DES484 BLOCKCHAIN DEVELOPMENT PROJECT

Digital advertising media Dapps

Submitted to

School of Information, Computer and Communication Technology

Sirindhorn International Institute of Technology

Thammasat University

October 2023

by

Travis Numpa 6322771641

Chusana Rompo 6322774579

Sonam Dorji 6322790476

Advisor: Dr. Watthanasak Jeamwatthanachai

Table of contents

List of Figure

1 System Description

Bitpuen is a Digital Advertising Media Decentralized Application (DApp) envisioned to transform the advertising landscape by embracing decentralization. The project begins by utilizing Figma, a robust design tool, to craft a user-centric graphical interface for the website. This strategic use of Figma ensures an intuitive and visually engaging experience for Bitpuen users, laying the foundation for a user-friendly platform.

Following the design phase, Bitpuen seamlessly integrates React, a powerful JavaScript library, to establish connectivity. React's implementation is pivotal, fostering smooth user interactions and enabling effortless navigation throughout the application. Its adaptive nature contributes to maintaining a responsive and dynamic user interface, enhancing the overall user experience.

Bitpuen is designed in alignment with Web 3.0 principles, transitioning into a decentralized application. This pivotal shift empowers users by decentralizing control and promoting transparency within the advertising ecosystem. It marks a significant leap towards a more equitable and inclusive platform.

At the core of Bitpuen's functionality lies the creation and deployment of smart contracts. These contracts serve as the backbone of the platform, facilitating various functionalities. Users can create advertisements, both with and without images, contributing diverse content to the Bitpuen platform. The implementation of smart contracts extends to a unique voting system, enabling users to express their appreciation or critique by upvoting or downvoting posts. This mechanism ensures a fair and authentic evaluation of content, limiting each user to a single vote per post to maintain fairness and authenticity.

Bitpuen encapsulates a suite of features, including the ability for users to view advertisements, create engaging content, and participate in a voting system designed to gauge the community's appreciation of posted content. Through the amalgamation of Figma's design capabilities, React's seamless connectivity, the adoption of Web 3.0 principles, and the implementation of smart contracts, Bitpuen aims to redefine digital advertising, offering users a transparent, interactive, and equitable ecosystem for advertising content.

1.1 Perspective

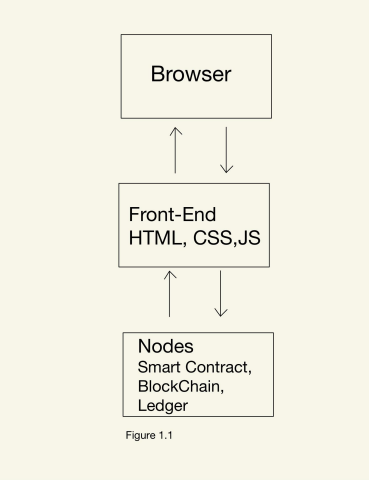

Figure 1.1

Bitpuen represents a pivotal transformation in digital advertising. From its inception in the thoughtful design phase using Figma to the integration of React for seamless interaction, and culminating in the development of a Web 3.0 DApp, every step reflects our dedication to user-centricity and decentralization. The fusion of these technologies, coupled with the deployment of smart contracts on the blockchain, embodies our commitment to transparency and security. Bitpuen's vision transcends mere innovation; it heralds a new era in digital advertising, where fairness, transparency, and technological ingenuity converge to redefine industry norms. As shown in Figure 1.1.

