มหาวิทยาลัยธรรมศาสตร์
THAMMASAT UNIVERSITY

DES484 Blockchain Development

Project: Word annoucement

MEMBERS

1. Praveenask Kammayee 6322773464

2. Purin Songamnuaykun 6322773183

3. Nanthawich Pitichaichan 6322773746

4. Phongsakorn Tang 6322773878

5.Thammatouch Poosawatratana 6322773407

PREPARED FOR

Dr. Watthanasak Jeamwatthanachai

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE

ENGINEERING IN DIGITAL ENGINEERING (INTERNATIONAL PROGRAM)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2023

TABLE OF CONTENTS

TABLE OF CONTENTS 1
LIST OF FIGURES 2
CHAPTER 1 INTRODUCTION 3
1.1 Software name and description 5
1.2 Background and Rationale 6
CHAPTER 2 Background AND Literature 8
2.1 Background and Development Approach 8
2.2 Related Software Development 11
CHAPTER 3 SOFTWARE DESIGN 13
3.1 Frontend Design (UX/UI) Wireframe 14
3.1.2 Mockup 15
3.1.3 Mockup*Confess Dapp Smart Contract* *16*
CHAPTER 4 SOFTWARE DEVELOPMENT 18
4.1 Frontend Development 18
4.2 Backend Development 28
CHAPTER 5 PROTOTYPE 33
5.1 Prototype 33
CHAPTER 6 DISCUSSION AND CONCLUSION *35*
6.1 Discussion 35
6.2 Conclusion 36
6.3 Further Development 37

LIST OF FIGURES

Figure	Page
Figure 1.1: Confess Dapp	4
Figure 3.1.1: Confess Dapp Wireframe	13
Figure 3.1.2.1: Confess Dapp Mockup	14
Figure 3.1.3.1: Confess Dapp Smart Contract on Remix IDE	15
Figure 4.1.1: index.css	17
Figure 4.1.2: App.css	19
Figure 4.1.3: Buy.css	21
Figure 4.1.4: Memos.css	23
Figure 4.2.1: Buy.jsx	25
Figure 4.2.2: Memos.jsx	27
Figure 4.2.3: App.jsx	29
Figure 4.2.4: main.jsx	31
Figure 5.1.1: prototype picture	33
Figure 5.1.2: prototype picture	34

CHAPTER 1 INTRODUCTION

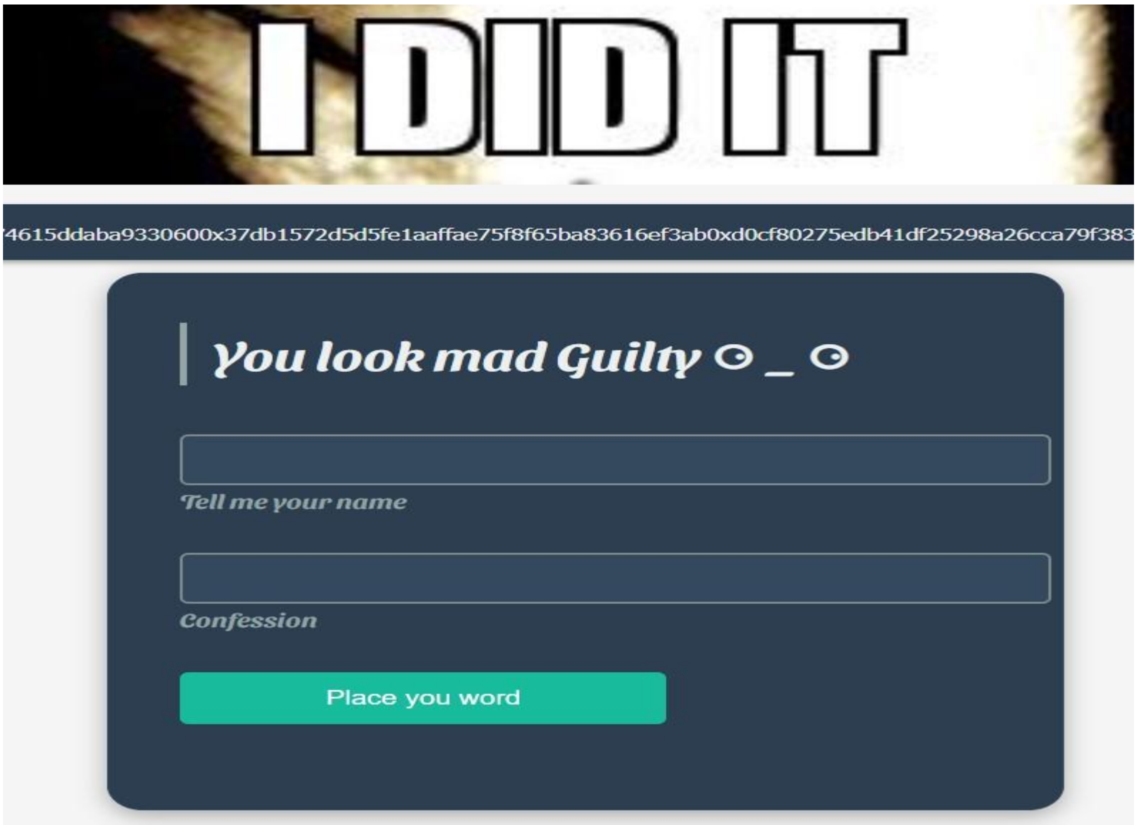
Confess DApp is a DApp that redefines community interaction. It's a space where members utilize their Metamask wallets and tokens to post impactful words or sentences, creating a decentralized tapestry of announcements.

Key Features that implemented in the Dapp:

1. Token-Powered Announcements: Your words, your tokens. Word Announcement ensures that every announcement is backed by the authenticity of your tokens, giving weight to each contribution.

2. Metamask Integration: Seamlessly connect with your Metamask wallet, making the posting process secure, convenient, and in your control.

3. Immutable Record on Blockchain: Every word etched in the blockchain—a tamper-proof record that guarantees transparency and authenticity for every announcement.



```
'4615ddaba9330600x37db1572d5d5fe1aaffae75f8f65ba83616ef3ab0xd0cf80275edb41df25298a26cca79f383
```

Figure 1.1: Confess Dapp

1.1 Software name and description

With the Confess DApp, you can share your ideas and confessions on the blockchain in a secure and permanent manner. Using Ethereum's strong network, this decentralized application (DApp) lets you preserve your messages with an emphasis on user privacy and simplicity. Every admission is documented with a timestamp, guaranteeing that your statements stay unaltered and available for as long as the blockchain is in existence.

1.2 Background and Rationale

Why We Made the "Confess" App

Nowadays, a lot of what we say online disappears fast. But sometimes, people want to say something that stays around for longer. That's why we built the "Confess" app.

Keeping Words Around Longer

We're used to posts and messages that go away after a little while. But the "Confess" app is different. It's a place where what you say stays put. It's like carving your words into a digital tree that never gets cut down.

Say It Out Loud

That picture with the words "I CONFESS I DID IT" is just like our app. You can say something big or small, serious or funny, and it'll stay there. It's for anyone who wants to make sure their words don't just vanish into thin air.

Trust in Technology

We use something called blockchain to make sure that once you say something, it can't be changed or taken back. It's like writing in pen instead of pencil. It means you can trust that your words will stay just as you left them.

Making It Fair

When you want to confess something, we ask for a little bit of money. This isn't to make a lot of cash but to keep the app running and make sure people think before they share. It's like putting a coin in a wishing well; it gives your words importance.

1.3 Expected Feature

What It Is:

Imagine a big digital board where everyone can see the messages people have left. When you use the "Confess" app, you can write something you want to share, and it gets pinned to this board for everyone to read.

How It Works:

You write your message in the app.
You send it off with a little bit of money (like buying a stamp for a letter).
Your message then shows up on the board for other people to see.
Once it's up there, it can't be changed or taken down. It's there for good.

Why It's Cool:

You can see what others have said and they can see your message too.
It's like a time capsule. One day, you can come back and see what you wrote a long time ago.
It's open to everyone, so all kinds of stories and secrets get shared.

CHAPTER 2 Background AND Literature

2.1 Background and Development Approach

Why We Started This:

We noticed that everything online is temporary. Comments, posts, chats — they all can disappear or change. But what if you want to say something that stays put? That's why we made Confess DApp. It's for anyone who wants to leave a message that lasts forever, like a digital time capsule.

What's Different About It:

Unlike social media, Confess DApp doesn't let messages just vanish. Once you put your words out there, they're set in stone (or, in this case, set in blockchain). It's a new way to be honest and open, without worrying about your words disappearing.

Development Approach for Confess DApp

Building It Simple:

We wanted to make sure anyone could use Confess DApp. So we kept it simple. No complicated steps — just write your message, pay a small fee (to keep things running smoothly), and hit send. That's it. Your words are now part of the blockchain.

Keeping It Secure:

Security is a big deal. We use blockchain technology because it's really secure. When you confess something, it's like locking it in a safe that everyone can see into but no one can mess with.

Making It Fair:

We don't want just a bunch of random stuff on Confess DApp. The small fee makes sure that people think before they share. And it also helps us keep the app working well for everyone.

Looking Ahead:

We're starting with confessions, but we've got big plans. We're thinking about ways to let people react to messages or even have private confessions just for them. But don't worry, we'll keep it easy and safe, just like it is now.

Case Study: Digital Identity Management on Blockchain

This case study explores the use of blockchain technology in managing digital identities, offering insights into how these concepts can be applied to the "Confess" DApp for ensuring user privacy and security.

Background

In the digital world, identity management is a critical issue, involving the protection of personal data and verification of user identities. Traditional systems often face challenges like data breaches, identity theft, and lack of user control over personal information.

Blockchain Solution

A blockchain-based digital identity management system was introduced to address these challenges. This system utilized decentralized ledger technology to give users full control over their personal information.

Key Features

Immutable Records: Once personal data is recorded on the blockchain, it cannot be altered or deleted, ensuring data integrity.
User-Controlled Privacy: Users have the power to decide what information they share and with whom, enhancing privacy.
Decentralized Verification: Identity verification is done without the need for a central authority, reducing the risk of data breaches and fraud.

Transparency and Security: The blockchain's transparent nature, combined with strong encryption, ensures that user data is both visible (where appropriate) and secure.

Implementation

The system was implemented in a pilot program with a group of users. Each user created a digital identity on the blockchain, which they could use to authenticate themselves for various online services.

Results

Increased Security: There were no reported instances of identity theft or data breaches within the system.

User Empowerment: Users reported feeling more in control of their personal data.

Efficiency in Verification: The time and resources needed for identity verification were significantly reduced.

Relevance to Confess DApp

The principles of this blockchain-based digital identity management system are highly relevant to the "Confess" DApp. The DApp can leverage similar features to ensure that users' confessions remain anonymous yet verifiable, ensuring a secure and private platform for expression.

Applying Immutable Records: Ensuring that once a confession is made, it remains unaltered on the blockchain.

Enhancing User Privacy: Users can share confessions without revealing their identity, knowing their data is secure and private.

