

Corso di Programmazione Web e Mobile
A.A. 2021-2022

Coinbox

Raul Sorrentino

Indice

Coinbox

1. Introduzione	3
1.1. Breve analisi dei requisiti	
1.1.1. Destinatari	
1.1.2. Modello di valore	
1.1.3. Flusso dei dati	
1.1.4. Aspetti tecnologici	
2. Interfacce	6
3. Architettura	11
3.1. Diagramma dell'ordine gerarchico e flusso delle risorse	
3.2. Descrizione delle risorse	
3.3. Schema MongoDB	
4. Codice	14
4.1. HTML (wallet.ejs)	
4.2. CSS e JS (dark mode)	
4.3. API	
4.4. Node.js	
5. Conclusioni	19
6. Nota bibliografica e sitografica	20

1. Introduzione

Oggi si fa un gran parlare di criptovalute. Stanno determinando un cambiamento radicale nell'economia globale, con particolare riferimento al settore finanziario, sotto il profilo delle modalità di scambio di beni, servizi e ogni attività finanziaria.

Questo enorme sviluppo ha portato alla nascita di una svariata quantità di coin e token differenti. Da qui nasce l'idea dei crypto data aggregator.

Gli aggregator di dati sono una potente risorsa per gli investitori che sono alla ricerca di un'analisi più approfondita del semplice prezzo di una singola coin.

In Coinbox sono presenti strumenti utili per l'analisi, la ricerca e l'informazione di criptovalute.

Il nome **Coinbox** nasce dall'unione del termine Coin (moneta, criptovaluta) e Box (scatola) che racchiude tutte le informazioni, così come avviene nella UI.

Source code on [GitHub](#).

1.1. Breve analisi dei requisiti

1.1.1. Destinatari

I destinatari che ci si aspetta andranno ad utilizzare la dashboard saranno principalmente del settore finanziario e tecnologico, già conoscono cosa cercano. Con un livello di esperienza degli strumenti finanziari medio-alti. Per questo l'homepage del sito è proprio la dashboard. In questo modo viene fin da subito messo a disposizione dell'utente strumenti e informazioni.

Con l'aiuto del **modello di ricerca delle informazioni di Marcia Bates**

	Active	Passive
Directed	Searching	Monitoring
Undirected	Browsing	Being aware

Si trova un **target** di utente attivo che sa cosa sta cercando e consapevole, ovvero che ha una visione chiara di ciò che vuole ottenere. L'interfaccia è stata progettata fin da subito seguendo queste linee guida in modo flessibile.

Definito quale sarà il target di utenti per tipologia, possiamo intuire anche che il linguaggio con cui hanno esperienza è quello finanziario e tecnologico. Quindi un linguaggio chiaro e conciso che permetta di capire immediatamente il contesto.

Gli utenti utilizzeranno device di diverso genere per connettersi a Coinbox, da computer a smartphone e tablet. Coinbox infatti è stato progettato per essere fruito su più tipologie di device differenti. In particolare è stato progettato su lato desktop per strumenti più avanzati come i grafici e su smartphone per informazioni generali. In ogni caso su tutti i device si avrà a disposizione tutte le funzionalità sviluppate.

1.1.2. Modello di valore

Coinbox non è l'unico crypto data aggregator presente nel web, ma si è cercato di diversificarsi attraverso una semplicità d'uso, immediatezza per la fruizione degli strumenti e un'interfaccia utente semplice, minimale e moderna.

Ad oggi lo sviluppo di Coinbox non può considerarsi un aggregatore di dati completo, ma offre un punto d'inizio ottimo di quello che potrà essere sviluppato.

Per la **monetizzazione** del progetto è possibile sviluppare modelli di business differenti. Nello specifico quello freemium permette di mantenere un'interfaccia pulita senza banner pubblicitari fastidiosi in modo tale da avere introiti non sulla base di adv ma su abbonamenti premium che ne aumentano le funzionalità. Come per esempio la possibilità di creare un portafoglio personale diversificato e con analisi statistiche a supporto.

1.1.3. Flusso dei dati

I contenuti presenti in Coinbox sono in parte ottenuti da servizi che mettono a disposizione API e in parte prodotti lato server.

I contenuti quali informazioni di prezzo, capitalizzazione e news sono reperiti dalle API di CoinGecko e CryptoPanic mentre i grafici sono prodotti dal server utilizzando i dati a disposizione.

I grafici a 7 giorni della dashboard vengono aggiornati ogni giorno al 1.00am e sono salvati in png, mentre le notizie ogni 10 minuti, memorizzate in json.

Per la pubblicazione del progetto online dovranno essere inserite le menzioni dei servizi da cui vengono ottenuti i dati, come per esempio "Data provided by CoinGecko".

I dati riguardanti gli utenti sono memorizzati in un database MongoDB. Questi sono ottenuti dalle operazioni che l'utente che esegue nel sito. In particolare, al momento della registrazione, l'utente inserirà un nome, una email e una password.

1.1.4. Aspetti tecnologici

Si è utilizzato un server **Node.js** con Express.js per esporre un API REST che riceve le richieste dal client, le elabora, interroga il database quando necessario e risponde con pagine HTML, talvolta renderizzate con **EJS**, o **JSON**.

Express non si basa su nessuna struttura o layout in cui organizzare il codice, lascia questa decisione all'utente (sviluppatore). Quindi è stata eseguita una ricerca sul possibile modello utilizzabile ed è stata trovata un'organizzazione del codice in cartelle e moduli che offre semplicità, efficienza e minimalismo. Lo schema dell'organizzazione è presente nella sezione 3.2.

Sono stati utilizzati **oltre 15 pacchetti** npm all'interno del progetto per permettere di eseguire la produzione dei grafici, la creazione di schemi per il database, l'interrogazione con il database, funzioni hash unidirezionali e tante altre funzionalità utili.

I contenuti sono memorizzati seguendo lo standard JSON, selezionati dalla logica del server e resi disponibili per il client.

Nello specifico, i dati degli utenti sono memorizzati utilizzando un database NoSql **MongoDB**, lo schema è presente nella sezione 3.3. Le password sono memorizzate in hash + salt con il pacchetto **bcrypt**.