1.2 Functions

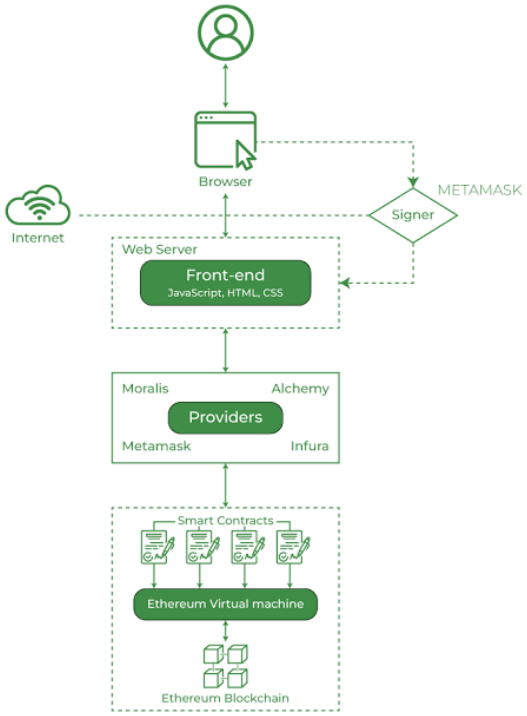

Figure 1.2.1

Bitpuen, a Digital Advertising Media Decentralized Application (DApp), revolutionizes advertising through decentralization. Utilizing Figma for intuitive design and React for seamless connectivity, Bitpuen prioritizes user experience. Embracing Web 3.0 principles, it empowers users by decentralizing control and fostering transparency. Smart contracts enable diverse advertisements and a fair voting system, driving engagement and authenticity.

Bitpuen offers a user-friendly interface, seamless connectivity, and decentralized control. Through Figma's design, React's connectivity, Web 3.0's principles, and smart contracts, Bitpuen redefines digital advertising, creating a transparent and engaging platform. As shown in Figure 1.2.1.

2.Requirement

To ensure the successful development and deployment of Bitpuen, the following tools, frameworks, platforms, and accounts are integral to the project:

2.1. **Remix IDE**:

Utilized for Ethereum smart contract development, Remix IDE offers an integrated development environment that facilitates writing, testing, and deploying smart contracts efficiently.

2.2. **Node.js**:

This runtime environment is essential for executing JavaScript code outside a web browser. Node.js enables seamless server-side scripting and serves as a fundamental component for various development processes in Bitpuen.

2.3. **React.js**:

React.js, a JavaScript library, is fundamental for creating dynamic user interfaces. It forms the core of Bitpuen's frontend development, ensuring an interactive and responsive user experience.

2.4. **Ethers.js**:

As a powerful JavaScript library for interacting with the Ethereum blockchain, Ethers.js plays a crucial role in Bitpuen's backend development. It facilitates seamless interaction with smart contracts and Ethereum transactions.

2.5. **Etherscan**:

This platform serves as a fundamental tool for Bitpuen, enabling developers and users to explore, verify, and monitor transactions and smart contracts on the Ethereum blockchain.

2.6. **MetaMask Account**:

A MetaMask account is essential for seamless integration with the Ethereum blockchain. It provides a user-friendly interface for managing Ethereum wallets and interacting with decentralized applications like Bitpuen.

2.7. **Sepolia Testnet**:

Utilized as a testing environment, the Sepolia testnet is crucial for conducting trials, debugging, and validating Bitpuen's functionalities before deployment. It ensures that the platform operates smoothly in a controlled environment before being introduced to the live network.

3.Mockup

Steps in Mock-up Creation for Bitpuen:

1. Define Objectives and Scope:
   Outline the specific goals and functionalities that the mock-up intends to represent. Identify the key features, user interactions, and essential elements required for Bitpuen's interface.

2. Gather Requirements and User Stories:
   Collect and analyze the project requirements, user stories, and use cases. Understand the needs and expectations of various user roles interacting with Bitpuen to create a comprehensive mock-up.

3. Sketch Initial Ideas:
   Begin with rough sketches or wireframes, capturing the layout and basic structure of Bitpuen's interface. Focus on arranging elements such as navigation, content areas, buttons, and forms.

4. Use Figma for Mock-up Creation:
   Leverage Figma's design capabilities to translate the initial sketches into a digital format. Utilize Figma's tools to create detailed designs, incorporating color schemes, typography, and graphical elements.

5. Design Core Functionalities:
   Implement the core functionalities within the mock-up. Design screens for viewing advertisements, creating new posts, voting mechanisms, user profiles, and any other critical features identified in the project requirements.