Learning from Decentralized Systems: Implementing decentralized systems for managing confessions, removing any single point of failure or control.

2.2 Related Software Development

The development integrates Ethereum blockchain technology for recording confessions.

It uses smart contracts, like the one provided in your Solidity code, to handle transactions and store messages immutably.

Front-End Development with React:

The project likely uses React (as suggested by the .jsx files) for building its user interface. This allows for a dynamic, responsive web application that interacts with the blockchain.

The front-end captures user inputs (confessions) and displays data from the blockchain, providing an interactive user experience.

Styling with CSS:

The CSS files indicate a focus on the visual and interactive aspects of the application. This includes designing a user-friendly and aesthetically pleasing interface.

Responsive design ensures the app is accessible across different devices and screen sizes.

Security and Privacy Considerations:

Given the sensitive nature of confessions, the developer must prioritize security features to protect user data.

Implementing encryption and anonymity features ensures that while confessions are public, user identities remain protected.

Web3 Integration:

The application likely utilizes Web3 technologies to connect the React front-end with the Ethereum blockchain, enabling users to interact with the smart contract.

This involves handling cryptocurrency transactions, wallet integration for users to pay the required fee for their confessions, and reading data from the blockchain.

Testing and Deployment:

Rigorous testing, including unit tests and integration tests, ensures the reliability and performance of the smart contract and the overall application.

Continuous integration and deployment practices allow for efficient updates and maintenance of the application.

User Experience (UX) Focus:

The application design centers around a seamless and intuitive user experience, crucial for non-technical users engaging with blockchain technology.

It includes clear navigation, easy-to-understand instructions, and feedback on user actions (such as successful transaction confirmation).

Scalability and Performance:

Considering the potentially high volume of transactions and data, the app's architecture is designed for scalability.

This could involve optimizing smart contract code for efficiency and exploring layer 2 solutions for Ethereum to improve transaction speed and reduce costs.

Legal and Ethical Compliance:

Adherence to regulations, particularly those concerning digital transactions and user data, is a critical aspect of development.

Ethical considerations, especially in handling user-generated content in a public and permanent manner, are addressed.

Community and Feedback Loop:

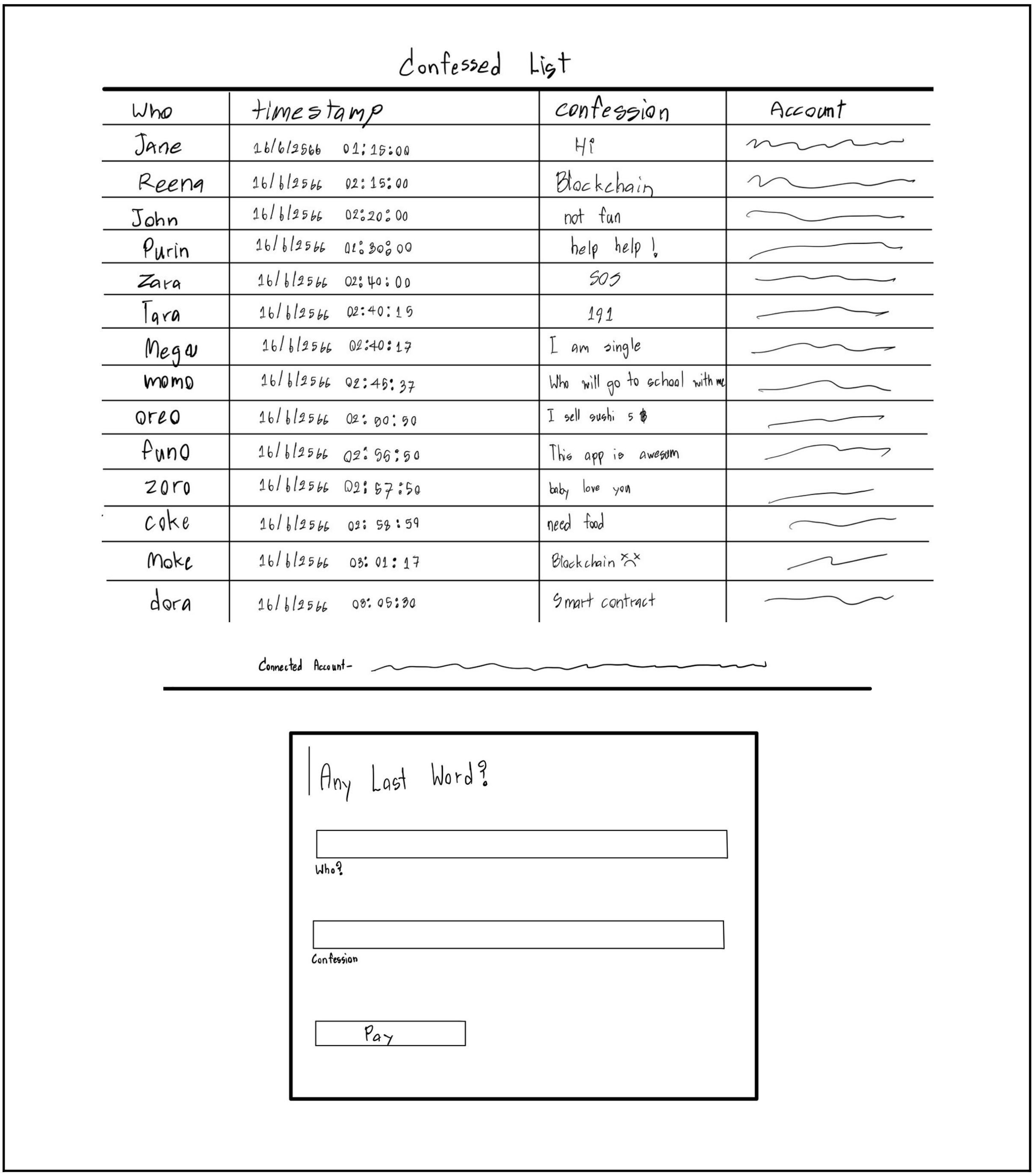
The development process incorporates user feedback to evolve the app features and user interface.

Engaging with the user community for suggestions and improvements forms a part of the development lifecycle.

CHAPTER 3 SOFTWARE DESIGN

3.1 Frontend Design (UX/UI) Wireframe, Mockup

3.1.1Wireframe



Confessed List

Who	timestamp	confession	Account
Jane	16/6/2566 01:15:00	Hi	
Reena	16/6/2566 02:15:00	Blockchain	
John	16/6/2566 02:20:00	not fun	
Purin	16/6/2566 02:30:00	help help !	
Zara	16/6/2566 02:40:00	SOS	
Tara	16/6/2566 02:40:15	191	
Mega	16/6/2566 02:40:17	I am single	
momo	16/6/2566 02:45:37	Who will go to school with me	
oreo	16/6/2566 02:50:50	I sell sushi 5 ฿	
funo	16/6/2566 02:56:50	This app is awesom	
zoro	16/6/2566 02:57:50	baby love you	
coke	16/6/2566 02:58:59	need food	
Moke	16/6/2566 03:01:17	Blockchain	
dora	16/6/2566 03:05:30	Smart contract	

Connected Account-

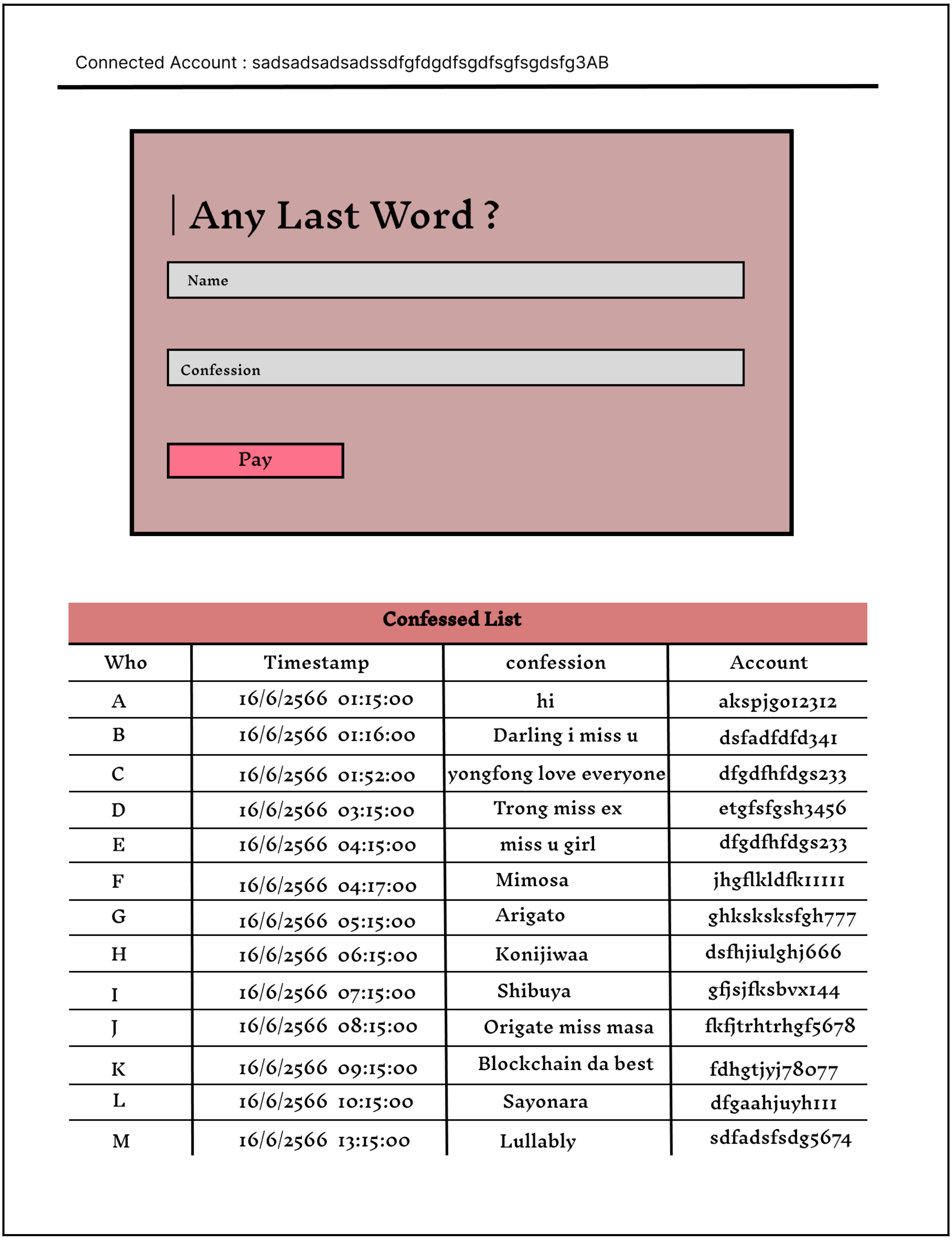
Any Last Word?

Who?

Confession

Figure 3.3.1.1: Confess Dapp Wireframe

3.1.2 Mockup



Connected Account : sadsadsadsadssdfgfdgdfsgdfsgfsgdsfg3AB

Any Last Word ?

