BAB III

METODE PENELITIAN

3.1 Desain Penelitian

Desain penelitian adalah kerangka kerja yang digunakan oleh peneliti untuk mengatur dan mengelola berbagai elemen penelitian secara menyeluruh dan sistematis. Desain ini bertujuan untuk menjawab pertanyaan penelitian atau menguji hipotesis dengan cara yang koheren dan logis. Desain penelitian yang baik memungkinkan peneliti untuk mengatasi berbagai tantangan metodologis dan mencapai hasil yang akurat serta dapat dipercaya.

Berikut adalah desain dari penelitian ini:

Gambar 3. 1 Desain Penelitian

(Sumber: Data olahan peneliti, 2024)

3.1.1 Perencanaan

Perencanaan bertujuan untuk menentukan tujuan dan sasaran penelitian, yang mencakup pengembangan sistem *e-voting* yang aman, efisien, dan transparan menggunakan mekanisme konsensus *Proof-of-Stake (PoS)* pada platform Ethereum. Selanjutnya, dilakukan identifikasi dan pembatasan lingkup penelitian untuk memastikan fokus pada aspek keamanan dan efisiensi sistem *e-voting*, dengan studi kasus pemilihan vendor di PT Bintan Teknologi Kreatif.

Dalam perencanaan juga mencakup penjadwalan kegiatan penelitian secara terperinci, yang mencakup tahapan perencanaan, kegiatan studi literatur, perancangan, implementasi, hingga pengujian. Setiap tahap memiliki tenggat waktu yang jelas untuk memastikan penelitian berjalan sesuai jadwal.

3.1.2 Studi Literatur

Studi literatur dilakukan untuk mengumpulkan dan menganalisis penelitian sebelumnya yang relevan dengan *e-voting*, *blockchain*, dan *PoS*. Studi ini bertujuan untuk memahami teori dan konsep dasar yang mendasari penelitian, seperti mekanisme konsensus *PoS*, *smart contract*, dan keamanan *blockchain*. Literatur yang relevan mencakup penelitian tentang sistem *e-voting* berbasis *blockchain* dan penerapan *smart contract*. Studi ini juga mengkaji berbagai studi kasus implementasi *blockchain* di sektor lain untuk memahami tantangan dan keberhasilan yang dihadapi. Hasil dari studi

literatur ini menjadi dasar untuk merancang sistem *e-voting* yang lebih baik dan aman.

3.1.3 Perancangan Sistem

Tahap perancangan melibatkan pembuatan gambaran alur kerja dan proses yang dilakukan untuk merancang sistem *e-voting* berbasis *blockchain* sesuai dengan alur kerja metode pengembangan *Waterfall*. Pada tahap ini, peneliti akan melakukan perancangan sistem secara menyeluruh, termasuk perancangan *smart contract* dan komponen utama sistem. Proses perancangan dimulai dengan membuat diagram alur kerja yang menjelaskan langkah-langkah dari registrasi pemilih hingga pengumuman hasil pemilihan. Alur kerja ini akan diuraikan melalui diagram *UML (Unified Modeling Language)* untuk memberikan representasi visual yang jelas dan detail.

3.1.4 Implementasi

Pada tahap implementasi, sistem *e-voting* yang telah dirancang kemudian dikembangkan sesuai dengan alur kerja yang telah ditetapkan sebelumnya. Proses implementasi dimulai dengan pengembangan antarmuka pengguna yang intuitif dan mudah digunakan oleh pemilih. Antarmuka ini dirancang untuk memastikan bahwa pengguna dapat berinteraksi dengan sistem dengan mudah dan jelas.

Setelah antarmuka pengguna selesai, implementasi berlanjut dengan pengembangan *back-end*, yang mencakup perancangan *smart contract* dan integrasi dengan *blockchain Ethereum*. Pada tahap ini, *smart contract* yang

telah dirancang diimplementasikan dan di unggah ke jaringan *Ethereum*. Proses ini melibatkan penulisan kode untuk *smart contract* menggunakan bahasa pemrograman *Solidity*, serta pengujian awal *smart contract* untuk memastikan tidak ada *bug* atau celah keamanan.

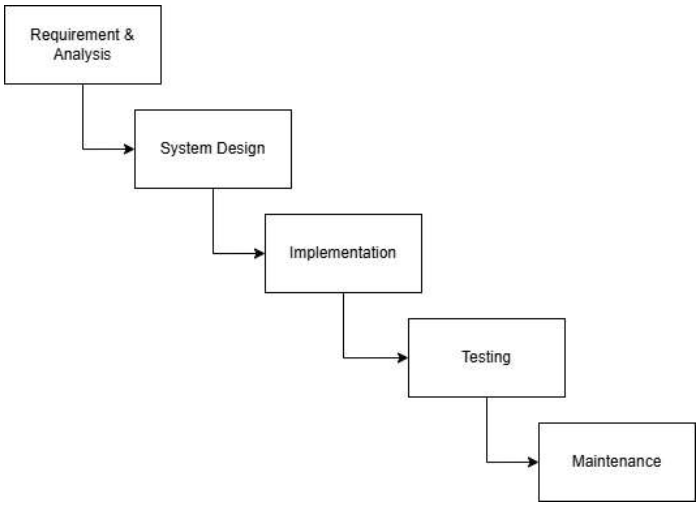
3.1.5 Pengujian

Tahap pengujian dalam penelitian ini dilakukan menggunakan metode *blackbox testing*. *Blackbox testing* dilakukan untuk menguji fungsionalitas sistem tanpa melihat ke dalam kode sumber, memastikan bahwa semua fitur sistem bekerja sesuai dengan yang diharapkan. Setiap fitur diuji secara terpisah untuk memastikan bahwa tidak ada kesalahan dalam implementasi.

3.1.6 Hasil

Tahap akhir dari desain penelitian ini adalah hasil. Hasil dari penelitian ini berupa sistem *e-voting* berbasis blockchain yang telah diuji dan diimplementasikan untuk pemilihan vendor di PT Bintan Teknologi Kreatif. Sistem ini diharapkan dapat meningkatkan keamanan, transparansi, dan efisiensi dalam proses pemilihan vendor.

3.2 Metode Perancangan Sistem



Gambar 3. 2 Alur metode perancangan *Waterfall Method*

(Sumber: Data olahan peneliti, 2024)

Pada penelitian ini, metode perancangan sistem yang digunakan adalah metode *Waterfall.* Metode perancangan sistem ini adalah salah satu metode perancangan yang tertua dan banyak digunakan. Pada metode ini, proses pengembangan dilakukan secara berurutan, di mana setiap tahap harus diselesaikan untuk bisa mengerjakan tahap berikutnya.

Berikut adalah bagaimana metode *Waterfall* diaplikasikan dalam penelitian ini:

a. *Requirement & analysis*: Proses pengembangan akan dimulai dengan melakukan identifikasi kebutuhan fungsional dan non-fungsional dari sistem *e-voting* yang akan dibangun, seperti keamanan, skalabilitas dan antar muka pengguna yang ramah dan mudah digunakan.

b. *System Design:* Merancang arsitektur dari sistem *e-voting,* mencakup komponen-komponen utama seperti bagian *front-end, back-end* dan platform *blockchain Ethereum.* Juga pada tahap ini akan dilakukan perancangan dari *smart contract* yang akan digunakan.
c. *Implementation:* Implementasi dari desain sistem yang telah dirancang pada tahap sebelumnya. Pada tahap ini akan dilakukan implementasi antar muka pengguna dan implementasi *smart contract* pada platform *blockchain Ethereum.*
d. *Testing:* Pengujian dilakukan dengan metode *blackbox testing* untuk memastikan semua fitur sistem bekerja sesuai dengan yang diharapkan.
e. *Maintenance: Monitoring* sistem untuk mendeteksi dan memperbaiki *bug* atau masalah yang muncul setelah pengujian, serta melakukan pembaruan sesuai kebutuhan.

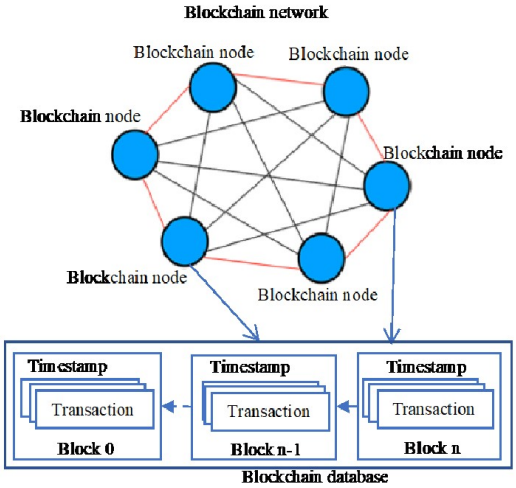
3.2.1 Arsitektur Blockchain

Arsitektur *blockchain* merupakan fondasi dari sistem *e-voting* yang dirancang dalam penelitian ini. Dalam konteks sistem *e-voting*, arsitektur *blockchain* yang diimplementasikan terdiri dari beberapa komponen kunci:

a. Jaringan *Peer-to-Peer*: Sistem ini beroperasi pada jaringan terdesentralisasi, di mana setiap *node* (komputer dalam jaringan) memiliki salinan lengkap dari *blockchain*.
b. Blok: Setiap blok dalam rantai berisi sejumlah transaksi *voting* yang valid. Blok-blok ini terhubung secara kronologis dan kriptografis.

c. Konsensus: Mekanisme *Proof-of-Stake (PoS)* digunakan untuk mencapai kesepakatan tentang validitas transaksi dan blok baru.

d. Kriptografi: Teknik kriptografi asimetris digunakan untuk mengamankan transaksi dan identitas pemilih.



Gambar 3. 3 Arsitektur Blockchain

(Sumber: Salman et al., 2018)

Arsitektur ini memungkinkan sistem e-voting untuk mencapai beberapa karakteristik penting:

a. Immutabilitas

Setiap blok dalam blockchain mengandung *hash* dari blok sebelumnya, membentuk rantai yang tidak dapat diubah. Untuk sistem *e-voting*, ini berarti bahwa sekali suara dicatat, sangat sulit untuk diubah tanpa mengganggu seluruh rantai.

b. Transparansi

Dalam arsitektur *blockchain*, setiap *node* memiliki salinan lengkap dari seluruh rantai. Ini memungkinkan semua partisipan untuk memverifikasi integritas pemilihan.

c. Desentralisasi

Jaringan *peer-to-peer* yang mendasari blockchain menghilangkan kebutuhan akan otoritas pusat. Dalam konteks e-voting, ini berarti tidak ada satu entitas yang dapat memanipulasi hasil.

3.2.2 *Smart contract*

Smart contract adalah inti dari sistem e-voting berbasis blockchain yang kita rancang. Seperti yang didefinisikan oleh (Szabo, 1994), *smart contract* adalah "protokol transaksi terkomputerisasi yang mengeksekusi ketentuan sebuah kontrak". Dalam konteks *e-voting*, *smart contract* berfungsi sebagai mekanisme otomatis yang mengatur seluruh proses pemilihan tanpa perlu campur tangan manusia.

