A Blockchain based Autonomous Decentralized Online Social Network

A Project Report Submitted in Partial Fulfilment of the Requirements for the Award of the Degree of

Bachelor of Technology
In
Computer Science and Engineering

Submitted by

BANDLAMUDI ROHIT JAYARAJU (Reg.No: 318506408040)
SAI SREEKAR AYYAGARI (Reg.No: 318506408041)
SAI SUDHEER MEESALA (Reg.No: 318506408042)
SAI SWETHA NAMBARI (Reg.No: 318506408043)

Under the Supervision of

Dr. S. JHANSI RANI
ASSISTANT PROFESSOR
DEPT. OF CS & SE
ANDHRA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ENGINEERING
ANDHRA UNIVERSITY COLLEGE OF ENGINEERING(A)
VISAKHAPATNAM - 530003

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ENGINEERING
ANDHRA UNIVERSITY COLLEGE OF ENGINEERING(A)
VISAKHAPATNAM - 530003
ANDHRA PRADESH, INDIA

BONAFIDE CERTIFICATE

This is to certify the project titled **"A Blockchain based Autonomous Decentralized Online Social Network"** is a certified record of work done by: **BANDLAMUDI ROHIT JAYARAJU (Reg.No:318506408040), SAI SREEKAR AYYAGARI (Reg.No: 318506408041), SAI SUDHEER MEESALA (Reg.No: 318506408042), SAI SWETHA NAMBARI (Reg.No: 318506408043**) students of 4/6 Computer Science & Networking, Department of Computer Science & Systems Engineering, Andhra University College of Engineering(A), Andhra University, Visakhapatnam during the period 2020-2021, in the partial fulfillment of the requirements for the award of Bachelor of Technology. This work has not been submitted to any other university for the award of Degree or Diploma.

Dr. S. Jhansi Rani
ASSITANT PROFESSOR
Dept. of CS & SE
ANDHRA UNIVERSITY

Prof. Dr. Kunjam Nageswara Rao
HEAD OF THE DEPARTMENT
DEPT. OF CS & SE
ANDHRA UNIVERSITY

DECLARATION

We hereby declare that this project entitled **"A Blockchain based Autonomous Decentralized Online Social Network"** is an original and authentic work done in the Department of Computer Science & Systems Engineering, Andhra University College of Engineering(A), Andhra University, Visakhapatnam, submitted in partial fulfillment of the requirements of the degree of Bachelor of Technology.

BANDLAMUDI ROHIT JAYARAJU (Reg.No: 318506408040)

SAI SREEKAR AYYAGARI (Reg.No: 318506408041)

SAI SUDHEER MEESALA (Reg.No: 318506408042)

SAI SWETHA NAMBARI (Reg.No: 318506408043)

ACKNOWLEDGEMENT

We have immense pleasure in expressing our earnest gratitude to our Project Supervisor **DR. S. JHANSI RANI**, Department of CS&SE, AUCE(A), Andhra University, for her inspiring and scholarly guidance. Despite of her preoccupation with several assignments, she has been kind enough to spare her valuable time and gave us the necessary counsel and guidance, that received at every stage of planning and constitution of this work and for helping us graciously throughout the execution of the work.

We express sincere gratitude to **Prof. DR. Kunjam Nageswara Rao**, Head of the Department, CS&SE, for having accorded us on taking up this project work and for helping us graciously throughout the execution of this work.

We express sincere thanks to **Prof. DR. P.Srinivasa Rao**, Principal, Andhra University College of Engineering(A), for his keen interest and for providing necessary facilities for this project study.

We extend our gratitude to Mrs.S.S.Nandini and the academic teaching staff and non-teaching staff for their affection and help throughout our study.

CONTENTS

TABLES

ABSTRACT

The decentralized twitter is a perfect illustration of the decentralized social media networks that operate independently rather than centralized systems owned by the companies. The decentralized Twitter encourages independence without central authority by giving the users more control over their tweets and other content. The web 3.0 with decentralization as its key feature lends itself to important technology named blockchain which takes an important role in developing the decentralized twitter. With the emergence of decentralized social network, the use of cryptocurrency has enhanced. Cryptocurrencies are digital currencies that use blockchain as a ledger. Blockchain technology is renowned for its decentralization, security, immutability, privacy and mainly its transparency. This application uses wallets supporting the cryptocurrency named Solana. The decentralized twitter is a fully functional twitter like application where anyone can use their wallets to connect and start publishing tweets. Some of the benefits of using decentralized twitter includes censorship resistance, ownership over personal data and decentralized platforms are more likely to be open development platforms.

1. INTRODUCTION

1.1 DECENTRALIZED SOCIAL MEDIA NETWORKS

Decentralized social media, also known as blockchain-based social media, refers to social media platforms powered by distributed ledger technologies (DLT) like blockchain or DAG. As such, activities on these platforms are irrevocably recorded on a decentralized protocol where no central authority can control or oversee, unlike centralized networks like Facebook, Twitter, etc. The main feature is the use of blockchain and smart contract systems**.** This way, users have more control and autonomy. The user can decide the settings of the social network and determine how it works and what users can say. Decentralized social media platforms live on public blockchains, and for the most part, anyone, anywhere, can operate a node, access the back end, create an app and curate a feed.

Conventional social platforms such as facebook, twitter not only control users posts but also what they see, since these networks tend to prioritize revenue generation. Therefore, they present users with attention-grabbing advertising content, making them more entertained than well-informed.On the other hand, users on decentralized social media networks have the freedom to interact however they wish without censorship as the developers usually only provide guiding rules, leaving the rest to a distributed group of users.

Decentralized social networks make up the fediverse, a term for a collection of interconnected servers used for social networking and other activities such as blogging and web publishing. An independently hosted federated network can interact with other networks in the fediverse.

Decentralized social media platforms prevent the unauthorized sale of user data which is one of the contentious issues with centralized social networks. Moreover, blockchain technology increases user privacy and data security using cryptography methods. Economic neutrality is essential for many who turn to decentralized social networks, who seek freedom from invasive advertising .Decentralized networks are looking for new forms of monetization to stay solvent. For example, paying their users with cryptocurrencies to create or curate content. This way, content creators are incentivized to focus on quality.

1.2 Decentralized Twitter

Decentralized twitter is a fully functional twitter like application where anyone can use their crypto wallets to connect and start posting tweets. The main ideology behind this application is web 3.0 which is a more intelligent web that will be able to interpret and

interconnect a greater number of data in decentralized way without any intermediatories. The web 3.0 provides a very useful technology named blockchain which enables the existence of cryptocurrency. In the decentralized Twitter the users have more control and authority rather than the companies. Some of the user control features of this application are ownership of personal data, control over the authority, content of censorship, privacy and security.

The decentralized Twitter can be used to tweet content, update the tweets, delete the tweets and search desired tweets based on the hashtags. All these operations are done with the cryptocurrency in the wallets of the users. The type of cryptocurrency used in this application is Solana. This type of application has more privacy than the normal social networking sites due to the use of cryptocurrency. The data theft is almost zero.

2. SYSTEM ANALYSIS

2.1 Web 1.0

Web 1.0 refers to the first stage of the World Wide Web evolution. Earlier, there were only a few content creators in Web 1.0 with a huge majority of users who are consumers of content. Personal web pages were common, consisting mainly of static pages hosted on ISP-run web servers, or on free web hosting services.

In Web 1.0 advertisements on websites while surfing the internet are banned. Also, in Web 1.0, Ofoto is an online digital photography website, on which users could store, share, view, and print digital pictures. Web 1.0 is a content delivery network (CDN) that enables the showcase of the piece of information on the websites. It can be used as a personal website. It costs the user as per pages viewed. It has directories that enable users to retrieve a particular piece of information.

Web 1.0 was the first stage of the World Wide Web revolution, usually referred to as read-only web. Websites were informational and contained only static content that was hyperlinked together or in simple words there was no CSS, dynamic links, interactivity like logging in the users, comments on the blog posts etc.

The websites were built using Server Side Includes or Common Gateway Interface (CGI) instead of a web application written in a dynamic programming language such as Perl, PHP, Python or Ruby.

In the era of Web 1.0 i.e. from 1991 to 2004, users on the internet were consumers of content created by content creators.

- **Four design essentials of a Web 1.0 site include:**

1. Static pages.
2. Content is served from the server's file system.
3. Pages built using Server Side Includes or Common Gateway Interface (CGI).
4. Frames and Tables are used to position and align the elements on a page.

2.2 Web 2.0

Web 2.0 from 2004 till now, is the second stage of the World Wide Web revolution, usually referred to as read-write web. Emphasis was given to user-generated content, ease of use, participatory culture and interoperability.

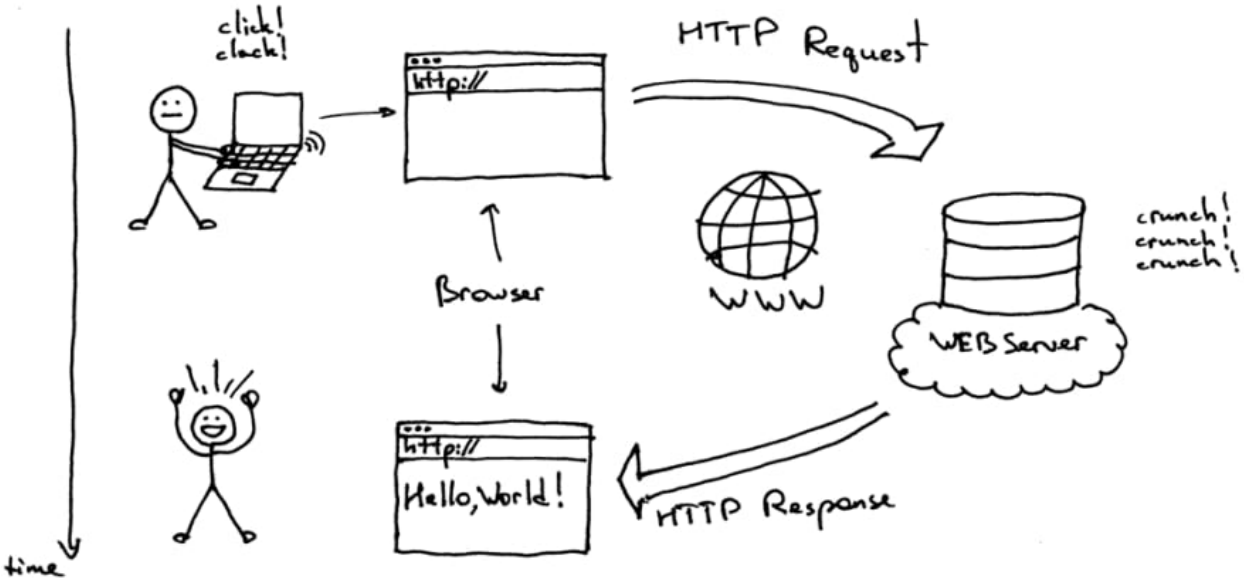
Some of the early platform based on web 2.0 are YouTube, Facebook, Amazon and so on. Due to CMS technologies such as WordPress, blogging, creating a ecommerce shop became extremely popular.

Web 2.0 brought about a fundamental shift, where people could share their perspectives, opinions, thoughts, and experiences through a variety of online tools and platforms.

- **Web 2.0 Disadvantages**

Before looking at disadvantages let's see how a traditional web 2.0 application works.
A client(user) will make a HTTP request to the server and if everything is correct it will send back that webpage as a response.

A major flaw here is that all data is stored on a centralized server, controlled by the companies. Facebook, Google, and Twitter began storing users data in their servers so that they can serve us better content through machine.



Facebook user data shapes algorithms that determine what users see in their news feeds in a way that can present a one-sided, echo-chamber view of the world. A view that only affirms and never challenges our views can increase divisiveness and social unrest.
Danger 3: Centralized services are easy to hack.
You're probably familiar with this argument if you've been reading about blockchain for a while. When a company uses centralized learning. This in turn would make us stay on their websites longer, therefore providing more ad revenue for these companies.
The companies eventually started selling our information to advertisers, which meant more money for them. To summarize web 2.0 we can say that users are the product.

In February 2021, WhatsApp changed its privacy rules in a take-it-or-leave-it announcement: it would harvest more user data for profit. Millions swore they would ditch the app for more private alternatives — including yours truly. Not enough to escape the network's gravitational pull, it turns out. While many chat on Signal and Telegram these days, few managed to get off WhatsApp completely. You still want to talk to Mom and Mom still wants to talk to Dad. Platforms own everything we create online.
That includes:
- Profile data we fill out
-The behavioural data we generate
-The images, videos, songs, status updates and comments we upload.
Whatever we do on platform is platform property.

- **Deplatforming & censorship**

When Twitter and Facebook banned Donald Trump, he told his supporters to follow him to Parler. Next thing Apple and Google removed Parler's mobile app from their app stores. Whereupon Amazon delivered the final blow by kicking Parler's website from its hosting servers. Trump is digitally homeless.

- **Hacker's paradise**

An interconnected economy that combines decentralized data creation with centralized storage provides enormous rewards for hackers.

- **Trust Issues**

We can't trust banks, Facebook and Servers to take care of trust because we can't ultimately trust the individuals that constitute them

2.3 EXISTING SYSTEM

- **Social networks of today**

Social media is a collective term for websites and applications that focus on communication, community-based input, interaction, content-sharing and collaboration.

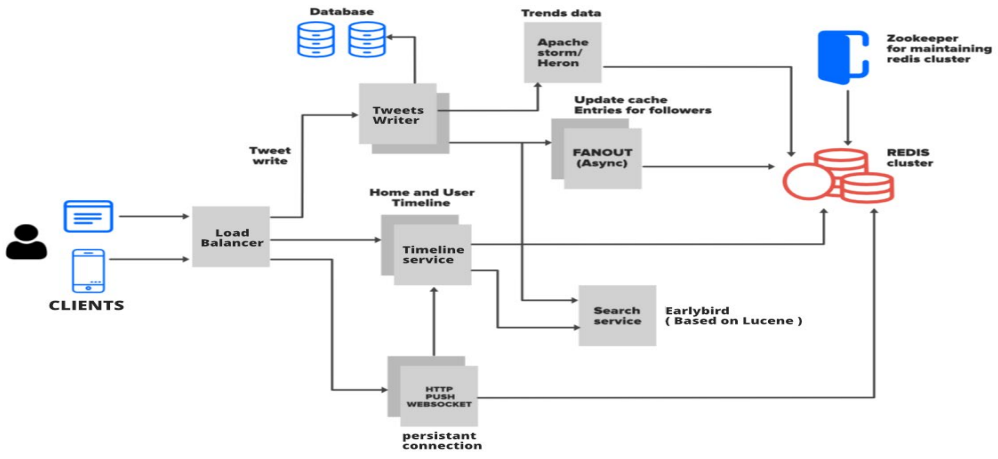
People use social media to stay in touch and interact with friends, family and various communities. Businesses use social applications to market and promote their products and track customer concerns.

Business-to-consumer websites include social components, such as comment fields for users. Various tools help businesses track, measure and analyze the attention the company gets from social media, including brand perception and customer insight.

Social media has enormous traction globally. Mobile applications make these platforms easily accessible. Some popular examples of general social media platforms include Twitter, Facebook and LinkedIn.

- **TWITTER**

Twitter is a a microblogging platform where messages (tweets) are posted in 140 characters or less . On Twitter, communities are built around tweets (less about who you know than what you have to say)



- **DRAWBACKS OF EXISTING SYSTEM**

Social media has become a huge part of our lives. We turn to it to form groups, foster relationships, and keep in touch with long-distance friends. We use it to share our hopes, our fears, our vacation photos, and our cat videos. And we use it to stay on top of local, national, and international news and politics.

But centralized social media continues to breach our trust. Seemingly benign quizzes and games leak our data and our friends' data to huge organizations with vote-garnering and money-making motives. News feeds don't provide accurate and well-rounded perspectives on current events. Advertisers stalk us from one website to the next.

Fortunately, a wave of visionary computer programmers are developing new services that circumvent the dangers of our current social media framework through decentralization. Here's an overview of the problems with the most popular platforms and how they might be solved.

Danger 1: Users are the product, not the customer.

Centralized social media is not a benign way to connect people to one another. While Facebook may connect people better than any other service, it does so at a cost: it gathers troves of data on participants and uses that data to sell targeted advertising.

