

Contents

I. Introduction	3
1. Blockchain payment ecosystem includes three components:	3
2. In Chapter 3, we have 4 parts.....	3
3. Flow of blockchain payment ecosystem (Frontend → API → Backend).....	4
II. Faux e-commerce web page with ether payment gateway	5
1. The components	5
Root (gateway)	6
public/	6
src/	8
App.js — core của flow thanh toán.....	10
src/Components/	11
src/Items/	11
2. Implementation	12
2.1. Create a React application.....	12
2.2. Edit the package.json file to add the necessary dependencies	13
2.3. Install the dependencies listed in package.json	13
2.4. Inside the src folder, create subfolders named Components and Items, along with the corresponding code files.....	13
2.5. Inside the public folder, create subfolders named images and logos to store the respective files	14
2.6. Create the file App.js.....	14
2.6.1 Importing the dependencies	14
2.6.2 Constructor and State Initialization.....	15
2.6.3 newPayment().....	15
2.6.4 PaymentWait().....	16
2.6.5 MMaskTransfer()	17
2.6.6 tick()	18
2.6.7 bCheck()	19
2.6.8 startTimer().....	20
2.6.9 componentDidMount().....	20
2.6.7 render()	21
II. Merchant HD wallet generator	22
1. The components	22

Root (hdwallet)	22
2. Implementation	23
2.1 Create a new Node.js project called hdwallet	23
2.2 Create your package.json file	23
2.3. Cài các dependencies trong package.json	24
2.4. Tạo file App.js	24
2.5 Run API backend.....	26
III. Merchant wallet interface	27
1. The Component.....	27
Root (MWallet)	28
2. Implementation	28
2.1 Create a React application.....	28
2.2. Edit the package.json file to add the necessary dependencies	28
2.3. Install the dependencies listed in package.json.....	29
2.4. Create Mnemonic.js file	29
2.5. Inside the src/Components folder.....	29
2.6. Create the file App.js.....	35
2.7 Bring merchant wallet online	38
IV. Running the payment ecosystem	38

Designing a Payment Gateway for Online Merchants

- Defining our blockchain payment ecosystem
- Generating dynamic merchant addresses using HD wallets
- Creating an e-commerce website and payment gateway
- Creating an API for generating dynamic payment addresses
- Building the merchant HD wallet
- Running the payment ecosystem

I. Introduction

1. Blockchain payment ecosystem includes three components:

1. A faux e-commerce web page with an ether payment gateway

A mock/demo e-commerce site, used to demonstrate/test the purchase and checkout process

2. A merchant HD wallet generator

generate new address for each payment request

3. A merchant wallet interface

track payments and confirmations on the blockchain

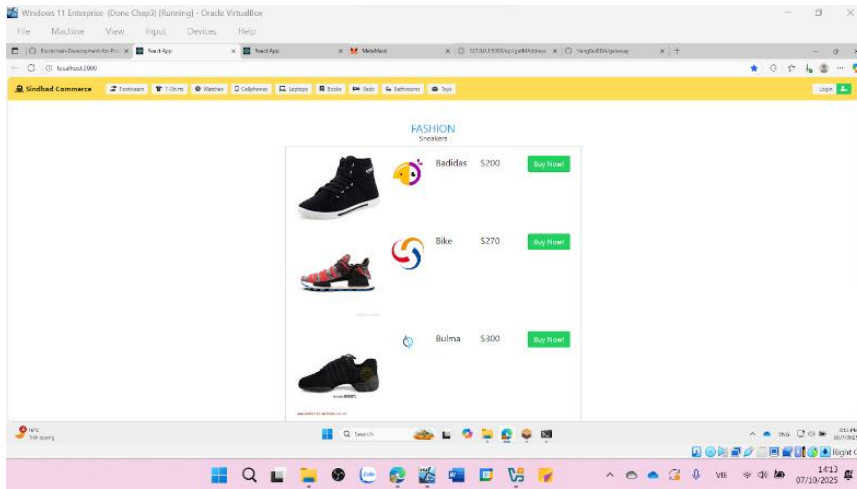
2. In Chapter 3, we have 4 parts

- **Gateway (faux e-commerce) — Frontend**
 - A simulated shop page: display products, checkout, call MetaMask/ethers.js for buyers to deposit money.
 - Call backend API to get dynamic address when user clicks **Buy**
 - Normally run `npm start` → `http://localhost:3000`.
- **Merchant HD-wallet generator — API (Backend)**
 - HTTP service (e.g. Node.js + Express) is responsible for deriving the sub-address from the mnemonic according to BIP44 and returning MAddress.
 - Do not expose private keys; store the order→address mapping; possibly at `http://localhost:5000/api/getMAddress`.
 - Belongs to the **backend**.
- **Merchant wallet interface (dashboard) — Frontend + Backend (Watcher)**
 - **Frontend**: Dashboard displays list of addresses, balance, transactions, status (Confirmed/Unconfirmed)
 - **Backend/Watcher**: Background process connects blockchain node to detect tx, count confirmations, update DB, expose API to dashboard.
 - Dashboard can run on another port (eg 3001)
- **Ganache — Local Ethereum node (dev/test only)**
 - Not an application backend; a mock **blockchain node** for testing (RPC <http://127.0.0.1:7545>).
 - MetaMask and backend/watchers connect to Ganache in development environment.

3. Flow of blockchain payment ecosystem (Frontend → API → Backend)

1. **Frontend: Người dùng bấm Buy Now**
 - UI gọi API: POST/allocate với thông tin cơ bản (ví dụ productId, amount).
 - Frontend chờ và hiển thị trạng thái chờ cấp địa chỉ.
2. **API: Cấp địa chỉ tạm thời**
 - POST/allocate do backend xử lý: backend **tạo orderId**, tính nextIndex, derive địa chỉ từ HD wallet theo đường dẫn m/44'/60'/0'/0/{index}, lưu mapping và metadata (orderId, index, address, amount, expiresAt, status='allocated').
 - API trả về: { orderId, address, amount, expiresAt }. Frontend lưu orderId và hiển thị address + QR + hướng dẫn gửi tiền.
3. **Frontend: Người dùng gửi TX**
 - Người dùng ký và gửi giao dịch (to = address, value = amount) bằng ví (MetaMask hoặc quét QR).
 - **Trong dev:** giao dịch được broadcast tới **Ganache** (http://127.0.0.1:7545).
4. **Backend/Worker: Phát hiện giao dịch**
 - Worker (kết nối cùng RPC của Ganache trong dev) scan block/txs hoặc nhận txHash từ frontend (POST /notifyTx).
 - Khi tìm thấy tx tới address, ghi txHash, from, value, cập nhật status → received_pending.
5. **Backend/Worker: Chờ confirmations**
 - Worker theo dõi confirmations: `confirmations = currentBlock - tx.blockNumber + 1`.
 - Khi `confirmations >= CONFIRM_THRESHOLD` (production ví dụ 12; test trên Ganache có thể dùng 1), cập nhật status → confirmed.
6. **Fulfillment & hậu xử lý**
 - Khi confirmed: cập nhật order (paid), notify merchant/user, bắt đầu fulfillment (ship) và/hoặc sweep tiền vào ví chính (server-side).
 - Gửi event/notification tới frontend (WebSocket/SSE) hoặc cho frontend poll GET /allocation/:orderId để cập nhật trạng thái.
7. **Edge-cases & vận hành**
 - underpaid/overpaid xử lý theo policy; expired khi quá expiresAt.
 - Không lộ mnemonic/privateKey cho client; lưu mapping vào DB/file bền; cập nhật lastIndex phải atomic để tránh cấp trùng.
 - Trong dev Ganache đóng vai trò là node RPC: cả frontend (MetaMask) và backend/worker kết nối tới Ganache để gửi, mine, detect và confirm txs.

