# Blockchain-Based Electronic Voting System with E2E-V Integration

## Complete Java Implementation Documentation

---

## Table of Contents

---

## 1. Project Overview

### 1.1 Project Description

A secure, transparent, and tamper-proof electronic voting system built using blockchain technology with End-to-End Verifiability (E2E-V) and cryptographic receipts. The system eliminates centralized authorities, prevents fraud and double-voting, and provides complete auditability.

### 1.2 Key Features

- **Blockchain Integration**: Immutable vote storage using blockchain principles
- **E2E-V (End-to-End Verifiability)**: Voters can verify their vote at every stage
- **Cryptographic Receipts**: Unique receipt generation for vote verification
- **Multi-Role System**: Admin, Voter, and Candidate modules
- **Real-time Results**: Transparent vote tallying and visualization
- **Security Measures**: Double-voting prevention, voter eligibility checks, timestamp validation

### 1.3 Problem Statement

Traditional voting systems suffer from:

- Lack of transparency
- Vulnerability to fraud and manipulation
- Centralized control
- Inability to verify votes
- Double-voting possibilities

### 1.4 Solution Approach

Blockchain-based voting with E2E-V provides:

- Cast-as-intended verification (voter confirms their choice)
- Recorded-as-cast verification (vote stored correctly on blockchain)
- Counted-as-recorded verification (accurate tallying)

---

## 2. System Architecture

### 2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────┐
│                   Frontend Layer                        │
│         (Web Interface - React/HTML/CSS)                │
└────────────────────────┬────────────────────────────────┘
                         │ REST API Calls
┌────────────────────────▼────────────────────────────────┐
│              Backend Layer (Java)                       │
│  ┌──────────────┬──────────────┬──────────────────┐     │
│  │ Voter Module │ Admin Module │ Candidate Module │     │
│  └──────────────┴──────────────┴──────────────────┘     │
└────────────────────────┬────────────────────────────────┘
                         │
┌────────────────────────▼────────────────────────────────┐
│         Business Logic Layer (Services)                 │
│  ┌────────────────┬──────────────────┬──────────────┐   │
│  │ Authentication │ Blockchain Engine│ Validation   │   │
│  │ Service        │ Service          │ Service      │   │
│  └────────────────┴──────────────────┴──────────────┘   │
└────────────────────────┬────────────────────────────────┘
                         │
┌────────────────────────▼────────────────────────────────┐
```

```
│          Data Layer                          │
│  ┌───────────────────────────────┬──────────────────────────────┬───────────────────────────┐   │
│  │ MySQL Database │ Blockchain Chain │ File Storage   │  │
│  │ (User Data)    │ (Vote Records)   │ (Receipts)     │  │
│  └───────────────────────────────┴──────────────────────────────┴───────────────────────────┘   │
└────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

## 2.2 Workflow Architecture

```
Registration → Authentication → Vote Casting → Blockchain Recording
   ↓           ↓             ↓             ↓
 Database    Session Token  Encryption    Block Creation
                                        ↓
Receipt Generation ← Vote Verification ← Block Validation
   ↓
Vote Counted in Results
```

---

# 3. Technology Stack

## 3.1 Core Technologies

- **Programming Language**: Java 17 or higher
- **Framework**: Spring Boot 3.x
- **Database**: MySQL 8.0
- **Build Tool**: Maven
- **Security**: Spring Security, JWT
- **Cryptography**: Java Security API, BouncyCastle

## 3.2 Required Libraries (Maven Dependencies)

```xml
  spring-boot-starter-web (REST API)
```

```
- spring-boot-starter-web (REST API)
- spring-boot-starter-data-jpa (Database ORM)
- spring-boot-starter-security (Authentication)
- mysql-connector-java (MySQL driver)
- lombok (Reduce boilerplate code)
- jjwt (JWT token generation)
- bcrypt (Password hashing)
- jackson-databind (JSON processing)
- commons-codec (Apache Commons for encoding)
- bouncycastle (Cryptographic operations)
```

### 3.3 Development Tools

- **IDE**: IntelliJ IDEA / Eclipse
- **Database Tool**: MySQL Workbench
- **API Testing**: Postman
- **Version Control**: Git
- **Frontend**: React.js (or plain HTML/CSS/JavaScript)

---

## 4. Core Concepts Required

### 4.1 Java Concepts

1. **Object-Oriented Programming (OOP)**
   - Classes and Objects
   - Inheritance and Polymorphism
   - Encapsulation and Abstraction
   - Interfaces and Abstract Classes
2. **Collections Framework**
   - ArrayList, HashMap, LinkedList
   - Iteration and Stream API
3. **Exception Handling**
   - Try-catch blocks
   - Custom exceptions
   - Exception propagation
4. **Multithreading** (Optional for performance)
   - Thread creation and management
   - Synchronization
5. **File I/O**

3. **File I/O**
    - Reading/Writing files
    - Serialization

## 4.2 Spring Boot Concepts

1. **Dependency Injection (DI)**
    - @Autowired, @Component, @Service, @Repository

2. **RESTful API Development**
    - @RestController, @RequestMapping
    - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping
    - Request/Response handling
3. **Spring Data JPA**
    - Entity mapping (@Entity, @Table)
    - Repository pattern (@Repository)
    - JPQL and native queries
4. **Spring Security**
    - Authentication and Authorization
    - JWT token management
    - Password encoding

## 4.3 Blockchain Concepts

1. **Block Structure**
    - Index, Timestamp, Data, Previous Hash, Current Hash, Nonce
2. **Hashing**
    - SHA-256 algorithm
    - Hash generation and verification
3. **Chain Validation**
    - Verifying block integrity
    - Checking hash linkage
4. **Immutability**
    - Preventing data tampering
    - Detecting chain modifications

## 4.4 Cryptography Concepts

1. **Hash Functions**
   - One-way hashing (SHA-256)
   - Message digests
2. **Digital Signatures** (Optional for advanced implementation)
   - Public-key cryptography
   - Signature verification
3. **Random Number Generation**
   - Secure random for receipt generation
   - Nonce generation
4. **Encryption** (Optional)
   - Symmetric encryption (AES)
   - Asymmetric encryption (RSA)

### 4.5 Database Concepts

1. **SQL Basics**
   - CRUD operations (Create, Read, Update, Delete)
   - Joins and relationships
2. **Database Design**
   - Entity-Relationship modeling
   - Normalization
   - Primary and Foreign keys
3. **Transactions**
   - ACID properties
   - Transaction management

---

## 5. Module Breakdown

### 5.1 Voter Module

#### 5.1.1 Responsibilities

- Voter registration
- Voter authentication
- Vote casting
- Receipt generation
- Vote verification

- Result viewing

### 5.1.2 Key Classes

```java
// VoterController.java
- registerVoter()
- loginVoter()
- castVote()
- verifyVote()
- getResults()

// VoterService.java
- createVoter()
- authenticateVoter()
- submitVote()
- generateReceipt()
- validateVoterEligibility()

// Voter.java (Entity)
- voterId (PK)
- name
- email
- password (hashed)
- votedStatus
- registrationTimestamp
```

### 5.1.3 Voter Flow

```
1. Registration
  - Voter fills registration form
  - System validates unique email/ID
  - Password is hashed using BCrypt
  - Admin approval pending
  - Voter record saved to database

2. Authentication
  - Voter enters credentials
  - System verifies password
  - JWT token generated
```

```
   - Session created

3. Vote Casting
   - System checks if voter already voted (double-vote prevention)
   - System verifies voter is approved
   - Voter selects candidate
   - System generates vote ID
   - Vote encrypted (optional)
   - Blockchain block created with vote data
   - Block added to chain
   - Cryptographic receipt generated
   - Voter's votedStatus updated to TRUE

4. Vote Verification
   - Voter enters receipt ID
   - System searches blockchain for matching vote
   - Returns verification status and vote details
```

---

## 5.2 Admin Module

### 5.2.1 Responsibilities

- Election creation and management
- Candidate registration and approval
- Voter approval
- Election start/stop control
- Result publishing
- System monitoring

### 5.2.2 Key Classes

```
java












// AdminController.java
// AdminController.java
- createElection()
- approveVoter()
- addCandidate()
- startElection()
- endElection()
- publishResults()
- getBlockchain()

// AdminService.java
- setupElection()
- manageVoters()
- manageCandidates()
- controlElectionStatus()
- generateResultReport()

// Admin.java (Entity)
- adminId (PK)
- username
- password (hashed)
- role
```

### 5.2.3 Admin Flow

```
1. Election Setup
  - Admin logs in
  - Creates new election
  - Sets election name, description, start/end dates
  - Election saved to database

2. Candidate Management
  - Admin adds candidates
  - Specifies candidate name, party, symbol
  - Candidates linked to election
  - Candidates visible to voters

3. Voter Approval
  - Admin views pending voter registrations
  - Approves/rejects voters
  - Approved voters can vote

4. Election Control
  - Admin starts election (voting enabled)
  - Admin ends election (voting disabled)
  - Status updates affect voter access

5. Result Publishing
  - Admin triggers result calculation
```

```
   - Admin triggers result calculation
   - System tallies votes from blockchain
   - Results displayed with candidate-wise vote counts
```