La generazione dei grafici avviene in parte lato server con **chartjs** e in parte lato client con **Google Charts API**. È stata fatta questa separazione per motivi di prestazioni e di utilizzo. Infatti i 100 grafici a 7 giorni presenti nella dashboard richiedevano una notevole computazione da parte del client e non necessitavano di essere generati aggiornati ad ogni caricamento, per questo si è preferito produrli lato server, aggiornandoli *ogni giorno al 1.00am* e rendendoli disponibili al client. Per quanto riguarda invece i grafici a linee della pagina specifica della coin e del grafico a donut del wallet personale viene utilizzata l'API di google dato che non richiedono grossa computazione ed è utile aggiornarli tempestivamente alla richiesta dell'utente.

Il rendering dei 100 grafici della dashboard viene eseguito attraverso un **Web worker** per evitare il blocco temporaneo della pagina.

Per le interrogazioni API è stato utilizzato **AXIOS** npm e **Fetch API**.

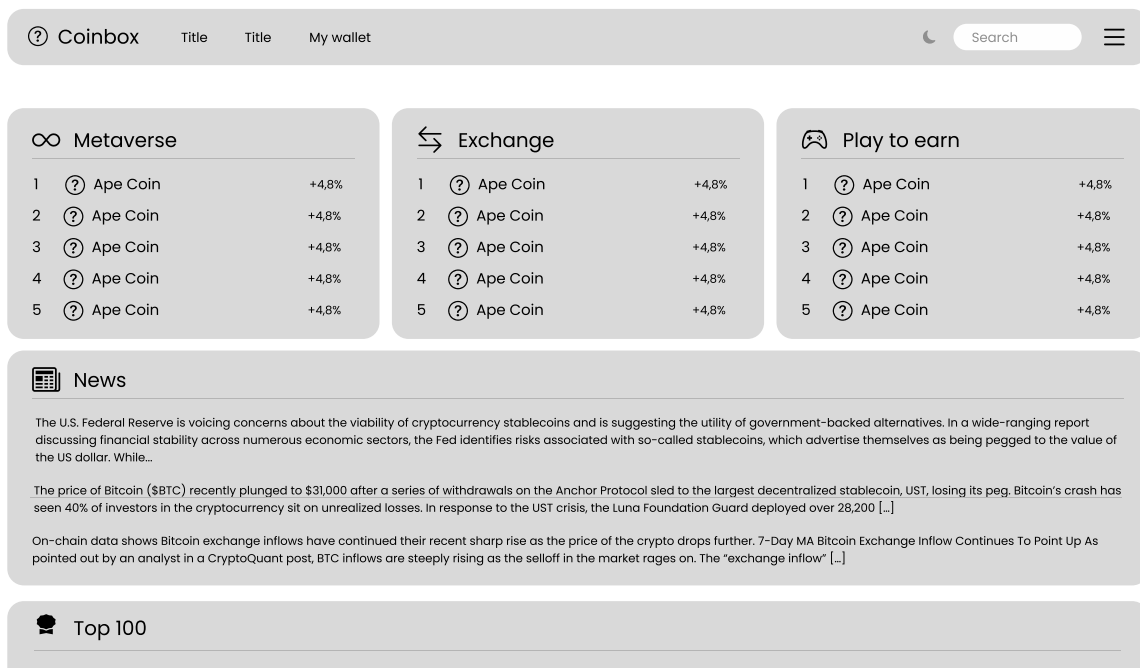
La scelta tra **Dark mode e Light mode** viene eseguita in automatico durante il rendering della pagina, a seconda della preferenza impostata nel sistema operativo oppure, se presente, con la preferenza dell'utente memorizzata in Local storage.

Generata la pagina, la mode può essere modificata dall'utente con il click di un'icona (luna o sole) oppure modificata in automatico se durante la fruizione del sito, la preferenza del sistema operativo viene modificata.

2. Interfacce

Le **interfacce** si suddividono in pagine HTML ed EJS, di cui una, quella del wallet, accessibile solo dopo aver effettuato il login.

La **struttura** delle pagine, prima di essere sviluppate con il codice, ha seguito un processo di prototipazione, partendo da alcuni sketch in carta e penna per un inquadramento generale del layout della pagine e la strutturazione dei contenuti, per poi passare al wireframe *low fidelity* con l'utilizzo di un editor di grafica vettoriale, Figma.