Advertisers are not just companies that want us to buy their goods and services, but also ones that want us to think a certain way about a political candidate or a social issue so that we'll vote a certain way. With centralized social media applications like Facebook, Instagram, LinkedIn, Twitter, and YouTube, advertisers are the customer, and user data is the product.

Danger 2: Algorithmically curated feeds can increase bias and polarization.

Many of us get at least some of our news, or links to news stories, from our Facebook and Twitter, feeds. So it matters how our social media feeds present information to us. The Wall Street Journal's 2016 application, Blue Feed, Red Feed, shows how different a user's news feed might look based on what Facebook knows about that user. Someone categorized as liberal will probably see dramatically different news stories than someone categorized as conservative.

It's true that users sometimes seek out biased news sources that support their worldview. But it's one thing to make a conscious choice to do that; it's another thing to have Facebook do it to you because its advertisers, which include media outlets, have identified you as a certain type of person based on your likes and other data the platform has gathered about you, analyzed, and packaged up.Storage for user data, any breach of that system exposes enormous troves of data in one fell swoop.

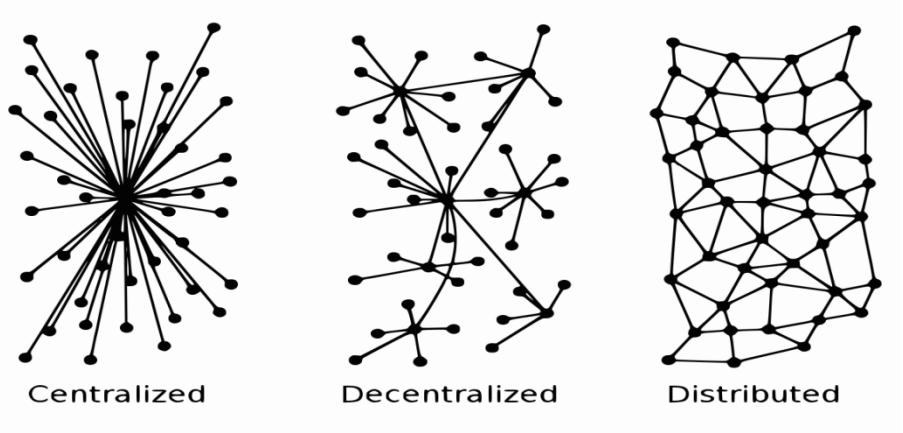
We've seen it happen to Facebook, Google+, and LinkedIn, not to mention the Securities and Exchange Commission, Equifax, Deloitte, Anthem Blue Cross, Sony, Target, and many more companies that store our data this way. The question no longer seems to be if but when a given database will be compromised. We're all vulnerable: every company, every organization, every individual.

- **Decentralized Solutions to Centralized Problems**

Developers have been working on decentralized alternatives to centralized social media. These decentralized alternatives are not owned by major companies. Because of that, they don't have shareholders whose interests might not be aligned with those of users. They also don't store data on those centralized servers that create an irresistible target for hackers.

2.4 PROPOSED SYSTEM

- **Decentralized System**



- **Decentralized system**

A decentralized system is an interconnected information system where no single entity is the sole authority. In the context of computing and information technology, decentralized systems usually take the form of networked computers. For example, the Internet is a decentralized system, although it has become increasingly centralized over time.
Decentralized systems are powered by blockchain networks

What Is a Blockchain?

A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure

and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

One key difference between a typical database and a blockchain is how the data is structured. A blockchain collects information together in groups, known as blocks, that hold sets of information. Blocks have certain storage capacities and, when filled, are closed and linked to the previously filled block, forming a chain of data known as the blockchain. All new information that follows that freshly added block is compiled into a newly formed block that will then also be added to the chain once filled.

A database usually structures its data into tables, whereas a blockchain, like its name implies, structures its data into chunks (blocks) that are strung together. This data structure inherently makes an irreversible timeline of data when implemented in a decentralized nature. When a block is filled, it is set in stone and becomes a part of this timeline. Each block in the chain is given an exact time stamp when it is added to the chain.

- **KEY TAKEAWAYS**

Blockchain is a type of shared database that differs from a typical database in the way that it stores information; blockchains store data in blocks that are then linked together via cryptography.

As new data comes in, it is entered into a fresh block. Once the block is filled with data, it is chained onto the previous block, which makes the data chained together in chronological order.
Different types of information can be stored on a blockchain, but the most common use so far has been as a ledger for transactions.

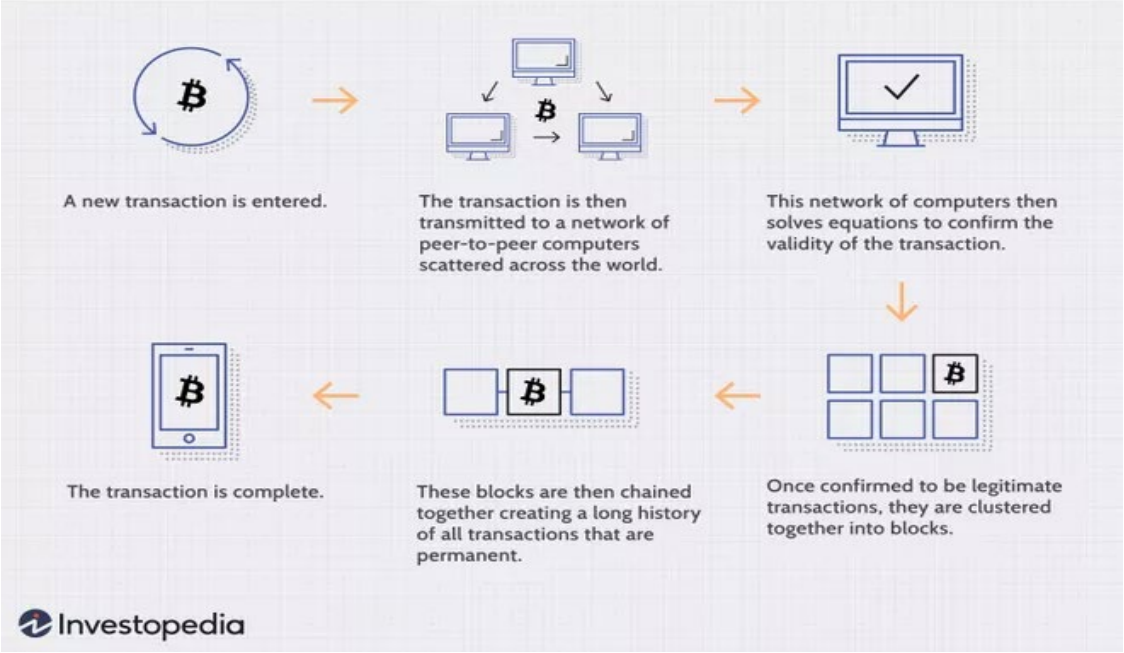
In Bitcoin's case, blockchain is used in a decentralized way so that no single person or group has control—rather, all users collectively retain control.
Decentralized blockchains are immutable, which means that the data entered is irreversible. For Bitcoin, this means that transactions are permanently recorded and viewable to anyone.

- **How Does a Blockchain Work?**

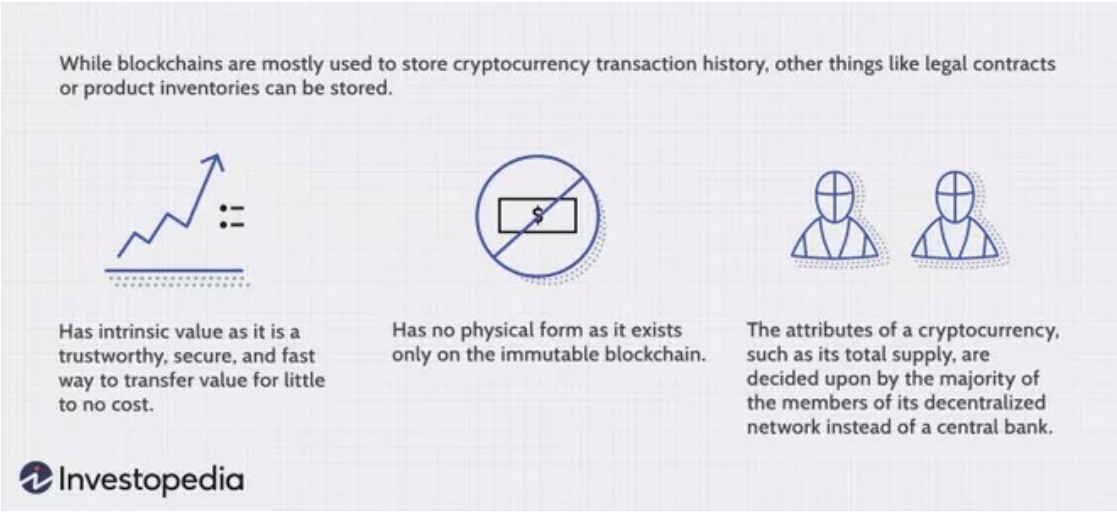
The goal of blockchain is to allow digital information to be recorded and distributed, but not edited. In this way, a blockchain is the foundation for immutable ledgers, or records of transactions that cannot be altered, deleted, or destroyed. This is why blockchains are also known as a distributed ledger technology (DLT).

First proposed as a research project in 1991, the blockchain concept predated its first widespread application in use: Bitcoin, in 2009. In the years since, the use of blockchains has exploded via the creation of various cryptocurrencies, decentralized finance (DeFi) applications, non-fungible tokens (NFTs), and smart contracts.

- **Transaction Process**



- **Attributes of Cryptocurrency**



- **Blockchain Decentralization**

Magine that a company owns a server farm with 10,000 computers used to maintain a database holding all of its client's account information. This company owns a warehouse building that contains all of these computers under one roof and has full control of each of these computers and all of the information contained within them. This, however, provides a single point of failure. What happens if the electricity at that location goes out? What if its Internet connection is severed? What if it burns to the ground? What if a bad actor erases everything with a single keystroke? In any case, the data is lost or corrupted.

What a blockchain does is to allow the data held in that database to be spread out among several network nodes at various locations. This not only creates redundancy but also maintains the fidelity of the data stored therein—if somebody tries to alter a record at one instance of the database, the other nodes would not be altered and thus would prevent a bad actor from doing so. If one user tampers with Bitcoin's record of transactions, all other nodes would cross-reference each other and easily pinpoint the node with the incorrect information. This system helps to establish an exact and transparent order of events. This way, no single node within the network can alter information held within it.

Because of this, the information and history (such as of transactions of a cryptocurrency) are irreversible. Such a record could be a list of transactions (such as with a cryptocurrency), but it also is possible for a blockchain to hold a variety of other information like legal contracts, state identifications, or a company's product inventory.

To validate new entries or records to a block, a majority of the decentralized network's computing power would need to agree to it. To prevent bad actors from validating bad transactions or double spends, blockchains are secured by a consensus mechanism such as proof of work (PoW) or proof of stake (PoS). These mechanisms allow for agreement even when no single node is in charge.

- **Transparency**

Because of the decentralized nature of Bitcoin's blockchain, all transactions can be transparently viewed by either having a personal node or using blockchain explorers that allow anyone to see transactions occurring live. Each node has its own copy of the chain that gets updated as fresh blocks are confirmed and added. This means that if you wanted to, you could track Bitcoin wherever it goes.

For example, exchanges have been hacked in the past, where those who kept Bitcoin on the exchange lost everything. While the hacker may be entirely anonymous, the Bitcoins that

they extracted are easily traceable. If the Bitcoins stolen in some of these hacks were to be moved or spent somewhere, it would be known.

Of course, the records stored in the Bitcoin blockchain (as well as most others) are encrypted. This means that only the owner of a record can decrypt it to reveal their identity (using a public-private key pair). As a result, users of blockchains can remain anonymous while preserving transparency.

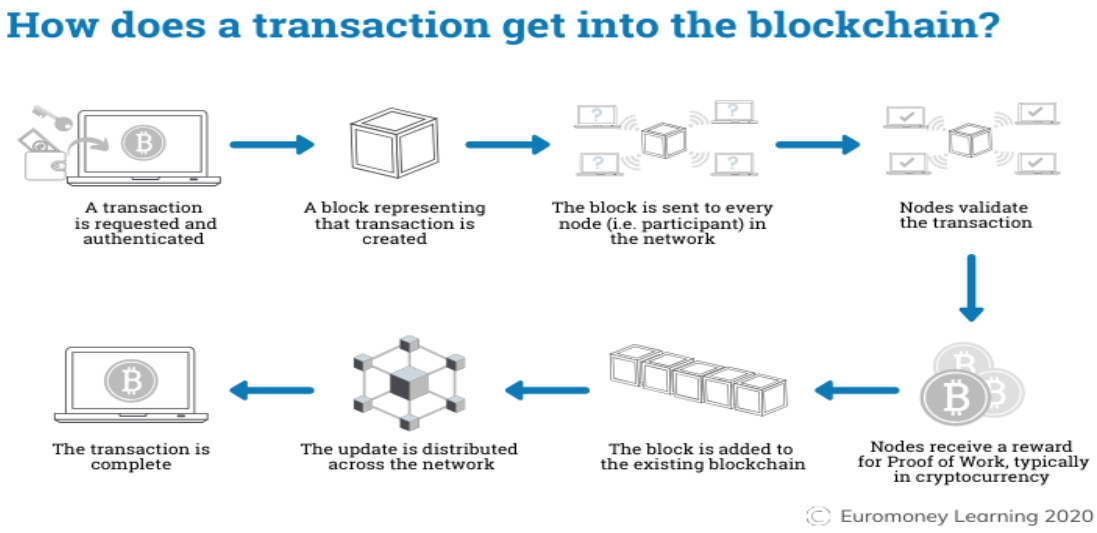
- **Is Blockchain Secure?**

Blockchain technology achieves decentralized security and trust in several ways. To begin with, new blocks are always stored linearly and chronologically. That is, they are always added to the "end" of the blockchain. After a block has been added to the end of the blockchain, it is extremely difficult to go back and alter the contents of the block unless a majority of the network has reached a consensus to do so. That's because each block contains its own hash, along with the hash of the block before it, as well as the previously mentioned time stamp. Hash codes are created by a mathematical function that turns digital information into a string of numbers and letters. If that information is edited in any way, then the hash code changes as well.

Let's say that a hacker, who also runs a node on a blockchain network, wants to alter a blockchain and steal cryptocurrency from everyone else. If they were to alter their own single copy, it would no longer align with everyone else's copy. When everyone else cross-references their copies against each other, they would see this one copy stand out, and that hacker's version of the chain would be cast away as illegitimate.

Succeeding with such a hack would require that the hacker simultaneously control and alter 51% or more of the copies of the blockchain so that their new copy becomes the majority copy and, thus, the agreed-upon chain. Such an attack would also require an immense amount of money and resources, as they would need to redo all of the blocks because they would now have different time stamps and hash codes.

Due to the size of many cryptocurrency networks and how fast they are growing, the cost to pull off such a feat probably would be insurmountable. This would be not only extremely expensive but also likely fruitless. Doing such a thing would not go unnoticed, as network members would see such drastic alterations to the blockchain. The network members would then hard fork off to a new version of the chain that has not been affected. This would cause the attacked version of the token to plummet in value, making the attack ultimately pointless, as the bad actor has control of a worthless asset. The same would occur if the bad actor were to attack the new fork of Bitcoin. It is built this way so that taking part in the network is far more economically incentivized than attacking it.



- **Consensus Mechanisms**

Blockchain is a distributed decentralized network that provides immutability, privacy, security, and transparency. There is no central authority present to validate and verify the transactions, yet every transaction in the Blockchain is considered to be completely secured and verified. This is possible only because of the presence of the consensus protocol which is a core part of any Blockchain network. A consensus algorithm is a procedure through which all the peers of the Blockchain network reach a common agreement about the present state of the distributed ledger. In this way, consensus algorithms achieve reliability in the Blockchain network and establish trust between unknown peers in a distributed computing environment. Essentially, the consensus protocol makes sure that every new block that is added to the Blockchain is the one and only version of the truth that is agreed upon by all the nodes in the Blockchain. The Blockchain consensus protocol consists of some specific objectives such as coming to an agreement, collaboration, co-operation, equal rights to every node, and mandatory participation of each node in the consensus process. Thus, a consensus algorithm aims at finding a common agreement that is a win for the entire network. Now, we will discuss various consensus algorithms and how they work.