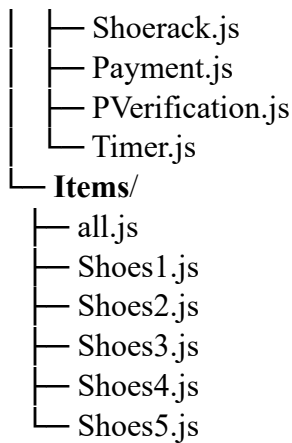
II. Faux e-commerce web page with ether payment gateway



1. The components

gateway/

- package.json
- package-lock.json
- README.md
- **public/**
 - index.html
 - favicon.ico
 - manifest.json
 - **logos/**
 - Badidas.png
 - Bike.jpg
 - ... (brand logos)
 - **images/**
 - Shoe1.jpg
 - Shoe2.jpg
 - ... (product images)
- **src/**
 - index.js
 - index.css
 - App.js
 - App.css
 - App.test.js
 - logo.svg
 - serviceWorker.js
 - **Components/**
 - Nav.js
 - Description.js
 - Container.js

**Lưu ý:**

- Khi chạy `npm install` thư mục **node_modules** sẽ xuất hiện.
- `package.json` liệt kê các **dependency**. `npm install` tải tất cả các package đó về và lưu trong `node_modules` để project có thể chạy.
- `node_modules` chứa mã nguồn thư viện

Giải thích các thành phần trong cây thư mục ở trên:**Root (gateway)**

- **package.json / package-lock.json**
 package.json: Khai báo tên và phiên bản project, scripts start, dependencies
 package-lock.json: Ghi lại phiên bản chính xác của từng gói thư viện đã cài
- **README.md**
 tổng quan: mục đích, cách cài đặt, hướng dẫn chạy ứng dụng

public/

Tất cả file **tĩnh** được serve nguyên bản (không qua webpack bundling hash).

- **index.html**

<code><!DOCTYPE html></code>	Khai báo tài liệu HTML5, ngôn ngữ chính là tiếng Anh.
<code><html lang="en"></code>	
<code><head></code>	Dùng UTF-8 để hiển thị tốt mọi ký tự (có dấu tiếng Việt).
<code><meta charset="utf-8"></code>	
<code><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"></code>	Cho phép giao diện responsive (tự co giãn trên điện thoại, tablet,...)

<code><meta name="theme-color" content="#000000"></code>	Màu chủ đề khi app được mở kiểu “progressive web app” trên Android
<code><script defer src = "https://use.fontawesome.com/releases/v5.1.0/js/all.js"></script></code>	Nạp thư viện Font Awesome (v5.1.0) để bạn có thể dùng icon như: <code><i class="fas fa-shopping-cart"></i></code> trong React
<code><link rel="manifest" href="%PUBLIC_URL%/manifest.json"></code>	Liên kết tới file manifest.json trong thư mục public/. File đó chứa metadata cho PWA (Progressive Web App), ví dụ tên app, icon, màu nền,...
<code><link rel="shortcut icon" href = "%PUBLIC_URL%/favicon.ico"></code>	Icon hiển thị trên tab trình duyệt (favicon).
<code><title>React App</title></code>	Tiêu đề của trang (hiện trên thanh tab trình duyệt). Bạn có thể đổi thành tên ứng dụng của bạn
<code><body></code> <code><noscript></code> You need to enable JavaScript to run this app. <code></noscript></code> <code><div id="root"></div></code> <code></body></code>	<code><noscript></code> : hiển thị cảnh báo nếu trình duyệt tắt JavaScript. React không thể chạy nếu JS bị tắt. <code><div id="root"></div></code>

Thành phần`<head>``<script>` Font Awesome`<link>` manifest / favicon`<body>` + `<div id="root">``<noscript>`**Mục đích**

Cấu hình metadata, icon, manifest, script ngoài

Cho phép dùng icon đẹp

Hỗ trợ hiển thị và PWA

Nơi React sẽ render toàn bộ app

Cảnh báo khi JS bị tắt

- **favicon.ico:** Biểu tượng nhỏ xuất hiện trên tab trình duyệt

- **manifest.json:** Mô tả Progressive Web App – giúp trình duyệt hiểu giao diện khi cài trên điện thoại
- **public/images/:** ảnh sản phẩm miêu tả trên giao diện
- **public/logos/:** logo sản phẩm miêu tả trên giao diện

src/

Mã nguồn React, được build qua Webpack.

- **index.js:** Điểm khởi đầu của React app. File này render `<App />` vào index.html

import React from 'react';	
import ReactDOM from 'react-dom';	
import './index.css';	
import App from './App';	
import '../node_modules/bulma-start/css/main.css'	
import * as serviceWorker from './serviceWorker';	
ReactDOM.render(<App />, document.getElementById('root'));	
serviceWorker.unregister();	

- **index.css**

<pre>body { margin: 0; font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen', 'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue', sans-serif; -webkit-font-smoothing: antialiased; -moz-osx-font-smoothing: grayscale; }</pre>	File CSS gốc áp dụng cho toàn bộ ứng dụng
--	---

<pre>code { font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace; }</pre>	
--	--

- **App.css**

<pre>img.meta-trademark { width: 20%; } #item-lists { height: 300px; overflow-y: scroll; } #item-lists div.shoe:nth-child(even) { background: #f5f5f5; } #item-lists div.shoe { cursor: pointer; } .sortby { font-weight: 300; cursor: pointer; } .item-logo {</pre>	<p>Giao diện và định dạng cho các thành phần trong App.js</p>
---	---

<pre> width: 10px; height: 10px; } .item-image { width: 100px; height: 100px; } .download-metamask { height: 30%; cursor: pointer; width: 70%; margin: auto; } .is-ellipsis { overflow: hidden; text-overflow: ellipsis; } </pre>	
---	--

- **logo.svg:** Logo mặc định của React

App.js — core của flow thanh toán

App.js trong chapter 3 chứa state + các method quản lý luồng thanh toán. Những chức năng chính trong file này:

import	Importing the dependencies and the app components
class	The class is used to define a React component as a class, containing all the state and methods that control the payment logic. It serves as a blueprint that tells React how to initialize, manage data, and render the payment application's interface
newPayment()	The newPayment() method is called every time a user clicks the Buy Now button next to an item.

PaymentWait()	This method is called to set the state variable every time the customer selects Other Wallet for payment, as shown here. It indicates the container component that a payment page needs to be rendered with a 15-minute window for accepting payments to the merchant's address
MMaskTransfer()	This method is called for initiating transfers through the MetaMask wallet of the customer directly to the merchant address
startTimer()	The startTimer() method is used to set a timer for 15 minutes in the payment window as shown here. The customer has 15 minutes to make the payment from their wallet to the merchant's wallet address before the request expires
tick()	The tick() method sets a timer that runs for 15 minutes
bCheck()	The bCheck() method runs a balance check on the merchant's address 10-second intervals
componentDidMount()	The componentDidMount method maps the Shoes constant to our state variables, fetch the individual shoe details from the Shoes.js file that declared earlier
render()	The render() method for App.js renders the Nav, Description, and Container components. It also passes all of the state and methods to the Container component for use by the child components

src/Components/

Các component con chia trách nhiệm UI

Nav.js	This renders a navigation bar for the e-commerce page
Description.js	This renders a single-line description for the page
Container.js	The Container component holds several other components, toggles components, display as per state changes, and passes down their props to the components after it receives them from app.js
Payment.js	Payment.js renders a payment page. It dynamically fetches a new Ethereum address for payment using an API from the merchant wallet. It also sets the number of ether to be transferred for a successful payment. The user can pay using the MetaMask wallet or any other wallet
PVerification.js	This component is rendered when the user selects Other Wallets. It renders a tracking page that gives the user a 15-minute window to transfer to the merchant's address
Shoerack.js	The Shoerack.js app component lists all of the shoes. It renders the e-commerce page for viewing and buying the shoes. On clicking the “Buy Now” button, the customer is redirected to the payment page
Timer.js	The Timer component renders a 15 minutes timer and checks whether the complete amount has been transferred to the merchant's wallet at an interval of 10 seconds

src/Items/

Dữ liệu sản phẩm (static dataset dùng trong lab).

- **all.js**

<pre>import Shoes1 from './Shoes1'; import Shoes2 from './Shoes2'; import Shoes3 from './Shoes3'; import Shoes4 from './Shoes4'; import Shoes5 from './Shoes5';</pre>	Dòng này import (nhập) các module (hoặc 5 file JS) riêng biệt: Shoes1.js, Shoes2.js, ..., Shoes5.js
<pre>const Shoes = [Shoes1, Shoes2, Shoes3, Shoes4, Shoes5,];</pre>	Tạo ra một mảng (array) tên là Shoes, chứa 5 phần tử là các đối tượng vừa import ở trên chứa tất cả thông tin của từng sản phẩm riêng lẻ lại thành một danh sách tổng hợp.

