Week 11 :   Analysis and Design:

Data model/data set @ page 2 & 3

https://www.scirp.org/journal/paperinformation?paperid=118849#f5

Class diagram @ page 4 , 5 & 6

https://www.freeprojectz.com/entity-relationship/voting-management-system-er-diagram

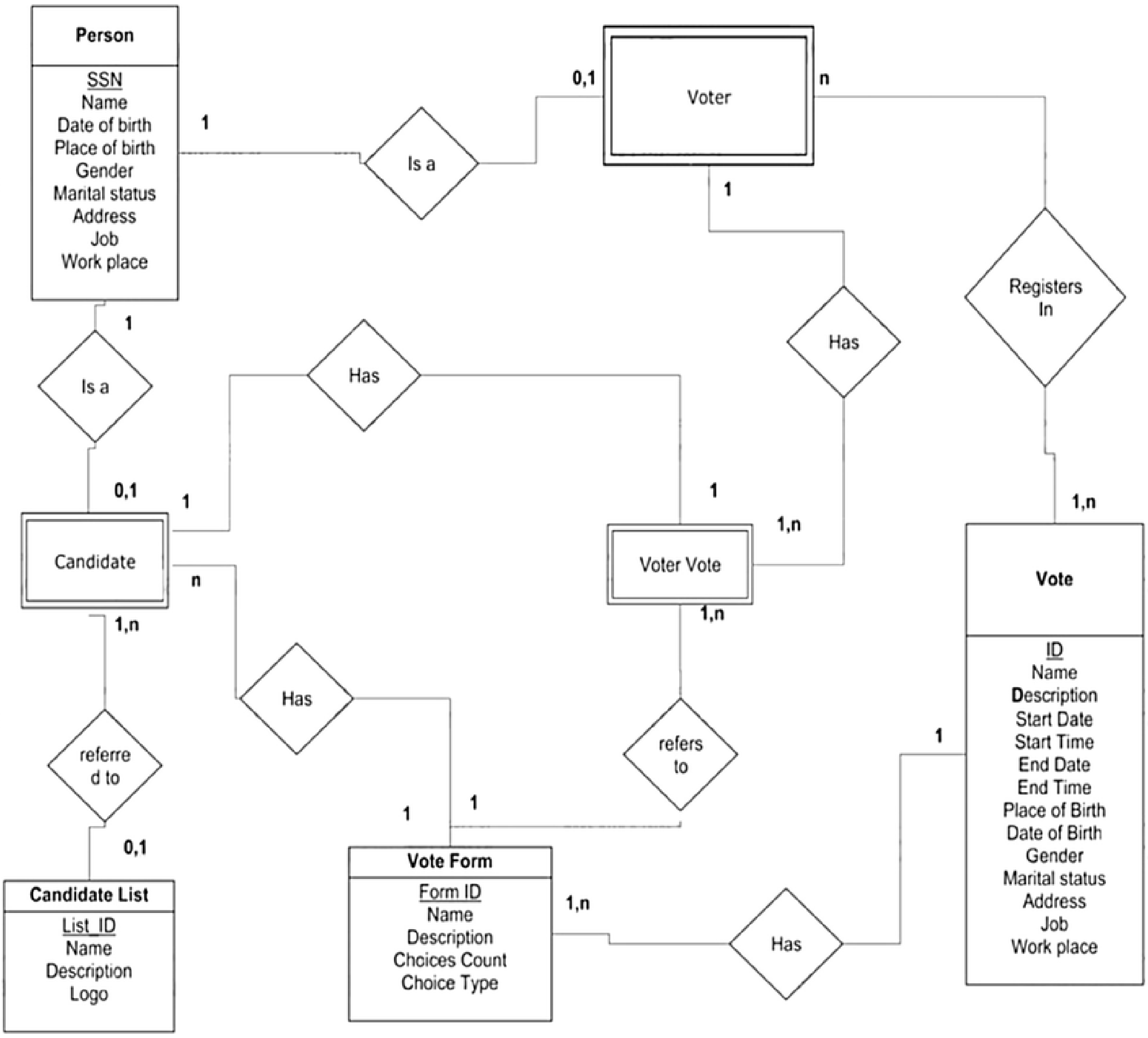
https://www.freeprojectz.com/uml-diagram/e-voting-management-system-sequence-diagram