Proof of Work (PoW): This consensus algorithm is used to select a miner for the next block generation. Bitcoin uses this PoW consensus algorithm. The central idea behind this algorithm is to solve a complex mathematical puzzle and easily give out a solution. This mathematical puzzle requires a lot of computational power and thus, the node who solves the puzzle as soon as possible gets to mine the next block.

Proof of Stake (PoS): This is the most common alternative to PoW. Ethereum has shifted from PoW to PoS consensus. In this type of consensus algorithm, instead of investing in expensive hardware to solve a complex puzzle, validators invest in the coins of the system by locking up some of their coins as stake. After that, all the validators will start validating the blocks. Validators will validate blocks by placing a bet on it if they discover a block which they think can be added to the chain. Based on the actual blocks added in the Blockchain, all the validators get a reward proportionate to their bets and their stake increase accordingly. In the end, a validator is chosen to generate a new block based on their economic stake in the network. Thus, PoS encourages validators through an incentive mechanism to reach to an agreement.

- **Proof of history**

Instead of trusting the timestamp on the transaction, you could prove that the transaction occurred sometime before and after an event. The Proof of History is a high frequency Verifiable Delay Function. A Verifiable Delay Function requires a specific number of sequential steps to evaluate, yet produces a unique output that can be efficiently and publicly verified.

Proof of History is a sequence of computation that can provide a way to cryptographically verify passage of time between two events. It uses a cryptographically secure function written so that output cannot be predicted from the input, and must be completely executed to generate the output. The function is run in a sequence on a single core, its previous output as the current input, periodically recording the current output, and how many times its been called. The output can then be re-computed and verified by external computers in parallel by checking each sequence segment on a separate core. Data can be timestamped into this sequence by appending the data (or a hash of some data) into the state of the function. The recording of the state, index and data as it was appended into the sequences provides a timestamp that can guarantee that the data was created sometime before the next hash was generated in the sequence. This design also supports horizontal scaling as multiple generators can synchronize amongst each other by mixing their state into each others sequences.

- **Smart Contracts**

A smart contract is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement.The objectives of smart contracts are the reduction of need in trusted intermediators, arbitrations and enforcement costs, fraud losses, as well as the reduction of malicious and accidental exceptions.

Vending machines are mentioned as the oldest piece of technology equivalent to smart contract implementation.[3] 2014's white paper about the cryptocurrency Ethereum[6] describes the Bitcoin protocol as a weak version of the smart contract concept as defined by computer scientist, lawyer and cryptographer Nick Szabo. Since Bitcoin, various cryptocurrencies support scripting languages which allow for more advanced smart contracts between untrusted parties.Smart contracts should be distinguished from smart legal contracts. The latter refers to a traditional natural language legally-binding agreement which has certain terms expressed and implemented in machine-readable code.

2.5 Blockchain network used in this project:

- **Solana:**

Solana is a public blockchain platform with smart contract functionality. Its native cryptocurrency is SOL.

Solana was proposed in a white paper Anatoly Yakovenko which was published in November of 2017. This paper described a technique called "proof of history".

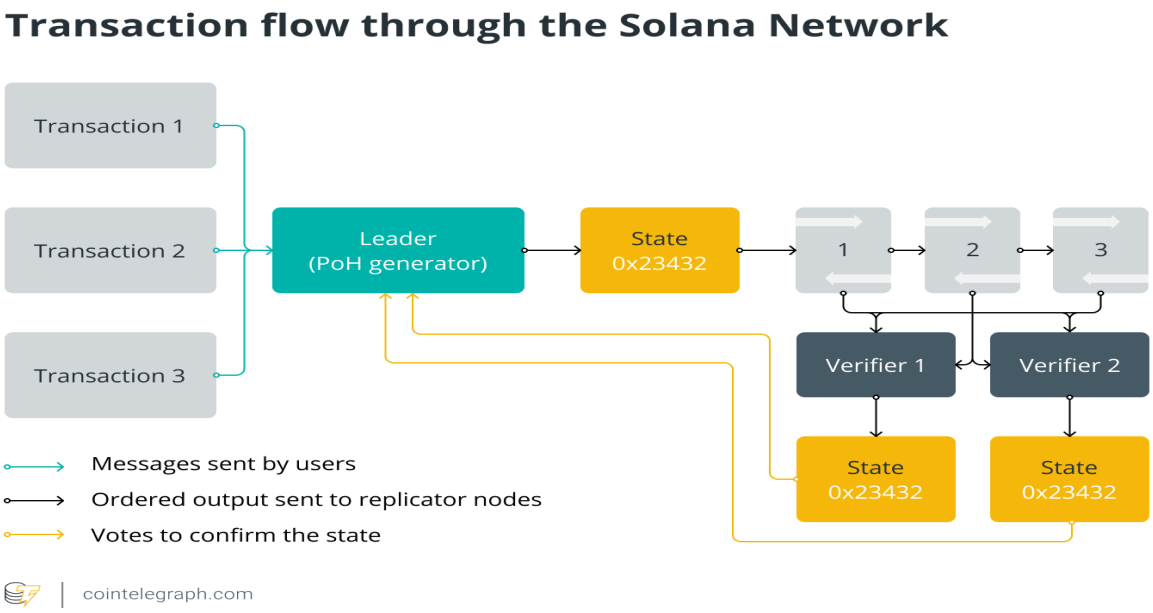
On 16 March 2020, Solana's first block was created.

In September 2021, *Bloomberg* journalist Joanna Ossinger described Solana as "a potential long-term rival for Ethereum", citing superior transaction speeds and lower associated costs.

On 14 September 2021, the Solana blockchain went offline after a surge of transactions caused the network to fork, and different validators had different views on the state of the network. The network was brought back online the next day on 15 September 2021.

On 16 December 2021, the former First Lady of the United States Melania Trump announced her plans to use Solana to launch a non-fungible token (NFT). The Solana Foundation issued a press release to clarify that her choice of the platform was not officially "part of any Solana-led initiative."

Solana achieves consensus using a proof-of-stake mechanism and a "proof-of-history" mechanism. Like various other blockchains, Solana can run smart contracts.Solana's execution environment is based on eBPF, which allows the Rust, C, and C++ programming languages to be used.



- **The Solana (SOL) token**

Solana's cryptocurrency is SOL. It is Solana's native and utility token that provides a means of transferring value as well as blockchain security through staking. SOL was launched in March 2020 and has strived to become one of the top 10 cryptocurrencies entering the space by means of total market capitalization.

SOL token operation scheme is similar to that used in the Ethereum blockchain. Even though they function similarly, Solana token holders stake the token in order to validate transactions through the PoS consensus mechanism. Furthermore, the Solana token is used to receive rewards and pay transaction fees while also SOL enabling users to participate in governance.

Answering the question of how many Solana coins are there, there will be more than 500 million tokens released in circulation with the current total supply of Solana exceeding 511 million tokens — Solana's circulating supply is just over half that. Around 60% of SOL tokens are controlled by Solana's founders and the Solana Foundation, with only 38% reserved for the community.
If you would like to know where to buy Solana, SOL tokens can be purchased on most exchanges. The top cryptocurrency exchanges for trading in Solana are Binance, Coinbase, KuCoin, Huobi, FTX and others.

2.6 TWITTER VS DECENTRALIZED TWITTER

- **TWITTER:**

  - When you sign up for Twitter as @ROHIT, a @ROHIT named directory is created in Twitter's database — hosted on a central server like AWS.

  - Every time @ROHIT tweets, that goes into the server and now Twitter owns it.
  - If the tweet went viral, twitter is getting profited.

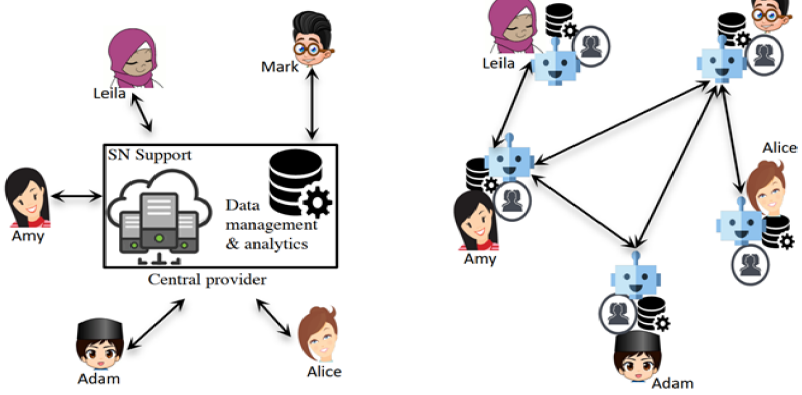
- **DECENTRALIZED TWITTER**

  - @ROHIT joins dTwitter by connecting his crypto wallet.

  - When @ROHIT tweets, a new file is created and stored on a decentralised file storage system like the InterPlanetary File System (IPFS).

  - At the same time, a token that represents the tweet-file is 'minted' on the dTwitter blockchain and allocated to @ROHIT wallet address. It sits in @ROHIT'S wallet and so @ROHIT effectively 'owns' it.

  - @ROHIT can transfer ownership of his tweets to other wallets.



2.7 REQUIREMENTS

2.7.1 Software requirements

- ***An operating system from the unix family (Preferably Ubuntu)***

- ***A crypto-wallet*** (Eg. Phantom)
  Phantom makes it safe & easy for you to store, buy, send, receive, swap tokens and collect NFTs on the Solana blockchain.

- ***Solana CLI Tools***

- ***Rust***
  Rust is a multi-paradigm, general-purpose programming language designed for performance and safety, especially safe concurrency.

- ***Cargo***
  Cargo is the Rust package manager. Cargo downloads your Rust package's dependencies, compiles your packages, makes distributable packages, and uploads them to crates.io, the Rust community's package registry.

- **Anchor**
  Anchor is a framework for Solana's Sealevel (opens new window)runtime providing several convenient developer tools.

- **Vue.js**
  Vue.js is an open-source model–view–viewmodel front end JavaScript framework for building user interfaces and single-page applications.

- **Node.js**
  Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

2.7.2Hardware Requirements:

- **Memory:** 16GB
- **Processor:** 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 8
- **Disk Capacity:** 512GB

3. SYSTEM DESIGN

System Design is the process of defining the elements of a system such as the architecture, modules, and components, the different interfaces of those components, and the data that goes through that system. It is meant to satisfy specific needs and requirements of a business or organisation through the engineering of a coherent and well-running system.
Systems design mainly concentrates on defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

Systems design implies a systematic approach to the design of a system. It may take a bottom-up or top-down approach, but either way, the process is systematic wherein it takes into account all related variables of the system that needs to be created—from the architecture to the required hardware and software, right down to the data and how it travels and transforms throughout its travel through the system. Systems design then overlaps with systems analysis, systems engineering, and systems architecture.
The Decentralized twitter application in this project has the following modules.

3.1 Modules of the proposed system

- **Structuring of the tweet account**

The tweet account must be structured in such a way that it must include all the necessary fields such as author, content , topic etc.

- **Writing the instructions**

The off-chain code communicates with the on chain code using instructions. The instructions can be accessed from the front end through RPC's (Remote procedure calls).

- **Testing the instructions**

After writing the instructions that need to be executed on the back end , we can write tests on the client side(off chain) to make sure all our instructions execute as expected.

- **Fetching tweets from the program:**

We must fetch multiple tweet accounts at once and also filter these tweets by various properties.

- **Scaffolding the frontend**

We want to abstract all of that into a nice user interface (UI) that resembles what they are familiar with. For that reason, we will build a frontend client and we'll build it using VueJS.

- **Integrating with Solana wallets**

We'll focus on integrating our frontend with Solana wallet providers such as Phantom or Solfare so we can send transactions on behalf of a user. Once we'll have access to the connected wallet, we'll be able to create a "Program" object just like we did in our tests.

- **Fetching tweets in the frontend**

Now, it's time to use that program object in the previous module to remove all the mock data and fetch real tweets from the blockchain.

- **Sending new tweets from the frontend**

We need to define some components on the front end to enable ourselves to send tweets from the frontend.

- **Deploying to devnet**

In Solana, there are multiple clusters we could deploy to. The main one that everybody uses with real money is called "mainnet". Another one, called "devnet", can be used to test our program on a live cluster that uses fake money.When deploying dApps, it is common to first deploy to the "devnet" cluster as you would on a staging server and then, when you're happy with everything, deploy to the "mainnet" cluster analogous to a production server.

- **Updating tweets**

Apart from adding the tweets we can also add some features like updating an existing tweet.

- **Deleting tweets**

Deleting on the blockchain can get quite tricky as the accounts on the blockchain are immutable. This module is also implemented here.

- **Paginating tweets**

When a program starts getting lots of accounts, it can be tricky to retrieve them whilst keeping a decent user experience. In this module, we use tricks to paginate and order accounts from our program.

3.2 UML DIAGRAM

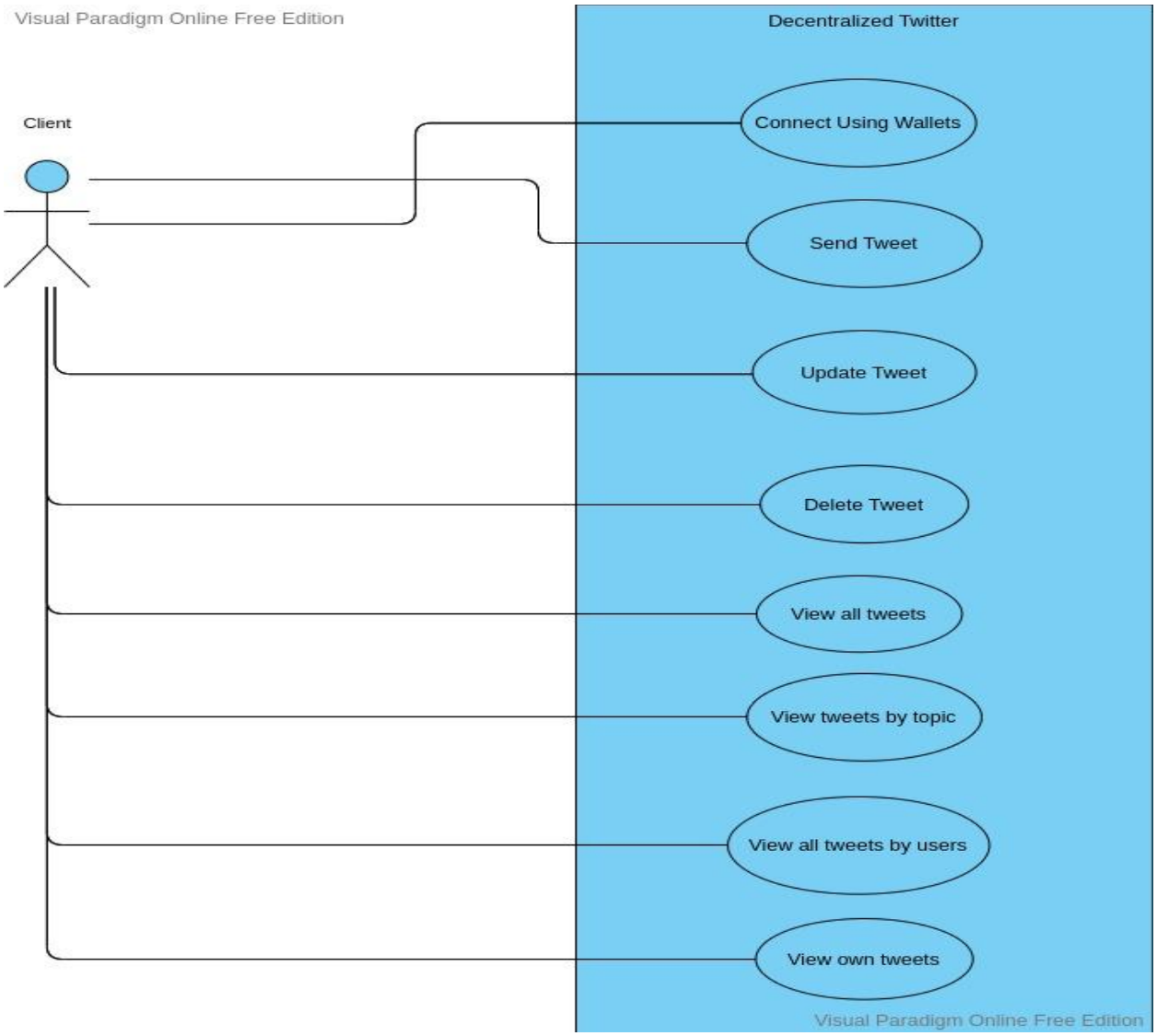
UML is a standard language for specifying, visualising, constructing, and documenting the artefacts of software systems. The UML uses mostly graphical notations to express the design of software projects. It is a very important part of involving object-oriented software and the software development process.