- **Shoes1.js ... Shoes5.js**

Shoes1.js (Shoes2.js, Shoes3.js, Shoes4.js, Shoes5.js tương tự)

<pre>export default { price: 200, name: "Badidas", logo: "Badidas.png", image: "Shoe1.jpg" }</pre>	<p>Xuất (export) mảng Shoes ra ngoài để các file khác có thể dùng</p> <p>Mỗi file ShoesX.js mô tả thông tin chi tiết của một sản phẩm (tên, giá, hình ảnh,...)</p>
--	--

2. Implementation

2.1. Create a React application

- Tạo một React app có tên là “gateway” bằng câu lệnh sau:

`npx create-react-app gateway`

2.2. Edit the package.json file to add the necessary dependencies

- Chỉnh file `package.json` để bổ khai các dependencies cần thiết

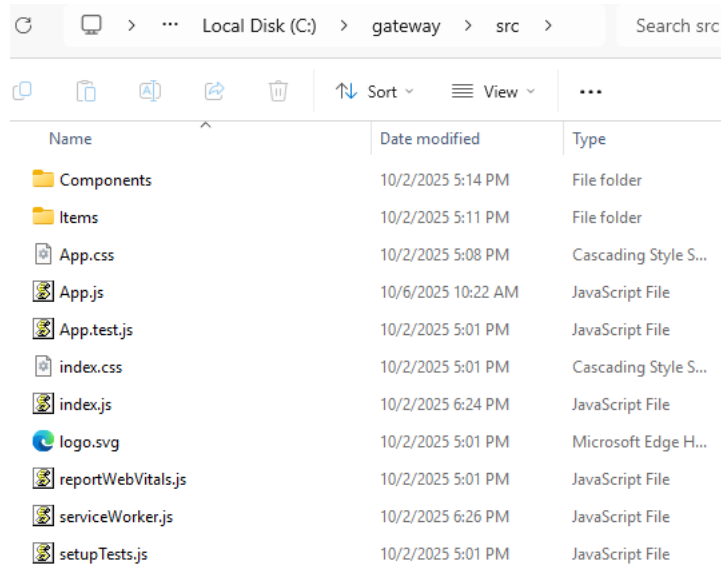
<pre>{ "name": "gateway", "version": "0.1.0", "private": true, "dependencies": { "bulma-start": "0.0.2", "react": "^16.4.1", "react-dom": "^16.4.1", "react-icons": "^5.5.0", "react-scripts": "1.1.4", "web3": "^1.2.0" }, "scripts": { "start": "react-scripts start", "build": "react-scripts build", "test": "react-scripts test --env=jsdom", "eject": "react-scripts eject" } }</pre>	<pre>- react, react-dom, react-scripts: The core libraries required to run a React application. - web3: A library used to connect and interact with the Ethereum or Ganache blockchain network. - bulma-start: A CSS framework that helps build clean, responsive, and modern user interfaces quickly. - react-icons: Provides a collection of icons used in the user interface. - scripts: • start: Runs the application in development mode. • build: Creates an optimized production build of the app. • test: Runs unit tests for the application. • eject: Exposes the hidden configuration for deeper customization.</pre>
---	--

2.3. Install the dependencies listed in package.json

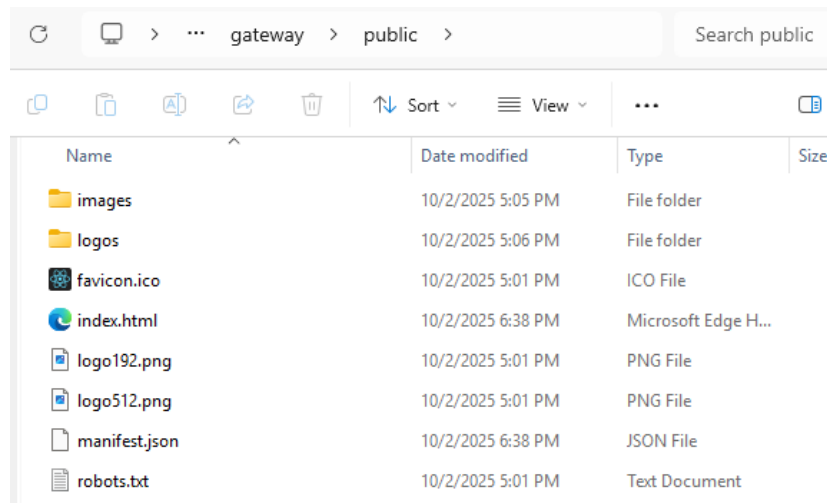
Run the “`npm install`” command to install the dependencies.

- When running `npm install`, the `node_modules` folder will be created automatically.
- The `package.json` file lists all the project dependencies. The `npm install` command downloads these packages and saves them in the `node_modules` directory so the project can run properly.
- The `node_modules` folder contains the source code of the installed libraries and frameworks used by the project.

2.4. Inside the src folder, create subfolders named Components and Items, along with the corresponding code files



2.5. Inside the public folder, create subfolders named images and logos to store the respective files



2.6. Create the file App.js

The `App.js` file will define all the methods invoked by the components and set the initial state of the App

2.6.1 Importing the dependencies

```
import React, { Component } from 'react';
import Web3 from 'web3'
import Nav from './Components/Nav'
import Description from './Components/Description'
import Container from './Components/Container'
import Shoes from './Items/all'
```

2.6.2 Constructor and State Initialization

Within the constructor, we initialize the global parameters of the requisite state variables. We also bind the methods that will be accessed by the child components for changing the state.

```
class App extends Component {
  constructor(){
    super();
    this.appName = 'Sindbad Commerce';
    this.shoes = Shoes;
    this.newPayment = this.newPayment.bind(this);
    this.closePayment = this.closePayment.bind(this);
    this.PaymentWait = this.PaymentWait.bind(this);
    this.tick = this.tick.bind(this);
    this.bCheck = this.bCheck.bind(this);
    this.startTimer = this.startTimer.bind(this);

    this.state = {
      shoes: [],
      PaymentDetail: {},
      Conv: 300,
      defaultGasPrice: null,
      defaultGasLimit: 200000,
      paymentf: false,
      mAddress: '0x',
      amount: 0,
      diff: 0,
      seconds: '00', // responsible for the seconds
      minutes: '15', // responsible for the minutes
      tflag: true
    };
  }
}
```

2.6.3 newPayment()

The `newPayment()` method is called every time a user clicks the **Buy Now** button next to an item. The method takes `index` as a parameter, which basically refers to the shoe model. The first asynchronous call is to our local API, `getMAddress`, which dynamically generates a new Merchant address for receiving payments for each payment request. We'll talk more on this API later. This newly generated address is then updated to the state along with the details of the item for which the payment is being processed.

`CryptoCompare` (min-api.cryptocompare.com/data) is a free service that returns the live conversion rate from USD to ETH. We store this conversion rate in the state variable, `Conv`. Since the e-commerce website displays all prices in USD, the `Payment` app component first

converts the price of the product into ether and then notifies the customer of the number of ether they need to send to the merchant's wallet address.

```

newPayment = (index) => {
    var mAddress;
    let app = this;
    (async function main(){
        await fetch('http://127.0.0.1:5000/api/getMAddress')
        .then(response => response.json())
        .then(data => {
            mAddress = data.MAddress;
            console.log(mAddress);
            app.setState({
                PaymentDetail: app.state.shoes[index],
                mAddress
            })
        }));

        var Conv;
        await fetch('https://min-api.cryptocompare.com/data/price?fsym=ETH&tsyms=USD')
        .then(response => response.json())
        .then(data => {
            Conv=data.USD;

            app.setState({
                Conv
            })
        });

    })();
};

closePayment = () => {
    clearInterval(this.intervalHandle);
    clearInterval(this.intervalBalance);
    this.setState({
        PaymentDetail: {},
        paymentf: false,
        mAddress: '0x',
        amount: 0,
        diff: 0,
        seconds: '00', // responsible for the seconds
        minutes: '15', // responsible for the minutes
        tflag: true,
        defaultGasPrice: null,
        defaultGasLimit: 200000
    })
};