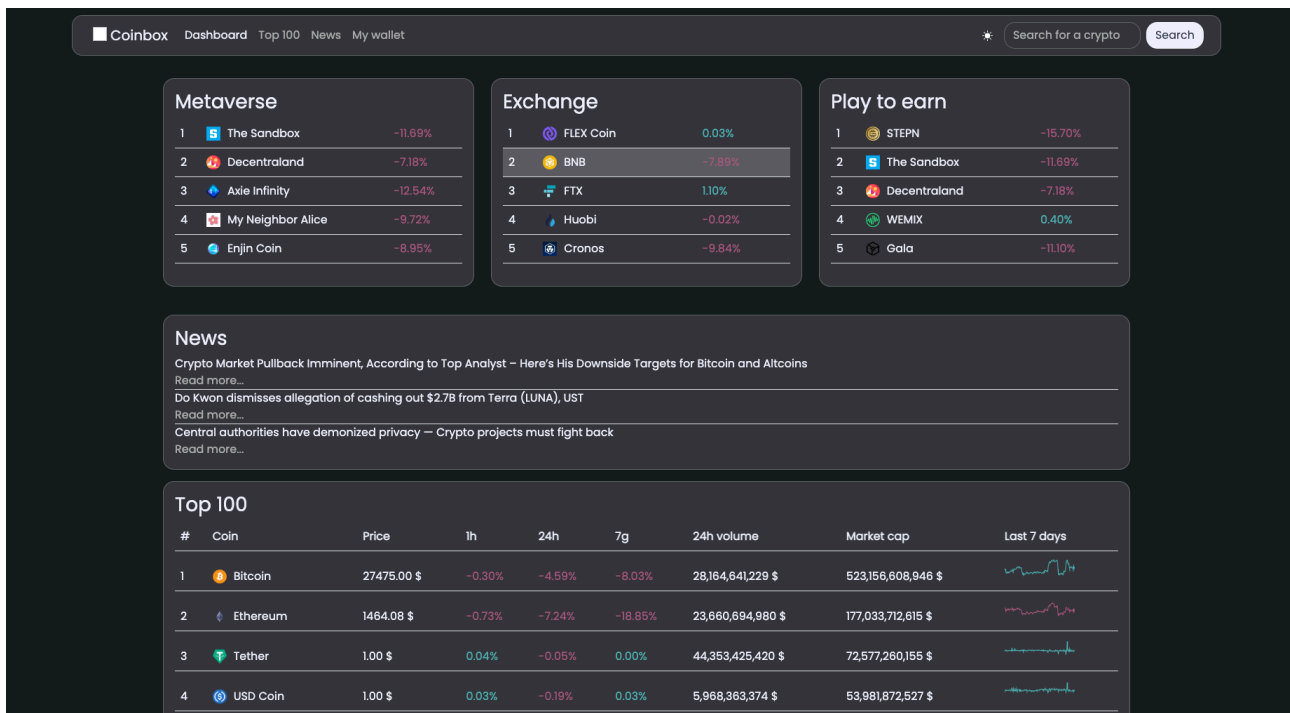


La **User Interface** comune a tutte le pagine è stata sviluppata seguendo un processo di ricerca e trend. È stata creata una moodboard di cui ne sono state seguite le linee. Il design utilizzato è quello che viene chiamato Glass UI, un design moderno utilizzato in alcuni dei più popolari sistemi operativi come iOS e Windows 11. Questo design mostra il massimo del suo valore quando si utilizza con più layer impilati.

La **User Experience**, ovvero l'interazione tra uomo e prodotto, si è cercato di portarla ad un buon livello, soprattutto grazie al minimalismo e semplicità di Coinbox. Durante e post sviluppo, dopo qualche test di utilizzo, alcune delle parti sono state modificate per aumentarne l'esperienza.

Lo **stile** della pagina è racchiuso all'interno del file coinbox.css, per mantenerlo diviso dalla strutturazione della pagina.

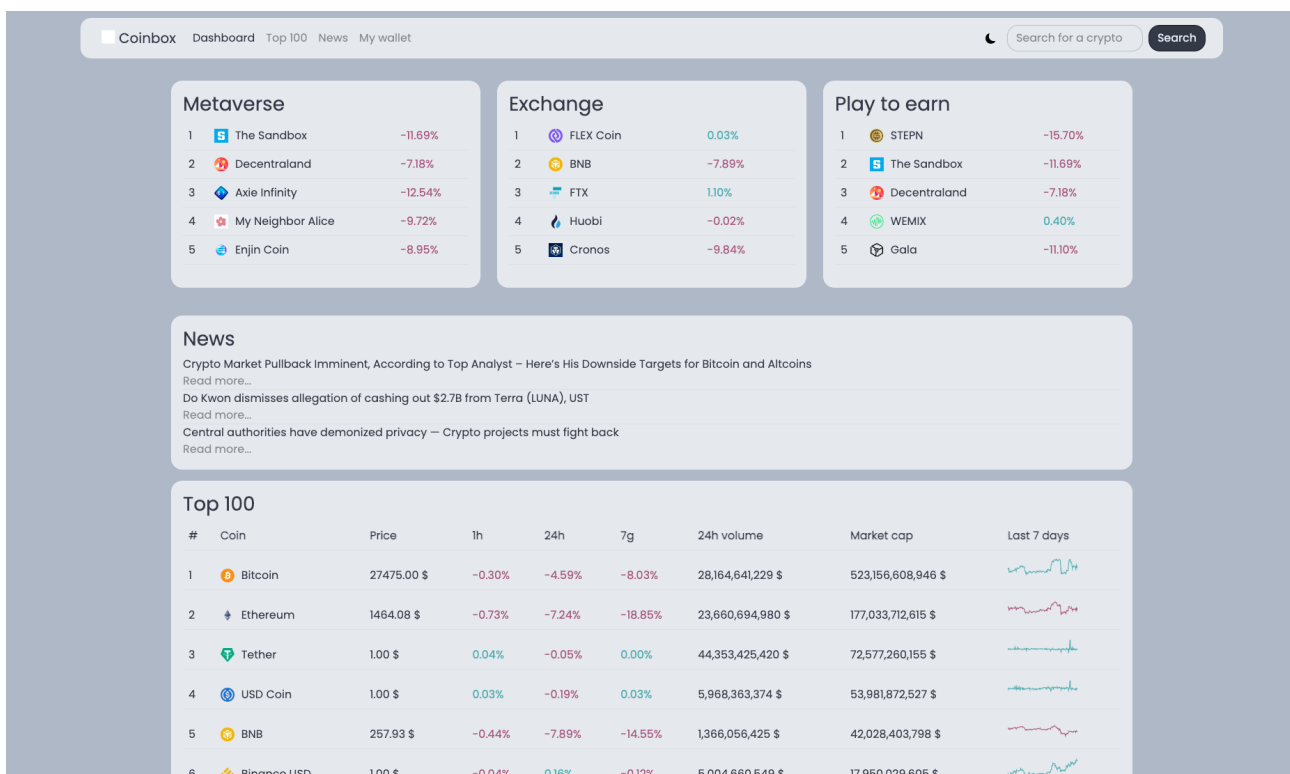
Dashboard (dark mode)



Questa è la pagina principale di Coinbox, una dashboard che racchiude un grande quantitativo di informazioni senza appesantire la visualizzazione. Sono presenti classifiche delle coin, prezzi, variazioni percentuali, capitalizzazioni, grafici a 7 giorni...

La navbar presenta link ad altre pagine, icona di preferenza della mode e un campo di ricerca per le coin.

