Decentralized Social Media MVP
Project Specification

Problem Statement

Centralized social media platforms collect and control user data, often leading to privacy concerns, data exploitation, and censorship. The increasing adoption of blockchain technology and decentralized identity systems highlights the need for a transparent, user-owned social media alternative.

This project aims to develop a Minimal Viable Product (MVP) of a decentralized social media platform enabling users to log in via their Ethereum wallet, post short messages, interact with others, and manage their profiles. The focus is on trustless authentication and a streamlined, user-centric experience.

Project Overview

Develop a full-stack decentralized microblogging platform with the following core features:Authentication

- Login using RainbowKit and ethers.js
- Ethereum wallet as the sole identity provider

User Profile

- Create and update profile details (username, bio, profile picture URL)
- Wallet address serves as the unique identifier

Post (Tweet)

- Publish brief text-based updates (maximum 280 characters)
- View a consolidated feed of all users' posts

Interactions

- Like posts
- Comment on posts

Tech Stack

Frontend

- React.js
- Next.js

- Tailwind CSS

Backend

- NestJS
- TypeScript
- PostgreSQL

Web3 Integration

- ethers.js
- RainbowKit

Development Guidelines

1. Wallet Integration

- Implement RainbowKit for a user-friendly wallet connection UI.
- Utilize ethers.js to retrieve user addresses and sign messages as necessary.
- Design the system so that the wallet address is the exclusive identifier for login.
- Store session information within the browser or employ a lightweight token-based session management.

2. Backend API (NestJS)

Construct a RESTful API with the following endpoints:Authentication

- `POST /auth/verify`: Accepts a signed message and validates the associated wallet address.

User Profile

- `GET /users/:wallet`: Retrieves a specific user's profile.
- `POST /users`: Creates or updates a user's profile.

Posts

- `GET /posts`: Returns a feed of the latest posts.
- `POST /posts`: Creates a new post.

- `POST /posts/:id/like`: Registers a "like" for a specific post.
- `POST /posts/:id/comment`: Adds a comment to a specific post.
- `GET /posts/:id`: Fetches detailed information for a specific post, including its comments and likes.

Database Schema (PostgreSQL)

Define the following tables:

- `users` (`wallet_address` PRIMARY KEY, `username`, `bio`, `profile_pic_url`)
- `posts` (`id` SERIAL PRIMARY KEY, `wallet_address`, `content`, `timestamp`)
- `likes` (`post_id`, `wallet_address`, PRIMARY KEY (`post_id`, `wallet_address`))
- `comments` (`id` SERIAL PRIMARY KEY, `post_id`, `wallet_address`, `content`, `timestamp`)

3. Frontend InterfacePages:

- `/`: Displays the public feed of posts.
- `/profile`: Shows the logged-in user's profile, with options for viewing and editing.
- `/post/[id]`: Presents the details of a specific post, including its comments.

Components:

- Wallet Connect Button
- Post Composer (for creating new posts)
- Feed/Post List (for displaying posts)
- Post Card (individual post display with like and comment actions)
- Profile Card and Edit Form

Employ Tailwind CSS for styling:

- Ensure a responsive and mobile-first design.

- Utilize utility-first CSS classes.
- Maintain a clean and minimal visual layout.

4. Utilization of AI Tools

You are encouraged to leverage AI-powered developer tools to accelerate development and enhance code quality. Examples include:

- ChatGPT (for logic implementation, code snippets, and rapid debugging)
- GitHub Copilot
- Cursor (especially version 0 for efficient coding workflows)
- Tabnine
- Figma AI (for quick UI prototyping and layout ideas)

Utilize these tools to generate foundational code, validate logical constructs, and resolve development obstacles effectively.

Deliverables

A GitHub repository (private or public) containing:

- Frontend codebase (React.js, Next.js, Tailwind CSS)
- Backend codebase (NestJS, TypeScript, PostgreSQL)
- Database schema migrations or a SQL dump
- A `.env.example` file with instructions for configuring environment variables
- A `README.md` file including:
    - Setup instructions for the project
    - A guide on how to use the application
    - A list of technologies used
    - An overview of the completed features
    - Optional screenshots or a brief demo video

Additional Notes

- Prioritize the creation of a functional prototype over a production-ready application.

- Maintain code readability and ease of maintenance.
- Include concise comments and basic documentation in both the frontend and backend code.
- Implement fundamental error handling mechanisms.
- Full completion of the application is not mandatory; submission of the GitHub URL reflecting progress within the allocated timeframe is expected.

Success Criteria

- Successful wallet connection via RainbowKit.
- Functionality for users to create and update their profiles.
- Ability for users to post short text messages.
- A feed displaying posts from all users.
- Working like and comment functionalities (optional depending on time constraints).