```

2.6.4 PaymentWait()

This method is called to set the state variable every time the customer selects **Other Wallet** for payment, as shown here. It indicates the container component that a payment page needs to be rendered with a 15-minute window for accepting payments to the merchant's address. It sets the `paymentf` flag to `true`, indicating that a payment is being made by a wallet other than MetaMask and sets the state variable for the amount to be paid in ETH and the merchant address (`mAddress`) to which the transfer has to be made.

```
PaymentWait = (mAddress,amount) => {
    this.setState({
        paymentf: true,
        amount,
        mAddress
    })
};
resetApp = () => {
    this.setState({
        PaymentDetail: {},
        paymentf: false,
        mAddress: '0x',
        amount: 0,
        diff: 0,
        seconds: '00', // responsible for the seconds
        minutes: '15', // responsible for the minutes
        tflag: true,
        defaultGasPrice: null,
        defaultGasLimit: 200000
    })
};
setGasPrice = (web3) => {
    web3.eth.getGasPrice((err,price) => {if(!err)
    {
        console.log(price);
        price = web3.utils.fromWei(price.toString(),'gwei');
        this.setState({defaultGasPrice: price})
    }
    else
    {
        console.log(err);
    }
    });
};
```

2.6.5 MMaskTransfer()

This method is called for initiating transfers through the MetaMask wallet of the customer directly to the merchant address.

- The method takes the merchant address and amount to be paid as input parameters.
- On being called, it first checks whether the `window.ethereum` object is present, essentially checking whether the browser has injected `web3` through MetaMask.
- If an injected `web3` is present, it maps the app's `web3` object to the injected `web3`.
- Next, it makes an asynchronous call to request permission to access the array of accounts available on the user's MetaMask wallet. This is done through `Ethereum.enable`.

```
MMaskTransfer = (MAddress,amount) => {
  let app = this;
  if (window.ethereum) {
    const ethereum = window.ethereum;
    let web3 = new Web3(ethereum);
    ethereum.enable().then((accounts) => {
      let account = accounts[0];
      web3.eth.defaultAccount = account ;
      this.setGasPrice(web3);
      let tAmount = amount * 1000000000000000000;
      let transObj = {to: MAddress, gas: this.state.defaultGasLimit,gasPrice:
this.state.defaultGasPrice, value: tAmount}
      web3.eth.sendTransaction(transObj,function (error, result){
        if(!error){
          console.log(result);
          app.resetApp();
        } else{
          console.log(error);
        }
      });
    });
  }
}
```

2.6.6 tick()

Every interval, the `tick()` method sets the minutes and seconds state variable. It also fixes the formatting as per the `00:00` format when the seconds or minutes are single digits.

When both minutes and seconds are equal to zero, it clears the timer interval and the balance check interval. This happens when the payment request expires. At the end of each interval, it decreases the `secondsRemaining` parameter by 1.

```
tick(){
  var min = Math.floor(this.secondsRemaining / 60);
  var sec = this.secondsRemaining - (min * 60);
  this.setState({
    minutes: min,
    seconds: sec
  })
}
```

```

if (sec < 10) {
  this.setState({
    seconds: "0" + this.state.seconds,
  })
}
if (min < 10) {
  this.setState({
    minutes: "0" + this.state.minutes,
  })
}
if (min === 0 & sec === 0) {
  clearInterval(this.intervalHandle);
  clearInterval(this.intervalBalance);
}
this.secondsRemaining--;
}

```

2.6.7 bCheck()

- The web3.getBalance() method checks the balance of mAddress, the merchant address, every 10 seconds. This balance is mapped to the diff variable.
- If the diff variable is greater than or equal to amount, bCheck() immediately clears the timer and balance check intervals and stops the timer.
- The diff variable is also updated to the state. It is used by the PVerificationcomponent in the payment window to display to the customer how much of the payment has been received by the merchant's wallet address.

```

bCheck(){
let app = this;
let amount = this.state.amount;
let intervalHandle = this.intervalHandle;
let intervalBalance = this.intervalBalance;
this.web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));
this.web3.eth.getBalance(this.state.mAddress,function (error, result){
  if(!error){
    let diff = result / 1000000000000000000;
    if(diff >= amount )
    {
      clearInterval(intervalHandle);
      clearInterval(intervalBalance);
    }
    app.setState ({
      diff
    })
  }
  else
  {

```

```

        console.log(error);
    }

    });
}

```

2.6.8 startTimer()

- `tflag` is set to `false` if the app has an ongoing 15-minute window. This prevents React from creating multiple timer intervals when a state variable is updated.
- If `tflag` is set to `true`, meaning no interval exists, the method sets up two intervals. The first interval calls the method `tick` at an interval of 15 minutes.
- The `tick()` method is the brains behind our timer. The second interval calls the `bCheck()` method at an interval of 10 seconds.
- The `bCheck()` method checks the balance of the merchant's wallet address within the 15-minute window.
- If the address receives an amount equal to or greater than the amount owed by the customer, it stops the timer and notifies the customer of the successful payment.
- This method also initializes the `secondsRemaining` parameter to 600 seconds or 15 minutes. This will be used by the `tick()` method.

```

startTimer = () => {
  if(this.state.tflag === true)
  {
    this.intervalHandle = setInterval(this.tick,1000);
    this.intervalBalance = setInterval(this.bCheck,10000);
    let time = this.state.minutes;
    this.secondsRemaining = time * 60;
    this.setState({
      tflag: false
    });
  }
}

```

2.6.9 componentDidMount()

```

componentDidMount() {
  let app = this;
  let shoes = app.state.shoes;
  Shoes.forEach((shoe) => {
    let logo = shoe.logo;
    let price = shoe.price;
    let image = shoe.image;
    let name = shoe.name;
    shoes.push({
      logo,

```

```

        price,
        name,
        image,
    });
    app.setState({
      shoes
    })
  });
}

```

2.6.7 render()

The `render()` method for `App.js` renders the `Nav`, `Description`, and `Container` components. It also passes all of the state and methods to the `Container` component for use by the child components. When the `Container` component is rendered, it will accept all props and forward them to the child components as and when they are rendered.

```

render() {
  return (
    <div>
      <Nav appName={this.appName} />
      <div>
        <Description />
        <Container
          shoes={this.state.shoes}
          newPayment={this.newPayment}
          closePayment={this.closePayment}
          PaymentDetail={this.state.PaymentDetail}
          mAddress={this.state.mAddress}
          amount={this.state.amount}
          diff={this.state.diff}
          paymentf={this.state.paymentf}
          Conv={this.state.Conv}
          MMaskTransfer={this.MMaskTransfer}
          PaymentWait={this.PaymentWait}
          startTimer={this.startTimer}
          tick={this.tick}

          privateToAddress={this.privateToAddress}
          getRandomWallet={this.getRandomWallet}
          defaultGasPrice={this.state.defaultGasPrice}
          defaultGasLimit={this.state.defaultGasLimit}
          minutes={this.state.minutes}
          seconds={this.state.seconds} />
        </div>
      </div>
    </div>
  )
}

```

```
}
export default App;
```

II. Merchant HD wallet generator

HD wallet (Node.js project) for the merchant who is accepting payment for their products from the e-commerce gateway

HD wallets allow us to create a set of hierarchical wallet addresses derived from the same mnemonic. If the merchant can safely preserve one mnemonic phrase, they can manage all of the addresses using the same string of words. For providing dynamically generated, hierarchically linked addresses to our payment gateway, we'll be setting up a Node.js app with a get API service.

- A wallet contains a private key (k), from which a public key (K) is derived. From the public key K, an address is generated.
- An HD Wallet (Hierarchical Deterministic Wallet) generates addresses from a single seed.
- Each time a purchase is made, the HD wallet creates a new unique address, all derived from the same seed.