---

## 5.3 Candidate Module

### 5.3.1 Responsibilities

- Candidate registration
- Profile management
- Vote count viewing

### 5.3.2 Key Classes

```java
// CandidateController.java
- registerCandidate()
- getCandidateProfile()
- getVoteCount()

// CandidateService.java
- createCandidate()
- updateCandidateInfo()
- fetchVoteStatistics()

// Candidate.java (Entity)
- candidateId (PK)
- name
- party
- symbol
- electionId (FK)
- voteCount
```

### 5.3.3 Candidate Flow

```
1. Registration
   - Candidate fills registration form
   - Data saved to database
   - Awaits admin approval

2. Profile Management
   - Candidate can view/update profile information

3. Vote Tracking
```

```
- Candidate can view real-time vote count
- System counts votes from blockchain
```

---

## 5.4 Blockchain Engine Module

### 5.4.1 Responsibilities

- Block creation
- Hash generation
- Chain validation
- Block linkage
- Immutability enforcement

### 5.4.2 Key Classes

```
java

// BlockchainService.java
- createGenesisBlock()
- addBlock()
- validateChain()
- getBlockchain()
- getBlockByHash()

// Block.java (Entity/POJO)
- index
- timestamp
- voteData (Vote object)
- previousHash
- currentHash
- nonce

// Blockchain.java (Main Chain)
- List<Block> chain
- difficulty (for proof-of-work, optional)
```

### 5.4.3 Block Structure

```
java
```

```
class Block {
    private int index;                  // Block position in chain
    private long timestamp;             // Block creation time
    private String voteData;            // Encrypted vote information
    private String previousHash;        // Hash of previous block
    private String currentHash;         // Hash of this block
    private int nonce;                  // For proof-of-work (optional)

    // Hash calculation method
    public String calculateHash() {
        String data = index + timestamp + voteData + previousHash + nonce;
        return SHA256(data);
    }
}
```

### 5.4.4 Blockchain Flow

```
1. Genesis Block Creation
   - First block in chain
   - previousHash = "0"
   - Contains no vote data (initialization block)

2. Adding New Block
   - Vote data received from Voter Service
   - New Block object created
   - index = previous block index + 1
   - previousHash = previous block's currentHash
   - currentHash calculated using SHA-256
   - Block added to chain (ArrayList)
   - Block persisted to database (optional)

3. Chain Validation
   - Iterate through all blocks
   - Verify each block's hash
   - Verify each block's previousHash matches previous block's currentHash
   - If any mismatch ⟶ chain is invalid (tampering detected)

4. Vote Retrieval
```

```
- Search blockchain for specific vote by receipt ID
- Return vote details for verification
```

### 5.4.5 Hash Generation (SHA-256)

java

```java
import java.security.MessageDigest;

public String calculateHash(String data) {
    try {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(data.getBytes("UTF-8"));
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

---

## 5.5 Security and Validation Module

### 5.5.1 Responsibilities

- Password hashing and verification
- JWT token generation and validation

- Double-voting prevention
- Voter eligibility verification
- Timestamp validation
- Input sanitization

### 5.5.2 Key Classes

```
java

// SecurityService.java
- hashPassword()
- verifyPassword()
- generateJWT()
- validateJWT()
- checkDoubleVoting()

// ValidationService.java
- validateVoterEligibility()
- validateElectionStatus()
- sanitizeInput()
- verifyTimestamp()
```

### 5.5.3 Security Flow

```
1. Password Management
  - Registration: password hashed with BCrypt (strength: 12 rounds)
  - Login: entered password hashed and compared with stored hash

2. JWT Token Management
  - Token generated upon successful login
  - Contains: voterId, username, expiration time
  - Token sent in Authorization header for subsequent requests

  - Backend validates token before processing requests

3. Double-Voting Prevention
  - Before casting vote, system checks voter's votedStatus
  - If TRUE → reject vote attempt
  - If FALSE → allow vote and update status to TRUE

4. Eligibility Verification
  - Check if voter is approved by admin
  - Check if election is active
  - Check if voter is registered for specific election
```

## 5.6 Results Module

### 5.6.1 Responsibilities

- Vote tallying from blockchain
- Result calculation
- Winner determination
- Result visualization

### 5.6.2 Key Classes

```java
// ResultsService.java
- tallyVotes()
- calculateResults()
- determineWinner()
- getResultsByElection()

// ElectionResult.java (Entity)
- electionId (PK, FK)
- candidateId (FK)
- voteCount
- percentage
- rank
```

### 5.6.3 Results Flow

```
1. Vote Tallying
  - Iterate through blockchain
  - Extract candidateId from each vote block
  - Count votes for each candidate
  - Store counts in HashMap<candidateId, count>

2. Result Calculation
  - Calculate total votes cast
  - Calculate percentage for each candidate
  - Determine winner (highest votes)
  - Save results to database

3. Result Visualization
  - Frontend fetches results via API
  - Display bar chart or pie chart
  - Show candidate names, vote counts, percentages
```

# 6. Database Schema

## 6.1 Tables

### 6.1.1 voters

sql

```
CREATE TABLE voters (
    voter_id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    age INT,
    address TEXT,
    phone VARCHAR(15),
    voted_status BOOLEAN DEFAULT FALSE,
    approved_status BOOLEAN DEFAULT FALSE,
    registration_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 6.1.2 candidates

sql

```
CREATE TABLE candidates (
    candidate_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    party VARCHAR(100),
    symbol VARCHAR(50),
    election_id INT,
    vote_count INT DEFAULT 0,
    FOREIGN KEY (election_id) REFERENCES elections(election_id)
);
```

### 6.1.3 elections

```
sql

CREATE TABLE elections (
    election_id INT PRIMARY KEY AUTO_INCREMENT,
    election_name VARCHAR(200) NOT NULL,
    description TEXT,
    start_date DATETIME,
    end_date DATETIME,
    status ENUM('pending', 'active', 'completed') DEFAULT 'pending',
    created_by VARCHAR(50),
    created_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 6.1.4 admins

```
sql

CREATE TABLE admins (
    admin_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) DEFAULT 'admin',
    created_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 6.1.5 votes (Optional - for non-blockchain storage)

```
sql

CREATE TABLE votes (
    vote_id VARCHAR(100) PRIMARY KEY,
    voter_id VARCHAR(50),
    candidate_id INT,
    election_id INT,
    receipt_id VARCHAR(100) UNIQUE,
    vote_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (voter_id) REFERENCES voters(voter_id),
    FOREIGN KEY (candidate_id) REFERENCES candidates(candidate_id),
    FOREIGN KEY (election_id) REFERENCES elections(election_id)
);
```

### 6.1.6 blockchain_blocks (For persistence)

```
sql

CREATE TABLE blockchain_blocks (
    block_id INT PRIMARY KEY AUTO_INCREMENT
    block_id INT PRIMARY KEY AUTO_INCREMENT,
    block_index INT NOT NULL,
    timestamp BIGINT NOT NULL,
    vote_data TEXT NOT NULL,
    previous_hash VARCHAR(64) NOT NULL,
    current_hash VARCHAR(64) UNIQUE NOT NULL,
    nonce INT DEFAULT 0,
    created_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 6.1.7 election_results

sql

```sql
CREATE TABLE election_results (
    result_id INT PRIMARY KEY AUTO_INCREMENT,
    election_id INT,
    candidate_id INT,
    vote_count INT,
    percentage DECIMAL(5,2),
    rank INT,
    FOREIGN KEY (election_id) REFERENCES elections(election_id),
    FOREIGN KEY (candidate_id) REFERENCES candidates(candidate_id)
);
```

## 6.2 Entity Relationships

```
elections (1) -----> (N) candidates
elections (1) -----> (N) votes
voters (1) -----> (N) votes
candidates (1) -----> (N) votes

elections (1) -----> (1) election_results
```

---

# 7. File Structure

```
blockchain-voting-system/
│
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── voting/
│   │   │           └── blockchain/
│   │   │               ├── BlockchainVotingApplication.java (Main class)
│   │   │               │
│   │   │         │
│   │   │         ├── config/
│   │   │         │   ├── SecurityConfig.java (Spring Security config)
│   │   │         │   ├── CorsConfig.java (CORS configuration)
│   │   │         │   └── DatabaseConfig.java (Database config)
│   │   │         │
│   │   │         ├── controller/
│   │   │         │   ├── VoterController.java
│   │   │         │   ├── AdminController.java
│   │   │         │   ├── CandidateController.java
│   │   │         │   ├── BlockchainController.java
│   │   │         │   └── ResultsController.java
│   │   │         │
│   │   │         ├── service/
│   │   │         │   ├── VoterService.java
│   │   │         │   ├── AdminService.java
│   │   │         │   ├── CandidateService.java
│   │   │         │   ├── BlockchainService.java
│   │   │         │   ├── SecurityService.java
│   │   │         │   ├── ValidationService.java
│   │   │         │   └── ResultsService.java
│   │   │         │
│   │   │         ├── model/
│   │   │         │   ├── Voter.java (Entity)
│   │   │         │   ├── Admin.java (Entity)
│   │   │         │   ├── Candidate.java (Entity)
│   │   │         │   ├── Election.java (Entity)
│   │   │         │   ├── Vote.java (Entity)
│   │   │         │   ├── Block.java (POJO)
│   │   │         │   ├── Blockchain.java (Main chain)
│   │   │         │   ├── BlockchainBlock.java (Entity for persistence)
│   │   │         │   └── ElectionResult.java (Entity)