6. Iterate and Refine:
   Review the initial mock-up with stakeholders and collect feedback. Iterate on the design based on the received input, refining the interface to enhance usability, aesthetics, and user experience.

7. Create Interactive Prototypes:
   Use Figma to create interactive prototypes that simulate user interactions within the Bitpuen interface. Incorporate clickable elements to demonstrate navigation, buttons, and user flows.

8. Test and Validate:

Conduct usability tests using the interactive prototypes. Gather feedback from potential users or stakeholders to validate the mock-up's functionality, ease of use, and intuitiveness.

9. Iterate Based on Feedback:

Analyze the test results and feedback received during user testing. Iterate on the mock-up, making necessary adjustments and improvements to address any identified issues or areas for enhancement.

10. Finalize and Document:

Once the mock-up meets the desired standards and requirements, finalize the design. Document the design choices, user flows, and specifications to provide a comprehensive guide for the development team.

11. Present and Share:

Share the finalized mock-up with the project team and stakeholders for approval and alignment. Present the design choices, rationale, and functionalities to ensure everyone is on the same page before moving to development.

Mock-up images are shown in the Figure 3.1.1

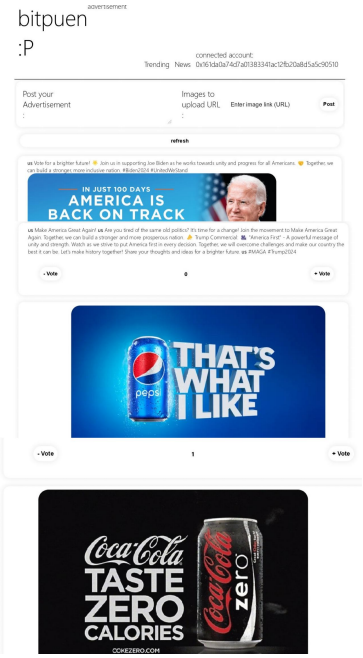

Figure 3.1.1 Mock-up of DApps

Steps to Use Bitpuen DApp

1. Access the Platform:

Open a compatible web browser and navigate to the Bitpuen website.

2. Install MetaMask:

If not already installed, add the MetaMask browser extension. Create a new MetaMask account or log in to an existing one. Ensure that your MetaMask account is connected to the Ethereum Mainnet or Sepolia Testnet, as required.

3. Explore the Interface:

Upon accessing Bitpuen, familiarize yourself with the platform's layout. Navigate through the different sections, such as advertisement viewing, post creation, and voting features.

4. View Advertisements:

Browse through the available advertisements posted by other users. Explore the diverse content presented on the platform.

5. Create Advertisements:

To create a new advertisement post:

– Click on the "Create Post" or relevant button.

– Input the necessary details, such as content, images (if applicable), and additional information required for the advertisement.

– Confirm the post creation and wait for it to be processed and added to the platform.

6. Vote on Posts:

Express your appreciation or critique by voting on posts:

– Choose a post you'd like to vote on.

– Use the provided voting options (upvote or downvote) to indicate your sentiment towards the post.

– Note that only one vote per user per post is allowed.

–When users click up vote or down vote, they will be asked to confirm their transaction in metamask, then after the transaction process is completed, users have to click refresh for the vote count to be updated

7. Interact with Smart Contracts:

Understand that the platform uses smart contracts to manage interactions. Be mindful of the system's rules, such as the single-vote-per-user-per-post policy enforced by smart contracts.

8. Monitor Transactions:

Utilize tools like Etherscan to track and verify your transactions, ensuring transparency and visibility into the blockchain interactions associated with Bitpuen.

9. Engage Responsibly:

Respect the platform's guidelines, adhere to ethical practices, and engage responsibly when creating content, voting on posts, and interacting with other users.

10. Provide Feedback:

Share your experiences and feedback regarding usability, functionalities, and any encountered issues with the platform's administrators or support channels.

11. Log Out and Secure Account:

After using Bitpuen, securely log out of your MetaMask account to protect your assets and personal information.