Berikut adalah proses alur kerja dari *smart contract* pada sistem *e-voting* yang akan dibuat:

1. Penulisan kontrak: *Smart contract* ditulis dengan menggunakan bahasa pemrograman khusus yaitu *Solidity*
2. *Deployment*: Setelah *smart contract* selesai ditulis, *smart contract* tersebut kemudian kita *deploy* ke jaringan *blockchain Ethereum*
3. *Trigger event*: Ketika kondisi yang ditentukan sesuai dalam kontrak terpenuhi, kontrak secara otomatis dieksekusi.

4. Eksekusi: *Smart contract* menjalankan fungsinya sesuai dengan logika yang telah diprogram.
5. *Update state*: Hasil eksekusi kontrak direkam di blockchain, memperbarui state dari kontrak.

Berikut adalah implementasi *smart contract* yang akan digunakan pada sistem *e-voting* yang dibuat:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EVoting {
    address public owner;

    struct Voter {
        uint256 voteChoice;
        bool isValidVoter;
    }

    struct Candidate {
        uint256 id;
        string name;
        uint256 voteCount;
    }

    struct Election {
        uint256 id;
        string name;
        uint256 startDate;
        uint256 endDate;
        address[] voterAddresses;
        mapping(address => Voter) voters;
        mapping(uint256 => Candidate) candidates;
        uint256 candidateCount;
        uint256 voterCount;
    }

    struct Result {
        mapping(uint256 => uint256) totalVotes; // candidateId =>
voteCount
        mapping(address => uint256) voteChoices; // voterAddress =>
voteChoice
    }
```

```
    mapping(uint256 => Election) public elections;
    uint256 public electionCount = 0;
    uint256 public candidateCount = 0;

    event ElectionCreated(uint256 electionId, string name, uint256
startDate, uint256 endDate);
    event VoteCasted(uint256 electionId, address voter, uint256
candidateId);

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can perform this
action!");
        _;
    }

    function createElection(
        string memory _name,
        uint256 _startDate,
        uint256 _endDate,
        address[] memory _voters,
        string[] memory _candidates
    ) public onlyOwner {
        require(_startDate < _endDate, "Start date must be before
end date");
        require(_endDate > block.timestamp, "End date must be in the
future");
        require(_voters.length > 1, "Must have atleast two voters");
        require(_candidates.length > 1, "Must have atleast two
candidate");

        electionCount++;
        Election storage newElection = elections[electionCount];
        newElection.id = electionCount;
        newElection.name = _name;
        newElection.startDate = _startDate;
        newElection.endDate = _endDate;
        newElection.candidateCount = 0;

        for (uint256 i = 0; i < _voters.length; i++) {
            require(_voters[i] != owner, "Owner can not be a
voter");
            newElection.voterCount++;
            newElection.voterAddresses.push(_voters[i]);
            newElection.voters[_voters[i]] = Voter({
                voteChoice: 0,
```

```
                isValidVoter: true
            });
        }

        for (uint256 i = 0; i < _candidates.length; i++) {
            candidateCount++;
            newElection.candidateCount++;
            newElection.candidates[newElection.candidateCount] =
Candidate({
                id: newElection.candidateCount,
                name: _candidates[i],
                voteCount: 0
            });
        }

        emit ElectionCreated(electionCount, _name, _startDate,
_endDate);
    }

    function isElectionEnded(uint256 _electionEndDate) internal view
returns (bool) {
        return block.timestamp > _electionEndDate;
    }

    function getElectionDetails(uint256 _electionId)
        public
        view
        returns (
            uint256 id,
            string memory name,
            uint256 startDate,
            uint256 endDate,
            Candidate[] memory candidates,
            address[] memory voterList,
            uint256[] memory voterChoices,
            uint256[] memory results,
            bool hasVoted,
            uint256 votedChoice,
            bool hasEnded
        )
    {
        Election storage election = elections[_electionId];

        name = election.name;
        endDate = election.endDate;
        hasVoted = election.voters[msg.sender].isValidVoter &&
election.voters[msg.sender].voteChoice != 0;
        hasEnded = isElectionEnded(endDate);
        votedChoice = 0;
```

```
        if (hasVoted) {
            votedChoice = election.voters[msg.sender].voteChoice;
        }

        candidates = new Candidate[](election.candidateCount);

        for (uint256 i = 1; i <= election.candidateCount; i++) {
            candidates[i - 1] = election.candidates[i];
        }

        if (isElectionEnded(endDate)) {
            voterList = new address[](election.voterCount);

            for (uint256 i = 0; i < election.voterCount; i++) {
                voterList[i] = election.voterAddresses[i];
            }
        } else {
            voterList = new address[](0);
        }

        if (isElectionEnded(endDate)) {
            results = new uint256[](election.candidateCount);
            voterChoices = new uint256[](election.voterCount);

            // get candidates voted count result
            for (uint256 i = 1; i <= election.candidateCount; i++) {
                results[i - 1] = election.candidates[i].voteCount;
            }

            for (uint256 i = 0; i < election.voterCount; i++) {
                address voterAddress = election.voterAddresses[i];
                voterChoices[i] =
election.voters[voterAddress].voteChoice;
            }
        }

        // Return the election details and candidates
        return (_electionId, name, startDate, endDate, candidates,
voterList, voterChoices, results, hasVoted, votedChoice, hasEnded);
    }

    struct ElectionInfo {
        uint256 id;
        string name;
```

```
        uint256 startDate;
        uint256 endDate;
        bool hasVoted;
    }

    function isRegisteredVoter(uint256 _electionId, address
_voterAddress) internal view returns (bool) {
        Election storage election = elections[_electionId];

        if (election.voters[_voterAddress].isValidVoter == true) {
            return true;
        }

        return false;
    }

    function getElections() public view returns (ElectionInfo[]
memory) {
        uint256 count = 0;

        // First pass to count the number of valid elections
        for (uint256 i = 1; i <= electionCount; i++) {
            if (isRegisteredVoter(i, msg.sender) || msg.sender ==
owner) {
                count++;
            }
        }

        // Initialize the result array with the correct size
        ElectionInfo[] memory electionList = new
ElectionInfo[](count);
        uint256 j = 0;

        // Second pass to populate the result array
        for (uint256 i = 1; i <= electionCount; i++) {
            if (isRegisteredVoter(i, msg.sender) || msg.sender ==
owner) {
                Election storage election = elections[i];
                electionList[j] = ElectionInfo({
                    id: election.id,
                    name: election.name,
                    startDate: election.startDate,
                    endDate: election.endDate,
```

```
                    hasVoted:
election.voters[msg.sender].isValidVoter &&
election.voters[msg.sender].voteChoice != 0
                });
                j++;
            }
        }

        return electionList;
    }

    function vote(uint256 _electionId, uint256 _candidateId) public
{
        Election storage election = elections[_electionId];

        require(block.timestamp >= election.startDate, "Election has
not started");
        require(block.timestamp < election.endDate, "Election has
ended");
        require(election.voters[msg.sender].isValidVoter, "Not a
valid voter");
        require(election.voters[msg.sender].voteChoice == 0,
"Already Voted");
        require(_candidateId > 0 && _candidateId <=
election.candidateCount, "Invalid Candidate ID");

        election.voters[msg.sender].voteChoice = _candidateId;
        election.candidates[_candidateId].voteCount++;