│   │   │         │
│   │   │         ├── repository/
│   │   │         │   ├── VoterRepository.java (JPA interface)
│   │   │         │   ├── AdminRepository.java
│   │   │         │   ├── CandidateRepository.java
│   │   │         │   ├── ElectionRepository.java
│   │   │         │   ├── VoteRepository.java
│   │   │         │   ├── BlockchainBlockRepository.java
│   │   │         │   └── ElectionResultRepository.java
│   │   │         │
│   │   │         ├── dto/
│   │   │         │   ├── VoterRegistrationDTO.java
│   │   │         │   ├── LoginRequestDTO.java
│   │   │         │   ├── VoteCastDTO.java
│   │   │         │   ├── CandidateDTO.java
│   │   │         │   ├── ElectionDTO.java
│   │   │               │   ├── ElectionDTO.java
│   │   │               │   └── ResultDTO.java
│   │   │               │
│   │   │               ├── exception/
│   │   │               │   ├── GlobalExceptionHandler.java
│   │   │               │   ├── VoterNotFoundException.java
│   │   │               │   ├── AlreadyVotedException.java
│   │   │               │   ├── UnauthorizedException.java
│   │   │               │   └── BlockchainException.java
│   │   │               │
│   │   │               └── util/
│   │   │                   ├── HashUtil.java (SHA-256 utility)
│   │   │                   ├── JWTUtil.java (JWT token utility)
│   │   │                   ├── ReceiptGenerator.java
│   │   │                   └── DateUtil.java
│   │   │
│   │   └── resources/
│   │       ├── application.properties (Database, server config)
│   │       ├── application.yml (Alternative config)
│   │       └── static/ (Frontend files if serving from Spring Boot)
│   │
│   └── test/
│       └── java/
│           └── com/
│               └── voting/
│                   └── blockchain/
│                       ├── BlockchainServiceTest.java
│                       ├── VoterServiceTest.java
│                       └── SecurityServiceTest.java
│
├── frontend/ (Optional separate React app)
│   ├── public/

│   ├── src/
│   │   ├── components/
│   │   │   ├── VoterDashboard.jsx
│   │   │   ├── AdminDashboard.jsx
│   │   │   ├── VotingPage.jsx
│   │   │   └── ResultsPage.jsx
│   │   ├── services/
│   │   │   └── api.js
│   │   └── App.js
│   └── package.json
│
├── pom.xml (Maven dependencies)
├── README.md
└── .gitignore
```

# 8. Implementation Flow

## 8.1 Step-by-Step Development Process

### Phase 1: Project Setup (Day 1-2)

```
1. Create Spring Boot project using Spring Initializr
  - Dependencies: Web, JPA, MySQL, Security, Lombok

2. Set up MySQL database
  - Create database: blockchain_voting
  - Create tables as per schema

3. Configure application.properties
  - Database connection
  - Server port
  - JWT secret key

4. Create base project structure (packages)
```

### Phase 2: Core Blockchain Implementation (Day 3-5)

```
1. Create Block.java
  - Define block structure
  - Implement calculateHash() method

2. Create Blockchain.java
  - Initialize with genesis block
  - Implement addBlock() method

  - Implement validateChain() method

3. Create BlockchainService.java
  - Dependency injection setup
  - Implement blockchain operations
  - Add persistence logic

4. Test blockchain functionality
  - Unit tests for block creation
  - Chain validation tests
```

### Phase 3: Entity and Repository Setup (Day 6-7)

```
1. Create all Entity classes
  - Voter.java
```

```
   - Voter.java
   - Admin.java
   - Candidate.java
   - Election.java
   - Vote.java
   - ElectionResult.java

2. Create Repository interfaces
   - Extend JpaRepository
   - Add custom query methods if needed

3. Test database connectivity
   - Run application and verify table creation
```

### Phase 4: Security Implementation (Day 8-9)

```
1. Create SecurityConfig.java
   - Configure HTTP security
   - Define public and protected endpoints
   - Set up password encoder (BCrypt)

2. Create JWTUtil.java
   - Token generation method
   - Token validation method
   - Extract claims method

3. Create SecurityService.java
   - Password hashing
   - Password verification
   - Authentication logic

4. Test authentication flow
```

### Phase 5: Service Layer Implementation (Day 10-14)

```
1. VoterService.java
   - registerVoter()
   - authenticateVoter()
   - castVote()
   - verifyVote()

2. AdminService.java
   - createElection()
   - approveVoter()
   - manageElection()

3. CandidateService.java
```

```
3. CandidateService.java
   - registerCandidate()
   - getCandidateInfo()

4. ResultsService.java
   - tallyVotes()
   - calculateResults()

5. Integrate services with BlockchainService
```

### Phase 6: Controller Layer Implementation (Day 15-17)

```
1. Create all Controllers
   - VoterController.java
   - AdminController.java
   - CandidateController.java
   - BlockchainController.java
   - ResultsController.java

2. Define REST endpoints
   - Map service methods to endpoints
   - Add request/response validation

3. Test API endpoints using Postman
   - Create test collections
   - Test all CRUD operations
```

### Phase 7: E2E-V and Receipt Generation (Day 18-19)

```
1. Create ReceiptGenerator.java
   - Generate unique receipt IDs
   - Store receipt-vote mapping

2. Implement vote verification
   - Search blockchain by receipt ID
   - Return vote details

3. Test E2E-V flow
   - Cast vote ⟶ Get receipt ⟶ Verify vote
```

### Phase 8: Frontend Development (Day 20-25)

```
1. Create React app (or HTML/CSS/JS)

2. Implement pages:
   - Login/Registration
   - Voter Dashboard
   - Voting Page
   - Admin Dashboard
   - Results Page

3. Integrate with backend APIs
   - Use Axios/Fetch for API calls
   - Handle JWT token storage

4. Test UI/UX flow
```

### Phase 9: Testing and Debugging (Day 26-28)

```
1. Unit testing
   - Test individual methods
   - Mock dependencies

2. Integration testing
   - Test module interactions
   - Test API endpoints

3. System testing
   - End-to-end testing
   - Test complete workflows

4. Bug fixing and optimization
```

### Phase 10: Deployment (Day 29-30)

```
1. Prepare for production

   - Update configurations
   - Security hardening

2. Deploy backend
   - Deploy to cloud (AWS/Heroku/Azure)
   - Set up MySQL database

3. Deploy frontend
   - Deploy to Netlify/Vercel

4. Final testing in production
```

---

## 9. API Endpoints

## 9.1 Voter Endpoints

### POST /api/voter/register

json

```
Request Body:
{
  "voterId": "V12345",
  "name": "John Doe",
  "email": "john@example.com",
  "password": "SecurePass123",
  "age": 25,
  "phone": "1234567890"
}

Response (201 Created):
{
  "message": "Voter registered successfully. Awaiting admin approval.",
  "voterId": "V12345"
}
```

### POST /api/voter/login

json

```
Request Body:
{
  "voterId": "V12345",
  "password": "SecurePass123"
}

Response (200 OK):
{
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "voterId": "V12345",
  "name": "John Doe"
}
```

### POST /api/voter/vote

```json
Request Headers:
Authorization: Bearer <JWT_TOKEN>

Request Body:
{
  "voterId": "V12345",
  "candidateId": 3,
  "electionId": 1
}

Response (200 OK):
{
  "message": "Vote cast successfully",
  "receiptId": "RCP-a8f3d9e2c1b4567890abcdef",
  "blockHash": "00000a8b9c7d6e5f4a3b2c1d0e9f8a7b6c5d4e3f2a1b0c9d8e7f6a5b4c3d2e1f",
  "timestamp": 1696435200000,
  "verificationMessage": "Your vote has been recorded on the blockchain. Keep this receipt for verification."
}
```

### GET /api/voter/verify/{receiptId}

```json
Request Headers:
Authorization: Bearer <JWT_TOKEN>

Response (200 OK):
{
  "verified": true,
  "voteId": "VOTE-12345"
  "voteId": "VOTE-12345",
  "receiptId": "RCP-a8f3d9e2c1b4567890abcdef",
  "blockIndex": 42,
  "blockHash": "00000a8b9c7d6e5f4a3b2c1d0e9f8a7b6c5d4e3f2a1b0c9d8e7f6a5b4c3d2e1f",
  "timestamp": 1696435200000,
  "candidateId": 3,
  "electionId": 1,
  "message": "Your vote is verified and securely recorded on the blockchain."
}
```

### GET /api/voter/profile

