized traffic data for planning purposes without compromising individual user privacy.

4. **Incentive Programs - Specific Examples:**

- **Low Emission Vehicle (LEV) Discounts:** Drivers of electric vehicles or other low-emission vehicles could receive automatic discounts on tolls to encourage environmentally friendly transportation choices.
- **Time-based Discounts:** The system could offer discounts for using the toll road during off-peak hours to incentivize smoother traffic flow and reduce congestion during peak periods.

5. Governance Model - Detailed Examples:

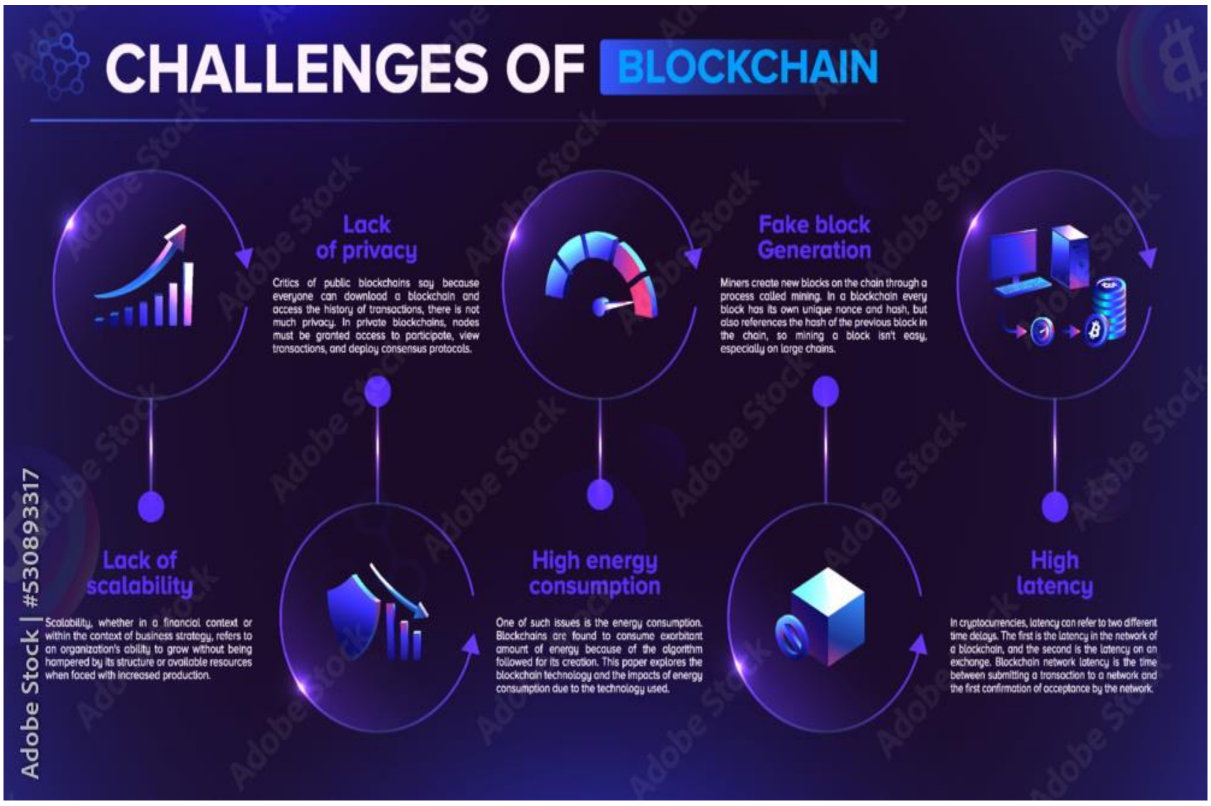
- **Consortium Blockchain:** Multiple toll road authorities could come together to form a consortium and jointly govern the blockchain network. This model allows for shared decision-making and cost distribution.
- **Decentralized Autonomous Organization (DAO):** A DAO could be established where stakeholders (toll authorities, government agencies, potentially even driver representatives) hold tokens that give them voting rights on proposals related to the system's operation and upgrades.

CHALLENGES:

1. Blockchain Interoperability with Other Transportation Systems:

- **Integration with Smart Cities Infrastructure:** The toll collection system could connect with other smart city initiatives, such as parking management systems or electric vehicle charging stations. Blockchain could enable seamless payments and data exchange across these systems, enhancing the overall user experience.

- **Supply Chain Management Integration:** For toll roads frequently used by trucks transporting goods, the system could potentially integrate with blockchain-based supply chain management solutions. This could streamline documentation processes and improve overall logistics efficiency.



Critics of public blockchains say because everyone can download a blockchain and access the history of transactions, there is not much privacy. In private blockchains, nodes must be granted access to participate, view transactions, and deploy consensus protocols.

Miners create new blocks on the chain through a process called mining. In a blockchain every block has its own unique nonce and hash, but also references the hash of the previous block in the chain, so mining a block isn't easy, especially on large chains.

Scalability, whether in a financial context or within the context of business strategy, refers to an organization's ability to grow without being hampered by its structure or available resources when faced with increased production.

One of such issues is the energy consumption. Blockchains are found to consume exorbitant amount of energy because of the algorithm followed for its creation. This paper explores the blockchain technology and the impacts of energy consumption due to the technology used.

In cryptocurrencies, latency can refer to two different time delays. The first is the latency in the network of a blockchain, and the second is the latency on an exchange. Blockchain network latency is the time between submitting a transaction to a network and the first confirmation of acceptance by the network.

Fig. 2.2 Challenges of Blockchain

2. Exploring Alternative Consensus Mechanisms:

- **Byzantine Fault Tolerance (BFT):** This consensus mechanism offers faster transaction processing compared to Proof of Stake (PoS). However, it requires a higher number of validator nodes, which could increase operational costs for the toll collection system.

- **Directed Acyclic Graphs (DAGs):** These alternative blockchain structures offer faster transaction speeds and potentially lower energy consumption compared to traditional blockchains. However, DAGs might require additional considerations regarding security and data immutability for a toll collection system.

3. Potential Challenges and Risks:

- **Technological Immaturity:** Blockchain technology is still evolving, and its long-term scalability and security in real-world applications like toll collection need further validation.
- **Regulatory Uncertainty:** The regulatory landscape surrounding blockchain technology is constantly evolving. The system needs to be designed to adapt to changing regulations regarding data privacy, digital currencies, and toll collection practices.

- **User Adoption and Acceptance:** Encouraging widespread user adoption of a new toll collection system can be challenging. Public education campaigns and user-friendly interfaces are crucial to overcome potential resistance to change.

5. The Future of Blockchain-based Toll Collection:

- **Standardization and Interoperability:** As blockchain technology matures, industry-wide standards for interoperability between different toll collection systems are likely to emerge. This would facilitate seamless travel across different toll roads, regardless of the underlying blockchain platform used.

- **Integration with Autonomous Vehicles (AVs):** In the future, toll collection systems might need to integrate with autonomous vehicles. Blockchain could potentially enable secure and automated toll payments for AVs without requiring any human intervention.

- **Focus on Sustainability:** As environmental concerns become a higher priority, blockchain-based toll collection systems could be designed to incentivize low-emission vehicles and promote sustainable transportation practices.

CHAPTER 3

Software and Project Design for a Blockchain-based Toll Collection System

Project Overview:

Develop a secure, efficient, and scalable blockchain-based toll collection system that integrates with existing infrastructure for optimized traffic management.

System Architecture:

- **Frontend:**
  - Mobile app or web interface for users (drivers) to manage accounts, top-up balances, track trip history, and potentially access loyalty programs.
  - Interface for toll road authorities for system administration, monitoring traffic flow, managing disputes, and accessing anonymized data for planning purposes. (Government agencies might also have access depending on regulations)

- **Backend:**
  - Blockchain network (permissioned or public) for secure and transparent transaction recording.
  - Smart contracts for automated toll calculations, deductions, and dispute resolution (initial stages).
  - Secure database for storing user information and detailed transaction records (linked to anonymized blockchain hashes).
  - AI module for traffic prediction, anomaly detection, and (potentially) dynamic toll pricing in the future.

- **Integration Layer:**
    - Interfaces to connect with existing On-Board Unit (OBU) infrastructure for vehicle identification.
    - Potential future integration with other intelligent transportation systems or smart city infrastructure.

Algorithms:

- **Consensus Mechanism:** Choose an algorithm based on transaction volume and desired level of decentralization (Proof of Stake (PoS) or Byzantine Fault Tolerance (BFT) are good candidates).
- **Encryption Algorithms:** Implement robust encryption algorithms (e.g., AES-256) to protect user data and secure communication throughout the system.
- **Cryptographic Algorithms:** Utilize cryptographic techniques like zero-knowledge proofs (zk-SNARKs) for privacy-preserving user authentication and attribute verification.

Blueprint Phases:

- **Phase 1: Proof of Concept (POC):**
    - Develop a scaled-down version of the system to test core functionalities like toll calculation, smart contract execution, and data storage on a blockchain platform.
    - Integrate with a simulated OBU for vehicle identification.
- **Phase 2: Pilot Program:**
    - Implement the system on a limited section of a toll road with a controlled number of users.
    - Collect real-world data to evaluate system performance, identify and address issues, and refine AI algorithms for traffic prediction.
- **Phase 3: Full-Scale Rollout:**
    - Gradually expand the system to cover more toll roads and users based on the success of the pilot program.
    - Continuously monitor and improve the system based on user feedback and real-world data.

Additional Considerations:

- **Scalability:** Design the system with scalability in mind, potentially using sharding or layer 2 solutions to handle high transaction volumes.
- **Security:** Conduct regular penetration testing and vulnerability assessments to identify and address potential security risks.
- **Privacy:** Implement privacy-preserving techniques like zero-knowledge proofs and differential privacy to protect user data.
- **Regulatory Compliance:** Ensure the system adheres to relevant data privacy regulations and toll collection practices.

Tools and Technologies:

- Programming languages (e.g., Solidity for smart contracts).
- Blockchain development platforms (e.g., Ethereum)
- Secure databases (e.g., MySQL with encryption)
- Cloud computing platforms (for scalability and easier deployment)

Frontend for a Blockchain-based Toll Collection System with React, Node.js, and CSS

User Interface (UI) Technologies:

The frontend for this toll collection system can be built using a combination of popular web development technologies:

- **React:** A JavaScript library for building user interfaces. Reacts component-based architecture makes it well-suited for creating complex and dynamic UIs like this system's frontend.
- **Node.js:** A JavaScript runtime environment that allows us to run JavaScript code outside of the browser. Node.js will be used for server-side rendering and potentially handling API interactions with the backend.
- **CSS (Cascading Style Sheets):** For styling the user interface elements and creating a visually appealing and user-friendly experience.
- **Additional Libraries:** You might also consider using frontend libraries like Material UI or Bootstrap to pre-built components and styling that can accelerate development.

Frontend Components:

Here's a breakdown of potential frontend components for the system:

- **Driver Dashboard:**
  - Account management (profile, linked payment methods)
  - Top-up balance functionality
  - Trip history with timestamps, routes, and toll amounts (anonymized)
  -
- **Toll Booth Interface (optional for toll booth personnel):**
  - Display vehicle information (license plate or OBU ID)
  - Manual toll processing (in case of OBU malfunctions or exceptions)
  - Dispute management interface (to initiate or review disputes)

Data Fetching and API Communication:

- The React application will likely fetch data from the backend Node.js server through APIs. These APIs would then interact with the blockchain network or the secure database to retrieve relevant user information, trip history, or potential loyalty program data (all while maintaining user privacy).
- Secure authentication mechanisms (e.g., JWT tokens) would be essential to ensure only authorized users can access their account information.

Security Considerations:

- **User Authentication:** Implement secure user authentication flows, potentially using OAuth or other industry standards.
- **Data Encryption:** Sensitive data like user passwords and payment information should be encrypted at rest and in transit.
- **Input Validation:** Validate user input to prevent potential security vulnerabilities like cross-site scripting (XSS) attacks.

1.1 Tool used:

1 Reacts.js:

React.js, a popular JavaScript library, is ideal for crafting our dapp's user interface. Its component-based architecture allows for building reusable and manageable UI elements. React's virtual DOM ensures efficient UI updates, and its state management capabilities handle user interactions smoothly. This, combined with React's vast variety of libraries and tools, allows us to design a user-friendly and visually appealing dapp experience.

2 Blockchain platform (Ethereum):

Ethereum is a well-established public blockchain platform with a strong focus on smart contract functionality. Ethereum allows anyone to deploy permanent and immutable decentralised applications onto it, with which users can interact.

3 Web3.js:

Web3.js talks to The Ethereum Blockchain with Json RPC, which stands for "Remote Procedure Call" protocol. Ethereum is a peer-to-peer network of nodes that stores a copy of all the data and code on the blockchain. Web3.js allows us to make requests to an individual Ethereum node with JSON RPC to read and write data to the network. It's kind of like using jQuery with a JSON API to read and write data with a web server.

4 Ganache:

Ganache is simulation software used to run blockchain transactions locally. It is also useful in regulating amount of gas fees used with each smart contract. Ganache is a component of the Truffle suite framework along with the other components, Truffle and Drizzle. Truffle serves as the development environment, testing framework and asset pipeline based on the Ethereum Virtual Machine.

5 Truffle:

Truffle is a development framework for building decentralized applications (dapps) on the blockchain, specifically on the Ethereum network. It provides tools and libraries to simplify development, including smart contract creation, testing, deployment, and management. Truffle streamlines the workflow and offers features like automated testing, network management, and an interactive console, making it easier for developers to build robust and secure dapps on the blockchain.

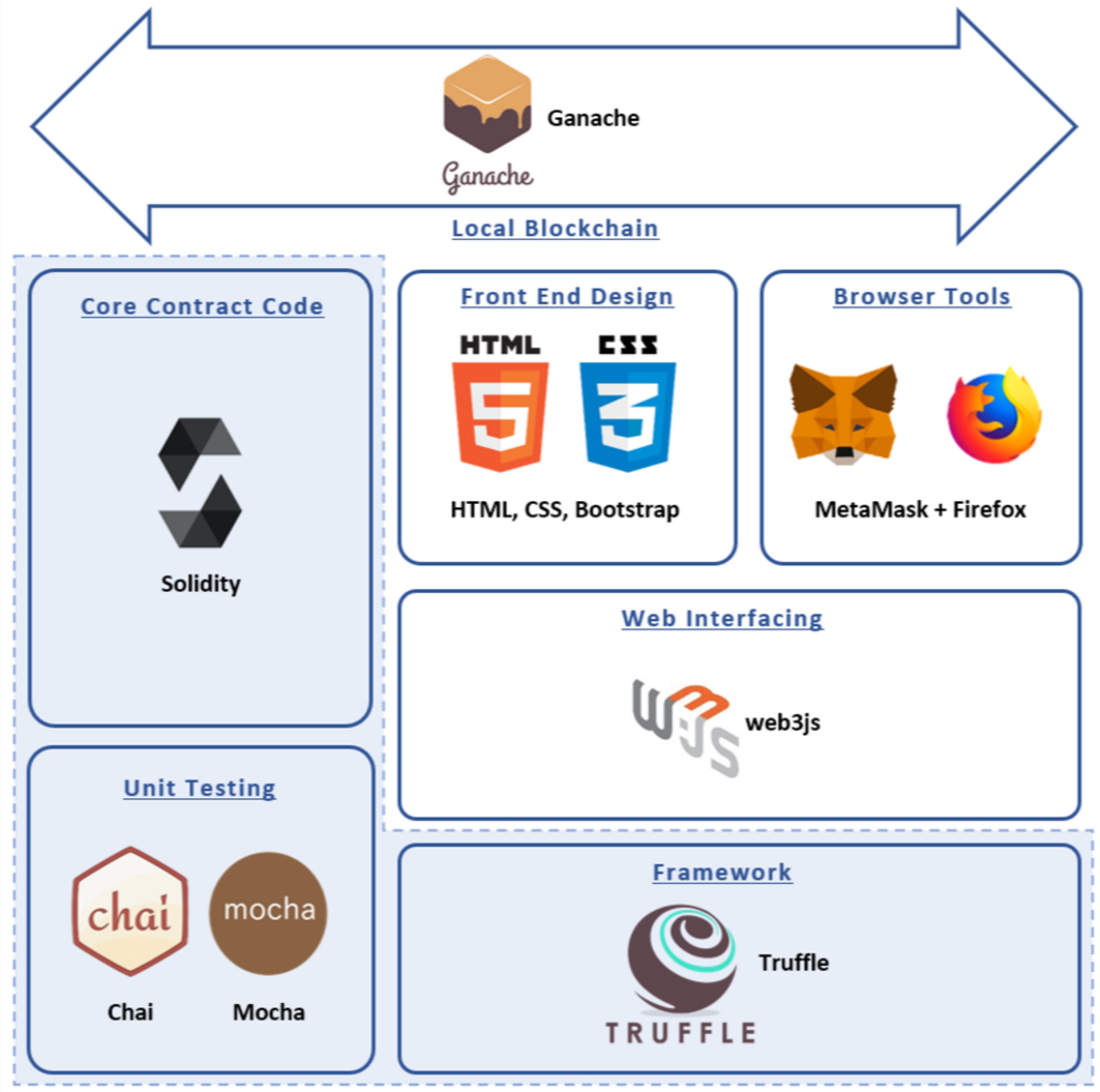


Fig. 3.1 System Framework and Architecture

1.2 Hardware Requirement:

The hardware requirements for the blockchain-based toll collection system can be divided into two main categories:

1. Development Environment:

- **Personal Computer:** A standard laptop or desktop computer with sufficient processing power and memory is adequate for development purposes.

  - Processor: A mid-range processor (e.g., Intel Core i5 or AMD Ryzen 5) should be sufficient for development and testing.
  - Memory (RAM): 8GB of RAM is a good starting point. More RAM can improve performance when running multiple development tools and simulators.
  - Storage: Enough storage space to accommodate development tools, codebase, and any local blockchain deployment (if applicable). Solid State Drive (SSD) is recommended for faster loading times.
  - Operating System: A recent version of Windows, macOS, or Linux is suitable depending on y developer preferences and the specific tools they choose.