        emit VoteCasted(_electionId, msg.sender, _candidateId);
    }
}
```

Berikut adalah struktur dan fungsi dari *smart contract* tersebut:

1. Struktur Data:
    - *Struct Voter*: Menyimpan pilihan suara dan status validitas pemilih.
    - *Struct Candidate*: Menyimpan ID, nama, dan jumlah suara kandidat.

- *Struct Election*: Mengelola detail pemilihan termasuk daftar pemilih dan kandidat.
- *Struct Result*: Menyimpan hasil pemilihan.

2. Fungsi Utama:
   a. Pembuatan pemilihan (*createElection*):
      - Hanya *owner* yang dapat membuat pemilihan baru.
      - Menetapkan nama, tanggal mulai dan berakhir, daftar pemilih, dan kandidat.
      - Memvalidasi *input* seperti tanggal dan jumlah minimal pemilih/kandidat.
   b. Pemungutan suara (*vote*):
      - Memverifikasi eligibilitas pemilih dan waktu pemilihan.
      - Mencatat suara pemilih dan memperbarui jumlah suara kandidat.
   c. Mendapatkan detail pemilihan (*getElectionDetails*):
      - Mengembalikan informasi lengkap tentang pemilihan tertentu.
      - Menyediakan hasil jika pemilihan telah berakhir.
   d. Mendapatkan daftar pemilihan (*getElections*):
      - Mengembalikan daftar pemilihan yang relevan untuk pemilih atau *owner*.
   e. Fitur keamanan:
      - *Modifier onlyOwner* untuk membatasi akses fungsi tertentu.

- Pengecekan status pemilih dan waktu pemilihan sebelum memproses suara.

f. Transparansi:

- *Event* untuk mencatat pembuatan pemilihan dan pemungutan suara.
- Fungsi *view* untuk memeriksa status dan detail pemilihan.

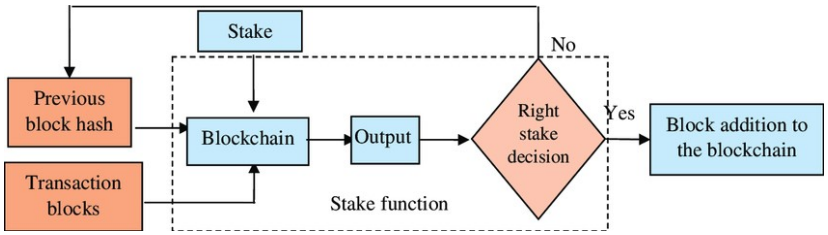
Dari implementasi *smart contract* tersebut, kita mendapat keunggulan-keunggulan berikut:

- Fleksibilitas: Memungkinkan pembuatan beberapa pemilihan dengan konfigurasi yang berbeda.
- Privasi: Hanya menampilkan hasil setelah pemilihan berakhir.
- Efisiensi: Menggunakan *mapping* untuk akses data yang cepat.
- Keamanan: Implementasi kontrol akses dan validasi *input* yang ketat.

Smart contract ini menjamin integritas proses *e-voting* dengan aturan yang terdefinisi jelas dan tidak dapat diubah setelah di-*deploy*, sambil menyediakan fleksibilitas untuk berbagai skenario pemilihan.

3.2.3 Keamanan konsensus *Proof-of-stake*

Proof-of-Stake adalah mekanisme konsensus *blockchain* yang digunakan untuk memvalidasi transaksi dan menciptakan blok baru. Berbeda dengan *Proof-of-Work (PoW)* yang mengandalkan daya komputasi, *PoS* bergantung pada kepemilikan dan *"staking"* token oleh partisipan.



Gambar 3. 4 Mekanisme kerja konsensus *Proof-of-stake*

(Sumber: (Ghosh et al., 2022))

Berikut adalah alur kerja dari konsensus *Proof-of-stake:*

a. *Staking*: pemegang *token* "mempertaruhkan" *(stake)* sejumlah *token* mereka sebagai jaminan.
b. *Validator selection*: Sistem memilih *validator* secara acak, dengan peluang terpilih sebanding dengan jumlah *token* yang di-*stake*.
c. *Block addition*: *Validator* yang terpilih mengusulkan blok baru.
d. *Validation*: *Validator* lain memverifikasi blok yang diusulkan.
e. *Finalization*: Jika blok valid, ditambahkan ke blockchain.
f. *Rewards*: *Validator* menerima imbalan berupa *token* tambahan.

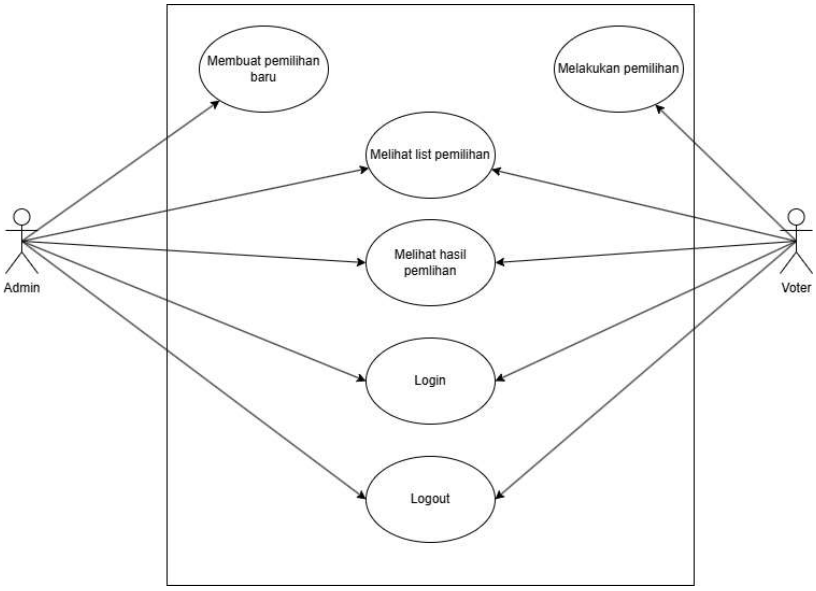
Konsensus *PoS* dengan integrasinya pada *platform blockchain* akan memberikan tambahan keamanan pada sistem *e-voting* yang dibuat berdasarkan alasan berikut:

a. Integritas data: *PoS* memastikan bahwa hanya blok yang valid yang ditambahkan ke *blockchain*, menjaga integritas data suara.

b. Ketahanan terhadap serangan: Ada satu celah bagi penyerang untuk melaksanakan aksinya. Akan tetapi, hal ini mengharuskan penyerang untuk mempunyai mayoritas dari jumlah *stake* yang terdapat pada jaringan *blockchain.* Hal ini tentunya akan membutuhkan biaya yang sangat tinggi sehingga akan menambah tingkat kesulitan untuk melakukan serangan.
c. Finalisasi cepat: *PoS* dapat mencapai finalisasi blok lebih cepat dikarenakan tidak bergantung pada proses penyelesaian teka-teki matematis seperti pada mekanis *Proof-of-Work*, sehingga memungkinkan penghitungan suara yang lebih cepat dan efisien.