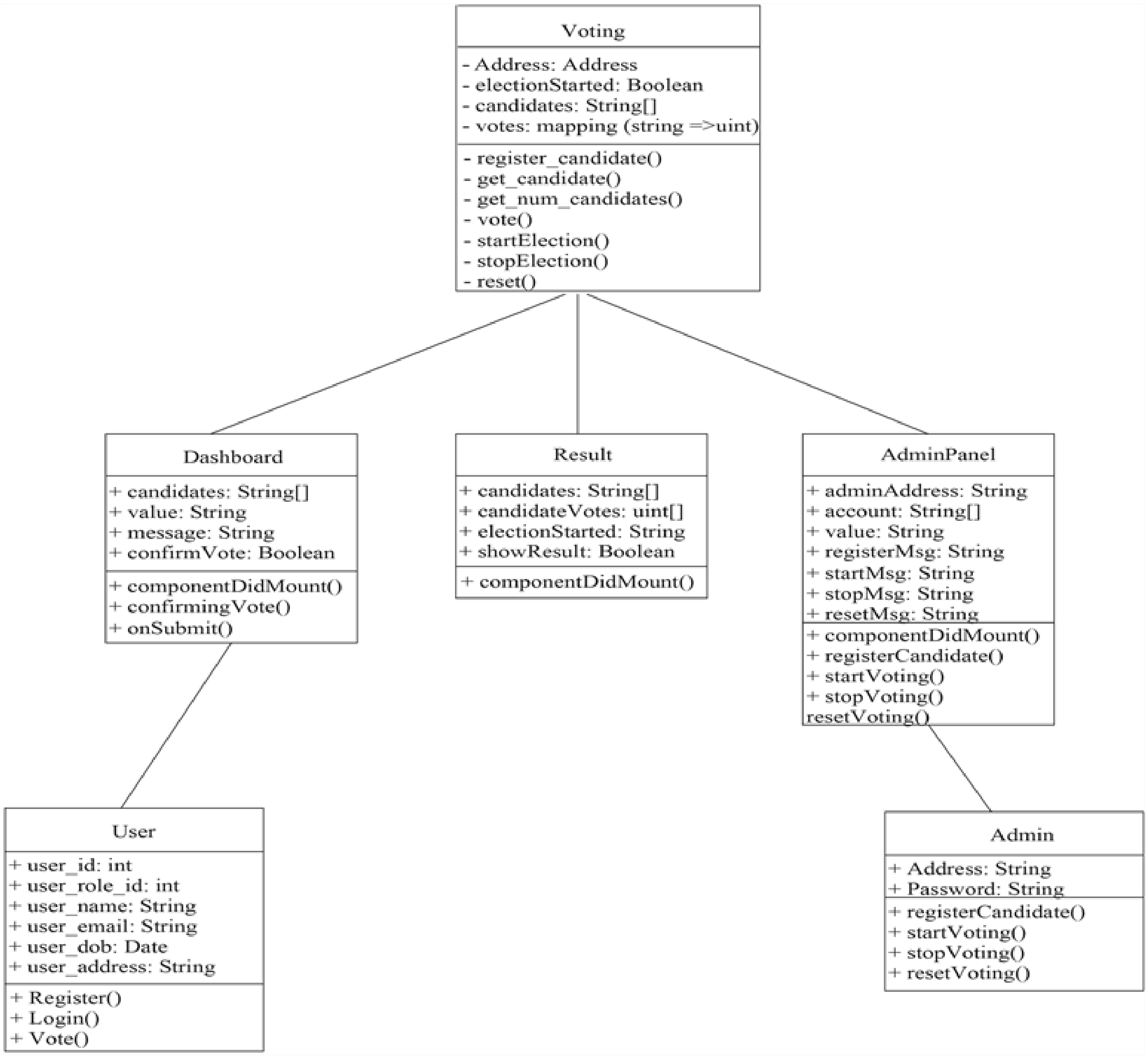
https://www.freeprojectz.com/uml-diagram/e-voting-management-system-uml-diagram