3.2.1 Use Case diagram

Use case diagrams are usually referred to as behaviour diagrams, used to describe a set of actions that some systems should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Table 3.1: Graphical representation of Use-case diagram

Actor	An actor in the Unified ModellingLanguage specifies a role played by a user or any other system that interacts with the subject.	
Use Case	A Use Case is the functionality provided by the system.Use Cases are depicated with an ellipse.The name of the Use Case is written in ellipse.	
Association	Association is a relationship between classifiers which is used to show that instances of classifiers could be either linked to each other or combined logically or physically into some aggregation.	



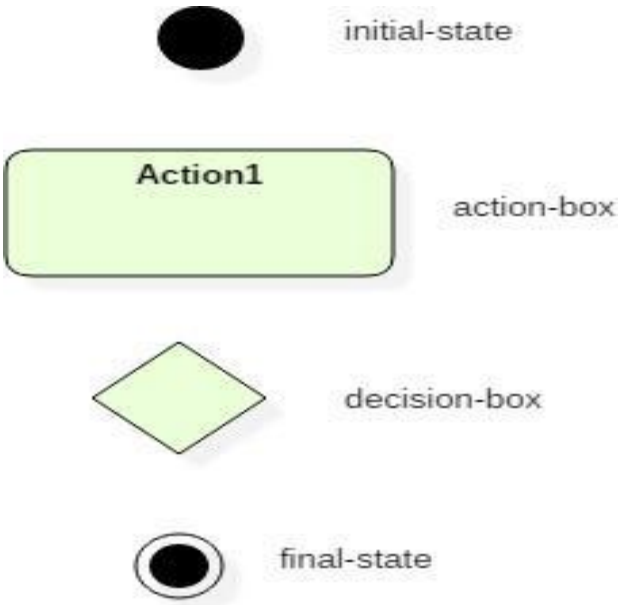
3.2.2 Activity diagram

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modelling. They can also describe the steps in a use case diagram. Activities modelled can be sequential and concurrent.

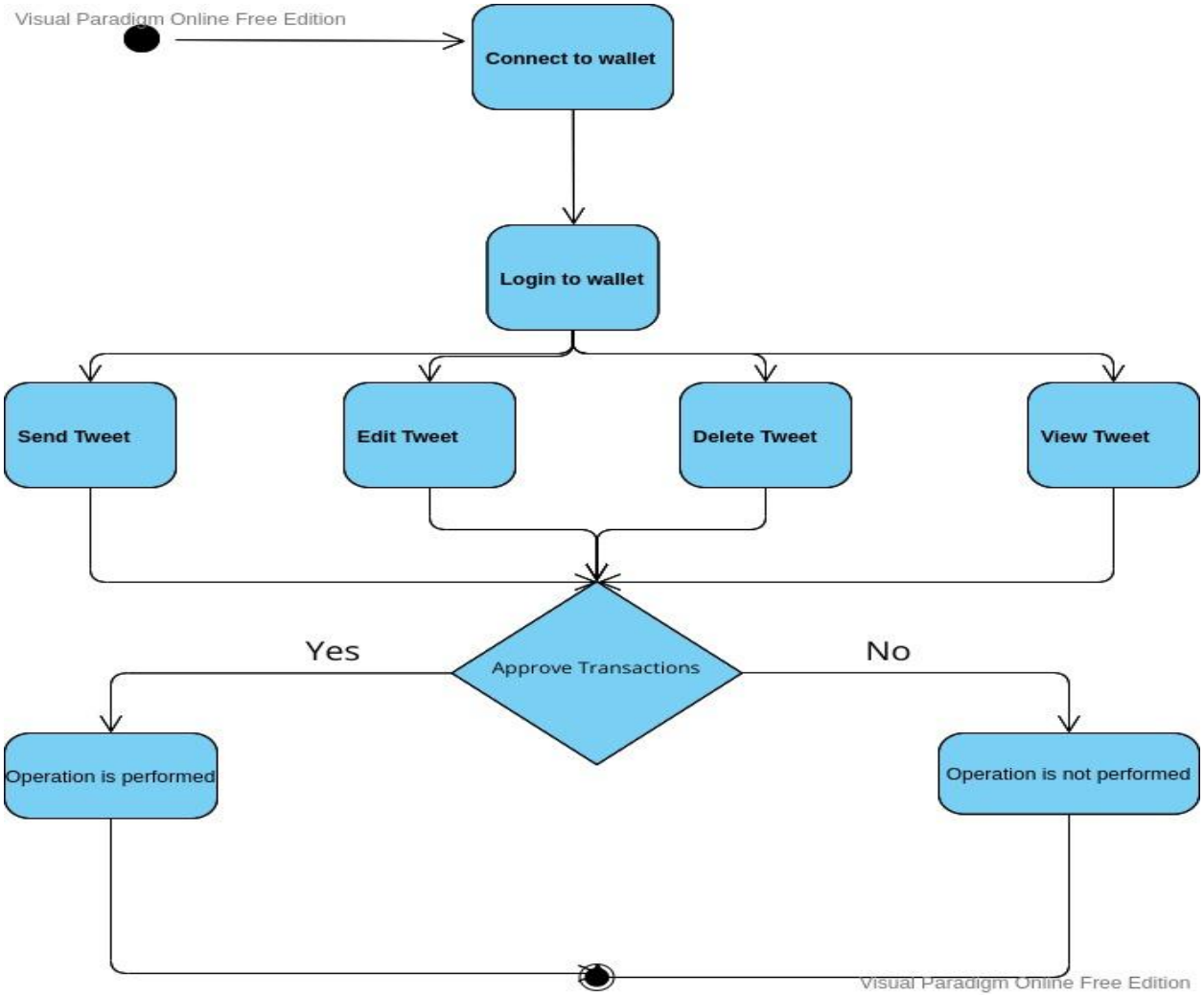
- **Activity Diagram Notations**

Activity diagrams symbol can be generated by using the following notations:

- **Initial states**: The starting stage before an activity takes place is depicted as the initial state
- **Final states**: The state which the system reaches when a specific process ends is known as a Final State.
- **Decision box:** It is a diamond shape box that represents a decision with alternate paths. It represents the flow of control.



Graphical representation of Activity Diagram



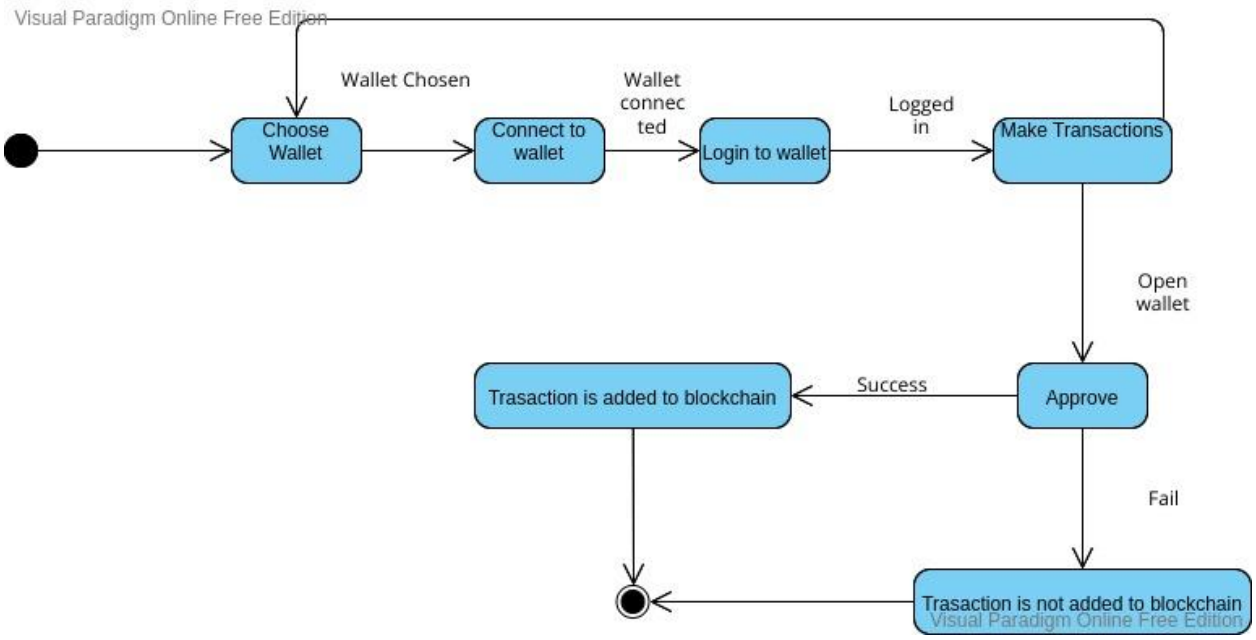
3.2.3 State chart Diagram

A Statechart diagram describes a state machine. The state machine can be defined as a machine that defines different states of an object and these states are controlled by external or internal events. A state diagram is used to describe the behaviour of systems.
This behaviour is analysed and represented in a series of events that could occur in one or more possible states. State diagrams require that the system described is composed of a finite number of states. Sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.
The main purpose of using State Chart diagrams –

- To model the dynamic aspect of a system.
- To model the lifetime of a reactive system.
- To describe different states of an object during its lifetime.
- Define a state machine to model the states of an object.

Table 3.2: Graphical representation of State Chart Diagram

Initial State	The initial state represents the source of all objects. A filled circle followed by an arrow represents the object's initial state.	
State	State represents situations during the life of an object. Rectangular boxes with curved edges represent a state	
Transition	A transition represents the change from one state to another. A solid arrow represents the path between different states of an object	
Final State	The final state represents the end of an object's existence. A final state is not a real state, because objects in this state don't exist anymore. A filled circle represents the final state.	

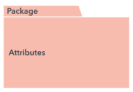
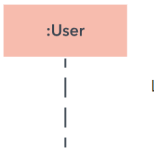


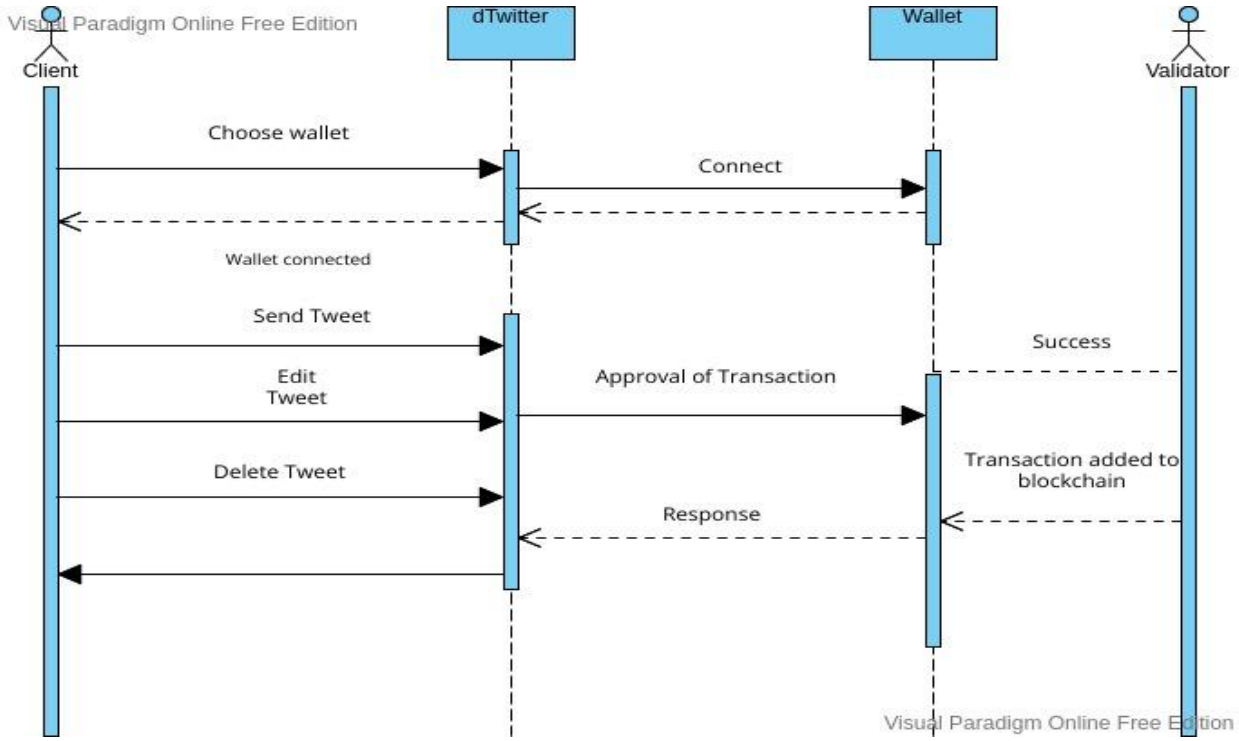
2.4 Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Table 3.3: Graphical representation of Sequence Diagram

Symbol	Name	Description
	Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.
	Activation box	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.

	Actor symbol	Shows entities that interact with or are external to the system.
	Package symbol	Used in UML 2.0 notation to contain interactive elements of the diagram. Also known as a frame, this rectangular shape has a small inner rectangle for labelling the diagram.
	Lifeline symbol	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labelled rectangle shape or an actor symbol.
	Option loop symbol	Used to model if/then scenarios, i.e., a circumstance that will only occur under certain conditions.



Sequence diagram

4. IMPLEMENTATION

4.1 INSTALLATION OF REQUIRED SOFTWARE

- **Installing Rust**

Rust is a type of programming language which could be installed from https://www.rust-lang.org/.Installing Rust is as simple as running this.

curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

After the installation, it will add the following path to the shell configurations

export PATH="$HOME/.cargo/bin:$PATH"

You can check that rust is properly installed by running the following commands.

rustup --version

rustc --version

cargo –version

- **Installing Solana**

Solana can be installed by running the following installer script.

sh -c "$(curl -sSfL https://release.solana.com/v1.9.4/install)"

After the installation, Solana will ask to add a new path to the shell configuration

export PATH="$HOME/.local/share/solana/install/active_release/bin:$PATH"

Run the following commands to check if Solana is installed properly

Check the Solana binary is available.

solana --version

Check you can run a local validator (Run Ctrl+C to exit).

Note this creates a "test-ledger" folder in your current directory.

solana-test-validator

This is to check if all the configurations point to localhost url's.

solana config set --url localhost

- **Generating local key pair:**

for generating a local key pair run the following code

solana-keygen new

- **Install Anchor:**

Install Anchor in your local machine by running the following command.

cargo install --git https://github.com/project-serum/anchor anchor-cli --locked

run the following command to check Anchor CLI is installed properly.

anchor –version

- **Installing yarn:**

 # Using npm global dependencies.

npm install -g yarn

 # Using homebrew on Mac.

brew install yarn

 # Using apt on Linux

apt install yarn

- **Create a new Anchor project:**

Run anchor init to start a new project

Go to your dev folder .

cd ~/Code

Create a new Anchor project.

anchor init solana-twitter

Cd into the newly created project.

cd solana-twitter

- **Build and deploy**

Anchor has two useful commands to build and deploy your programs for you.

compiles your program.

anchor build

deploys your compiled program.

anchor deploy

- **Update your program ID**

Now that we've run anchor build and anchor deploy for the first time, we need to update our program ID.

solana address -k target/deploy/solana_twitter-keypair.json

Outputs something like**:**

2EKFZUwMrNdo8YLRHn3CyZa98zp6WH7Zpg16qYGU7htD

now that we know our program ID, let's update it.

[programs.localnet]

solana_twitter =

"2EKFZUwMrNdo8YLRHn3CyZa98zp6WH7Zpg16qYGU7htD"

Then, in the lib.rs file of our Solana program. In my case, that's programs/solana-twitter/src/lib.rs.

```
use anchor_lang::prelude::*;
use anchor_lang::solana_program::system_program;
declare_id!("2EKFZUwMrNdo8YLRHn3CyZa98zp6WH7Zpg16qYGU7htD");
```

Finally, we need to build and deploy one more time to make sure our program is compiled with the right identifier.

anchor build

anchor deploy

4.2 On-Chain Code:

- ***Structuring our Tweet account***

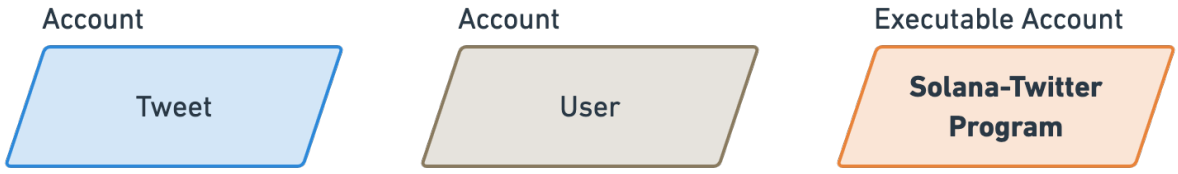
In Solana,account is everything.