```json
Request Headers:
Authorization: Bearer <JWT_TOKEN>

Response (200 OK):
{
  "voterId": "V12345",
  "name": "John Doe",
  "email": "john@example.com",
  "votedStatus": true,
  "approvedStatus": true,
  "registrationDate": "2024-10-01T10:30:00"
}
```

---

## 9.2 Admin Endpoints

### POST /api/admin/login

```json
Request Body:
{
  "username": "admin",
  "password": "admin123"
}

Response (200 OK):
{
  "message": "Admin login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "adminId": 1,
  "role": "admin"
}
```

## POST /api/admin/election/create

```json
Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Request Body:
{
  "electionName": "General Elections 2024",
  "description": "National parliamentary elections",
  "startDate": "2024-11-01T08:00:00",
  "endDate": "2024-11-01T18:00:00"
}

Response (201 Created):
{
  "message": "Election created successfully",
  "electionId": 1,
  "status": "pending"
}
```

## PUT /api/admin/voter/approve/{voterId}

```json
Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "message": "Voter approved successfully",
  "voterId": "V12345"
}
```

## POST /api/admin/candidate/add

```json
Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>
```

```
Request Body:
{
  "name": "Jane Smith",
  "party": "Progressive Party",
  "symbol": "Tree",
  "electionId": 1
}

Response (201 Created):
{
  "message": "Candidate added successfully",
  "candidateId": 3
}
```

### PUT /api/admin/election/start/{electionId}

```
json

Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "message": "Election started successfully",
  "electionId": 1,
  "status": "active"
}
```

### PUT /api/admin/election/end/{electionId}

```
json

Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "message": "Election ended successfully",
  "electionId": 1,
  "status": "completed"
}
```

### GET /api/admin/voters/pending

```
json

Request Headers:
```

```
Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "pendingVoters": [
    {
      "voterId": "V12346",
      "name": "Alice Johnson",
      "email": "alice@example.com",
      "registrationDate": "2024-10-02T14:20:00"
    },
    {
      "voterId": "V12347",
      "name": "Bob Williams",
      "email": "bob@example.com",
      "registrationDate": "2024-10-02T15:45:00"
    }
  ]
}
```

## 9.3 Candidate Endpoints

### GET /api/candidate/election/{electionId}

```json




Response (200 OK):
{
  "candidates": [
    {
      "candidateId": 1,
      "name": "Jane Smith",
      "party": "Progressive Party",
      "symbol": "Tree",
      "voteCount": 0
    },
    {
      "candidateId": 2,
      "name": "Robert Brown",
      "party": "Conservative Alliance",
      "symbol": "Lion",
      "voteCount": 0
    }
  ]
}
```

### GET /api/candidate/profile/{candidateId}

json

```
Response (200 OK):
{
  "candidateId": 1,
  "name": "Jane Smith",
  "party": "Progressive Party",
  "symbol": "Tree",
  "electionId": 1,
  "voteCount": 45
}
```

---

## 9.4 Blockchain Endpoints

### GET /api/blockchain/chain

json

```
Request Headers:
Authorization: Bearer <JWT_TOKEN>

Response (200 OK):
{
  "chain": [
    {
      "index": 0,
      "timestamp": 1696400000000,
      "voteData": "Genesis Block",
      "previousHash": "0",
      "currentHash": "00000genesis123456789abcdef",
      "nonce": 0
    },
    {
      "index": 1,
      "timestamp": 1696435200000,
      "voteData": "V12345->Candidate3->Election1",
      "previousHash": "00000genesis123456789abcdef",
      "currentHash": "00000a8b9c7d6e5f4a3b2c1d0e9f8a7b",
      "nonce": 12345
    }
  ],
  "length": 2,
  "isValid": true
}
```

### GET /api/blockchain/validate

```
json

Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "isValid": true,
  "message": "Blockchain integrity verified. All blocks are valid.",
  "totalBlocks": 150
}
```

### GET /api/blockchain/block/{blockHash}

json

```
Response (200 OK):
{
  "index": 42,
  "timestamp": 1696435200000,
  "voteData": "V12345->Candidate3->Election1",
  "previousHash": "00000a8b9c7d6e5f4a3b2c1d0e9f8a7a",
  "currentHash": "00000a8b9c7d6e5f4a3b2c1d0e9f8a7b",
  "nonce": 12345
}
```

## 9.5 Results Endpoints

### GET /api/results/election/{electionId}

json

```
Response (200 OK):
{
  "electionId": 1,
  "electionName": "General Elections 2024",
  "status": "completed",
  "totalVotes": 150,
  "results": [
    {
      "candidateId": 1,
      "name": "Jane Smith",
      "party": "Progressive Party",
      "voteCount": 75,
      "percentage": 50.0,
      "rank": 1
    },
    {
      "candidateId": 2,
      "name": "Robert Brown",
      "party": "Conservative Alliance",
      "voteCount": 60,
      "percentage": 40.0,
      "rank": 2
    },
    {
      "candidateId": 3,
      "name": "Maria Garcia",
      "party": "Independent",
      "voteCount": 15,
      "percentage": 10.0,
      "rank": 3
    }
  ],
  "winner": {
    "candidateId": 1,
    "name": "Jane Smith",
    "party": "Progressive Party"
  }
}
```

### POST /api/results/calculate/{electionId}

json

```
Request Headers:
Authorization: Bearer <ADMIN_JWT_TOKEN>

Response (200 OK):
{
  "message": "Results calculated successfully",
  "electionId": 1,
  "totalVotesCounted": 150,
  "calculatedAt": "2024-11-01T18:30:00"
}
```

---

# 10. Security Implementation

## 10.1 Password Security

### Implementation in SecurityService.java

java

```java
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Service
public class SecurityService {

  private BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder(12);

  // Hash password during registration
  public String hashPassword(String plainPassword) {
    return passwordEncoder.encode(plainPassword);
  }

  // Verify password during login
  public boolean verifyPassword(String plainPassword, String hashedPassword) {
    return passwordEncoder.matches(plainPassword, hashedPassword);
  }
}
```

**Flow:**

1. User registers with password "MyPassword123"
2. System hashes it: `$2a$12$abcd...xyz` (60 characters)

3. Hashed password stored in database
4. During login, entered password is hashed and compared
5. If match → authentication successful

---

## 10.2 JWT Token Management

### Implementation in JWTUtil.java

```java
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import java.util.Date;

@Component
public class JWTUtil {
    private String SECRET_KEY = "mySecretKey12345"; // Use environment variable in production
    private long EXPIRATION_TIME = 86400000; // 24 hours in milliseconds

    // Generate JWT token
    public String generateToken(String voterId, String username) {
        return Jwts.builder()
                .setSubject(voterId)
                .claim("username", username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }

    // Validate token
    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    // Extract voterId from token
    public String extractVoterId(String token) {
        Claims claims = Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody();
        return claims.getSubject();
    }
}
```

**Usage Flow:**

1. User logs in successfully
2. Backend generates JWT token with voterId
3. Token sent to frontend in response
4. Frontend stores token (localStorage or sessionStorage)
5. For protected requests, frontend sends token in header:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

6. Backend validates token before processing request

## 10.3 Double-Voting Prevention

### Implementation in VoterService.java

```java
@Service
public class VoterService {

    @Autowired
    private VoterRepository voterRepository;

    public void castVote(String voterId, int candidateId, int electionId) {
        // Fetch voter from database
        Voter voter = voterRepository.findById(voterId)
                .orElseThrow(() -> new VoterNotFoundException("Voter not found"));

        // Check if voter has already voted
        if (voter.isVotedStatus()) {
            throw new AlreadyVotedException("You have already cast your vote!");
        }

        // Check if voter is approved
        if (!voter.isApprovedStatus()) {
            throw new UnauthorizedException("Your voter registration is not approved yet.");
        }

        // Proceed with vote casting...
        // (Blockchain block creation logic here)

        // Update voter's voted status
        voter.setVotedStatus(true);
        voterRepository.save(voter);
    }
}
```

**Flow:**

1. Voter attempts to cast vote
2. System queries database for voter's `votedStatus`
3. If `TRUE` → Reject with error message
4. If `FALSE` → Allow vote and update status to `TRUE`
5. No voter can vote twice for the same election

## 10.4 Input Validation and Sanitization

### Implementation in ValidationService.java

java

```java
@Service
public class ValidationService {

    // Validate email format
    public boolean isValidEmail(String email) {
        String emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
        return email.matches(emailRegex);
    }

    // Validate voter ID format
    public boolean isValidVoterId(String voterId) {
        return voterId != null && voterId.matches("^V[0-9]{5,10}$");
    }

    // Sanitize input to prevent SQL injection
    public String sanitizeInput(String input) {
        if (input == null) return null;
        return input.replaceAll("[^a-zA-Z0-9@._\\s-]", "");
    }

    // Validate password strength
    public boolean isStrongPassword(String password) {
        // At least 8 characters, 1 uppercase, 1 lowercase, 1 digit
        String passwordRegex = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).{8,}$";
        return password.matches(passwordRegex);
    }