Activity diagram = Flowchart

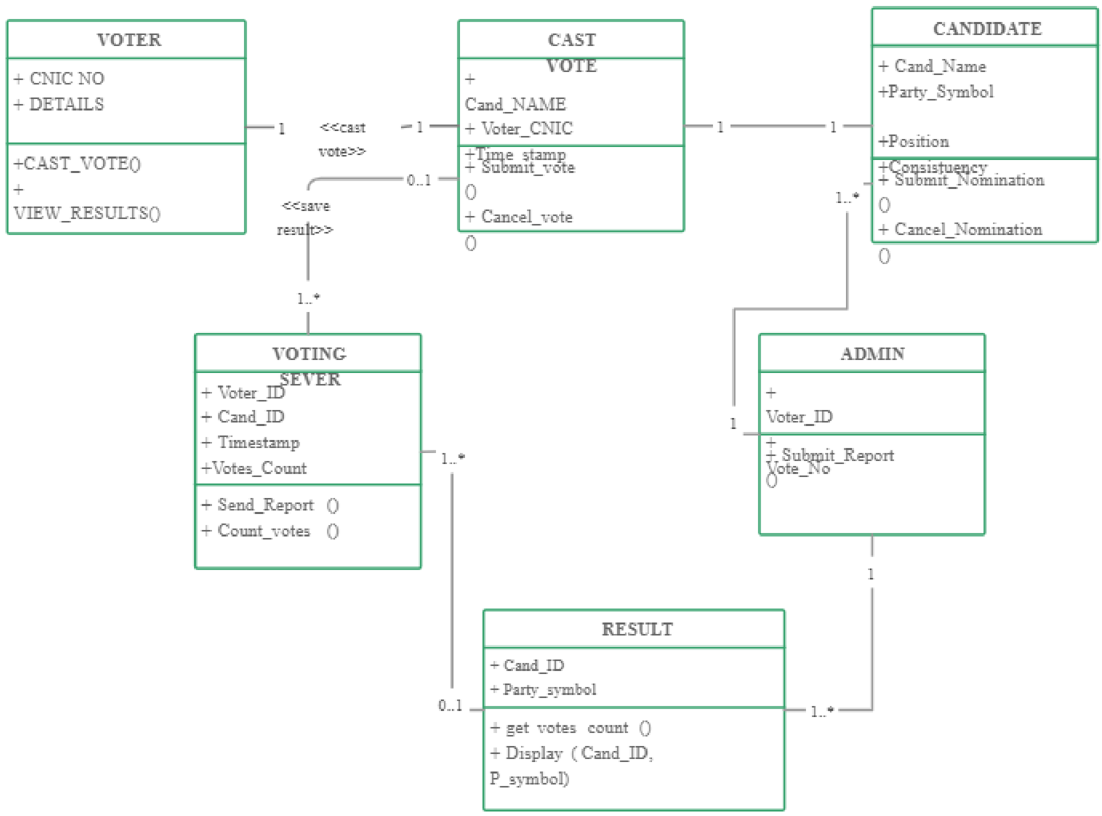
Deployment Diagram@ page 7

Data model/data set



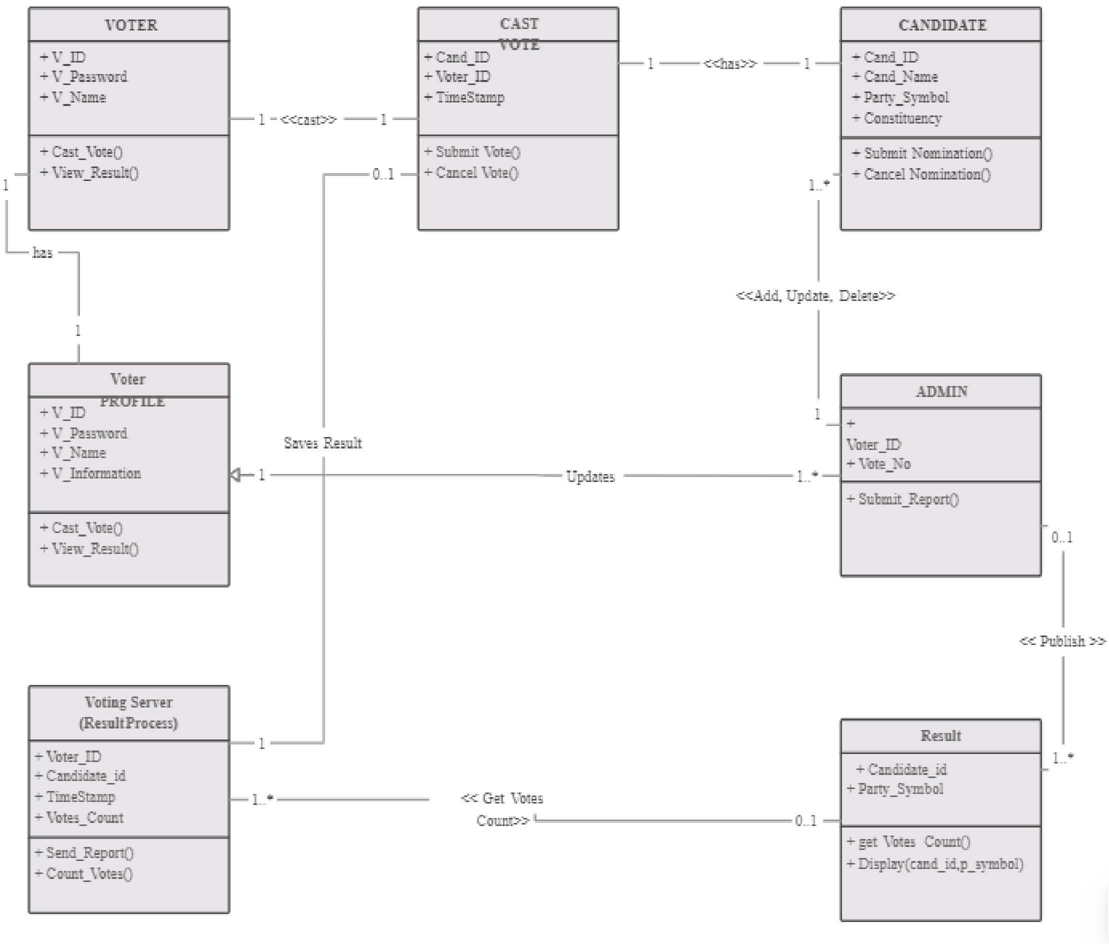

Voting
- Address: Address
- electionStarted: Boolean
- candidates: String[]
- votes: mapping (string =>uint)
- register_candidate()
- get_candidate()
- get_num_candidates()
- vote()
- startElection()
- stopElection()
- reset()
Dashboard
+ candidates: String[]
+ value: String
+ message: String
+ confirmVote: Boolean
+ componentDidMount()
+ confirmingVote()
+ onSubmit()
Result
+ candidates: String[]
+ candidateVotes: uint[]
+ electionStarted: String
+ showResult: Boolean
+ componentDidMount()
AdminPanel
+ adminAddress: String
+ account: String[]
+ value: String
+ registerMsg: String
+ startMsg: String
+ stopMsg: String
+ resetMsg: String
+ componentDidMount()
+ registerCandidate()
+ startVoting()
+ stopVoting()
resetVoting()
User
+ user_id: int
+ user_role_id: int
+ user_name: String
+ user_email: String
+ user_dob: Date
+ user_address: String
+ Register()
+ Login()
+ Vote()
Admin
+ Address: String
+ Password: String
+ registerCandidate()
+ startVoting()
+ stopVoting()
+ resetVoting()