Dashboard (light mode)

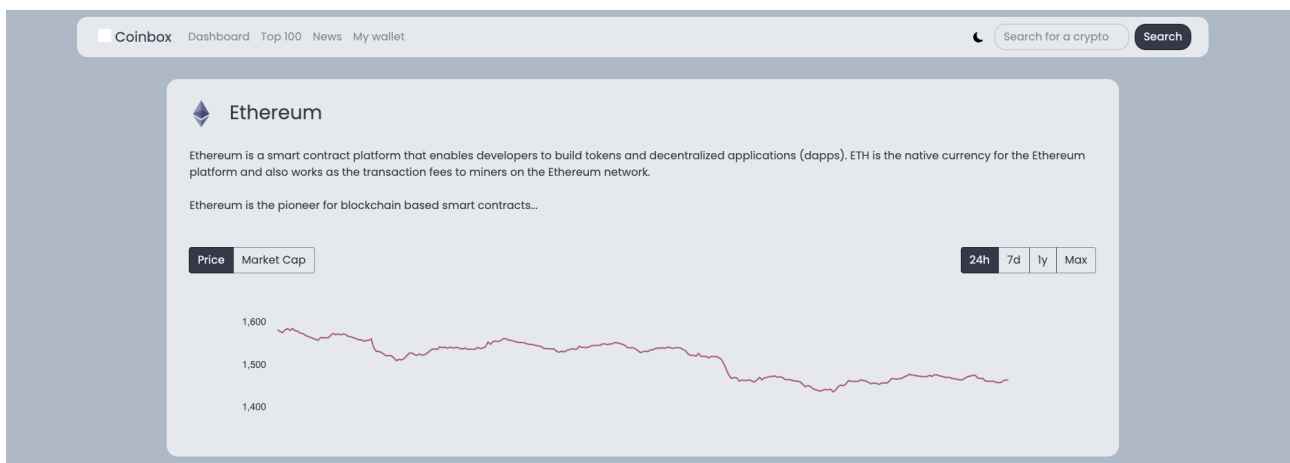


Coin page (dark mode)

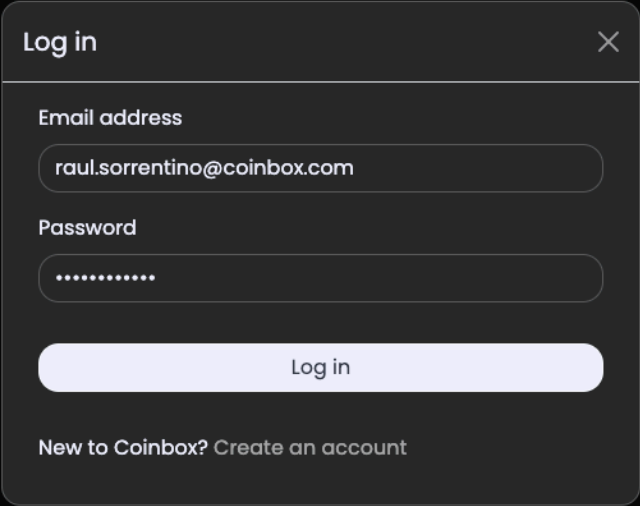


Questa è la pagina specifica della coin cercata. Permette di visualizzare una breve descrizione e il grafico di prezzo o capitalizzazione con 4 scelte di timeframe diversi. Spostando il mouse sulla linea viene mostrato un popover con data e valore di prezzo in quel preciso punto. Potranno essere aggiunte nuove funzionalità, per esempio, il sentiment e altri strumenti utili.

Coin page (light mode)

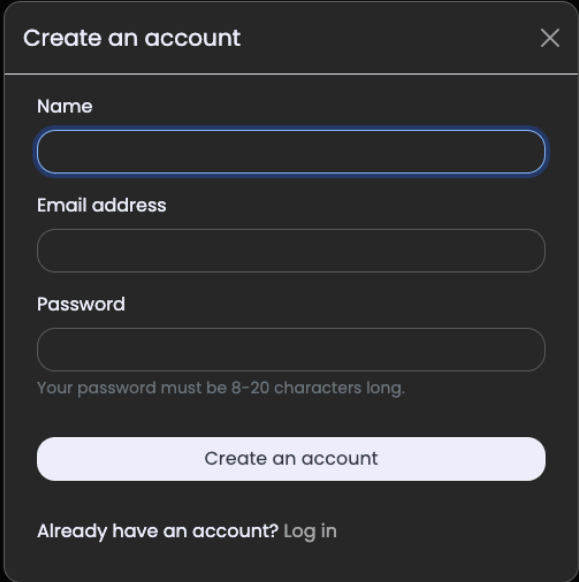


Log in (dark mode)



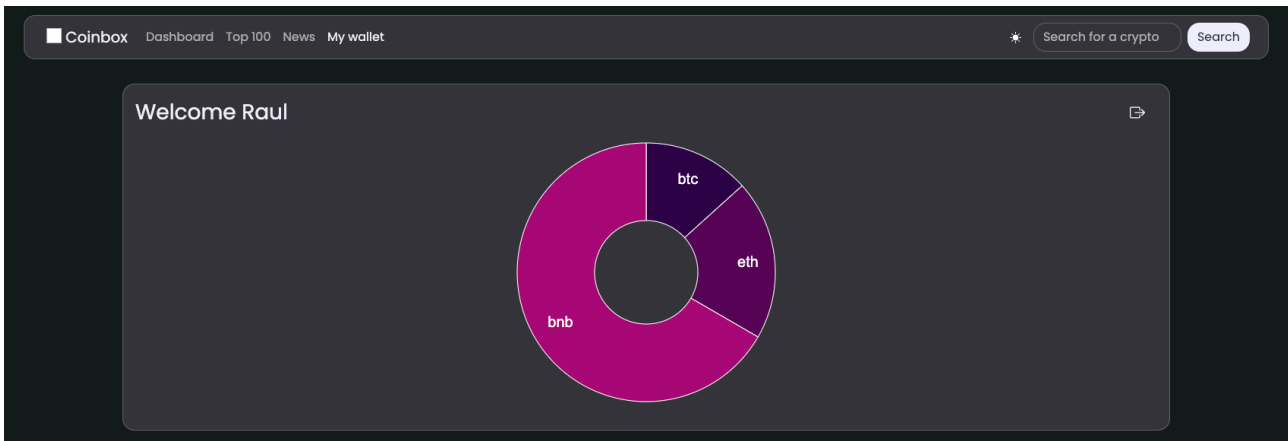
A dark-themed login modal window titled "Log in" with a close button (X) in the top right corner. It contains two input fields: "Email address" with the value "raul.sorrentino@coinbox.com" and "Password" with masked characters ".....". Below the inputs is a light blue "Log in" button. At the bottom, there is a link: "New to Coinbox? Create an account".