Name

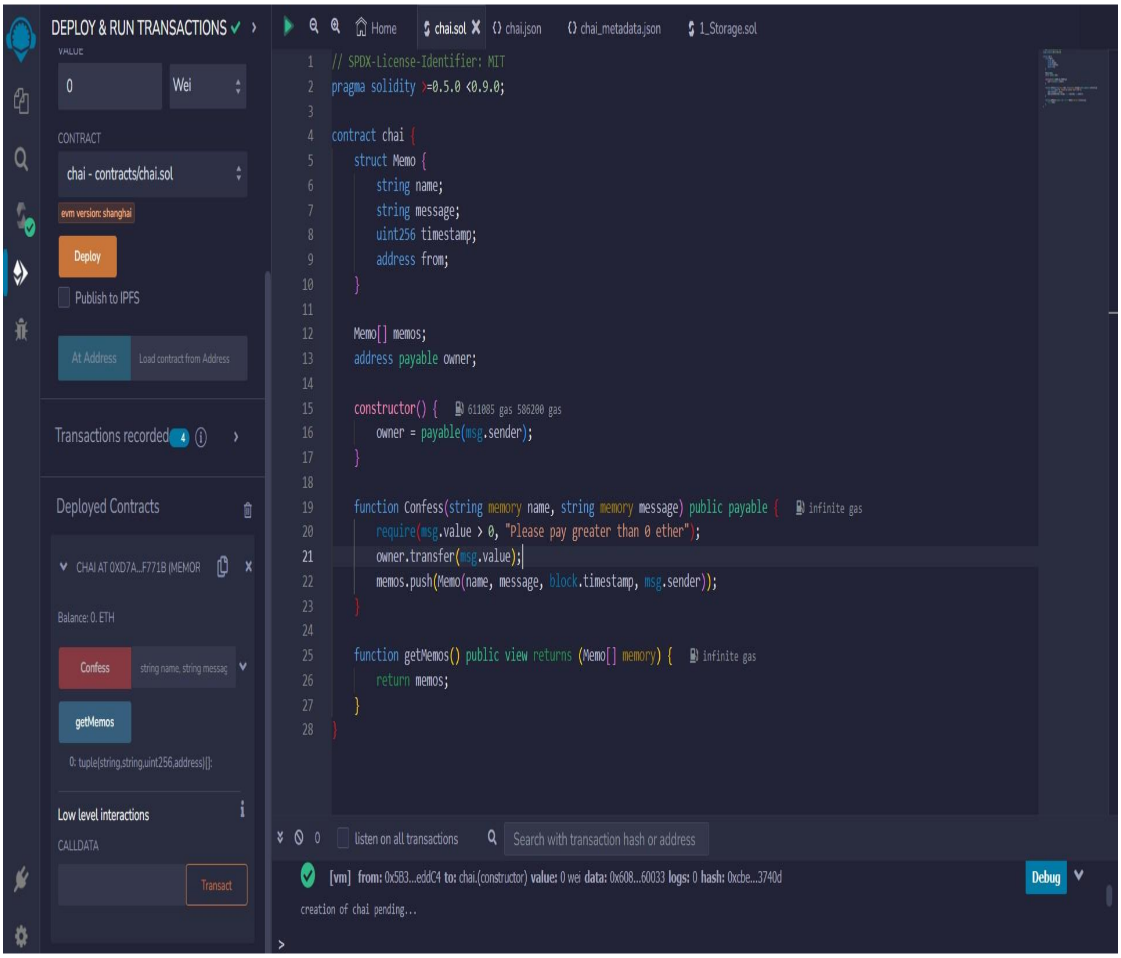
Confession

Pay

Confessed List			
Who	Timestamp	confession	Account
A	16/6/2566 01:15:00	hi	akspjg012312
B	16/6/2566 01:16:00	Darling i miss u	dsfadfdfd341
C	16/6/2566 01:52:00	yongfong love everyone	dfgdfhfdgs233
D	16/6/2566 03:15:00	Trong miss ex	etgfsfgsh3456
E	16/6/2566 04:15:00	miss u girl	dfgdfhfdgs233
F	16/6/2566 04:17:00	Mimosa	jhgflkldfk11111
G	16/6/2566 05:15:00	Arigato	ghksksksfgh777
H	16/6/2566 06:15:00	Konijiwaa	dsfhjiulghj666
I	16/6/2566 07:15:00	Shibuya	gfjsjfksbvx144
J	16/6/2566 08:15:00	Origate miss masa	fkfjtrhtrhgf5678
K	16/6/2566 09:15:00	Blockchain da best	fdhgtjyj78077
L	16/6/2566 10:15:00	Sayonara	dfgaahjuyh111
M	16/6/2566 13:15:00	Lullably	sdfadsfsdg5674

Figure 3.1.2.1: Confess Dapp Mockup

3.1.3 Mockup*Confess Dapp Smart Contract*



VALUE: 0 Wei

CONTRACT: chai - contracts/chai.sol

evm version: shanghai

☐ Publish to IPFS

CHAI AT 0XD7A...F771B (MEMOR

Balance: 0. ETH

Confess: string name, string messag

0: tuple(string,string,uint256,address)[]:

CALLDATA

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0 <0.9.0;

contract chai {
    struct Memo {
        string name;
        string message;
        uint256 timestamp;
        address from;
    }

    Memo[] memos;
    address payable owner;

    constructor() {    611085 gas 586200 gas
        owner = payable(msg.sender);
    }

    function Confess(string memory name, string memory message) public payable {    infinite gas
        require(msg.value > 0, "Please pay greater than 0 ether");
        owner.transfer(msg.value);
        memos.push(Memo(name, message, block.timestamp, msg.sender));
    }

    function getMemos() public view returns (Memo[] memory) {    infinite gas
        return memos;
    }
}
```

```
0  ☐ listen on all transactions   Search with transaction hash or address
[vm] from: 0x5B3...eddC4 to: chai.(constructor) value: 0 wei data: 0x608...60033 logs: 0 hash: 0xcbe...3740d
creation of chai pending...
>
```

Figure 3.1.3.1: Confess Dapp Smart Contract on Remix IDE

Contract Structure and Initialization

1. <u>Memo Structure:</u> The contract defines a struct called Memo, which is a custom data type for storing individual memos. Each Memo has four attributes:
   - ➔ **name:** A string representing the name associated with the memo.
   - ➔ **message:** The message content of the memo.
   - ➔ **timestamp:** A Unix timestamp marking when the memo was created.
   - ➔ **from:** The Ethereum address of the user who created the memo.
2. <u>Memos Array:</u> The contract maintains an array of Memo structs, named memos. This array stores all the memos created and sent to the contract.

3. Owner Address: There is an address payable variable named owner. This is set in the constructor to be the Ethereum address of the user who deploys the contract.
4. Constructor: The constructor is a special function that runs only once when the contract is deployed. It initializes the owner variable with the address of the contract deployer (msg.sender).

Contract Functions

1. Confess Function: The Confess function allows users to create a new memo. It has several key features:

   Parameters: Takes two strings, name and message, as input.

   Payable: It's a payable function, meaning it allows the function to receive Ether (ETH).

   Validation: The function requires that some ETH (more than 0) is sent with the transaction, enforced by the required statement.

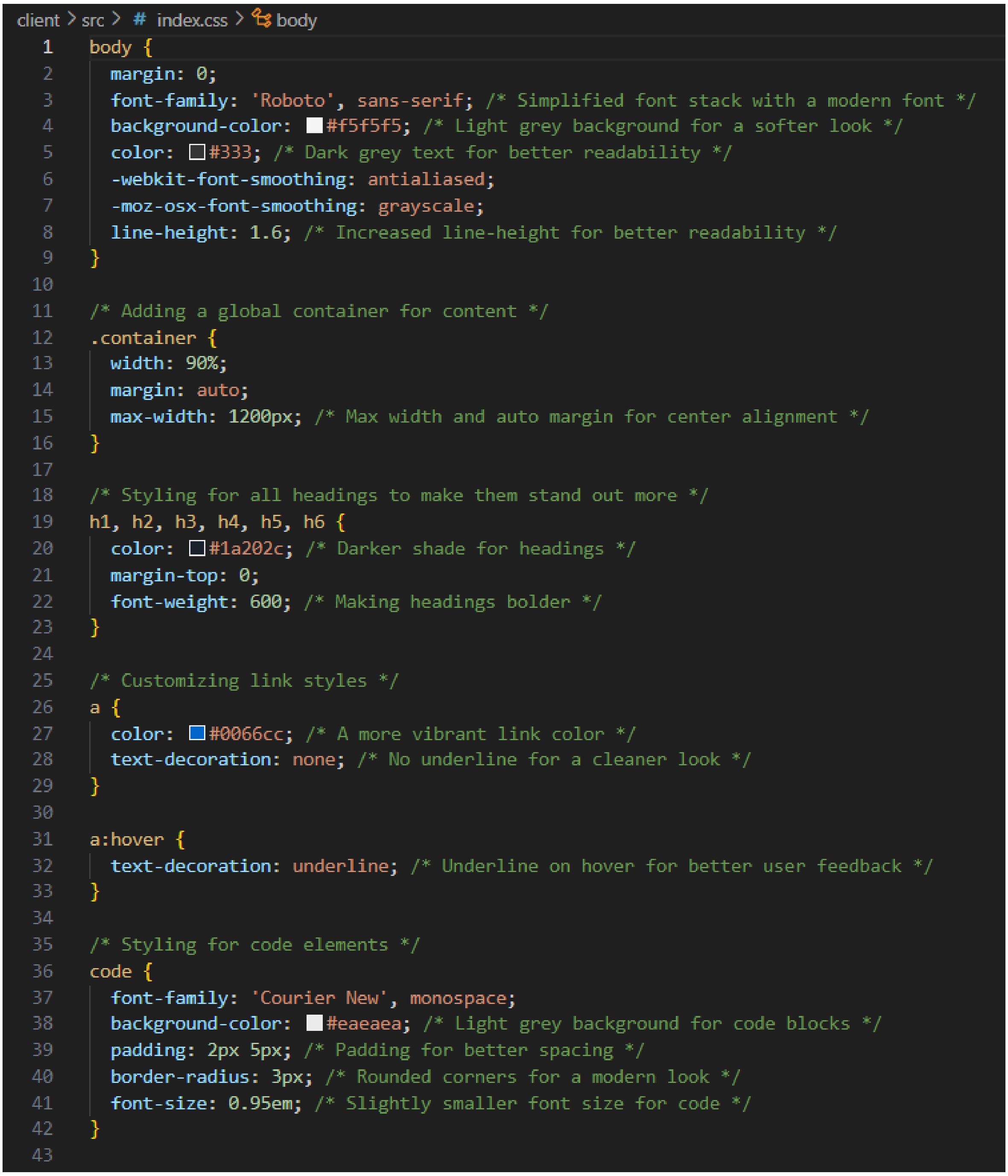
   Transfer of Funds: Any ETH sent with the transaction is immediately transferred to the owner of the contract.

   Memo Creation: A new Memo is created with the provided name, message, the current timestamp (block.timestamp), and the sender's address (msg.sender). This Memo is then added to the memos array.

2. getMemos Function: This is a view function that returns the entire array of memos. Being a view function, it doesn't modify the state of the blockchain and doesn't require any gas to execute. This allows anyone to retrieve and view all the memos stored in the contract.

CHAPTER 4 SOFTWARE DEVELOPMENT

4.1 Frontend Development



