

Міністерство освіти і науки України
Вінницький національний технічний університет
Кафедра програмного забезпечення

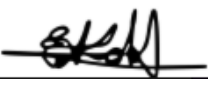
КУРСОВА РОБОТА
з дисципліни «Алгоритми та структури даних»
на тему: Дашборд по моніторингу
криптовалют

студента (ки) __2__ курсу _ групи _ПІ-206
галузі знань 12 «Інформаційні технології»
спеціальності 121
”Інженерія програмного забезпечення”
Кучера Максима

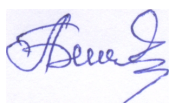
Керівник доцент кафедри ПЗ, к.т.н., Коваленко О.О.

Кількість балів: __82__ Оцінка: ECTS _____

Члени комісії


(підпис)

Коваленко О.О.
(прізвище та ініціали)


(підпис)


Ліщинська Л.Б.
(прізвище та ініціали)

м. Вінниця - 2021 рік

Міністерство освіти і науки України
Вінницький національний технічний університет Факультет
інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри ПЗ, проф., д.т.н.



(підпис) О.Н. Романюк

"28" Вересня 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ*

на курсову роботу з дисципліни "Алгоритми та структури
даних" студенту групи 2ПІ-20Б Кучера Максима
РОЗРОБКА ПРОГРАМИ ДАШБОРДУ ПО МОНІТОРИНГУ КРИПТОВАЛЮТ

Вихідні дані:

Теоретичні відомості щодо базових алгоритмів, предметної області використання програмних продуктів автоматизації роботи бібліотеки. Аналоги автоматизованих бібліотечних систем. Обґрунтований вибір мови та середовища розробки

Зміст ПЗ до курсової роботи:

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
ВСТУП

1. ПРЕДМЕТНА ОБЛАСТЬ РОЗРОБКИ ДАШБОРДУ ПО МОНІТОРИНГУ КРИПТОВАЛЮТ

- 1.1 Аналіз відомих підходів до побудови систем дашборду.
- 1.2 Обґрунтований вибір середовища та мови програмування.
- 1.3 Постановка задачі.

2. АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ РОЗРОБКИ ДАШБОРДУ

- 2.1 Базові алгоритми та структури даних.
- 2.2 Особливості загального алгоритму дашборду по моніторингу криптовалют.
- 2.3 Базові алгоритми для розробки дашборду по моніторингу криптовалют.

3. РЕАЛІЗАЦІЯ ПРОГРАМИ ДАШБОРДУ


- 3.1 Реалізація програмного продукту.
- 3.2. Реалізація вхідного та вихідного інтерфейсів.
- 3.3 Тестовий приклад. Відлагодження програмного продукту.


ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Додатки

Дата видачі "28" Вересня 2021 р.

Завдання отримав 
(підпис) Кучер М.Ф.

Керівник 
(підпис)

Зміст курсової роботи в індивідуальному завданні може бути сформовано без деталізації пунктів

АНОТАЦІЯ

Курсова робота присвячена вивченню та реалізації алгоритмів та структур даних для створення програми «Дашборду з моніторингу криптовалют» з використанням мови програмування JavaScript, технології React & Redux. Програма включає в себе роботу з типовими структурами даних, алгоритмами, обробку числової інформації і роботу з графікою, а також з системою керування базами даних. Програмний продукт допоможе освоїти сферу криптовалют за мінімальний час. Застосунок поділений на модулі, які розподілені в команді, це значно полегшує роботу та дає можливість працювати кожному над своїм функціоналом.

В результаті виконання курсової роботи отримано програмний продукт, який реагує на певні команди користувача і в якому було вирішено ряд проблем, які існують в розглянутих аналогах.

ЗМІСТ

ВСТУП	5
1 ПРЕДМЕТНА ОБЛАСТЬ РОЗРОБКИ ДАШБОРДУ ПО МОНІТОРИНГУ КРИПТОВАЛЮТ	7
1.1 Аналіз відомих підходів до побудови систем дашборду	7
1.2 Обґрунтований вибір середовища та мови програмування	9
1.3 Постановка задачі.	12
2 РОЗРОБКА ПІДСИСТЕМИ ГРАФІЧНОГО ВІДОБРАЖЕННЯ	14
2.1 Базові алгоритми та структури даних	14
2.2 Особливості загального алгоритму дашборду по моніторингу криптовалют	23
2.3 Базові алгоритми для розробки дашборду по моніторингу криптовалют	25
3 РОЗРОБКА ДІАГРАМ КЛАСІВ, ОБ'ЄКТІВ ТА СТАНУ	27
3.1 Реалізація програмного продукту	30
3.2 Реалізація вхідного та вихідного інтерфейсів	33
3.3 Тестовий приклад. Відлагодження програмного продукту	37
ВИСНОВОК	37
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	39
ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ	41

ВСТУП

Актуальність: сфера криптовалют дуже сильно зросла за останні роки, у цю сферу входить нові люди, яким важко звикнути до функціоналу і інтерфейсу великих бірж, але інтерфейс дашборда і інформативний, і простий у розумінні, і буде займати ланку для новачків у сфері криптовалют, щоб перевести їх на новий рівень.

Недоліки існуючих реалізацій: один з найголовніших недоліків вже створених дашбордів це перевантаження інтерфейсу непотрібною інформацією, яка заважає знайти потрібну інформацію.

Мета: метою курсової роботи було обрано дослідження алгоритмів хешування та сортування, а саме:

1. QuickSort -> Середня швидкодія $O(n\log(n))$
2. Shell's sort -> Середня швидкодія $O(n\log(n))$

та відповідні структури даних для зберігання даних та взаємодії з алгоритмами:

1. Array
2. Hash Table

Предметом курсової роботи є зберігання даних у різних видах структур, та модифікація даних через різні алгоритми, залежно від задачі, при створенні програмного продукту.

Об'єктом курсової роботи є готова програмна реалізація дашборду.

Завданнями курсової роботи є:

- проаналізувати доступні методи зберігання даних та алгоритми для взаємодії з ними.
- сформулювати загальну ідею та ідеологію програмного продукту.
- орієнтовно визначити методи зберігання даних та алгоритми, що будуть використовуватись при роботі додатку.

- проаналізувати предметну область використання програмного продукту.
- проаналізувати середовище розробки, систему контролю версій та мову програмування;
- вибрати доцільні алгоритми для додатку.
- реалізувати продукт;
- перевірити роботу додатку.

При розробці додатку використовувались методи аналізу і синтезу для більш раціонального використання пам'яті та ресурсів, і метод програмної реалізації для пришвидшення та спрощення розробки.

Перший розділ: опис потенційних користувачів, ідеї створення, і сама концепція проекту. Розкриття сутності продукту, постановка задач, вибір середовища та мови програмування.

Другий розділ: аналіз базових алгоритмів та структур даних, формування алгоритму роботи додатку, визначення як будуть зберігатись дані, та якими алгоритмами вони будуть модифікуватись.

Третій розділ: алгоритми поведінки користувача, програмна реалізація (опис головних модулів), скріншоти програми, лістинг (Додаток А),

Прикладне значення програмного продукту: ідея продукту спростити користувачам, які пов'язані з криптовалютами моніторинг актуального курсу певної монети або певної кількості монет. Одні з найголовніших особливостей проекту це легкий та мінімалістичний інтерфейс, який забезпечує легкість у користування, та простота у використанні.

Приклад використання: трейдер заключає угоди через біржу, де через перенавантаження інтерфейсу можуть відображатись не актуальні ціни на певні монети, саме для цього створювався цей проект, щоб забезпечити постійну актуальність цін.

1 ПРЕДМЕТНА ОБЛАСТЬ РОЗРОБКИ ДАШБОРДУ ПО МОНІТОРИНГУ КРИПТОВАЛЮТ

1.1 Аналіз відомих підходів до побудови систем дашборду

Ринок криптовалют доволі обширний, тому потрібно підходити з ґрунтовним аналізом до вибору пасива, який ви хочете купити, саме для цього створюються дашборди по моніторингу криптовалют. Одні з найвідоміших це: CoinMarketCap, CryptoCompare, Coin360. Деякі використовуються тільки для моніторингу інформації по монетам або конкретній монеті, деякі мають елементи для взаємодії з біржею, тобто купівля або продаж пасивів. Якщо порівнювати розроблений дашборд з існуючими аналогами, то можемо помітити, що у існуючих реалізацій дуже не зрозумілий інтерфейс, дуже велике нагромадження інформації, яка важко сприймається початківцями у сфері трейдингу.



Рисунок 1.1 – скріншот робочої області CoinMarketCap

На рисунку 1.1 зображено інтерфейс додатку CoinMarketCap, на якому відбувається вивід великої кількості інформації, що заплутує користувача, заважає йому зручно користуватись потрібною йому інформацією.

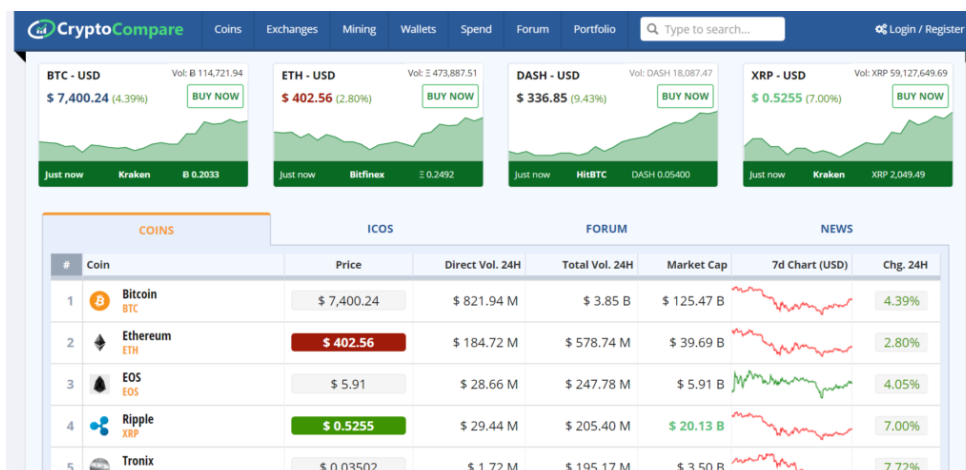


Рисунок 1.2 – скріншот робочої області CryptoCompare

Рисунок 1.2 представляє інтерфейс додатку cryptoCompare, з цього видно, що інтерфейс занадто перевантажений, має забагато лишнього функціоналу, який краще прибрати, та зробити у налаштуваннях можливість включити потрібні інструменти.