Class diagram



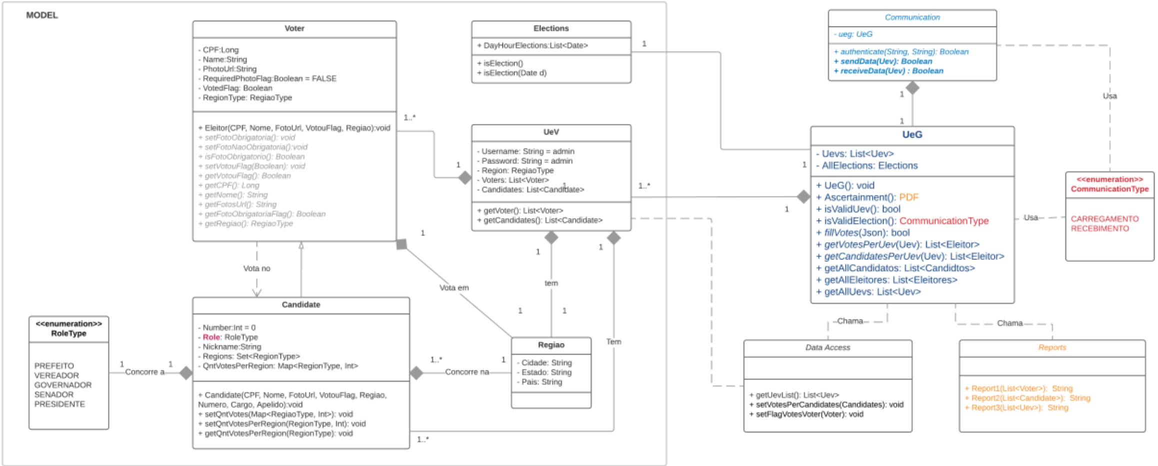
CLASS-DIAGRAMFOR ONLINE VOTING SYSTEM

E-Voting System
Class Diagram

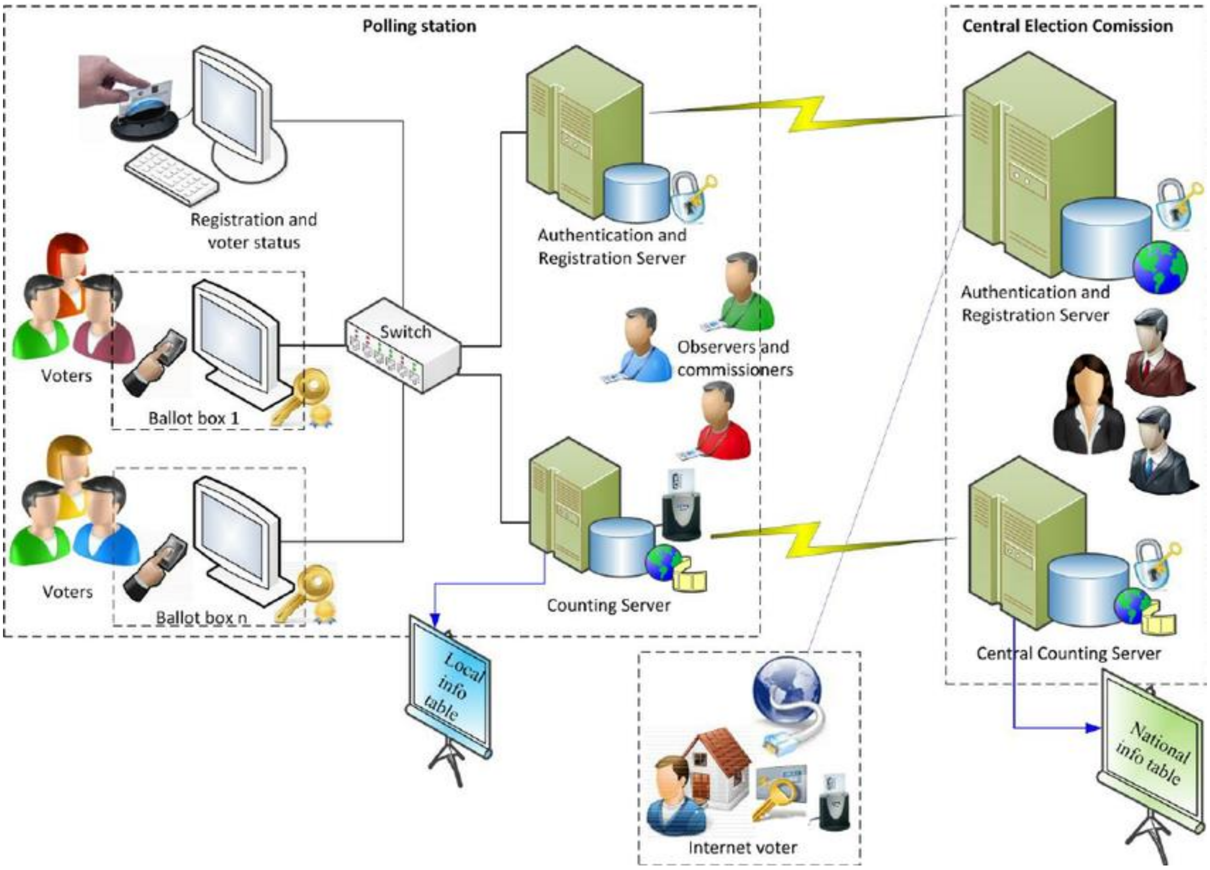