12. Stay Updated:

Keep an eye on announcements, updates, or changes made to Bitpuen. Stay informed about new features, improvements, or community guidelines.

When users arrive at the website, they will see a poster with no submit button, and connect to the metamask button. They have to click the connect to metamask button, then it will change to connected to : {that user wallet} , then users will be allowed to view the post in the website and the post button will appear in the poster.

Chapter 4

4. System Manual

4.1 Installation and Usage

Getting Started with Create React App

`npm start`

Runs the app in the development mode.
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.
See the section about running tests for more information.

`npm run build`

Builds the app for production to the `build` folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.
Your app is ready to be deployed!

See the section about deployment for more information.

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project.

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

Installation and Usage Guide

1. Installation Steps:

a. Setting up Node.js:

- Ensure Node.js is installed on your machine. If not, download and install Node.js from the official website.

b. Cloning the Bitpuen Repository:

- Clone the Bitpuen repository from the GitHub link provided or using the command:

```
git clone https://github.com/Bitpuen/Bitpuen.git
```

c. Installing Dependencies:

- Navigate to the project directory:

```
cd Bitpuen
```

- Install project dependencies using npm:

```

npm install

```

2. Usage Instructions:

a. Starting the Development Server:

- Start the development server to run Bitpuen locally:

```

npm start

```

- This command will launch the application on your local server (usually on http://localhost:3000).

b. MetaMask Integration:

- Ensure you have MetaMask installed in your browser.

- Log in to your MetaMask account and connect it to the Ethereum Mainnet or Sepolia Testnet as required for Bitpuen.

c. Exploring Bitpuen:

- Access Bitpuen via your web browser at the provided local server address.

- Explore the different sections such as viewing advertisements, creating new posts, and interacting with the voting system.

d. Creating Advertisements:

- Click on the "Create Post" button to create a new advertisement post.

- Input the required details, including content and optional images, to create your advertisement.

e. Voting on Posts:

- Navigate to a post you'd like to vote on.

- Use the provided upvote or downvote options to express your appreciation or critique for the post.

- Note that only one vote per user per post is allowed.

3. Additional Notes:

- **Smart Contract Interaction**: Bitpuen's functionalities rely on smart contracts. Ensure proper network connectivity and MetaMask integration for seamless interaction with the Ethereum blockchain.

- **Security Considerations**: Use Bitpuen responsibly, following ethical practices and respecting community guidelines. Be cautious with transactions and ensure the security of your MetaMask account.

4.2 Implementation Overview

4.2.1 Technology Stack

Bitpuen's implementation relies on a robust technology stack comprising various tools and frameworks:

- Remix IDE: Utilized for Ethereum smart contract development and deployment.

- Node.js: Facilitating server-side scripting and supporting backend development.

- React.js: Forming the core of the frontend development, ensuring a dynamic and interactive user interface.

- Ethers.js: Empowering backend functionalities by enabling seamless interaction with the Ethereum blockchain.

- Etherscan: Serving as a fundamental tool for transaction monitoring and smart contract verification.

- MetaMask: Providing a user-friendly interface for Ethereum wallet management and DApp interaction.

- Sepolia Testnet: Used as a testing environment for trialing and validating Bitpuen's functionalities before deployment.

4.2.2. Smart Contract Development

– Remix IDE: Used for writing, testing, and deploying Ethereum smart contracts that govern Bitpuen's functionalities, including post creation, voting mechanisms, and user interactions.

– Ethers.js: Integrated to interact with these smart contracts, enabling seamless communication between the frontend and the Ethereum blockchain.

4.2.3. Frontend Development

– React.js: Employed to create an intuitive and responsive user interface, allowing users to navigate through the platform, view advertisements, create posts, and interact with the voting system.

– MetaMask Integration: Integrated MetaMask to facilitate Ethereum wallet connectivity and transaction handling within the Bitpuen interface.

4.2.4. Backend Development:

– Node.js: Used for server-side scripting, managing data, and supporting backend operations required for Bitpuen's functionalities.