Рисунок 1.3 – скріншот робочої області Coin 360

З рисунку 1.3 можемо помітити, що цей додаток можна використовувати тільки для статистики, і не більше, адже не зрозуміло які дані виводяться, і у яких цілях їх можна використовувати.

Розробка підходить саме для початківців, тому що, їм для аналізу не потрібна велика кількість інформації, про більшу частину якої вони навіть не

знають, але для просунутих трейдерів розробка не підійде, через брак інформації, так як вони мають більше досвіду і підходять до аналізу покупки більш трезво з аналізом багатьох чинників.

1.2 Обґрунтований вибір середовища та мови програмування

Для розробки програмного засобу було обрано VS Code у якості середовища програмування, та мову програмування JavaScript, точніше бібліотеку React, як відповідну за рендеринг так званого View Layer (видимий слой, те що бачить користувач) та Redux, як стейт менеджер (зберігання інформації у додатку під час сесії юзера). Розглядались також такі мови програмування як Python, Java, та середовище програмування SublimeText 3 або IntelliJIDEA.

1.2.1 Обґрунтування середовища програмування:

У якості середовища програмування з-поміж SublimeText 3, IntelliJIDEA та VS Code було обрано останній по таким причинам:

1. Досвід використання з вибраною мовою програмування.
2. Кількість плагінів, які допомагають у розробці та пришвидшують її.
3. Можливості редактора в плані динамічних підказок, та сніпеттів.

Так як IntelliJIDEA не підходить для розробки на JavaScript, то цей варіант відпав відразу. SublimeText 3 підходить для написання коду на JavaScript, але він має доволі скудний функціонал, в порівнянні з VS Code, на додачу до цього VS Code це продукт від майкрософт, у яких є повноцінна IDE – Visual Studio, яка добре зарекомендувала себе на ринку, і це грає плюс на користь редактора від Microsoft.

1.2.2 Обґрунтування мови програмування:

Перед тим як обрати мову програмування, потрібно визначитись з типом додатку, у цій реалізації це веб-додаток, який скоріше за все має бути

реалізований на HTML5/CSS3 + інтерактив на сторінці, який має бути заданий мовою програмування JavaScript, але на нативному JavaScript це займе багато часу, взаємодія з DOM деревом, відсутність типізації, що може ускладнити подальшу підтримку проекту, тому було вибрано такий стек:

- HTML5/CSS3 – як основи веб-розробки, щоб браузер міг валідно відображати контент
- JavaScript – інтерактив на веб-сторінці, але не нативний, а з типізацією, який називається TypeScript
- React + Redux - бібліотеки JavaScript які використовуються для збільшення перформансу додатку, та спрощення розробки [1].

React – розробка компанії “Meta”, яка використовується для відображення View Layer для користувача, швидкий перформанс досягається особливістю React’а яка називається Virtual DOM – це маленький робот, який створює глибоку копію DOM і коли в додатку змінюються дані, то ререндеряться на сторінці тільки ті компоненти, де змінився контент, що значно пришвидшує роботу додатку, бо не відбувається ререндер цілої сторінки, а тільки потрібних елементів. [7]

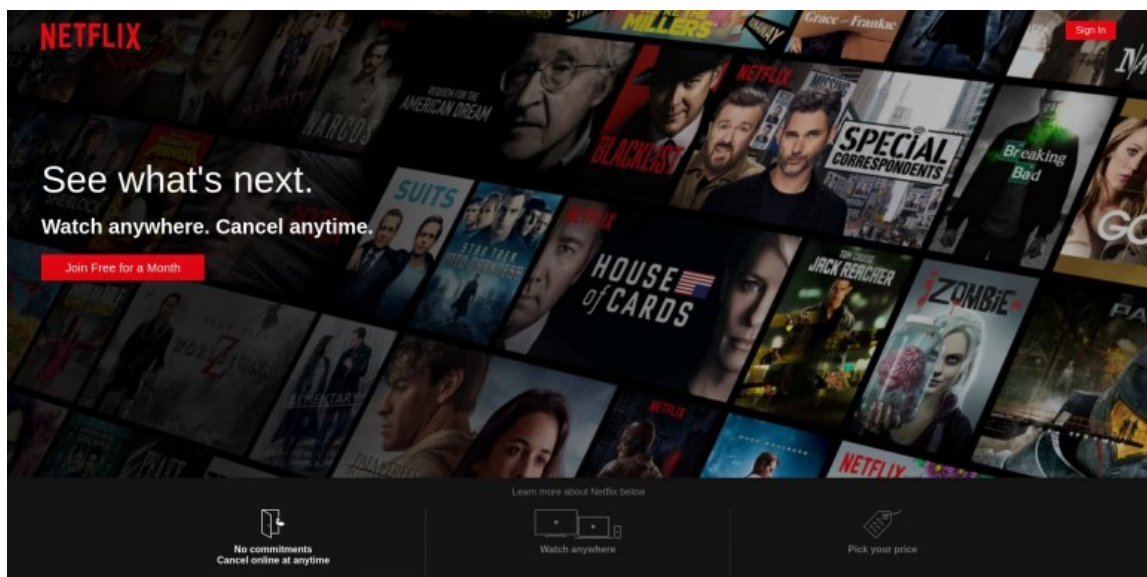


Рисунок 1.4 – Netflix веб додаток, реалізований на React

Також розглядалися такі альтернативи як Python, Java, C#, тут навіть не прийшлося глибоко порівнювати різниці при компіляції, або префоманс коду написаного на певній мові, вибір Java або C# відпав майже відразу, так як ми розробляємо фронтенд частину додатку, а дані приходять з API відкритого джерела, і зберігаються у стейт менеджері Redux, тому бекенд непотрібний у цьому додатку, а Java або C# дуже рідко використовується для розробки фронтенд частини, як мінімум через свій ООП підхід, коли у веб-сфері більше цінується функціональний метод програмування. Python більше підходить під веб, але немає досвіду розробки на ньому, також дивлячись на статистику Python далеко позаду JavaScript по вибору для розробки веб додатків. Веб-фреймворки JavaScript лідери: jQuery, React.js и Angular.js входять у трійку найпопулярніших веб-фреймворків. JavaScript такий популярний у веб-сфері, тому що від самого зародження вебу JavaScript створювався, як мова, яка додасть інтерактиву до статичного контенту на веб-сторінці, але в той час з'явилася платформа NodeJs, середовище виконання JavaScript коду, на якій можна писати серверну частину веб-сторінок, а з часом почали з'являтися фреймворки які дозволяють створювати повноцінні додатки у браузері, або десктопні додатки, мобільні.

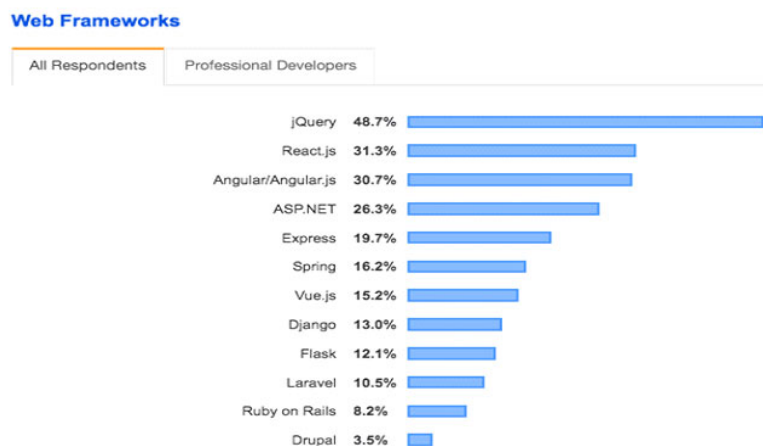


Рисунок 1.5 – Статистика фреймворків для веб-програмування [9]

Відповідно до опитування розробників 2021 года, JavaScript досі залишається лідируючою мовою програмування. Але популярність Python також виросла за останні роки, випередивши такі мови, як Java, C #, PHP и C ++.