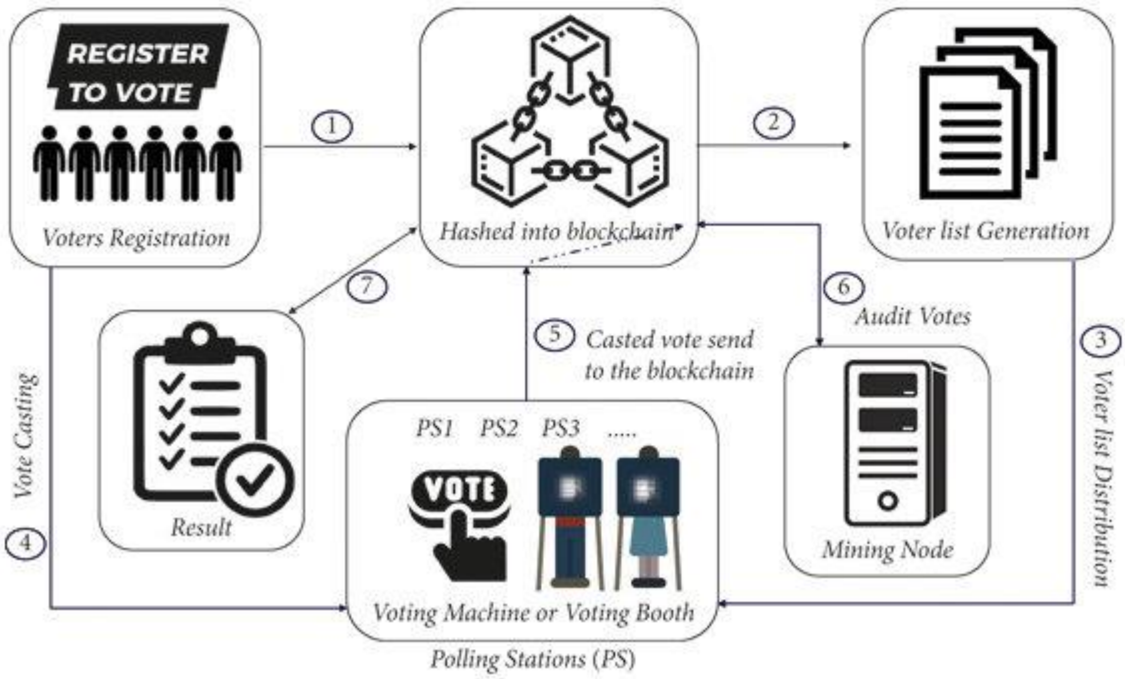
UEG - DIAGRAMA DE CLASSE

Pamela Iupi Peixinho | Ahmad Ali el Khatib



Deployment Diagram





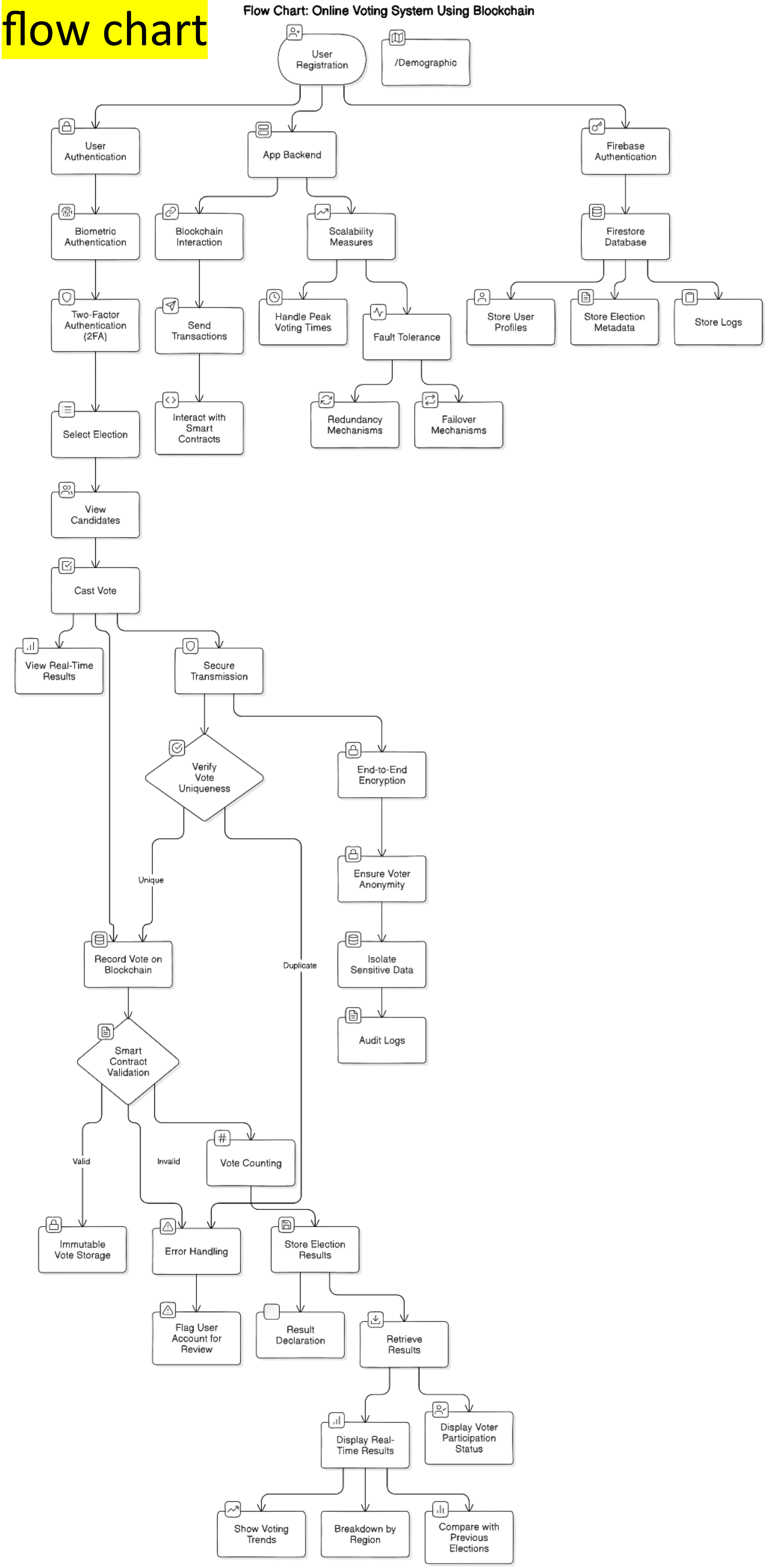
Week 12 :

