

Table of contents

- About the project
 - Problem
 - Solution
- Workflow
- Application
 - Prerequisite
 - Installation
 - How to run the application according to the workflow
- Chaincode functions
- Resources and Technologies Used

About the Project

Problem:

In a democratic system, ensuring transparent, secure, and tamper-proof elections is critical. Traditional voting methods, whether paper-based or electronic, face challenges such as voter fraud, tampering, and a lack of transparency. The problem is further compounded by the difficulty of maintaining voter confidentiality while ensuring that only eligible voters participate and their votes are accurately counted.

Key Challenges:

1. **Voter Registration Fraud:** Verifying voter eligibility and maintaining an up-to-date and secure voter list.
2. **Vote Tampering:** Ensuring that votes are not altered or miscounted during transmission or tabulation.
3. **Transparency vs. Privacy:** Balancing the need for election transparency with voter anonymity.
4. **Lack of Trust:** Limited public trust in election processes due to historical malpractices and inefficiencies.
5. **Auditing and Accountability:** Ensuring secure auditing of votes while maintaining voter confidentiality.

Solution:

The proposed Blockchain-Based Voting System addresses these challenges using Hyperledger Fabric, a permissioned blockchain framework. The system involves collaboration between:

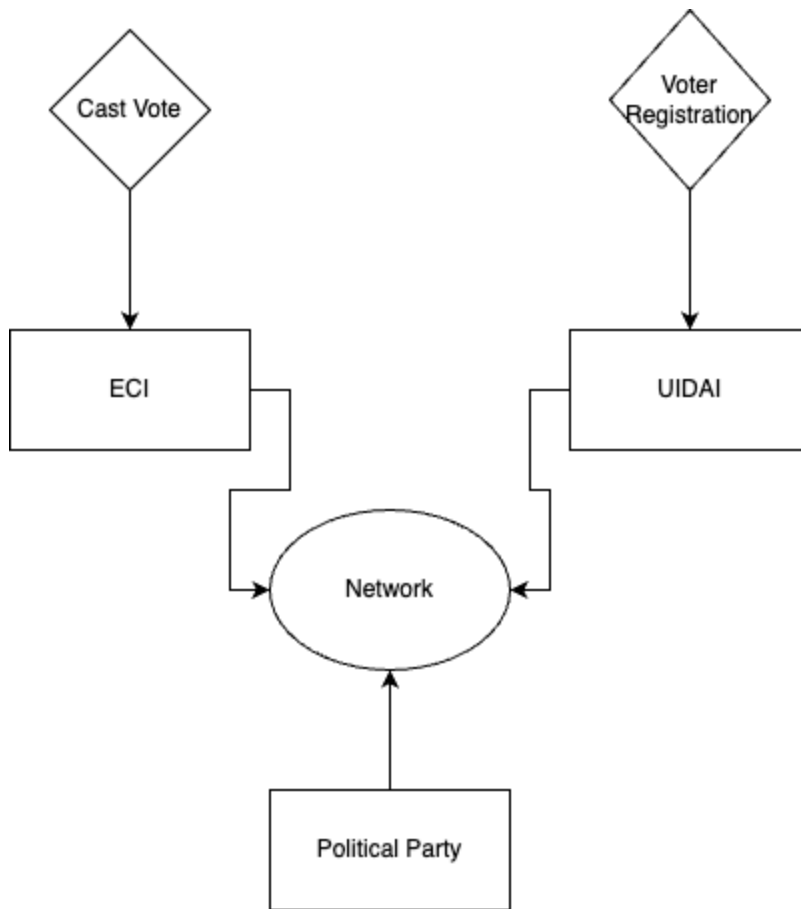
1. Election Commission of India (ECI) – Responsible for overseeing the election process and managing vote casting.
2. Unique Identification Authority of India (UIDAI) – Responsible for securely verifying voter identities and registering eligible voters.
3. Political Parties – Observers of vote aggregation and results verification.

Features of the Solution:

1. Voter Registration: Managed by UIDAI using the Aadhaar database to verify and register eligible voters securely.
2. Vote Casting: Managed by ECI through a secure and transparent blockchain-enabled voting platform.
3. Immutable Ledger: Blockchain ensures that all transactions (voter registration, vote casting) are tamper-proof and auditable.
4. Anonymity: Votes are stored using cryptographic methods to ensure voter privacy.
5. Real-Time Monitoring: Political parties and auditors can monitor aggregated voting data and audit the system in real-time.

Workflow:

1. Voter Creation by UIDAI
2. Caste Vote based on election by ECI
3. Party would be the observer



Running the Application:

Prerequisite:

Docker, docker-compose and fabric binaries should be present and installed on the system.

Installation:

Clone the repo:

git clone https://github.com/KBA-Learning/KBA_CHE_NPCI_B7.git

Install the binaries and dependencies and follow the instructions mentioned in readme file

OUTPUTs

1. Add Voter:

The screenshot displays a REST client interface with a list of HTTP methods at the top: POST Add, POST Cas, GET Get A, DEL Delete, GET Vote I, GET Vote I, [CONFLI], DEL DELETE, GET Get A, and GET Voter. The 'Add Voter' endpoint is selected, showing a POST request to localhost:8080/api/voter. The request body is a JSON object with the following fields: name (Maneesh), voterId (voter-08), aadharId (XXXXXXX), state (UP), and district (gkp). The response is a 200 OK status with a response time of 2.06 s and a body size of 233 B. The response body is shown in the 'Body' tab, displaying a JSON object with the same fields as the request, plus an additional 'status' field set to 'Enabled'.

```
1 {
2   "name": "Maneesh",
3   "voterId": "voter-08",
4   "aadharId": "XXXXXXX",
5   "state": "UP",
6   "district": "gkp"
7 }
```

```
1 {
2   "voterId": "voter-08",
3   "name": "Maneesh",
4   "aadharId": "XXXXXXX",
5   "state": "UP",
6   "district": "gkp",
7   "status": "Enabled"
8 }
```