2. Deployment Environment (Optional):

- The hardware requirements for deployment will depend on the chosen deployment scenario:

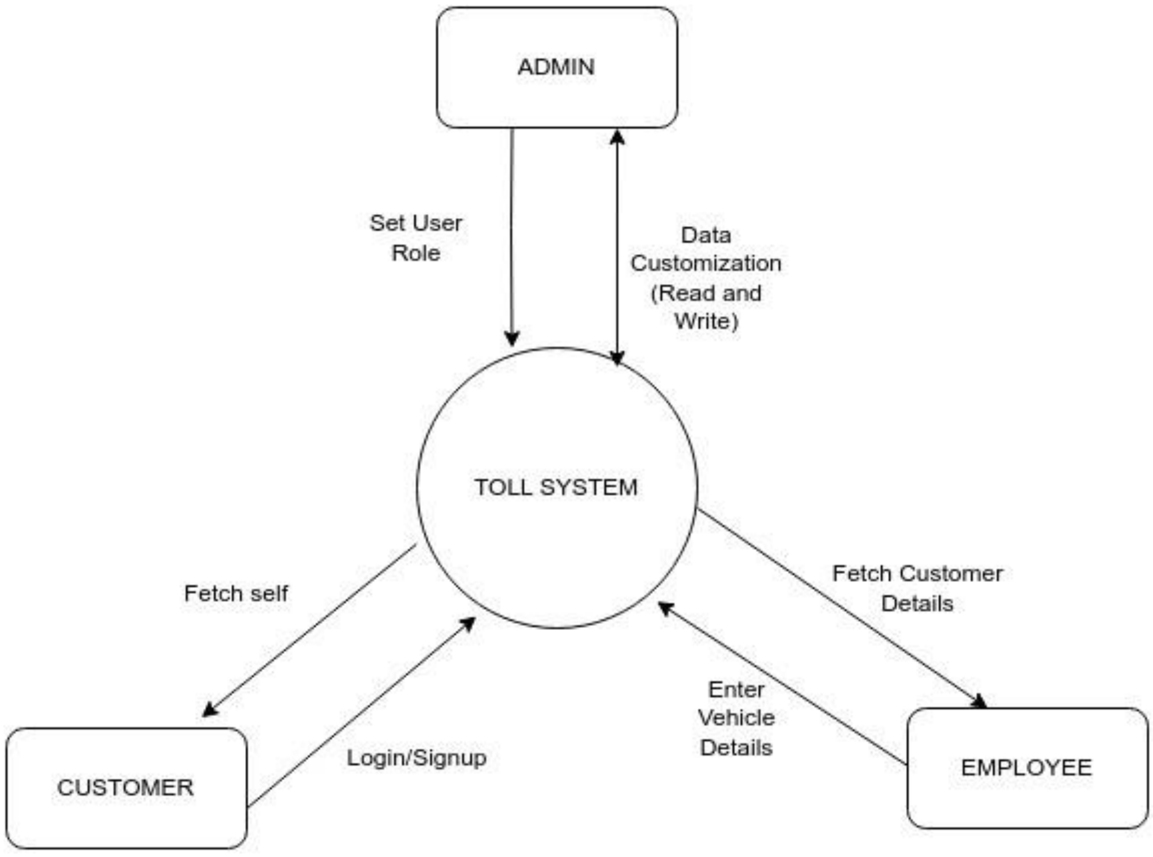
  - **Public Blockchain (e.g., Ethereum):** We wouldn't need dedicated hardware as the blockchain infrastructure is already established. However, consider the cost of transaction fees on a public blockchain like Ethereum, which can be significant for a high-volume system like toll collection.
  - **Private Blockchain:** If developer choose a private blockchain deployment, they will need to set up the necessary hardware infrastructure. This could involve multiple servers with adequate processing power, memory, and storage to run the blockchain nodes and maintain the network.

NOTE: These are general guidelines. The specific hardware requirements might vary depending on the complexity of your project, chosen tools, and deployment strategy.

1.3 Data Flow:

Data Flow Diagram (DFD) is a graphical representation of data flow in any system. It is capable of illustrating incoming data flow, outgoing data flow and store data. Data flow diagram describes anything about how data flows through the system.

Level 0 Data Flow Diagram (DFD) Level 0 is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes. It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

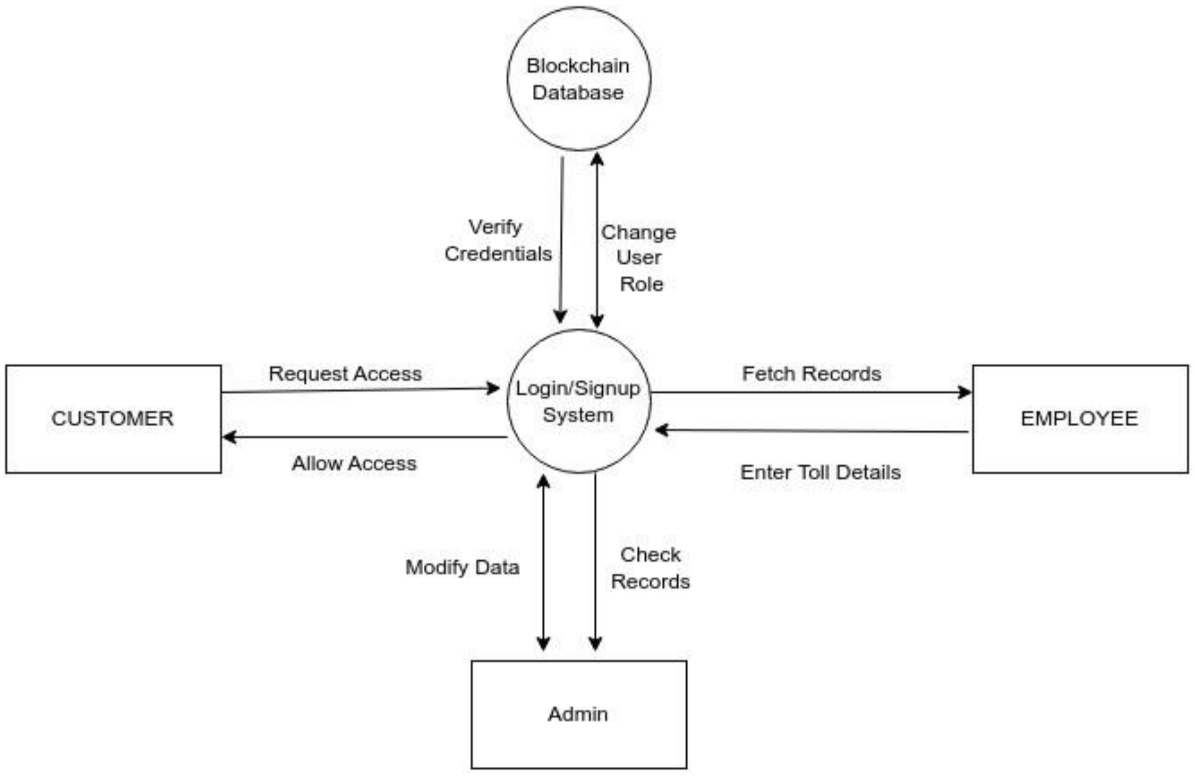


(Fig 3.2) Level 0 DFD diagram for the system

Here the Toll System serves as the main process, The three actors are Customer, Employee, and Admin:

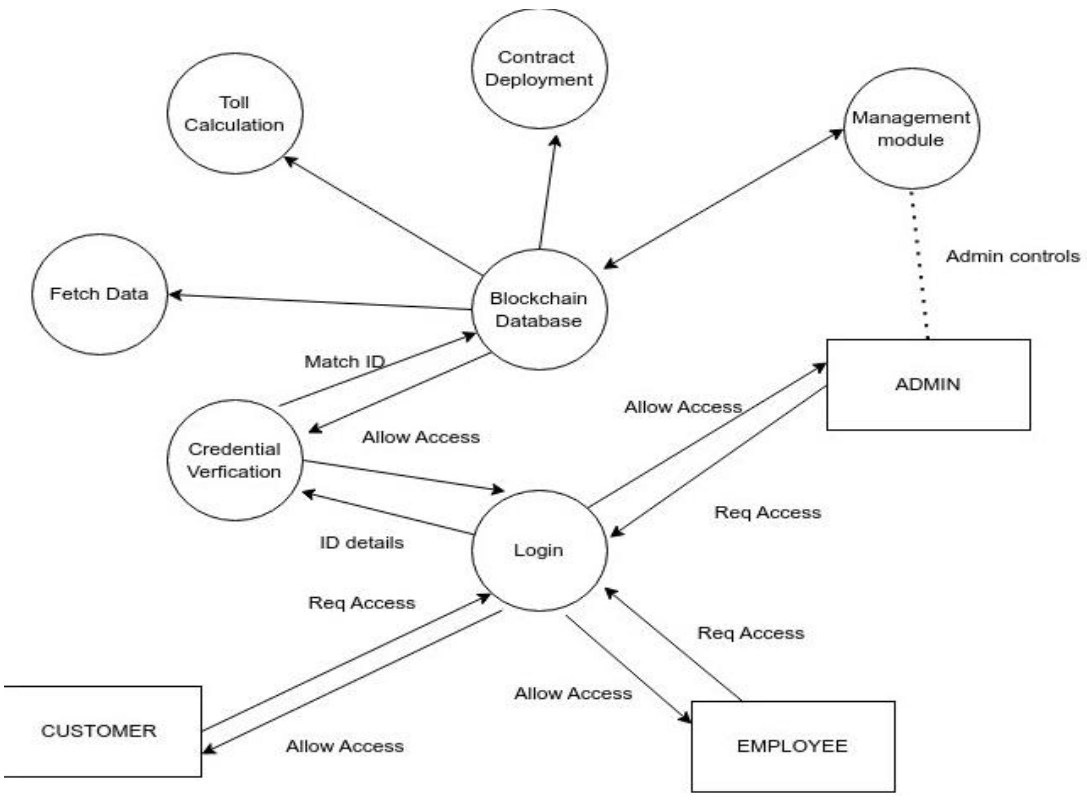
- Customer: Default user of the program, having least amount of access in the program. They can only access their own account details.
- Employee: This role is assigned by admin. They have moderate amount of access in the system. They are able to enter data in the system i.e. (the vehicle information to be entered when passing through toll plaza). They can also observe the list of users they entered the data for. But they are unable to edit the data once entered.

- Admin: This one has the highest amount of access in the system. They can change the user roles in the system as Customer and employee. On rare occasion a new admin maybe allowed based on proper protocols. Admin can also ban users if they no longer are following the rules and policies.



(Fig 3.3) Level 1 DFD Diagram for the system

The Data Flow Diagram (DFD) of Level 1shows a system where customers request access and verify credentials, employees log in or sign up and fetch records, and admins modify and check records. All these interactions involve a blockchain database that stores user information, credentials, and access permissions. The diagram highlights the flow of data between these user roles and the central database, ensuring secure and organized management of user roles and system records.



(Fig 3.4) Level 2 DFD Diagram for the System

The Level 2 Data Flow Diagram (DFD) shows a system where users, including customers, employees, and admins, interact with a blockchain database for access control. Customers request access, employees log in, and admins manage user roles. The system verifies user IDs against the blockchain database and either grants or denies access based on user roles. Admins have the authority to modify user roles within the database. The diagram emphasizes the login process and user verification against the blockchain database, without detailing post-login data flows.

REACT:

React is a powerful JavaScript library for building user interfaces (UIs). It allows developers to create reusable components that encapsulate both UI elements and their functionality. This component-based approach promotes code reusability, maintainability, and easier collaboration among developers.

Let’s get into what React offers:

- **Components:** The fundamental building blocks of React applications. Components are self-contained, reusable pieces of code that define how a UI element looks and behaves. They can be simple (like a button) or complex (like a login form).
- **JSX (JavaScript XML):** A syntax extension for JavaScript that allows you to write HTML-like structures within your code. This makes it easier to visualize and reason about the UI you're building. JSX is transformed into regular JavaScript code before being sent to the browser.
- **Virtual DOM (Document Object Model):** A lightweight representation of the UI in memory. When changes occur in the UI state, React efficiently calculates the minimal changes needed in the real DOM, resulting in faster updates and smoother user experiences.
- **State Management:** Components can maintain their own state, which represents the data that defines the component's current condition. Changes to the state trigger a re-render of the component, keeping the UI in sync with the underlying data.
- **Props:** A way to pass data down from parent components to child components. This allows for building flexible and reusable components that can be customized based on the data they receive.
- **Unidirectional Data Flow:** React promotes a unidirectional data flow, where data is passed down from parent components to child components, and changes bubble back up through event handlers. This makes reasoning about data flow and potential side effects easier.

Benefits of using React:

- **Declarative:** React focuses on describing what the UI should look like for a given state, rather than how to achieve it through complex imperative code. This makes code more readable and maintainable.
- **Component-based Architecture:** Components promote code reusability, modularity, and easier collaboration among developers working on the same project.
- **Virtual DOM:** React's virtual DOM optimization ensures efficient UI updates, leading to smoother user experiences and better performance.
- **Large Community and Ecosystem:** React has a vast developer community and a rich ecosystem of libraries and tools, providing extensive support and resources for developers.

1. Components - The Building Blocks of React Applications:

React applications are built using reusable components. Think of components as Lego bricks for your UI. Each component defines a specific part of your UI and its behaviour.

- **Types of Components:**
  - **Presentational Components:** These components focus on how the UI looks and how it renders the data it receives through props. They typically don't manage their own state. (e.g., a button component)
  - **Container Components:** These components handle more complex logic, manage state, and interact with other components or external APIs. They often contain presentational components as building blocks. (e.g., a login form component)

- **Component Lifecycle:** React components have a defined lifecycle with methods that are invoked at different stages, such as:
  - render (): Defines how the UI looks for a given state.
  - componentDidMount (): Invoked after the component is inserted into the DOM. Useful for fetching data or setting up subscriptions.
  - componentDidUpdate (): Invoked after the component updates due to changes in state or props.

2. JSX (JavaScript XML):

JSX is a syntax extension that allows you to write HTML-like structures within your JavaScript code. It improves readability and makes it easier to visualize the UI you're building. However, JSX is not actual HTML. It's transformed into plain JavaScript code before being sent to the browser.

3. Virtual DOM (Document Object Model):

The virtual DOM is a lightweight in-memory representation of the actual DOM (the HTML structure displayed in the browser). When a component's state changes, React efficiently calculates the minimal changes needed in the virtual DOM. Then, it updates only the necessary parts of the real DOM, resulting in faster UI updates and better performance.

4. State Management:

Components can maintain their own state, which represents the data that determines the component's current condition (e.g., the currently selected item in a dropdown menu). Changes to the state trigger a re-render of the component, keeping the UI in sync with the underlying data.

- **useState Hook:** A function introduced in React Hooks (introduced in React 16.8) to manage component state.
- **Lifting State Up:** When data needs to be shared by multiple components, the state can be "lifted up" to a common ancestor component that can then manage and provide the data to its children through props.

5. Props:

Props are a way to pass data down from parent components to child components. This allows for building flexible and reusable components that can be customized based on the data they receive. Props are read-only within the child component, promoting a unidirectional data flow.

Advantages of React:

- **Declarative:** React focuses on describing the UI for a given state, making code more readable and maintainable compared to imperative approaches.
- **Component-based Architecture:** Promotes code reusability, improves maintainability, and facilitates collaboration on large projects.

- **Virtual DOM:** Enables efficient UI updates, leading to smoother user experiences and better performance.
- **JSX:** Improves readability and makes it easier to reason about the UI you're building.
- **Unidirectional Data Flow:** Simplifies reasoning about data flow and helps avoid potential side effects.
- **Large Community and Ecosystem:** Extensive support, libraries, and tools are available for React development.

NODE JS:

Node.js is an open-source, cross-platform JavaScript runtime environment that allows you to execute JavaScript code outside of a web browser. This means you can use JavaScript to build server-side applications, command-line tools, and even network applications.

Event-Driven Architecture:

- Node.js is built on an event-driven architecture. This means that instead of waiting for one task to finish before starting another (like traditional web servers), Node.js can handle multiple requests concurrently. It efficiently manages events (like incoming requests, data received from a database) and triggers callbacks (functions) to handle them when they occur. This non-blocking approach allows Node.js to be highly scalable and efficient for handling a large number of concurrent connections.

JavaScript Everywhere:

- Traditionally, JavaScript was primarily used for adding interactivity to web pages within web browsers. Node.js allows developers to leverage their existing JavaScript knowledge to build applications beyond the browser. This can reduce the need to learn multiple programming languages for front-end and back-end development.

Package Ecosystem (npm):

- Node.js has a rich ecosystem of pre-built modules and libraries available through the Node Package Manager (npm). This vast repository allows developers to find and install code for various functionalities, saving them time and effort when building applications.

Popular Use Cases:

- **Web Servers:** Node.js is a popular choice for building real-time web applications (like chat applications) and APIs due to its non-blocking nature and event-driven architecture.
- **Microservices:** Node.js is well-suited for building small, modular, and independent services that can be combined to create complex applications.
- **Command-Line Tools (CLIs):** Node.js allows you to create powerful command-line tools for automating tasks or interacting with other systems.
- **Internet of Things (IoT):** Node.js can be used to build applications that interact with devices and sensors due to its lightweight and event-driven nature.

Advantages of Node.js:

- **Fast and Scalable:** The event-driven architecture allows Node.js to handle many concurrent connections efficiently.
- **JavaScript Everywhere:** Developers can leverage their existing JavaScript knowledge for both front-end and back-end development.
- **Rich Ecosystem:** The vast npm package repository provides readily available solutions for various functionalities.
- **Active Community:** Node.js has a large and active developer community providing support and resources.