    // Validate election is active
    public boolean isElectionActive(Election election) {
        Date now = new Date();
        return election.getStatus().equals("active")
                && now.after(election.getStartDate())
                && now.before(election.getEndDate());
    }
}
```

## 10.5 Spring Security Configuration

### Implementation in SecurityConfig.java

java

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable() // Disable CSRF for REST API
            .authorizeHttpRequests(auth -> auth
                // Public endpoints (no authentication required)
                .requestMatchers("/api/voter/register").permitAll()
                .requestMatchers("/api/voter/login").permitAll()
                .requestMatchers("/api/admin/login").permitAll()
                .requestMatchers("/api/candidate/election/**").permitAll()

                // Protected endpoints (authentication required)
                .requestMatchers("/api/voter/**").authenticated()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .requestMatchers("/api/blockchain/**").authenticated()
                .requestMatchers("/api/results/**").authenticated()

                // All other requests require authentication
                .anyRequest().authenticated()
            )
            .httpBasic(); // Use HTTP Basic or implement JWT filter

        return http.build();
    }
}
```

# 11. Testing Strategy

## 11.1 Unit Testing

### Example: BlockchainService Test

java

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;

class BlockchainServiceTest {

    private BlockchainService blockchainService;
    @BeforeEach
    void setUp() {
        blockchainService = new BlockchainService();
    }

    @Test
    void testGenesisBlockCreation() {
        Block genesisBlock = blockchainService.getGenesisBlock();
        assertNotNull(genesisBlock);
        assertEquals(0, genesisBlock.getIndex());
        assertEquals("0", genesisBlock.getPreviousHash());
    }

    @Test
    void testAddBlock() {
        String voteData = "V12345->Candidate1->Election1";
        Block newBlock = blockchainService.addBlock(voteData);

        assertNotNull(newBlock);
        assertEquals(1, newBlock.getIndex());
        assertNotNull(newBlock.getCurrentHash());
    }

    @Test
    void testChainValidation() {
        blockchainService.addBlock("V12345->Candidate1->Election1");
        blockchainService.addBlock("V12346->Candidate2->Election1");

        boolean isValid = blockchainService.validateChain();
        assertTrue(isValid);
    }

    @Test
    void testChainTampering() {
        blockchainService.addBlock("V12345->Candidate1->Election1");
        Block secondBlock = blockchainService.addBlock("V12346->Candidate2->Election1");

        // Tamper with block data
        secondBlock.setVoteData("TAMPERED_DATA");

        boolean isValid = blockchainService.validateChain();
        assertFalse(isValid); // Should detect tampering
    }
}
```

### Example: VoterService Test

java

```java
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

class VoterServiceTest {

    @Mock
    private VoterRepository voterRepository;

    @InjectMocks
    private VoterService voterService;

    @Test
    void testRegisterVoter() {
        Voter voter = new Voter();
        voter.setVoterId("V12345");
        voter.setName("John Doe");
        voter.setEmail("john@example.com");

        when(voterRepository.save(any(Voter.class))).thenReturn(voter);

        Voter result = voterService.registerVoter(voter);

        assertNotNull(result);
        assertEquals("V12345", result.getVoterId());
        verify(voterRepository, times(1)).save(voter);
    }
    @Test
    void testPreventDoubleVoting() {
        Voter voter = new Voter();
        voter.setVoterId("V12345");
        voter.setVotedStatus(true); // Already voted

        when(voterRepository.findById("V12345")).thenReturn(Optional.of(voter));

        assertThrows(AlreadyVotedException.class, () -> {
            voterService.castVote("V12345", 1, 1);
        });
    }
}
```

---

## 11.2 Integration Testing

### Example: API Integration Test

```java
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@SpringBootTest
@AutoConfigureMockMvc
class VoterControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testVoterRegistration() throws Exception {
        String voterJson = "{\"voterId\":\"V12345\",\"name\":\"John Doe\","
                + "\"email\":\"john@example.com\",\"password\":\"Pass123\"}";

        mockMvc.perform(post("/api/voter/register")
                .contentType("application/json")
                .content(voterJson))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.voterId").value("V12345"));
    }
    @Test
    void testVoterLogin() throws Exception {
        String loginJson = "{\"voterId\":\"V12345\",\"password\":\"Pass123\"}";

        mockMvc.perform(post("/api/voter/login")
                .contentType("application/json")
                .content(loginJson))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.token").exists());
    }
}
```

---

### 11.3 Test Cases Checklist

**Blockchain Tests:**

- ✓ Genesis block creation
- ✓ Block addition to chain
- ✓ Hash calculation correctness
- ✓ Chain validation (valid chain)
- ✓ Chain validation (tampered chain)
- ✓ Block retrieval by hash

**Voter Module Tests:**

- ✓ Voter registration (valid data)
- ✓ Voter registration (duplicate email)
- ✓ Voter login (correct credentials)
- ✓ Voter login (incorrect credentials)
- ✓ Vote casting (first time)
- ✓ Vote casting (double-voting attempt)
- ✓ Vote verification with receipt

**Admin Module Tests:**

- ✓ Election creation
- ✓ Voter approval
- ✓ Candidate addition
- ✓ Election start/stop
- ✓ Result calculation

- ✓ Result calculation

**Security Tests:**

- ✓ Password hashing
- ✓ JWT token generation
- ✓ JWT token validation (valid token)
- ✓ JWT token validation (expired token)
- ✓ Unauthorized access prevention

---

# 12. Deployment Guide

## 12.1 Local Development Setup

### Step 1: Install Prerequisites

```bash
# Install Java 17
sudo apt install openjdk-17-jdk

# Install Maven
sudo apt install maven

# Install MySQL
sudo apt install mysql-server

# Verify installations
java -version
mvn -version
mysql --version
```

### Step 2: Configure Database

```sql
-- Login to MySQL
mysql -u root -p

-- Create database
CREATE DATABASE blockchain_voting;

-- Create user (optional)
CREATE USER 'voting_admin'@'localhost' IDENTIFIED BY 'secure_password';
GRANT ALL PRIVILEGES ON blockchain_voting.* TO 'voting_admin'@'localhost';
FLUSH PRIVILEGES;
FLUSH PRIVILEGES;
```

### Step 3: Configure application.properties

properties

```properties
# Server Configuration
server.port=8080

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/blockchain_voting
spring.datasource.username=voting_admin
spring.datasource.password=secure_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# JWT Configuration
jwt.secret=mySecretKey123456789
jwt.expiration=86400000

# Logging
logging.level.com.voting.blockchain=DEBUG
```

### Step 4: Build and Run

bash

```bash
# Navigate to project directory
cd blockchain-voting-system

# Clean and build project
mvn clean install

# Run application
mvn spring-boot:run

# Or run JAR file
java -jar target/blockchain-voting-0.0.1-SNAPSHOT.jar
```

---

## 12.2 Production Deployment (AWS EC2)

### Step 1: Set Up EC2 Instance

bash

```
# Launch Ubuntu EC2 instance (t2.medium or higher)
# Configure security group:
# - Allow HTTP (port 80)
# - Allow HTTPS (port 443)
# - Allow Custom TCP (port 8080 for Spring Boot)
# - Allow MySQL (port 3306) only from application security group

# SSH into instance
ssh -i your-key.pem ubuntu@ec2-xx-xxx-xxx-xxx.compute.amazonaws.com
```

### Step 2: Install Required Software

bash

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Java 17
sudo apt install openjdk-17-jdk -y

# Install MySQL
sudo apt install mysql-server -y

# Install Maven
sudo apt install maven -y

# Install Git
sudo apt install git -y
```

### Step 3: Configure MySQL for Production

bash

```
# Secure MySQL installation
sudo mysql_secure_installation

# Login and create database
sudo mysql -u root -p

CREATE DATABASE blockchain_voting;
CREATE USER 'voting_admin'@'localhost' IDENTIFIED BY 'STRONG_PASSWORD_HERE';
GRANT ALL PRIVILEGES ON blockchain_voting.* TO 'voting_admin'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

### Step 4: Deploy Application

bash

```
# Clone repository
git clone https://github.com/your-repo/blockchain-voting-system.git
cd blockchain-voting-system

# Update application.properties for production
nano src/main/resources/application.properties

# Build application
mvn clean package -DskipTests

# Run application in background
nohup java -jar target/blockchain-voting-0.0.1-SNAPSHOT.jar > app.log 2>&1 &

# Check if running
ps aux | grep java
```

### Step 5: Set Up Nginx Reverse Proxy (Optional)

bash

```
# Install Nginx
sudo apt install nginx -y

# Configure Nginx
sudo nano /etc/nginx/sites-available/voting-app

# Add configuration:
server {
    listen 80;
    server_name your-domain.com;

    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

# Enable site
sudo ln -s /etc/nginx/sites-available/voting-app /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

### Step 6: Set Up SSL Certificate (Let's Encrypt)

bash

```
# Install Certbot
sudo apt install certbot python3-certbot-nginx -y

# Obtain SSL certificate
sudo certbot --nginx -d your-domain.com

# Auto-renewal is set up automatically
```