1. The components

```
hdwallet\
├── README.md
├── node_modules
├── .state.json
├── MAddress.js
├── app.js
├── package.json
├── package-lock.json
├── .gitignore
└── .mnemonic
```

Giải thích các thành phần trong cây thư mục ở trên:

Root (hdwallet)

- package.json / package-lock.json
package.json: Khai báo tên và phiên bản project, scripts start, dependencies
package-lock.json: Ghi lại phiên bản chính xác của từng gói thư viện đã cài
- README.md: tổng quan mục đích, cách cài đặt, hướng dẫn chạy ứng dụng
- .state.json
- MAddress.js
- app.js
- .mnemonic: Store the mnemonic phrase (seed phrase) — a sequence of 12 or 24 words that serves as the master key used to generate all derived wallets (child addresses). From that mnemonic, use hdkey to derive the wallet tree according to the BIP-44 standard — each child wallet uses a different index in the path: m/44'/60'/0'/0/i.

2. Implementation

2.1 Create a new Node.js project called hdwallet

Run the `mkdir hdwallet` command.

2.2 Create your package.json file

Run `npm init` command to create `package.json` file. Updating `package.json` to the following values

```
{
  "name": "hdwallet",
  "version": "1.0.0",
  "description": "hdwallet",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bip39": "^3.1.0",
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "ethereumjs-tx": "^2.1.2",
    "ethereumjs-util": "^6.2.1",
    "express": "^4.21.2",
    "hdkey": "^1.1.2",
    "web3": "^1.2.0"
```

```
}
}
```

2.3. Cài các dependencies trong package.json

Run the “`npm install`” command to install the dependencies.

- When running `npm install`, the `node_modules` folder will be created automatically.
- The `package.json` file lists all the project dependencies. The `npm install` command downloads these packages and saves them in the `node_modules` directory so the project can run properly.
- The `node_modules` folder contains the source code of the installed libraries and frameworks used by the project.

2.4. Tạo file App.js

Declaring all of the dependencies

- The `bip39` module is used to generate a random mnemonic string. This string can be used to recover a wallet or import the wallet to a third-party wallet service provider such as MetaMask. A mnemonic in the `bip39` format looks somewhat like `feature concert truth service energy egg able bind comfort candy harvest similar`.

```
const bip39 = require('bip39');
const hdkey = require('hdkey');
const ethUtil = require('ethereumjs-util');
const ethTx = require('ethereumjs-tx');
const express = require('express');
const Web3 = require('web3');
```

```
// Kết nối tới Ganache (Blockchain local)

const web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));

// Kiểm tra kết nối
web3.eth.getBlockNumber()
  .then(num => console.log("Đã kết nối Ganache. Block hiện tại:", num))
  .catch(err => console.error("Lỗi kết nối Ganache:", err));

// Khởi tạo ứng dụng Express
const app = express();
```

The `middleware` layer here allows CORS or Cross-Origin Resource Sharing. This is required so our gateway app can request a new merchant address from our API.

```
// Middleware CORS (cho phép frontend truy cập API này)
// =====
app.use(function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
  res.setHeader('Access-Control-Allow-Credentials', true);
  next();
});
```

```
// Cấu hình cổng chạy server

var server = app.listen(process.env.PORT || 5000, function () {
  var port = server.address().port;
  console.log("Server đang chạy trên cổng:", port);
});
```

The `pathid` parameter will keep a track of the last address index used to derive a hierarchical address and increment after generating a new address.

The app checks whether `pathid` is less than 100 before incrementing because the app is designed so that it loops between 100 hierarchical addresses. This is done so that the app only generates 100 new hierarchical addresses and then reuses the existing addresses. Our merchant wallet will track these 100 addresses for payments. If `pathid` is equal to or greater than 100, it resets it to zero instead of incrementing it again.

```
// Biến toàn cục lưu chỉ số path

var pathid = 0;

// API tạo địa chỉ ví mới

app.get("/api/getMAddress", function (req, res) {
  // Giữ nguyên mnemonic này cho đồng bộ với mwallet
  const mnemonic = "dove vague inherit hazard rubber abandon corn crowd awake manual unique cry";
  console.log("Mnemonic:", mnemonic);

  var sendResponseObject = {};
  var address;

  (async function main() {
    try {
      // Tạo seed từ mnemonic
      const seed = await bip39.mnemonicToSeed(mnemonic);
```

```
// Tạo root node từ seed
const root = hdkey.fromMasterSeed(seed);

// Derive theo HD path
var path = "m/44'/60'/0'/0/" + pathid;
const addrNode = root.derive(path);

// thêm "0x" trước khi checksum
const pubKey = ethUtil.privateToPublic(addrNode._privateKey);
const addr = ethUtil.publicToAddress(pubKey).toString('hex');
address = ethUtil.toChecksumAddress("0x" + addr);

console.log(`Address [${pathid}]: ${address}`);

// Kiểm tra balance thực trên blockchain
const balanceWei = await web3.eth.getBalance(address);
const balanceEth = web3.utils.fromWei(balanceWei, 'ether');
console.log(`Balance hiện tại: ${balanceEth} ETH`);

// Gửi địa chỉ và balance về frontend
sendResponseObject['MAddress'] = address;
sendResponseObject['Balance'] = balanceEth;

res.json(sendResponseObject);

// Tăng pathid cho lần sau
pathid = (pathid + 1) % 100;

} catch (error) {
  console.error("Lỗi khi tạo địa chỉ:", error);
  res.status(500).send("Đã xảy ra lỗi khi tạo địa chỉ ví");
}
})0;
});
```

2.5 Run API backend

Start the application using node App.js. The app will print the mnemonic in the console on its first run. This mnemonic is generated only once and all addresses generated will be mapped to it.

```

C:\hdwallet>node app.js
Server đang chạy trên cổng: 5000
Mnemonic: theory swing festival parade vast kiwi save arrest lake organ cute nest
Seed: <Buffer f0 84 3a 65 38 ee 19 64 c1 2a 1f 45 a3 80 68 54 e8 f6 80 7b e2 a2 ad 37 a5 e8 04 3f 23 20 47 35 0b 30 68 8
9 e0 ec 49 f3 5e 8f f9 7c 7e 49 78 04 0d 2d ... 14 more bytes>
Master Private Key: 412c2e483cce81cdd99ab3a443ee342e9a447bbc6cdeaf10233a631c2951463
Master Public Key: 03af87b20caecb4a98e58fc82eccf11963edd2c6b991d4bb1fb9e9791dc184dc23
Path: m/44'/60'/0'/0/0
Public Key: 0888e309206127a29f95cab6a2fcfaa4d506bbe3e43c738ee99f6da23b3c43dc4b46a3d7b04d0a15234681c11d0a5767c07e66a7405b
590a9a442277732148e1
Address (raw): e84caa5bda4e963537c80d585422b32547331130
Address (checksum): 0xE84caa5BdA4e963537c80d585422B32547331130
Mnemonic: sound choose step caution illness film detect scrub tattoo embody allow air
Seed: <Buffer d9 ba f3 12 a7 75 fa cd e1 74 2b ec a0 bf dd b3 ad 93 c0 55 d4 3c bf 25 ea 57 a8 a1 dc 5c 06 ef e6 fd 51 3
1 0a 92 64 15 6f 72 e9 74 3f f7 bb 90 fd 3a ... 14 more bytes>
Master Private Key: 1fc6657dbe47e0e37aa330d0b984cfaa98ed4104b33f8623e52a3d0aa0af47b2
Master Public Key: 03881162485ebef4e61c107f5204a49f62f142e7301609f0f27195d287c7cd8d9f
Path: m/44'/60'/0'/0/1
Public Key: 20cc4187b75ce9ee7f278bcbcb2fc95cf7559a77d1a9bb9a9fd4b7631c407577e16e4636d0a395d26b9f2e9641d4e130e7693f6aa6c8
4776584fc14b7f590d70
Address (raw): 455e7eef547d0d52e0de9cba5f2145bd6e1fe56e
Address (checksum): 0x455E7eEF547D0d52e0de9CBa5F2145bD6e1FE56E
Mnemonic: police pattern avoid right sniff gap buzz knee electric check detail promote
Seed: <Buffer db 31 70 6b 5a d4 81 0e a7 fc 44 6a a2 9d a7 0d 26 78 a5 d4 7c c9 ed 8c 87 9a 83 bc 05 80 9d 4b cd 5a 83 0
1 92 02 2f 5c 50 fa d5 93 73 05 58 1f a6 83 ... 14 more bytes>
Master Private Key: 402e6cfc80a7783342253d4388b03e8a6fa9123a2f3831cf6359ba7abc66e3ce
Master Public Key: 022536676b7531750dc4d56399300f0dde6444cab70cca8845f1c17fde5ee23e69
Path: m/44'/60'/0'/0/2
Public Key: 5ad4507c5acce721c9d37b59725ead6a1961798cb5aa4de1bd8a30cdddb8c9a3aef3a08760aab8cbe117b8f3248fab9c588353c5ad6
b2cd1b786105dde493a8