ARCHITECTURAL EXPLANATION:

1. Event-Driven Architecture - Explained in Detail:

At the heart of Node.js lies its event-driven architecture. Unlike traditional web servers that handle requests one at a time, Node.js operates asynchronously, meaning it can handle multiple requests concurrently. This efficiency is achieved through the following mechanisms:

- **Event Loop:** The event loop is the core concept in Node.js. It's a single-threaded loop that continuously monitors for events (like incoming requests, data received from a database, or timers expiring).

- **Callbacks:** When an event occurs, the event loop triggers a corresponding callback function. This function contains the code that handles the event.
- **Non-Blocking, I/O (Input/Output):** Node.js uses non-blocking I/O operations. This means it doesn't wait for I/O operations (like reading from a file or network) to complete before moving on to the next event. Instead, it can handle other events while the I/O operation is in progress. Once the I/O operation finishes, a callback is added to the event queue to be executed later.

Benefits of Event-Driven Architecture:

- **Scalability:** Node.js can efficiently handle many concurrent connections because it's not blocked by slow I/O operations.
- **Performance:** The asynchronous nature allows Node.js to be more responsive, especially for tasks involving network or database interactions.
- **Resource Efficiency:** Since Node.js uses a single thread, it has a lower memory footprint compared to traditional multi-threaded servers.

2. Unveiling the Power of npm (Node Package Manager):

The Node Package Manager (npm) is an essential component of the Node.js ecosystem. It's the world's largest software registry containing an extensive collection of open-source packages (modules) that provide pre-written code for various functionalities.

- **Finding and Installing Packages:** You can use the npm install <package name> command to search for and install packages from the npm registry. These packages can be anything from web frameworks like Express.js to database drivers or utility libraries.
- **Package Versioning and Dependencies:** npm manages package versions and dependencies. When you install a package, it might also install other packages it depends on to function correctly. npm helps manage these dependencies and ensures compatibility between different package versions.
- **Community-driven Development:** The vast majority of npm packages are developed and maintained by the open-source community. This allows developers to find solutions for various needs and contribute back to the community by creating or improving packages.

3. Exploring Advanced Use Cases for Node.js:

While Node.js is popular for building web servers and APIs, its capabilities extend beyond that:

- **Microservices Architecture:** Node.js is well-suited for building microservices, which are small, independent, and modular services that can be combined to create complex applications. This approach promotes scalability, maintainability, and easier deployment.
- **Real-time Applications:** The event-driven nature of Node.js makes it ideal for building real-time applications like chat applications, online collaboration tools, or multiplayer games. These applications require constant communication between users, and Node.js can efficiently handle this flow.
- **Internet of Things (IoT):** With its lightweight and event-driven architecture, Node.js can be used to build applications that interact with devices and sensors in the Internet of Things (IoT) landscape. It can handle data streams from sensors, process information, and trigger actions based on real-time data.
- **Command-Line Tools (CLIs):** Node.js allows you to create powerful command-line tools (CLIs) for automating tasks, interacting with other systems, or managing infrastructure. These tools can be used to streamline development workflows or system administration tasks.

CSS:

In the context of our blockchain-based toll collection system, CSS (Cascading Style Sheets) plays a crucial role in defining the visual appearance and user experience of the frontend application. Here's how CSS would be applied:

1. **User Interface (UI) Styling:**

- **React Components:** When building the UI with React components, CSS can be used to style individual components. This allows for modular and maintainable styling, where each component has its own styles defined separately.
- **CSS Frameworks or Libraries:** You can leverage popular CSS frameworks like Bootstrap or Material UI to achieve a consistent look and feel across the application. These frameworks provide pre-built styles for common UI elements like buttons, forms, and navigation bars.
- **Custom Styles:** For more unique UI elements or to tailor the application's appearance to specific branding guidelines, you'll likely need to create custom CSS styles.

2. Responsiveness and Accessibility:

- **Responsive Design:** CSS media queries can be used to ensure the UI adapts seamlessly to different screen sizes (desktop, mobile, tablets). This is crucial for a system that might be accessed through various devices by drivers or toll booth personnel.
- **Accessibility:** CSS can be used to implement accessibility features that make the UI usable for people with disabilities. This might involve using proper colour contrast, ensuring proper focus styles for interactive elements, and potentially using semantic HTML elements for better screen reader compatibility.

3. Theming:

- **Dynamic Styles:** CSS can be used to implement dynamic themes where the user interface can change its appearance based on user preferences or system settings (e.g., light, or dark mode themes).

4. Integration with React:

There are several ways to integrate CSS styles with React components:

- **Inline Styles:** Inline styles can be defined directly within the JSX of a component, but this approach is generally discouraged due to maintainability concerns.
- **Styled Components Libraries:** Libraries like styled components allow you to write CSS directly within your JavaScript code, promoting better separation of concerns and style encapsulation within components.
- **CSS Modules:** This approach allows you to define local styles for each component using a unique class name prefix, preventing style conflicts when multiple components are used together.

1. Advanced CSS Techniques for Enhanced User Experience:

- **CSS Grid and Flexbox:** These layout systems offer more flexibility and control over the layout of UI elements compared to traditional floats. They can be used to create responsive layouts and complex UI structures efficiently.
- **CSS Animations and Transitions:** CSS animations and transitions can add subtle visual polish to the UI. For example, you could use animations for loading indicators, transitions for smoother UI state changes (like opening or closing menus), or subtle hover effects on buttons.
- **CSS Variables:** CSS variables allow you to define reusable values like colours, fonts, or spacing throughout your stylesheets. This promotes maintainability and easier theme changes (e.g., switching between light and dark mode themes).

2. Preprocessors and Build Tools:

- **CSS Preprocessors (like Sass or LESS):** These tools extend the functionality of CSS by allowing you to use variables, mix ins (reusable code snippets), and nesting for more organized and maintainable stylesheets.
- **CSS Build Tools (like Gulp or Webpack):** These tools automate tasks like compiling preprocessor code, minifying CSS (reducing file size for faster loading), and potentially combining multiple CSS files into a single file for better performance.

3. Security Considerations in CSS:

- **Sanitize User Input:** Be cautious when using user-generated content within CSS styles. Sanitize any user input to prevent potential security vulnerabilities like cross-site scripting (XSS) attacks.

- **Avoid Inline Styles with Sensitive Data:** Inline styles within JSX should not be used for sensitive data like user information or API keys. Store such data securely in the backend and use CSS classes or dynamic styles to apply them to the UI.

4. Best Practices for Maintainable CSS:

- **Modular Styles with Separation of Concerns:** Organize your styles according to components or features to improve maintainability and prevent conflicts.
- **Meaningful Class Names:** Use descriptive and consistent class names that reflect the purpose of the style they apply. Avoid overly generic class names that can lead to confusion.
- **Proper Documentation:** Document your CSS styles, especially for custom styles or complex layouts. This can help other developers understand the purpose of specific styles and how they are used.

1. Micro interactions - Subtle Cues for a Delightful Experience:

Micro interactions are subtle visual, animated, or auditory cues that enhance user interaction with the UI. They can make the application feel more responsive, engaging, and polished. Here are some CSS techniques for micro interactions:

- **Hover Effects:** Change the appearance (opacity, colour, or slight transformation) of buttons, links, or other interactive elements on hover to provide visual feedback that an action can be taken.
- **Focus Styles:** Use CSS to style the visual appearance of an element when it receives focus (e.g., when a user tabs to an input field). This can help guide the user and improve accessibility.
- **Loading Indicators:** Create visually appealing loading spinners or animations using CSS animations to indicate ongoing processes (e.g., fetching data from the blockchain).
- **Success and Error States:** Use subtle colour changes, animations, or icons to visually communicate successful actions (e.g., balance top-up) or error conditions (e.g., insufficient balance).

2. Performance Optimization for a Smooth User Experience:

Fast loading times and a smooth user experience are crucial for any web application. Here's how CSS can contribute to performance optimization:

- **Critical CSS:** Identify and extract the critical CSS styles needed for initial page load and render them inline or in a separate critical CSS file. This ensures a faster perceived loading time.
- **CSS Specificity:** Manage CSS specificity carefully to avoid unintended style overrides and potential performance bottlenecks. Consider using tools like Specificity Finder [https://specificity.keegan.st/] to identify potential conflicts.

- **Image Optimization:** Images can significantly impact page load times. Use image optimization techniques like resizing, compression, and lazy loading to ensure images load efficiently.

3. Futureproofing with Emerging CSS Trends:

The world of CSS is constantly evolving. Here are some emerging trends you might consider for the future:

- **CSS Houdini:** An umbrella term for a set of new CSS features that provide more low-level control over the browser's rendering engine. This allows for more creative and performant animations and visual effects.
- **CSS Grid Level 2:** The second level of the CSS Grid specification offers even more flexibility for complex layouts, with features like sub grids and named grid areas.
- **Variable Fonts:** Variable fonts allow a single font file to contain multiple styles (e.g., weight, width) within the same file. This can reduce the number of HTTP requests needed for fonts, improving performance.

INTEGRATIONS:

In the context of the blockchain-based toll collection system, Web3 refers to a conceptual shift in how web applications interact with data and users. Here's how Web3 principles can be applied to this project:

1. Decentralization:

- **Traditional Toll Collection:** Currently, toll collection systems are often centralized, with a single entity controlling the data (transactions, user accounts) and the infrastructure.
- **Web3 Approach:** By leveraging blockchain technology, the toll collection system can be decentralized. Data about transactions, user accounts, and potentially even road maintenance records could be stored on a distributed ledger (blockchain). This would remove the reliance on a single central authority and potentially increase transparency and trust.

2. User Ownership and Control:

- **Traditional System:** Users have limited control over their data in traditional toll collection systems.
- **Web3 Approach:** In a Web3 system, users could hold cryptographically secure tokens representing their accounts and potentially even loyalty points earned through toll payments. This would give them more control over their data and potentially enable them to interact with other blockchain-based applications in the future (if the system is designed to be interoperable).

3. Potential Benefits:

- **Increased Security and Transparency:** Blockchain technology offers a high degree of security and immutability for data storage. This can potentially reduce the risk of fraud or manipulation of toll data. Additionally, a transparent and auditable ledger can increase public trust in the system.
- **Improved Efficiency:** Automation of toll collection processes through smart contracts (self-executing code on the blockchain) could potentially improve efficiency and reduce administrative costs.
- **Potential for Innovation:** Web3 opens doors for future innovation. For example, the system could integrate with other blockchain-based applications for seamless payments or loyalty programs.

4. Challenges and Considerations:

- **Scalability and Transaction Fees:** Current public blockchains might not yet be scalable enough to handle the high transaction volume of a large-scale toll collection system. Private or consortium blockchains could be considered as alternatives. Additionally, transaction fees on some blockchains can be high, potentially impacting the feasibility of the system.

- **User Adoption and Education:** Web3 technologies are still relatively new, and user adoption might be a challenge. Educating users about the benefits and security aspects of the system is crucial.

1. **Decentralized Infrastructure using Blockchain Technology:**

- **Moving Beyond Centralized Control:** Traditional toll collection systems rely on a central authority to manage data (transactions, user accounts) and infrastructure. This can create a single point of failure and limit transparency.

- **Web3 Approach:** By utilizing blockchain technology, the system can be decentralized. Data is stored on a distributed ledger, accessible by all participants in the network. This eliminates the need for a central authority, fostering trust and potentially increasing data integrity.

2. **User Empowerment through Crypto Wallets and Tokens:**

- **Limited User Control in Traditional Systems:** In traditional systems, users have minimal control over their data and limited ways to interact with the system beyond paying tolls.
- **Web3 Approach:** Users can leverage crypto wallets to hold tokens representing their accounts and potentially even loyalty points earned through toll payments. These tokens could be based on various standards like ERC-20 (Ethereum) or SPL (Solana).

Benefits of User-owned Tokens:

- **Enhanced Security:** Crypto wallets provide users with cryptographic control over their tokens, potentially reducing the risk of unauthorized access compared to traditional account systems.

- **Interoperability Potential:** Standardized tokens could enable future integration with other blockchain-based applications, allowing users to spend their earned loyalty points in a wider ecosystem (if designed with interoperability in mind).
- **Transparency and Traceability:** All transactions involving tokens would be recorded immutably on the blockchain, providing users with a transparent record of their toll payments and potential loyalty program activity.

3. Potential Advantages of a Web3-based Toll Collection System:

- **Increased Security and Trust:** Blockchain technology offers a high degree of security and immutability for data storage. This can potentially reduce the risk of fraud or manipulation of toll data. Additionally, a transparent and auditable ledger can increase public trust in the system.
- **Improved Efficiency and Automation:** Smart contracts, self-executing code on the blockchain, can automate toll collection processes. This could streamline operations, reduce administrative costs, and potentially eliminate the need for manual toll booths.
- **Potential for Innovation:** Web3 opens doors for future innovation. The system could integrate with other blockchain-based applications for features like seamless payments, dynamic pricing based on traffic conditions, or even usage-based insurance models.

4. Challenges and Considerations for Web3 Implementation:

- **Scalability and Transaction Fees:** Public blockchains like Ethereum might not yet be scalable enough to handle the high transaction volume of a large-scale toll collection system. Private or consortium blockchains could be explored as alternatives, with trade-offs in terms of decentralization. Additionally, transaction fees on some blockchains can be high, potentially impacting the feasibility of the system.
- **User Adoption and Education:** Web3 technologies are still relatively new, and user adoption might be a challenge. Educating users about the benefits

and security aspects of the system, including crypto wallet management, is crucial for successful implementation.

- **Regulatory Landscape:** The regulatory landscape around blockchain and cryptocurrencies is still evolving. Staying updated on regulations and ensuring compliance is essential.

6. Additional Considerations:

- **Security Audits for Smart Contracts:** Thorough security audits of the smart contracts managing toll collection logic are crucial to ensure the system is robust against potential vulnerabilities and exploits.
- **Integration with React Frontend:** The Web3 backend with smart contracts needs to securely interact with the React frontend application. APIs can be built to facilitate data exchange and user interactions with the blockchain and their crypto wallets.
- **Phased Implementation:** A phased approach might be considered, initially deploying a hybrid system that leverages existing infrastructure while gradually integrating Web3 functionalities.

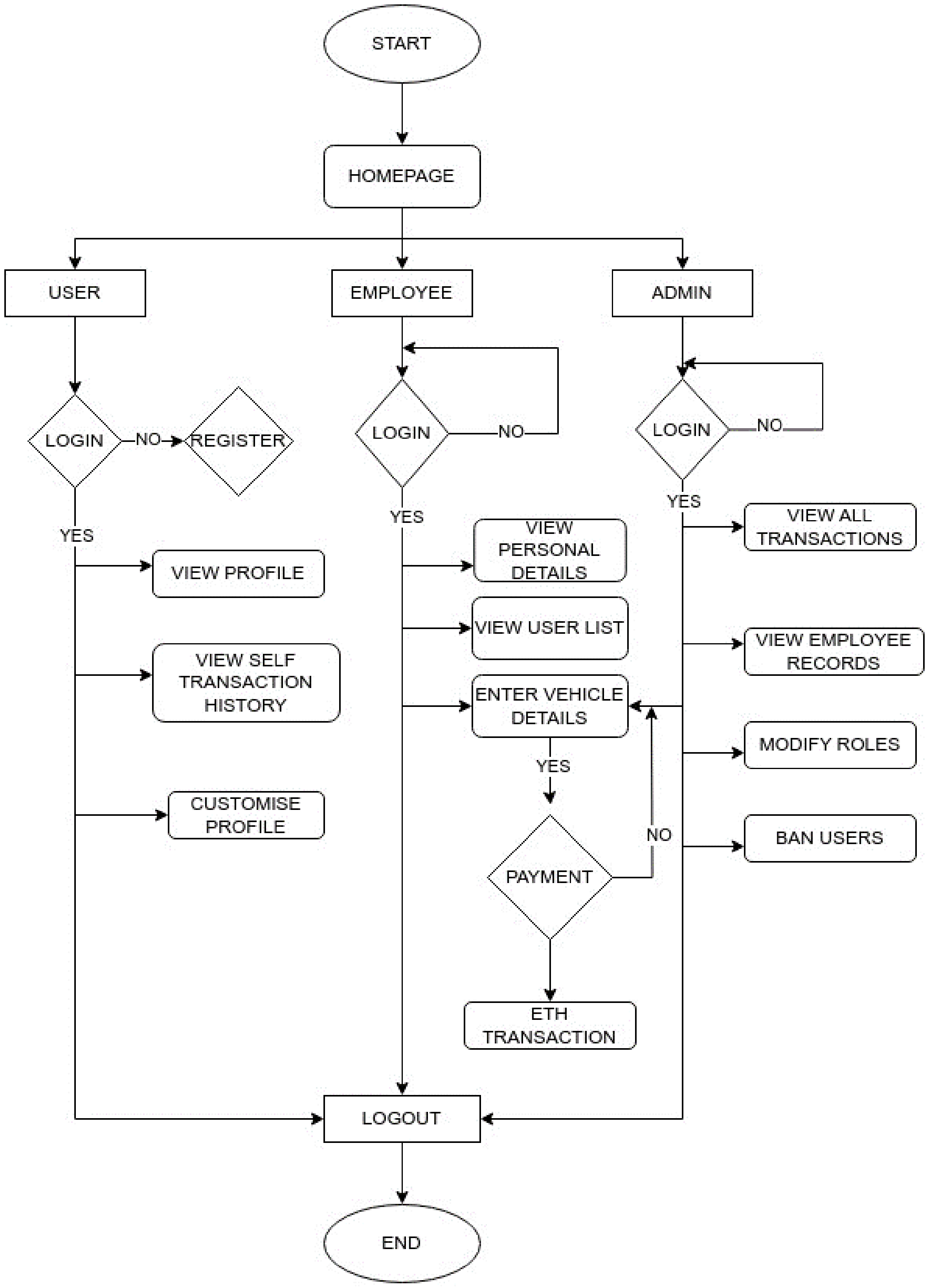


Fig. 3.5 Flowchart of the system

Explanation:

1- User Initiates the program.