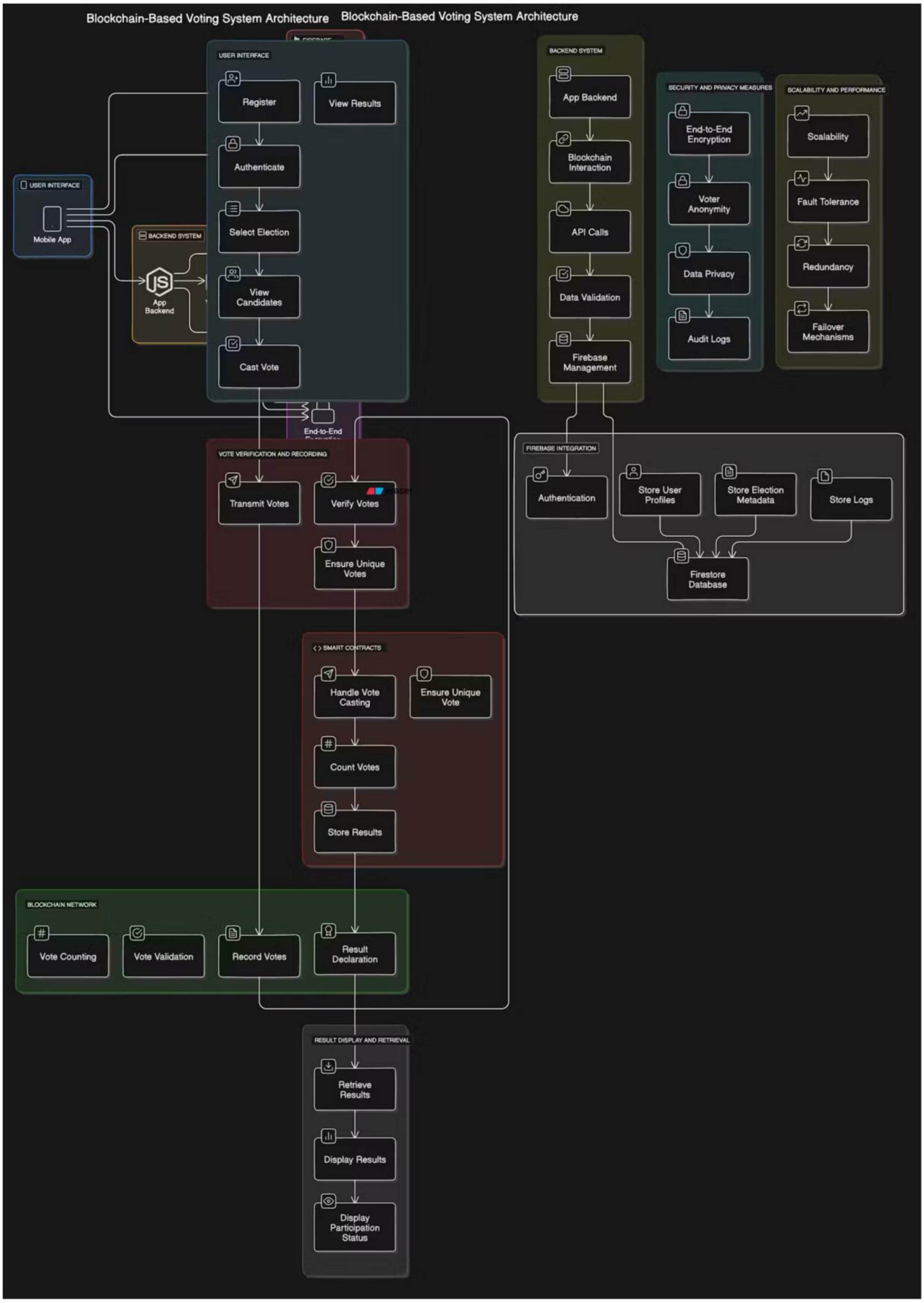
Wireframe/Algorithm

→ UI not ready

→ Algorithm == Flowchart

flow chart/Input form design/Output Report Design@ page 9 & 10 flow chart


Flow Chart: Online Voting System Using Blockchain
User Registration
/Demographic
User Authentication
App Backend
Firebase Authentication
Biometric Authentication
Blockchain Interaction
Scalability Measures
Firestore Database
Two-Factor Authentication (2FA)
Send Transactions
Handle Peak Voting Times
Fault Tolerance
Store User Profiles
Store Election Metadata
Store Logs
Select Election
Interact with Smart Contracts
Redundancy Mechanisms
Failover Mechanisms
View Candidates
Cast Vote
View Real-Time Results
Secure Transmission
Verify Vote Uniqueness
End-to-End Encryption
Unique
Ensure Voter Anonymity
Record Vote on Blockchain
Duplicate
Isolate Sensitive Data
Audit Logs
Smart Contract Validation
Valid
Invalid
Vote Counting
Immutable Vote Storage
Error Handling
Store Election Results
Flag User Account for Review
Result Declaration
Retrieve Results
Display Real-Time Results
Display Voter Participation Status
Show Voting Trends
Breakdown by Region
Compare with Previous Elections


Blockchain-Based Voting System Architecture
Blockchain-Based Voting System Architecture
USER INTERFACE
Mobile App
BACKEND SYSTEM
App Backend
USER INTERFACE
Register
View Results
Authenticate
Select Election
View Candidates
Cast Vote
End-to-End
BACKEND SYSTEM
App Backend
Blockchain Interaction
API Calls
Data Validation
Firebase Management
SECURITY AND PRIVACY MEASURES
End-to-End Encryption
Voter Anonymity
Data Privacy
Audit Logs
SCALABILITY AND PERFORMANCE
Scalability
Fault Tolerance
Redundancy
Failover Mechanisms
VOTE VERIFICATION AND RECORDING
Transmit Votes
Verify Votes
Ensure Unique Votes
FIREBASE INTEGRATION
Authentication
Store User Profiles
Store Election Metadata
Store Logs
Firestore Database
SMART CONTRACTS
Handle Vote Casting
Ensure Unique Vote
Count Votes
Store Results
BLOCKCHAIN NETWORK
Vote Counting
Vote Validation
Record Votes
Result Declaration
RESULT DISPLAY AND RETRIEVAL
Retrieve Results
Display Results
Display Participation Status

Week 13:

RMMM Plan @ page 12 & 13

Feasibility analysis @ page 14

Test case Dsign @ page 15

RMMM Plan

RMMM Plan (Risk Mitigation, Monitoring, and Management)