d. Skalabilitas: Kemampuan *PoS* untuk menangani lebih banyak transaksi per detik memungkinkan sistem *e-voting* untuk melayani jumlah pemilih yang besar.
e. Transparansi dan auditabilitas: Semua transaksi dan validasi dalam *PoS* tercatat dalam *blockchain*, sehingga memungkinkan audit yang transparan. Hal ini dapat menambah kepercayaan *voter,* karena mereka bisa ikut serta memastikan validitas data yang tercatat pada *blockchain.*
f. Keamanan ekonomi: Setiap *validator* pada jaringan *blockchain* yang menggunakan mekanisme konsensus *Proof-of-stake* memiliki insentif ekonomi untuk menjaga integritas sistem. Karena sekecil apapun hal curang yang terdeteksi yang mereka lakukan pada sistem, maka tanpa ampun sistem akan menghukum dengan menghanguskan semua *stake* yang dimiliki oleh

validator tersebut. Hal ini tentunya akan memaksa *validator* untuk patuh pada aturan dan menjaga integritas jaringan *blockchain.*

3.2.4 *Use case diagram*



Gambar 3. 5 Use Case Diagram

(Sumber: Data olahan peneliti, 2024)

Berdasarkan *use case diagram* di atas, maka dapat diidentifikasi 2 aktor:

Tabel 3. 1 Definisi aktor

Aktor	Definisi
Admin	Bertugas untuk mengelola pemilihan. Admin punya wewenang untuk membuat pemilihan baru dan meregistrasi pemilih yang berhak untuk memilih pada setiap pemilihan yang dibuat.
Voter	Para pengguna yang berhak untuk memberikan hak pilih mereka.

Berikut adalah definisi dari tiap *use case:*

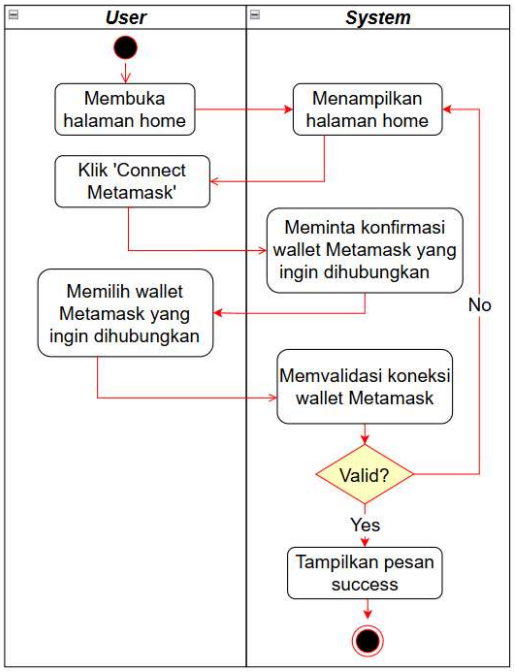
Tabel 3. 2 Definisi *use case*

Use Case	Definisi
Login	Semua pengguna perlu melakukan *login* untuk bisa mengakses semua fitur. *Login* bisa dilakukan dengan cara menghubungkan *wallet Metamask* dari pengguna
Logout	Pengguna bisa keluar dari sistem dengan memutuskan hubungan *wallet Metamask.*
Membuat pemilihan baru	Admin punya wewenang untuk membuat *event* pemilihan baru. Admin bisa mengisi data *event* seperti nama *event* pemilihan, kapan berakhir, kandidat yang bisa dipilih, dan *voter* yang punya berhak untuk melakukan pemilihan.
Melihat *list* pemilihan	Pengguna bisa melihat *list* pemilihan yang di mana pengguna berhak untuk memberikan hak suaranya.
Melihat hasil pemilihan	Pengguna bisa melihat hasil dari pemilihan yang sudah berakhir.
Melakukan pemilihan	Pengguna yang punya hak, bisa memberikan hak suaranya dengan memilih salah satu kandidat.

3.2.5 *Activity Diagram*

Activity diagram membantu memvisualisasikan langkah-langkah yang terlibat dalam suatu proses atau operasi, menunjukkan urutan tindakan yang dilakukan, dan bagaimana alur kerja berpindah dari satu aktivitas ke aktivitas lainnya.

a. *Activity Diagram Login*



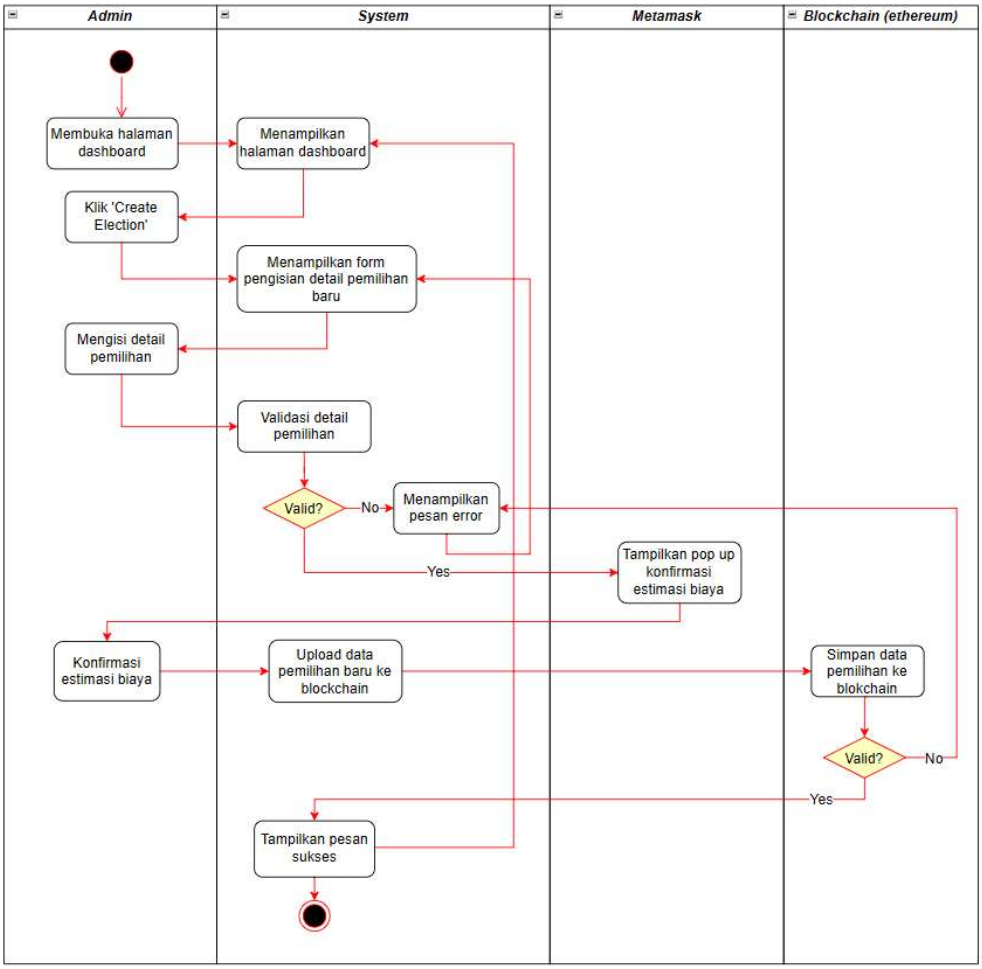
Gambar 3. 6 *Activity Diagram Login*

(Sumber: Data olahan peneliti, 2024)

Proses dimulai dengan pengguna membuka halaman *home*. Sistem kemudian menampilkan halaman *home* kepada pengguna. Di halaman ini, pengguna mengklik tombol "*Connect MetaMask*" untuk memulai proses *login*. Setelah itu, *Metamask* menampilkan *pop-up* untuk meminta konfirmasi dari pengguna mengenai *wallet MetaMask* yang ingin dihubungkan. Pengguna kemudian memilih *wallet MetaMask* yang ingin dihubungkan dengan sistem.

Sistem akan memvalidasi koneksi *wallet MetaMask* tersebut. Jika koneksi ke *Metamask* sukses, sistem akan menampilkan pesan sukses yang menunjukkan bahwa proses *login* berhasil.

b. *Activity diagram* membuat pemilihan baru



Gambar 3. 7 *Activity diagram* membuat pemilihan baru

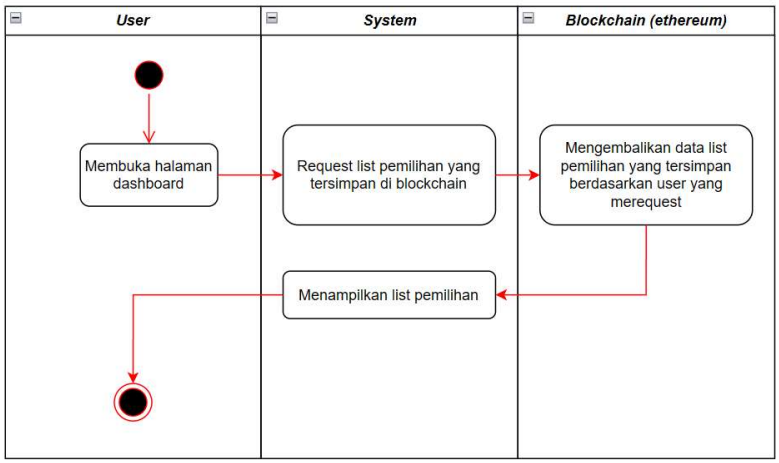
(Sumber: Data olahan peneliti, 2024)

Proses dimulai ketika *admin* membuka halaman *dashboard* sistem. Setelah itu, admin mengklik tombol *"Create Election"* untuk memulai pembuatan pemilihan baru. Sistem kemudian menampilkan formulir pengisian detail pemilihan yang harus diisi oleh admin. Admin mengisi detail pemilihan tersebut, seperti nama pemilihan, kandidat, dan jadwal pemilihan. Setelah semua detail diisi, sistem melakukan validasi untuk memastikan data yang dimasukkan sudah benar dan