---

## 12.3 Environment Variables (Production Best Practice)

bash

```
# Create .env file
nano .env

# Add sensitive data
DB_URL=jdbc:mysql://localhost:3306/blockchain_voting
DB_USERNAME=voting_admin
DB_PASSWORD=STRONG_PASSWORD
JWT_SECRET=VERY_LONG_RANDOM_SECRET_KEY_HERE

# Load environment variables
export $(cat .env | xargs)

# Update application.properties to use environment variables:
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
jwt.secret=${JWT_SECRET}
```

---

## 12.4 Monitoring and Maintenance

### Health Check Endpoint

```java
@RestController
@RequestMapping("/api/health")
public class HealthController {

    @GetMapping
    public ResponseEntity<String> healthCheck() {
        return ResponseEntity.ok("Application is running");
    }
}
```

### Logging Setup

properties

```properties
# application.properties
logging.file.name=logs/application.log
logging.level.root=INFO
logging.level.com.voting.blockchain=DEBUG
logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %msg%n
```

## Database Backup Script

bash

```bash
#!/bin/bash
# backup-db.sh

BACKUP_DIR="/home/ubuntu/backups"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/blockchain_voting_$DATE.sql"

# Create backup
mysqldump -u voting_admin -p blockchain_voting > $BACKUP_FILE

# Compress backup
gzip $BACKUP_FILE

# Delete backups older than 30 days
find $BACKUP_DIR -name "*.gz" -mtime +30 -delete

echo "Backup completed: $BACKUP_FILE.gz"
```

## Schedule Backups with Cron

bash

```bash
# Edit crontab
crontab -e
```

```
# Add daily backup at 2 AM
0 2 * * * /home/ubuntu/backup-db.sh
```

---

## 13. Common Issues and Solutions

### Issue 1: MySQL Connection Refused

**Solution:**

```bash
# Check MySQL status
sudo systemctl status mysql

# Start MySQL if stopped
sudo systemctl start mysql

# Verify port 3306 is open
netstat -an | grep 3306
```

### Issue 2: Port 8080 Already in Use

**Solution:**

```bash
# Find process using port 8080
lsof -i :8080

# Kill the process
kill -9 <PID>

# Or change port in application.properties
server.port=8081
```

### Issue 3: JWT Token Expiration

**Solution:**

- Implement token refresh mechanism

- Store refresh token in database
- Create /api/auth/refresh endpoint

### Issue 4: Blockchain Validation Failure

**Solution:**

- Check if any block data was modified
- Verify hash calculations are consistent
- Rebuild chain from database if necessary

---

## 14. Future Enhancements

1. **Zero-Knowledge Proofs (ZKPs)** - Advanced privacy preservation
2. **Homomorphic Encryption** - Vote tallying without decryption
3. **Multi-Signature Wallets** - Enhanced admin control
4. **Proof-of-Work** - Mining simulation for block validation
5. **Smart Contracts** - Automated election rules
6. **Mobile App** - Android/iOS applications
7. **Biometric Authentication** - Fingerprint/Face recognition
8. **Real-time Analytics Dashboard** - Live vote tracking
9. **Multi-Language Support** - Internationalization
10. **Accessibility Features** - Screen reader support, high contrast mode

---

## 15. References and Resources

### Documentation

- Spring Boot: https://spring.io/projects/spring-boot
- Spring Security: https://spring.io/projects/spring-security
- MySQL: https://dev.mysql.com/doc/
- JWT: https://jwt.io/introduction
- Blockchain Basics: https://www.blockchain.com/learning-portal

### Research Papers

- Your Project Paper: "Blockchain-Based E-Voting Systems: A Systematic Literature Review"

- ACB-Vote: Efficient, Flexible, and Privacy-Preserving Blockchain-Based E-Voting
- HAC-Bchain: A Secure and Scalable Blockchain-Shard Based E-Voting System

### GitHub Resources

- Spring Boot Examples: https://github.com/spring-projects/spring-boot
- Blockchain Java Implementation: https://github.com/topics/blockchain-java
- E-Voting Projects: https://github.com/topics/e-voting

---

## 16. Code Examples - Core Classes

### 16.1 Block.java (Complete Implementation)

```java
package com.voting.blockchain.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.security.MessageDigest;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Block {
    private int index;
    private long timestamp;
    private String voteData;      // Format: "VoterId->CandidateId->ElectionId"
    private String previousHash;
    private String currentHash;
    private int nonce;            // For proof-of-work (optional)

    // Constructor without hash (hash will be calculated)
    public Block(int index, String voteData, String previousHash) {
        this.index = index;
        this.timestamp = System.currentTimeMillis();
        this.voteData = voteData;
        this.previousHash = previousHash;
        this.nonce = 0;
        this.currentHash = calculateHash();
    }

    // Calculate hash for this block
    public String calculateHash() {
        try {
```

```
        try {
            String data = index + Long.toString(timestamp) + voteData + previousHash + nonce;
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(data.getBytes("UTF-8"));

            // Convert byte array to hex string
            StringBuilder hexString = new StringBuilder();
            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (Exception e) {
            throw new RuntimeException("Error calculating hash", e);
        }
    }

    // Optional: Mine block with proof-of-work (difficulty = number of leading zeros)
    public void mineBlock(int difficulty) {
        String target = new String(new char[difficulty]).replace('\0', '0');
        while (!currentHash.substring(0, difficulty).equals(target)) {
            nonce++;
            currentHash = calculateHash();
        }
        System.out.println("Block mined: " + currentHash);
    }
}
```

## 16.2 Blockchain.java (Complete Implementation)

java

```
package com.voting.blockchain.model;

import lombok.Data;
import java.util.ArrayList;
import java.util.List;


@Data
public class Blockchain {
    private List<Block> chain;
    private int difficulty;  // For mining (optional)

    public Blockchain() {
        this.chain = new ArrayList<>();
        this.difficulty = 2;  // Number of leading zeros required in hash
        this.difficulty = 2; // Number of leading zeros required in hash
        createGenesisBlock();
    }

    // Create the first block in the chain
    private void createGenesisBlock() {
        Block genesisBlock = new Block(0, "Genesis Block", "0");
        chain.add(genesisBlock);
    }

    // Get the latest block in the chain
    public Block getLatestBlock() {
        return chain.get(chain.size() - 1);
    }

    // Add a new block to the chain
    public Block addBlock(String voteData) {
        Block previousBlock = getLatestBlock();
        Block newBlock = new Block(
            previousBlock.getIndex() + 1,
            voteData,
            previousBlock.getCurrentHash()
        );

        // Optional: Mine the block
        // newBlock.mineBlock(difficulty);

        chain.add(newBlock);
        return newBlock;
    }

    // Validate the entire blockchain
    public boolean isChainValid() {
        for (int i = 1; i < chain.size(); i++) {
            Block currentBlock = chain.get(i);
            Block previousBlock = chain.get(i - 1);

            // Check if current block's hash is correct
            if (!currentBlock.getCurrentHash().equals(currentBlock.calculateHash())) {

                System.out.println("Current hash is invalid at block " + i);
                return false;
            }

            // Check if current block's previousHash matches previous block's hash
            if (!currentBlock.getPreviousHash().equals(previousBlock.getCurrentHash())) {
                System.out.println("Previous hash is invalid at block " + i);
                return false;
            }
            }
        }
        return true;
    }

    // Get block by hash
    public Block getBlockByHash(String hash) {
        return chain.stream()
                .filter(block -> block.getCurrentHash().equals(hash))
                .findFirst()
                .orElse(null);
    }

    // Get chain length
    public int getChainLength() {
        return chain.size();
    }
}
```

---

## 16.3 BlockchainService.java (Complete Implementation)

```
java

package com.voting.blockchain.service;

import com.voting.blockchain.model.Block;
import com.voting.blockchain.model.Blockchain;
import com.voting.blockchain.model.BlockchainBlock;
import com.voting.blockchain.repository.BlockchainBlockRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.util.List;

@Service
public class BlockchainService {


    private Blockchain blockchain;

    @Autowired
    private BlockchainBlockRepository blockchainBlockRepository;

    @PostConstruct
    public void init() {
        blockchain = new Blockchain();
```

```java
    loadBlockchainFromDatabase();
}

// Load existing blockchain from database on startup
private void loadBlockchainFromDatabase() {
    List<BlockchainBlock> savedBlocks = blockchainBlockRepository.findAllByOrderByBlockIndexAsc();
    if (!savedBlocks.isEmpty()) {
        blockchain.getChain().clear();
        for (BlockchainBlock bb : savedBlocks) {
            Block block = new Block(
                bb.getBlockIndex(),
                bb.getTimestamp(),
                bb.getVoteData(),
                bb.getPreviousHash(),
                bb.getCurrentHash(),
                bb.getNonce()
            );
            blockchain.getChain().add(block);
        }
    }
}

// Add vote to blockchain
public Block addVoteToBlockchain(String voteData) {
    Block newBlock = blockchain.addBlock(voteData);

    // Persist to database
    BlockchainBlock blockEntity = new BlockchainBlock();
    blockEntity.setBlockIndex(newBlock.getIndex());
    blockEntity.setTimestamp(newBlock.getTimestamp());
    blockEntity.setVoteData(newBlock.getVoteData());
    blockEntity.setPreviousHash(newBlock.getPreviousHash());
    blockEntity.setCurrentHash(newBlock.getCurrentHash());
    blockEntity.setNonce(newBlock.getNonce());

    blockchainBlockRepository.save(blockEntity);

    return newBlock;
}


// Validate blockchain integrity
public boolean validateBlockchain() {
    return blockchain.isChainValid();
}

// Get entire blockchain
public Blockchain getBlockchain() {
        return blockchain;
    }

    // Get block by hash
    public Block getBlockByHash(String hash) {
        return blockchain.getBlockByHash(hash);
    }

    // Get genesis block
    public Block getGenesisBlock() {
        return blockchain.getChain().get(0);
    }

    // Get chain length
    public int getChainLength() {
        return blockchain.getChainLength();
    }
}
```

