Workflow: Setting Up GoalForge from Scratch

This updated workflow removes **Hardhat** and uses **Remix** for smart contract compilation and deployment. The frontend integration remains the same.

---

1. Setting Up Project Dependencies

Step 1.1: Initialize the Frontend Project

Run the following commands to set up the project:

bash

Copy code

```
npx create-next-app@latest goalforge
cd goalforge
```

Step 1.2: Install Dependencies

Install the necessary libraries for blockchain interaction and UI:

bash

Copy code

```
npm install ethers avalanche tailwindcss postcss autoprefixer @mui/material @emotion/react @emotion/styled
npx shadcn@latest init -d
```

---

2. Organize the Project Structure

Structure your project as follows:

plaintext

Copy code

```
src/
├── app/
│   ├── api/       # API routes
│   ├── dashboard/ # User dashboard
│   ├── home/      # Landing page
│   └── login/     # Authentication
├── components/    # Reusable UI components
├── contracts/     # Smart contracts
├── context/       # Context API files
├── styles/        # Global styles
├── lib/           # Utility functions
├── public/        # Static assets
└── types/         # TypeScript types
```

---

3. Develop and Deploy Smart Contract with Remix

Step 3.1: Write the Smart Contract

1. Open Remix IDE.
   - Create a new file `GoalForge.sol` in Remix and paste the following contract:
     solidity
     Copy code

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

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract GoalForge is ERC721 {
    struct Habit {
        address user;
        uint256 stake;
        string goal;
        uint256 deadline;
        bool completed;
    }

```

```
    mapping(uint256 => Habit) public habits;
    uint256 public habitCount;

    constructor() ERC721("GoalForgeNFT", "GFNFT") {}

    function createHabit(string memory goal, uint256 deadline) external payable {
        require(msg.value > 0, "Stake must be greater than 0");
        habits[habitCount] = Habit(msg.sender, msg.value, goal, deadline, false);
        habitCount++;
    }

    function completeHabit(uint256 habitId) external {
        Habit storage habit = habits[habitId];
        require(msg.sender == habit.user, "Not your habit");
        require(!habit.completed, "Habit already completed");
        require(block.timestamp <= habit.deadline, "Deadline passed");

        habit.completed = true;
        payable(habit.user).transfer(habit.stake);
        _mint(habit.user, habitId);
    }
}
```

2.

Step 3.2: Deploy the Contract

1. Compile the contract in Remix.
2. Switch your MetaMask wallet to the **Avalanche Fuji Testnet** (or Mainnet for production):
    - RPC URL: `https://api.avax-test.network/ext/bc/C/rpc`
    - Chain ID: `43113`
    - Currency Symbol: `AVAX`.
3. Deploy the contract via Remix using **Injected Web3**.
4. Copy the deployed contract address and download the ABI.

---

4. Frontend Integration

Step 4.1: Configure Environment Variables

Create a `.env` file in your project and add the following:

bash

Copy code

```
NEXT_PUBLIC_AVALANCHE_RPC_URL=https://api.avax-test.network/ext/bc/C/rpc
NEXT_PUBLIC_CONTRACT_ADDRESS=<DEPLOYED_CONTRACT_ADDRESS>
```

Step 4.2: Setup Ethers.js Interaction

Create a file `src/lib/ethers.ts`:

typescript

Copy code

```
import { ethers } from "ethers";

const provider = new ethers.providers.JsonRpcProvider(
    process.env.NEXT_PUBLIC_AVALANCHE_RPC_URL
);

export const getContract = (address: string, abi: any) => {
    const signer = provider.getSigner();
    return new ethers.Contract(address, abi, signer);
};
```

Step 4.3: Fetch and Display Habits

1. Add the contract's ABI file to `src/contracts/GoalForge.json`.
    - Use `getContract` in components like `BrowseHabits.tsx`:
      typescript
      Copy code
      `import { getContract } from "@/lib/ethers";`

```
import GoalForgeABI from "@/contracts/GoalForge.json";

const GoalForgeAddress =
process.env.NEXT_PUBLIC_CONTRACT_ADDRESS;

const fetchHabits = async () => {
    const contract = getContract(GoalForgeAddress,
GoalForgeABI);
    const habitCount = await contract.habitCount();
    for (let i = 0; i < habitCount; i++) {
        const habit = await contract.habits(i);
        console.log(habit);
    }
};

fetchHabits();
```

2.

---

5. Backend API Setup

Use the `GoalForge` smart contract in your backend API routes.

Step 5.1: Example Habit Creation API

Create a route file `src/app/api/createHabit/route.ts`:

typescript

Copy code

```
import { NextApiRequest, NextApiResponse } from "next";
import { getContract } from "@/lib/ethers";
import GoalForgeABI from "@/contracts/GoalForge.json";

const GoalForgeAddress =
process.env.NEXT_PUBLIC_CONTRACT_ADDRESS;

export default async function handler(req: NextApiRequest,
res: NextApiResponse) {
```

```
    const { goal, deadline } = req.body;
    const contract = getContract(GoalForgeAddress, GoalForgeABI);

    try {
        const tx = await contract.createHabit(goal, deadline, {
            value: ethers.utils.parseEther("0.1"),
        });
        await tx.wait();
        res.status(200).json({ message: "Habit created!" });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
}
```

---

6. Final Steps

1. **Configure Tailwind CSS:**
   Add styles to `globals.css` and configure `tailwind.config.js`.
2. **Test End-to-End Functionality:**
   - Interact with the deployed contract through the frontend.
   - Verify transactions on Avalanche Testnet Explorer.
   - **Deploy the Frontend:**
     Build and deploy the app using your preferred hosting service:
     bash
     Copy code
     `npm run build`
3. `npm run start`