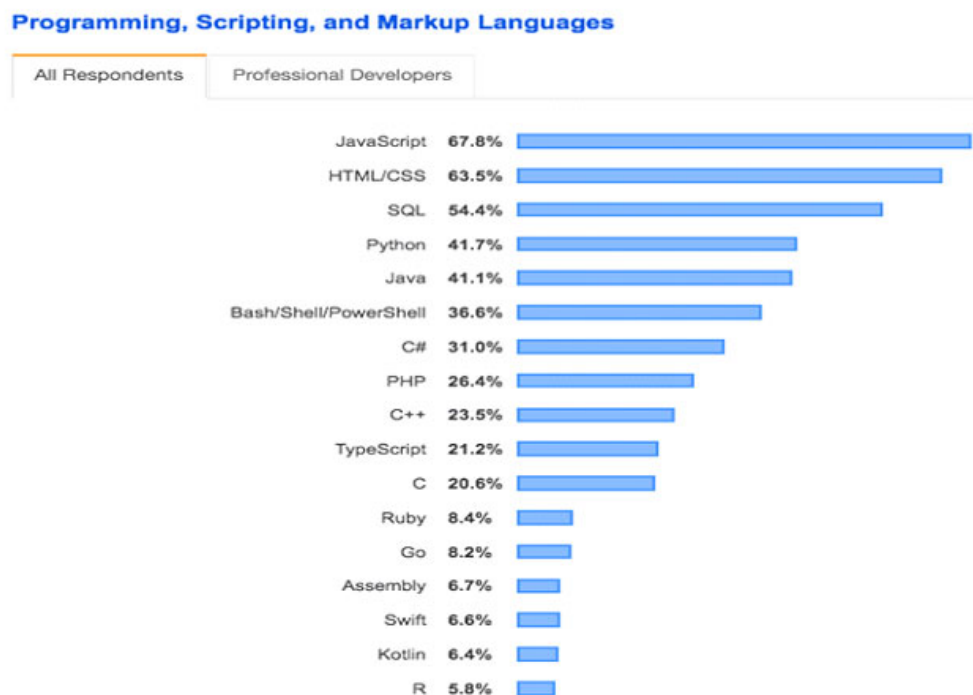


Рисунок 1.6 – Статистика мов програмування [2]

І останніми факторами у сторону JavaScript це була кількість бібліотек написаних на JS, пакетний менеджер npm, через який у тебе є доступ до мільйонів бібліотек для JavaScript, та досвід програмування на JavaScript ~2 роки [5].

1.3 Постановка задачі.

Мета курсової роботи це створити програмну реалізацію “Дашборду з моніторингу криптовалют”, та використати при розробці алгоритми відповідно до варіанту, для досягнення цієї мети потрібно виконати такі завдання:

1. Аналіз недоліків існуючих реалізацій.
2. Продумати логіку роботи власної реалізації додатку. Розробити Data

Flow – це потік даних у додатку, звідки вони з'являються, як модифікуються, де виводяться.

3. Продумати функціонал під алгоритми поставленні у завданні. Візуалізувати функціонал який буде сумісний з даними алгоритмами. Наприклад даний алгоритм QuickSort, підібрати для нього сумісний функціонал, найоптимальнішим функціоналом для алгоритма сортування може бути сортування даних, і послідуочий їх вивід юзеру.
4. Розібрати метод роботи алгоритмів.
5. Створити дизайн робочої області додатку.
6. Взяти дані для роботи додатку, та правильно вибрати структуру для їхнього зберігання. Знайти API по запиту на яке буде віддавати дані, і вибрати структуру даних, при роботі з якою швидкість алгоритмів буде найоптимальнішою.
7. Реалізувати вивід інформації користувачу, та реалізувати інтерактив, через дані алгоритми. Розробити компоненти, які можна буде перевикистати у різних частинах додатку, та налаштувати рендеринг і компонування компонентів користувачу.
8. Протестувати реалізацію додатку.
9. Написати керівництво користувача.

Отже у першому розділі було проаналізовано аналоги, виявлено їхні недоліки. Вибрано середовище програмування та мову програмування, також було обґрунтовано їх вибір. Було поставлено задачі для послідовної роботи над програмною реалізацією додатку, та відслідковування прогресу роботи. Задачі було поставлено та описано відповідно до списку виявлених проблем аналогів, мови програмування, яка буде використовуватись при реалізації, та орієнтованого типу користувачів.

2 АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ РОЗРОБКИ ДАШБОРДУ

2.1 Базові алгоритми та структури даних

У курсовій використовуються такі алгоритми як: Quick sort, Shell sort, Hashing, та такі структури даних, як: HashMap, Stack, Array, тому у цьому розділі будуть розібрані саме ці алгоритми та структури.

Quick sort: швидкий алгоритм сортування, створений Тоні Гоаром, який не потребує додаткової пам'яті, тобто він оперує пам'яттю, яка виділена під початковий масив, та виконує повний цикл сортування в середньому за $n \log n$. Алгоритм реалізує наступну логіку:

1. З масиву обирали один елемент, а весь масив розбивали на дві частини за принципом: у першій частині — не більші за даний елемент, в другій — не менші за даний елемент.
2. Процедура QuickSort(A, p, q) здійснює часткове впорядкування масиву A з p-го по q-й індекс, тобто функція QuickSort рекурсивно виконується, доки туди не потрапить масив A з 2-ма впорядкованими елементами, а потім виконуються інші рекурсивні виклики, від базового, до першого.

Алгоритм QuickSort може бути реалізований як на прикладі масиву, так і на двозв'язному списку. Час роботи алгоритму сортування залежить від збалансованості, від якої залежить розбиття масиву на 2 підмасиви. Збалансованість у свою чергу залежить від того, який елемент обрано як розбивний (відносно якого елемента виконується розбиття). Якщо розбиття збалансоване, то ітеративно алгоритм працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку ітеративна поведінка алгоритму приблизно така сама, як і в алгоритму сортування включенням.

Програмну реалізацію цього алгоритму сортування зображено на рис(2.1)

```

const swap = (items: CoinType[], firstIndex: number, secondIndex: number) => {
  const temp = items[firstIndex];
  items[firstIndex] = items[secondIndex];
  items[secondIndex] = temp;
};

const partition = (arr: CoinType[], left: number, right: number) => {
  let center: number =
    +arr[Math.floor((right + left) / 2)]["1d"]["price_change"],
    i = left,
    j = right;
  while (i <= j) {
    while (+arr[i]["1d"]["price_change"] < center) {
      i++;
    }
    while (+arr[j]["1d"]["price_change"] > center) {
      j--;
    }
    if (i <= j) {
      swap(arr, i, j);
      i++;
      j--;
    }
  }
  return i;
};

let quickCount = 0;
const quickSortStart = (arr: CoinType[], left: number, right: number) => {
  let index;
  quickCount++;
  if (arr.length > 1) {
    index = partition(arr, left, right);
    if (left < index - 1) {
      quickSort(arr, left, index - 1);
    }
    if (index < right) {
      quickSort(arr, index, right);
    }
  }
  return arr;
};

export const quickSort = (arr: CoinType[], left: number, right: number) => {
  const sortedData = quickSortStart(arr, left, right);
  return { sortedData, quickCount };
};

```

Рисунок 2.1 – програмна реалізація алгоритму швидкого сортування [4]

Shell sort: це алгоритм сортування, що є узагальненням сортування включенням (Сортування включенням — простий алгоритм сортування на основі порівнянь елементів. На великих масивах є значно менш ефективним за такі алгоритми, як швидке сортування, сортування злиттям. Однак, має цілу низку переваг:

- простота у реалізації.
- простота відлагодження.
- ефективний (зазвичай) на маленьких масивах.
- ефективний при сортуванні масивів, дані в яких вже непогано відсортовані: продуктивність рівна $O(n + d)$, де d — кількість інверсій).

Алгоритм сортування включенням є ефективним для майже впорядкованих масивів, та неефективним тому що переміщує елемент тільки на одну позицію за раз. Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що розташовані на різній відстані один від одного. Сортування Шелла все ще не є стабільним, але для початкових алгоритмів від чудово підходить, через простоту реалізацію, та доволі просте відлагодження.

Алгоритм роботи: На кожному кроці алгоритму вибирається один з елементів початкового масиву і вставляється на потрібну позицію у вже відсортованому списку. Ці дії будуть виконуватись до того часу, поки початковий масив не буде вичерпано.

Метод вибору чергового елементу з початкового масиву довільний; може використовуватися практично будь-який алгоритм вибору. Часто (з метою отримання стійкого алгоритму сортування), елементи вставляються за порядком їх появи у вхідному масиві.

Програмна реалізація зображена на (рисунок 2.2)

```
export const shellSort = (
  arr: CoinType[]
): { sortedData: CoinType[]; shellCount: number } => {
  let shellCount = 0;
  let n = arr.length;
  for (let gap = Math.floor(n / 2); gap > 0; gap = Math.floor(gap / 2)) {
    for (let i = gap; i < n; i += 1) {
      let temp = JSON.parse(JSON.stringify(arr[i]));
      let j;
      for (j = i; j >= gap && +arr[j - gap].price < +temp.price; j -= gap) {
        shellCount++;
        arr[j] = JSON.parse(JSON.stringify(arr[j - gap]));
      }
      arr[j] = JSON.parse(JSON.stringify(temp));
    }
  }

  return { sortedData: arr, shellCount };
};
```

Рисунок 2.2 – програмна реалізація алгоритму сортування Шелла

Hashing: у курсовій роботі хешування використовується для завантаження даних по монетам у Json форматі, хешування виконує роль конвертації з Object формату (Рисунок 2.3) до Json (Рисунок 2.4)

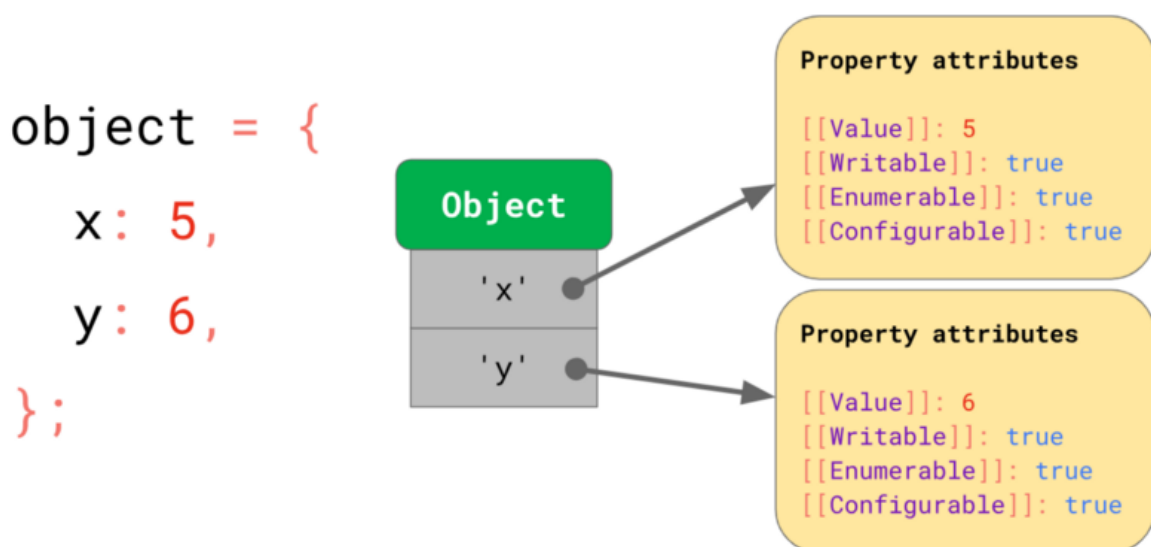
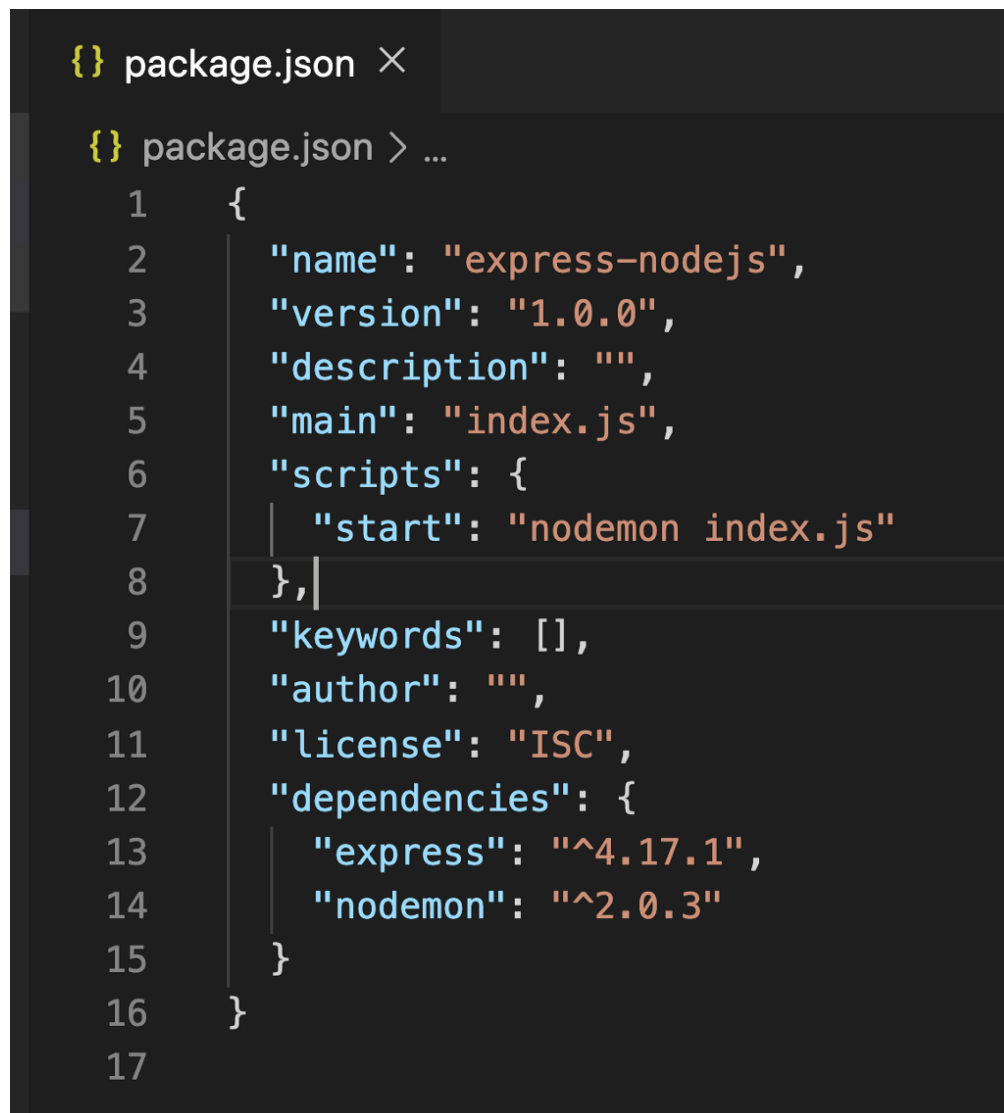


Рисунок 2.3 – приклад JavaScript об'єкта [4]

Об'єкт у мові програмування JavaScript використовується для зберігання даних, та швидкого доступу до них, так як доступ відбувається за ключем. Також JavaScript дає можливості перебору масиву з допомогою методу `Object.keys(obj)`, який повертає ключі, і з допомогою цього масиву, ітеруючи кожен ключ, можна перебирати всі елементи об'єкта.



```
{} package.json ×
{} package.json > ...
1  {
2    "name": "express-nodejs",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "express": "^4.17.1",
14     "nodemon": "^2.0.3"
15   }
16 }
17
```

Рисунок 2.4 – приклад Json формату

Хешування – це перетворення вхідного масиву або об'єкта даних довільної довжини у вихідну структуру даних фіксованої довжини. Такі перетворення також називаються хеш-функціями.

У веб додатках хешування використовується для перетворення даних в один формат при відправці іншому клієнту або серверу, щоб зменшити кількість помилок, через невідповідність формату. Залишилось 2 види хешів у вебі, це JSON, та XML, останній не дуже користується популярністю, та вже доживає свої роки. Програмна реалізація хешування зображена на Рисунок 2.5. [3]

```
export const downloadData = (
  exportObj: CoinType[] | null,
  exportName: string
) => {
  if (!exportObj) {
    console.log("no data provided");
  }
  const reducedCoins = exportObj
    ?.map(({ name, id, logo_url, price, rank, status, ...rest }) => ({
      name,
      id,
      logo_url,
      price,
      rank,
      status,
      "id": {
        price_change: rest["id"].price_change,
      },
    }))
    .reduce((acc, item) => {
      const { id, ...rest } = item;
      acc[id] = { ...rest };
      return acc;
    }, {} as ExportedCoinType);

  const dataJson =
    "data:text/json;charset=utf-8," +
    encodeURIComponent(JSON.stringify(reducedCoins));

  const downloadTriggerNode = document.createElement("a");
  downloadTriggerNode.setAttribute("href", dataJson);
  downloadTriggerNode.setAttribute("download", exportName + ".json");
  document.body.appendChild(downloadTriggerNode); // required for firefox
  downloadTriggerNode.click();
  downloadTriggerNode.remove();
};
```

Рисунок 2.5 – програмна реалізація скачування та хешування даних у додатку

Object (Рисунок 2.6): У JavaScript містить два типи властивостей:

властивість-значення та властивість-акцесор (властивість, обернута в геттер та сеттер).

Конструктор `Object` створює об'єкт-обгортку для переданого значення. Якщо значенням є `null` або `undefined`, створює і повертає порожній об'єкт, в іншому випадку повертає об'єкт такого типу, який відповідає переданому значенню.

Всі об'єкти в JavaScript підтримують властивості і методи, які можуть додаватися і видалятися під час виконання. Ці властивості і методи можуть мати будь-які імена і визначатися числами. Якщо ім'я властивості або методу є простим ідентифікатором, воно може бути зазначено через точку після імені об'єкта, наприклад `myObj.name`, `myObj.age` і `myObj.getAge`.

HashMap: заснований на хеш-таблицях, реалізує інтерфейс `Map` (що передбачає зберігання даних у вигляді пар ключ/значення). Ключі та значення можуть бути будь-яких типів, у тому числі й `null`. Ця реалізація не дає гарантій щодо порядку елементів з часом. Дозвіл колізій здійснюється за допомогою методу ланцюжків. При додаванні елемента послідовність кроків наступна:

1. Спочатку ключ перевіряється на рівність `null`.
2. Якщо ця перевірка повернула `true`, буде викликаний метод `putForNullKey(value)`.
3. Далі генерується хеш на основі ключа.
4. За допомогою методу `indexFor(hash, tableLength)` визначається позиція в масиві, куди буде поміщений елемент. Тепер, знаючи індекс у масиві, ми отримуємо список (ланцюжок) елементів, прив'язаних до цього осередку.
5. Хеш і ключ нового елемента по черзі порівнюються з хешами та ключами елементів зі списку та, при збігу цих параметрів, значення елемента перезаписується.

6. Якщо попередній крок не виявив збігів, буде викликаний метод `addEntry(hash, key, value, index)` для додавання нового елемента.

Назви методів вигадані, для пояснення роботи HashMap.

Пропускається, що ключ не дорівнює `null`, тоді порядок дій буде такий:

1. Визначення позиції у масиві
2. Подібних елементів не знайдено
3. Додавання елемента

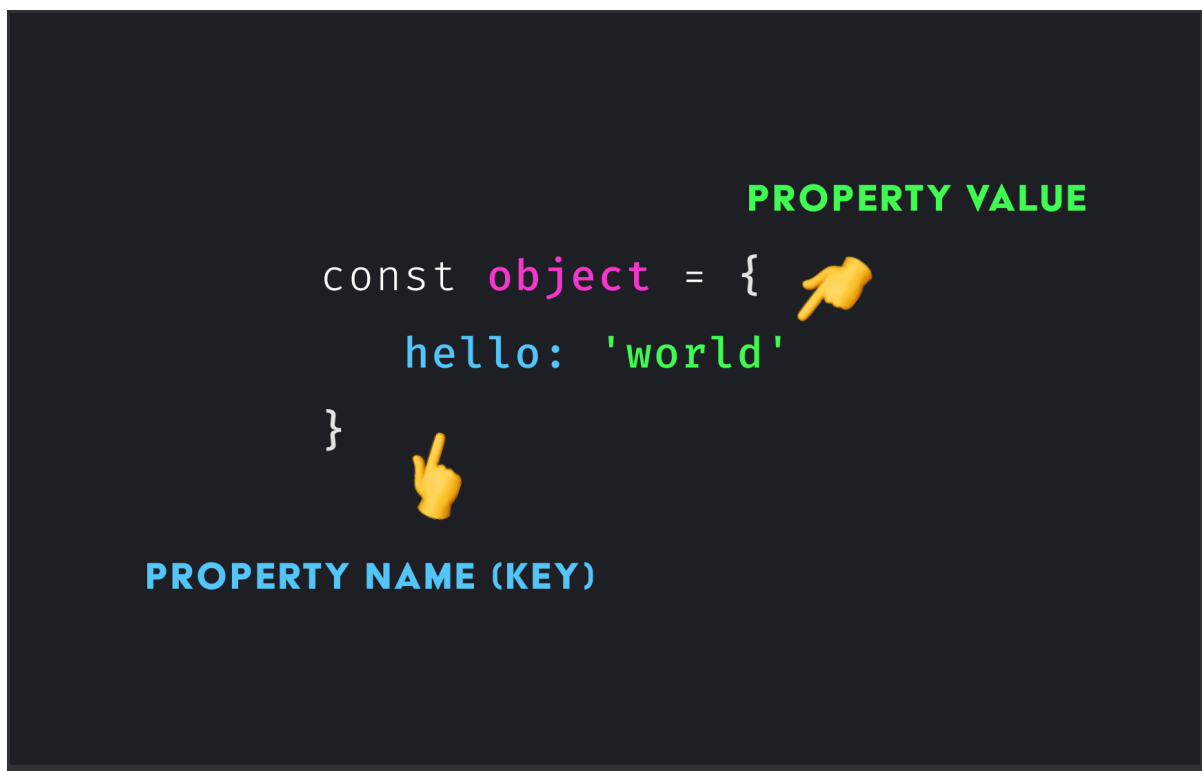


Рисунок 2.6 – Приклад об'єкту / HashMap з JavaScript

Array: Масиви є спископодібними об'єктами, чиї прототипи містять методи для операцій обходу змінних масиву.

Ні розмір JavaScript масиву, ні типи його елементів не є фіксованими. Оскільки розмір масиву може збільшуватися і зменшуватися в будь-який час, то немає гарантії, що масив виявиться щільним. Тобто, при роботі з масивом може виникнути ситуація, що елемент масиву, до якого ви звернетесь, буде порожнім і поверне `undefined`. В цілому, це зручна характеристика але якщо ця особливість

масиву небажана в вашому специфічному випадку, ви можете розглянути можливість використання типізованих масивів, така можливість є у мові програмування TypeScript основаної на JavaScript.

Для доступу до окремих елементів створеного масиву використовуються квадратні дужки "[]". Масиви в JavaScript індексуються з нуля: перший елемент масиву має індекс, що дорівнює 0, а індекс останнього елемента дорівнює значенню `length-1`. У JavaScript елементу масиву можна присвоювати будь який тип даних: рядок, число, `Object`, `Date`, `Function`, `Array`. Для проходження масиву використовують цикли. Зазвичай це `for` але також використовують спеціальні цикли `for...in`, `for...of`.

`Stack` – це колекція, елементи якої отримують за принципом "останній увійшов, перший вийшов" (`Last-In-First-Out`, або `LIFO`). Це означає, що ми матимемо доступ лише до останнього доданого елемента.

На відміну від списків ми не можемо отримати доступ до довільного елемента стека. Ми можемо лише додавати або видаляти елементи за допомогою спеціальних методів. У стека немає також методу `Contains`, як у списків. Крім того, стек не має ітератора.

Найчастіше зустрічається аналогія пояснення стека — стос тарілок. Незалежно від того, скільки тарілок у стосі, ми завжди можемо зняти верхню. Чисті тарілки так само кладуться на верх стопки, і ми завжди будемо першою брати ту тарілку, яка була покладена останньою. Якщо ми покладемо, наприклад, червону тарілку, потім синю, а потім зелену, спочатку треба буде зняти зелену, потім синю, і, нарешті, червону. Головне, що треба запам'ятати – тарілки завжди ставляться і на гору стопки. Коли хтось бере тарілку, він також знімає її згори. Виходить, що тарілки розуміються на порядку, зворотному тому, в якому ставилися.

Операція додавання елемента у стек називається "`push`", видалення – "`pop`". Останній доданий елемент називається верхівкою стека, або `top`, і його можна

подивитися за допомогою операції `peek`. Програмна реалізація зображена на рисунку 2.7

```

1  public class Stack
2  {
3      LinkedList _items = new LinkedList();
4
5      public void Push(T value)
6      {
7          throw new NotImplementedException();
8      }
9
10     public T Pop()
11     {
12         throw new NotImplementedException();
13     }
14
15     public T Peek()
16     {
17         throw new NotImplementedException();
18     }
19
20     public int Count
21     {
22         get;
23     }
24 }

```

Рисунок 2.7 – Приклад реалізації структури даних Stack з JavaScript

2.2 Особливості загального алгоритму дашборду по моніторингу криптовалют

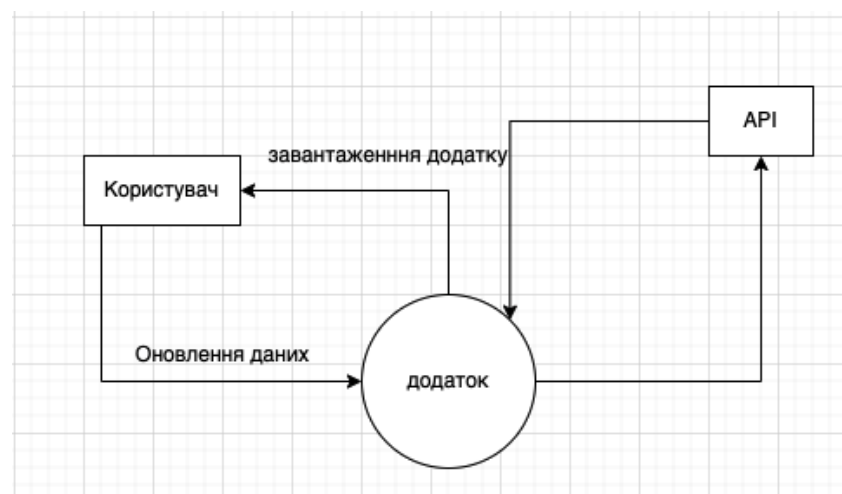


Рисунок 2.8 – Схема взаємодії користувач - додаток - API

Загальний алгоритм програми поділений на кілька модулів: отримання інформації з API, модифікація даних (підготовка їх до виводу), вивід даних, а також на окремий модуль, який відповідає за взаємодію клієнтської частини та серверу – інтерфейс користувач-сервер (Рисунок 2.8). Під час запуску додатку клієнт буде бачити відразу головну область, авторизація, та кабінет юзера відсутні у додатку, тому що додаток не має REST зв'язку з біржею, для створення запитів, на купівлю/продаж монет, тому і кабінет користувача не має сенсу. При запуску додатку буде виконуватись запит до стороннього API, щоб заповнити додаток даними, під час запиту користувачу буде показуватись Loader (Рисунок 2.9)



Рисунок 2.9 – Приклад Loader

Відразу після завантаження даних вони будуть встановленні у стейт додатку для їх відображення, та React помітить зміну даних, і згенерує список монет, для показу їх користувачу. Модуль взаємодії клієнт-сервер реалізований таким чином: при оновленні даних буде виконуватись запит до сервера, для оновлення даних у додатку. За допомогою швидкого сортування, з усіх монет є можливість вибрати ті, які за зміною ціни будуть задовольняти користувача. Даний алгоритм було обрано так як він швидко справляється з сортуванням. Він є одночасно одним з найшвидших і найпростішим в реалізації алгоритмом. В

свою чергу алгоритм сортування Шелла сортує всі монети за кількістю алфавітом. До інших елементів на головній сторінці можна виділити наявність кнопок для оновлення списку монет та завантаження даних у форматі JSON, та елементів Select для вибору сортування.

2.3 Базові алгоритми для розробки дашборду по моніторингу криптовалют.

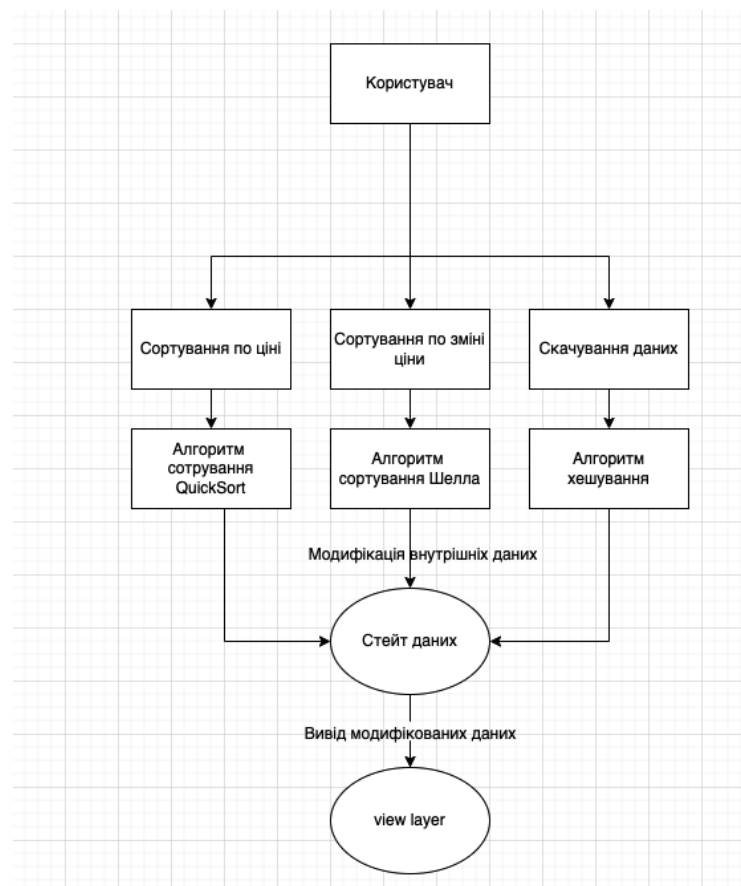


Рисунок 2.10 – Модифікація даних користувачем, через алгоритми

При розробці дашборду було використано алгоритм хешування та 2 алгоритми сортування:

- Quick Sort
- Shell Sort

Алгоритми сортування використовуються в робочій області, для

сортування списку монет, по різних критеріям, та детально протестовано, на валідність роботи. Алгоритм хешування використовується для скачування даних, отриманих при запиті до стороннього API, на пристрій користувача для зручності перегляду, та зменшення розміру даних.

Також були використані такі структури даних, як:

- Object
- HashMap
- Array
- Stack

Object використовується при зберіганні усього стейту додатку, тобто усі дані з API, застосовані методи сортування та монета, яку вибрав користувач, HashMap використовується при переведенні даних у JSON, для завантаження, щоб спростити mapping масива монет, Stack використовується для додавання нової монети до списку монет, якщо прийшли відповідні дані з API, та Array використовується для зберігання списку монет, та маніпуляцій з ними, таких як сортування та оновлення.

Всі алгоритми та структури даних використані під час розробки дашборду були описані у 1-му підрозділі, 2-го розділу.

Отже в цьому розділі було розглянуто загальний алгоритм програмного застосунку, а також загальні інтерфейсу і те, як з їхньою допомогою можна запросити дані з віддаленого сервера. Власне ця інтерактивність зробила можливим створення інтерфейсу користувач-сервер. Також було описано базові алгоритми, які будуть використовуватись у додатку, для досягнення взаємодії користувач - додаток. Було приведено приклади програмних реалізацій усіх алгоритмів, та ознайомлення з їх теоретичною частиною, аналогічна робота була проведена зі структурами даних.

3 РОЗРОБКА ДІАГРАМ КЛАСІВ, ОБ'ЄКТІВ ТА СТАНУ

3.1 Реалізація програмного продукту

При вході в додаток перед користувачем з'являється робоча область (рисунок 3.1). Якщо ж користувач вперше знайомиться із застосунком він може перейти на документацію до продукту та ознайомитись з можливостями додатку.

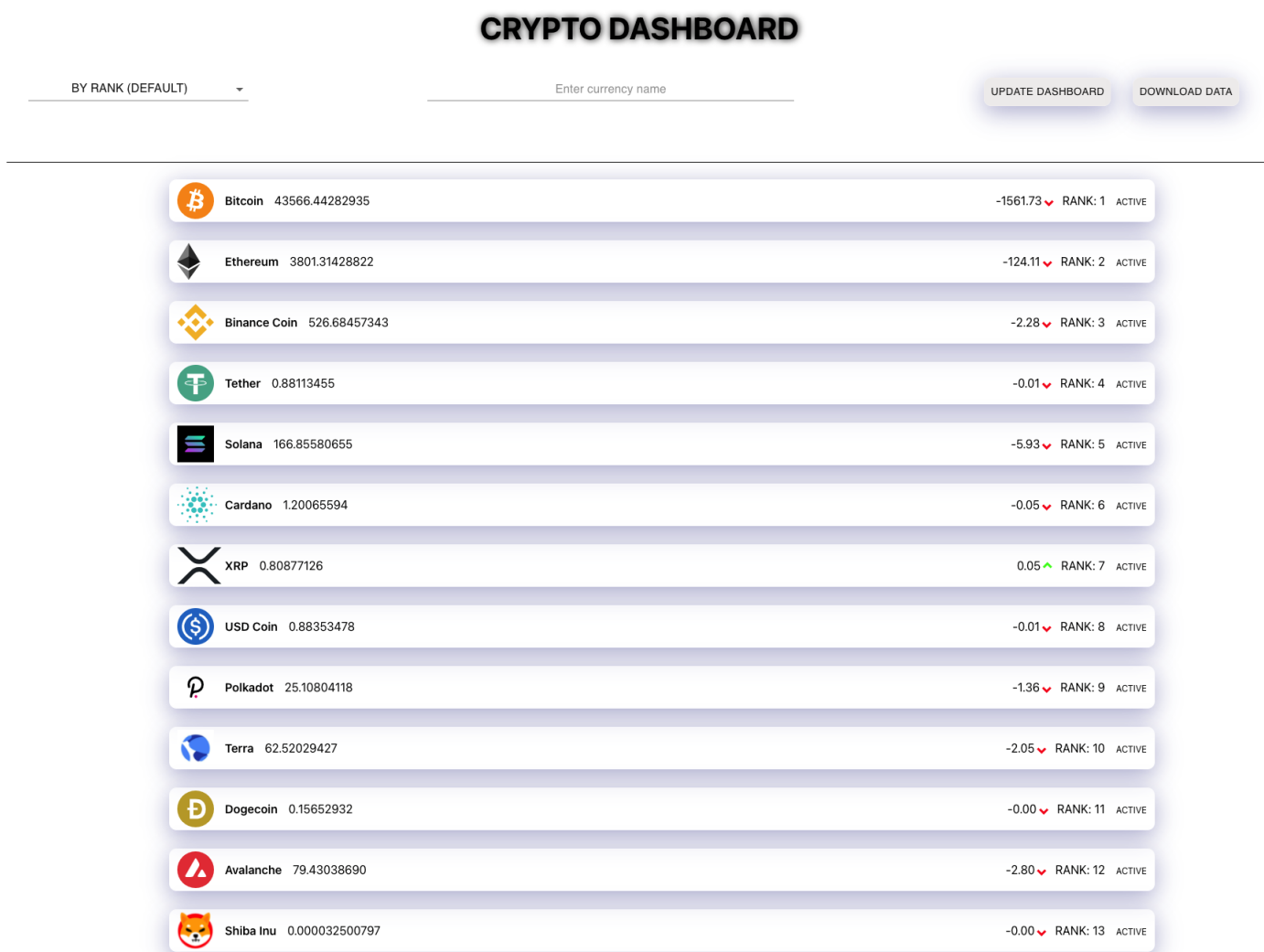


Рисунок 3.1 – Зображення робочої області

На робочій області користувач бачить селект для вибору метода сортування, селект реалізований як дропдаун, або ж випадаюче меню (Рисунок 3.2).

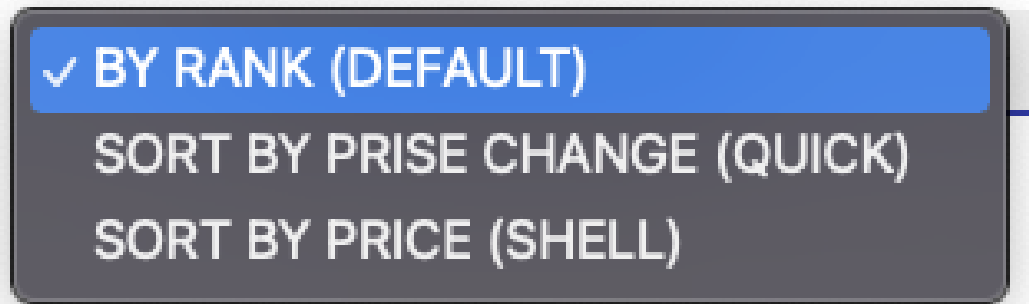


Рисунок 3.2 – Зображення dropDown меню для сортування

Також на робочій області представлено інпут для вводу назви монети, та пошуку її у списку всіх монет (Рисунок 3.3), на прикладі ввели пошуковий запит за ключовим словом “bitcoin”, регістр букв не враховується, для забезпечення дружнього інтерфейсу.

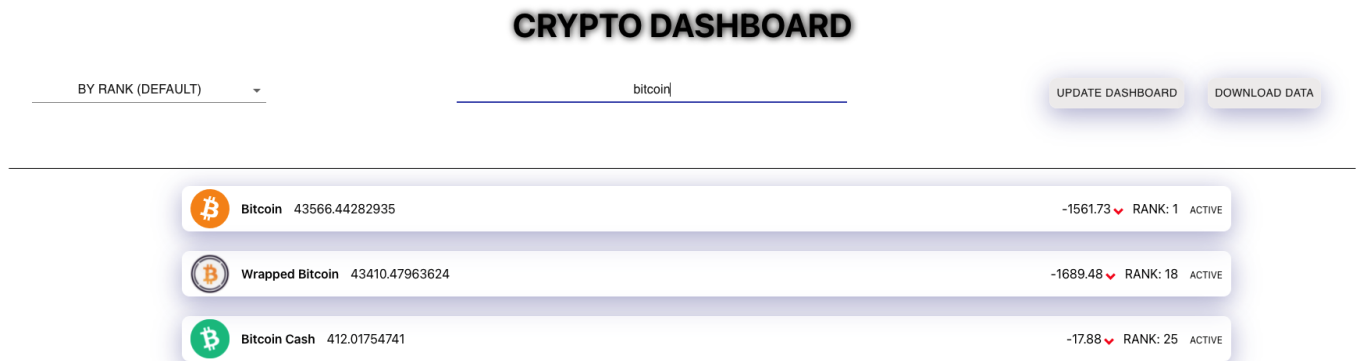


Рисунок 3.3 – Зображення приклад пошуку за ключовим словом

Справа від інпуту для пошуку розташовані кнопки “update dashboard” & “download data” (Рисунок 3.4), які відповідають відповідно за оновлення даних, та за їх скачування у форматі JSON. Програмну реалізацію роботи кнопки “Update Dashboard” продемонстровано на Рисунку 3.8



Рисунок 3.4 – Зображення кнопок “update dashboard” & “download data”, та hover ефекту на них

Через розділювальну лінію від елементів керування виводиться список всіх монет, які повертає сервер (Рисунок 3.5), які початково сортуються за рангом, який їм ставить API. Також при виводі елементів списку використовується така технологія, як lazyLoad, що дозволяє завантажувати тільки ті елементи, які бачить користувач, та динамічно підвантажувати решту елементів.



