2- The react app opens the home page.

3- Here the Login/signup page is available

4- User can log in based on their role.

5- If no account then signup as user role by default.

6- User role allows us to View our profile, Transaction history, and also customize our own profile.

7- Other roles are granted by the admin.

8- Now logging in as employee.

9- Now you can, view details, user list, enter vehicle details, do payment transaction.

10- Now login as admin.

11- Supreme authority on the system.

12- You can view all transactions, view employee records, modify roles, ban users.

13- Log out of the system.

Further Explanation:

1. Homepage and Login:

- The user starts on the homepage and enters their login credentials (username and password).

2. Registration Check:

- The system checks if the user is registered.
    - If not registered, the user is directed to a registration page to sign up.
    - If registered, the system checks the user's role.

3. User Role Identification:

- The system identifies the user's role, which could be:
    - User
    - Employee
    - Admin

4. Role-Based Navigation:

- Depending on the user's role, the system directs the user to different sections:

- User:

  - View profile
  - View transaction history
  - Customize profile

- Employee:

  - View profile

- View personal details

- View self-transaction history

- Process payments

- Enter vehicle details

- Admin:

- View all transactions

- View user list

- View employee records

- Modify roles

- Ban users

5. Logout:

- Users can log out, returning them to the homepage.

This outline provides the login system with the different user roles and the actions they can perform.

1. Decentralized Governance through DAOs (Decentralized Autonomous Organizations):

- **Traditional Governance:** Currently, decisions regarding toll collection systems are likely made by centralized authorities.
- **Web3 Approach:** The system could leverage a DAO to facilitate decentralized governance. Token holders could participate in voting on proposals related to toll pricing, system upgrades, or even profit sharing from the collected fees.

Benefits of DAO Governance:

- **Community Involvement:** Token holders have a stake in the system and can participate in decision-making, fostering a sense of ownership and community.
- **Transparency and Fairness:** DAO proposals and voting records are typically publicly viewable on the blockchain, promoting transparency and fairness in decision-making.
- **Flexibility and Adaptability:** DAOs allow for more agile governance compared to traditional models, enabling the system to adapt to changing needs and user feedback.

Challenges and Considerations:

- **Reaching Consensus:** Finding consensus among token holders for proposals can be challenging, especially with a large and diverse user base.
- **Security of Voting Mechanisms:** The security of the DAO's voting mechanisms needs careful consideration to prevent manipulation or attacks.
- **Regulatory Uncertainty:** The legal and regulatory landscape surrounding DAOs is still evolving. Careful legal counsel is recommended when designing the DAO structure.

2. Interoperability for a Broader Ecosystem:

- **Limited Functionality in Siloed Systems:** Traditional toll collection systems often operate in isolation, limiting user options and benefits.
- **Web3 Approach:** By leveraging interoperable token standards and blockchain protocols, the system could integrate with other blockchain-based applications.

Benefits of Interoperability:

- **Seamless Payments and Integration:** Users could potentially use their toll tokens for other purposes within a broader blockchain ecosystem, such as paying for parking, electric vehicle charging, or even ride-sharing services.

- **Increased User Utility:** Interoperability expands the utility of the system's tokens, potentially attracting more users and driving adoption.
- **Innovation and Future Applications:** Interoperability opens doors for unforeseen future applications and partnerships within the broader Web3 landscape.

Challenges and Considerations:

- **Standardization and Compatibility:** Ensuring compatibility with different blockchains and token standards can be complex. Careful selection of protocols and standards is crucial for successful interoperability.
- **Technical Complexity:** Integrating with other blockchain-based applications can add technical complexity to the system development process.
- **Network Effects and Ecosystem Development:** Interoperability benefits increase as more participants join the ecosystem. Building a strong network effect and fostering collaboration with other blockchain projects is essential.

3. Scalability Solutions for High Transaction Volume:

- **Limitations of Public Blockchains:** Public blockchains like Ethereum might not yet handle the high transaction volume required for a large-scale toll collection system due to scalability limitations.
- **Web3 Approach:** Several potential solutions exist for scaling the system:
    - **Layer 2 Solutions:** These solutions process transactions off the main blockchain, reducing congestion on the main chain and potentially increasing transaction throughput. Examples include Polygon or Lightning Network.
    - **Private or Consortium Blockchains:** These permissioned blockchains offer more control and scalability but come with trade-offs in terms of decentralization.

    - **Rollups:** A type of Layer 2 solution that bundles multiple transactions into a single transaction on the main chain, improving scalability while maintaining security benefits.

Challenges and Considerations:

- **Choosing the Right Solution:** The choice of scaling solution depends on factors like desired transaction speed, security requirements, and the level of decentralization needed for the system.
- **Integration Complexity:** Integrating with Layer 2 solutions or alternative blockchains can add complexity to the system architecture.
- **Future-proofing the System:** Scalability solutions should be chosen with an eye towards future growth and potential increases in transaction volume.

Conclusion:

By exploring advanced Web3 concepts like DAO governance, interoperability, and scalable solutions, the blockchain-based toll collection system can move towards a more robust, user-centric, and future-proof design. Remember, Web3 is a rapidly evolving landscape.

CHAPTER 4
RESULTS AND TESTINGS

Results and Testing of the Blockchain-Based Toll Collection System

Results:

1. Transparency and Accountability:

- Blockchain Integration: All transactions are securely recorded on the blockchain, creating an immutable ledger. This ensures that once a transaction is recorded, it cannot be altered or deleted, providing transparency and traceability for all toll payments. Users and auditors can independently verify transactions, building trust and accountability.

- Audit Trails: Detailed audit trails are automatically created, offering a clear history of all transactions. This is crucial for regulatory compliance and resolving disputes.

2. Improved Efficiency:

- Smart Contracts: Automate the toll collection process, reducing the need for manual intervention. This results in faster transaction processing and minimizes the chances of human error.

- Scalability: The system can handle many transactions simultaneously, making it suitable for busy toll roads. High throughput ensures efficient processing of multiple transactions.

- Reduced Operational Costs: Automation and reduced need for physical infrastructure (like toll booths) lower operational costs.

3. Enhanced Security:

- Cryptographic Security: Advanced cryptographic algorithms secure data. Public-key cryptography ensures that only authorized parties can initiate or confirm transactions.

- Decentralization: A decentralized network reduces the risk of a single point of failure, making the system more resilient to attacks.

- Tamper Resistance: Data recorded on the blockchain is nearly impossible to alter, preventing fraud and ensuring reliable toll transactions.

4. User Satisfaction:

- Real-Time Updates: Users receive instant notifications for each toll transaction, enhancing confidence and reducing uncertainties.

- User Interface: The React-based front end provides a responsive and intuitive experience. Features like seamless navigation, clear transaction histories, and easy account management improve user satisfaction.

- Customer Support: Integrated support features like chatbots or help desks ensure users can quickly resolve issues.

Results of the Frontend Using React with Admin, Employee, and User Interfaces

In the blockchain-based toll collection system, the frontend is developed using React, creating distinct interfaces for Admin, Employee, and User roles. Node.js and Local Storage handle backend interactions, with Ganache serving as the local blockchain environment for testing.

Admin Interface:

Uses:

- Dashboard Overview: Access to real-time data on toll collections, system performance, and user activities.

- User Management: View, add, edit, and remove users (both employees and toll users). Assign roles and permissions.

- Transaction Monitoring: Detailed logs of all transactions for auditing and compliance.

- System Configuration: Configure system settings, update toll rates, and manage blockchain nodes.

Results:

- Efficiency: Responsive and dynamic dashboard updating in real-time.
- Control: Granular control over user permissions and system settings.
- Transparency: Detailed transaction logs enhance transparency and accountability.

Employee Interface:

Uses:

- Toll Collection Management: Manage toll collection points, verifying and processing toll transactions.
- User Assistance: Access user accounts for troubleshooting, updating information, and processing manual payments.
- Reporting: Generate and view reports on toll collection statistics, shifts, and transaction summaries.

Results:

- User Assistance: Real-time access to user accounts and transaction data for prompt assistance.
- Efficiency: Automated transaction processing and reporting reduce manual workload.
- Accuracy: Real-time updates and validation through the blockchain reduce errors.

User Interface:

Uses:

- Account Management: Create and manage accounts, view transaction history, and update personal information.

- Toll Payments: Link payment methods, check account balances, and make toll payments.

- Transaction History: Access detailed history of toll transactions, including timestamps and amounts paid.

Results:

- User Experience: Intuitive navigation ensures a positive user experience with real-time updates and notifications.

- Transparency: Users can verify their transactions on the blockchain.

- Convenience: Seamless toll payment processing and account management improve user satisfaction.

Backend: Node.js, Local Storage, and Ganache:

Node.js:

- Server-Side Logic: Processes requests from the frontend and interacts with the blockchain.

- API Development: Facilitates communication between the React frontend and the blockchain.

- Scalability: Non-blocking I/O model handles multiple simultaneous requests efficiently.

Local Storage:

- Session Management: Keeps users logged in and maintains state across different pages.

- Data Caching: Reduces the need for frequent blockchain queries, improving performance.

Ganache:

- Testing Environment: Provides a local Ethereum blockchain environment for testing.

- Transaction Simulation: Allows developers to simulate transactions and monitor their effects before deployment.

Combined Results

- Seamless Integration: React interfaces for Admin, Employee, and User roles interact with the backend via Node.js, ensuring smooth data flow and real-time updates.

- Enhanced User Experience: Real-time responsiveness and intuitive interfaces enhance the user experience for all roles.

- Security and Transparency: Blockchain integration ensures transaction transparency and security, with Ganache providing a testing environment to validate functionality.

- Efficiency and Scalability: Node.js and local storage optimize performance and scalability, handling high transaction volumes and multiple user interactions concurrently.

By integrating React for the frontend, Node.js for backend logic, local storage for session management, and Ganache for blockchain testing, the toll collection system achieves high performance, security, and user satisfaction across all user interfaces.

Testing:

1. Functional Testing:

- Smart Contract Testing: Ensuring that smart contracts perform as expected, including toll payment processing, account balance updates, and error handling.

- UI/UX Testing: Testing the functionality and usability of each UI component, including form submissions, navigation, and user interactions.

2. Integration Testing:

- Backend Integration: Verifying that the integration between the React front end and the blockchain backend ensures seamless data flow.

- API Testing: Testing all APIs used for communication between the toll system and external services to ensure correct functionality and error handling.

3. Performance Testing:

- Load Testing: Simulating high volumes of transactions to test the system's ability to handle peak loads and identify performance bottlenecks.

- Stress Testing: Subjecting the system to extreme conditions to test robustness and identify breaking points.

4. Security Testing:

- Penetration Testing: Conducting simulated cyber-attacks to identify and address vulnerabilities, including testing for common security threats.

- Encryption Verification: Ensuring all sensitive data is encrypted both in transit and at rest and verifying key management practices.

- Access Control: Verifying that access controls prevent unauthorized access to the system.

5. User Acceptance Testing (UAT):

- Real-World Scenarios: Testing the system with real users to ensure it meets their needs and expectations, performing typical tasks like paying tolls and managing accounts.

- Feedback Incorporation: Gathering user feedback and incorporating suggestions to improve the system.

Testing Phase for the Blockchain-Based Toll Collection System:

Admin Interface Testing:

Uses:

- Dashboard Overview: Ensure real-time data on toll collections, system performance, and user activities are accurate and promptly updated.

- User Management: Verify functionalities for adding, editing, and removing users. Test role-based access controls and permission settings.

- Transaction Monitoring: Check the accuracy and completeness of transaction logs and ensure audit trails are correctly generated.

- System Configuration: Test configurations for system settings, toll rate updates, and blockchain node management.

Results:

- Functionality: Admin functionalities perform as expected without errors. Edge cases such as invalid input or permission violations are thoroughly tested.

- Performance: The dashboard's real-time data display handles peak loads without lag.

- Security: Admin access controls prevent unauthorized access. Data transactions and configurations maintain integrity and security.

Employee Interface Testing

Uses:

- Toll Collection Management: Verify the processing of toll transactions, ensuring smooth operation and correct deductions.

- User Assistance: Ensure employees can access user accounts for troubleshooting, updating information, and processing manual payments.

- Reporting: Test the generation and accuracy of toll collection reports and transaction summaries.

Results:

- Functionality: Toll collection management and user assistance functionalities are accurate and reliable.

- Performance: Employee interfaces handle high volumes of transactions and user queries efficiently.

- Security: Access controls ensure employees have appropriate permissions, and sensitive user information is protected.

User Interface Testing

Uses:

- Account Management: Test functionalities for account creation, management, viewing transaction history, and updating personal information.

- Toll Payments: Ensure users can link payment methods, check balances, and make toll payments without issues.

- Transaction History: Verify the accuracy and completeness of transaction history, including timestamps and amounts paid.

Results:

- Functionality: User functionalities are easy to use and accurate, performing as expected.

- Performance: The user interface is responsive and performs well during peak usage times.

- Security: User data is protected against unauthorized access, ensuring all transactions are secure and private.

Backend Testing: Node.js, Local Storage, and Ganache

Node.js:

- Server-Side Logic: Test the backend logic to ensure requests from the frontend are processed correctly and efficiently.

- API Development: Test APIs for secure and accurate communication between the React frontend and the blockchain.

- Scalability: Stress-test the Node.js backend to ensure it handles multiple simultaneous requests without performance degradation.

Local Storage:

- Session Management: Test session persistence to ensure users remain logged in across different pages and sessions.

- Data Caching: Verify that local storage effectively caches temporary data, reducing the need for frequent blockchain queries and improving performance.

Ganache:

- Testing Environment: Use Ganache to simulate the Ethereum blockchain for safe testing of smart contracts and interactions.

- Transaction Simulation: Test simulated transactions to ensure smart contracts work as intended and all interactions are correctly recorded on the blockchain.

Combined Testing Results

- Functional Testing: All components of the system (Admin, Employee, User interfaces) perform as intended, with no bugs or errors in typical and edge case scenarios.

- Integration Testing: Data flows and interactions between the frontend and backend, and between the various user interfaces and the blockchain, are seamless.

- Performance Testing: The system handles high transaction volumes and user interactions without lag or failures during peak loads.

- Security Testing: Data is protected from unauthorized access, and the system is secure from common vulnerabilities.

Thorough testing across all these areas ensures the blockchain-based toll collection system operates smoothly and reliably, leveraging Node.js for backend processing, Local Storage for session management, and Ganache for blockchain testing. This rigorous approach ensures high performance, security, and user satisfaction before deployment.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

The blockchain-based toll collection system marks a significant advancement in toll management technology, addressing critical issues such as transaction transparency, efficiency, and fraud prevention. Leveraging blockchain technology ensures unparalleled transparency and security, with each transaction recorded on the immutable blockchain ledger, reducing fraud and building trust among users, employees, and administrators.

Key Achievements

Enhanced Transparency and Security:

Transactions recorded on the blockchain ensure transparency and immutability, reducing the possibility of fraud. Cryptographic techniques secure data, making it tamper-proof and enhancing overall security.

Improved Efficiency:

Automating toll collection processes reduces manual intervention, minimizes errors, and speeds up transaction processing, leading to quicker and more convenient toll payments for users and streamlined operations for administrators and employees.

User-Centric Design:

Tailored interfaces for admin, employee, and general user roles ensure an intuitive and user-friendly experience. Admins have comprehensive control, employees manage tolls efficiently, and users experience seamless account management and toll payments.

Scalability and Flexibility:

Built to be scalable, the system can handle increasing transaction volumes and users. Node.js backend processing and local storage for session management ensure efficient performance, while blockchain's decentralized nature allows for easy integration of new features.

Reliable Testing and Deployment:

Rigorous testing using Ganache for local blockchain testing ensured all components functioned as intended before deployment, meeting security and performance standards.

Challenges and Solutions

Integration with Existing Infrastructure:

Careful integration planning allowed for a smooth transition without disrupting ongoing operations, ensuring compatibility with existing technologies like RFID and FASTag.

User Adoption:

Comprehensive user education and support, including user guides and responsive customer support, facilitated the transition to the new system.

Regulatory Compliance:

Adherence to local and international regulations regarding data privacy, transaction security, and financial transparency was ensured through regular audits and updates.

Future Prospects:

Advanced Data Analytics:

Integration of advanced data analytics can provide deeper insights into toll collection patterns and system performance, informing decision-making and strategic planning.

Expansion to Other Areas:

The system's principles and technologies can be extended to other transportation and infrastructure management areas such as parking management and public transportation ticketing.

Continuous Improvement:

Feedback from users and stakeholders will drive continuous improvement, ensuring the system remains at the forefront of toll collection technology with regular updates and feature enhancements.

Conclusion Summary:

The blockchain-based toll collection system represents a significant advancement in toll management, offering enhanced transparency, security, and efficiency. Its successful implementation demonstrates the potential of blockchain technology in public infrastructure management, paving the way for more innovative solutions in the future.