Create an account (dark mode)



A dark-themed "Create an account" modal window with a close button (X) in the top right corner. It features three input fields: "Name", "Email address", and "Password". Below the "Password" field is a text requirement: "Your password must be 8-20 characters long." A light blue "Create an account" button is positioned below the inputs. At the bottom, there is a link: "Already have an account? Log in".

My wallet (dark mode)



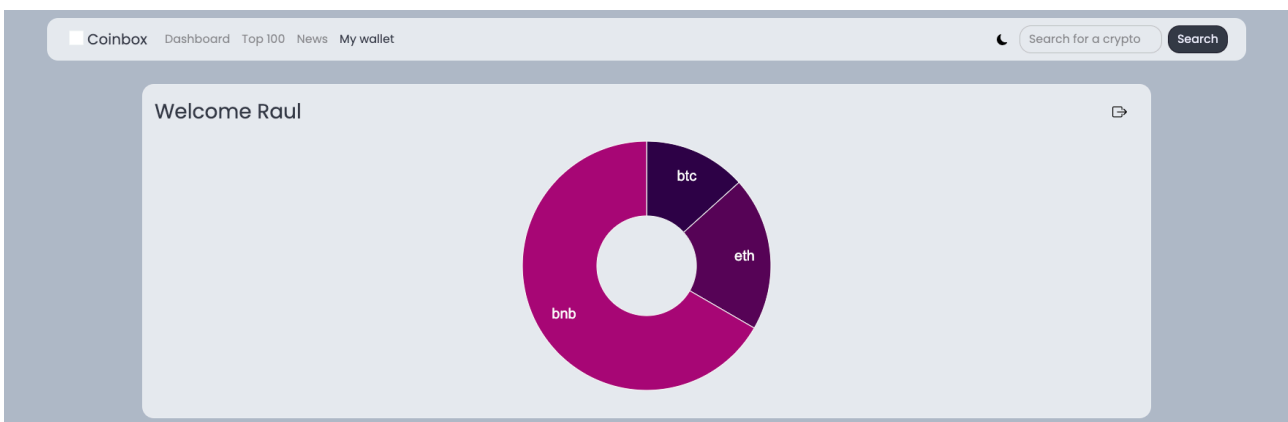
Nella pagina “My wallet” è possibile, dopo aver eseguito l’accesso, visualizzare la suddivisione del portafoglio totale attraverso un grafico a donut.

Il grafico mostra con un hover del mouse su una fetta, la sua percentuale.

In questo caso potranno essere aggiunte altre funzionalità utili come analisi del portafoglio, diversificazione, percentuale di profit/loss, ...

È possibile effettuare il logout con il click dell'icona in alto a destra.

My wallet (light mode)

