.e-Ballot

E-Voting System based on Encrypted Blockchain Technology

Submitted as a project in Hackathon: **Code Fellas**

Submitted on **December 15th, 2021**, by

Team **BlackFlag**

Mayank Patel | 19/BCS/057

Kartikey Pandey | 19/BCS/049

Divyanshu Mishra | 19/BCS/045

E-Voting System based on Encrypted Blockchain Technology

Overview:

In a bid to address digital interfaces, this E-voting application serves as a solution for anyone who is unable to reach the polling booth or prefers to cast his or her vote remotely. The greater purpose of the application is that it makes use of blockchain technology - hence this system is invulnerable to fraud and cyberattacks as the blocks of data cannot be tampered with. Blockchain makes it more secure, crucial in an electoral arena, where there could be a tendency to manipulate voting. The application primarily aims at increasing trust, efficiency, and security within the voting system of any kind.

Hardware Requirements:

	MINIMUM	USED
CPU	Any 1GHz or faster dual-core 64bit processor from Intel/AMD	Intel i5-1135G7 at 2.4GHz
RAM	4GB of RAM	16GB
STORAGE	512 MB	1 TB SSD
CONNECTIVITY	constant broadband	constant broadband

Software requirements:

	MINIMUM	USED
OS	Windows 10, 11/any Linux based OS	Windows 11
SOFTWARE	Visual Studio Code, Sublime Text or any IDE, Any Web Browser	Visual Studio Code v.1.63.0, Mozilla Firefox

Technologies Used:

This **React** application was initialized with **create-near-app**.

React.js, **Node.js**, **NEAR Blockchain Network** – which runs in concert with **Ethereum**, **Polkadot**, **Cosmos**.

Languages used:

JavaScript, TypeScript, HTML, CSS

Project Description:

The project aims at utilizing the **blockchain technology** as it plays a key role in the domain of electronic voting due to inherent nature of **preserving anonymity**, maintaining **decentralized** and **publicly distributed ledger of transactions** across all the nodes.

In this project, we explore the use of blockchain to facilitate e-voting applications with the ability to **assure voter anonymity**, **vote integrity**, and **end-to-end verification** through hashes. The application is written in **JavaScript**, **Typescript** and uses **React.js** for **frontend** development. We have made a **smart contract** which is deployed on the **NEAR Blockchain** which runs when the predetermined conditions are met. The smart contract is used to automate the execution of the agreement so that all voters can be immediately certain of the outcome, without any intermediary's involvement or time loss.

As a voter is registered into the system, a **voter hash** is generated by blockchain which is the **unique identifier of a voter** into the blockchain and is **protected from misuse due to collision resistance property** of the cryptographic hash. Upon casting their vote successfully, a **user is provided with their unique transaction ID** in the form of a cryptographic hash. A user can use this transaction ID to **track if their vote was included in the tallying process**. However, this process does not enable a user to view how they voted which has been adopted to mitigate threats when under duress. This **blockchain is then updated** when the transaction (here – vote casted) is completed. That means the **transaction cannot be changed**, and only parties who have granted permission can see the results.

Scope of the Application:

Features:

1. The application keeps an individual's vote secret.

2. It allows only registered voters to register only once.

3. The application is portable hence it allows them to vote easily and from anywhere.

4. Since the application is built on blockchain, the user can trust the vote tallying process.

Functions:

1. The system can be used to make the election process transparent and reliable.

2. Blockchain is a technology with strong cryptographic foundations enabling the application to leverage these abilities.

3. It is a distributed decentralized database that maintains complete data records secured from unauthorized manipulating, tampering and revision.

Challenges Faced:

1. Updated modules weren't available readily for the application.

2. The application was having compatibility issues on different web browsers.

3. The access time for NEAR blockchain was very slow which caused a hurdle in the development process.

4. All the problems were resolved as we progressed through the project.

Conclusion:

We present a brief description of each requirement for a typical e-voting system and how our system fulfils it:

1. Privacy – The system leverages cryptographic properties of blockchain to achieve privacy of a voter.

2. Eligibility – The system allows only registered voters to vote, with each such voter voting only once.

3. Receipt Freeness – The system enables a voter to vote as per their choice and creates a cryptographic hash for each such event. This is important to verify if a certain vote ws included in the count.

4. Convenience – The system has been implemented using a user-friendly web-based interface with the voting process requiring minimal input from the user.

5. Verifiability - Upon casting their vote successfully, a user is provided with their unique transaction ID in the form of a cryptographic hash. A user can use this transaction ID to track if their vote was included in the tallying process.

Steps to run this project locally:

Prerequisites:

1. Make sure you've installed Node.js ≥ 12

2. Install dependencies: `npm install`

3. Run the local development server: `npm run dev` (see `package.json` for a full list of scripts you can run with yarn)

Exploring The Code:

1. The **backend** code lives in the `/contract` folder.

2. The **frontend** code lives in the `/src` folder. `/src/index.html` loads in `/src/index.js`, where you can learn how the frontend connects to the **NEAR** blockchain.

Deploy:

Every smart contract in NEAR has its own associated account. When you run `npm run dev`, your smart contract gets deployed to the live NEAR TestNet with a throwaway account. When you're ready to make it permanent, here's how.

Step 1: Create an account for the contract

Each account on NEAR can have at most one contract deployed to it. If you've already created an account such as `your-name.testnet`, you can deploy your contract to `evoting.your-name.testnet`. Assuming you've already created an account on NEAR Wallet, here's how to create `evoting.your-name.testnet`:

1. Authorize NEAR CLI, following the commands it gives you: near login

2. Create a subaccount (replace `YOUR-NAME` below with your actual account name):

```
near create-account evoting.YOUR-NAME.testnet --masterAccount YOUR-NAME.testnet
```

Step 2: Set contract name in code

Modify the line in `src/config.js` that sets the account name of the contract. Set it to the account id you used above.

```
const CONTRACT_NAME = process.env.CONTRACT_NAME || 'evoting.YOUR-NAME.testnet'
```

Step 3: Deploy!

```
npm deploy
```

1. This does two things: builds & deploys smart contract to NEAR TestNet

2. Builds & deploys frontend code to GitHub using gh-pages. This will only work if the project already has a repository set up on GitHub.