```css
body {
  margin: 0;
  font-family: 'Roboto', sans-serif; /* Simplified font stack with a modern font */
  background-color: #f5f5f5; /* Light grey background for a softer look */
  color: #333; /* Dark grey text for better readability */
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  line-height: 1.6; /* Increased line-height for better readability */
}

/* Adding a global container for content */
.container {
  width: 90%;
  margin: auto;
  max-width: 1200px; /* Max width and auto margin for center alignment */
}

/* Styling for all headings to make them stand out more */
h1, h2, h3, h4, h5, h6 {
  color: #1a202c; /* Darker shade for headings */
  margin-top: 0;
  font-weight: 600; /* Making headings bolder */
}

/* Customizing link styles */
a {
  color: #0066cc; /* A more vibrant link color */
  text-decoration: none; /* No underline for a cleaner look */
}

a:hover {
  text-decoration: underline; /* Underline on hover for better user feedback */
}

/* Styling for code elements */
code {
  font-family: 'Courier New', monospace;
  background-color: #eaeaea; /* Light grey background for code blocks */
  padding: 2px 5px; /* Padding for better spacing */
  border-radius: 3px; /* Rounded corners for a modern look */
  font-size: 0.95em; /* Slightly smaller font size for code */
}

```

Figure 4.1.1: index.css

1.Body Reset and Font:

The 'margin: 0;' directive ensures a consistent layout for the entire webpage.
The 'font-family' property emphasizes a modern aesthetic with the 'Roboto' font, contributing to a sleek appearance.

2.Background and Text Color:

The background color '#f5f5f5' and dark gray text color '#333' are meticulously chosen for a softer and more readable visual experience.

3.Readability Enhancements:

The 'line-height: 1.6;' is strategically implemented to enhance overall readability.

4.Global Container:

A global container class is introduced with a width of 90%, centered using 'margin: auto,' and a maximum width of 1200px, optimizing content presentation.

5.Headings Styling:

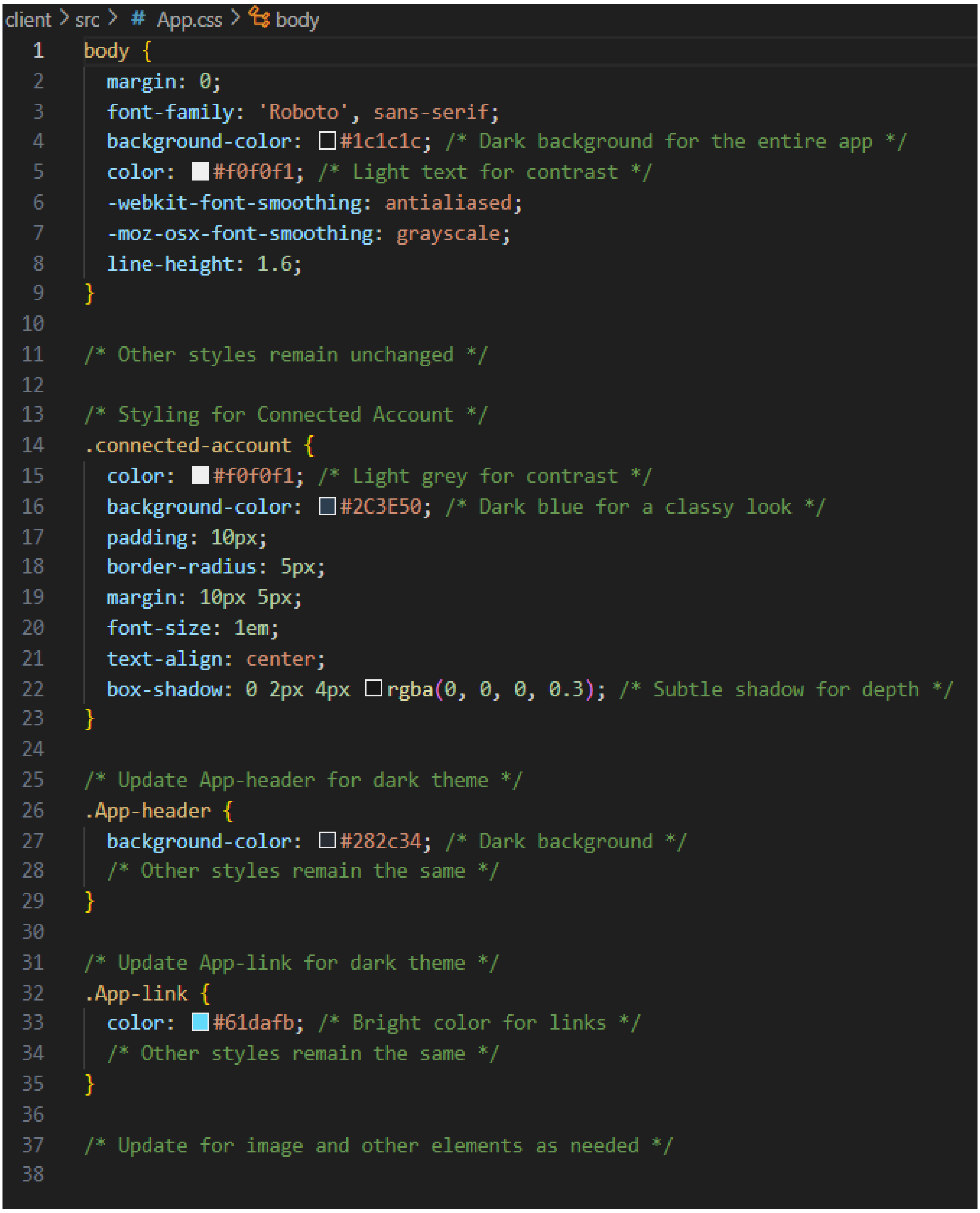
Headings from h1 to h6 are styled with a darker shade, increased font weight, and no top margin, ensuring prominence and a consistent visual hierarchy.

6.Link Styles:

Customized link styles are implemented to enhance user engagement.

7.Code Elements Styling:

Code elements receive special attention with 'Courier New' font, light gray background, padding, rounded corners, and a slightly smaller font size for clarity.



client > src > # App.css > body

```css
body {
  margin: 0;
  font-family: 'Roboto', sans-serif;
  background-color: #1c1c1c; /* Dark background for the entire app */
  color: #f0f0f1; /* Light text for contrast */
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  line-height: 1.6;
}

/* Other styles remain unchanged */

/* Styling for Connected Account */
.connected-account {
  color: #f0f0f1; /* Light grey for contrast */
  background-color: #2C3E50; /* Dark blue for a classy look */
  padding: 10px;
  border-radius: 5px;
  margin: 10px 5px;
  font-size: 1em;
  text-align: center;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3); /* Subtle shadow for depth */
}

/* Update App-header for dark theme */
.App-header {
  background-color: #282c34; /* Dark background */
  /* Other styles remain the same */
}

/* Update App-link for dark theme */
.App-link {
  color: #61dafb; /* Bright color for links */
  /* Other styles remain the same */
}

/* Update for image and other elements as needed */

```

Figure 4.1.2: App.css

1.Body Styling:

The 'body' styling establishes a clean canvas with a dark background color (#1c1c1c) for the entire app, creating a visually striking backdrop.

Light text color (#f0f0f1) is applied for optimal contrast, ensuring readability.

2.Readability Enhancements:

The 'line-height: 1.6;' and font properties are carefully selected to enhance overall readability.

3.Connected Account Styling:

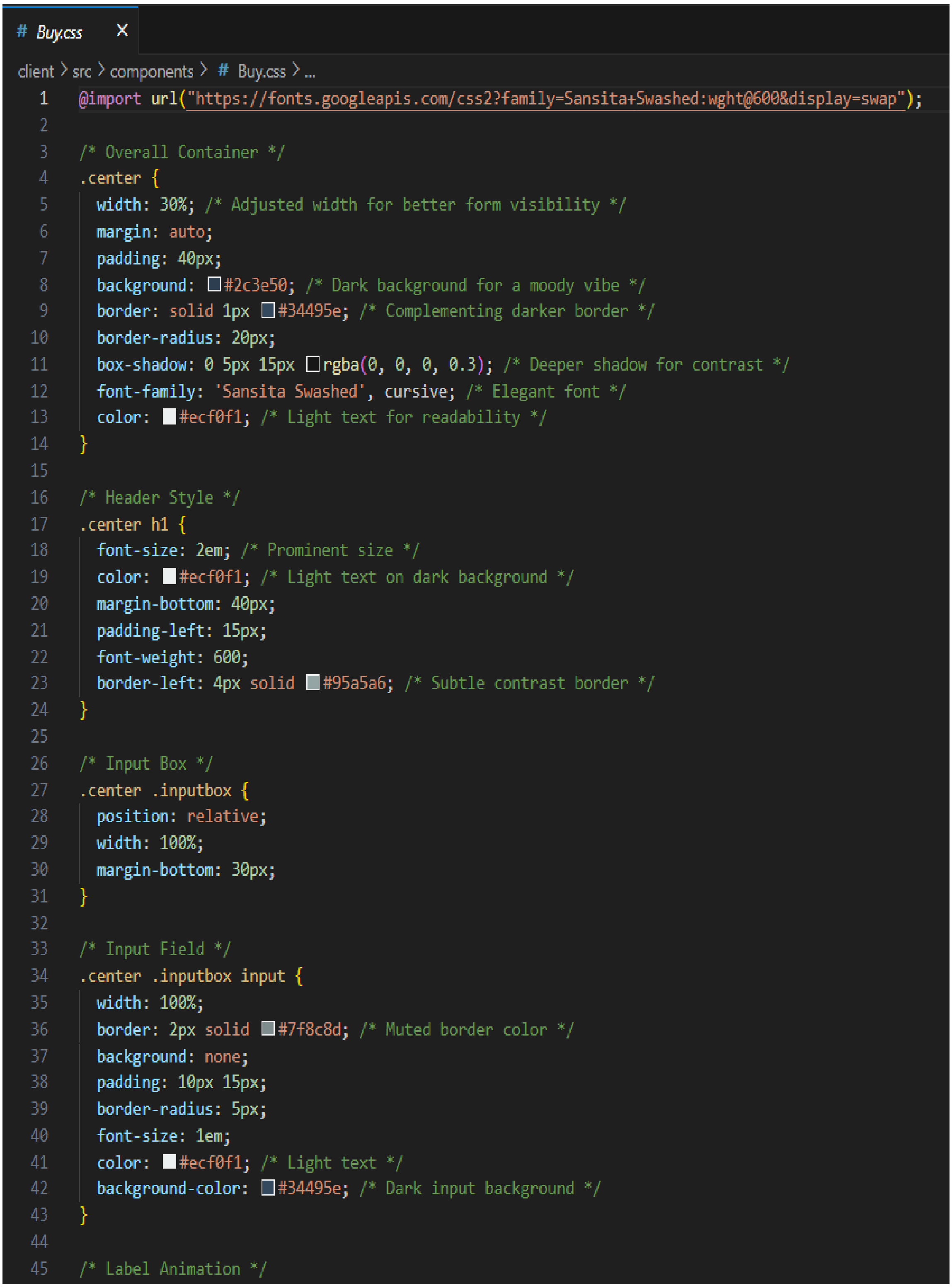
A distinctive style is introduced for the 'connected-account' class, featuring light grey text on a dark blue background (#2C3E50).

Padding, border-radius, and a subtle box shadow are applied for a polished and classy appearance.

4.Header and Link Styles Update:

'App-header' and 'App-link' styles are updated to seamlessly integrate into the dark theme.

Adjusted background and text colors ensure consistency within the overall design.



```css
@import url("https://fonts.googleapis.com/css2?family=Sansita+Swashed:wght@600&display=swap");

/* Overall Container */
.center {
  width: 30%; /* Adjusted width for better form visibility */
  margin: auto;
  padding: 40px;
  background: #2c3e50; /* Dark background for a moody vibe */
  border: solid 1px #34495e; /* Complementing darker border */
  border-radius: 20px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3); /* Deeper shadow for contrast */
  font-family: 'Sansita Swashed', cursive; /* Elegant font */
  color: #ecf0f1; /* Light text for readability */
}

/* Header Style */
.center h1 {
  font-size: 2em; /* Prominent size */
  color: #ecf0f1; /* Light text on dark background */
  margin-bottom: 40px;
  padding-left: 15px;
  font-weight: 600;
  border-left: 4px solid #95a5a6; /* Subtle contrast border */
}

/* Input Box */
.center .inputbox {
  position: relative;
  width: 100%;
  margin-bottom: 30px;
}

/* Input Field */
.center .inputbox input {
  width: 100%;
  border: 2px solid #7f8c8d; /* Muted border color */
  background: none;
  padding: 10px 15px;
  border-radius: 5px;
  font-size: 1em;
  color: #ecf0f1; /* Light text */
  background-color: #34495e; /* Dark input background */
}

/* Label Animation */
```

Figure 4.1.3: Buy.css

1.Font and Background:

The 'Sansita Swashed' font, imported from Google Fonts, adds an elegant touch to the overall container.
A dark background (#2c3e50) is chosen to create a moody vibe, setting the tone for the form interface.

2.Container Styling:

The container is adjusted to 30% width for improved form visibility.
Rounded corners, a subtle border, and a deeper shadow are applied for a stylish and contrasting appearance.

3.Header Styling:

The header ('h1') is styled with a prominent size, featuring light text color (#ecf0f1) for optimal readability.
A subtle left border is introduced to provide visual contrast and interest.

4.Input Box Styling:

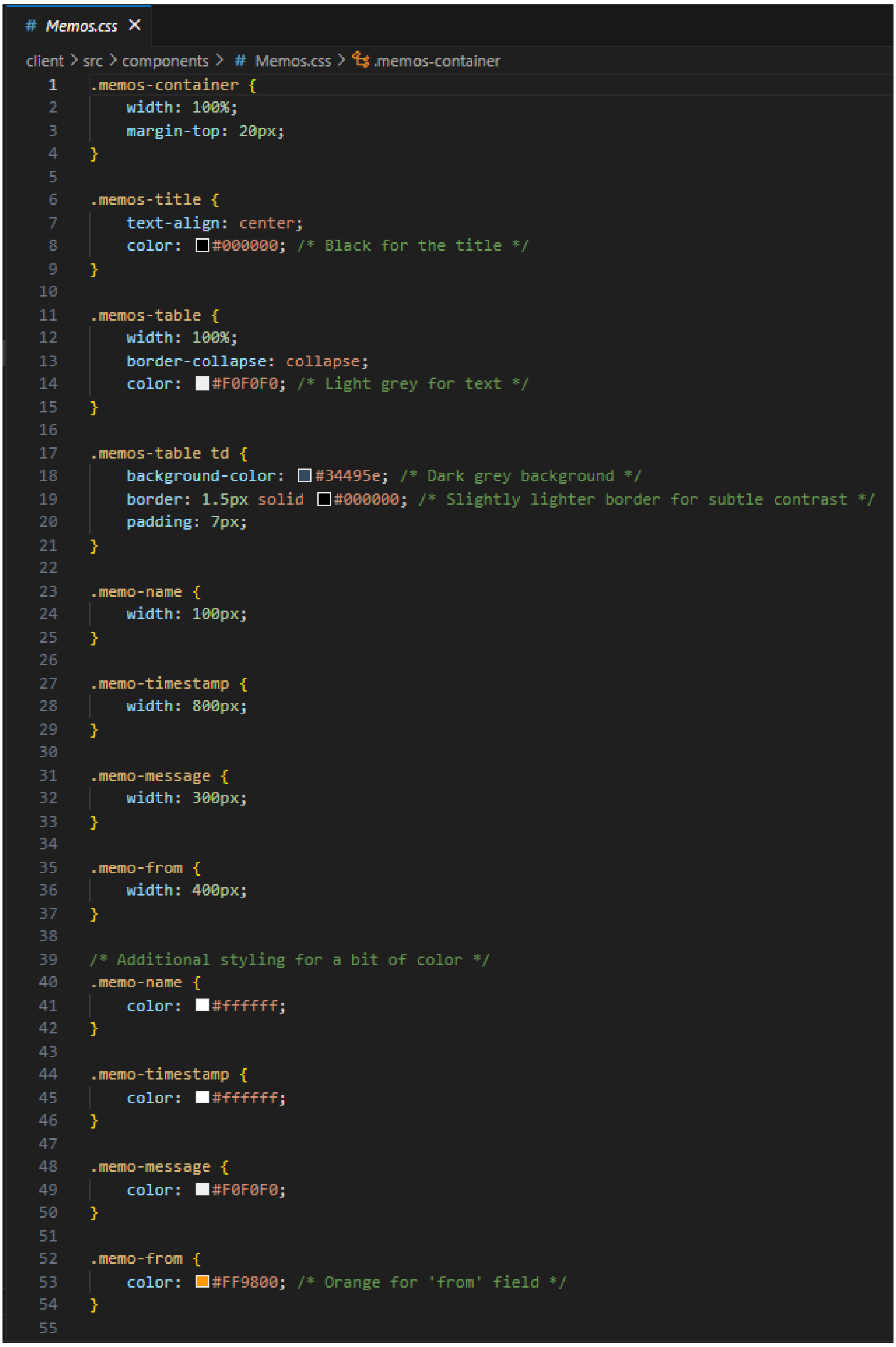
The input box is styled with a dark background, contributing to the overall aesthetic.
Muted border color enhances the form's visual appeal.
Individual input fields have a distinct style on focus or when valid, enhancing the interactive user experience.

5.Label Animation and Focus Styles:

Label animation is implemented to provide a polished and visually appealing form interaction.Focus styles ensure a smooth and responsive user experience.

6.Submit Button Styling:

The submit button has a calm green tone, creating a cohesive color scheme.
A subtle blue hover effect adds interactivity and engagement to the overall form design.



```css
.memos-container {
    width: 100%;
    margin-top: 20px;
}

.memos-title {
    text-align: center;
    color: #000000; /* Black for the title */
}

.memos-table {
    width: 100%;
    border-collapse: collapse;
    color: #F0F0F0; /* Light grey for text */
}

.memos-table td {
    background-color: #34495e; /* Dark grey background */
    border: 1.5px solid #000000; /* Slightly lighter border for subtle contrast */
    padding: 7px;
}

.memo-name {
    width: 100px;
}

.memo-timestamp {
    width: 800px;
}

.memo-message {
    width: 300px;
}

.memo-from {
    width: 400px;
}

/* Additional styling for a bit of color */
.memo-name {
    color: #ffffff;
}

.memo-timestamp {
    color: #ffffff;
}

.memo-message {
    color: #F0F0F0;
}

.memo-from {
    color: #FF9800; /* Orange for 'from' field */
}

```

Figure 4.1.4: Memos.css

1.Container Presentation:

The 'memos-container' spans the full width of its container and includes a top margin for effective spacing.

This creates a clean and organized presentation for the memos.

2.Title Styling:

The 'memos-title' is centered and styled with black text, providing a clear and visually distinct title for the memos container.

3.Table Design:

The 'memos-table' is designed with a dark gray background, contributing to a cohesive and organized visual presentation.

Slightly lighter borders are implemented for subtle contrast, and light gray text ensures readability.

4.Cell Padding for Spacing:

Each table cell ('td') is configured with padding for better spacing, improving the overall layout.

5.Column Width Assignment:

Specific widths are assigned to different columns, such as 'memo-name,' 'memo-timestamp,' 'memo-message,' and 'memo-from.'

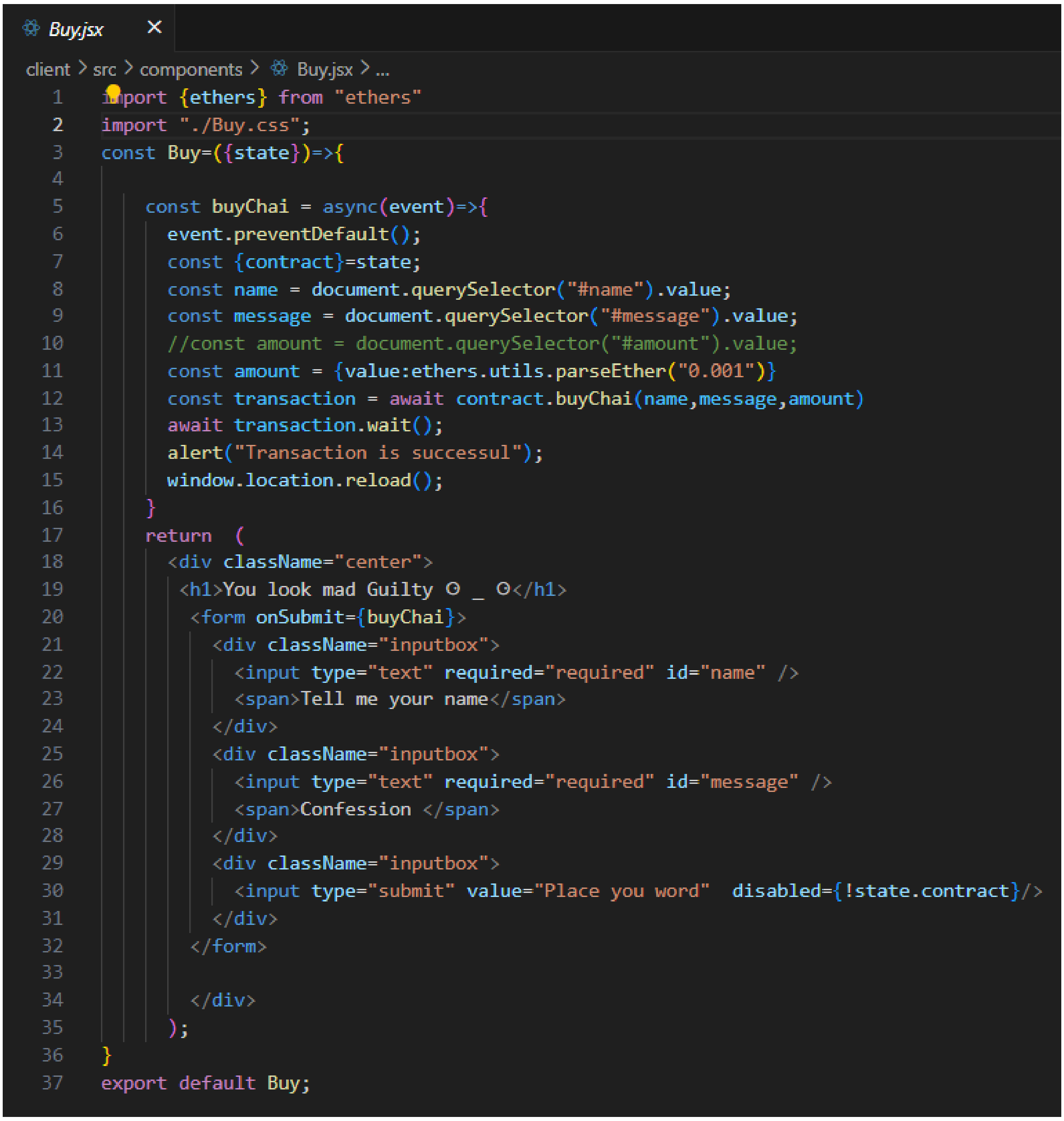
This deliberate structuring enhances the overall organization and readability of the memo display.

6.Individual Element Styling:

Individual elements within columns are styled with specific colors to enhance visual appeal.

White color is applied for 'memo-name,' 'memo-timestamp,' and 'memo-message,' while orange is chosen for the 'memo-from' field.

4.2 Backend Development