---

## 16.4 VoterService.java (Complete Implementation)

```java
package com.voting.blockchain.service;

import com.voting.blockchain.exception.AlreadyVotedException;
import com.voting.blockchain.exception.UnauthorizedException;
import com.voting.blockchain.exception.VoterNotFoundException;
import com.voting.blockchain.model.*;
import com.voting.blockchain.repository.VoterRepository;
import com.voting.blockchain.repository.VoteRepository;
import com.voting.blockchain.repository.CandidateRepository;
import com.voting.blockchain.util.ReceiptGenerator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Date;
import java.util.Optional;

@Service
public class VoterService {

    @Autowired
    private VoterRepository voterRepository;

    @Autowired
    private VoteRepository voteRepository;

    @Autowired
    private CandidateRepository candidateRepository;

    @Autowired
    private BlockchainService blockchainService;

    @Autowired
    private SecurityService securityService;

    @Autowired
    private ValidationService validationService;

    // Register new voter
    @Transactional
    public Voter registerVoter(Voter voter) {
        // Validate input
        if (!validationService.isValidEmail(voter.getEmail())) {
            throw new IllegalArgumentException("Invalid email format");
        }

        if (!validationService.isStrongPassword(voter.getPassword())) {
            throw new IllegalArgumentException("Password must be at least 8 characters with uppercase, lowercase, and dig
        }

        // Check if email already exists
        if (voterRepository.findByEmail(voter.getEmail()).isPresent()) {
            throw new IllegalArgumentException("Email already registered");
        }

        // Hash password
        voter.setPassword(securityService.hashPassword(voter.getPassword()));
        voter.setVotedStatus(false);
        voter.setApprovedStatus(false);
        voter.setRegistrationTimestamp(new Date());

        return voterRepository.save(voter);

    }

    // Authenticate voter
    public Voter authenticateVoter(String voterId, String password) {
        Voter voter = voterRepository.findById(voterId)
                .orElseThrow(() -> new VoterNotFoundException("Voter not found"));

        if (!securityService.verifyPassword(password, voter.getPassword())) {
            throw new UnauthorizedException("Invalid credentials");
        }

        return voter;
    }

    // Cast vote
    @Transactional
    public VoteCastResponse castVote(String voterId, int candidateId, int electionId) {
        // Fetch voter
        Voter voter = voterRepository.findById(voterId)
                .orElseThrow(() -> new VoterNotFoundException("Voter not found"));

        // Check if already voted
        if (voter.isVotedStatus()) {
            throw new AlreadyVotedException("You have already cast your vote!");
        }

        // Check if approved
        if (!voter.isApprovedStatus()) {
            throw new UnauthorizedException("Your voter registration is pending approval");
        }

        // Validate candidate exists
        Candidate candidate = candidateRepository.findById(candidateId)
                .orElseThrow(() -> new IllegalArgumentException("Invalid candidate"));

        // Create vote data for blockchain
        String voteData = String.format("%s->%d->%d", voterId, candidateId, electionId);

        // Add to blockchain
        Block block = blockchainService.addVoteToBlockchain(voteData);

        // Generate receipt
        String receiptId = ReceiptGenerator.generateReceipt();

        // Save vote record to database
        Vote vote = new Vote();
        vote.setVoteId("VOTE-" + System.currentTimeMillis());

        vote.setVoterId(voterId);
        vote.setCandidateId(candidateId);
        vote.setElectionId(electionId);
        vote.setReceiptId(receiptId);
        vote.setVoteTimestamp(new Date());
        voteRepository.save(vote);

        // Update voter status
        voter.setVotedStatus(true);
        voterRepository.save(voter);

        // Update candidate vote count
        candidate.setVoteCount(candidate.getVoteCount() + 1);
        candidateRepository.save(candidate);

        // Prepare response
        VoteCastResponse response = new VoteCastResponse();
        response.setMessage("Vote cast successfully");
        response.setReceiptId(receiptId);
        response.setBlockHash(block.getCurrentHash());
        response.setTimestamp(block.getTimestamp());
        response.setVerificationMessage("Your vote has been recorded on the blockchain. Keep this receipt for verification."

        return response;
    }

    // Verify vote using receipt
    public VoteVerificationResponse verifyVote(String receiptId) {
        Vote vote = voteRepository.findByReceiptId(receiptId)
                .orElseThrow(() -> new IllegalArgumentException("Invalid receipt ID"));

        // Search blockchain for this vote
        String voteData = String.format("%s->%d->%d", vote.getVoterId(), vote.getCandidateId(), vote.getElectionId());

        Block foundBlock = null;
        for (Block block : blockchainService.getBlockchain().getChain()) {
            if (block.getVoteData().equals(voteData)) {
                foundBlock = block;
                break;
            }
        }

        VoteVerificationResponse response = new VoteVerificationResponse();
        if (foundBlock != null) {
            response.setVerified(true);
            response.setVoteId(vote.getVoteId());
            response.setReceiptId(receiptId);

            response.setBlockIndex(foundBlock.getIndex());
            response.setBlockHash(foundBlock.getCurrentHash());
            response.setTimestamp(foundBlock.getTimestamp());
            response.setCandidateId(vote.getCandidateId());
            response.setElectionId(vote.getElectionId());
            response.setMessage("Your vote is verified and securely recorded on the blockchain.");
        } else {
            response.setVerified(false);
```

```java
            response.setMessage("Vote not found in blockchain. Please contact support.");
        }

        return response;
    }

    // Get voter profile
    public Voter getVoterProfile(String voterId) {
        return voterRepository.findById(voterId)
                .orElseThrow(() -> new VoterNotFoundException("Voter not found"));
    }
}
```

---

## 16.5 VoterController.java (Complete Implementation)

java

```java
package com.voting.blockchain.controller;

import com.voting.blockchain.dto.LoginRequestDTO;
import com.voting.blockchain.dto.VoteCastDTO;
import com.voting.blockchain.model.Voter;
import com.voting.blockchain.model.VoteCastResponse;
import com.voting.blockchain.model.VoteVerificationResponse;
import com.voting.blockchain.service.VoterService;
import com.voting.blockchain.util.JWTUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/api/voter")
@CrossOrigin(origins = "*")

public class VoterController {

    @Autowired
    private VoterService voterService;

    @Autowired
    private JWTUtil jwtUtil;

    // Register voter
// Register voter
@PostMapping("/register")
public ResponseEntity<?> registerVoter(@RequestBody Voter voter) {
    try {
        Voter registeredVoter = voterService.registerVoter(voter);
        Map<String, Object> response = new HashMap<>();
        response.put("message", "Voter registered successfully. Awaiting admin approval.");
        response.put("voterId", registeredVoter.getVoterId());
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    } catch (Exception e) {
        Map<String, String> error = new HashMap<>();
        error.put("error", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
    }
}

// Login voter
@PostMapping("/login")
public ResponseEntity<?> loginVoter(@RequestBody LoginRequestDTO loginRequest) {
    try {
        Voter voter = voterService.authenticateVoter(
            loginRequest.getVoterId(),
            loginRequest.getPassword()
        );

        // Generate JWT token
        String token = jwtUtil.generateToken(voter.getVoterId(), voter.getName());

        Map<String, Object> response = new HashMap<>();
        response.put("message", "Login successful");
        response.put("token", token);
        response.put("voterId", voter.getVoterId());
        response.put("name", voter.getName());
        response.put("votedStatus", voter.isVotedStatus());

        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, String> error = new HashMap<>();
        error.put("error", e.getMessage());

        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(error);
    }
}

// Cast vote
@PostMapping("/vote")
public ResponseEntity<?> castVote(
        @RequestHeader("Authorization") String authHeader,
        @RequestBody VoteCastDTO voteCastDTO) {
```

```
        @RequestBody VoteCastDTO voteCastDTO) {
      try {
        // Extract and validate JWT token
        String token = authHeader.substring(7); // Remove "Bearer "
        if (!jwtUtil.validateToken(token)) {
          return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
              .body(Map.of("error", "Invalid token"));
        }

        String voterId = jwtUtil.extractVoterId(token);

        VoteCastResponse response = voterService.castVote(
          voterId,
          voteCastDTO.getCandidateId(),
          voteCastDTO.getElectionId()
        );

        return ResponseEntity.ok(response);
      } catch (Exception e) {
        Map<String, String> error = new HashMap<>();
        error.put("error", e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
      }
    }

    // Verify vote
    @GetMapping("/verify/{receiptId}")
    public ResponseEntity<?> verifyVote(
        @RequestHeader("Authorization") String authHeader,
        @PathVariable String receiptId) {
      try {
        // Validate token
        String token = authHeader.substring(7);
        if (!jwtUtil.validateToken(token)) {
          return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
              .body(Map.of("error", "Invalid token"));
        }

        VoteVerificationResponse response = voterService.verifyVote(receiptId);

        return ResponseEntity.ok(response);
      } catch (Exception e) {
        Map<String, String> error = new HashMap<>();
        error.put("error", e.getMessage());
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
      }
    }

    // Get voter profile
    // Get voter profile
    @GetMapping("/profile")
    public ResponseEntity<?> getProfile(@RequestHeader("Authorization") String authHeader) {
        try {
            String token = authHeader.substring(7);
            if (!jwtUtil.validateToken(token)) {
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                        .body(Map.of("error", "Invalid token"));
            }

            String voterId = jwtUtil.extractVoterId(token);
            Voter voter = voterService.getVoterProfile(voterId);

            // Remove password from response
            voter.setPassword(null);

            return ResponseEntity.ok(voter);
        } catch (Exception e) {
            Map<String, String> error = new HashMap<>();
            error.put("error", e.getMessage());
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
        }
    }
}
```