This is a fundamental concept that differs from most other blockchains. For instance, if you've ever created a smart contract in Solidity, then you ended up with a bunch of code that can store a bunch of data and interact with it. Any user that interacts with a smart contract ends up updating data inside a smart contract. That's not the case in Solana.

In Solana, if you want to store data somewhere, you've got to create a new account. You can think of accounts as little clouds of data that store information for you in the blockchain. You can have one big account that stores all the information you need, or you can have many little accounts that store more granular information.

Programs may create, retrieve, update or delete accounts but they need accounts to store information as it cannot be stored directly in the program.But here is where it becomes more interesting: even programs are accounts.

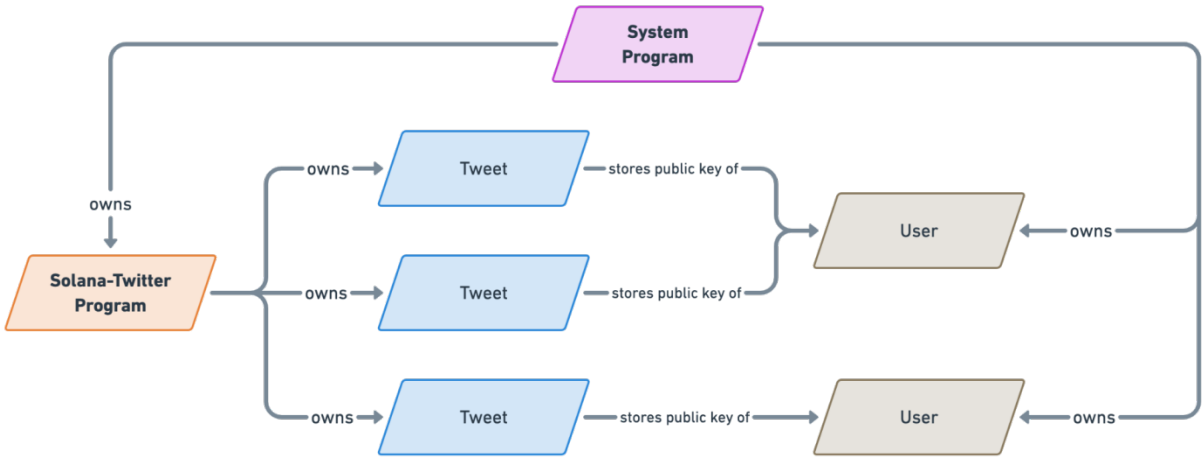
Programs are special accounts that store their own code, are read-only and are marked as "executable". There's literally an executable boolean on every single account that tells us if this account is a program or a regular account that stores data.So remember, everything is an account in Solana. Programs, Wallets, NFTs, Tweets, there're all made of accounts.




```
#[account]
pub struct Tweet {
    pub author: Pubkey,
    pub timestamp: i64,
    pub topic: String,
    pub content: String,
}
```

#[account]: This line is a custom Rust attribute provided by the Anchor framework. It removes a huge amount of boilerplate for us when it comes to defining accounts — such as parsing the account to and from an array of bytes. Thanks to Anchor, we don't need to know much more than that so let's be grateful for it.
pub struct Tweet: This is a Rust struct that defines the properties of our Tweet. If you're not familiar with structs (in Rust, C or other languages), you can think of them as classes that only define properties (no methods).
Author: We keep track of the user that published the tweet by storing its public key.
Timestamp: We keep track of the time the tweet by published by storing the current timestamp.

Topic: We keep track of an optional "topic" field that can be provided by the user so their tweet can appear on that topic's page. Twitter does that differently by parsing all the hashtags (#) from the tweet's content but that would be pretty challenging to achieve in Solana so we'll extract that to a different field for the sake of simplicity.
Content: Finally, we keep track of the actual content of the tweet.



- **Final Code:**

```
#[account]
pub struct Tweet {
   pub author: Pubkey,
   pub timestamp: i64,
   pub topic: String,
   pub content: String,
}
```

- *Add some useful constants for sizing propeties.*

```
const DISCRIMINATOR_LENGTH: usize = 8;
const PUBLIC_KEY_LENGTH: usize = 32;
const TIMESTAMP_LENGTH: usize = 8;
const STRING_LENGTH_PREFIX: usize = 4; // Stores the size of the string.
const MAX_TOPIC_LENGTH: usize = 50 * 4; // 50 chars max.
const MAX_CONTENT_LENGTH: usize = 280 * 4; // 280 chars max.
```

- *Add a constant on the Tweet account that provides its total size.*

```
impl Tweet {
   const LEN: usize = DISCRIMINATOR_LENGTH
```

```
    + PUBLIC_KEY_LENGTH // Author.
    + TIMESTAMP_LENGTH // Timestamp.
    + STRING_LENGTH_PREFIX + MAX_TOPIC_LENGTH // Topic.
    + STRING_LENGTH_PREFIX + MAX_CONTENT_LENGTH; // Content.
}
```

- **Writing Instructions**

Defining the context

```
#[derive(Accounts)]
pub struct SendTweet<'info> {
  pub tweet: Account<'info, Tweet>,
  pub author: Signer<'info>,
  pub system_program: AccountInfo<'info>,
}
```

Tweet: This is the account that the instruction will create. You might be wondering why we are giving an account to an instruction if that instruction creates it. The answer is simple: we're simply passing the public key that should be used when creating the account. We'll also need to sign using its private key to tell the instruction we own the public key. Essentially, we're telling the instruction: "here's a public key that I own, create a Tweet account there for me please".

Author: As mentioned above, we need to know who is sending the tweet and we need their signature to prove it.

System program: This is the official System Program from Solana. As you can see, because programs are stateless, we even need to pass through the official System Program. This program will be used to initialize the tweet account and figure out how much money we need to be rent-exempt.

#[derive(Accounts): This is a derive attribute provided by Anchor that allows the framework to generate a lot of code and macros for our struct context. Without it, these few lines of code would be a lot more complex.

<'info>: This is a Rust lifetime. It is defined as a generic type but it is not a type. Its purpose is to tell the Rust compiler how long a variable will stay alive for.

AccountInfo: This is a low-level Solana structure that can represent any account. When using AccountInfo, the account's data will be an unparsed array of bytes.

Account: This is an account type provided by Anchor. It wraps the AccountInfo in another struct that parses the data according to an account struct provided as a generic type. In the example above, Account<'info, Tweet> means this is an account of type Tweet and the data should be parsed accordingly.

Singer: This is the same as the AccountInfo type except we're also saying this account should sign the instruction.

Note that, if we want to ensure that an account of type Account is a signer, we can do this using account constraints.

- **Account constraints**

```
#[derive(Accounts)]
pub struct SendTweet<'info> {
    #[account(init, payer = author, space = Tweet::LEN)]
    pub tweet: Account<'info, Tweet>,
    #[account(mut)]
    pub author: Signer<'info>,
    #[account(address = system_program::ID)]
    pub system_program: AccountInfo<'info>,
}
```

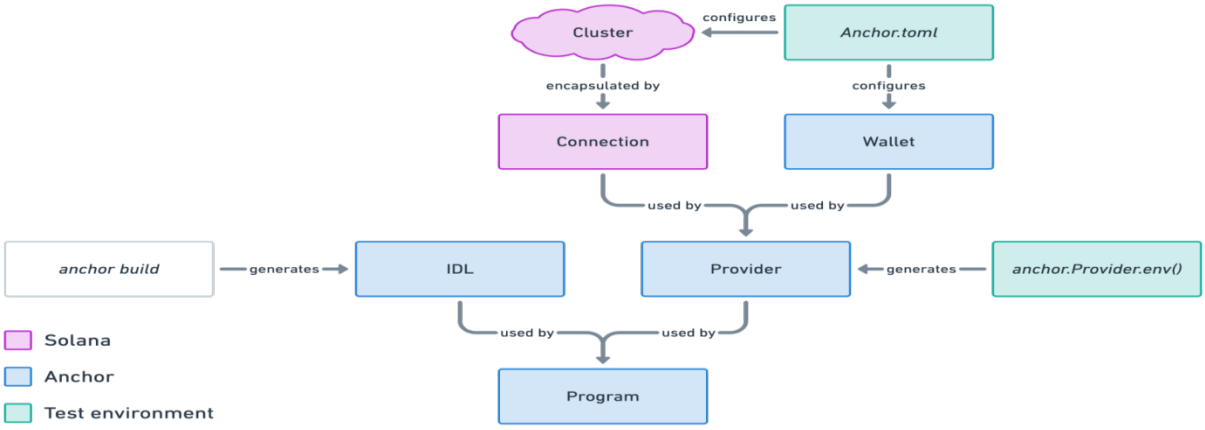
- **Instruction:**

```
pub fn send_tweet(ctx: Context<SendTweet>, topic: String, content: String) -> ProgramResult {
    let tweet: &mut Account<Tweet> = &mut ctx.accounts.tweet;
    let author: &Signer = &ctx.accounts.author;
    let clock: Clock = Clock::get().unwrap();

    if topic.chars().count() > 50 {
        return Err(ErrorCode::TopicTooLong.into())
    }

    if content.chars().count() > 280 {
        return Err(ErrorCode::ContentTooLong.into())
    }

    tweet.author = *author.key;
    tweet.timestamp = clock.unix_timestamp;
    tweet.topic = topic;
    tweet.content = content;
```

```
    Ok(())
}
```

4.3 Off-chain code:

5. Testing the instruction



- **Sending A Tweet:**

```
it('can send a new tweet', async () => {
    // Execute the "SendTweet" instruction.
    const tweet = anchor.web3.Keypair.generate();
    await program.rpc.sendTweet('veganism', 'Hummus, am I right?', {
        accounts: {
            tweet: tweet.publicKey,
            author: program.provider.wallet.publicKey,
            systemProgram: anchor.web3.SystemProgram.programId,
        },
        signers: [tweet],
    });

    // Fetch the account details of the created tweet.
    const tweetAccount = await program.account.tweet.fetch(tweet.publicKey);

    // Ensure it has the right data.
    assert.equal(tweetAccount.author.toBase58(), program.provider.wallet.publicKey.toBase58());
    assert.equal(tweetAccount.topic, 'veganism');
    assert.equal(tweetAccount.content, 'Hummus, am I right?');
    assert.ok(tweetAccount.timestamp);
});
```

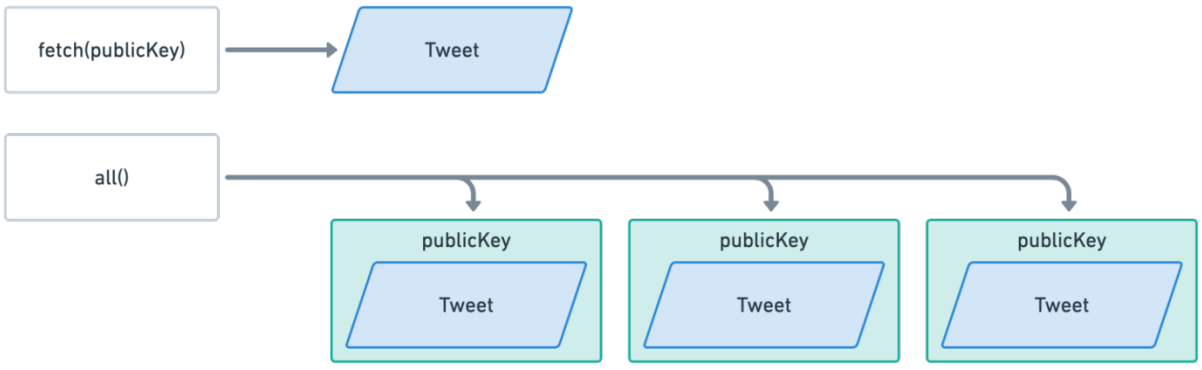
- **Fetching Tweets from the program**
-

Fetching all tweets:

```
it('can fetch all tweets', async () => {
   const tweetAccounts = await program.account.tweet.all();
   assert.equal(tweetAccounts.length, 3);
});
```

- **Filtering tweets by author:**

```
it('can filter tweets by author', async () => {
   const authorPublicKey = program.provider.wallet.publicKey
   const tweetAccounts = await program.account.tweet.all([
      {
         memcmp: {
            offset: 8, // Discriminator.
            bytes: authorPublicKey.toBase58(),
         }
      }
   ]);

   assert.equal(tweetAccounts.length, 2);
   assert.ok(tweetAccounts.every(tweetAccount => {
      return tweetAccount.account.author.toBase58() === authorPublicKey.toBase58()
   }))
});
```



- **Filtering tweets by topic:**

```
it('can filter tweets by topics', async () => {
  const tweetAccounts = await program.account.tweet.all([
    {
      memcmp: {
        offset: 8 + // Discriminator.
          32 + // Author public key.
          8 + // Timestamp.
          4, // Topic string prefix.
        bytes: bs58.encode(Buffer.from('veganism')),
      }
    }
  ]);

  assert.equal(tweetAccounts.length, 2);
  assert.ok(tweetAccounts.every(tweetAccount => {
    return tweetAccount.account.topic === 'veganism'
  }))
});
```

- **All files:**

solana-twitter/app/src/api/delete-tweet.js

```
import { useWorkspace } from '@/composables'

export const deleteTweet = async (tweet) => {
  const { wallet, program } = useWorkspace()
  await program.value.rpc.deleteTweet({
    accounts: {
      author: wallet.value.publicKey,
      tweet: tweet.publicKey,
    },
  })}
```

solana-twitter/app/src/api/get-tweet.js

```
import { useWorkspace } from '@/composables'
```

```
import { Tweet } from '@/models'

export const getTweet = async (publicKey) => {
  const { program } = useWorkspace()
  const account = await program.value.account.tweet.fetch(publicKey);
  return new Tweet(publicKey, account)
}
```

solana-twitter/app/src/api/send-tweet.js

```
import { web3 } from '@project-serum/anchor'
import { useWorkspace } from '@/composables'
import { Tweet } from '@/models'

export const sendTweet = async (topic, content) => {
  const { wallet, program } = useWorkspace()
  const tweet = web3.Keypair.generate()

  await program.value.rpc.sendTweet(topic, content, {
    accounts: {
      author: wallet.value.publicKey,
      tweet: tweet.publicKey,
      systemProgram: web3.SystemProgram.programId,
    },
    signers: [tweet]
  })

  const tweetAccount = await program.value.account.tweet.fetch(tweet.publicKey)
  return new Tweet(tweet.publicKey, tweetAccount)
}
```

solana-twitter/app/src/api/update-tweet.js

```
import { useWorkspace } from '@/composables'

export const updateTweet = async (tweet, topic, content) => {
  const { wallet, program } = useWorkspace()
  await program.value.rpc.updateTweet(topic, content, {
    accounts: {
      author: wallet.value.publicKey,
      tweet: tweet.publicKey,
    },
```

```
    })

    tweet.topic = topic
    tweet.content = content
}
```

solana-twitter/app/src/api/index.js

```
export * from './delete-tweet'
export * from './fetch-tweets'
export * from './get-tweet'
export * from './send-tweet'
export * from './update-tweet'
```