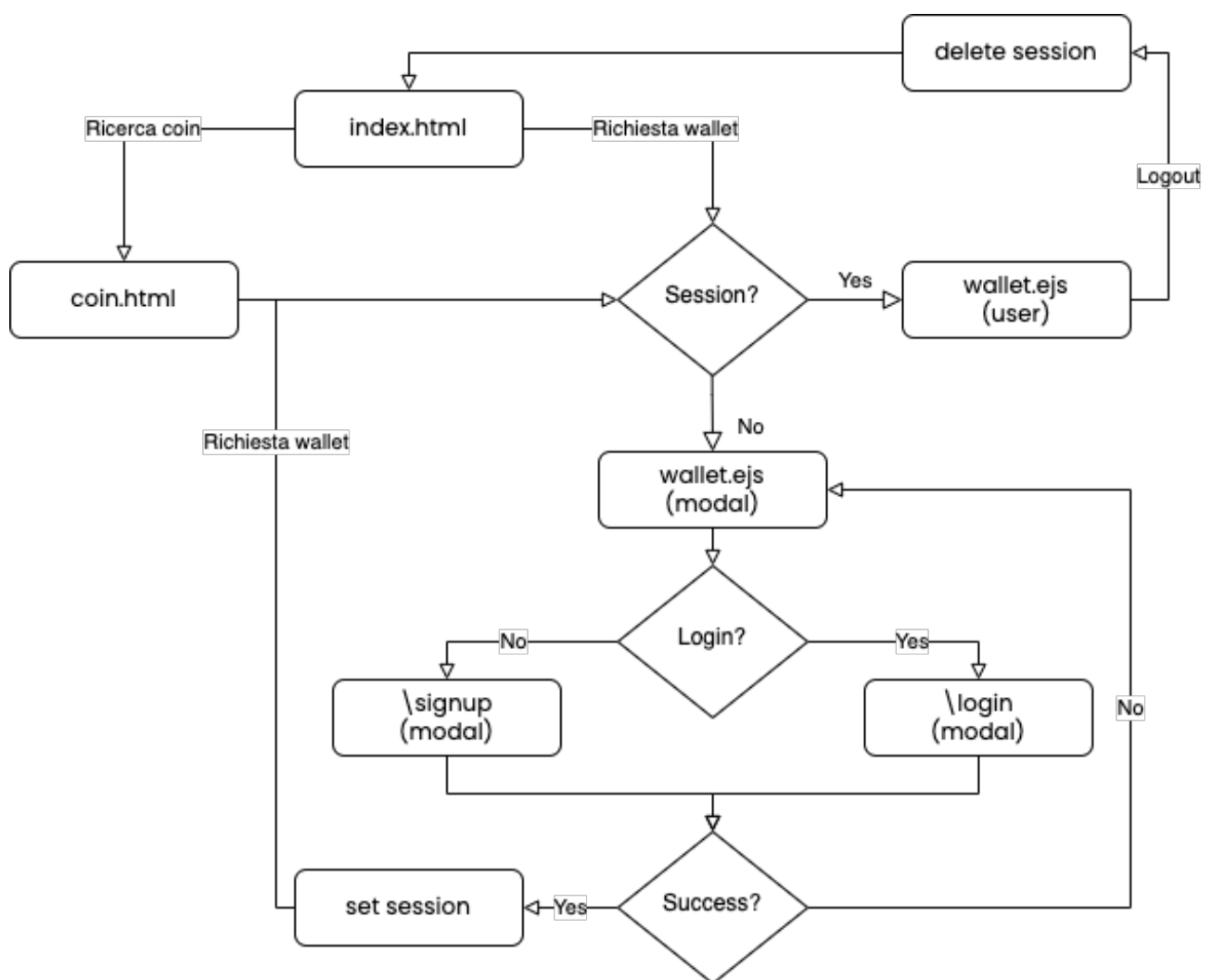


3. Architettura

3.1. Diagramma dell'ordine gerarchico e flusso delle risorse

L'interfaccia utente del sito è suddivisa su più pagine. Ognuna di queste pagine è dotata di una navbar che permette di navigare da una all'altra in modo rapido.

Lo schema rappresenta anche un esempio del flusso che l'utente potrebbe seguire. Alcune parti sono state tralasciate per non appesantirlo.



3.2. Descrizione delle risorse

Tutte le pagine di Coinbox sono dinamiche ed interagiscono con servizi esterni e server.

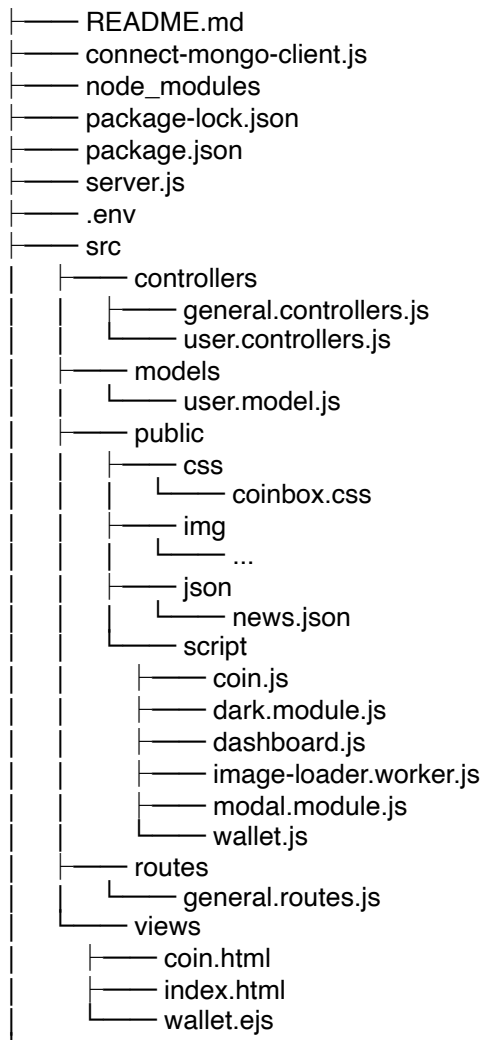
La dashboard, per esempio, è una pagina descrittiva contenente informazioni strutturate in forma tabellare. Le informazioni sono ottenute con richieste API a servizi esterni e verso il server di Coinbox.

I grafici sono ottenuti con una richiesta GET al server, il quale risponde con 100 png, già pronte, che poi vengono renderizzate dal Web worker.

Le notizie visualizzate sono ottenute con una richiesta GET verso il server di Coinbox che risponde con un file JSON. Le notizie sono aggiornate ogni 10 minuti dal server che fa una richiesta verso un servizio esterno.

L'organizzazione del codice in cartelle e moduli offre semplicità, efficienza e permette di scrivere codice mantenibile. Il progetto segue questa struttura:

Coinbox



Nella cartella controllers sono presenti le funzioni che ottengono la richiesta dalle route e le convertono in risposte HTTP.

Nella cartella models sono presenti gli schemi del database. Nello specifico schemi delle Collections.

Nella cartella routes sono presenti i percorsi utilizzati dalle richieste del client con metodi HTTP a un endpoint particolare che richiamano i controller.

Nella cartella public sono presenti le risorse utilizzate dalle pagine, come script, file html o ejs, file di stile, immagini, ...

Nel file .env sono contenute le API KEY, porta, nome del database, url del database, ...

Nella cartella views sono contenute le interfacce, i modelli che devono essere visualizzati.

L'organizzazione potrebbe avere più suddivisioni in cartelle specifiche, ma si è scelto di ignorarne alcune data la dimensione del progetto e per evitare di avere un codice più difficile da sviluppare con divisioni più specifiche. Alcune altre possibili sottocartelle sono configs, services, utils, ...

3.3. Schema MongoDB

Lo schema della **collection Users** in cui sono contenute le informazioni dell'utente.

```
{
  "_id" : ObjectId("62a12e700f45d3bd5dc6e345"),
  "name" : "Raul",
  "email" : "raul.sorrentino@coinbox.com",
  "password" : "$2b$10$mYBrtgyXu17oRY4XysSdWuKlokXZrOgcrKuQhz9npjaz3zkZaJgJy",
  "wallet" : {
    "btc" : 2,
    "eth" : 3,
    "bnb" : 10
  },
  "__v" : 0
}
```

La password è memorizzata in hash e salt con bcrypt.

La stringa di hash contiene queste informazioni:

- Funzione di hash bcrypt \$2b\$.
- Parametro di costo che è un input dell'algoritmo di cifratura, specifica un conteggio iterativo di espansione della chiave in una potenza di due.
- Sale.
- Hash risultante.

4. Codice

Solo alcune delle parti significative del progetto. All'interno del codice sono presenti commenti con lo scopo di descrivere le caratteristiche funzionali. Questi hanno notevole importanza per il programmatore stesso e per persone diverse per aumentarne la leggibilità e favorendo la manutenzione.