## 16.6 ReceiptGenerator.java (Utility Class)

java

```
package com.voting.blockchain.util;

import java.security.SecureRandom;
import java.util.Base64;

public class ReceiptGenerator {

    private static final SecureRandom secureRandom = new SecureRandom();
    private static final SecureRandom secureRandom = new SecureRandom();
    private static final Base64.Encoder base64Encoder = Base64.getUrlEncoder();

    // Generate unique receipt ID
    public static String generateReceipt() {
        byte[] randomBytes = new byte[24];
        secureRandom.nextBytes(randomBytes);
        String encoded = base64Encoder.encodeToString(randomBytes);

        // Format: RCP-<random_string>
        return "RCP-" + encoded.substring(0, 24).replace("=", "");
    }

    // Alternative format with timestamp
    public static String generateReceiptWithTimestamp() {
        long timestamp = System.currentTimeMillis();
        byte[] randomBytes = new byte[16];
        secureRandom.nextBytes(randomBytes);
        String encoded = base64Encoder.encodeToString(randomBytes);

        return String.format("RCP-%d-%s", timestamp, encoded.substring(0, 16).replace("=", ""));
    }
}
```

## 16.7 Response DTOs

```
java
```

```
// VoteCastResponse.java
package com.voting.blockchain.model;

import lombok.Data;

@Data
public class VoteCastResponse {
    private String message;
    private String receiptId;
    private String blockHash;
    private long timestamp;
    private String verificationMessage;
}

// VoteVerificationResponse.java
package com.voting.blockchain.model;

import lombok.Data;

@Data
public class VoteVerificationResponse {
    private boolean verified;
    private String voteId;
    private String receiptId;
    private int blockIndex;
    private String blockHash;
    private long timestamp;
    private int candidateId;
    private int electionId;
    private String message;
}
```

# 17. Frontend Integration Example

## 17.1 Voter Registration Page (React)

```javascript
// VoterRegistration.jsx
import React, { useState } from 'react';
import axios from 'axios';

const VoterRegistration = () => {
  const [formData, setFormData] = useState({
    voterId: '',
    name: '',
    name: '',
    email: '',
    password: '',
    age: '',
    phone: ''
  });
  const [message, setMessage] = useState('');

  const handleChange = (e) => {

    setFormData({
      ...formData,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post(
        'http://localhost:8080/api/voter/register',
        formData
      );
      setMessage(response.data.message);
      alert('Registration successful! Please wait for admin approval.');
    } catch (error) {
      setMessage(error.response?.data?.error || 'Registration failed');
      alert(message);
    }
  };

  return (
    <div className="registration-container">
      <h2>Voter Registration</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="voterId"
          placeholder="Voter ID"
          value={formData.voterId}
          onChange={handleChange}
          required
        />
        <input
          type="text"
          name="name"
          placeholder="Full Name"
          value={formData.name}
          onChange={handleChange}
          onChange={handleChange}
          required
        />
        <input
          type="email"
          name="email"
          placeholder="Email"
          value={formData.email}
          onChange={handleChange}

          required
        />
        <input
          type="password"
          name="password"
          placeholder="Password"
          value={formData.password}
          onChange={handleChange}
          required
        />
        <input
          type="number"
          name="age"
          placeholder="Age"
          value={formData.age}
          onChange={handleChange}
          required
        />
        <input
          type="tel"
          name="phone"
          placeholder="Phone Number"
          value={formData.phone}
          onChange={handleChange}
          required
        />
        <button type="submit">Register</button>
      </form>
    </div>
  );
};

export default VoterRegistration;
```

## 17.2 Voting Page (React)

```
javascript
```

```javascript
// VotingPage.jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const VotingPage = () => {
  const [candidates, setCandidates] = useState([]);
  const [selectedCandidate, setSelectedCandidate] = useState(null);
  const [receipt, setReceipt] = useState(null);
  const electionId = 1; // Hardcoded for example

  useEffect(() => {
    fetchCandidates();
  }, []);

  const fetchCandidates = async () => {
    try {
      const response = await axios.get(
        `http://localhost:8080/api/candidate/election/${electionId}`
      );
      setCandidates(response.data.candidates);
    } catch (error) {
      console.error('Error fetching candidates:', error);
    }
  };

  const castVote = async () => {
    if (!selectedCandidate) {
      alert('Please select a candidate');
      return;
    }

    const token = localStorage.getItem('token');
    try {
      const response = await axios.post(
        'http://localhost:8080/api/voter/vote',
        {
          candidateId: selectedCandidate,
          electionId: electionId
        },
        {
          headers: {
            Authorization: `Bearer ${token}`
          }
        }
      );

      setReceipt(response.data);
      setReceipt(response.data);
      alert('Vote cast successfully! Your receipt: ' + response.data.receiptId);
    } catch (error) {
      alert(error.response?.data?.error || 'Failed to cast vote');
    }
  };

  return (
    <div className="voting-container">

      <h2>Cast Your Vote</h2>
      <div className="candidates-list">
        {candidates.map((candidate) => (
          <div
            key={candidate.candidateId}
            className={`candidate-card ${selectedCandidate === candidate.candidateId ? 'selected' : ''}`}
            onClick={() => setSelectedCandidate(candidate.candidateId)}
          >
            <h3>{candidate.name}</h3>
            <p>Party: {candidate.party}</p>
            <p>Symbol: {candidate.symbol}</p>
          </div>
        ))}
      </div>
      <button onClick={castVote} className="vote-button">
        Submit Vote
      </button>

      {receipt && (
        <div className="receipt">
          <h3>Vote Receipt</h3>
          <p><strong>Receipt ID:</strong> {receipt.receiptId}</p>
          <p><strong>Block Hash:</strong> {receipt.blockHash}</p>
          <p><strong>Timestamp:</strong> {new Date(receipt.timestamp).toLocaleString()}</p>
          <p className="verification-msg">{receipt.verificationMessage}</p>
        </div>
      )}
    </div>
  );
};

export default VotingPage;
```

## 18. Project Checklist

### Development Checklist

□ Java 17+ installed

□ MySQL installed and configured

□ Maven installed

□ IDE setup (IntelliJ/Eclipse)

□ Spring Boot project created

□ Database schema created

□ Entity classes implemented

□ Repository interfaces created

□ Service layer implemented

□ Controller layer implemented

□ Blockchain logic implemented

□ Security configured (JWT, BCrypt)

□ API endpoints tested

□ Frontend developed

□ Frontend-backend integration complete

□ Unit tests written

□ Integration tests written

□ Documentation complete

### Deployment Checklist

□ Production database configured

□ Environment variables set

□ Application properties updated for production

□ SSL certificate installed

□ Reverse proxy configured (Nginx)

□ Firewall rules configured

□ Database backup script created

□ Cron jobs scheduled

□ Logging configured

□ Health check endpoint tested

□ Performance testing done

□ Security audit completed

---

## 19. Conclusion

This comprehensive documentation provides everything needed to build a blockchain-based e-voting system with E2E-V integration in Java. The system ensures:

✅ **Security**: Password hashing, JWT authentication, tamper-proof blockchain ✅ **Transparency**: Public blockchain verification, audit trails ✅ **Integrity**: Immutable vote records, chain validation ✅ **Privacy**: