**Working of the project.**

## 1.Compile and Run

## i)truffle compile –all ii) truffle migrate --reset --network development iii)truffle console –network development

## 2. Contract Setup & Registration Phase

**const** instance = **await** Voting.deployed()

**await** instance.getPhase()*// Check initial phase (should be Registration=0)*

*// Register voters (admin=accounts[0])*

**await** instance.registerVoter(accounts[1], {from: accounts[0]})

**await** instance.registerVoter(accounts[2], {from: accounts[0]})

*// Verify registration*

**await** instance.isRegistered(accounts[1])

## 3. Voting Phase

*// Advance to Voting phase (Phase 1)*

**await** instance.changePhase(1, {from: accounts[0]})

*// Cast votes from registered accounts*

**await** instance.vote(0, {from: accounts[1]}) *// Alice*

**await** instance.vote(1, {from: accounts[2]}) *// Bob*

*// Verify votes*

**await** instance.hasVoted(accounts[1])

**await** instance.getCandidate(0) *// Check Alice's count*

## 

## 4. Results Phase

*// End election (Phase 2)*

**await** instance.changePhase(2, {from: accounts[0]})

*// Get winner*

**const** winner = **await** instance.getWinner()

console.log("Winner:", winner) *// Should show candidate with most votes*

## Verification Helpers

*// Check all candidates*

**const** candidateCount = **await** instance.getCandidatesCount()

**for**(**let** i = 0; i < candidateCount; i++) {

console.log(**await** instance.getCandidate(i))

}

*// Check current phase (0=Reg, 1=Voting, 2=Ended)*

**await** instance.getPhase()

--------------------------------------------------------------------------------------------------------------------------------------  
signatures:

undefined

truffle(development)> console.log(sig1)

0xcc22a0dab63dfe584102267ed5f552d4d576e620f7ffb09dd9f8ccb5550c47372bf111f5c1cd245e168e20d69751d78ecdb5d0069f00c791f0317fd327c8f9f000

undefined

truffle(development)> console.log(sig2)

### 0xdb91157e1270074810d2c10b5d2c72bf3585f71dae96206e2066b4b152a743340fdad91d7647514b a01a96e5e0d7f2ca2d4a35247462a74417e34bda0a4afea900 -------------------------------------- **1. Compile and Run:**

bash

CopyEdit

# Compile all contracts

truffle compile --all

# Deploy contracts (reset the deployment)

truffle migrate --reset --network development

# Open Truffle console for interaction

truffle console --network development

### **2. Contract Setup & Registration Phase:**

javascript

CopyEdit

// Get the deployed instance of the Voting contract

const instance = await Voting.deployed();

// Check the current phase (should return 0 for Registration)

await instance.getPhase();

// Register voters (admin is accounts[0])

await instance.registerVoter(accounts[1], {from: accounts[0]});

await instance.registerVoter(accounts[2], {from: accounts[0]});

// Verify that accounts[1] is registered

await instance.isRegistered(accounts[1]);

### **3. Voting Phase:**

javascript

CopyEdit

// Change phase to Voting (Phase 1)

await instance.changePhase(1, {from: accounts[0]});

// Cast votes (accounts[1] votes for candidate 0, accounts[2] votes for candidate 1)

await instance.vote(0, {from: accounts[1]}); // Alice votes

await instance.vote(1, {from: accounts[2]}); // Bob votes

// Verify that accounts[1] has voted

await instance.hasVoted(accounts[1]);

// Check the vote count for candidate 0 (Alice)

await instance.getCandidate(0);

### **4. Results Phase:**

javascript

CopyEdit

// Change phase to Ended (Phase 2)

await instance.changePhase(2, {from: accounts[0]});

// Get and log the winner of the election

const winner = await instance.getWinner();

console.log("Winner:", winner); // Should print the candidate with the most votes

### **5. Verification Helpers:**

javascript

CopyEdit

// Check the number of candidates

const candidateCount = await instance.getCandidatesCount();

// Loop through all candidates and print their details

for (let i = 0; i < candidateCount; i++) {

console.log(await instance.getCandidate(i));

}

// Check the current phase (0 = Registration, 1 = Voting, 2 = Ended)

await instance.getPhase();

### **6. Signature Debugging:**

javascript

CopyEdit

// Output signatures for debugging

console.log(sig1); // Example: 0xcc22a0dab63dfe...

console.log(sig2); // Example: 0xdb91157e1270...

-------------------------------------------------------------------------------------------------------------------------------------  
To thoroughly test the **backend functionality** of your blockchain voting system, here are the commands you can use in the **Truffle console** to check various functions and ensure the system behaves as expected.

### **1. Check Contract Deployment and Phase Management:**

* **Check Current Phase (Should be 0 for Registration initially):**

const instance = await Voting.deployed();

const phase = await instance.getPhase();

console.log("Current Phase:", phase.toString());

* **Change Phase to Voting (Phase 1) as Admin (accounts[0]):**

await instance.changePhase(1, {from: accounts[0]});

const phaseAfterVoting = await instance.getPhase();

console.log("Phase after changing to Voting:", phaseAfterVoting.toString());

* **Change Phase to Ended (Phase 2) as Admin (accounts[0]):**

await instance.changePhase(2, {from: accounts[0]});

const phaseAfterEnded = await instance.getPhase();

console.log("Phase after changing to Ended:", phaseAfterEnded.toString());

### **2. Voter Registration:**

* **Register a Voter (Admin registers account[1] and account[2]):**

await instance.registerVoter(accounts[1], {from: accounts[0]});

await instance.registerVoter(accounts[2], {from: accounts[0]});

* **Verify Registration:**

const isRegistered1 = await instance.isRegistered(accounts[1]);

console.log("Is account[1] registered?", isRegistered1);

const isRegistered2 = await instance.isRegistered(accounts[2]);

console.log("Is account[2] registered?", isRegistered2);

* **Check Registration Attempt for Already Registered Voter:**

try {

await instance.registerVoter(accounts[1], {from: accounts[0]});

} catch (error) {

console.log("Error registering account[1] again:", error.message);

}

### **3. Voting Phase:**

* **Vote as Registered Voter (account[1] votes for candidate 0):**

await instance.vote(0, {from: accounts[1]});

* **Verify if Voter has Voted:**

const hasVoted1 = await instance.hasVoted(accounts[1]);

console.log("Has account[1] voted?", hasVoted1);

* **Vote as Another Registered Voter (account[2] votes for candidate 1):**

await instance.vote(1, {from: accounts[2]});

* **Verify if Voter has Voted:**

const hasVoted2 = await instance.hasVoted(accounts[2]);

console.log("Has account[2] voted?", hasVoted2);

* **Check the Vote Count for Candidate 0:**

const candidate0 = await instance.getCandidate(0);

console.log("Candidate 0 vote count:", candidate0.toString());

* **Check the Vote Count for Candidate 1:**

const candidate1 = await instance.getCandidate(1);

console.log("Candidate 1 vote count:", candidate1.toString());

### **4. Results Phase:**

* **End the Election (Phase 2):**

await instance.changePhase(2, {from: accounts[0]});

const phaseAfterResults = await instance.getPhase();

console.log("Phase after changing to Results (Ended):", phaseAfterResults.toString());

* **Get the Winner (Should return the candidate with the most votes):**

const winner = await instance.getWinner();

console.log("Winner:", winner);

### **5. Additional Verification Helpers:**

* **Get Number of Registered Candidates:**

const candidateCount = await instance.getCandidatesCount();

console.log("Number of candidates:", candidateCount.toString());

* **Loop Through All Candidates and Print Their Details:**

for (let i = 0; i < candidateCount; i++) {

const candidate = await instance.getCandidate(i);

console.log(`Candidate ${i}:`, candidate.toString());

}

* **Check the Current Phase (0 = Registration, 1 = Voting, 2 = Ended):**

const currentPhase = await instance.getPhase();

console.log("Current Phase:", currentPhase.toString());

### **6. Test Signature Handling:**

If your system uses signatures for validation or authentication, you can also check how signatures are being handled (e.g., for voter registration or vote verification):

* **Log Signatures for Debugging:**

console.log("Signature 1:", sig1);

console.log("Signature 2:", sig2);

* **Verify Signature Validity (if used in contract functions):**

// Assuming you have a function that validates signatures

const isValidSig1 = await instance.verifySignature(sig1);

console.log("Is Signature 1 valid?", isValidSig1);

const isValidSig2 = await instance.verifySignature(sig2);

console.log("Is Signature 2 valid?", isValidSig2);

### **7. Handling Edge Cases:**

* **Attempt Voting After Election End (Should fail):**

try {

await instance.vote(0, {from: accounts[1]});

} catch (error) {

console.log("Error voting after election ended:", error.message);

}

* **Attempt to Change Phase as Non-Admin (Should fail):**

try {

await instance.changePhase(1, {from: accounts[1]});

} catch (error) {

console.log("Error changing phase as non-admin:", error.message);

}

### **Summary of Testing Workflow:**

1. **Check and manage the current phase** using getPhase() and changePhase().
2. **Register voters** and verify their registration.
3. **Cast votes** and verify the vote count for each candidate.
4. **End the election**, determine the winner, and validate the phase transition.
5. **Loop through candidates** to verify vote counts and validate the winner.
6. **Test edge cases** such as trying to vote after the election has ended or attempting unauthorized actions.

These commands will ensure that the backend logic for your voting system works as expected and covers various edge cases.