# **Project Requirements: Technical Member**

Your primary responsibility is the end-to-end technical implementation of the project.

### Functional Requirements (What you will build)

**Smart Contract (NEAR):**

* **FR1: Create Swap:** Implement a payable function (new\_swap) that accepts NEAR tokens and locks them in the contract against a provided hashlock.
* **FR2: Redeem Swap:** Implement a function (redeem) that accepts a secret. If the secret's hash matches a stored hashlock and the swap is active, it must transfer the locked funds to the designated receiver.
* **FR3: Refund Swap:** Implement a function (refund) that allows the original sender to retrieve their locked funds if and only if the timelock has expired.
* **FR4: View Swap State:** Implement a view function to query the status and details of any swap stored in the contract.

**Frontend & Integration (UI):**

* **FR5: Wallet Integration:** Integrate the NEAR Wallet Selector to allow users to connect their testnet wallet to the application.
* **FR6: On-Chain Calls:** Write the client-side JavaScript (near-api-js) to call the new\_swap, redeem, and refund functions on the deployed smart contract from the UI.
* **FR7: Data Display:** Fetch and display the list of active swaps relevant to the connected user.
* **FR8: Secret Generation:** Implement a helper function in the UI to generate a random secret and its corresponding hash for the user to initiate a swap.

### Non-Functional Requirements (How you will build it)

* **NFR1: Security:** The smart contract logic must be secure against common exploits. Ensure funds cannot be drained or permanently locked. Access control must be strictly enforced (e.g., only the original sender can refund).
* **NFR2: Reliability & Robustness:** The code must handle edge cases gracefully (e.g., attempting to redeem an invalid or already completed swap, refunding before the timeout).
* **NFR3: Code Quality & Readability:** The TypeScript code for both the contract and the UI must be well-commented, organized, and easy for judges to understand.
* **NFR4: Testability:** Write unit tests for the smart contract to verify all core logic, including success paths and failure conditions.
* **NFR5: Deployment:** You are responsible for successfully deploying the smart contract to the NEAR testnet and the UI to a public hosting service (like Vercel or Netlify).