APPENDIX

Management.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;

// Import the UserManager contract from the "users.sol" file
import "./users.sol";

// Declare the Management contract which inherits from UserManager
contract Management is UserManager {

    // Define a struct for Vehicle with properties
    struct Vehicle {
        string vehicleType; // Type of vehicle
        address ownerAddress; // Address of the owner
        string color; // Color of the vehicle
        bool insuranceStatus; // Insurance status of the vehicle
    }

    // Define a struct for Highway with properties
    struct Highway {
        string highway_no; // Highway number or identifier
        string[] highwaypoints; // Array of points on the highway
        uint256 tollRate; // Toll rate for the highway
        mapping(string => uint128) highwayspoints; // Mapping of highway points to their
displacement values
    }
```

```
// Define a struct for Toll with properties
struct Toll {
    string vehicleNo; // Vehicle number
    uint256 Time; // Timestamp of when the toll was added
    string startingPoint; // Starting point of the toll journey
    string endingPoint; // Ending point of the toll journey
    uint256 totalDistance; // Total distance traveled
    string highwayNo; // Highway number
    address issuedby; // Address of the issuer
}

// Define a struct for Challan (fine/ticket) with properties
struct Challan {
    string vehicleNo; // Vehicle number
    uint256 challanPrice; // Price of the challan
    string reason; // Reason for the challan
    address issuedby; // Address of the issuer
}

// State variables
string[] vehicletypes; // Array of vehicle types
Toll[] tolldetail; // Array of toll details
Challan[] challan; // Array of challan details
uint tollcount; // Count of tolls
uint challancount; // Count of challans

// Mappings
mapping(string => Highway) highway; // Mapping of highway number to Highway struct
mapping(string => uint) vehicletype; // Mapping of vehicle type to toll rate
mapping(string => Vehicle) vehicledetails; // Mapping of vehicle number to Vehicle struct
```

```
// Events for logging
event ChallanIssued(string vehicleNo, uint256 challanPrice, string reason, address indexed issuedby);
event TollAdded(string vehicleNo, string startingPoint, string endingPoint, uint256 totalDistance, string highwayNo, address indexed issuedby);

// Function to add a new vehicle type with its toll rate
function addvehicletype(string memory types, uint32 tollrate) public onlyAdmin {
    vehicletypes.push(types); // Add the vehicle type to the array
    vehicletype[types] = tollrate; // Set the toll rate for this vehicle type
}

// Function to add a new vehicle
function addvehicle(string memory reg_no, address owner, string memory vehicle, string memory colour, bool reg_status) external onlyAdmin {
    vehicledetails[reg_no].ownerAddress = owner; // Set the owner address
    vehicledetails[reg_no].vehicleType = vehicle; // Set the vehicle type
    vehicledetails[reg_no].color = colour; // Set the color
    vehicledetails[reg_no].insuranceStatus = reg_status; // Set the insurance status
    users[owner].vehicleids.push(reg_no); // Add the vehicle registration number to the user's list of vehicles
}

// Function to add a point to a highway with its displacement
function addhighwaypoints(string memory highway_no, string memory pointname, uint128 displacement) public onlyAdmin {
    highway[highway_no].highwaypoints.push(pointname); // Add the point name to the highway's points array
    highway[highway_no].highwayspoints[pointname] = displacement; // Set the displacement for this point
}
```

```
    // Function to add a new highway
    function addhighway(string memory highwayname, uint256 tollrate) external onlyAdmin {
        highway[highwayname].highway_no = highwayname; // Set the highway number
        highway[highwayname].tollRate = tollrate; // Set the toll rate for the highway
    }

    // Function to add a new challan
    function addChallan(string memory _vehicleNo, uint256 _challanPrice, string memory
_reason) external onlyAdminOrEmployee {
        Challan memory newChallan = Challan(_vehicleNo, _challanPrice, _reason, msg.sender);
// Create a new Challan struct
        challan.push(newChallan); // Add the challan to the array
        emit ChallanIssued(_vehicleNo, _challanPrice, _reason, msg.sender); // Emit an event for
the new challan
        User storage owner = users[vehicledetails[_vehicleNo].ownerAddress]; // Get the owner
of the vehicle
        owner.challanids.push(challancount); // Add the challan ID to the user's list of challans
        challancount++; // Increment the challan count
    }

    // Function to add a new toll
    function addToll(string memory _vehicleNo, string memory _startingPoint, string memory
_endingPoint, uint256 _totalDistance, string memory _highwayNo) external
onlyAdminOrEmployee {
        Toll memory newtoll = Toll(_vehicleNo, block.timestamp, _startingPoint, _endingPoint,
_totalDistance, _highwayNo, msg.sender); // Create a new Toll struct
        tolldetail.push(newtoll); // Add the toll to the array
        emit TollAdded(_vehicleNo, _startingPoint, _endingPoint, _totalDistance, _highwayNo,
msg.sender); // Emit an event for the new toll
        User storage owner = users[vehicledetails[_vehicleNo].ownerAddress]; // Get the owner
of the vehicle
        owner.tollids.push(tollcount); // Add the toll ID to the user's list of tolls
        tollcount++; // Increment the toll count
    }
    // Function to get all points and their displacements for a given highway
    function getAllHighwayPoints(string memory highway_no) external view onlyAdminOrEmployee returns (string[] memory, uint128[] memory) {
        require(highway[highway_no].tollRate != 0, "Highway does not exist"); // Ensure the highway exists

        string[] memory points = new string[](highway[highway_no].highwaypoints.length); // Create an array for points
        uint128[] memory displacements = new uint128[](highway[highway_no].highwaypoints.length); // Create an array for displacements

        for (uint256 i = 0; i < highway[highway_no].highwaypoints.length; i++) {
            string memory point = highway[highway_no].highwaypoints[i]; // Get the point name
            points[i] = point; // Add the point to the array
            uint128 displacement = highway[highway_no].highwayspoints[point]; // Get the displacement for the point
            displacements[i] = displacement; // Add the displacement to the array
        }

        return (points, displacements); // Return the points and displacements
    }

    // Function to get all tolls
    function getAllTolls() external view onlyAdmin returns (Toll[] memory) {
        return tolldetail; // Return the array of toll details
    }

    // Function to get all challans
    function getAllChallans() external view onlyAdmin returns (Challan[] memory) {
        return challan; // Return the array of challan details
    }
    // Function to get all highways with their details
    function getAllHighways() external view onlyAdmin returns (string[] memory, uint256[] memory, uint128[][] memory) {
        string[] memory highwayNumbers = new string[](vehicletypes.length); // Create an array for highway numbers
        uint256[] memory tollRates = new uint256[](vehicletypes.length); // Create an array for toll rates
        uint128[][] memory highwayPointsDisplacements = new uint128[][](vehicletypes.length); // Create a 2D array for points displacements

        for (uint256 i = 0; i < vehicletypes.length; i++) {
            Highway storage currentHighway = highway[vehicletypes[i]]; // Get the current highway
            highwayNumbers[i] = currentHighway.highway_no; // Add the highway number to the array
            tollRates[i] = currentHighway.tollRate; // Add the toll rate to the array

            uint256 numPoints = currentHighway.highwaypoints.length; // Get the number of points on the highway
            uint128[] memory displacements = new uint128[](numPoints); // Create an array for displacements

            for (uint256 j = 0; j < numPoints; j++) {
                string memory pointName = currentHighway.highwaypoints[j]; // Get the point name
                displacements[j] = currentHighway.highwayspoints[pointName]; // Add the displacement for the point
            }

            highwayPointsDisplacements[i] = displacements; // Add the displacements array to the 2D array
        }
        return (highwayNumbers, tollRates, highwayPointsDisplacements); // Return the highway details
    }

    // Function to get toll details based on indices
    function getTollDetails(uint256[] memory indices) external view returns (Toll[] memory) {
        uint256 length = indices.length; // Get the length of the indices array
        Toll[] memory tollDetails = new Toll[](length); // Create an array for toll details

        for (uint256 i = 0; i < length; i++) {
            require(indices[i] < tolldetail.length, "Index out of bounds"); // Ensure the index is within bounds
            tollDetails[i] = tolldetail[indices[i]]; // Get the toll details for the index
        }

        return tollDetails; // Return the toll details
    }

    // Function to get challan details based on indices
    function getChallanDetails(uint256[] memory indices) external view returns (Challan[] memory) {
        uint256 length = indices.length; // Get the length of the indices array
        Challan[] memory challanDetails = new Challan[](length); // Create an array for challan details

        for (uint256 i = 0; i < length; i++) {
            require(indices[i] < challan.length, "Index out of bounds"); // Ensure the index is within bounds
            challanDetails[i] = challan[indices[i]]; // Get the challan details for the index
        }

        return challanDetails; // Return the challan details
    }

    // Function to get vehicle details based on vehicle IDs
    function getVehicleDetails(string[] memory vehicleIds) external view returns (Vehicle[] memory) {
        uint256 length = vehicleIds.length; // Get the length of the vehicle IDs array
        Vehicle[] memory vehicleDetails = new Vehicle[](length); // Create an array for vehicle details

        for (uint256 i = 0; i < length; i++) {
            require(vehicledetails[vehicleIds[i]].ownerAddress != address(0), "Vehicle does not exist"); // Ensure the vehicle exists
            vehicleDetails[i] = vehicledetails[vehicleIds[i]]; // Get the vehicle details for the ID
        }

        return vehicleDetails; // Return the vehicle details
    }
}
```

users.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;

contract UserManager {
    // User struct to store user details
    struct User {
        string name;
        string email;
        string password;
        bool isActive;
        UserRole role;
        string mobileNumber;
        string homeAddress;
        uint[] tollids;
        string[] vehicleids;
        uint[] challanids;
    }

    // Enum to define user roles
    enum UserRole { Admin, Employee, Customer }

    // Mappings and arrays to store user information
    mapping(address => User) public users;
    address[] public adminAddresses;
    address[] public employeeAddresses;
    address[] public customerAddresses;
    address[] public activeAddresses;
    address[] public blockedAddresses;

    // Use the ArrayUtils library
    using ArrayUtils for address[];

    // Modifiers for access control
    modifier onlyAdminOrEmployee() {
        require(users[msg.sender].role == UserRole.Admin || users[msg.sender].role ==
UserRole.Employee, "Only admin or employee can call this function");
        _;
    }

    modifier onlyUser() {
        require(users[msg.sender].isActive, "User is blocked");
        _;
    }

    modifier onlyAdmin() {
        require(users[msg.sender].role == UserRole.Admin, "Only admin can call this function");
        _;
    }

    // Events for logging actions
    event UserSignUp(address indexed userAddress, string name, string email, string password,
bool isActive, UserRole role);
    event UserLogin(address indexed userAddress, string name, UserRole role);
    event UserRoleChanged(address indexed userAddress, UserRole oldRole, UserRole
newRole);
    event UserBlocked(address indexed userAddress);
    event UserUnblocked(address indexed userAddress);
    event UserDetailsViewed(string viewerName, string userName, string userEmail, bool
isActive, UserRole role);
    event AllUserCounts(uint256 adminCount, uint256 employeeCount, uint256
customerCount, uint256 activeUserCount, uint256 blockedUserCount);
    event TollIDAdded(address indexed userAddress, string tollName, string tollID);
    event VehicleAdded(address indexed userAddress, string registrationNumber, string make,
string model, string color);
    event ChallanAdded(address indexed userAddress, string description, uint256 amount);

    // Constructor to initialize the contract with the first admin
    constructor() {
        address userAddress = msg.sender;
        users[userAddress] = User({
            name: "shailesh",
            email: "shai@gmail.com",
            password: "1234",
            isActive: true,
            role: UserRole.Admin,
            mobileNumber: "",
            homeAddress: "",
            tollids: new uint ,
            vehicleids: new string ,
            challanids: new uint
        });
        adminAddresses.push(userAddress);
        activeAddresses.push(userAddress);
        emit UserSignUp(userAddress, "shailesh", "shai@gmail.com", "1234", true,
UserRole.Admin);
    }

    // Function for users to sign up
    function signUp(string memory _name, string memory _email, string memory _password)
external {
        require(bytes(users[msg.sender].name).length == 0, "User is already signed up");
        users[msg.sender] = User({
            name: _name,
            email: _email,
            password: _password,
            isActive: true,
            role: UserRole.Customer,
            mobileNumber: "",
            homeAddress: "",
            tollids: new uint ,
            vehicleids: new string ,
            challanids: new uint
        });
        customerAddresses.push(msg.sender);
        activeAddresses.push(msg.sender);
        emit UserSignUp(msg.sender, _name, _email, _password, true, UserRole.Customer);
    }

    // Function for users to log in
    function login(string memory _password, UserRole _role) external returns (bool) {
        require(bytes(users[msg.sender].name).length > 0, "User is not registered yet.");
        require(users[msg.sender].isActive, "User is blocked");
        if (keccak256(bytes(users[msg.sender].password)) != keccak256(bytes(_password))) {
            return false; // Invalid password
        }
        if (users[msg.sender].role != _role) {
            return false; // Invalid role
        }
        emit UserLogin(msg.sender, users[msg.sender].name, _role);
        return true; // All checks passed, login successful
    }

    // Function to change user role
    function changeUserRole(address _userAddress, UserRole _newRole) external
onlyAdminOrEmployee {
        UserRole oldRole = users[_userAddress].role;
        users[_userAddress].role = _newRole;
        if (oldRole == UserRole.Admin) {
            adminAddresses.removeAddress(_userAddress);
        } else if (oldRole == UserRole.Employee) {
            employeeAddresses.removeAddress(_userAddress);
        } else if (oldRole == UserRole.Customer) {
            customerAddresses.removeAddress(_userAddress);
        }
        if (_newRole == UserRole.Admin) {
            adminAddresses.push(_userAddress);
        } else if (_newRole == UserRole.Employee) {
            employeeAddresses.push(_userAddress);
        } else if (_newRole == UserRole.Customer) {
            customerAddresses.push(_userAddress);
        }
        emit UserRoleChanged(_userAddress, oldRole, _newRole);
    }

    // Function to block a user
    function blockUser(address _userAddress) external onlyAdminOrEmployee {
        require(users[_userAddress].isActive, "User is already blocked");
        users[_userAddress].isActive = false;
        activeAddresses.removeAddress(_userAddress);
        blockedAddresses.push(_userAddress);
        emit UserBlocked(_userAddress);
    }

    // Function to unblock a user
    function unblockUser(address _userAddress) external onlyAdminOrEmployee {
        require(!users[_userAddress].isActive, "User is already active");
        users[_userAddress].isActive = true;
        blockedAddresses.removeAddress(_userAddress);
        activeAddresses.push(_userAddress);
        emit UserUnblocked(_userAddress);
    }

    // Function for users to save their details
    function saveUserDetails(string memory _mobileNumber, string memory _homeAddress)
external onlyUser {
        users[msg.sender].mobileNumber = _mobileNumber;
        users[msg.sender].homeAddress = _homeAddress;
    }

    // Function to get user details
    function getUserDetails() external view onlyUser returns (string memory, string memory, bool, UserRole, uint, uint, uint) {
        User storage user = users[msg.sender];
        return (user.name, user.email, user.isActive, user.role, user.tollids.length, user.vehicleids.length, user.challanids.length);
    }

    // Function to view details of a specific user (only accessible by admin or employee)
    function viewUserDetails(address _userAddress) external view onlyAdminOrEmployee returns (string memory, string memory, bool, UserRole, uint, uint, uint) {
        User storage user = users[_userAddress];
        return (user.name, user.email, user.isActive, user.role, user.tollids.length, user.vehicleids.length, user.challanids.length);
    }

    // Function to get counts of different user categories
    function usercount() external view onlyAdmin returns (uint, uint, uint, uint, uint) {
        return (adminAddresses.length, employeeAddresses.length, customerAddresses.length, activeAddresses.length, blockedAddresses.length);
    }

    // Function to get counts of tolls, vehicles, and challans for the logged-in user
    function personalcount() public view returns (uint, uint, uint) {
        User storage user = users[msg.sender];
        return (user.tollids.length, user.vehicleids.length, user.challanids.length);
    }
    // Function to get toll details for the logged-in user
    function tolldetails() public view returns (uint[] memory) {
        return users[msg.sender].tollids;
    }

    // Function to get challan details for the logged-in user
    function challandetails() public view returns (uint[] memory) {
        return users[msg.sender].challanids;
    }

    // Function to get vehicle details for the logged-in user
    function vehiclesdetails() public view returns (string[] memory) {
        return users[msg.sender].vehicleids;
    }
}

// Library for array utilities
library ArrayUtils {
    // Function to remove an address from an array
    function removeAddress(address[] storage array, address element) internal {
        for (uint256 i = 0; i < array.length; i++) {
            if (array[i] == element) {
                array[i] = array[array.length - 1];
                array.pop();
                break;
            }
        }
    }
}
```

userdetails.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;

import "./users.sol";

contract UserDetail is UserManager {
  // Function to get all customer details
  function getAllCustomerDetails() external view onlyAdminOrEmployee returns (User[]
memory) {
      uint256 customerCount = customerAddresses.length;
      User[] memory allCustomers = new User[](customerCount);
      uint256 index = 0;
      for (uint256 i = 0; i < customerCount; i++) {
        address userAddress = customerAddresses[i];
        if (users[userAddress].isActive && users[userAddress].role == UserRole.Customer) {
          allCustomers[index] = users[userAddress];
          index++;
        }
      }
      return allCustomers;
  }

  // Function to get all admin details
  function getAllAdminDetails() external view onlyAdmin returns (User[] memory) {
      uint256 adminCount = adminAddresses.length;
      User[] memory allAdmins = new User[](adminCount);
      uint256 index = 0;
      for (uint256 i = 0; i < adminCount; i++) {
        address userAddress = adminAddresses[i];
        if (users[userAddress].isActive && users[userAddress].role == UserRole.Admin) {
          allAdmins[index] = users[userAddress];
          index++;
        }
      }
      return allAdmins;
  }

  // Function to get all employee details
  function getAllEmployeeDetails() external view onlyAdmin returns (User[] memory) {
      uint256 employeeCount = employeeAddresses.length;
      User[] memory allEmployees = new User[](employeeCount);
      uint256 index = 0;
      for (uint256 i = 0; i < employeeCount; i++) {
        address userAddress = employeeAddresses[i];
        if (users[userAddress].isActive && users[userAddress].role == UserRole.Employee) {
          allEmployees[index] = users[userAddress];
          index++;
        }
      }
      return allEmployees;
  }

  // Function to get all active user details
  function getAllActive() external view onlyAdmin returns (User[] memory) {
      uint256 activeCount = activeAddresses.length;
      User[] memory allActive = new User[](activeCount);
      uint256 index = 0;
      for (uint256 i = 0; i < activeCount; i++) {
        address userAddress = activeAddresses[i];
        if (users[userAddress].isActive) {
            allActive[index] = users[userAddress];
            index++;
          }
        }
        return allActive;
    }