	Bitcoin	43566.44282935	-1561.73 ▼	RANK: 1	ACTIVE
	Ethereum	3801.31428822	-124.11 ▼	RANK: 2	ACTIVE
	Binance Coin	526.68457343	-2.28 ▼	RANK: 3	ACTIVE
	Tether	0.88113455	-0.01 ▼	RANK: 4	ACTIVE
	Solana	166.85580655	-5.93 ▼	RANK: 5	ACTIVE
	Cardano	1.20065594	-0.05 ▼	RANK: 6	ACTIVE
	XRP	0.80877126	0.05 ▲	RANK: 7	ACTIVE
	USD Coin	0.88353478	-0.01 ▼	RANK: 8	ACTIVE
	Polkadot	25.10804118	-1.36 ▼	RANK: 9	ACTIVE
	Terra	62.52029427	-2.05 ▼	RANK: 10	ACTIVE
	Dogecoin	0.15652932	-0.00 ▼	RANK: 11	ACTIVE
	Avalanche	79.43038690	-2.80 ▼	RANK: 12	ACTIVE
	Shiba Inu	0.000032500797	-0.00 ▼	RANK: 13	ACTIVE
	HEX	0.11090845	-0.01 ▼	RANK: 14	ACTIVE
	Polygon	1.98229915	-0.28 ▼	RANK: 15	ACTIVE
	Crypto.com Chain	0.52686270	-0.02 ▼	RANK: 16	ACTIVE
	Binance USD	0.87981128	-0.01 ▼	RANK: 17	ACTIVE
	Wrapped Bitcoin	43410.47963624	-1689.48 ▼	RANK: 18	ACTIVE

Рисунок 3.5 – Зображення списку елементів

Варто зазначити, що при наведенні на елемент, на ньому застосовується hover ефект, який приближує, та виділяє задній фон вибраного елемента (Рисунок 3.6). Це було реалізовано для забезпечення Friendly User Interface, щоб користувач чіткіше бачив елемент, який його зацікавив.



Рисунок 3.6 – Зображення hover ефекту на елементу списку

Сам елемент списку складається з таких елементів (Рисунок 3.7):

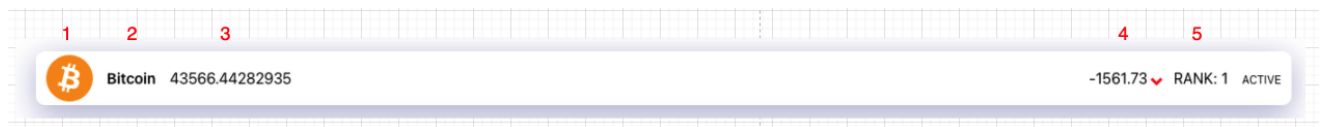


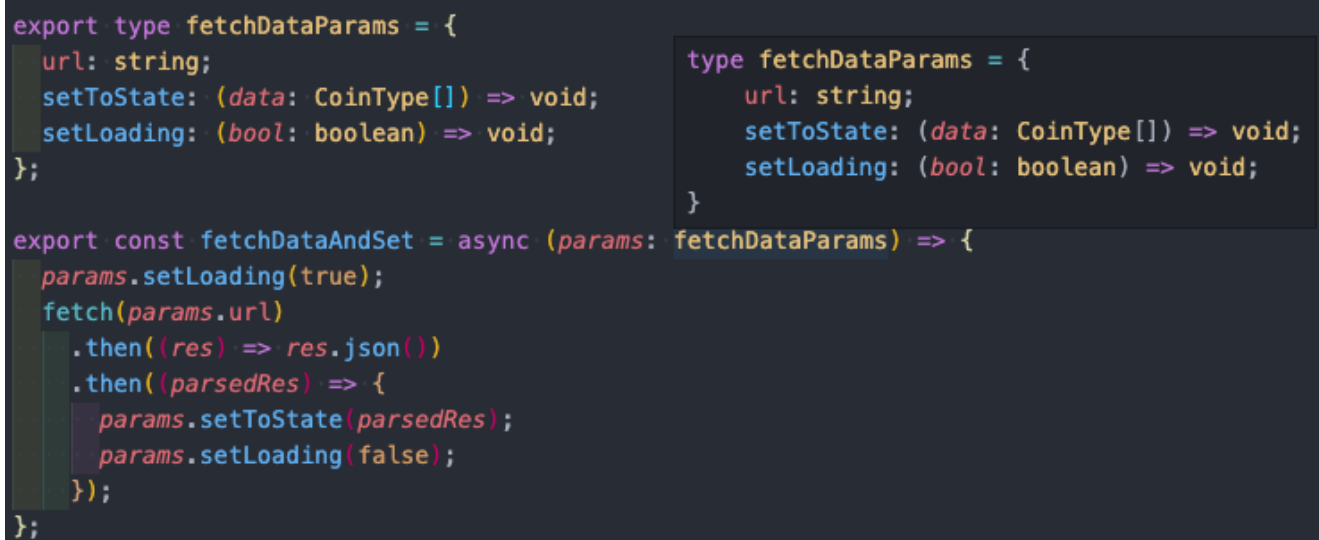
Рисунок 3.7 – Частини елементу списку

1. Логотип монети;
2. Назва монети;
3. Ціна криптовалюти, на момент останнього оновлення;
4. Зміна ціни за останні 24 години;
5. Ранг монети, який їй надало API;

3.2 Реалізація вхідного та вихідного інтерфейсів

Вхідний інтерфейс ґрунтується на поведінці користувача, тобто він може взаємодіяти з якими елементами, і певним чином будуть відображатись зміни з даними. Також додаток реалізує вихідний інтерфейс, який ґрунтується на

взаємодії клієнт-сервер (Рисунок 3.8) [6]. Він реалізується змогою додатку оновлювати дані, по натисканні на кнопку “Update dashboard” (Рисунок 3.6).



```
export type fetchDataParams = {
  url: string;
  setToState: (data: CoinType[]) => void;
  setLoading: (bool: boolean) => void;
};

export const fetchDataAndSet = async (params: fetchDataParams) => {
  params.setLoading(true);
  fetch(params.url)
    .then((res) => res.json())
    .then((parsedRes) => {
      params.setToState(parsedRes);
      params.setLoading(false);
    });
};
```

Рисунок 3.8 – лістинг коду, для взаємодії клієнт-сервер

Функція `fetchDataAndSet` приймає як параметри функції які керують стейтом додатку, `setLoading` – встановлює поле `isLoading` в положення `true`, тим самим проковує рендер спінера, який показує користувачу, що додаток завантажує якусь інформацію. `setToState` – встановлює поле `isLoading` в положення `true`, тим самим проковує рендер спінера, який показує користувачу, що додаток завантажує якусь інформацію. Потім `fetch` API виконує запит по `url`, який ми передали параметром у функцію, та встановлює дані у стейт і переводить поле `isLoading` в положення `false`, тим самим забираючи ладер з робочої області, та показ елементів.

До прикладів вхідного інтерфейсу можна описати пошук елементу в списку. Спершу користувач вводить у якості підстроки назву, або частину назви) монети яку він шукає у інпуті (Рисунок 3.3), та цим самим проковує роботу алгоритму пошуку монети, який працює методом фільтрації всіх елементів, назва яких не містить введену частину назви, програмну реалізацію алгоритму зображено на рисунку 3.9 та 3.10. Сортування також слугує прикладом вхідного

інтерфейсу, його реалізація описується вище.

```
const InputCmp = (props: {
  inputValue: string;
  setInputValue: (val: string) => void;
}) => {
  return (
    <div className="input_wrapper">
      <Input
        className="input"
        placeholder="Enter currency name"
        value={props.inputValue}
        onChange={(e) => props.setInputValue(e.target.value)}
      />
    </div>
  );
};

const mapStateToProps = (state: StateType) => ({
  inputValue: state.inputValue,
});

const mapDispatchToProps = (dispatch: Dispatch<Action>) => ({
  setInputValue: (val: string) => dispatch(SetInputValue(val)),
});

export default connect(mapStateToProps, mapDispatchToProps)(InputCmp);
```

Рисунок 3.9 – лістинг коду, для пошуку елемента у списку(1)

На рисунку 3.9 відбувається рендер компонента `Input`, якому передається обробчик зміни, який на кожен ввід символу у полі вводу буде змінювати вміст поля вводу, який зберігається у внутрішньому стейті, та від якого залежить вивід елементів на Рисунку 3.10.


```

const List = (props: Props) => {
  const filteredList = props.data.filter((el) =>
    el.name.toLowerCase().includes(props.inputValue)
  );
  return (
    <div className="list">
      <ul>
        {filteredList.map((item) => (
          <ListItem listItem={item} key={item.id} />
        ))}
      </ul>
    </div>
  );
};

```

Рисунок 3.10 – лістинг коду, для пошуку елемента у списку(2)

На рисунку 3.10 виводяться всі елементи, які фільтруються відповідно до строки, яка введена у поле вводу. І відповідно до коду, всі елементи, які не містять у собі цю субстроку будуть відфільтровані. Те чим цей алгоритм чудовий, це тим, що внутрішній масив елементів не змінюється, змінюється тільки вивід, подібні конструкції називаються: Conditional render.

