Web3 Engineer Take-Home Assignment

Objective: Build a simple decentralized application (DApp) that interacts with an Ethereum smart contract, using Next.js for the frontend and Web3.js or Ethers.js for blockchain interaction. Implement a CI/CD pipeline using GitHub Actions to lint, test, and deploy the application. Tech Stack:

- **Frontend**: Next.js
  - **Linting**: ESLint and Prettier
  - **Testing**: Vitest for unit tests
  - **CI/CD**: GitHub Actions
  - **Blockchain Interaction**: Web3.js or Ethers.js
- **Smart Contract**
  - **Language:** Solidity (use Hardhat or Foundry for development)
  - **Network**: Monad testnet (or a local Hardhat network for development)

Part 1: Smart Contract Development

Write a simple Solidity smart contract called Counter with the following functionality:

- A public variable count initialized to 0.
- A function increment() that increases count by 1.
- A function decrement() that decreases count by 1, but prevents count from going below 0.
- An event CountChanged(uint256 newCount) emitted when count changes.

Deploy the contract to the Monad testnet (or a local Hardhat network if testnet access is limited). Provide the contract address and ABI in your submission repo readme file

Deliverables:

- Solidity contract code.
- Deployment script (e.g., Hardhat script).
- Deployed contract address and ABI.

Part 2: Frontend Development with Next.js

Create a Next.js application that interacts with the Counter contract. The frontend should:

- Connect to the user's wallet.

- Display the current count value from the smart contract.
- Provide buttons to call increment() and decrement() functions.
- Show a loading state during transaction processing.
- Handle errors (e.g., transaction rejected).
- Use Web3.js/Ethers.js (or similar) to interact with the contract.

Requirements:

- Use TypeScript for type safety.
- Style the UI for a clean, responsive design.
- Display transaction confirmations or errors using toast notifications.
- Ensure the UI is intuitive and user-friendly, with clear feedback for actions.

Deliverables:

- Next.js project with app routing page, components, and Web3/Ethers.js integration.
- Instructions for running the frontend locally.

Part 3: Testing and Linting

Write unit tests for the smart contract and frontend logic:

- **Smart Contract Tests**:
  - Test increment() increases count correctly.
  - Test decrement() decreases count and prevents negative values.
  - Test event emission for CountChanged.
- **Frontend Tests with vitest**:
  - Test that the count is displayed correctly.
  - Test button clicks trigger the correct contract calls (mock Web3/Ethers.js interactions).

Set up ESLint and Prettier for code consistency:

- Ensure no linting errors in the codebase.

Part 4: CI/CD with GitHub Actions

Set up a GitHub Actions workflow to automate linting, testing, and deployment:

- **Linting**: Run ESLint to check for code quality issues.

- **Testing**: Run smart contract and frontend tests using Vitest.
- **Deployment**: Deploy the Next.js app to Vercel (or another static hosting service) on push to the main branch.
- Ensure the workflow fails if linting or tests fail.

Deliverables:

- GitHub Actions workflow file
- Frontend deployment on Vercel

Submission Guidelines

- Push all code to one sinhle public GitHub repository, random repo name is preferred and without giving any hints it is about mu digital among all source files
- Include a README.md with at least:
    - Instructions to run the project locally (smart contract and frontend).
        - The deployed contract address on the Monad testnet (or note if using a local Hardhat network, with instructions for local setup).
    - A link to the deployed frontend (e.g., Vercel URL).
    - Justification for your choice of Web3 library (e.g., Web3.js, Ethers.js, or other)
- Ensure the project is well-organized, with a clear folder structure and comments explaining key logic.