4.1. HTML (wallet.ejs)

Snippet della pagina wallet.ejs che permette la visualizzazione del portafoglio personale dopo aver fatto il login. Controlla la presenza della sessione con il parametro inserito al momento del render nel server. Se non esiste la sessione viene mostrato il modale per il login.

```
<!doctype html>
<html lang="en">
<head>
  ...

  <!-- Coinbox CSS -->
  <link href="css/coinbox.css" rel="stylesheet"> ...

  <title>Coinbox |
    <%= title %>
  </title>
</head>

<body>
  ...

  <!-- Modal Log in -->
  ...

  <!-- Modal Create an account -->
  ...
  <form action="/signup" method="POST">
    ...
    <button type="submit" class="btn btn-outline-light">Create an account</button>
    <p>Already have an account? <a data-bs-dismiss="modal" data-bs-toggle="modal" href="#loginModal">Log in</a></p>
  </form>
  ...

  <script src="https://cdn.jsdelivr.net/npm/@popper..."></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3..."></script>

  <!-- Coinbox script -->
  <% if( logStatus ) { %>
    <script src="script/wallet.js"></script>
    <% }; %>
    <script src="script/dark.module.js"></script>
    <script src="script/modal.module.js"></script>

    <%if((!logStatus) && (logMode == 'signup')){%>
      <script>
        $(window).load(function() {
          $('#createAccountModal').modal('show');
        });
      </script>
    <% }; %>

    <%if((!logStatus) && (logMode == 'login')){%>
      <script>
        $(window).load(function() {
          $('#loginModal').modal('show');
        });
      </script>
    <% }; %>

  </body>
</html>
```

4.2. CSS e JS (dark mode)

La dark mode e lightmode è definita utilizzando all'interno del foglio di stile delle variabili che permettono di modificare il valore di un attributo a seconda della presenza o meno della classe `.dark-theme-variables` nel body.

```
:root {
  --color-light: #edeffd;
  --color-dark: #363949;
  --color-dark-disable: rgba(0, 0, 0, 0.45);
  --color-dark-variant: #677483;
  --color-background: rgb(174, 185, 201);
  --mode-img-darkmode: url("../img/icon/moon-fill.svg");
  --mode-img-logout: url("../img/icon/box-arrow-right.svg");
  --filter-icon: none;
  --color-dark-border: rgba(0, 0, 0, 0.20);
  --color-glass: rgba(255, 255, 255, 0.7);
}

.dark-theme-variables {
  --color-light: #363949;
  --color-dark: #edeffd;
  --color-dark-disable: rgba(255, 255, 255, .55);
  --color-dark-variant: #a3bdcc;
  --color-background: #181a1e;
  --mode-img-darkmode: url("../img/icon/brightness-high-fill.svg");
  --mode-img-logout: url("../img/icon/box-arrow-right.svg");
  --filter-icon: invert(100%) sepia(100%) saturate(2%) hue-rotate(120deg) brightness(107%) contrast(101%);
  --color-dark-border: rgba(255, 255, 255, 0.2);
  --color-glass: rgba(255, 255, 255, 0.1);
}

body {
  background: var(--color-background);
  font-family: 'Poppins';
  color: var(--color-dark);
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}
. . .
```

La logica è gestita in un modulo specifico, denominato `dark.module.js`, importato in tutte le views.

```
/**
 * It takes the local storage content and manipulates it,
 * if the content is not present it creates it according to the system preferences.
 */
function allStorage() {
  let objDarkMode = JSON.parse(localStorage.getItem('objDarkMode'));
  if (!objDarkMode) {
    objDarkMode = {
      darkMode: (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches),
      date: new Date()
    };
    localStorage.setItem('objDarkMode', JSON.stringify(objDarkMode));
  }
  updateToggleDarkMode(objDarkMode.darkMode);
}

document.addEventListener('DOMContentLoaded', allStorage());

/**
 * Sets dark mode or light mode depending on the input boolean value.
 * @param {Boolean} darkMode True for dark mode.
 */
function updateToggleDarkMode(darkMode) {
  let body = document.body;
  if (darkMode) {
    body.classList.add('dark-theme-variables');
  } else {
    body.classList.remove('dark-theme-variables');
  }

  let obj = {
    darkMode: darkMode,
    date: new Date()
  };
  localStorage.setItem('objDarkMode', JSON.stringify(obj));
}
```

```

/**
 * Depending on the state of the switch, it changes the mode and saves the new preference in local storage.
 */
function switchDarkMode() {
  updateToggleDarkMode(!document.body.classList.contains('dark-theme-variables'));
}

. . .

```

Nello snippet riportato, sono presenti le funzioni utilizzate per impostare la mode seguendo le preferenze dell'utente, prelevate dal Local Storage oppure, se quest'ultime non sono presenti allora verrà impostata secondo la preferenza del sistema.

Seguono altre funzioni ed eventi relativi alla mode, presenti nel codice sorgente, che permettono di mantenere aggiornata la preferenza del sistema operativo se questa viene modificata durante l'utilizzo di Coinbox.

4.3. API

La ricerca di una coin implica di eseguire delle chiamate API. Con Fetch è stato possibile recuperare le risorse in modo asincrono attraverso la rete.

Le risorse ricevute vengono manipolate e rese disponibili all'interno della pagina. Nel caso della description, il testo conteneva codice html di formattazione del testo, questo è stato ripulito utilizzando `.replace()` con regex.

```

/**
 * Returns the searched coin.
 */
async function getCoin() {
  let responseSearch = await fetch(`https://api.coingecko.com/api/v3/search?query=${coin}`, {
    method: "GET"
  });
  let jsonSearch = await responseSearch.json();
  id = jsonSearch.coins[0].id;

  let responseCoin = await fetch(`https://api.coingecko.com/api/v3/coins/${jsonSearch.coins[0].id}?
  localization=false&tickers=false...`, {
    method: "GET"
  });
  let jsonCoin = await responseCoin.json();

  const title = document.getElementById('coinTitle');
  const icon = document.getElementById('coinIcon');
  const description = document.getElementById('coinDescription');

  icon.src = jsonSearch.coins[0].large;
  title.textContent = jsonSearch.coins[0].name;

  // regex looks for <, an optional slash /, one or more characters that are not >, then either > or $ (the end of the
  line)
  let descriptionText = (jsonCoin.description.en).replace(/(<([^\>]+)>)/ig, '');
  const descriptionArr = descriptionText.split('.');
  descriptionText = descriptionArr[0] + '.' + descriptionArr[1] + '.' + descriptionArr[2] + '...';

  description.innerText = descriptionText;

  getChart(jsonSearch.coins[0].id, 'price', 1);
}

async function getChart(id, type, days) {
  let responseChart = await fetch(`https://api.coingecko.com/api/v3/coins/${id}/market_chart?vs_currency=usd&days=${days}`, {
    method: "GET"
  });
  let jsonChart = await responseChart.json();

  drawChart('chart', (type == 'price') ? jsonChart.prices : jsonChart.market_caps);
}

```


Il grafico invece, utilizzando l'API di Google Chart, viene creato con i dati ricevuti dalle API di CoinGecko. Di seguito lo snippet di codice che permette la generazione:

```
/**
 * Callback that creates and populates a data table, instantiates the chart, passes in the data and draws it.
 * The drawn chart will be green if (firstPrice < lastPrice) otherwise it will be red.
 * @param {String} id Id of the HTML element in which to insert the chart.
 * @param {Array} arrayPrice Price array
 */
function drawChart(id, arrayPrice) {

    var dark = document.body.classList.contains('dark-theme-variables');

    let dataChart = [
        ['Time', 'Price']
    ];
    for (let index = 0; index < arrayPrice.length; index++) {
        const date = new Date(arrayPrice[index][0]).toLocaleDateString();
        dataChart.push([date, arrayPrice[index][1]])
    }

    const firstPrice = dataChart[1][1];
    const lastPrice = dataChart[dataChart.length - 1][1];

    var data = google.visualization.arrayToDataTable(dataChart);

    var options = {
        colors: (firstPrice < lastPrice) ? ['#50afb2'] : ['#ab5780'],
        legend: 'none',
        vAxis: {
            textStyle: { color: (dark) ? '#edeffd' : '#363949' },
            gridlines: {
                color: 'transparent'
            },
            baselineColor: 'transparent'
        },
        hAxis: {
            textPosition: 'none',
            textStyle: { color: (dark) ? '#edeffd' : '#363949' },
            gridlines: {
                color: 'transparent'
            },
            baselineColor: 'transparent'
        },
        backgroundColor: { fill: 'transparent' }
    };

    var chart = new google.visualization.LineChart(document.getElementById(id));
    chart.draw(data, options);
}
```

4.4. Node.js

Abbiamo visto come sono generati i grafici lato client, ora vediamo la generazione dei 100 grafici della top100 crypto per capitalizzazione che avviene lato server. È stato utilizzato Chart.js, un modulo npm che permette la costruzione di grafici.

```
const createImage = async(arrayData) => {

  const firstPrice = arrayData[0];
  const lastPrice = arrayData[arrayData.length - 1];

  const config = {
    type: 'line',
    data: {
      labels: Array.from(Array(arrayData.length).keys()),
      datasets: [{
        data: arrayData,
        backgroundColor: 'rgba(120, 120, 0, 0)',
        fill: false,
        borderColor: (firstPrice <= lastPrice) ? '#50afb2' : '#ab5780',
        borderWidth: 1,
        tension: 0,
      }]
    },
    options: {
      responsive: true,
      elements: {
        point: {
          radius: 0
        }
      },
      scales: {
        x: {
          display: false
        },
        y: {
          display: false
        },
      },
      plugins: {
        legend: {
          display: false,
        },
        title: {
          display: false,
          text: ''
        }
      }
    }
  },
};

const dataUrl = await chartJSNodeCanvas.renderToDataURL(config); // converts chart to image
return dataUrl;
};
```

L'output della funzione createImage() restituisce l'immagine del grafico codificata in base64. Con lo snippet di seguito, viene fatto l'encoding così da poterla scrivere, salvare con il modulo fs all'interno del server.

```
createImage(element.sparkline_in_7d.price).then(val => {

  let base64Image = val.split(';base64,').pop();
  fs.writeFile('./src/public/img/sparkline/spark' + (index + 1) + '.png', base64Image, { encoding: 'base64' }, function(err) {
    console.log('Update spark' + (index + 1));
  });

}).catch(err => {
  console.log(err);
});
```

5. Conclusioni

Come espresso nell'introduzione:

Ad oggi lo sviluppo di Coinbox non può considerarsi un aggregatore di dati completo, ma offre un punto d'inizio ottimo di quello che potrà essere sviluppato.

Coinbox può essere perfezionato in alcuni aspetti lato UX e con nuove funzionalità aggiuntive.

Il progetto ha seguito un processo di prototipazione, partendo da alcuni sketch in carta e penna, così da avere un inquadramento generale dei possibili layout delle pagine, per poi passare al wireframe low fidelity e quindi alle interfacce finali.

Anche il codice è stato sviluppato seguendo una struttura, per moduli e funzionalità, definita prima ancora di iniziare a scrivere.

Durante lo sviluppo sono emerse problematiche relative all'UX che sono state corrette modificando in parte la struttura del wireframe.

Sono state fatte delle scelte di carattere tecnologico spinte dai requisiti necessari, di funzionalità ed economici.

Le difficoltà riscontrate, come la generazione dei grafici con due modalità differenti, o con la parte di login, o con la memorizzazione della password in hash, sono state un momento di crescita e mi hanno permesso di ampliare le conoscenze tratte da questo corso.

Source code on [GitHub](#).

6. Nota bibliografica e sitografica

- (1) Materiale didattico
- (2) <https://coderrocketfuel.com/article/store-passwords-in-mongodb-with-node-js-mongoose-and-bcrypt>, Nick Major, 10 novembre 2021
- (3) <https://www.mongodb.com/blog/post/password-authentication-with-mongoose-part-1>, mongoDB, 4 ottobre 2012, aggiornato 30 aprile 2019
- (4) <https://developers.google.com/chart>, Google
- (5) <https://www.coingecko.com>
- (6) <https://cryptopanic.com>
- (7) <https://www.npmjs.com>
- (8) <https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/>
- (9) <https://expressjs.com>