API:

solana-twitter/app/src/api/delete-tweet.js

```
import { useWorkspace } from '@/composables'

export const deleteTweet = async (tweet) => {
    const { wallet, program } = useWorkspace()
    await program.value.rpc.deleteTweet({
        accounts: {
            author: wallet.value.publicKey,
            tweet: tweet.publicKey,
        },
    })
}
```

solana-twitter/app/src/api/get-tweet.js :

```
import { useWorkspace } from '@/composables'
import { Tweet } from '@/models'

export const getTweet = async (publicKey) => {
    const { program } = useWorkspace()
    const account = await program.value.account.tweet.fetch(publicKey);
    return new Tweet(publicKey, account)
}
```

solana-twitter/app/src/api/send-tweet.js

```
import { web3 } from '@project-serum/anchor'
import { useWorkspace } from '@/composables'
import { Tweet } from '@/models'

export const sendTweet = async (topic, content) => {
  const { wallet, program } = useWorkspace()
  const tweet = web3.Keypair.generate()

  await program.value.rpc.sendTweet(topic, content, {
    accounts: {
      author: wallet.value.publicKey,
      tweet: tweet.publicKey,
      systemProgram: web3.SystemProgram.programId,
    },
    signers: [tweet]
  })

  const tweetAccount = await program.value.account.tweet.fetch(tweet.publicKey)
  return new Tweet(tweet.publicKey, tweetAccount)
}
```

solana-twitter/app/src/api/update-tweet.js

```
import { useWorkspace } from '@/composables'

export const updateTweet = async (tweet, topic, content) => {
  const { wallet, program } = useWorkspace()
  await program.value.rpc.updateTweet(topic, content, {
    accounts: {
      author: wallet.value.publicKey,
      tweet: tweet.publicKey,
    },
  })

  tweet.topic = topic
  tweet.content = content
}
```

solana-twitter/app/src/api/index.js

```
export * from './delete-tweet'
export * from './fetch-tweets'
export * from './get-tweet'
export * from './send-tweet'
export * from './update-tweet'
```

- **MODELS:**

- **IDL FILE: (Interface Description Language)**

solana-twitter/app/src/idl/solana_twitter.json

```
{
 "version": "0.1.0",
 "name": "solana_twitter",
 "instructions": [
  {
   "name": "sendTweet",
   "accounts": [
    {
     "name": "tweet",
     "isMut": true,
     "isSigner": true
    },
    {
     "name": "author",
     "isMut": true,
     "isSigner": true
    },
    {
     "name": "systemProgram",
     "isMut": false,
     "isSigner": false
    }
   ],
   "args": [
    {
     "name": "topic",
     "type": "string"
```

```
    },
    {
     "name": "content",
     "type": "string"
    }
   ]
},
{
 "name": "updateTweet",
 "accounts": [
    {
     "name": "tweet",
     "isMut": true,
     "isSigner": false
    },
    {
     "name": "author",
     "isMut": false,
     "isSigner": true
    }
 ],
 "args": [
    {
     "name": "topic",
     "type": "string"
    },
    {
     "name": "content",
     "type": "string"
    }
 ]
},
{
 "name": "deleteTweet",
 "accounts": [
    {
     "name": "tweet",
     "isMut": true,
     "isSigner": false
    },
```

```
      {
       "name": "author",
       "isMut": false,
       "isSigner": true
      }
    ],
    "args": []
  }
],
"accounts": [
  {
    "name": "Tweet",
    "type": {
      "kind": "struct",
      "fields": [
        {
          "name": "author",
          "type": "publicKey"
        },
        {
          "name": "timestamp",
          "type": "i64"
        },
        {
          "name": "topic",
          "type": "string"   },
        {
          "name": "content",
          "type": "string"
        }
      ]
    }
  }
],
"errors": [
  {
    "code": 6000,
    "name": "TopicTooLong",
    "msg": "The provided topic should be 50 characters long maximum."
  },
```

```
    {
      "code": 6001,
      "name": "ContentTooLong",
      "msg": "The provided content should be 280 characters long maximum."
    }
  ],
  "metadata": {
    "address": "BNDCEb5uXCuWDxJW9BGmbfvR1JBMAKckfhYrEKW2Bv1W"
  }
}
```

Anchor.toml:

```
[programs.localnet]
solana_twitter = "BNDCEb5uXCuWDxJW9BGmbfvR1JBMAKckfhYrEKW2Bv1W"
[programs.devnet]
solana_twitter = "BNDCEb5uXCuWDxJW9BGmbfvR1JBMAKckfhYrEKW2Bv1W"
[programs.mainnet]
solana_twitter = "BNDCEb5uXCuWDxJW9BGmbfvR1JBMAKckfhYrEKW2Bv1W"
[registry]
url = "https://anchor.projectserum.com"
[provider]
cluster = "localnet"
wallet = "/Users/loris/.config/solana/id.json"
[scripts]
test = "yarn run ts-mocha -p ./tsconfig.json -t 1000000 tests/**/*.ts"
copy-idl = "mkdir -p app/src/idl && cp target/idl/solana_twitter.json
app/src/idl/solana_twitter.json"
```

6. **Results**

- **Setting up the local test ledger (validator):**

```
rohit@pph-vizag-00177:~/solana-twitter$ solana-test-validator
--faucet-sol argument ignored, ledger already exists
Ledger location: test-ledger
Log: test-ledger/validator.log
. Initializing...
Identity: A8rTVM2LsBXCFqjWEdjhzwfEKNm53kSnYN6NdKXRVtRZ
Genesis Hash: HH2kw1a2h2rszfjr6j3nxaKeiSytcyxxZjwkFNyCZmPa
Version: 1.9.4
Shred Version: 44187
Gossip Address: 127.0.0.1:1024
TPU Address: 127.0.0.1:1027
JSON RPC URL: http://127.0.0.1:8899
.. 00:05:28 | Processed Slot: 52960 | Confirmed Slot: 52960 | Finalized Slot: 52928 | Full Snapshot Slot: 52902 |
```

- **Building the program:**

```
rohit@pph-vizag-00177:~/solana-twitter$ anchor build
BPF SDK: /home/rohit/.local/share/solana/install/releases/1.9.4/solana-release/bin/sdk/bpf
cargo-build-bpf child: rustup toolchain list -v
cargo-build-bpf child: cargo +bpf build --target bpfel-unknown-unknown --release
    Finished release [optimized] target(s) in 0.36s
cargo-build-bpf child: /home/rohit/.local/share/solana/install/releases/1.9.4/solana-release/bin/sdk/bpf/dependen
cies/bpf-tools/llvm/bin/llvm-readelf --dyn-symbols /home/rohit/solana-twitter/target/deploy/solana_twitter.so

To deploy this program:
  $ solana program deploy /home/rohit/solana-twitter/target/deploy/solana_twitter.so
The program address will default to this keypair (override with --program-id):
  /home/rohit/solana-twitter/target/deploy/solana_twitter-keypair.json
Error:
        /home/rohit/solana-twitter/programs/solana-twitter/src/lib.rs:59:8
        Struct field "system_program" is unsafe, but is not documented.
        Please add a `/// CHECK:` doc comment explaining why no checks through types are necessary.
        See https://book.anchor-lang.com/anchor_in_depth/the_accounts_struct.html#safety-checks for more informat
ion.
```

- **Deploying the program:**

```
rohit@pph-vizag-00177:~/solana-twitter$ solana program deploy /home/rohit/solana-twitter/target/deploy/solana_twi
tter.so
Program Id: 4NsRMBzfBhcZ5KFtmWBqdvQCwFU1tvLn4hkvEQQaCAi8
```

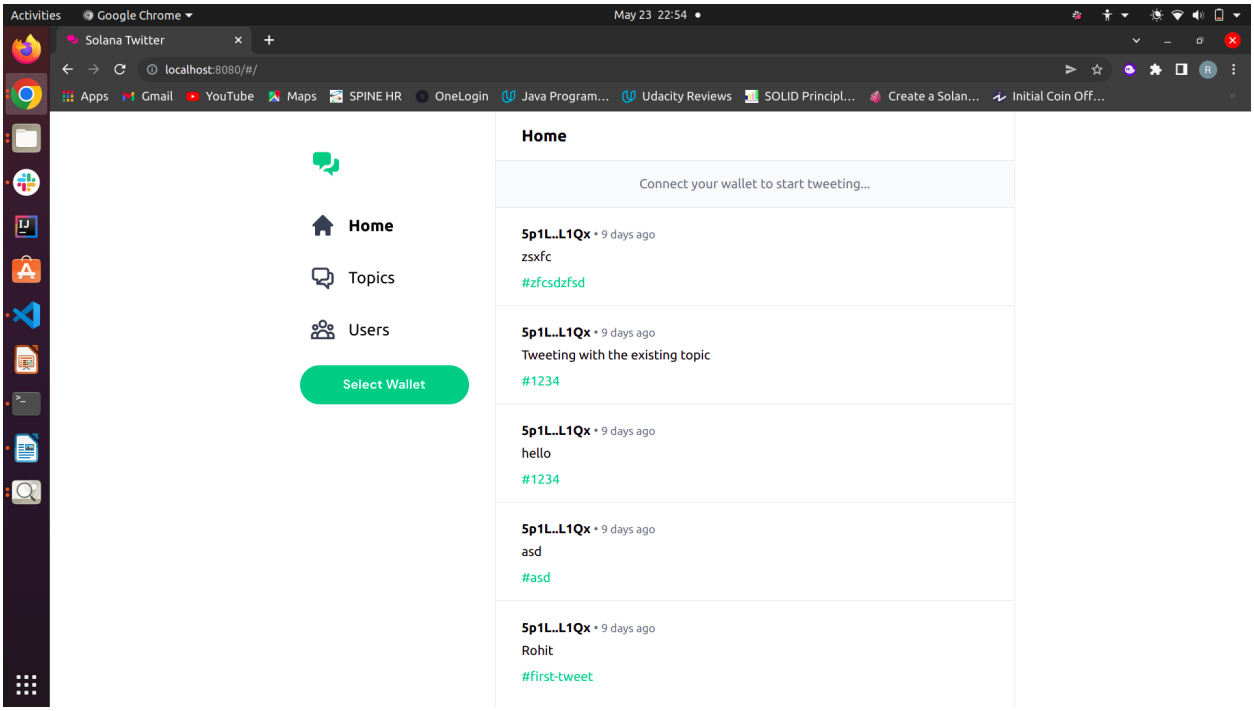
- **Starting the client:**

```
rohit@pph-vizag-00177:~/solana-twitter/app$ npm run serve

> app@0.1.0 serve
> vue-cli-service serve

Browserslist: caniuse-lite is outdated. Please run:
  npx browserslist@latest --update-db
  Why you should do it regularly: https://github.com/browserslist/browserslist#browsers-data-updating
 INFO  Starting development server...


 DONE  Compiled successfully in 9051ms                                        19:52:53


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.107:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

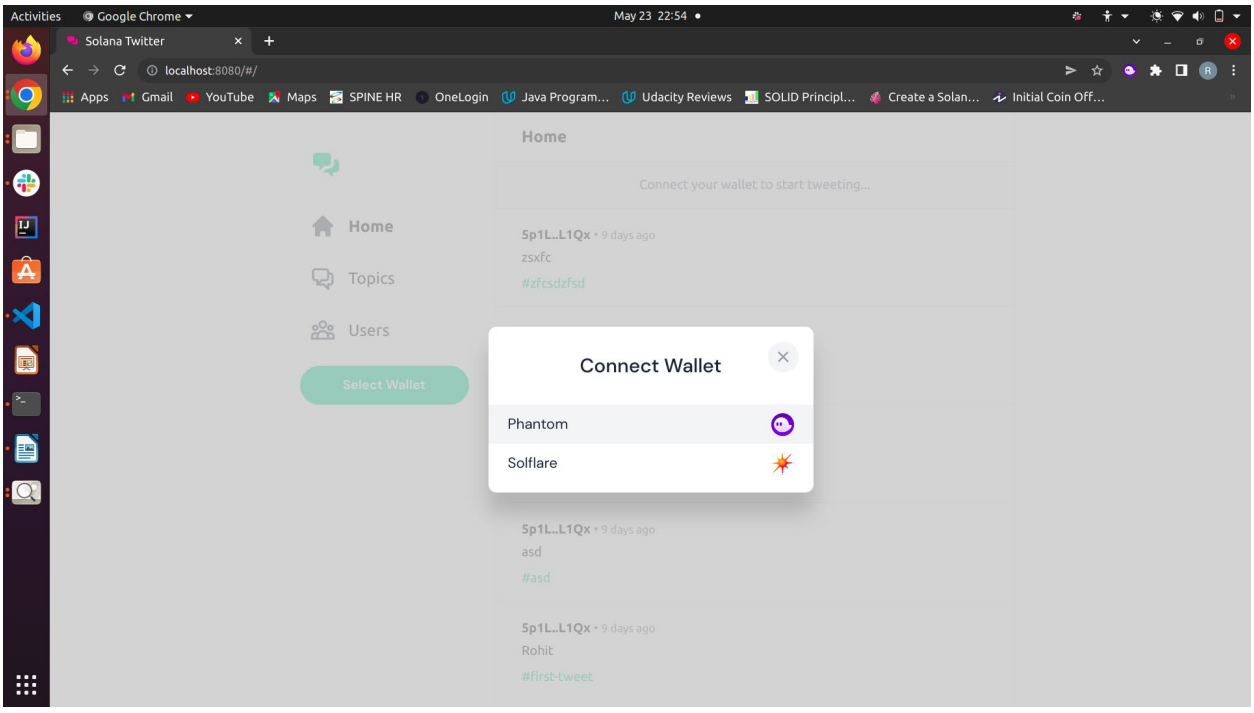
- **Airdropping SOL into the wallet:**

```
rohit@pph-vizag-00177: ~/solana-twitter
rohit@pph-vizag-00177:~/solana-twitter$ solana airdrop 100 5p1LC8Ld2cznJHFcXco78
TucjGsf1dBfoLCRhxAwL1Qx
Requesting airdrop of 100 SOL

Signature: yAy2ZoGewYhenaq8dGXKpdZ2kCwJaSxiVw8w6bCKjQxFoQDamBfADAjXsxgGM67fmHD8e
AD4gUyTLGKVwpsnpZb

