**Electronic voting machine (EVM) on the blockchain**

**User Interface:**

1. Voter registration: The UI will include a registration process that verifies the identity of the voter and creates a unique cryptographic key for them to use during the voting process.
2. Candidate information: The UI will provide information about the candidates, such as their names, party affiliations, and positions on key issues.
3. Ballot design: The UI will display the ballot in a clear and intuitive way, allowing voters to select their preferred candidate with ease. The UI will also include a way for voters to verify that their vote was recorded correctly.
4. Vote confirmation: Once the voter has cast their vote, the UI will display a confirmation message to ensure that the vote was recorded correctly.
5. Security measures: The UI will include security measures such as two-factor authentication, biometric identification, and end-to-end encryption to protect the privacy and security of the voting process.
6. Accessibility: The UI will be designed to be accessible to all voters, including those with visual or hearing impairments.

**Developing the Smart contract**

**Defining smart contract**

1. Voter registration: The contract will allow users to register as voters by providing their personal information, such as name, address, and voter ID. The contract will verify the identity of the voter and ensure that each voter is only registered once.
2. Ballot design: The contract will define the ballot design, including the candidates or options for the election. The contract will allow the administrator to create and modify the ballot design and ensure that the ballot is consistent with the election rules and regulations.
3. Voting process: The contract will define the voting process, including how voters can cast their votes and how their votes will be recorded on the blockchain. The contract will ensure that each vote is anonymous and can only be cast once.
4. Vote counting: The contract will define the vote counting process, including how the votes will be tallied and how the results will be declared. The contract will ensure that the vote counting process is transparent and tamper-proof.
5. Security and privacy: The contract will ensure that the voting process is secure and private, and that the personal information of the voters is protected. The contract will include security measures such as encryption, access control, and authentication to prevent unauthorized access and tampering.
6. Auditability and transparency: The contract will provide an audit trail that allows for the verification and auditing of the voting process. The contract will ensure that the voting process is transparent and that all voters can verify that their votes have been recorded correctly.
7. User feedback: The contract will provide feedback to the users throughout the voting process, including notifications, status updates, and confirmation messages, to ensure that the user is aware of the voting progress and can verify that their vote was recorded correctly.

**Blockchain Platform to use**

I would recommend using Ethereum for developing an EVM on the blockchain.

Ethereum is a decentralized platform that supports smart contracts and allows developers to create decentralized applications (dApps) using Solidity programming language. Ethereum is widely used in the blockchain industry and has a large developer community, making it easy to find resources and support for developing EVMs.

Ethereum also provides several tools and frameworks for smart contract development, such as Remix IDE, Truffle Suite, and OpenZeppelin, which can simplify the development process and make it easier to test and deploy the smart contract.

Furthermore, Ethereum has several security features, such as gas fees, that prevent malicious actors from exploiting the network by spamming it with transactions. The gas fees ensure that each transaction on the network has a cost associated with it, which prevents users from spamming the network with a large number of transactions.

Overall, Ethereum is a secure, robust, and widely used blockchain platform that supports smart contracts and can be used to develop an EVM on the blockchain

**Smart Contract Code:**

The smart contract code starts by defining two data structures for the EVM - **Voter** and **Candidate**. The **Voter** struct contains the voter's ID, name, and whether they have voted. The **Candidate** struct contains the candidate's ID, name, and vote count.

Next, we define the variables required for the EVM. **voters** is a mapping between the address of a voter and their **Voter** struct. **candidates** is an array of **Candidate** structs. **administrator** is the address of the administrator who is responsible for registering voters, adding candidates, and closing the voting process. **votingClosed** is a boolean variable that indicates whether the voting process is open or closed.

We then define three functions for the EVM:

**registerVoter**: This function allows the administrator to register a voter by providing their voter ID and name. It first checks that the caller is the administrator and that the caller's address has not already been registered as a voter. If both checks pass, it adds the voter's details to the **voters** mapping and emits the **VoterRegistered** event.

* **addCandidate**: This function allows the administrator to add a candidate by providing their candidate ID and name. It first checks that the caller is the administrator and then adds the candidate's details to the **candidates** array.
* **castVote**: This function allows a registered voter to cast a vote for a candidate. It first checks that the caller is a registered voter, that they have not already cast their vote, and that the voting process is still open. If all checks pass, it increments the vote count for the specified candidate and marks the voter as having voted. It then emits the **VoteCast** event.
* **closeVoting**: This function allows the administrator to close the voting process. It first checks that the caller is the administrator and that the voting process is still open. If both checks pass, it sets the **votingClosed** variable to true and emits the **VotingClosed** event.

The smart contract code also includes an event for each of the functions (**VoterRegistered**, **VoteCast**, and **VotingClosed**). These events allow external systems to track the state changes in the EVM and can be useful for auditing purposes.

**To test the smart contract, you can follow these steps:**

1. Open the Remix IDE at <https://remix.ethereum.org/>.
2. Create a new file and name it **ElectronicVotingMachine.sol**.
3. Copy and paste the code for the **ElectronicVotingMachine** contract into the file.
4. Compile the contract by clicking the "Solidity Compiler" tab on the left-hand side of the screen, and then clicking the "Compile ElectronicVotingMachine.sol" button.
5. Once the contract has compiled successfully, click the "Deploy & Run Transactions" tab.
6. Select the "Injected Web3" environment from the "Environment" dropdown list.
7. Connect your Ethereum wallet to the Remix IDE by clicking the "Connect to Wallet" button.
8. Set the gas limit to 3000000 (or a higher value if needed).
9. Click the "Deploy" button to deploy the contract.
10. Once the contract has been deployed, you can test the functions by calling them from the Remix IDE.
11. To test the **registerVoter** function, call it with an Ethereum address as the input parameter.
12. To test the **addCandidate** function, call it with a string parameter for the candidate's name.
13. To test the **castVote** function, call it with the ID of the candidate that the voter wants to vote for.
14. To test the **closeVoting** function, call it to close the voting process.