    // Function to get all blocked user details
    function allBlockedDetails() external view onlyAdmin returns (User[] memory) {
        uint256 blockedCount = blockedAddresses.length;
        User[] memory allBlocked = new User[](blockedCount);
        uint256 index = 0;
        for (uint256 i = 0; i < blockedCount; i++) {
            address userAddress = blockedAddresses[i];
            if (!users[userAddress].isActive) {
                allBlocked[index] = users[userAddress];
                index++;
            }
        }
        return allBlocked;
    }
}
```

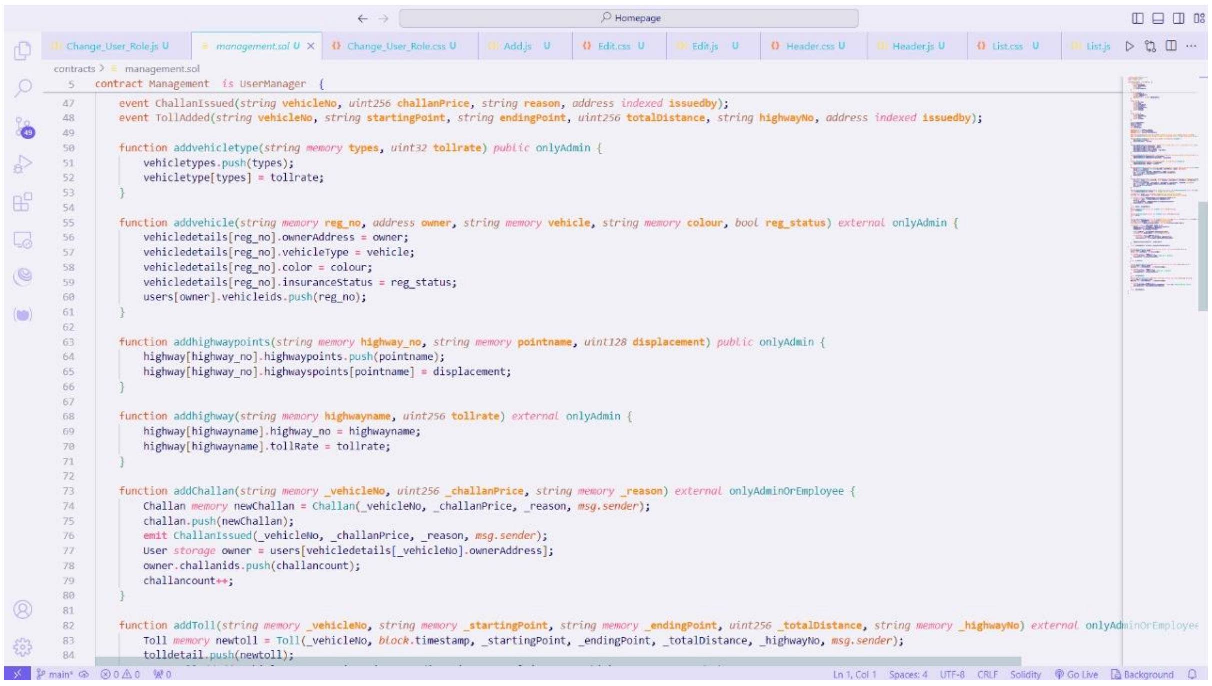
Web3helpers.js:

```
import Web3 from "web3"; // Import the Web3 library for interacting with the Ethereum blockchain

import Auth from "./backend/build/contracts/Auth.json"; // Import the compiled contract's ABI and network information
```

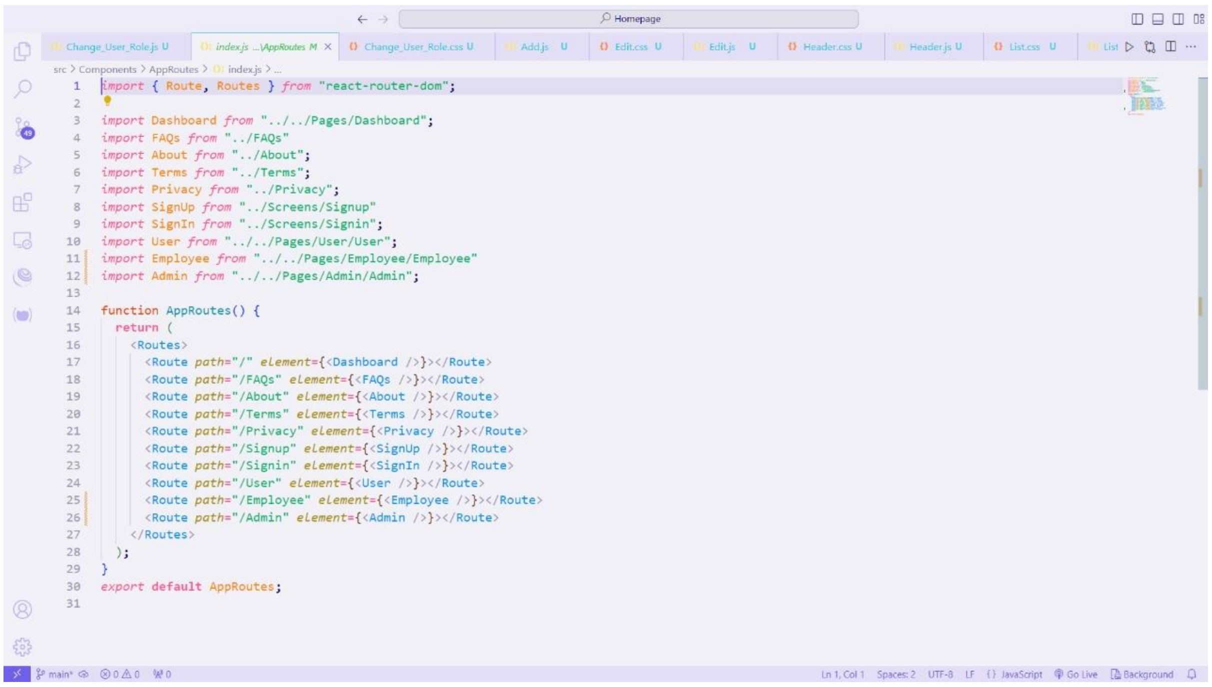
```
// Function to load Web3 instance and connect to the Ethereum blockchain
export const loadWeb3 = async () => {
  // Check if the browser has an injected Ethereum provider (e.g., MetaMask)
  if (window.ethereum) {
    // Create a new Web3 instance with the injected Ethereum provider
    window.web3 = new Web3(window.ethereum);
    // Request account access if needed
    await window.ethereum.enable();
  } else if (window.web3) {
    // If already injected Web3 provider (legacy dapp browsers)
    window.web3 = new Web3(window.web3.currentProvider);
  } else {
    // If no Ethereum browser is detected, alert the user
    window.alert(
      "Non-Ethereum browser detected. You should consider trying MetaMask!"
    );
  }
};

// Function to load blockchain data, including the contract instance and user account
export const loadBlockchainData = async () => {
  const web3 = window.web3; // Get the web3 instance from the window object

  // Load accounts from the web3 provider
  const accounts = await web3.eth.getAccounts();

  // Get the network ID of the connected Ethereum network
  const networkId = await web3.eth.net.getId();
```

```
    // Check if the network ID is available in the contract's network data
    if (networkId) {
        // Create a new contract instance with the contract's ABI and address for the current network
        const auth = new web3.eth.Contract(
            Auth.abi,
            Auth.networks[networkId].address
        );
        // Return the contract instance and the first account
        return { auth, accounts: accounts[0] };
    }
};
```

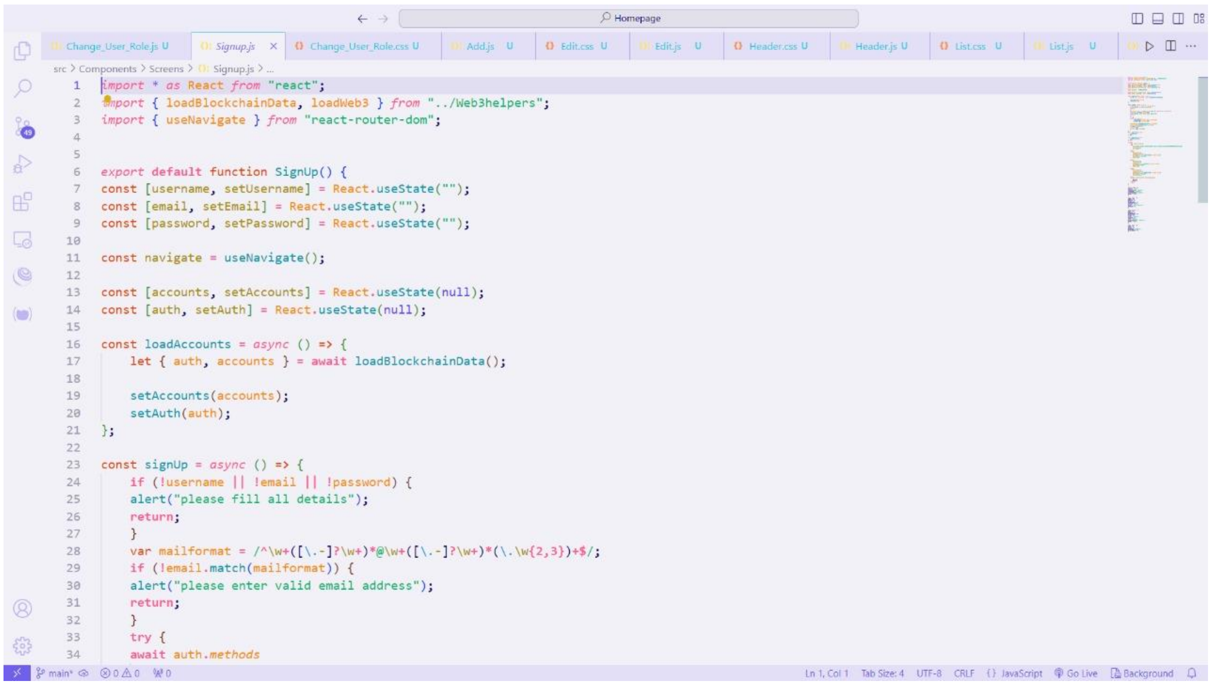
RUNNING CODE SNIPPETS AND OUTPUTS:



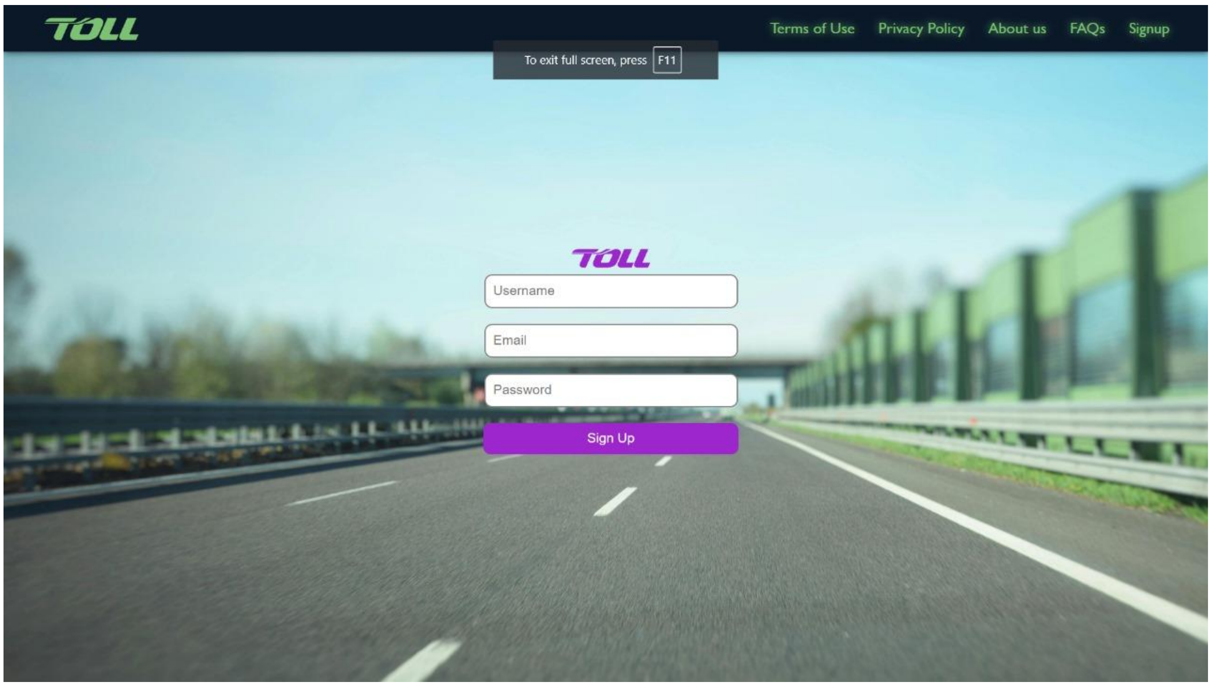
```
contract Management  is UserManager  {
    event ChallanIssued(string vehicleNo, uint256 challanPrice, string reason, address indexed issuedby);
    event TollAdded(string vehicleNo, string startingPoint, string endingPoint, uint256 totalDistance, string highwayNo, address indexed issuedby);

    function addvehicletype(string memory types, uint32 tollrate) public onlyAdmin {
        vehicletypes.push(types);
        vehicletype[types] = tollrate;
    }

    function addvehicle(string memory reg_no, address owner, string memory vehicle, string memory colour, bool reg_status) external onlyAdmin {
        vehicledetails[reg_no].ownerAddress = owner;
        vehicledetails[reg_no].vehicleType = vehicle;
        vehicledetails[reg_no].color = colour;
        vehicledetails[reg_no].insuranceStatus = reg_status;
        users[owner].vehicleids.push(reg_no);
    }

    function addhighwaypoints(string memory highway_no, string memory pointname, uint128 displacement) public onlyAdmin {
        highway[highway_no].highwaypoints.push(pointname);
        highway[highway_no].highwayspoints[pointname] = displacement;
    }

    function addhighway(string memory highwayname, uint256 tollrate) external onlyAdmin {
        highway[highwayname].highway_no = highwayname;
        highway[highwayname].tollRate = tollrate;
    }

    function addChallan(string memory _vehicleNo, uint256 _challanPrice, string memory _reason) external onlyAdminOrEmployee {
        Challan memory newChallan = Challan(_vehicleNo, _challanPrice, _reason, msg.sender);
        challan.push(newChallan);
        emit ChallanIssued(_vehicleNo, _challanPrice, _reason, msg.sender);
        User storage owner = users[vehicledetails[_vehicleNo].ownerAddress];
        owner.challanids.push(challancount);
        challancount++;
    }

    function addToll(string memory _vehicleNo, string memory _startingPoint, string memory _endingPoint, uint256 _totalDistance, string memory _highwayNo) external onlyAdminOrEmployee
        Toll memory newtoll = Toll(_vehicleNo, block.timestamp, _startingPoint, _endingPoint, _totalDistance, _highwayNo, msg.sender);
        tolldetail.push(newtoll);
```

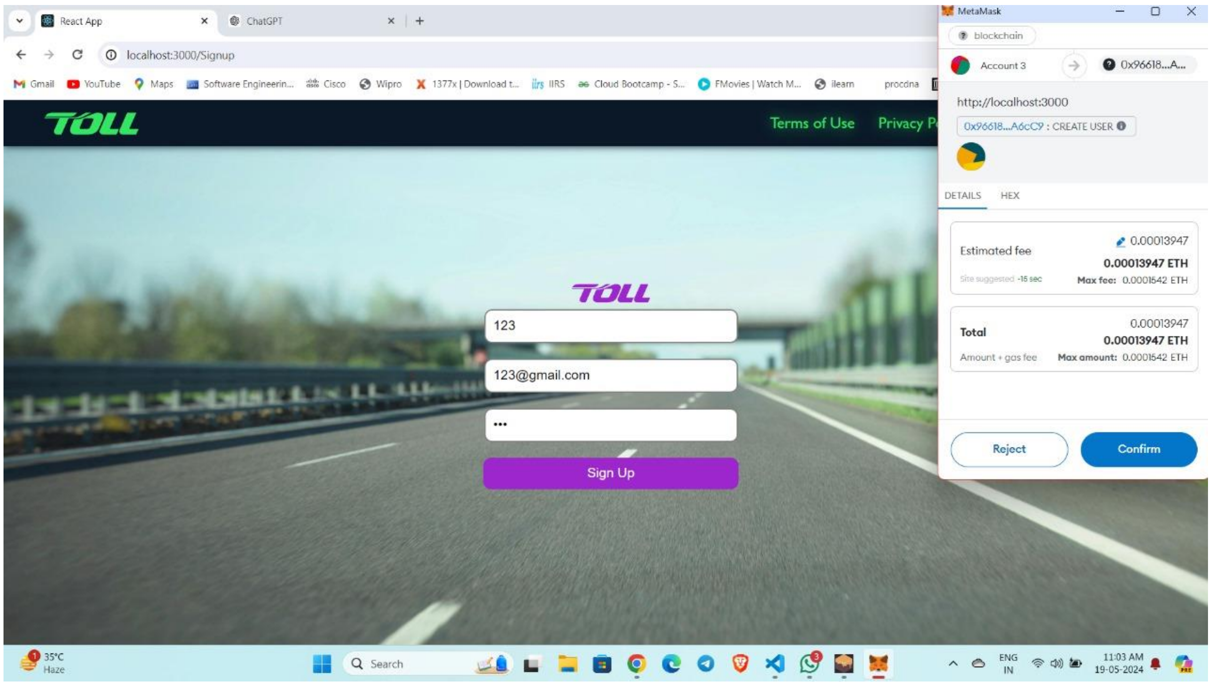