lengkap. Jika data tidak valid, sistem akan menampilkan pesan kesalahan kepada admin untuk diperbaiki. Jika data valid, sistem menampilkan *pop-up* konfirmasi estimasi biaya yang perlu dikonfirmasi oleh admin melalui *Metamask*. Setelah konfirmasi biaya, data pemilihan baru diunggah ke blockchain Ethereum.

Proses ini melibatkan penyimpanan data pemilihan ke blockchain dan validasi kembali platform *blockchain*. Jika semua langkah berhasil, sistem akan menampilkan pesan sukses kepada admin, menandakan bahwa pemilihan baru telah berhasil dibuat dan disimpan di blockchain.

c. *Activity diagram* melihat *list* pemilihan



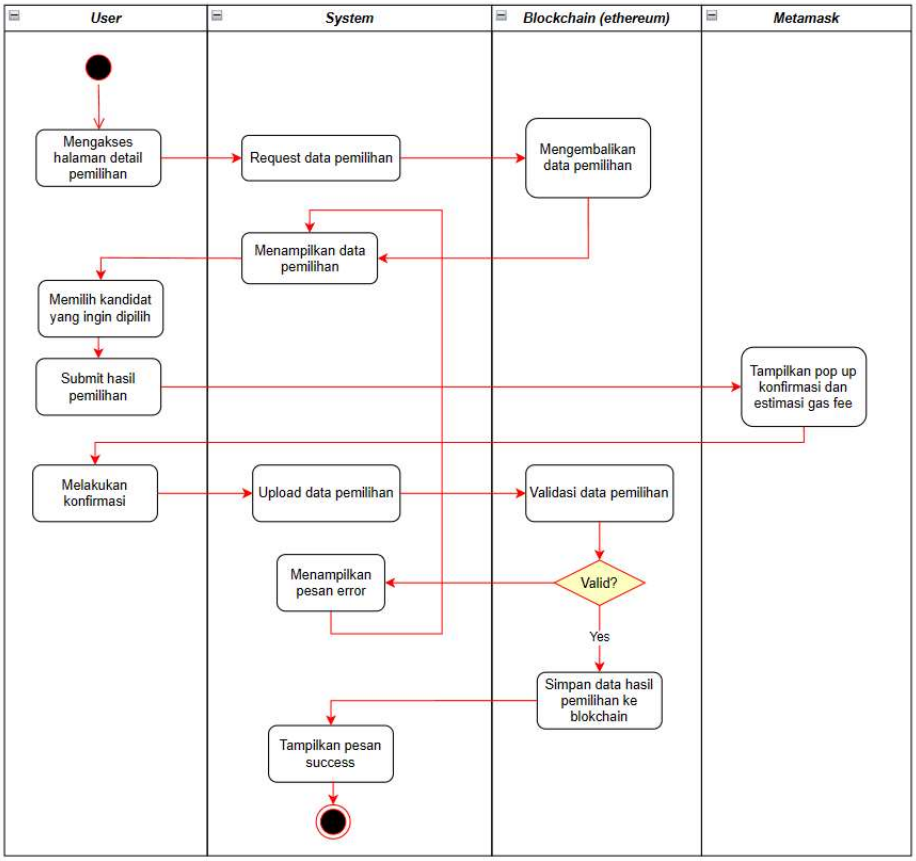
Gambar 3. 8 *Activity diagram* melihat *list* pemilihan

(Sumber: Data olahan peneliti, 2024)

Proses dimulai dengan user membuka halaman *dashboard*. Sistem kemudian mengirimkan permintaan (*request*) untuk mendapatkan daftar pemilihan yang tersimpan di *blockchain Ethereum*. *Blockchain Ethereum* merespons dengan mengembalikan data daftar pemilihan berdasarkan *user* yang melakukan

permintaan tersebut. Setelah menerima data dari blockchain, sistem menampilkan daftar pemilihan kepada *user*.

d. *Activity diagram* melakukan pemilihan



Gambar 3. 9 *Activity diagram* melakukan pemilihan

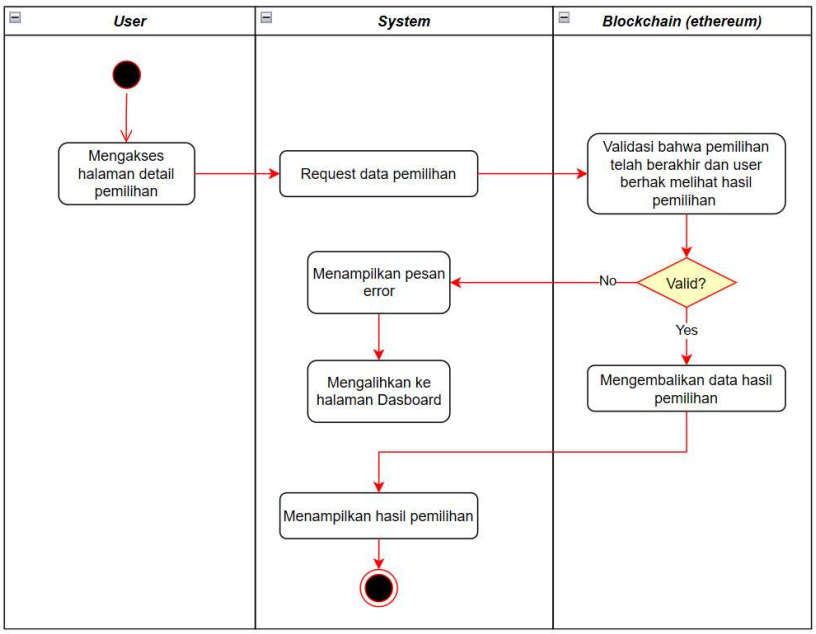
(Sumber: Data olahan peneliti, 2024)

Proses dimulai ketika *user* mengakses halaman detail pemilihan. Sistem kemudian mengirimkan permintaan *(request)* untuk mendapatkan data pemilihan dari *blockchain Ethereum*, yang kemudian mengembalikan data tersebut. Sistem menampilkan data pemilihan kepada *user*, dan *user* memilih kandidat yang ingin

dipilih. Setelah memilih kandidat, user melakukan *submit* hasil pemilihan. Kemudian muncul *pop up* konfirmasi dari *Metamask* beserta estimasi *gas fee* yang akan dikeluarkan. *User* perlu menkonfirmasinya. Sistem kemudian mengunggah data pemilihan ke *blockchain* untuk validasi.

Blockchain Ethereum memvalidasi data pemilihan tersebut. Jika data tidak valid, sistem akan menampilkan pesan *error* kepada user. Jika data valid, sistem menyimpan data hasil pemilihan ke *blockchain* dan menampilkan pesan sukses kepada user.

e. *Activity diagram* melihat hasil pemilihan



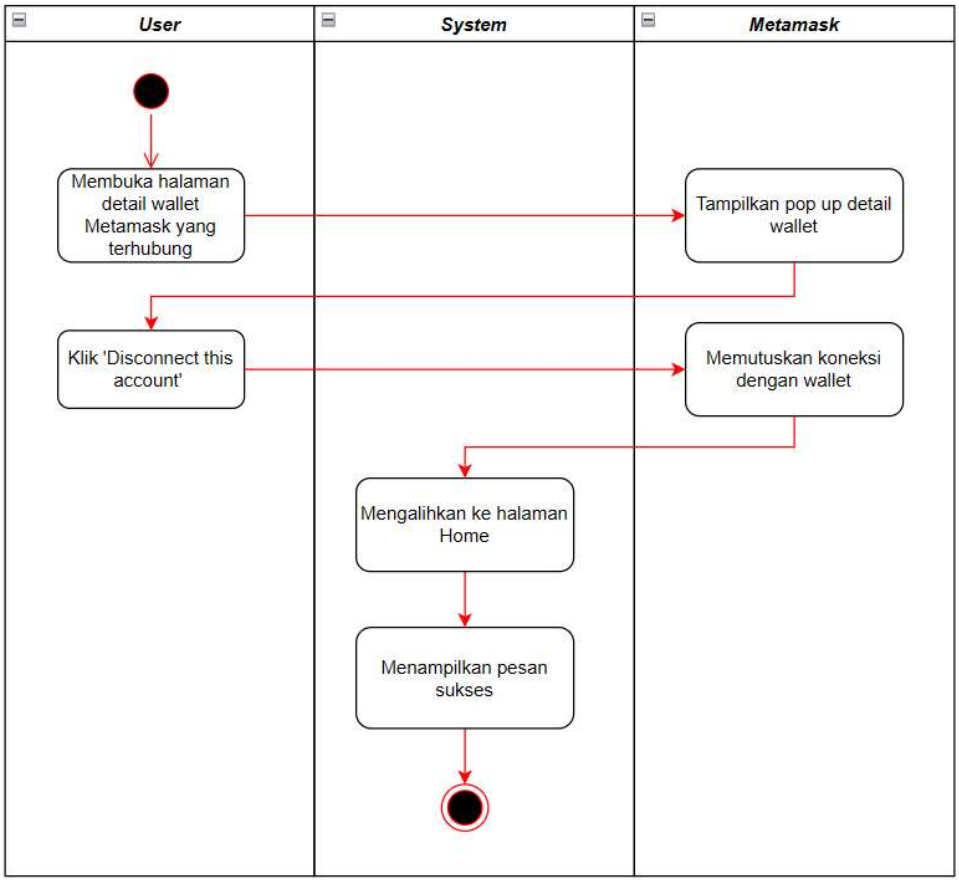
Gambar 3. 10 *Activity diagram* melihat hasil pemilihan

(Sumber: Data olahan peneliti, 2024)

Proses dimulai ketika *user* membuka halaman detail pemilihan. Sistem kemudian mengirimkan permintaan *(request)* untuk mendapatkan data pemilihan

dari *blockchain Ethereum*. *Blockchain* memvalidasi bahwa pemilihan telah berakhir dan *user* berhak melihat hasil pemilihan. Jika validasi gagal, sistem akan menampilkan pesan kesalahan kepada user dan mengarahkan user kembali ke halaman *dashboard*. Jika validasi berhasil, blockchain mengembalikan data hasil pemilihan kepada sistem. Sistem kemudian menampilkan hasil pemilihan kepada user.

f. *Activity diagram* logout



Gambar 3. 11 *Activity diagram logout*

(Sumber: Data olahan peneliti, 2024)

Proses logout dimulai ketika pengguna yang sudah login sebelumnya membuka halaman detail *wallet Metamask* yang terhubung. Kemudian akan tampil *pop up* detail wallet *Metamask* yang sedang terhubung. Pengguna mengklik tombol *"Disconnect this account"* untuk memulai proses *logout*. Setelah tombol diklik, *Metamask* akan memutuskan koneksi *wallet* dengan sistem.

Setelah koneksi terputus, sistem secara otomatis mengalihkan pengguna ke halaman *home* untuk memastikan bahwa sesi pengguna telah diakhiri dengan benar. Di halaman *home*, sistem menampilkan pesan sukses yang mengonfirmasi bahwa proses *logout* telah berhasil.

3.2.6 *Sequence diagram*

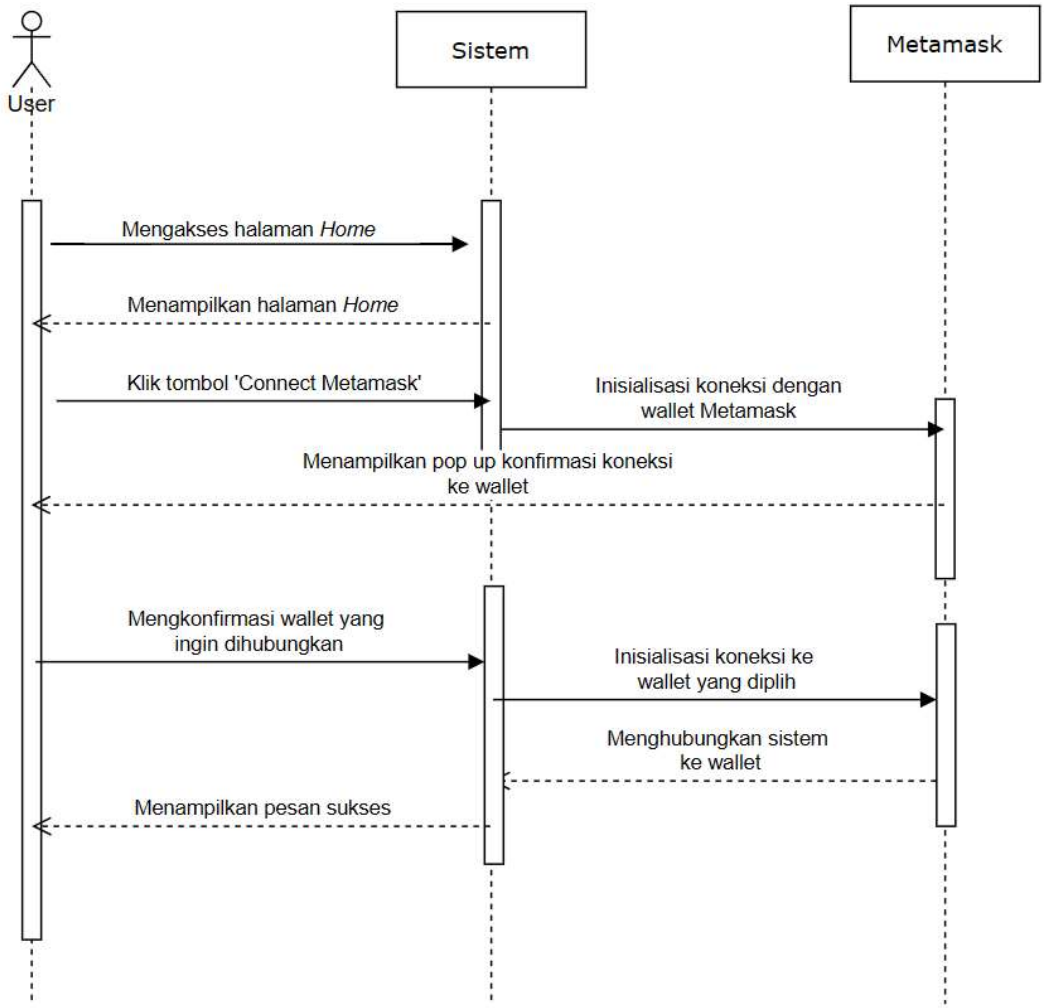
Sequence diagram digunakan untuk menggambarkan bagaimana objek dalam sistem berinteraksi satu sama lain dalam urutan waktu tertentu.

a. *Sequence diagram login*