```
import {ethers} from "ethers"
import "./Buy.css";
const Buy=({state})=>{

    const buyChai = async(event)=>{
      event.preventDefault();
      const {contract}=state;
      const name = document.querySelector("#name").value;
      const message = document.querySelector("#message").value;
      //const amount = document.querySelector("#amount").value;
      const amount = {value:ethers.utils.parseEther("0.001")}
      const transaction = await contract.buyChai(name,message,amount)
      await transaction.wait();
      alert("Transaction is successul");
      window.location.reload();
    }
    return  (
      <div className="center">
       <h1>You look mad Guilty ʘ _ ʘ</h1>
        <form onSubmit={buyChai}>
          <div className="inputbox">
            <input type="text" required="required" id="name" />
            <span>Tell me your name</span>
          </div>
          <div className="inputbox">
            <input type="text" required="required" id="message" />
            <span>Confession </span>
          </div>
          <div className="inputbox">
            <input type="submit" value="Place you word"  disabled={!state.contract}/>
          </div>
        </form>

        </div>
      );
}
export default Buy;
```

Figure 4.2.1: Buy.jsx

1.Component Definition:

This JavaScript and React code define a component named Buy, responsible for allowing users to submit confessions using a form.

2.Imports: The component imports the ethers library from the "ethers" package for Ethereum-related functionality.
Additionally, a CSS file named "Buy.css" is imported for styling.

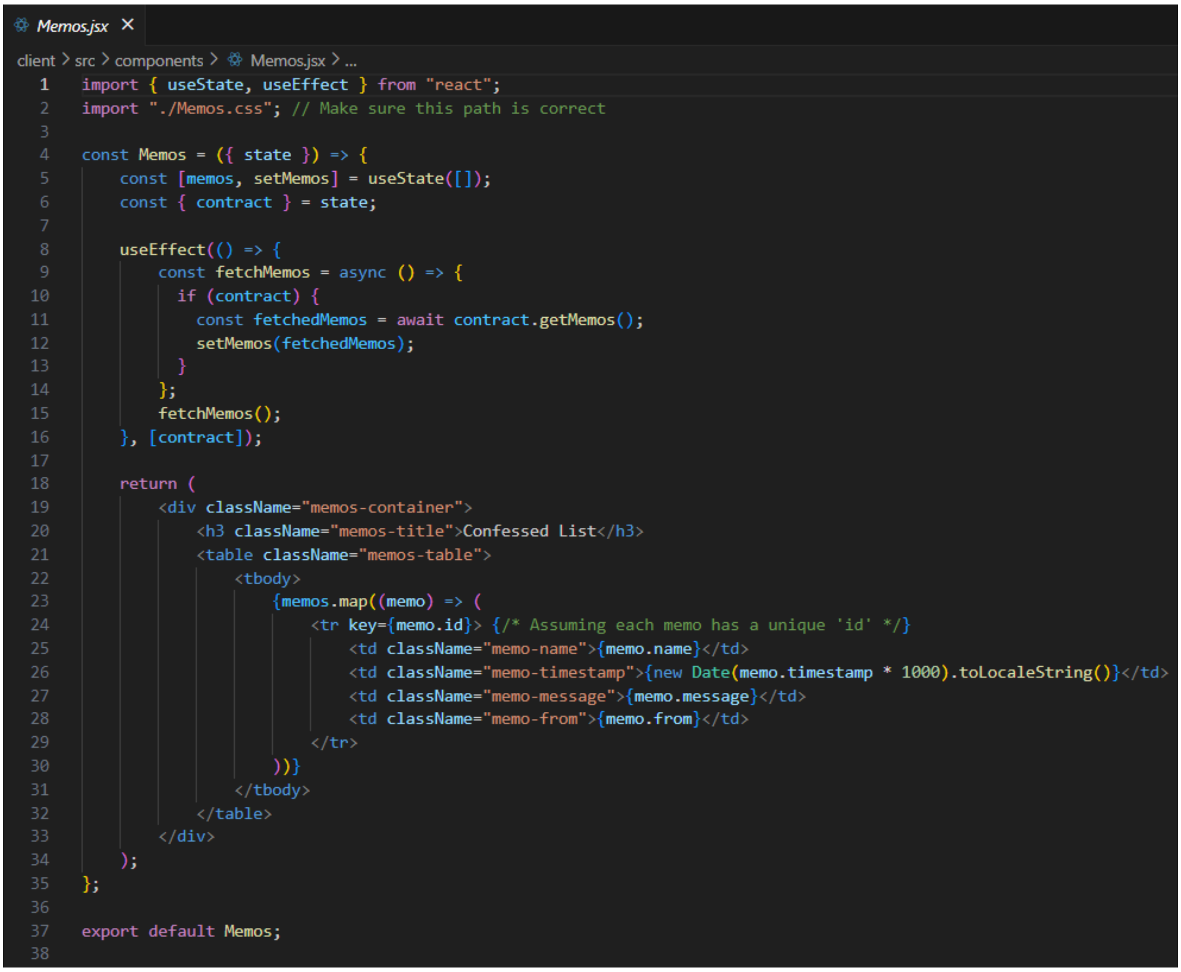
3.Functionality:
Inside the Buy component, there's a function named buyChai that is triggered when the form is submitted.
This function retrieves the user's name and confession message from input fields, sets a fixed amount (0.001 ethers) for the transaction, and initiates a transaction to buy "Chai" using the provided contract.

4.Transaction Handling:
After the transaction is confirmed, a success alert is displayed, and the page is reloaded.

5.JSX Structure:
The JSX returned by the Buy component consists of a form with input fields for the user's name and message, a submit button, and additional styling elements.
The form has an onSubmit event handler that invokes the buyChai function.
The disabled attribute of the submit button is set based on the presence of the contract in the state object.

6.Component Display:
The component includes a title and a brief message to provide context and guidance for users.

7.Interaction with Ethereum Contract:
Overall, this component is designed for users to submit confessions through a form, interacting with an Ethereum contract represented by the state.contract object in the React component's state.



```
import { useState, useEffect } from "react";
import "./Memos.css"; // Make sure this path is correct

const Memos = ({ state }) => {
    const [memos, setMemos] = useState([]);
    const { contract } = state;

    useEffect(() => {
        const fetchMemos = async () => {
          if (contract) {
            const fetchedMemos = await contract.getMemos();
            setMemos(fetchedMemos);
          }
        };
        fetchMemos();
    }, [contract]);

    return (
        <div className="memos-container">
            <h3 className="memos-title">Confessed List</h3>
            <table className="memos-table">
                <tbody>
                    {memos.map((memo) => (
                        <tr key={memo.id}> {/* Assuming each memo has a unique 'id' */}
                            <td className="memo-name">{memo.name}</td>
                            <td className="memo-timestamp">{new Date(memo.timestamp * 1000).toLocaleString()}</td>
                            <td className="memo-message">{memo.message}</td>
                            <td className="memo-from">{memo.from}</td>
                        </tr>
                    ))}
                </tbody>
            </table>
        </div>
    );
};