199.91614228 SOL
rohit@pph-vizag-00177:~/solana-twitter$
```

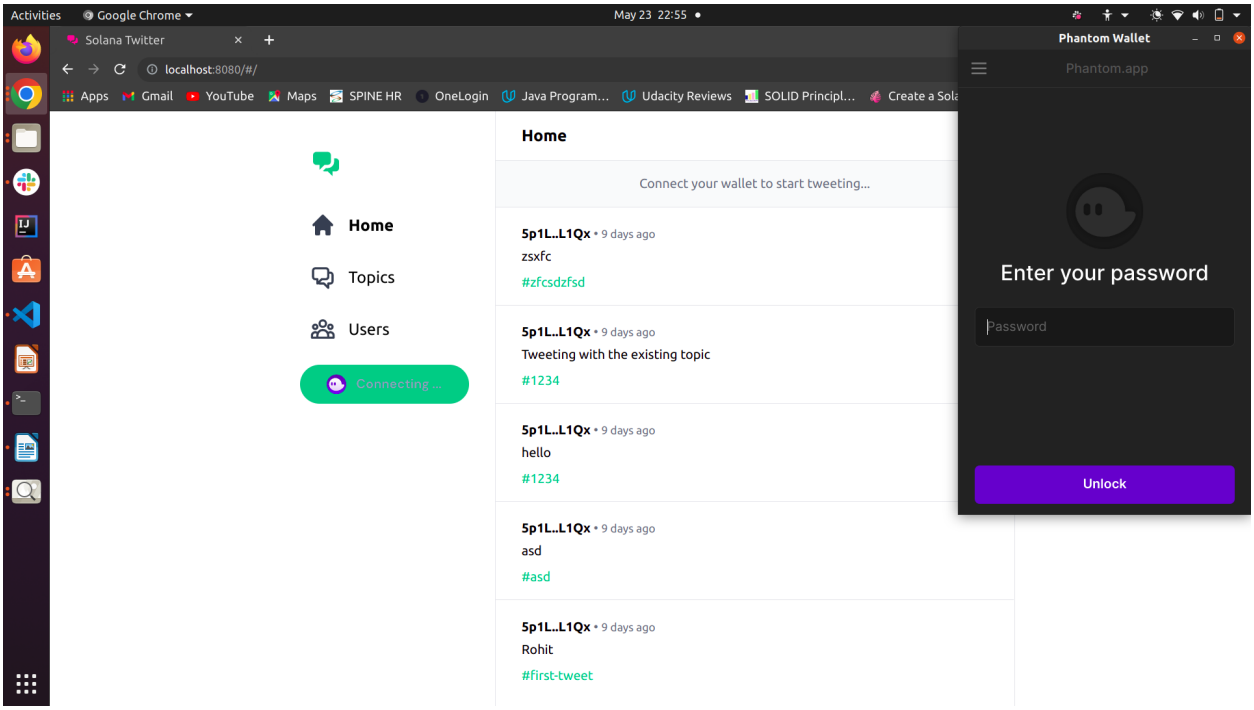
- **Landing Page without wallet connection:**



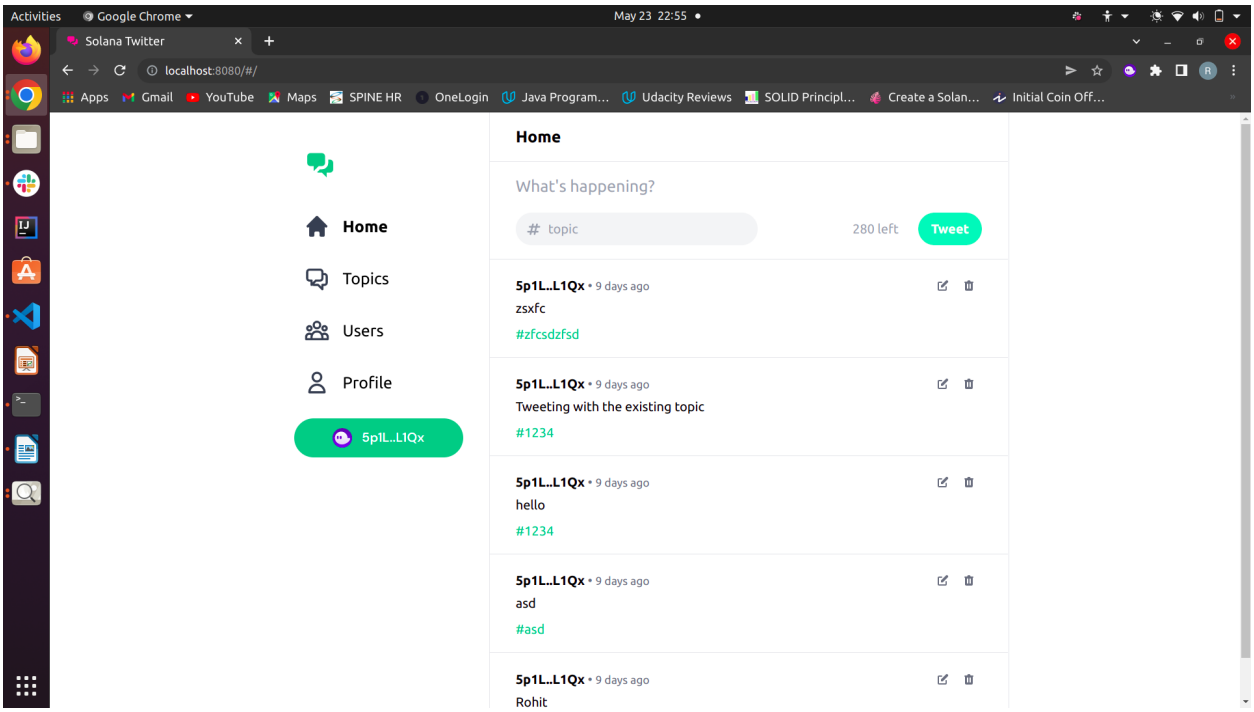
- **Choose Wallet from wallet select menu**



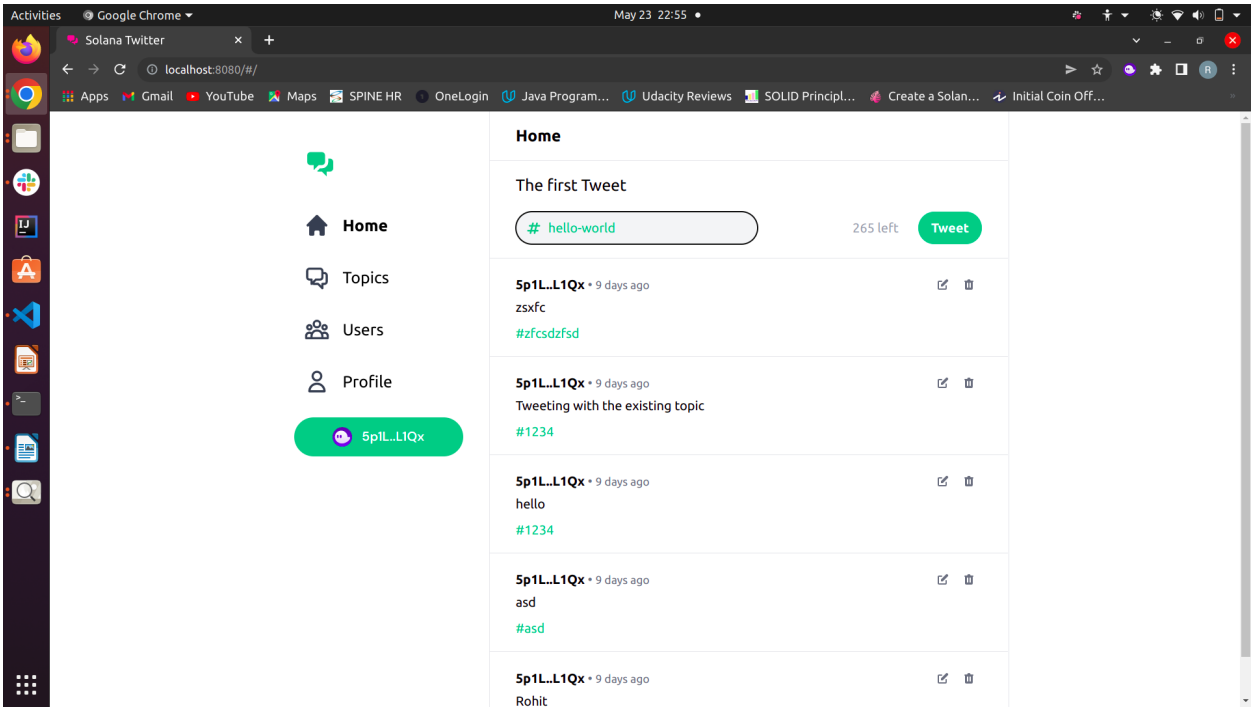
- **Unlock your wallet using your password**



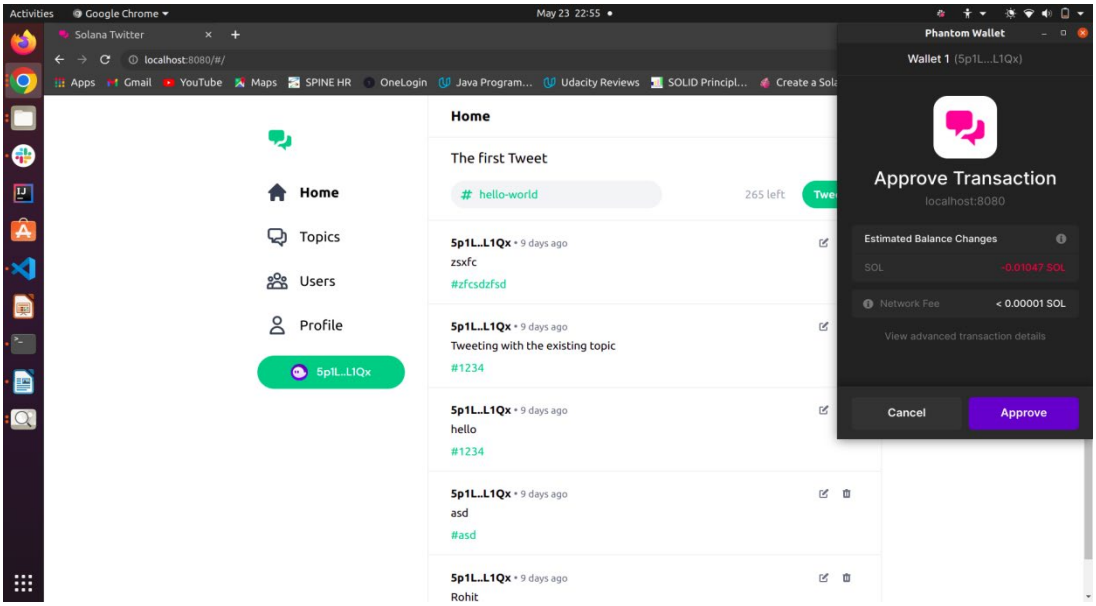
- **The landing page with the wallet connected**



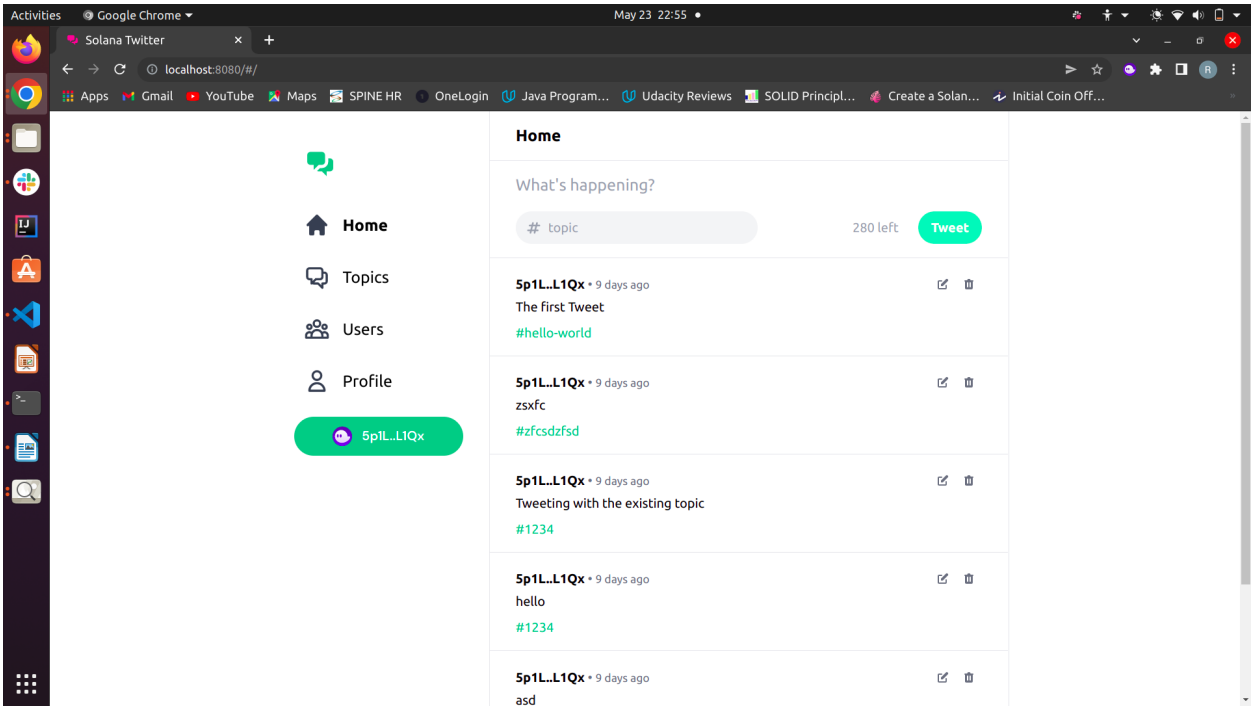
- **Entering a tweet with content and topic**



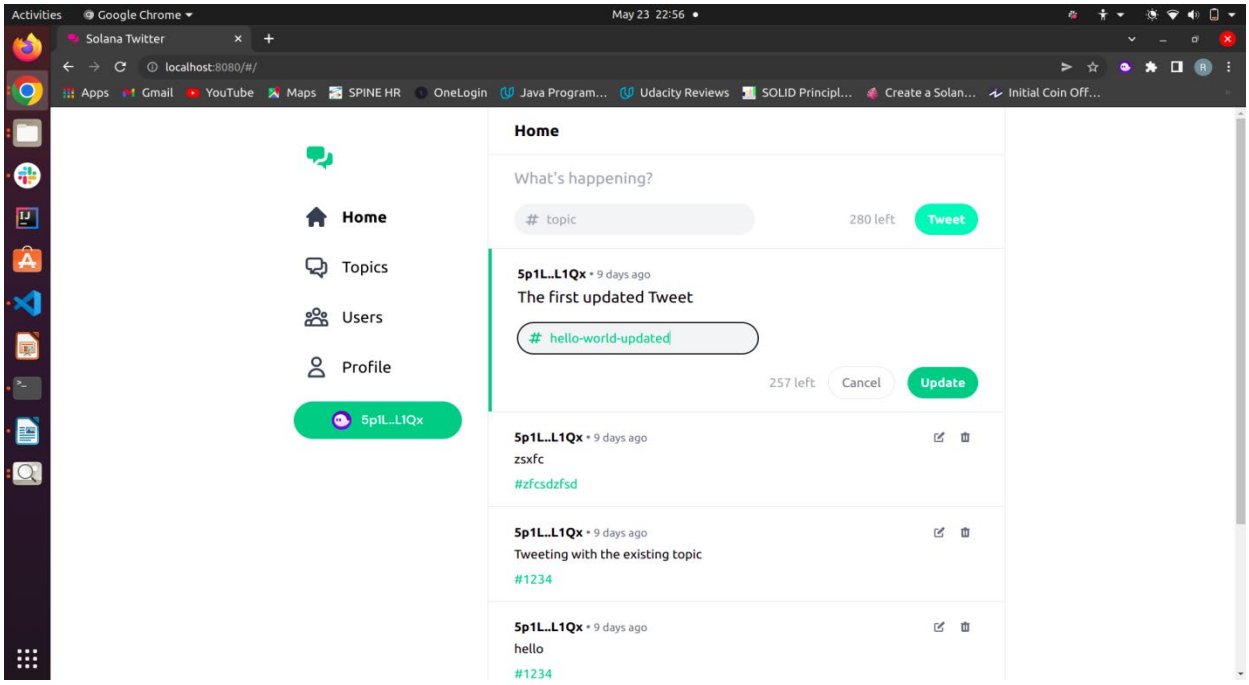
- **Approval for transaction of creating a new account on the solana blockchain. Some SOL is deducted as rent for adding a new account to the blockchain.**



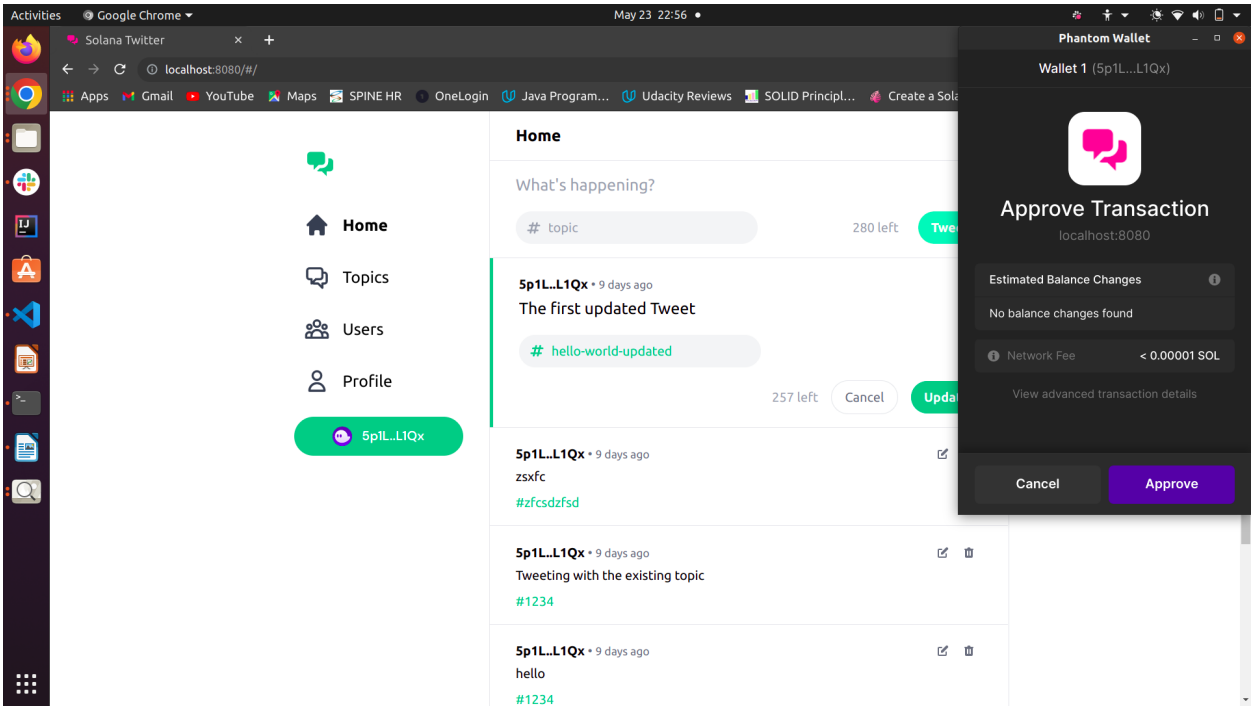
- **The tweet is added to the blockchain**



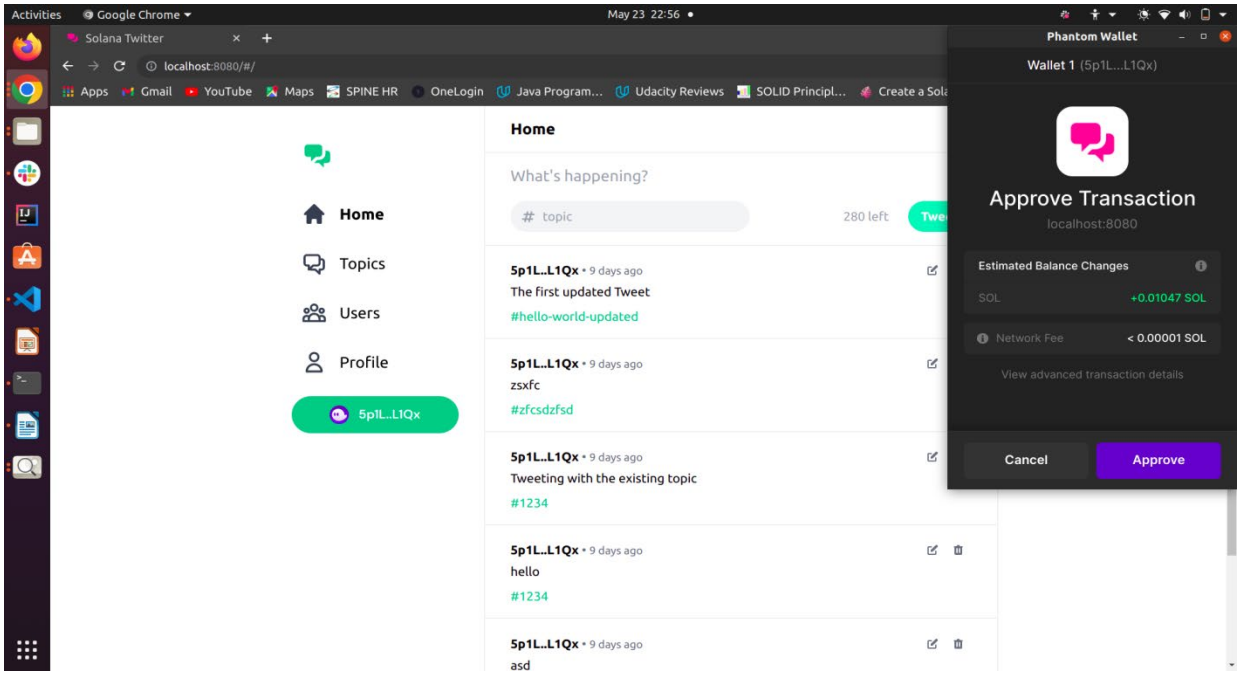
- **Updating the tweets that are sent only from the user's wallet**



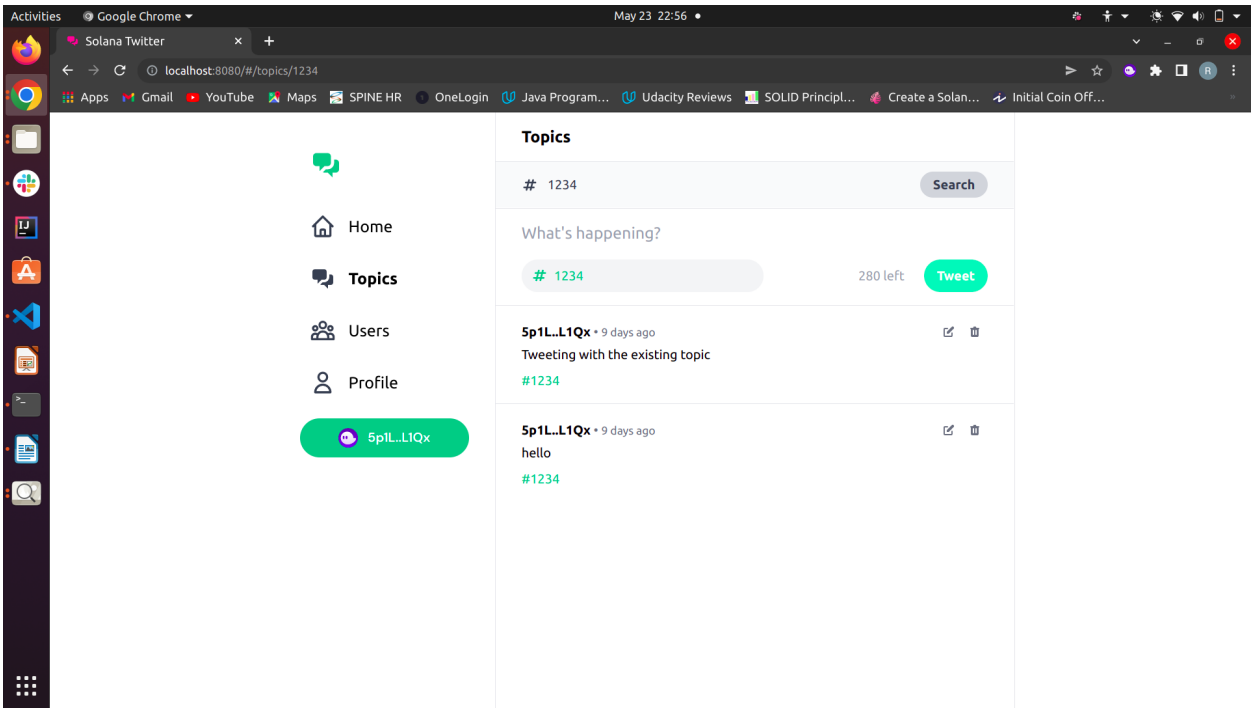
- **No charges are required for updating an existing account on the blockchain.**



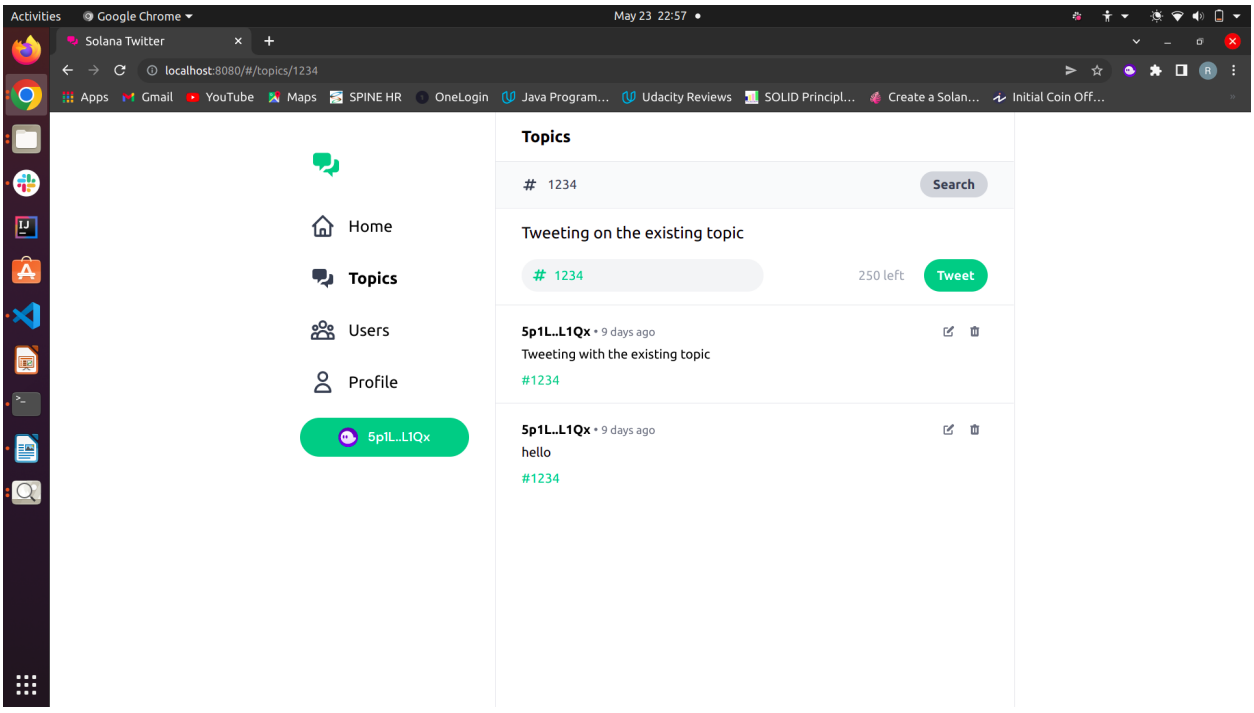
- **Deleting an account on the blockchain.**



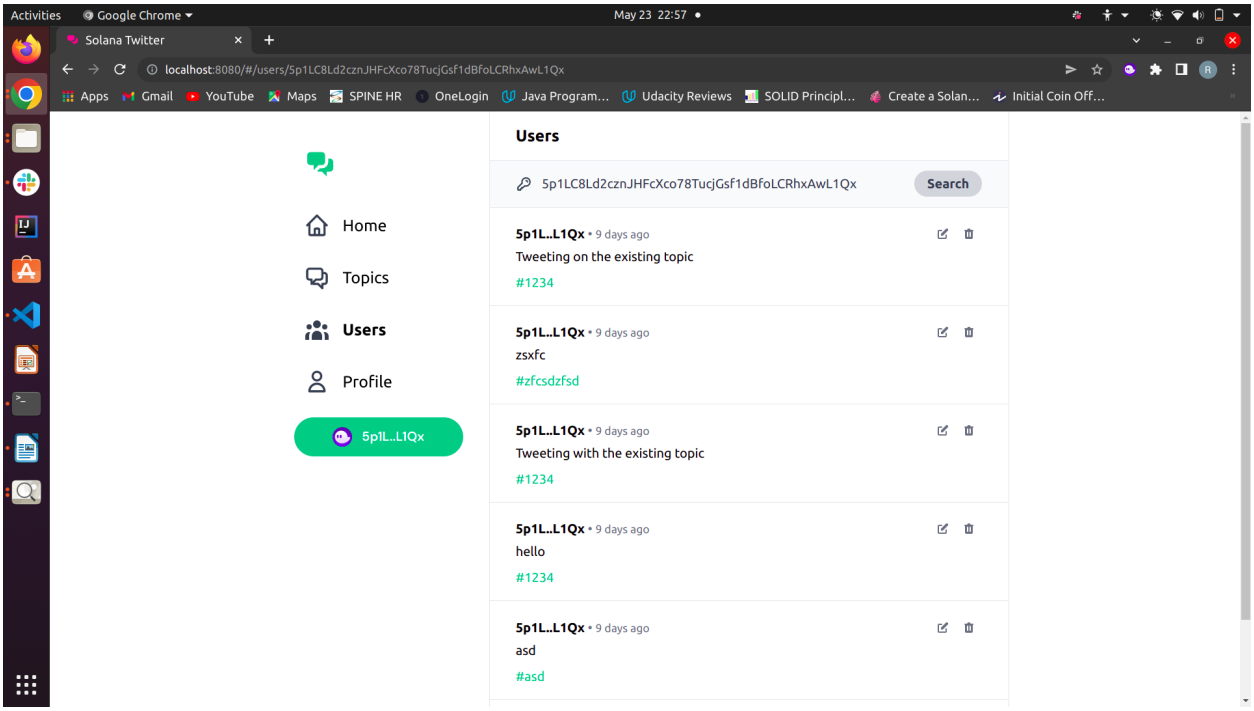
- **The topic page facilitates the users to filter the tweets by tweet topic**



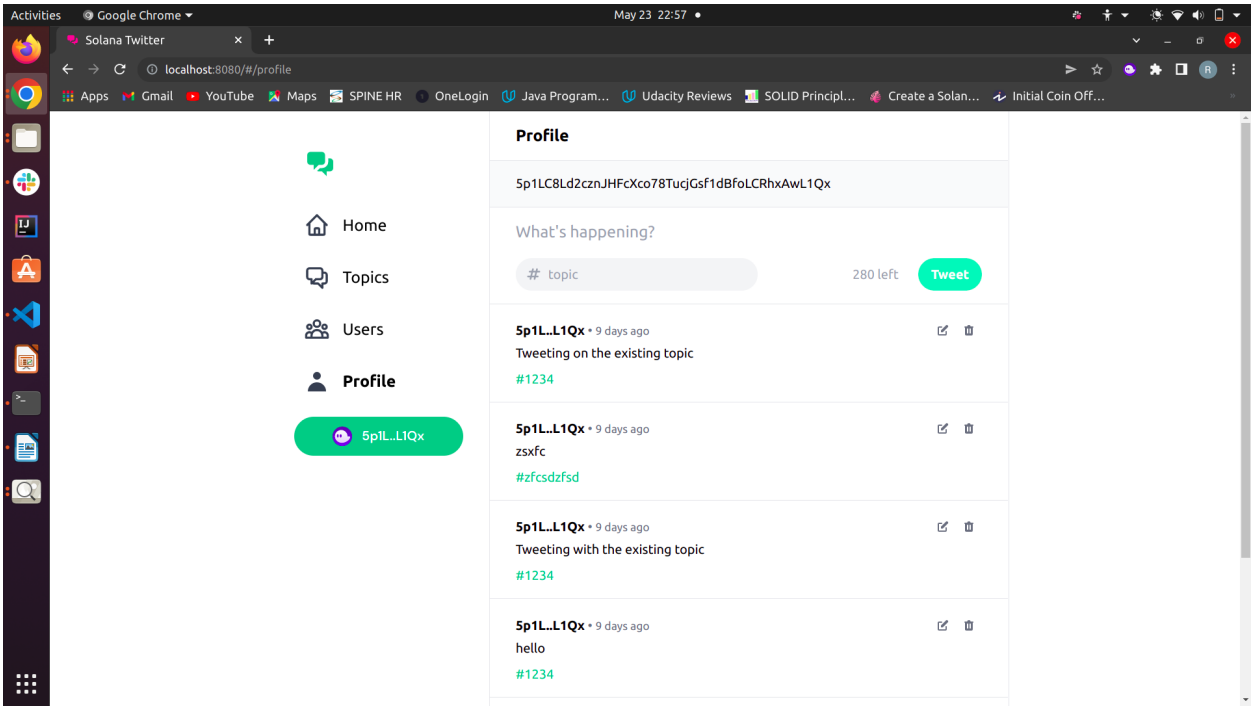
- **Tweeting on the existing topic**



- **Filtering tweets through users**



- **Show all tweets sent from the wallet of the user**



7. Conclusion