```
import { Route, Routes } from "react-router-dom";

import Dashboard from "../../Pages/Dashboard";
import FAQs from "../FAQs"
import About from "../About";
import Terms from "../Terms";
import Privacy from "../Privacy";
import SignUp from "../Screens/Signup"
import SignIn from "../Screens/Signin";
import User from "../../Pages/User/User";
import Employee from "../../Pages/Employee/Employee"
import Admin from "../../Pages/Admin/Admin";

function AppRoutes() {
  return (
    <Routes>
      <Route path="/" element={<Dashboard />}></Route>
      <Route path="/FAQs" element={<FAQs />}></Route>
      <Route path="/About" element={<About />}></Route>
      <Route path="/Terms" element={<Terms />}></Route>
      <Route path="/Privacy" element={<Privacy />}></Route>
      <Route path="/Signup" element={<SignUp />}></Route>
      <Route path="/Signin" element={<SignIn />}></Route>
      <Route path="/User" element={<User />}></Route>
      <Route path="/Employee" element={<Employee />}></Route>
      <Route path="/Admin" element={<Admin />}></Route>
    </Routes>
  );
}
export default AppRoutes;

```



```
import * as React from "react";
import { loadBlockchainData, loadWeb3 } from "../Web3helpers";
import { useNavigate } from "react-router-dom";


export default function SignUp() {
const [username, setUsername] = React.useState("");
const [email, setEmail] = React.useState("");
const [password, setPassword] = React.useState("");

const navigate = useNavigate();

const [accounts, setAccounts] = React.useState(null);
const [auth, setAuth] = React.useState(null);

const loadAccounts = async () => {
    let { auth, accounts } = await loadBlockchainData();

    setAccounts(accounts);
    setAuth(auth);
};

const signUp = async () => {
    if (!username || !email || !password) {
    alert("please fill all details");
    return;
    }
    var mailformat = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    if (!email.match(mailformat)) {
    alert("please enter valid email address");
    return;
    }
    try {
    await auth.methods
```





123

123@gmail.com

•••

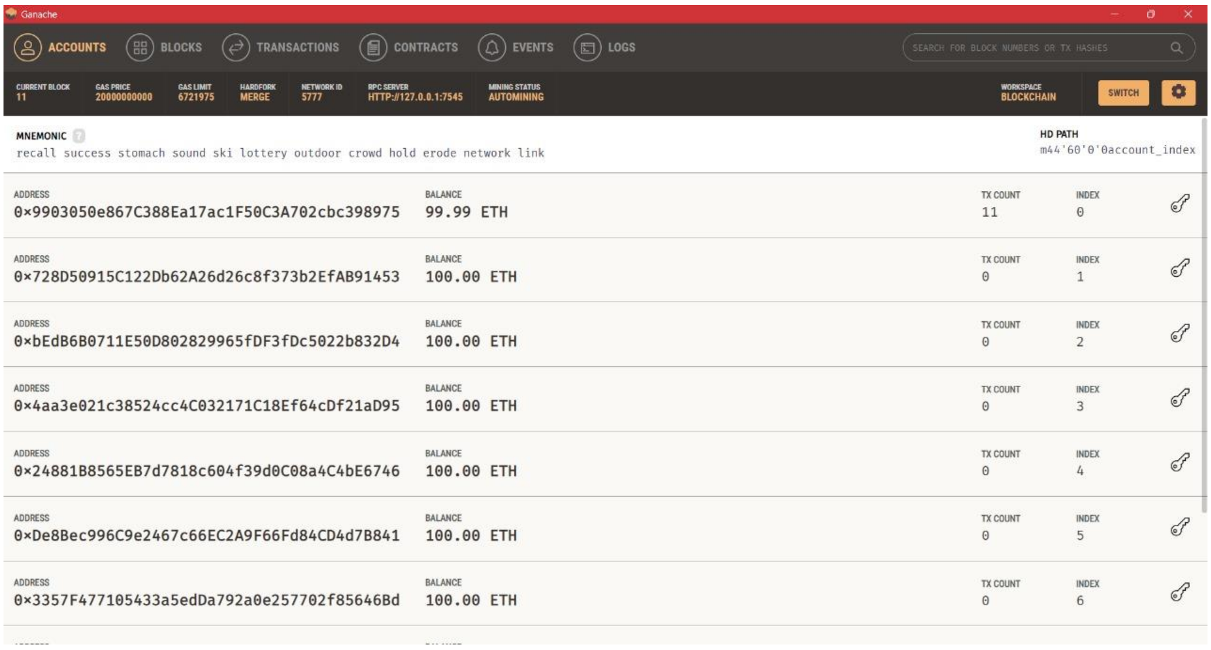
Account 3 → 0x96618...A...

http://localhost:3000

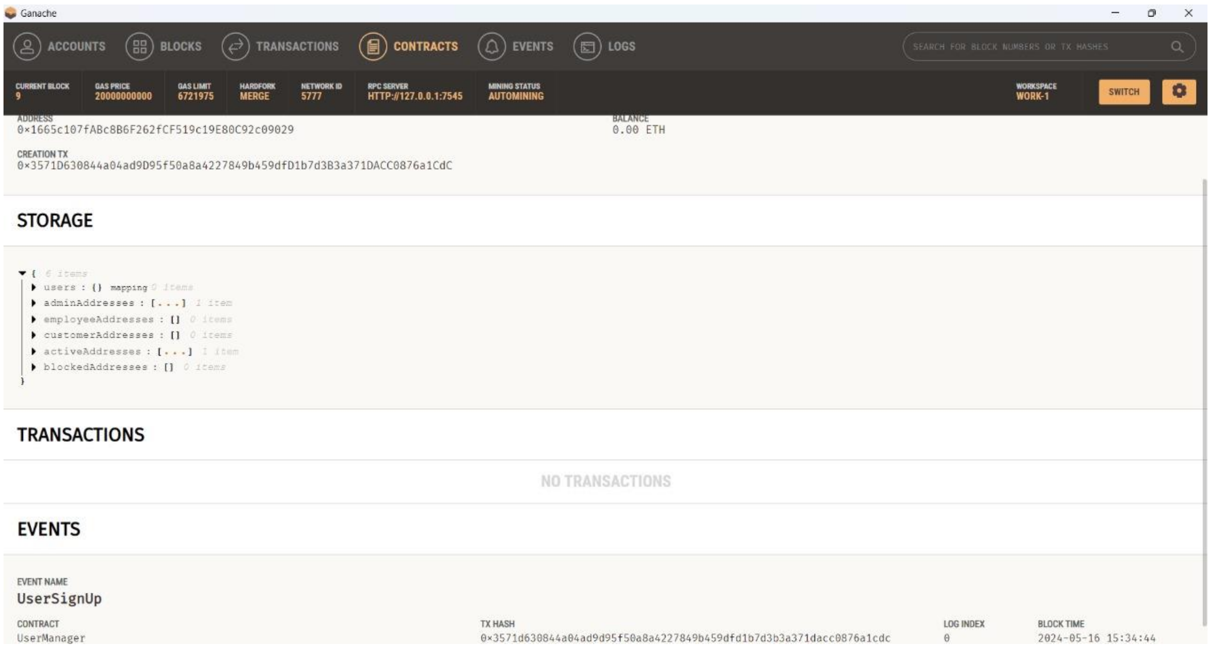
0x96618...A6cC9 : CREATE USER

Estimated fee 0.00013947
0.00013947 ETH
Site suggested ~15 sec Max fee: 0.0001542 ETH

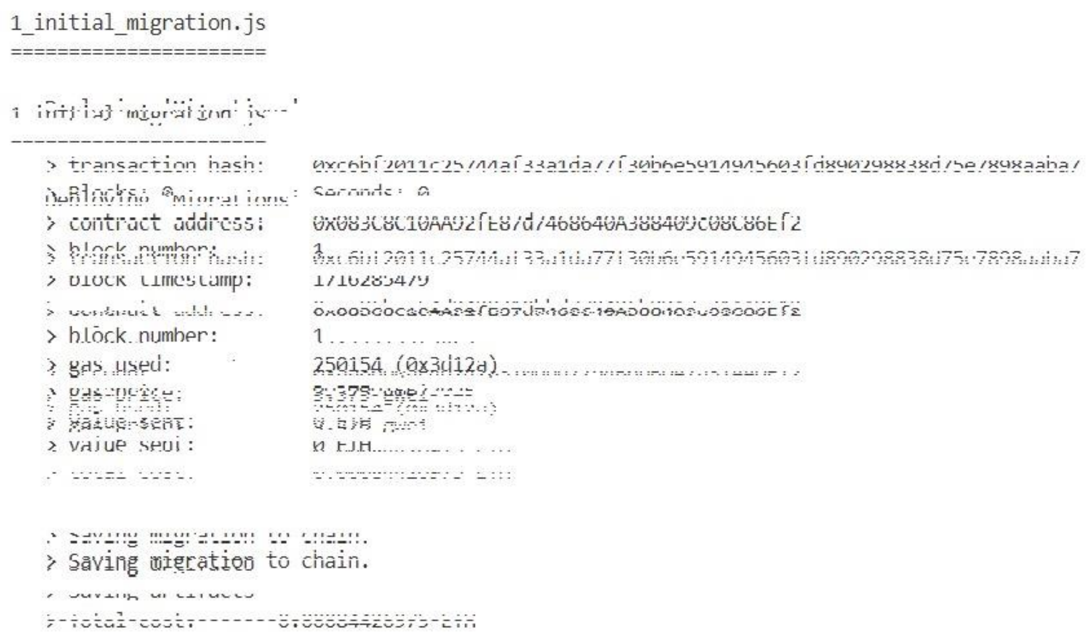
Total 0.00013947
0.00013947 ETH
Amount + gas fee Max amount: 0.0001542 ETH



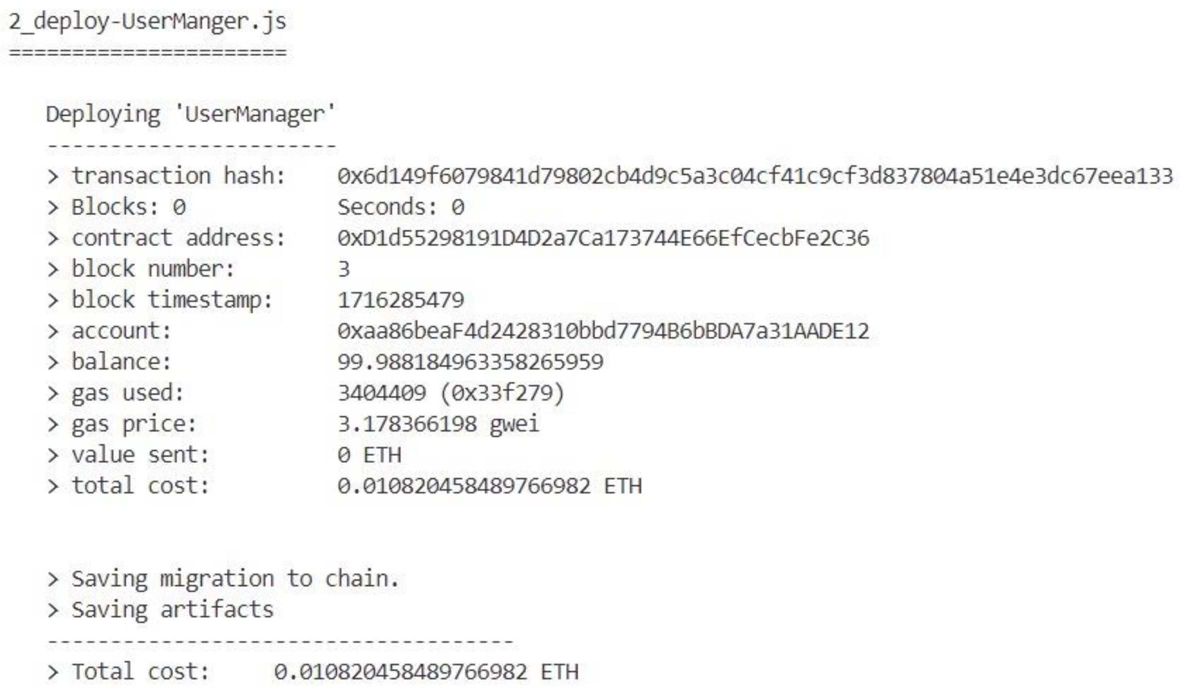
```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
11             20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      BLOCKCHAIN

MNEMONIC                                                                          HD PATH
recall success stomach sound ski lottery outdoor crowd hold erode network link    m44'60'0'0account_index

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×9903050e867C388Ea17ac1F50C3A702cbc398975  99.99 ETH    11        0

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×728D50915C122Db62A26d26c8f373b2EfAB91453  100.00 ETH   0         1

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×bEdB6B0711E50D802829965fDF3fDc5022b832D4  100.00 ETH   0         2

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×4aa3e021c38524cc4C032171C18Ef64cDf21aD95  100.00 ETH   0         3

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×24881B8565EB7d7818c604f39d0C08a4C4bE6746  100.00 ETH   0         4

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×De8Bec996C9e2467c66EC2A9F66Fd84CD4d7B841  100.00 ETH   0         5

ADDRESS                                     BALANCE      TX COUNT  INDEX
0×3357F477105433a5edDa792a0e257702f85646Bd  100.00 ETH   0         6
```



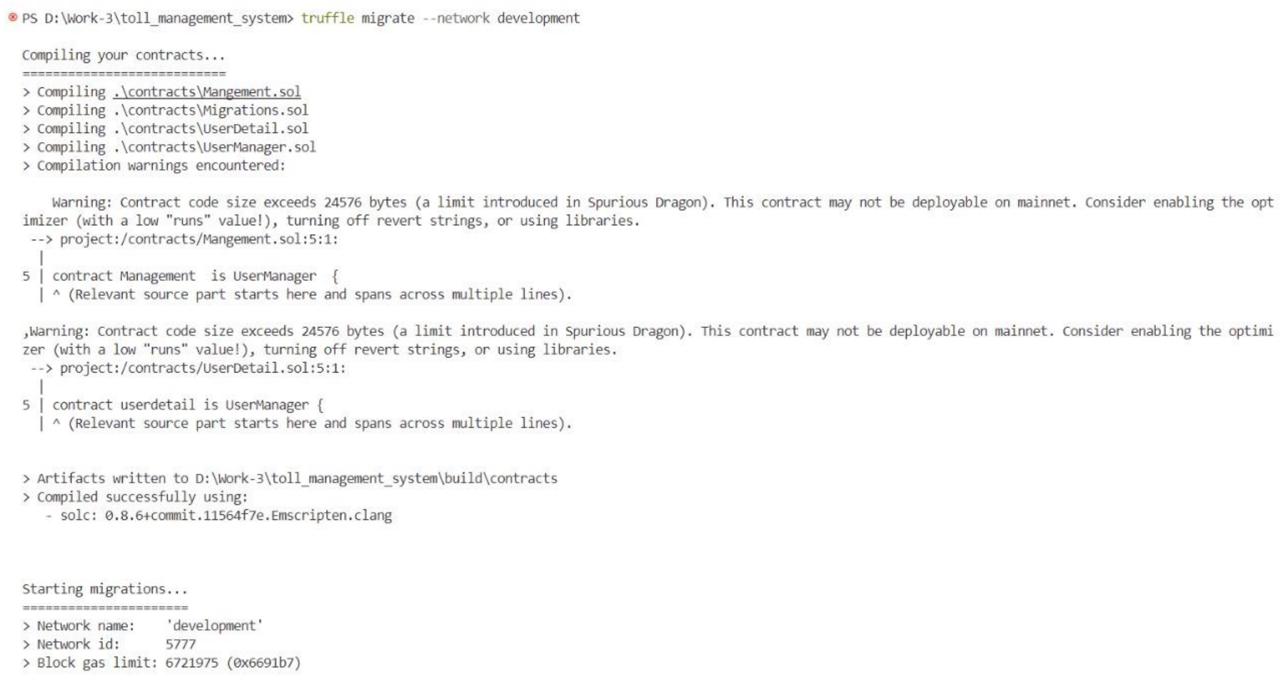
```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
9              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      WORK-1

ADDRESS                                                              BALANCE
0×1665c107fABc8B6F262fCF519c19E80C92c09029                           0.00 ETH

CREATION TX
0×3571D630844a04ad9D95f50a8a4227849b459dfD1b7d3B3a371DACC0876a1CdC

STORAGE

▼ { 6 items
  ▶ users : {} mapping 0 items
  ▶ adminAddresses : [...] 1 item
  ▶ employeeAddresses : [] 0 items
  ▶ customerAddresses : [] 0 items
  ▶ activeAddresses : [...] 1 item
  ▶ blockedAddresses : [] 0 items
}

TRANSACTIONS

NO TRANSACTIONS

EVENTS

EVENT NAME
UserSignUp

CONTRACT      TX HASH                                                             LOG INDEX  BLOCK TIME
UserManager   0×3571d630844a04ad9d95f50a8a4227849b459dfd1b7d3b3a371dacc0876a1cdc  0          2024-05-16 15:34:44
```



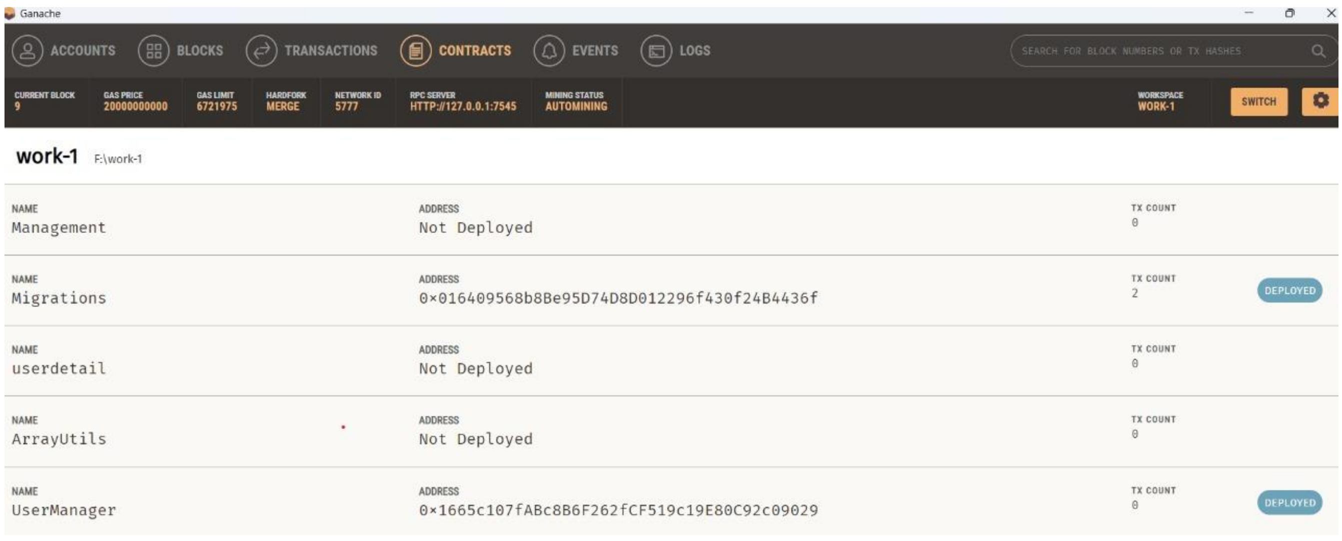
```
1_initial_migration.js
======================