2. Get Voter

Import

POST Add

POST Cas

GET Get A

DEL Deletr

GET Vote I

GET Vote I

CONFLIK

DEL DELE

GET Get A

GET Voter

local

Election / GET Voter

Save

Share

GET

localhost:8080/api/voter/-voterid

Send

Params

Authorization

Headers (6)

Body

Scripts

Tests

Settings

Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Path Variables

Key	Value	Description	Bulk Edit
voterid	voter-01	Description	

Body

Cookies

Headers (3)

Test Results

200 OK

12 ms

311 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "data": "{\n  \"voterId\": \"voter-01\",\n  \"name\": \"Maneesh\",\n  \"aadharId\": \"XXXXXXXXXX\",\n  \"state\": \"UP\",\n  \"district\": \"Gorakhpur\",\n  \"status\": \"enabled\"\n}"
3 }
```

3. Delete Voter

HTTP Client interface showing a DELETE request to `localhost:8080/api/voter/:voterid`. The response is `200 OK` with a body: `"data": "*** Transaction submitted successfully: \n"`.

DELETE `localhost:8080/api/voter/:voterid` Send

Params • Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Path Variables

Key	Value	Description	Bulk Edit
voterid	voter-08	Description	

Body Cookies Headers (3) Test Results ⌚

200 OK • 2.08 s • 176 B • Save Response •••

Pretty Raw Preview Visualize JSON ⌵ ⌵

```
1
2 "data": "*** Transaction submitted successfully: \n"
3
```

4. Get All voters

Postman interface showing a REST client request for `GET /api/voters` on `localhost:8080`. The request is successful, returning a `200 OK` status with a response time of 19 ms and a body size of 461 B.

The response body is displayed in JSON format, showing two voter records:

```
{
  "voterId": "001",
  "name": "Maneesh",
  "aadharId": "XXXXXXx",
  "state": "UP",
  "district": "GKP",
  "status": "enabled"
},
{
  "voterId": "voter-01",
  "name": "Maneesh",
  "aadharId": "XXXXXXXXXXXX",
  "state": "UP",
  "district": "Gorakhpur",
  "status": "enabled"
}
```

6. Get Voter By range

Election / Voter By Range

GET

localhost:8080/api/votersRange

Send

Params

Authorization

Headers (8)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"startKey": "001",

3

"endKey": "voter-02"

4

}

Body

Cookies

Headers (3)

Test Results

200 OK

10 ms

351 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

2

{

3

"voterId": "001",

4

"name": "Maneesh",

5

"aadharId": "XXXXXXx",

6

"state": "UP",

7

"district": "GKP",

8

"status": "enabled"

9

},

10

{

11

"voterId": "voter-01",

12

"name": "Maneesh",

13

"aadharId": "XXXXXXXXXXx",

14

"state": "UP",

15

"district": "Gorakhpur",

16

"status": "enabled"

17

}

Postbot

^

⌘

P

7. Caste Vote

HTTP Election / Cast Vote

POST localhost:8080/vote

Send

Params Authorization Headers (8) Body Scripts Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "voteId": "vote-01",
3   "voterId": "voter-02",
4   "electionId": "e-01",
5   "party": "BJP"
6 }
```

Body Cookies Headers (3) Test Results

200 OK • 2.06 s • 174 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 "*** Transaction submitted successfully: success\n"
```

8. Get all votes

Election / Get All Votes

Save

GETlocalhost:8080/votes

Send

ParamsAuthorizationHeaders (6)BodyScriptsTestsSettings

Cookies

Query Params

	Key	Value	Description		Bulk Edit
	Key	Value	Description		

BodyCookiesHeaders (3)Test Results

200 OK14 ms192 BSave Response

PrettyRawPreviewVisualizeJSON

```
1 [
2   {
3     "voteId": "vote-01",
4     "voterId": "",
5     "electionId": "e-01",
6     "party": "BJP"
7   }
8 ]
```

Postbot

9. Delete Vote

HTTP Election / Delete Vote

Save

Share

DELETE

localhost:8080/vote/vh01

Send

Params

Authorization

Headers (6)

Body

Scripts

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (3)

Test Results

200 OK • 2.08 s • 174 B • Save Response •

Pretty

Raw

Preview

Visualize

JSON

1 "*** Transaction submitted successfully: Success\n"

10. Vote by Range

The screenshot shows a REST client interface with the following details:

- URL:** localhost:8080/vote
- Method:** GET
- Body (raw):**

```
{  "startKey": "vote-01",  "endKey": "vote-05"}
```
- Response:** 200 OK, 15 ms, 192 B. The response body is a JSON array:

```
[  {    "voteId": "vote-01",    "voterId": "",    "electionId": "e-01",    "party": "BJP"  }]
```

Chaincode functions:

1. **CastVote:** Used to cast the vote
2. **RevokeVote:** to revoke the vote
3. **GetAllVotes:** to get all available votes
4. **GetVotesByRange:** to get all votes in a range via start and end key
5. **GetVoteHistory:** to history of the votes
6. **GetVoteWithPagination:** get all votes based on the pagesize and bookmark
7. **AddVoter:** To add new voter into the system
8. **DeleteVoter:** to delete the voter from the system
9. **GetVoter:** get the details of the voter
10. **GetAllVoters:** to get all available voter
11. **GetVotersByRange:** get all voter into the range

Resources and Technologies Used:

1. Hyperledger Fabric
2. Go Lang
3. Grpc
4. Gin
5. Postman
6. Git and github