NFTs and the Metaverse are the trending topics in the cryptocurrency ecosystem right now, but the next big thing might just be decentralized social media**.** Like decentralized finance, decentralized social media platforms don't have a centralized governing body and may, someday, provide viable alternatives to established platforms like Twitter, Instagram, Facebook and TikTok. The technology is currently evolving just beyond the embryonic stage of development. Unlike decentralized social media, the centralized social media industry is plagued by global censorship, a lack of customization, unfair monetization, algorithm dictatorship and a monopoly on network effects which makes decentralized social media to be trended. The decentralized social media is really about "giving the power back to the people."

One of the main technologies used in this project is blockchain technology. A blockchain is a decentralized ledger of all transactions across a peer-to-peer network. Using this technology, participants can confirm transactions without a need for a central clearing authority.The programming language mainly used in this project is Rust. Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling, an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.This project could be enchanced to other social netwoks in the future for a feasible usability of other application.

REFERENCES

- https://en.wikipedia.org/wiki/InterPlanetary
- https://blog-b0c8db.webflow.io/web3/intro
- https://sopa.tulane.edu/blog/decentralized-social-networks#:~:text=Decentralized%20social%20networks%20operate%20on,runr%20on%20a%20social%20blockchain.
- https://docs.deso.org/about-deso-chain
- https://academy.moralis.io/blog/blockchain-social-media-platforms-benefits-of-decentralized-social-media
- https://www.investopedia.com/terms/b/blockchain.asp
- https://youtu.be/pWqbX372vrc
- https://youtu.be/AwbjalCp5zc