[illegible]
----------------------
   > transaction hash:    0xc6bf2011c25744af33a1da77f30b6e5914945603fd8902988 38d75e7898aaba7
   [illegible]            Seconds: 0
   > contract address:    0x083C8C10AA02fE87d7468640A388409C08C86Ef2
   [illegible]            [illegible]
   > block timestamp:     1716285479
   [illegible]            [illegible]
   > block number:        1
   > gas used:            250154 (0x3d12a)
   [illegible]            [illegible]
   > value sent:          0 ETH
   [illegible]            [illegible]

   [illegible]
   > Saving migration to chain.
   [illegible]
   > Total cost:          [illegible] ETH
```



```
2_deploy-UserManger.js
======================

   Deploying 'UserManager'
   -----------------------
   > transaction hash:    0x6d149f6079841d79802cb4d9c5a3c04cf41c9cf3d837804a51e4e3dc67eea133
   > Blocks: 0            Seconds: 0
   > contract address:    0xD1d55298191D4D2a7Ca173744E66EfCecbFe2C36
   > block number:        3
   > block timestamp:     1716285479
   > account:             0xaa86beaF4d2428310bbd7794B6bBDA7a31AADE12
   > balance:             99.988184963358265959
   > gas used:            3404409 (0x33f279)
   > gas price:           3.178366198 gwei
   > value sent:          0 ETH
   > total cost:          0.010820458489766982 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.010820458489766982 ETH
```



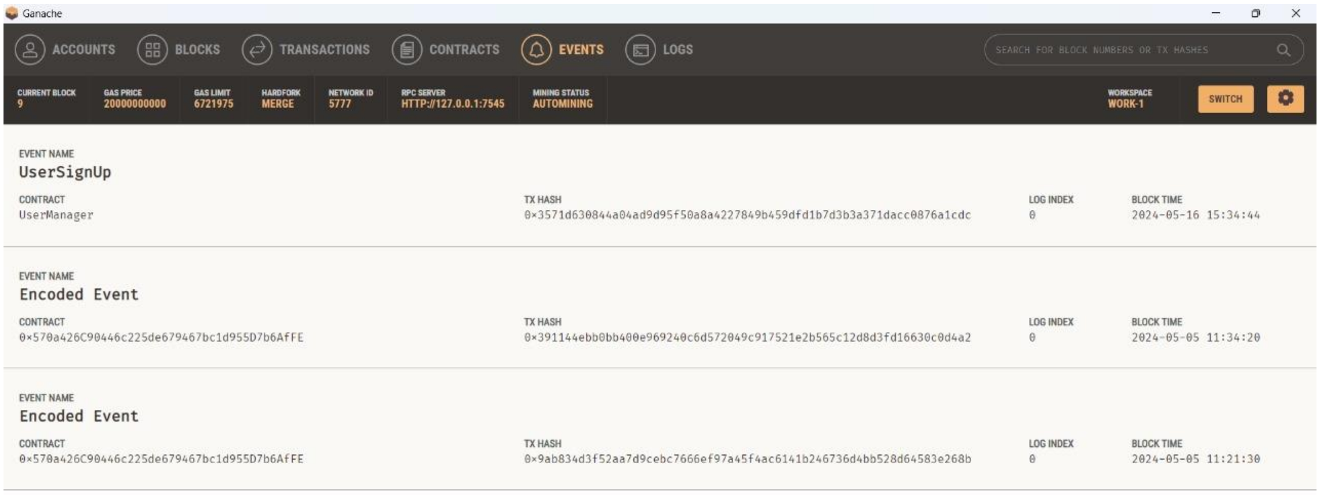
```
PS D:\Work-3\toll_management_system> truffle migrate --network development

Compiling your contracts...
===========================
> Compiling .\contracts\Mangement.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\UserDetail.sol
> Compiling .\contracts\UserManager.sol
> Compilation warnings encountered:

    Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the opt
imizer (with a low "runs" value!), turning off revert strings, or using libraries.
 --> project:/contracts/Mangement.sol:5:1:
  |
5 | contract Management  is UserManager  {
  | ^ (Relevant source part starts here and spans across multiple lines).

,Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimi
zer (with a low "runs" value!), turning off revert strings, or using libraries.
 --> project:/contracts/UserDetail.sol:5:1:
  |
5 | contract userdetail is UserManager {
  | ^ (Relevant source part starts here and spans across multiple lines).


> Artifacts written to D:\Work-3\toll_management_system\build\contracts
> Compiled successfully using:
   - solc: 0.8.6+commit.11564f7e.Emscripten.clang



Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)
```



CURRENT BLOCK 9 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE WORK-1

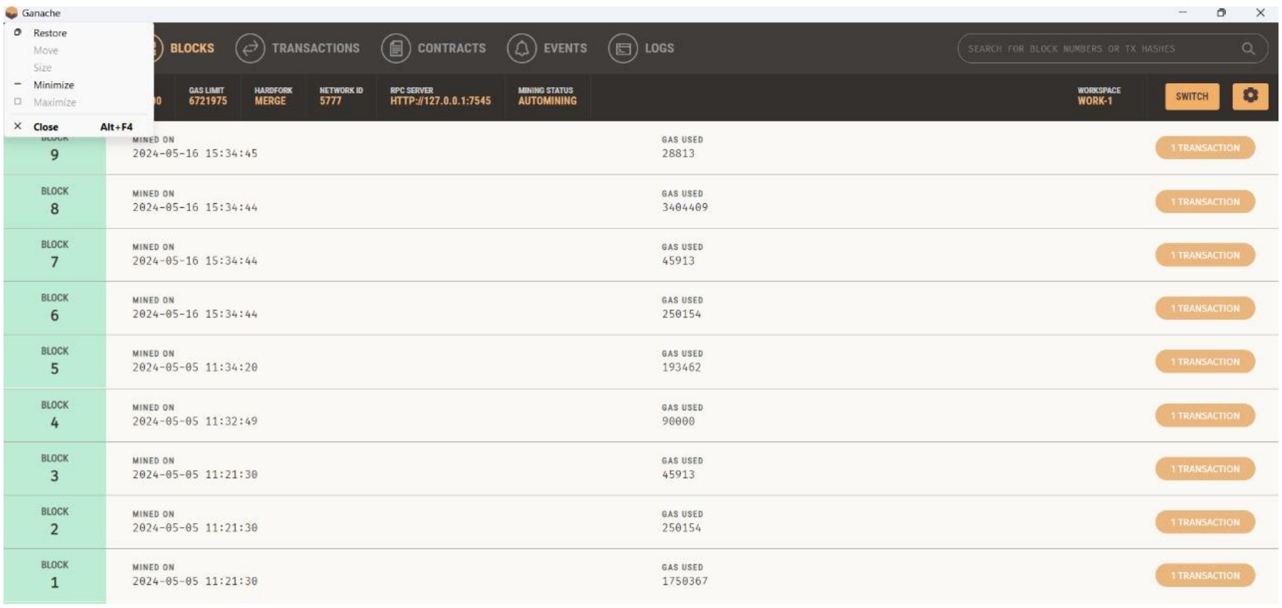
work-1 F:\work-1

NAME	ADDRESS	TX COUNT	
Management	Not Deployed	0	
Migrations	0x016409568b8Be95D74D8D012296f430f24B4436f	2	DEPLOYED
userdetail	Not Deployed	0	
ArrayUtils	Not Deployed	0	
UserManager	0x1665c107fABc8B6F262fCF519c19E80C92c09029	0	DEPLOYED



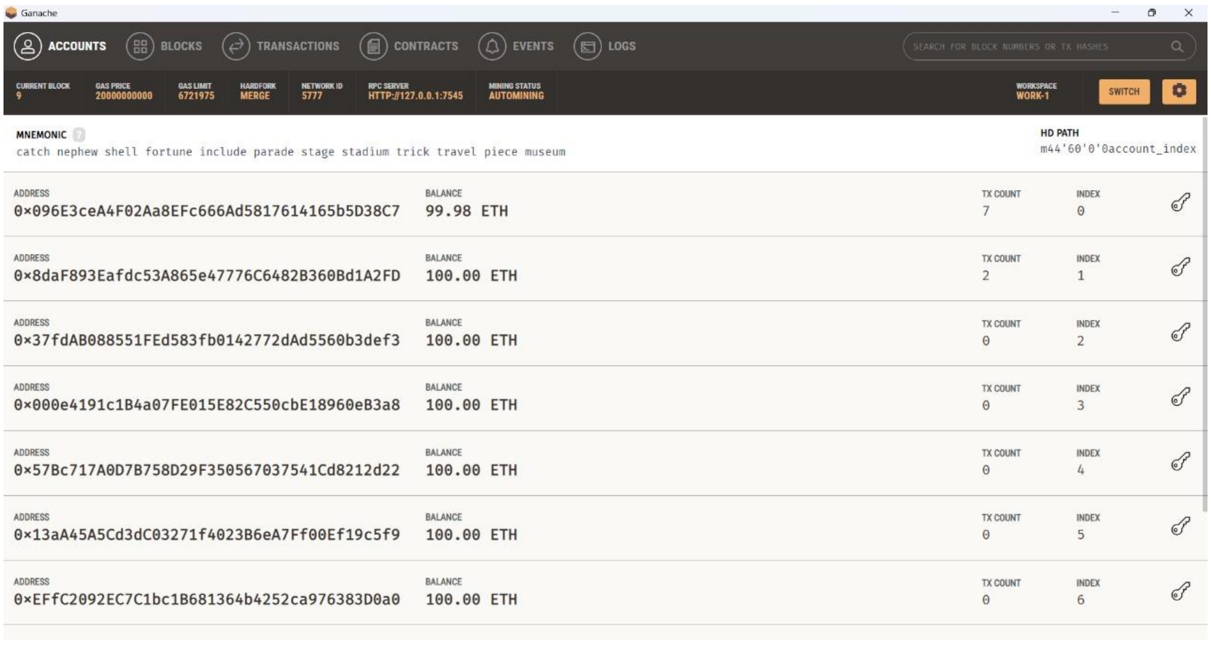
CURRENT BLOCK 9 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE WORK-1

EVENT NAME	CONTRACT	TX HASH	LOG INDEX	BLOCK TIME
UserSignUp	UserManager	0×3571d630844a04ad9d95f50a8a4227849b459dfd1b7d3b3a371dacc0876a1cdc	0	2024-05-16 15:34:44
Encoded Event	0×570a426C90446c225de679467bc1d955D7b6AfFE	0×391144ebb0bb400e969240c6d572049c917521e2b565c12d8d3fd16630c0d4a2	0	2024-05-05 11:34:20
Encoded Event	0×570a426C90446c225de679467bc1d955D7b6AfFE	0×9ab834d3f52aa7d9cebc7666ef97a45f4ac6141b246736d4bb528d64583e268b	0	2024-05-05 11:21:30

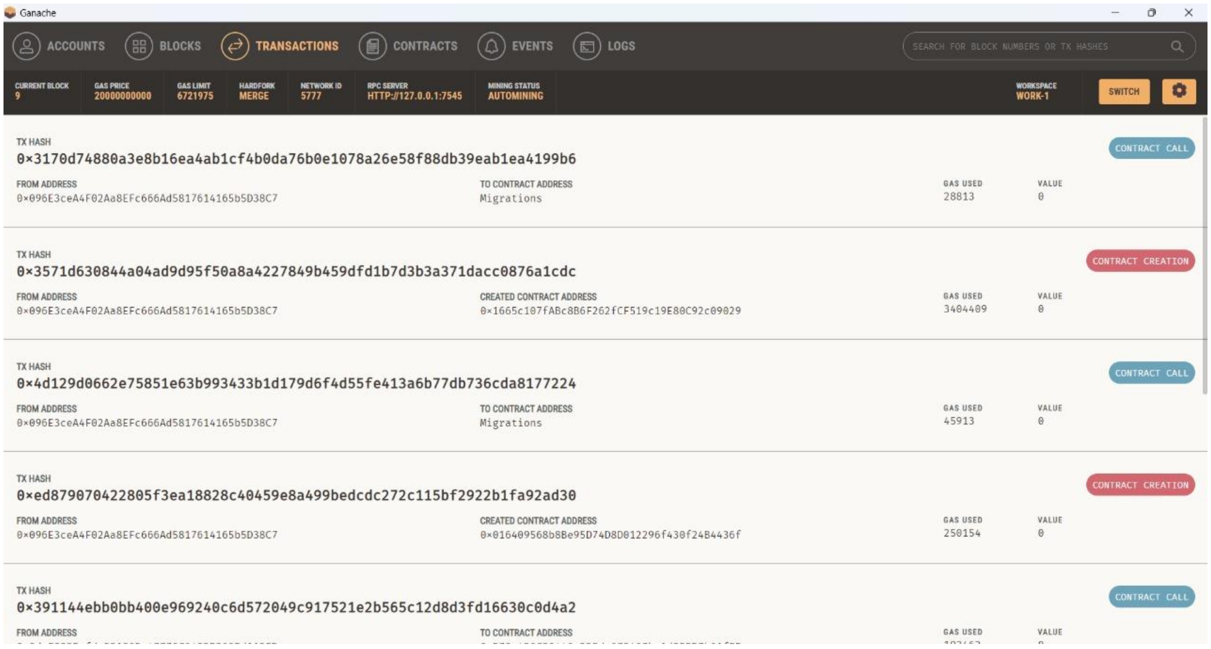


GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE WORK-1

BLOCK	MINED ON	GAS USED	
9	2024-05-16 15:34:45	28813	1 TRANSACTION
8	2024-05-16 15:34:44	3404409	1 TRANSACTION
7	2024-05-16 15:34:44	45913	1 TRANSACTION
6	2024-05-16 15:34:44	250154	1 TRANSACTION
5	2024-05-05 11:34:20	193462	1 TRANSACTION
4	2024-05-05 11:32:49	90000	1 TRANSACTION
3	2024-05-05 11:21:30	45913	1 TRANSACTION
2	2024-05-05 11:21:30	250154	1 TRANSACTION
1	2024-05-05 11:21:30	1750367	1 TRANSACTION



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
9              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      WORK-1

MNEMONIC                                                                           HD PATH
catch nephew shell fortune include parade stage stadium trick travel piece museum  m44'60'0'0account_index

ADDRESS                                     BALANCE     TX COUNT  INDEX
0×096E3ceA4F02Aa8EFc666Ad5817614165b5D38C7  99.98 ETH   7         0
0×8daF893Eafdc53A865e47776C6482B360Bd1A2FD  100.00 ETH  2         1
0×37fdAB088551FEd583fb0142772dAd5560b3def3  100.00 ETH  0         2
0×000e4191c1B4a07FE015E82C550cbE18960eB3a8  100.00 ETH  0         3
0×57Bc717A0D7B758D29F350567037541Cd8212d22  100.00 ETH  0         4
0×13aA45A5Cd3dC03271f4023B6eA7Ff00Ef19c5f9  100.00 ETH  0         5
0×EFfC2092EC7C1bc1B681364b4252ca976383D0a0  100.00 ETH  0         6
```



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
9              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      WORK-1

TX HASH                                                                                                CONTRACT CALL
0×3170d74880a3e8b16ea4ab1cf4b0da76b0e1078a26e58f88db39eab1ea4199b6
FROM ADDRESS                                TO CONTRACT ADDRESS                          GAS USED  VALUE
0×096E3ceA4F02Aa8EFc666Ad5817614165b5D38C7  Migrations                                   28813     0

TX HASH                                                                                                CONTRACT CREATION
0×3571d630844a04ad9d95f50a8a4227849b459dfd1b7d3b3a371dacc0876a1cdc
FROM ADDRESS                                CREATED CONTRACT ADDRESS                     GAS USED  VALUE
0×096E3ceA4F02Aa8EFc666Ad5817614165b5D38C7  0×1665c107fABc8B6F262fCF519c19E80C92c09029   3404409   0

TX HASH                                                                                                CONTRACT CALL
0×4d129d0662e75851e63b993433b1d179d6f4d55fe413a6b77db736cda8177224
FROM ADDRESS                                TO CONTRACT ADDRESS                          GAS USED  VALUE
0×096E3ceA4F02Aa8EFc666Ad5817614165b5D38C7  Migrations                                   45913     0

TX HASH                                                                                                CONTRACT CREATION
0×ed879070422805f3ea18828c40459e8a499bedcdc272c115bf2922b1fa92ad30
FROM ADDRESS                                CREATED CONTRACT ADDRESS                     GAS USED  VALUE
0×096E3ceA4F02Aa8EFc666Ad5817614165b5D38C7  0×016409568b8Be95D74D8D012296f430f24B4436f   250154    0

TX HASH                                                                                                CONTRACT CALL
0×391144ebb0bb400e969240c6d572049c917521e2b565c12d8d3fd16630c0d4a2
FROM ADDRESS                                TO CONTRACT ADDRESS                          GAS USED  VALUE
```