Proses dimulai ketika pengguna mengakses halaman *home* dari sistem *e-voting*. Sistem kemudian menampilkan halaman *home* kepada pengguna. Di halaman *home*, pengguna mengklik tombol "*Connect Metamask*" untuk memulai proses *login* melalui *Metamask*. Sistem merespons dengan menginisialisasi koneksi dengan *wallet Metamask* dan menampilkan *pop-up* konfirmasi untuk menghubungkan *wallet*.

Pengguna kemudian mengonfirmasi *wallet* yang ingin dihubungkan melalui *pop-up Metamask*. Setelah konfirmasi dilakukan, *MetaMask* menginisialisasi

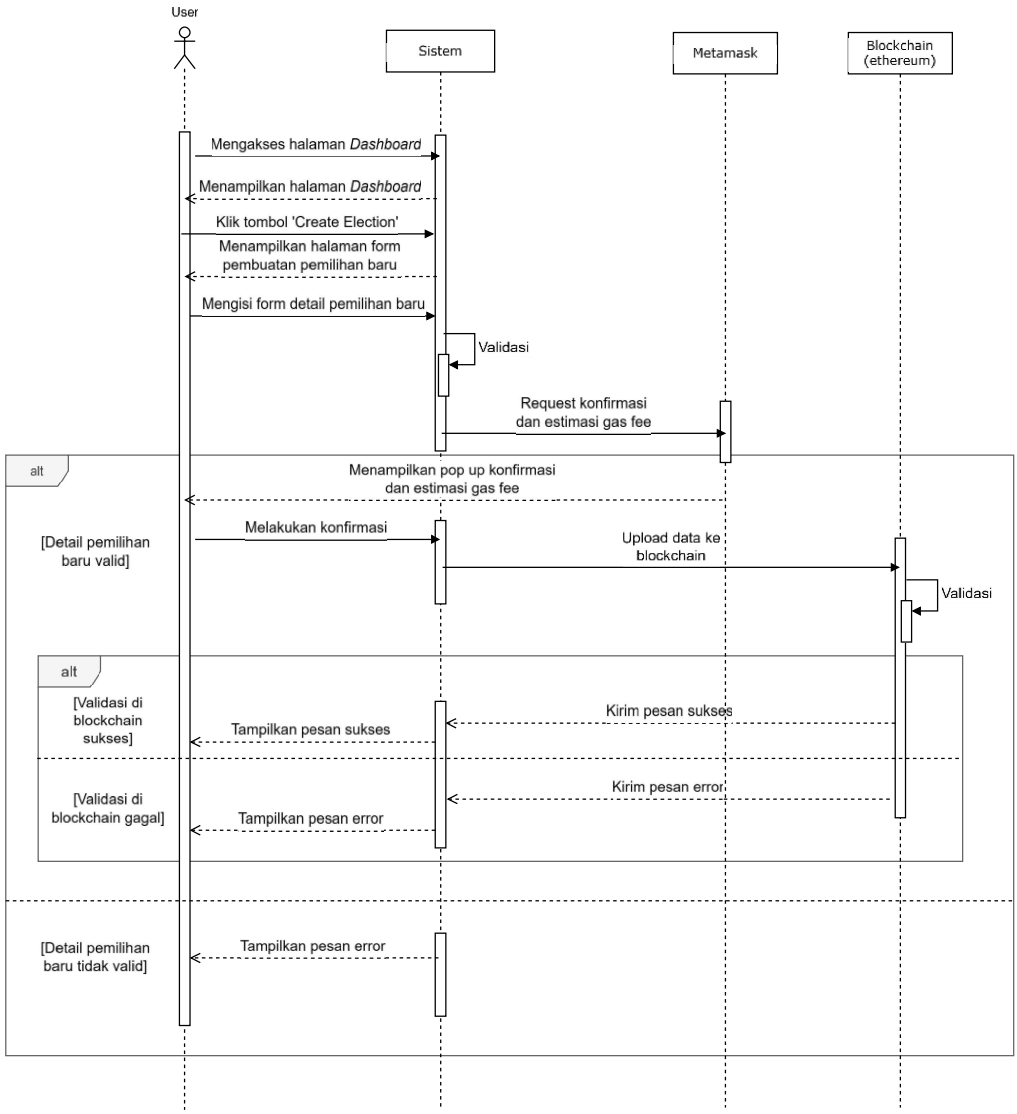
koneksi ke *wallet* yang dipilih dan menghubungkan sistem ke *wallet* tersebut. Setelah terkoneksi, sistem menampilkan pesan sukses kepada pengguna, menunjukkan bahwa mereka telah berhasil *login* ke sistem *e-voting* menggunakan *wallet Metamask* mereka.



Gambar 3. 12 *Sequence diagram login*

(Sumber: Data olahan peneliti, 2024)

b. *Sequence diagram* membuat pemilihan baru



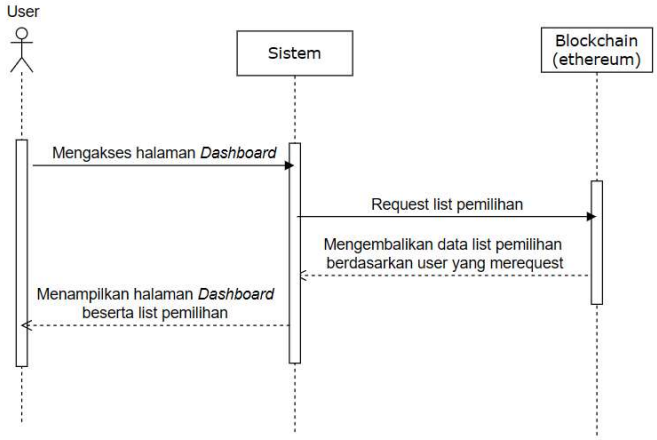
Gambar 3. 13 *Sequence diagram* membuat pemilihan baru

(Sumber: Data olahan peneliti, 2024)

Proses dimulai ketika user mengakses halaman *dashboard*, yang kemudian ditampilkan oleh sistem. *User* mengklik tombol *"Create Election"* dan sistem merespons dengan menampilkan formulir untuk pembuatan pemilihan baru. *User* mengisi detail pemilihan dalam formulir tersebut, yang kemudian divalidasi oleh

sistem. Jika detail pemilihan valid, sistem mengirim permintaan konfirmasi dan estimasi *gas fee* ke *Metamask*, yang kemudian menampilkan *pop-up* konfirmasi kepada *user*. Setelah *user* melakukan konfirmasi, data pemilihan diunggah ke *blockchain Ethereum* untuk validasi lebih lanjut. Jika validasi di *blockchain* berhasil, *blockchain* mengirim pesan sukses kembali ke sistem, yang kemudian menampilkan pesan sukses kepada *user*. Jika validasi di *blockchain* gagal, blockchain mengirim pesan *error*, dan sistem menampilkan pesan *error* kepada *user*. Jika detail pemilihan tidak valid sejak awal, sistem langsung menampilkan pesan *error* kepada user.

c. *Sequence diagram* melihat *list* pemilihan



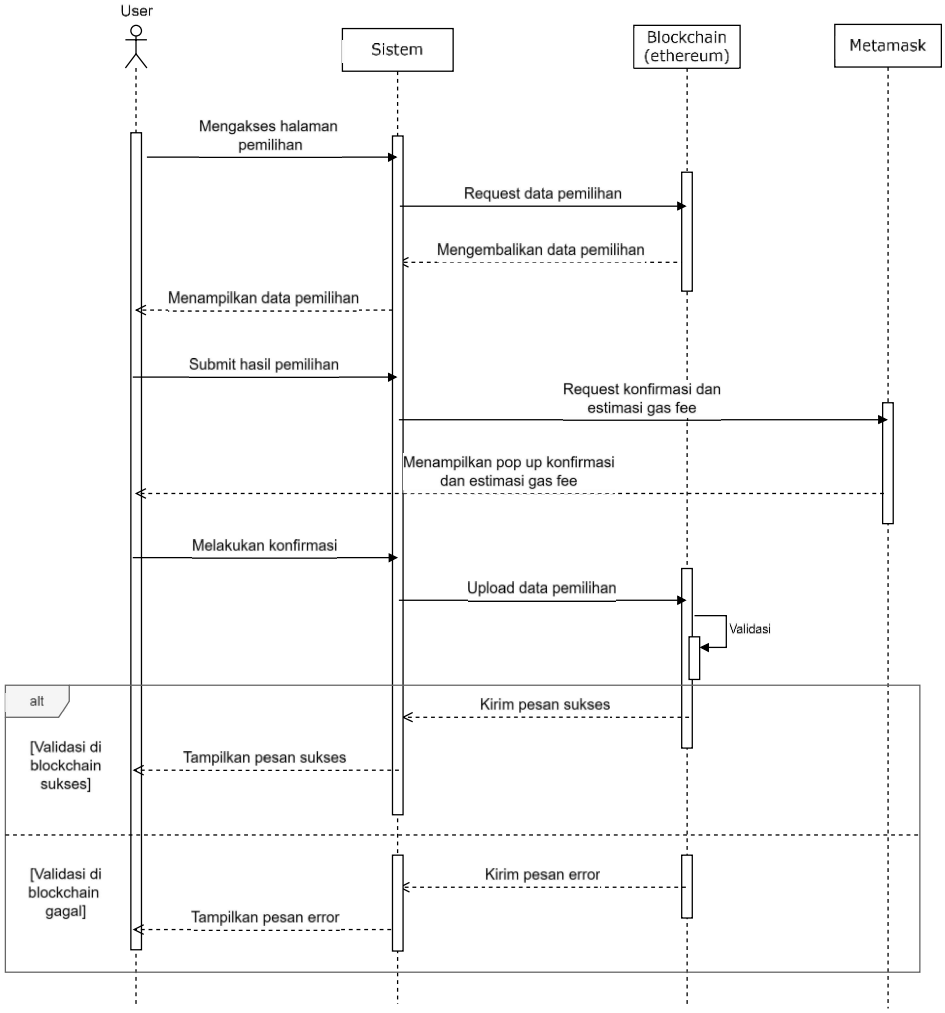
Gambar 3. 14 *Sequence diagram* melihat *list* pemilihan

(Sumber: Data olahan peneliti, 2024)

Langkah pertama, *user* membuka halaman *dashboard*. Sistem kemudian mengirim permintaan ke *blockchain Ethereum* untuk mendapatkan daftar pemilihan. *Blockchain Ethereum* merespons dengan mengirimkan data daftar

pemilihan berdasarkan *user* tersebut. Setelah menerima data dari *blockchain*, sistem menampilkan halaman *dashboard* yang telah diperbarui dengan daftar pemilihan kepada *user*. Proses ini memastikan bahwa *user* selalu melihat data pemilihan yang terbaru dan valid, yang diperoleh langsung dari *blockchain* untuk menjaga keamanannya.

d. *Sequence diagram* melakukan pemilihan



Gambar 3. 15 *Sequence diagram* melakukan pemilihan

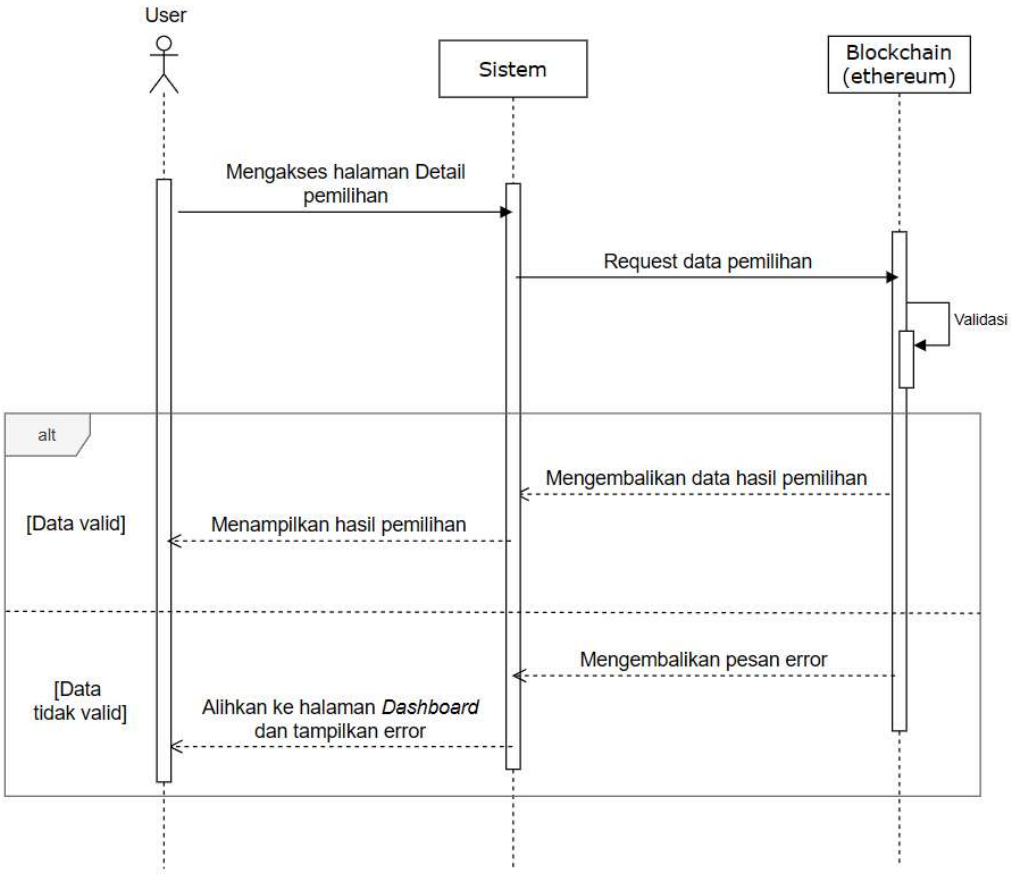
(Sumber: Data olahan peneliti, 2024)

Sequence diagram ini menggambarkan proses ketika seorang *user* mengakses halaman pemilihan dan mengirimkan hasil pemilihannya dalam sistem *e-voting* berbasis *blockchain* dengan bantuan *Metamask*. Proses dimulai saat *user* membuka halaman pemilihan dan sistem menampilkan data pemilihan yang relevan. Setelah user memilih kandidat yang diinginkan, user melakukan *submit* hasil pemilihannya.

Sistem kemudian mengirim permintaan ke *blockchain Ethereum* untuk mendapatkan data pemilihan terkait dan mengembalikan data tersebut. Setelah itu, sistem mengirim permintaan konfirmasi dan estimasi *gas fee* ke *Metamask*, yang menampilkan *pop-up* konfirmasi kepada *user*. *User* kemudian melakukan konfirmasi melalui *Metamask*.

Setelah konfirmasi, data pemilihan diunggah ke *blockchain* untuk validasi lebih lanjut. Jika validasi di *blockchain* berhasil, *blockchain* mengirimkan pesan sukses kembali ke sistem, yang kemudian menampilkan pesan sukses kepada user. Namun, jika validasi di *blockchain* gagal, *blockchain* mengirim pesan *error*, dan sistem menampilkan pesan *error* kepada user. Proses ini memastikan bahwa hasil pemilihan yang di-*submit* oleh user diverifikasi dan disimpan dengan aman di *blockchain*, serta user mendapatkan umpan balik langsung mengenai status pengiriman hasil pemilihannya.