Table: Risk Categories

Risk ID	**Risk Category**	**Probability**	**Impact**	**Description**	**Mitigation Plan**	**Monitoring Strategy**	**Management Strategy**
RMMM 1	**Scalability**	60%	High	Ethereum's transaction throughput may cause delays, especially in large elections.	Implement transaction batching and use a private blockchain for low-cost, high-speed transactions.	Monitor transaction processing times and user load during peak voting periods.	Scale the infrastructure dynamically during high traffic voting periods.
RMMM 2	**Security**	50%	High	DDoS attacks could affect the backend or blockchain interaction, causing system downtime.	Strengthen backend security with rate limiting, firewalls, and distributed denial-of-service (DDoS) prevention mechanisms.	Continuous monitoring for unusual traffic patterns.	Activate fallback systems and reroute traffic to unaffected nodes.
RMMM 3	**Privacy**	30%	High	Sensitive voter data may be exposed due to insufficient encryption or data management practices.	Implement end-to-end encryption for all sensitive data transfers and storage.	Regular audits of encryption protocols and data flow paths.	Ensure legal compliance with privacy regulations; isolate sensitive data in blockchain.

Risk ID	Risk Category	Probability	Impact	Description	Mitigation Plan	Monitoring Strategy	Management Strategy
RMMM 4	**System Downtime**	20%	Moderate	Unexpected system downtime may cause disruption in elections.	Deploy backup servers and failover systems to ensure high availability.	Track system performance, uptime, and downtime metrics.	Have emergency support available for immediate issue resolution.
RMMM 5	**Integration Failures**	40%	Moderate	Integration between the mobile app, backend, and blockchain may face challenges.	Implement continuous integration testing for app-backend-blockchain workflows.	Monitor API responses, transaction failures, and timeout issues.	Develop fallback logic to handle temporary disconnections gracefully.

Feasibility analysis

Feasibility Analysis

Technical Feasibility

- **Blockchain for Vote Storage**: Ethereum or a similar blockchain network ensures secure, immutable, and tamper-proof vote storage. Smart contracts enable vote counting and ensure that no duplicate voting occurs.
- **Firebase for Metadata**: Storing non-sensitive data such as voter logs and metadata in Firebase provides a cost-effective and efficient solution for off-chain operations, reducing blockchain transaction costs.
- **User-Friendly Authentication**: Multi-factor authentication (MFA) with biometric verification provides a secure and easy way for voters to register and authenticate.

Financial Feasibility

- **Cost of Blockchain Transactions**: Public blockchains like Ethereum have transaction costs, but this can be minimized by using Layer 2 scaling solutions or consortium blockchains. Additionally, operations such as vote recording are optimized to avoid unnecessary costs.
- **Firebase Usage Costs**: Firebase is utilized for handling non-voting-related metadata and logs, which reduces the need for on-chain storage, keeping costs low.

Operational Feasibility

- **Ease of Use**: The mobile app is designed for ease of use, with a straightforward interface for voters to register, select elections, and cast votes. Multi-factor authentication ensures that even non-technical users can participate securely.
- **Handling Peak Voting Periods**: The system is scalable to handle a large number of users during peak voting times, ensuring that even national elections can be conducted smoothly.

Test case Dsign

Test Case Design

1. Authentication Test

- **Description**: Verify that the user authentication system works correctly with multi-factor authentication (password, biometric, OTP).
- **Input**: Voter enters email/password, biometric data, and receives OTP.
- **Expected Output**: Successful login, access granted to the dashboard.

2. Vote Casting Test

- **Description**: Ensure that votes are securely cast and recorded on the blockchain.
- **Input**: User selects election and candidate, confirms vote.
- **Expected Output**: Vote is recorded on the blockchain, transaction hash is returned, and confirmation message is displayed to the user.

3. Duplicate Voting Prevention

- **Description**: Ensure that users cannot vote more than once in the same election.
- **Input**: User attempts to cast a vote for the second time in the same election.
- **Expected Output**: Error message indicating that the user has already voted, and no new transaction is created on the blockchain.

4. Result Display Test

- **Description**: Validate that real-time results are fetched from the blockchain and displayed correctly.
- **Input**: User queries election results after voting ends.
- **Expected Output**: Real-time results fetched and displayed from the blockchain, showing vote counts by candidate and overall turnout.

5. Data Encryption and Security Test

- **Description**: Verify that all sensitive data, including votes and user details, are encrypted during transmission and storage.
- **Input**: Vote data submitted by the user.
- **Expected Output**: Data is encrypted during transmission, verified by checking the blockchain records and ensuring sensitive information remains confidential.

6. Scalability Test

- **Description**: Assess the system's ability to handle a high number of concurrent users.
- **Input**: Simulate thousands of users casting votes simultaneously.
- **Expected Output**: The system should process all votes with minimal latency and no downtime, showing no degradation in performance.

7. Failover and Recovery Test

- **Description**: Ensure that the system can recover from failures such as server crashes or blockchain node disconnection.
- **Input**: Simulate a backend server crash during voting.
- **Expected Output**: The system automatically switches to a backup server with no data loss or impact on ongoing voting processes.