– Ethers.js Integration: Leveraged Ethers.js for backend processes that interact with the Ethereum blockchain, such as transaction handling and smart contract interactions.

4.2.5. Testing and Validation:

– Sepolia Testnet: Utilized as a controlled environment for conducting thorough testing, debugging, and validation of Bitpuen's functionalities before deploying to the live Ethereum network.

– User Testing: Involved potential users and stakeholders to gather feedback, ensuring usability, functionality, and performance met expectations.

4.2.6. Deployment and Monitoring:

– Etherscan: Utilized for monitoring and verifying transactions and smart contracts on the Ethereum blockchain, ensuring transparency and visibility post-deployment.

– Live Deployment: After successful testing and validation, Bitpuen deployed to the Ethereum Mainnet or selected network for public access.

4.2.7. Maintenance and Updates:

– Regular maintenance, updates, and improvements based on user feedback, technological advancements, and evolving industry standards to ensure Bitpuen's continued reliability and functionality.

4.3 Feature List

Our smart contract system is developed using Solidity in the Remix IDE. The contract comprises several essential functions aimed at enabling user interaction and management of posts on your website. The `createPost` function facilitates users in generating posts with or without images to promote their content on the website. These posts are assigned a Degree of Appreciation (DOA) value, which can be positive or negative.

The `isVoter` function acts as the primary security gate, ensuring that users can only upvote or downvote a post once. It checks whether a user has already voted on a specific post, thereby maintaining engagement and visibility while preventing multiple

votes from the same user. Subsequently, the `upVote` and `downVote` functions enable users to respectively increase or decrease the DOA associated with a post.

To access specific posts, your contract includes the `getPost` function, requiring the post's unique ID as an input. Additionally, there's the `getAllPost` function designed to retrieve all posts that have been posted on the platform.

Your contract utilizes Solidity compiler versions above 0.8.0, leveraging built-in overflow checking, thereby eliminating the necessity for safeMath. Security measures such as the `onlyOwner` modifier ensure that only the owner of a post can modify it. Furthermore, various `require()` functions are incorporated throughout the code to prevent unauthorized actions. For instance, both `downVote` and `upVote` functions validate the existence of the post and check whether the voter has already interacted with it.

Gas limit specifications are crucial for functions involving significant transactions, ensuring appropriate allocation of computational resources for these operations. These gas limits help manage the cost and execution of critical transactions within the Ethereum network.

Appendix: Methodology and Code Implementation

I. Introduction

The introduction section provides an overview of Bitpuen, highlighting its goals, objectives, and the need for a decentralized advertising platform. It sets the context for the implementation details covered in this report.

II. Implementation

Pseudo Code:

Smart Contract Interaction Example:

```solidity
// Pseudo code for voting mechanism in Bitpuen smart contract

// Define a structure to hold post details
struct Post {
    uint postId;
    string content;
    uint upvotes;
    uint downvotes;
    mapping(address => bool) votedUsers;
}

// Initialize posts mapping
mapping(uint => Post) public posts;

// Function to allow users to vote on a post
```

```
function voteOnPost(uint _postId, bool _upvote) public {
    require(!posts[_postId].votedUsers[msg.sender], "Already voted");

    if (_upvote) {
        posts[_postId].upvotes++;
    } else {
        posts[_postId].downvotes++;
    }

    posts[_postId].votedUsers[msg.sender] = true;
}
```

III. Feature Lists and Methodology Details

Feature Lists:

1. Advertisement Viewing: Users can browse and view advertisements posted on Bitpuen.
2. Post Creation: Users can create new advertisement posts, with or without images, contributing to the platform's content.
3. Voting Mechanism: Bitpuen's smart contracts facilitate a voting system where users can upvote or downvote posts, expressing their appreciation or critique.

Methodology Details:

– The methodology section details the approach used for implementing Bitpuen's functionalities, including the tools, technologies, and frameworks employed.
– It outlines the process for smart contract development, frontend and backend implementation, testing, deployment, and ongoing maintenance strategies.