e. *Sequence diagram* melihat hasil pemilihan



Gambar 3. 16 *Sequence diagram* melihat hasil pemilihan

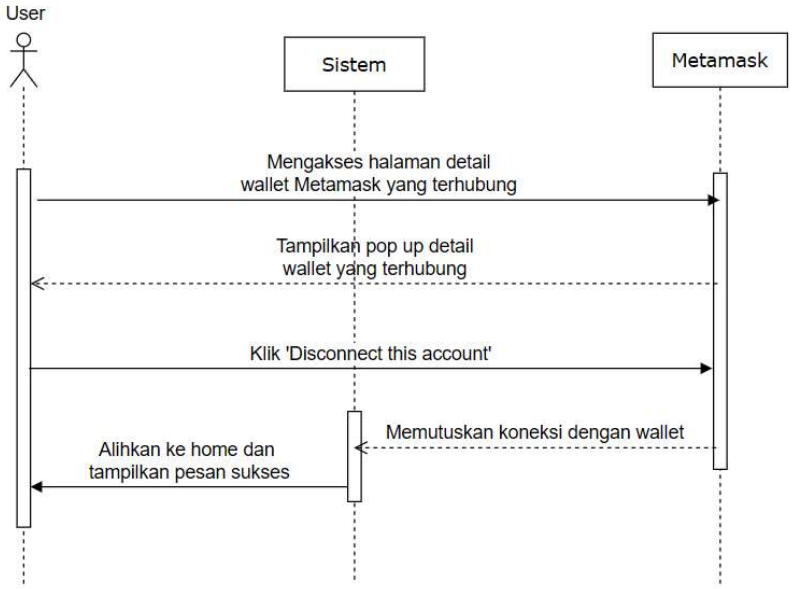
(Sumber: Data olahan peneliti, 2024)

Langkah pertama adalah *user* membuka halaman detail pemilihan. Sistem kemudian mengirim permintaan *(request)* ke *blockchain Ethereum* untuk mendapatkan data hasil pemilihan.

Blockchain Ethereum memvalidasi permintaan tersebut. Jika *request nya* valid, *blockchain* mengembalikan data hasil pemilihan ke sistem. Sistem kemudian menampilkan hasil pemilihan kepada *user*. Namun, jika tidak valid, *blockchain*

mengembalikan pesan *error* ke sistem. Sistem kemudian mengalihkan user ke halaman *dashboard* dan menampilkan pesan *error*.

f. *Sequence diagram logout*



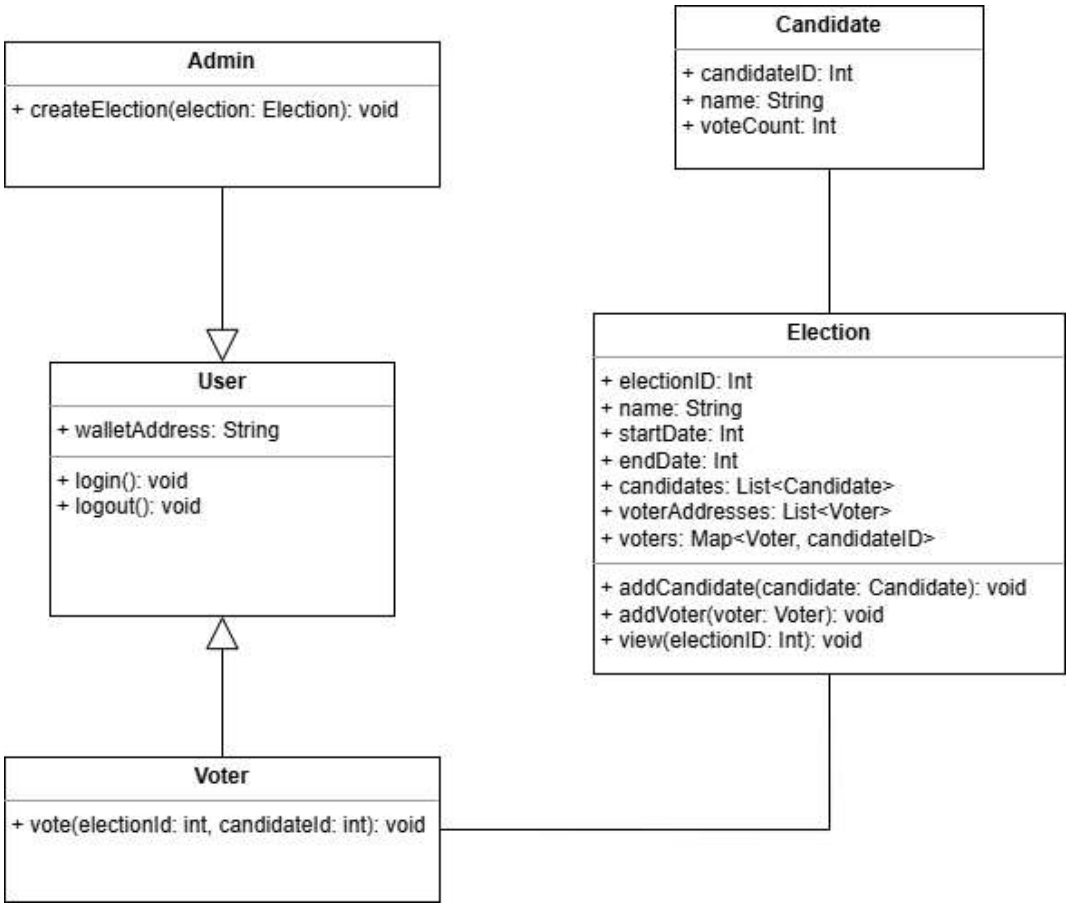
Gambar 3. 17 *Sequence diagram logout*

(Sumber: Data olahan peneliti, 2024)

Proses dimulai ketika *user* mengakses halaman detail *wallet Metamask* yang terhubung. Sistem kemudian menampilkan *pop-up* yang berisi detail *wallet* yang terhubung. *User* kemudian mengklik tombol *"Disconnect this account"* untuk memutuskan koneksi.

Setelah *user* mengklik tombol tersebut, sistem mengirim permintaan ke *Metamask* untuk memutuskan koneksi dengan *wallet*. *Metamask* kemudian memutuskan koneksi dengan *wallet* tersebut. Setelah koneksi berhasil diputus, sistem mengalihkan *user* ke halaman *home* dan menampilkan pesan sukses.

3.2.7 *Class diagram*



Gambar 3. 18 *Class diagram*

(Sumber: Data olahan peneliti, 2024)

Class diagram menggambarkan struktur objek dalam suatu sistem dengan menampilkan kelas-kelas yang menyusun sistem tersebut serta hubungan antar kelas tersebut. Diagram ini memberikan visualisasi grafis dari elemen-elemen yang membentuk sistem dan cara mereka saling berinteraksi.

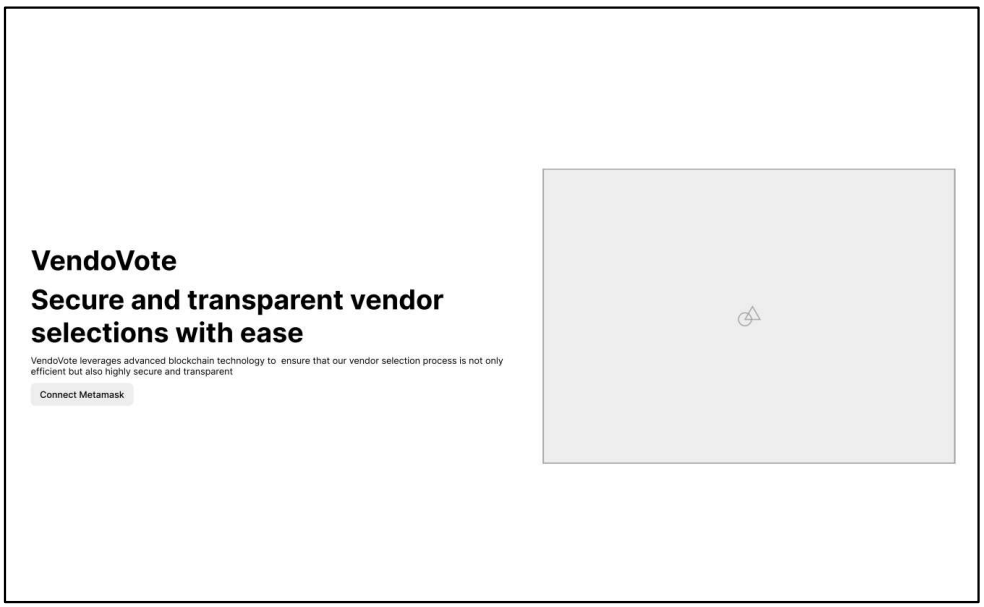
Class diagram mendeskripsikan kelas-kelas, paket-paket, dan objek-objek dalam sistem, beserta hubungan di antara mereka. Kelas dalam diagram ini digambarkan sebagai kotak persegi panjang yang terbagi menjadi tiga bagian: nama

kelas, atribut, dan metode. Setiap kelas memiliki properti (atribut) dan operasi (metode) yang diilustrasikan dalam diagram.

3.3 Perancangan antarmuka

Perancangan antarmuka pengguna adalah proses mendesain tampilan dan interaksi yang akan digunakan oleh pengguna untuk berinteraksi dengan sistem. Perancangan ini sangat penting karena akan mempengaruhi pengalaman pengguna, aksesibilitas, dan juga meningkatkan efisiensi dan produktivitas dari sistem ketika digunakan oleh pengguna

a. *Home page*



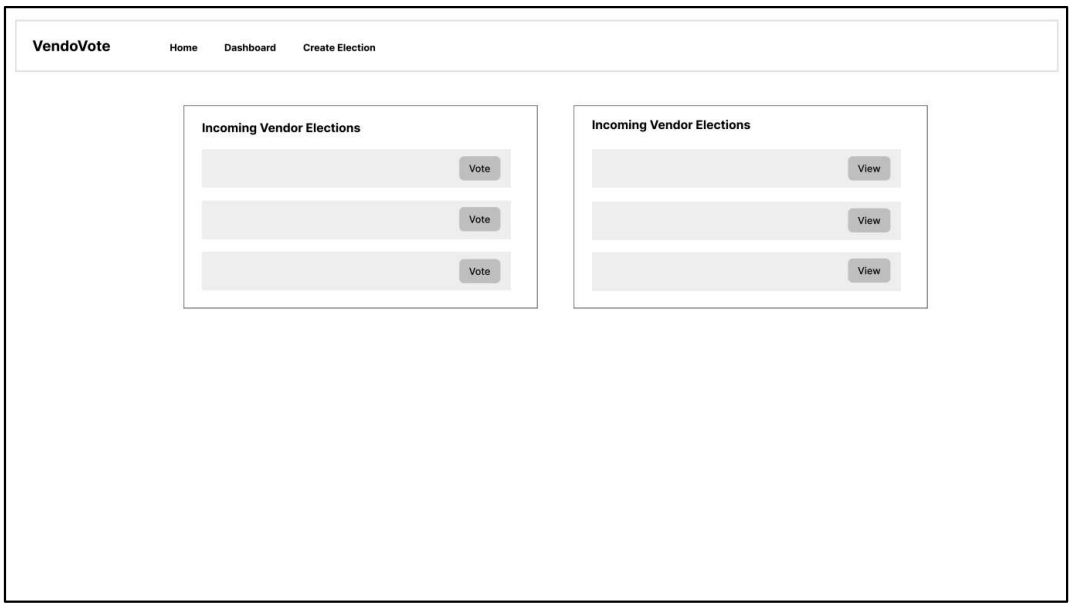
Gambar 3. 19 Rancangan antarmuka *home page*

(Sumber: Data olahan peneliti, 2024)

Halaman *home page* adalah halaman pertama yang dilihat pengguna ketika mereka mengakses aplikasi. Antarmuka ini didesain untuk memberikan kesan

pertama yang kuat dan menggambarkan misi utama dari sistem *e-voting* yang dibuat, yaitu memberikan proses seleksi vendor yang aman dan transparan dengan menggunakan teknologi *blockchain*.

b. *Dashboard page*

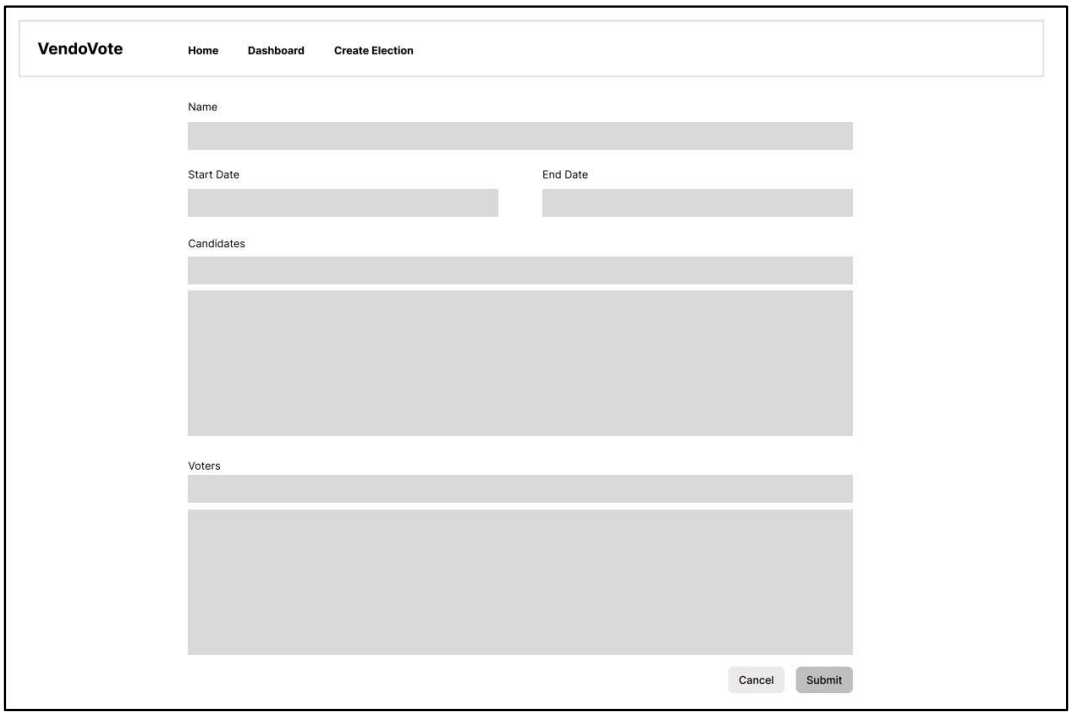


Gambar 3. 20 Rancangan antarmuka *dashboard page*

(Sumber: Data olahan peneliti, 2024)

Dashboard adalah halaman yang bisa diakses setelah pengguna *login*. Halaman ini akan menampilkan pemilihan vendor yang sedang berlangsung dan mendatang, serta memungkinkan akses cepat ke fitur-fitur utama aplikasi. Pada halaman ini, daftar pemilihan yang ditampilkan hanya pemilihan yang berhak diakses oleh pengguna sebagai *voter* yang valid.

c. *Create election page*



Gambar 3. 21 Rancangan antarmuka *create election page*

(Sumber: Data olahan peneliti, 2024)