export default Memos;
```

Figure 4.2.2: Memos.jsx

1.Component Name:

This React component is named Memos, and its primary responsibility is to manage and display a list of confessions obtained from an Ethereum smart contract.

2.Hooks Usage:

The component utilizes the useState and useEffect hooks to manage the state of memos and fetch the memos from the contract when the component mounts or when the contract changes.

3.Effect on Contract Change:

The useEffect hook is triggered whenever the contract in the state object changes.

If a valid contract exists, the fetchMemos function is invoked, which uses the contract's getMemos function to retrieve the list of confessions.

The obtained memos are then set as the state using the setMemos function.

4.JSX Rendering:

In the JSX, the component renders a container with a title ("Confessed List") and a table for displaying individual memos.

The table is dynamically populated by mapping over the memos array and creating a table row (<tr>) for each confession.

Each row includes columns (<td>) for the memo's name, timestamp (converted from Unix timestamp to a readable format), message, and sender ("from").

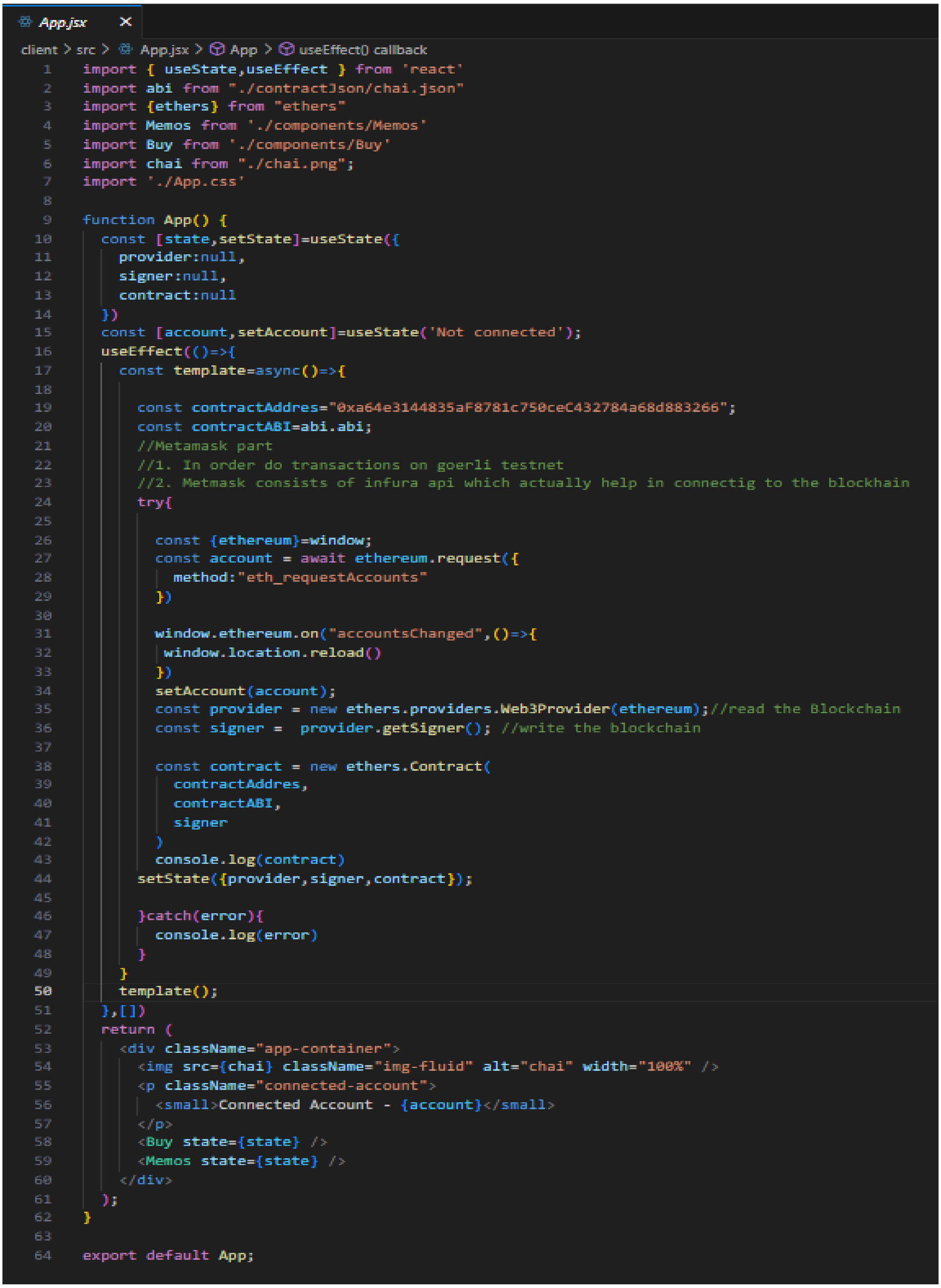
Each memo is uniquely identified by its id.

5.External Styling:

The styling of the component is managed by an external CSS file named "Memos.css."

6.Responsive UI:

Overall, this component is designed to dynamically display a list of confessions obtained from an Ethereum smart contract, providing a responsive and up-to-date user interface.



```jsx
import { useState,useEffect } from 'react'
import abi from "./contractJson/chai.json"
import {ethers} from "ethers"
import Memos from './components/Memos'
import Buy from './components/Buy'
import chai from "./chai.png";
import './App.css'

function App() {
  const [state,setState]=useState({
    provider:null,
    signer:null,
    contract:null
  })
  const [account,setAccount]=useState('Not connected');
  useEffect(()=>{
    const template=async()=>{

      const contractAddres="0xa64e3144835aF8781c750ceC432784a68d883266";
      const contractABI=abi.abi;
      //Metamask part
      //1. In order do transactions on goerli testnet
      //2. Metmask consists of infura api which actually help in connectig to the blockhain
      try{

        const {ethereum}=window;
        const account = await ethereum.request({
          method:"eth_requestAccounts"
        })

        window.ethereum.on("accountsChanged",()=>{
         window.location.reload()
        })
        setAccount(account);
        const provider = new ethers.providers.Web3Provider(ethereum);//read the Blockchain
        const signer =  provider.getSigner(); //write the blockchain

        const contract = new ethers.Contract(
          contractAddres,
          contractABI,
          signer
        )
        console.log(contract)
      setState({provider,signer,contract});

      }catch(error){
        console.log(error)
      }
    }
    template();
  },[])
  return (
    <div className="app-container">
      <img src={chai} className="img-fluid" alt="chai" width="100%" />
      <p className="connected-account">
        <small>Connected Account - {account}</small>
      </p>
      <Buy state={state} />
      <Memos state={state} />
    </div>
  );
}

export default App;
```

Figure 4.2.3: App.jsx

1.Component Name:

This React component is named App, serving as the main application component.

2.State Management:

The component initializes and manages the state using the useState hook to store instances of the Ethereum provider, signer, and contract, as well as tracking the connected account using the account state.

3.Blockchain Connection:

In the useEffect hook, the application connects to the Ethereum blockchain using the MetaMask browser extension.

It requests the user's Ethereum accounts and updates the account state accordingly.

The ethereum.on method is utilized to listen for changes in connected accounts, triggering a page reload when such changes occur.

4.Ethereum Provider and Signer Instances:

Ethereum provider and signer instances are created using the ethers library within the useEffect hook.

The contract instance is instantiated using the contract's address and ABI.

5.JSX Rendering:

The application's JSX renders an image of "chai," a message indicating the connected account, and two child components: Buy and Memos.

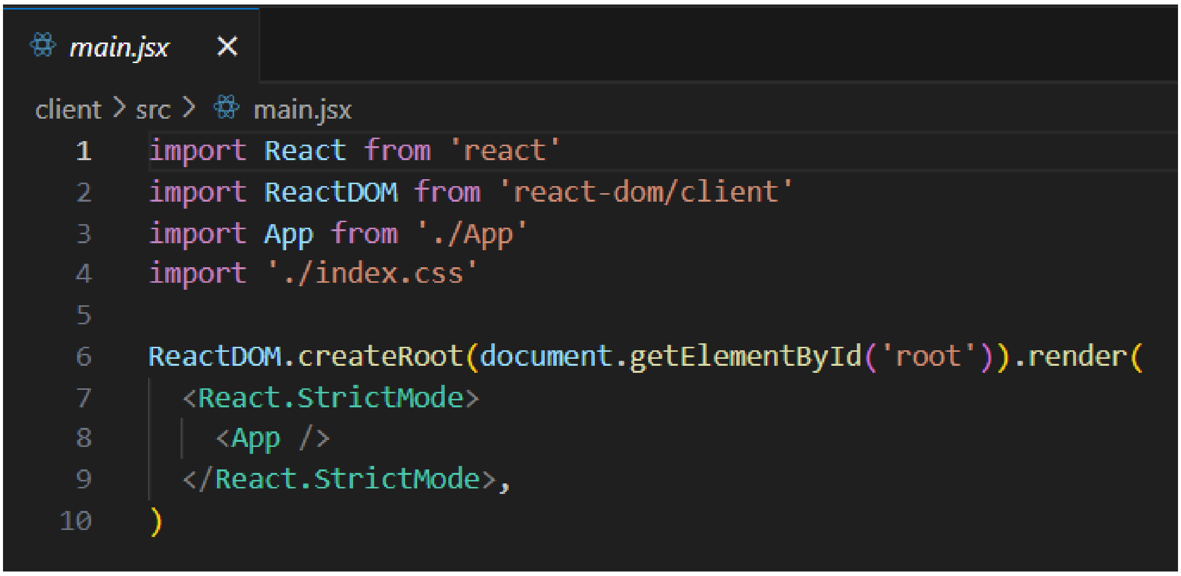
The Buy component allows users to submit confessions, while the Memos component displays a list of existing confessions fetched from the Ethereum smart contract.

6.Application Structure:

Overall, this application is designed to interact with an Ethereum smart contract for submitting and displaying confessions in a decentralized manner.

The UI elements are styled using an external CSS file named "App.css."

The application structure promotes modularity and separation of concerns by using distinct components for specific functionalities.



```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Figure 4.2.4: main.jsx

1.Concurrent Mode Usage:

This code serves as the entry point for a React application, utilizing React's Concurrent Mode.

The createRoot and render methods from ReactDOM are employed to render the App component into the HTML element with the id 'root.'

2.Strict Mode Implementation:

The use of React.StrictMode around the App component facilitates the detection of potential issues during development.

This mode helps identify common pitfalls and deprecated features, promoting best practices and a smoother debugging experience.

3.Global Styles Import:

The global styles defined in 'index.css' are imported into the application.

This ensures consistent styling throughout the entire application, providing a unified visual presentation.

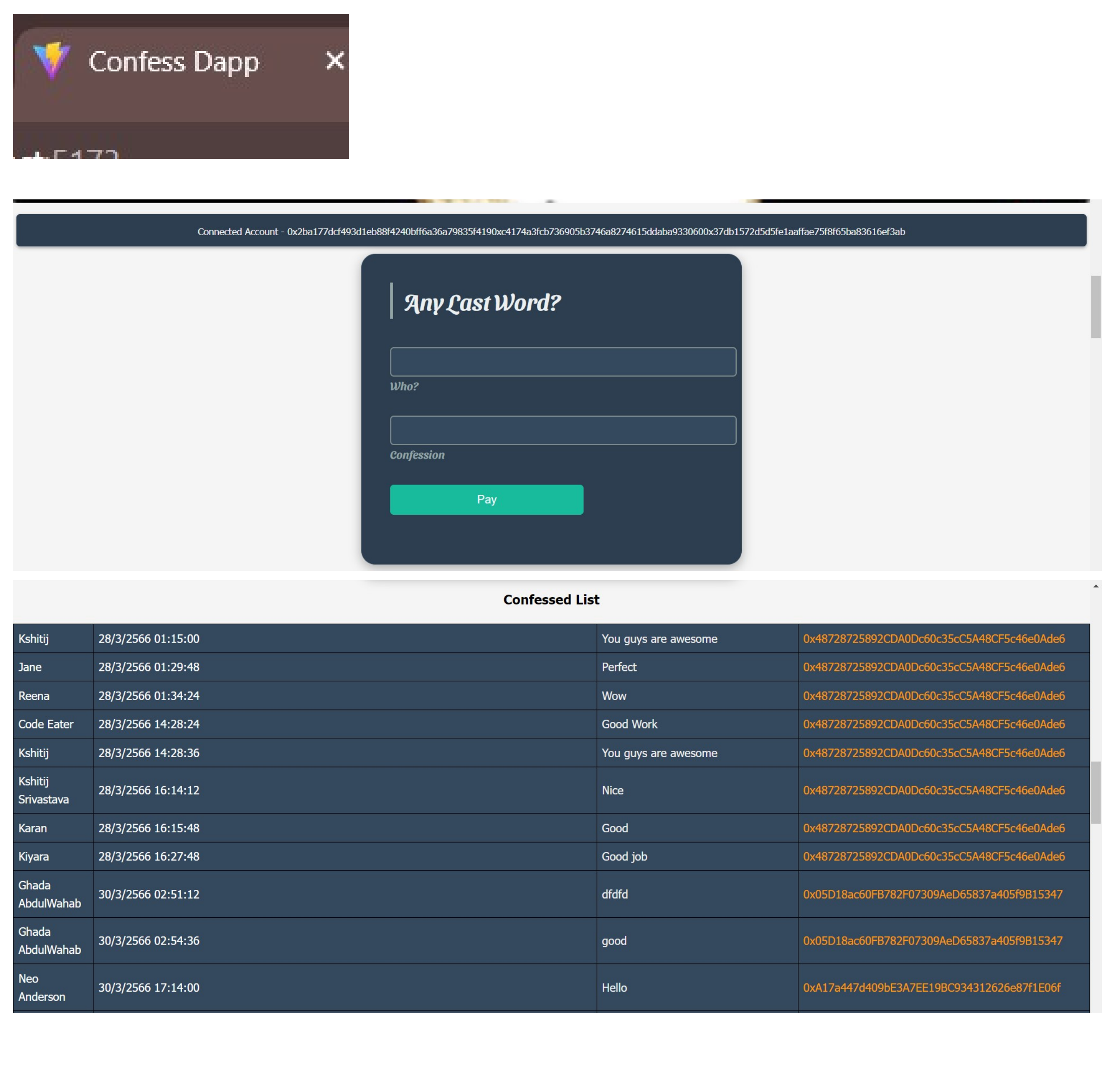
4.Initialization of React App:

Overall, this script initializes the React app, rendering the main App component within a strict development mode.

The use of Concurrent Mode enhances the app's performance by enabling asynchronous rendering, allowing components to work on rendering updates concurrently.

CHAPTER 5 PROTOTYPE

5.1 Prototype



Connected Account - 0x2ba177dcf493d1eb88f4240bff6a36a79835f4190xc4174a3fcb736905b3746a8274615ddaba9330600x37db1572d5d5fe1aaffae75f8f65ba83616ef3ab

Any Last Word?

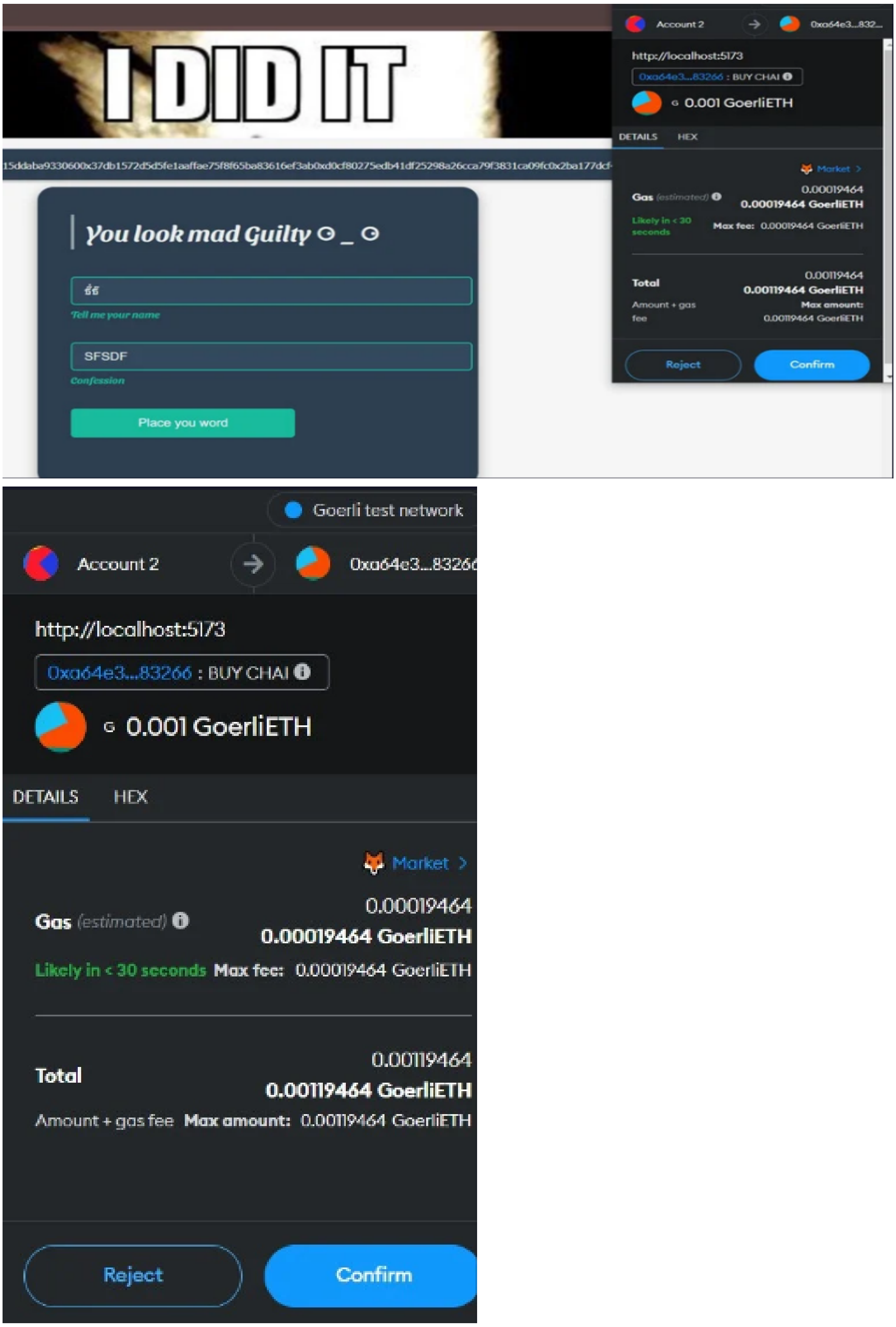
Who?

Confession

Confessed List

Kshitij	28/3/2566 01:15:00	You guys are awesome	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Jane	28/3/2566 01:29:48	Perfect	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Reena	28/3/2566 01:34:24	Wow	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Code Eater	28/3/2566 14:28:24	Good Work	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Kshitij	28/3/2566 14:28:36	You guys are awesome	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Kshitij Srivastava	28/3/2566 16:14:12	Nice	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Karan	28/3/2566 16:15:48	Good	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Kiyara	28/3/2566 16:27:48	Good job	0x48728725892CDA0Dc60c35cC5A48CF5c46e0Ade6
Ghada AbdulWahab	30/3/2566 02:51:12	dfdfd	0x05D18ac60FB782F07309AeD65837a405f9B15347
Ghada AbdulWahab	30/3/2566 02:54:36	good	0x05D18ac60FB782F07309AeD65837a405f9B15347
Neo Anderson	30/3/2566 17:14:00	Hello	0xA17a447d409bE3A7EE19BC934312626e87f1E06f

Figure 5.1.1: prototype pictures



15ddaba9330600x37db1572d5d5fe1aaffae75f8f65ba83616ef3ab0xd0cf80275edb41df25298a26cca79f3831ca09fc0x2ba177dcf

You look mad Guilty ⊙ _ ⊙

Tell me your name: ธ์ธ

Confession: SFSDF

http://localhost:5173

0xa64e3...83266 : BUY CHAI

0.001 GoerliETH

Gas (estimated) 0.00019464 0.00019464 GoerliETH

Likely in < 30 seconds Max fee: 0.00019464 GoerliETH

Total 0.00119464 0.00119464 GoerliETH

Amount + gas fee Max amount: 0.00119464 GoerliETH

http://localhost:5173

0xa64e3...83266 : BUY CHAI

0.001 GoerliETH

Gas (estimated) 0.00019464 0.00019464 GoerliETH

Likely in < 30 seconds Max fee: 0.00019464 GoerliETH

Total 0.00119464 0.00119464 GoerliETH

Amount + gas fee Max amount: 0.00119464 GoerliETH

Figure 5.1.2: prototype pictures

CHAPTER 6 DISCUSSION AND CONCLUSION

6.1 Discussion

6.1.1 Agile Methodology

Within our Agile project management framework, we prioritize effective communication and collaboration
through regular team catch-ups scheduled every Sunday and Monday at 1 pm.

Meeting Agenda:

Project Progress Discussion:

- Review and discuss the achievements and challenges encountered during the past week.
- Analyze the overall progress of the project, addressing any roadblocks or impediments hindering advancement.

Task Allocation for the Upcoming Week:

- Collaboratively determine and allocate tasks for the upcoming week, ensuring alignment with project milestones and goals.
- Evaluate individual team members' workloads and expertise to optimize task distribution for enhanced efficiency.

Skills Matching:

- Assess team members' skill sets and strengths.
- Match team members with tasks based on their proficiency and expertise, aligning project requirements with
individual capabilities for optimal performance.

6.2 Conclusion

The development of the "Confess" DApp represents a significant step forward in the innovative use of blockchain technology for creating a unique platform for personal expression. This project successfully merges the immutable and transparent nature of blockchain with a user-friendly interface, making it accessible and appealing to a broad audience.

At its core, the "Confess" DApp is not just about technology; it's about providing a safe, secure, and permanent space for individuals to share their thoughts and confessions. By leveraging the Ethereum blockchain, the app ensures that every message is preserved forever, unaltered and tamper-proof. This permanence is a key differentiator in an online world dominated by fleeting and often forgettable digital interactions.
Users can now enjoy a smooth and interactive experience because of the integration of React for front-end development and the strong foundation of Solidity smart contracts. The app is not only useful but also visually pleasing and easy to use due to the careful attention to detail in the layout and design.

Understanding the delicate nature of the content users are entrusting to the platform, security and privacy have been of the utmost importance throughout the development process. Users are reassured by the usage of encryption and anonymity technologies that even though their confessions are made public, their identities will stay hidden.

The project also serves as proof of how modern software development techniques, such as test-driven development, agile approaches, and continuous integration and deployment, may be applied successfully. Because of these procedures, the application is now stable, scalable, and adaptable to changing user requirements and technological breakthroughs.

In the future, the "Confess" DApp might develop into a standard for other blockchain apps in addition to becoming a platform for individual expression. Its capacity to use cutting-edge technology for the straightforward but important goal of providing a space for individuals to make a lasting digital impression accounts for its success.

6.3 Further development

1. Consider introducing features that enhance user engagement, such as themed challenges or prompts that encourage users to share specific types of confessions. This can keep the content dynamic and interesting.
2. Implement search and discovery features to allow users to explore confessions based on topics, emotions, or keywords. This can help users find relatable content and foster a sense of connection within the community.
3. Consider expanding the DApp to support multiple languages to cater to a more diverse global audience. This can broaden the reach and cultural relevance of the platform.
4. Consider developing a mobile app version of your confess DApp to make it more accessible to users on the go. Ensure a seamless and user-friendly experience across different devices.