```

III. Merchant wallet interface

1. The Component

```

MWallet\
├── README.md
├── package.json
├── package-lock.json
├── .gitignore
├── node_modules
├── public\
├── src\
│   ├── index.js
│   ├── index.css
│   ├── logo.svg
│   ├── reportWebVitals.js
│   ├── serviceWorker.js
│   ├── setupTests.js
│   ├── App.css
│   ├── App.js           # Hiển thị danh sách địa chỉ merchant + giao dịch
│   ├── Components\
│   │   ├── Nav.js
│   │   ├── Description.js
│   │   └── Container.js  # Kiểm tra acc có giao dịch ⇒ render WalletTrans

```

```

├── TransHeader.js
├── WalletMain.js    # Hiển thị địa chỉ merchant + balance
├── WalletTrans.js   # Liệt kê giao dịch và số block confirmations
└── Items\
    └── Mnemonic.js  # Chuỗi mnemonic HD Wallet của merchant

```

Giải thích các thành phần trong cây thư mục ở trên:

Root (MWallet)

2. Implementation

2.1 Create a React application

Run `npx create-react-app MWallet` command

2.2. Edit the package.json file to add the necessary dependencies

```

{
  "name": "MerchantWallet",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/dom": "^10.4.1",
    "@testing-library/jest-dom": "^6.9.1",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "assert": "^2.1.0",
    "bip39": "^3.1.0",
    "buffer": "^6.0.3",
    "bulma": "^1.0.4",
    "crypto-browserify": "^3.12.1",
    "ethereumjs-util": "^7.1.5",
    "hdkey": "^2.1.0",
    "process": "^0.11.10",
    "react": "^19.2.0",
    "react-dom": "^19.2.0",
    "react-scripts": "5.0.1",
    "stream-browserify": "^3.0.0",
    "util": "^0.12.5",
    "web-vitals": "^2.1.4",
    "web3": "^4.16.0"
  },
  "scripts": {
    "start": "set PORT=8000 && react-app-rewired start",
    "build": "react-app-rewired build",
    "test": "react-app-rewired test",

```

```
"eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browser": {
  "assert": "assert",
  "buffer": "buffer",
  "crypto": "crypto-browserify",
  "stream": "stream-browserify",
  "process": "process/browser",
  "util": "util"
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
},
"devDependencies": {
  "react-app-rewired": "^2.2.1"
}
}
```

2.3. Install the dependencies listed in package.json

Run the `npm install` command to install the dependencies.

2.4. Create Mnemonic.js file

src/Items

```
const Mnemonic = "dove vague inherit hazard rubber abandon corn crowd awake manual unique cry";
export default Mnemonic;
```

2.5. Inside the src/Components folder

Container.js

```

import React, { Component } from 'react';
import WalletMain from './WalletMain'
import WalletTrans from './WalletTrans'
import TransHeader from './TransHeader'
class Container extends Component {
  render(){

    return (
      <section className="container is-full">
        <div className="columns is-mobile">
          {
            this.props.acc?
              <div className="column is-full">
                <TransHeader/>
                <div className="panel">
                  <WalletTrans
transactions={this.props.transactions}
acc = {this.props.acc}
/>
                </div>
              </div>:
              <div className="column is-8 is-offset-2">
                <div className="panel">
                  <div className="panel-block is-paddingless is-12" >
                    <WalletMain accounts={this.props.accounts}
getAccountTransactions={this.props.getAccountTransactions}/>
                  </div>
                </div>
              </div>
            }
          </div>
        </section>
      )
    )
  }
}
export default Container;

```

WalletMain.js

```

import React from 'react';

class WalletMain extends React.Component {
  render() {

```

```

    return (
      <div className="column" id="item-lists">
        {
          this.props.accounts.map((account,index) => {
            return (
              <div
                className="columns">
                  <div
                    key={index}
                    className="column
is-7 " >
                    {account.address}
                  </div>
                  <div
                    className="column
is-2 ">
                    {account.balance} ETH
                  </div>
                  {
                    <div className="column is-1 ">
                      <a class="button is-
info" onClick={() => this.props.getAccountTransactions(account.address)}>
                        View Transactions
                      </a>
                    </div>
                  }
                </div>
              )
            })
          }
        </div>
      )
    }
  }
}
export default WalletMain;

```

WalletTrans.js

```

import React from 'react';

class WalletTrans extends React.Component {

```

```

render() {

  return (

    <div className="panel-block is-paddingless" >

      <div className="column is-full" id="item-lists">

        {
          this.props.transactions.map((tx,index) => {

            return (

              <div
                className="columns tx">

                  <div
                    className="column
is-small" >

                      {tx.transactionIndex}

                    <div className="column is-small">

                      <span className="tag is-spaced">

                        {tx.hash}

                      </span>

                    <div className="column is-small">

                      {tx.blockNumber}

                    <div className="column is-small">

                      {tx.from}

                    <div className="column is-small">

                      {tx.value} ETH

                    <div className="column is-small">

                      {tx.confirmations}


```

```

is-size-8" >
    {tx.cflag}
  </div>

  </div>

  )
  })
  </div>
  </div>

  )
  }
}

export default WalletTrans;

```

TransHeader.js

```

import React from 'react';

class TransHeader extends React.Component {

  render() {
    return (
      <div className="columns">
        <div className="column is-small" >
          <span className="tag is-info">
            Index
          </span>
        </div>

        <div className="column is-small" >

```

```

        <span className="tag is-info">
            Hash
        </span>
    </div>

    <div className="column is-small is-1" >
        <span className="tag is-info">
            Block No.
        </span>
    </div>

    <div className="column is-small is-offset-1" >
        <span className="tag is-info">
            From
        </span>
    </div>

    <div className="column is-small is-1" >
        <span className="tag is-info">
            Value
        </span>
    </div>

    <div className="column is-small is-1" >
        <span className="tag is-info">
            Blocks
        </span>
    </div>

    <div className="column is-small is-1" >
        <span className="tag is-info">
            Status
        </span>
    </div>
    </div>
    )
    }
    }

export default TransHeader;

```

Nav.js

```

import React, { Component } from 'react';

class Nav extends Component {
  render() {
    return (
      <nav className="navbar is-danger" aria-label="main navigation">

        <a className="navbar-item" href="/">
          <strong><i className="fa fa-wallet"></i> {this.props.appName}</strong>
        </a>
      </nav>
    );
  }
}

```

```

        <a role="button" className="navbar-burger" aria-label="menu" aria-
expanded="false">
          <span aria-hidden="true"></span>

        </a>

      </nav>
    )
  }
}

export default Nav;

```

Description.js

```

import React from 'react';

function Description(props) {
  return (
    <section className="container">
      <div className="has-text-centered content">
        <br/>
        <h1 className="title is-4 is-uppercase has-text-black-light">Merchant Wallet</h1>
        {
          props.acc?
          <div>
            <h2 className="subtitle is-6 has-text-black-light">Transaction
List</h2>
            </div>:
            <div>
              <h2 className="subtitle is-6 has-text-black-light">Address List</h2>
              </div>
            }
          </div>
        </section>
      )
    }
  }

export default Description;

```

2.6. Create the file App.js

```

import React, { Component } from 'react';
import Web3 from 'web3';

```

```

import Container from './Components/Container';
import Nav from './Components/Nav';
import Description from './Components/Description';
import Mnemonic from './Items/Mnemonic.js';
const bip39 = require('bip39');
const hdkey = require('hdkey');
const ethUtil = require('ethereumjs-util');

class App extends Component {
  constructor() {
    super();
    this.appName = 'Merchant Wallet';
    this.getAccountTransactions = this.getAccountTransactions.bind(this);

    this.state = {
      acc: null,
      accounts: [],
      transactions: [],
      mnemonic: Mnemonic,
    };
  }

  // Lấy tất cả giao dịch của 1 account
  getAccountTransactions = (accAddress) => {
    const startBlockNumber = 0;
    let app = this;
    let transactions = [];

    (async function main() {
      try {
        const endBlockNumber = await app.web3.eth.getBlockNumber();
        console.log(
          `Searching transactions to ${accAddress} between blocks ${startBlockNumber}
and ${endBlockNumber}`
        );

        for (let i = endBlockNumber; i >= 0; i--) {
          const block = await app.web3.eth.getBlock(i, true);
          if (block && block.transactions) {
            block.transactions.forEach((tx) => {
              if (accAddress === tx.to) {
                const value = Number(tx.value) / 1e18;
                const confirmations = endBlockNumber - tx.blockNumber;
                const cflag = confirmations > 40 ? 'Confirmed' : 'Unconfirmed';

                transactions.push({
                  transactionIndex: tx.transactionIndex,
                  hash: tx.hash,

```

```

        blockNumber: tx.blockNumber,
        from: tx.from,
        value,
        confirmations,
        cflag,
    });
    }
    });
}

app.setState({ transactions, acc: accAddress });
console.log('Transactions updated:', transactions.length);
} catch (err) {
    console.error('Error fetching transactions:', err);
}
})();
};

// Khi component mount -> lấy danh sách địa chỉ
async componentDidMount() {
    let app = this;
    let accounts = [];
    let pathid = 100;

    try {
        // Kết nối Ganache
        this.web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));
        const netId = await this.web3.eth.net.getId();
        console.log("Web3 connected to network:", netId);

        console.log("Mnemonic:", this.state.mnemonic);

        // Tạo seed và root key
        const seed = await bip39.mnemonicToSeed(this.state.mnemonic);
        const root = hdkey.fromMasterSeed(seed);

        // Duyệt qua 100 địa chỉ
        for (let i = 0; i <= pathid; i++) {
            const path = `m/44'/60'/0'/0/${i}`;
            const addrNode = root.derive(path);
            const pubKey = ethUtil.privateToPublic(addrNode._privateKey);
            const addr = ethUtil.publicToAddress(pubKey).toString('hex');
            const address = ethUtil.toChecksumAddress('0x' + addr);

            try {
                const result = await this.web3.eth.getBalance(address);
                const balance = Number(result) / 1e18;
            }
        }
    }
}

```

```

        if (balance > 0) {
            accounts.push({ address, balance });
            console.log(`Found account: ${address} | Balance: ${balance} ETH`);
        }
    } catch (err) {
        console.error("Error fetching balance:", err);
    }
}

this.setState({ accounts });
console.log(`Total active accounts: ${accounts.length}`);
} catch (err) {
    console.error("Web3 initialization failed:", err);
}
}

// Render giao diện
render() {
    return (
        <div>
            <Nav appName={this.appName} />
            <Description acc={this.state.acc} />
            <Container
                acc={this.state.acc}
                accounts={this.state.accounts}
                transactions={this.state.transactions}
                getAccountTransactions={this.getAccountTransactions}
            />
        </div>
    );
}
}

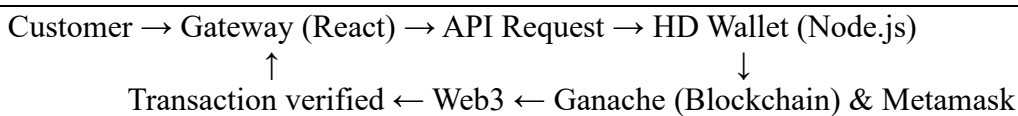
export default App;

```

2.7 Bring merchant wallet online

Run `npm start` command

IV. Running the payment ecosystem

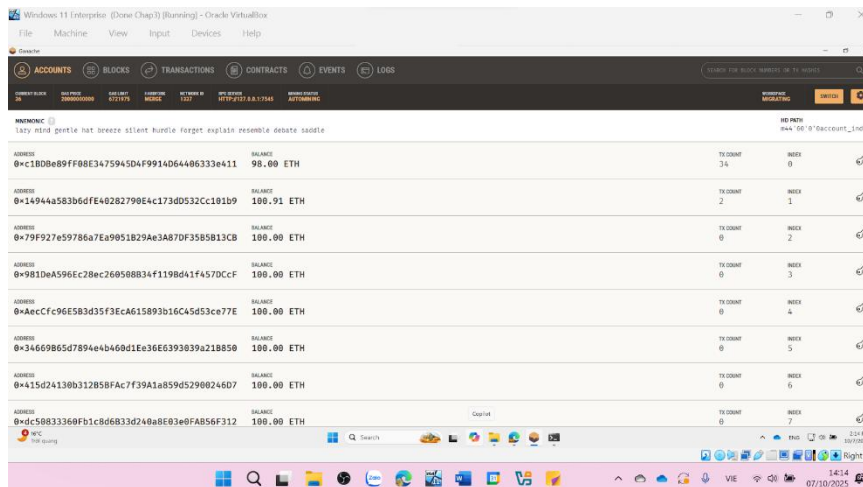


Luồng mua hàng và thanh toán giữa người mua và người bán:

Account address của Gananche được dùng như ví tiền của người mua. Còn address sinh ra từ backend là ví của người bán. Khi mua hàng ta sẽ chuyển tiền từ ví Ganache sang cho ví của người bán

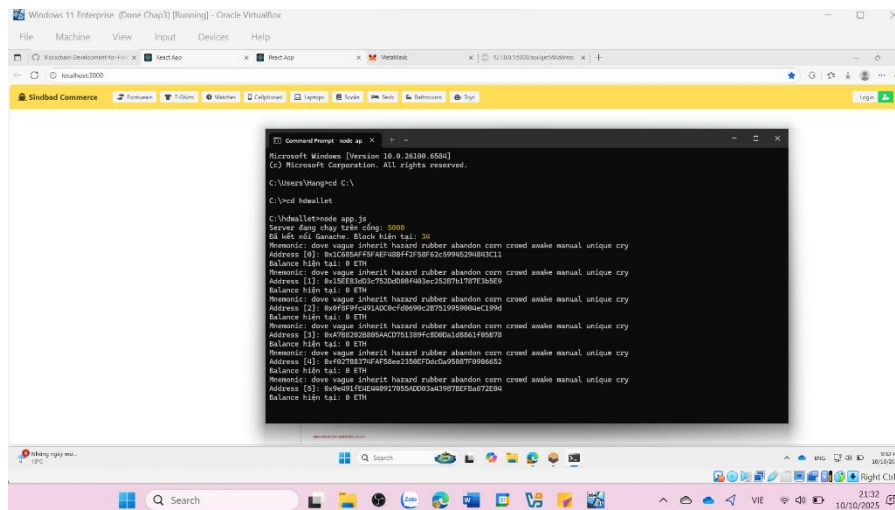
- **Các account của Ganache** (0x... ta thấy trong GUI của Ganache) được dùng làm **ví người mua (buyer)**. Ta sẽ lấy một account Ganache (hoặc import private key vào MetaMask) để làm người mua và dùng nó gửi ETH.

Ganache đóng vai trò là một mạng blockchain giả lập - blockchain payment ecosystem, sinh ví cho người mua. Mỗi lần mua chỉ cần 1 địa chỉ ví còn tiền. Trong khi đó, Metamask là ví của người mua, có thể connect vào Ganache để import tài khoản để mua hàng.

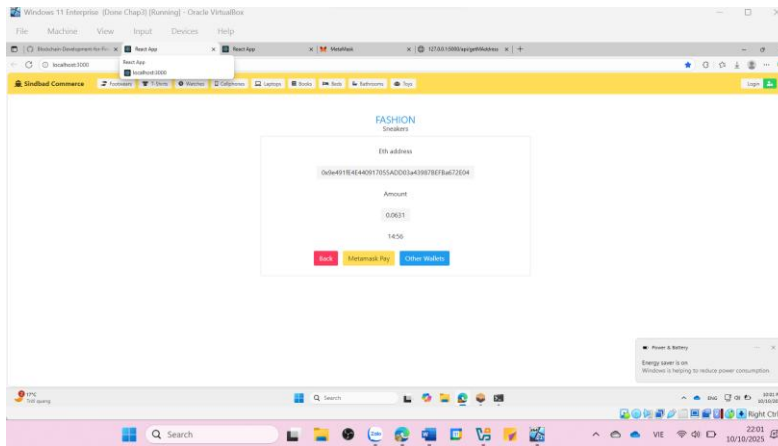


- Địa chỉ được sinh ra (derive) từ backend/hdwallet (API GET /api/getMAddress) là ví merchant (seller) — backend trả một address mới cho mỗi lần mua, rồi frontend hiển thị address đó cho người mua.

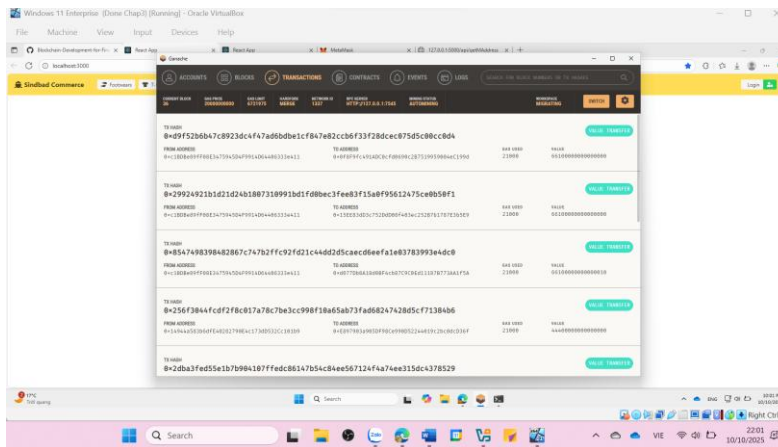
backend trả một address mới cho mỗi lần mua.



frontend hiển thị address đó cho người mua khi nhấn **Buy Now**

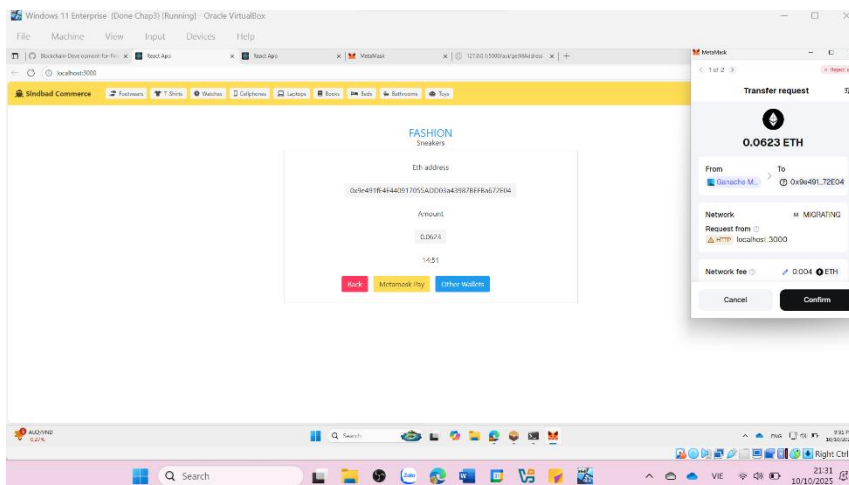


- Khi mua: **người mua chuyển ETH từ ví Ganache → ví merchant**. Backend/hdwallet sẽ kiểm tra balance của merchant address (qua node Ganache) để xác nhận thanh toán.

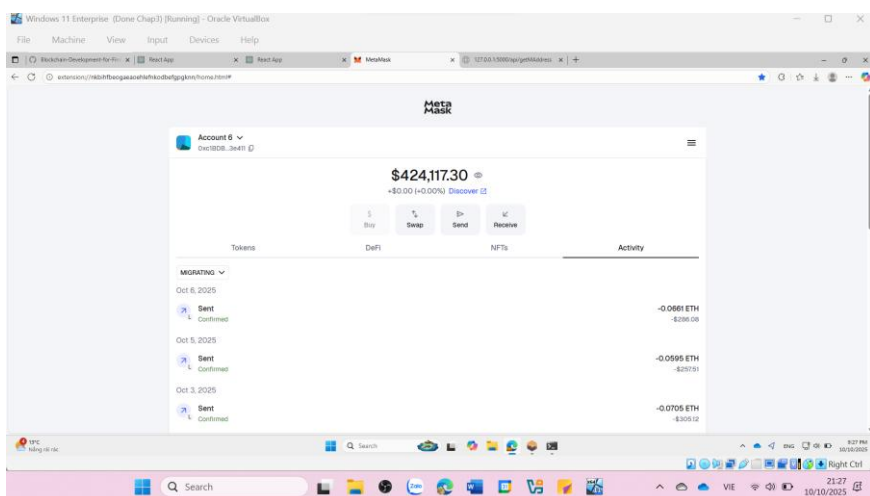
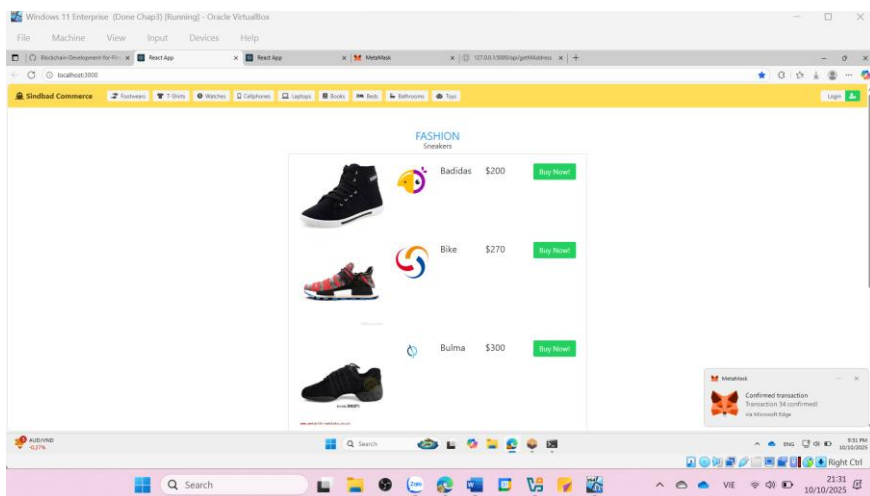


- Người mua có hai lựa chọn thanh toán: Metamask và Otherwallet

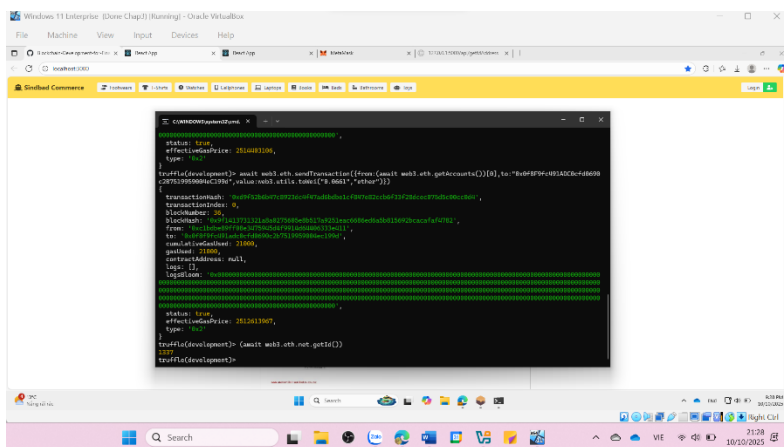
Trường hợp người mua lựa chọn thanh toán qua Metamask, trên giao diện sẽ hiện thông tin địa chỉ người mua, địa chỉ người bán, số ETH cần thanh toán. Khi người mua xác nhận thanh toán sẽ có



thông báo từ Metamask ngay trên giao diện (góc dưới bên phải) khi giao dịch thành công hay thất bại. Metamask lưu trữ lịch sử giao dịch ở mục “Activity”



Trường hợp người mua lựa chọn thanh toán qua OtherWallet, Blockchain sẽ trả về cho bạn hash của giao dịch và các thông tin chi tiết khác của giao dịch sau khi thực thi.



- Giao diện của Mwallet. Được sử dụng để người bán theo dõi thông tin chi tiết của các giao dịch