Halaman *Create Election* memungkinkan admin untuk membuat pemilihan vendor baru. Halaman ini hanya bisa diakses oleh *admin* saja. Antarmuka ini dirancang untuk menjadi intuitif dan mudah digunakan, memastikan bahwa admin dapat dengan cepat dan efisien mengatur pemilihan baru.

Pada halaman ini, akan tersedia *form* detail dari pemilihan baru yang akan dibaut. Pada *form* ini, *admin* bisa memasukkan daftar *voter* yang bisa berpartisipasi dalam pemilihan ini, dan kandidat pilihan yang bisa dipilih oleh *voter*.

Admin juga bisa menentukan kapan pemilihan ini dimulai dan kapan berakhirnya.

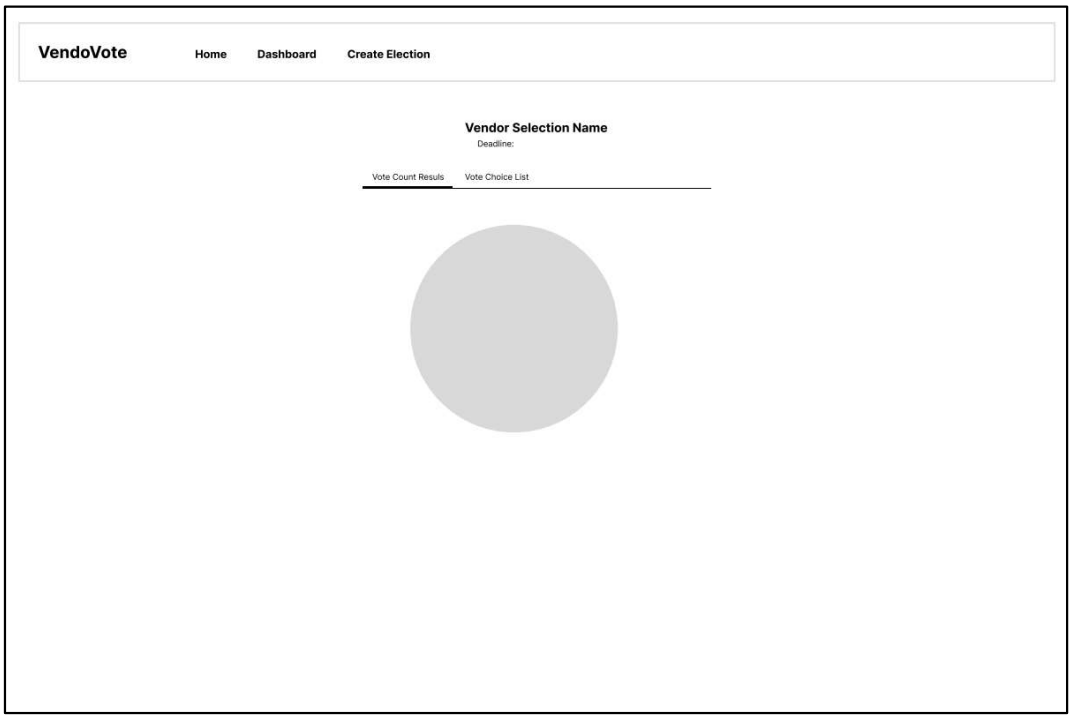
d. *Vote page*

Gambar 3. 22 Rancangan antar muka *vote page*

(Sumber: Data olahan peneliti, 2024)

Halaman *Vote* adalah tempat di mana pengguna dapat memberikan suara mereka dalam pemilihan vendor. Desain halaman ini memastikan bahwa proses pemilihan berjalan lancar dan pengguna dapat dengan mudah memilih kandidat pilihan mereka.

e. *View results page*



Gambar 3. 23 Rancangan antarmuka *view results page*

(Sumber: Data olahan peneliti, 2024)

Halaman *View Results* adalah halaman di mana pengguna dapat melihat hasil dari pemilihan vendor yang telah selesai. Halaman ini dirancang untuk menampilkan hasil dengan cara yang mudah dipahami.

Selain dari hasil pemilihan, di halaman ini juga ditampilkan *list voter* beserta kandidat yang dipilih. *Voter* ditampilkan berdasarkan *wallet addressnya* sehingga menjamin anonimitas dari *voter tersebut.* Fitur ini penting sebagai bentuk transparansi bagi tiap *voter* untuk memastikan bahwa suara mereka terekam dengan benar sesuai dengan pilihan yang mereka pilih pada saat proses pemilihan.

3.4 Metode Pengujian Sistem

Pengujian sistem merupakan langkah penting dalam pengembangan perangkat lunak untuk memastikan bahwa aplikasi yang dibuat sesuai dengan spesifikasi yang telah ditentukan dan bekerja dengan baik tanpa adanya kesalahan. Dalam penelitian ini, metode pengujian yang digunakan adalah *black box testing*.

Black box testing dilakukan dengan menguji setiap fitur aplikasi untuk memastikan bahwa semua fungsionalitas berjalan sesuai dengan spesifikasi. Berikut adalah beberapa langkah yang dilakukan dalam *black box testing* untuk sistem *e-voting* yang dibuat:

Tabel 3. 3 Skenario pengujian fungsionalitas

No.	Fungsionalitas	Skenario Uji	*Input*	*Output* yang diharapkan
1	*Login* menggunakan *MetaMask*	Pengguna mengakses halaman *login* dan mengklik tombol *"Connect MetaMask"*.	Tombol *"Connect MetaMask"* diklik.	*MetaMask pop-up* muncul, pengguna dapat memilih *wallet* dan berhasil *login* ke sistem.
2	Pembuatan pemilihan baru oleh *Admin*	*Admin* mengisi semua detail pemilihan dan mengklik tombol "*Submit*".	Nama pemilihan, tanggal mulai, tanggal berakhir, daftar kandidat, daftar pemilih.	Pemilihan baru berhasil dibuat dan muncul di *dashboard*.
3	Proses pemungutan suara oleh *voter*	Pemilih memilih kandidat dan mengklik tombol "Submit".	Pilihan kandidat yang dipilih.	Suara berhasil dikirim dan dicatat di blockchain.
4	Melihat daftar pemilihan yang tersedia	Pengguna mengakses halaman	Pengguna membuka	Daftar pemilihan yang sedang berlangsung dan

		dashboard untuk melihat pemilihan yang tersedia.	halaman *dashboard*.	mendatang ditampilkan.
5	Melihat hasil pemilihan yang telah selesai	Pengguna mengklik tombol *"View"* untuk melihat hasil pemilihan yang telah selesai.	Tombol *"View"* diklik pada pemilihan yang telah selesai.	Hasil pemilihan ditampilkan dengan grafik hasil suara dan daftar pilihan suara.

3.5 Lokasi dan jadwal penelitian

Penelitian dilakukan di rumah peneliti sendiri dan juga di PT Bintan Teknologi Kreatif yang berlokasi Jalan Kuantan KM 5 atas, Tanjung Pinang.

Penelitian dilakukan selama 6 bulan, dimulai dari pengajuan judul hingga dilaksanakannya sidang skripsi. Berikut adalah runtutan agenda pelaksanaan penelitian:

Tabel 3. 4 Jadwal Penelitian

No.	Kegiatan	Waktu					
		Maret 2024	**April 2024**	**Mei 2024**	**Juni 2024**	**Juli 2024**	**Agustus 2024**
1	Pengajuan judul						
2	Bab 1						
3	Bab 2						
4	Bab 3						
5	Bab 4						
6	Bab 5						
7	Pengumpulan						
8	Sidang						