3.3 Тестовий приклад. Відлагодження програмного продукту.

Написання юніт тестів відбувалось на JS фреймворку для тестування Jest, скріншот проходження всіх тестів, рисунок 3.11. Тестами було покрито функції сортування ShellSort та QuickSort.

Для чого взагалі використовується юніт тести, вони дуже зручні, при подальшій підтримці написаного коду, коли потрібно від рефакторити вже написаний код, без зміни функціоналу, і юніт тестами дуже легко перевірити валідність функціоналу. Також є багато фреймворків для снєпшотування компонентів, що дозволяє навіть перевіряти зміну візуалу, а не тільки функціоналу певних функцій[8].

```

> crypto-krsch@0.1.0 jest
> jest

PASS src/helpers/test.test.ts
  test quck sort
    ✓ fn is defined (1 ms)
    ✓ sorting is working properly (2 ms)
  test shell sort
    ✓ fn is defined
    ✓ sorting is working properly

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:  0 total
Time:        4.618 s
Ran all test suites.
Kucher-mx@MacBook-Pro crypto-krsch %

```

Рисунок 3.11 – скріншот проходження всіх юніт тестів

```

describe('test quck sort', () => {
  it('fn is defined', () => {
    expect(quickSort).toBeDefined();
  });

  it('sorting is working properly', () => {
    expect(quickSort(quickSortMockArray, 0, quickSortMockArray.length - 1).sortedData).toEqual(
      quickSortMockArrayResult.reverse(),
    );
  });
});

describe('test shell sort', () => {
  it('fn is defined', () => {
    expect(shellSort).toBeDefined();
  });

  it('sorting is working properly', () => {
    expect(shellSort(shellSortMockArray).sortedData).toEqual(shellSortMockArrayResult);
  });
});

```

Рисунок 3.12 – готові юніт тести, на QuickSort & ShellSort [10]

На рисунку 3.12 зображено готові юніт тести на функції для швидкого сортування та сортування Шелла, перший describe описує блок тестів, які будуть відноситись до функції швидкого сортування.

Тести які називаються “fn is defined”, перевіряють чи дана функція існує,

та чи її можна викликати. другий тест, наприклад “`sorting is working properly`” перевіряє валідність роботи алгоритми сортування, на замоканих даних (замокани дані – константні дані, які використовуються для тестування).

`ShellSortMockArray` & `ShellSortMockArrayResult` це відповідно замokаний масив даних, та вже відсортований масив, для перевірки правильності сортування, програмна реалізація мокання даних зображена на рисунку 3.13.

Jest надає можливість “шпіонажу” над функціями або певними даними, це корисно, тому що роботу і вихідний результат деяких функцій розробник не може контролювати, наприклад бібліотеку якусь, або асинхронний запит, тому використовується практика “мока”, тобто функція буде повертати результат, який вказав розробник, щоб налаштувати правильне тестування додатку.

```

const shellSortMockArray: CoinTypeOne[] = [
  {
    price: 45,
  },
  {
    price: 8,
  },
  {
    price: 13,
  },
  {
    price: 12,
  },
  {
    price: 142,
  },
  {
    price: 6,
  },
];

const shellSortMockArrayResult = [
  {
    price: 142,
  },
  {
    price: 45,
  },
  {
    price: 13,
  },
  {
    price: 12,
  },
  {
    price: 8,
  },
  {
    price: 6,
  },
];

```

Рисунок 3.13 – замокані дані, для функцій сортування QuickSort & ShellSort

Фреймворк jest був обраний тому що це найбільш використовуваний фреймворк для юніт тестування JS коду, який має детальну документацію, та багато source коду, по якому можна дізнатися Best Practices, також через велике ком'юніті, дуже легко знайти інформацію, по помилкам, тому що в когось ця помилка відтворювалась і скоріш всього, до неї вже є рішення. Також є такі бібліотеки для Jest як **react-test-renderer** або **react-testing-library**, які

використовуються для снєпшотування контенту, щоб було видно, чи є якісь зміни на візуальній частині додатку.

Jest має такий зручний функціонал як `mocking`, це функції шпіони, функціонал яких описується, і буде використовуватись під час виконання оригіналу функції. Це зручно тим, що коли якась функція або компонент залежить від результату функції невідконтрольної розробнику, то це означає, що потрібно замокати цю функцію, що він розумів, що вона повертає, і на основі цього, написав тест кейс.

Отже в цьому розділі було продемонстровано візуальну частину додатку, та реалізацію деяких з елементів керування. Також було описано вхідний та вихідний інтерфейс у додатку, тобто взаємодію типу користувач - додаток, додаток - API, користувач - додаток - API. Та було продемонстровано покриття додатка юніт тестами, для спрощення підтримки розробки дашборду, на додачу до цього було описано трішки теоретичної частини про юніт тестування та тестування програмними засобами в загальному.

ВИСНОВОК

Результати за списком типових завдань після реалізації програмного додатку:

- Проаналізувати доступні методи зберігання даних та алгоритми для взаємодії з ними: було проаналізовано велику кількість різних структур даних та алгоритмів для взаємодії з ними, і в результаті були вибрані найоптимальніші варіанти, для співпраці між собою.
- Сформулювати загальну ідею та ідеологію програмного продукту: у кінці розробки додатку було сформульовано таку ідею: “спростити важке”, та ідеологію чистого та чіткого інтерфейсу зрозумілого всім користувачам.
- Орієнтовно визначити методи зберігання даних та алгоритми, що будуть використовуватись при роботі додатку: було розроблено функціонал фетчингу даних з стороннього API та зберігання їх у внутрішньому стейті.
- Проаналізувати середовище розробки, систему контролю версій та мову програмування: було проаналізовано і вибрано таку мову програмування як JavaScript, середовище розробки VSCode, систему контролю версій Git + GitHub.
- Вибрати доцільні алгоритми для додатку: було сформовано список алгоритмів, для найшвидшої взаємодії з вибраними структурами даних.
- Реалізувати продукт: було реалізовано робочу версію додатку, відповідно до всіх пунктів технічного завдання;
- Перевірити роботу додатку: було протестовано працездатність додатку реальним користувачем, та роботу всіх функцій і алгоритмів було покрито юніт тестами.

Додаток себе чудово зарекомендував у використанні новачками, які недавно увійшли у сферу криптовалют, і отримав чудові відгуки. Проект можливо розвивати у таких напрямках, як монетизація (платні послуги або можливості у додатку), збільшення функціоналу, наприклад удосконалення до

рівня торгової біржі. Також якщо збільшити SEO оптимізацію він підніметься вище у пошукових результатах, та підніметься показник користувачів, через збільшення популярності, та пошукових видач дашборду.

У результаті виконання курсової роботи були отримані такі навички як:

- Робота із IDE VSCode
- Практичне використання мови програмування JavaScript.
- Робота з бібліотекою React та Redux.
- Використання та реалізація різноманітних алгоритмів та структур даних.
- Створення власних алгоритмів.
- Робота з пакетним менеджером npm та терміналом.
- Використання fetch API.
- Використання функціонального програмування.

Завдяки наявності open source code, присутня можливість доопрацювання програми іншими розробниками, через репозиторій на платформі GitHub (посилання: <https://github.com/Kucher-mx/crypto-terminal>).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React docs URL: <https://uk.reactjs.org/> (дата звернення 01.12.2021).
2. Рейтинг популярності мов програмування. URL: <https://habr.com/ru/post/543346/> (дата звернення 15.11.2021).
3. Мартин Роберт. Чистий код. Створення, аналіз і рефакторинг, 2017, 1208 р.
4. Фленаган Девід. Javascript: The Definitive Guide, 5th edition, 2015, 1008 р.
5. NPM: <https://www.npmjs.com/> (дата звернення 05.11.2021).
6. JS матеріали. URL: <https://learn.javascript.ru/> (дата звернення 04.12.2021).
7. FullStack React URL: <https://gumroad.com/a/934179955> (дата звернення 17.11.2021).в
8. StackOverFlow URL: <https://ru.stackoverflow.com/> (дата звернення 20.11.2021).
9. Рейтинг популярності фреймворків. URL: <https://habr.com/ru/company/rvuds/blog/519478/> (дата звернення 18.11.2021).
10. Документація фреймворка для unit-testing Jest. URL: <https://jestjs.io/docs> (дата звернення 05.12.2021).

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Файл Package.json

```
{
  "name": "crypto-krsch",
  "version": "0.1.0",
  "private": true,
  "homepage": "http://Kucher-mx.github.io/ASD_krsch",
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^1.2.36",
    "@fortawesome/free-solid-svg-icons": "^5.15.4",
    "@fortawesome/react-fontawesome": "^0.1.15",
    "@material-ui/core": "^4.12.3",
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "@types/jest": "^26.0.15",
    "@types/node": "^12.0.0",
    "@types/react": "^17.0.0",
    "@types/react-dom": "^17.0.0",
    "gh-pages": "^3.2.3",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-loader-spinner": "^4.0.0",
    "react-redux": "^7.2.5",
    "react-scripts": "4.0.3",
    "redux": "^4.1.1",
    "typescript": "^4.1.2",
    "web-vitals": "^1.0.1"
  },
  "scripts": {
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build",
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Файл tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": [
    "src"
  ]
}
```

Файл index.tsx

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
import { createStore } from "redux";
import rootReducer from "./redux/reducer";
import { Provider } from "react-redux";

let store = createStore(rootReducer);

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

Файл app.tsx

```
import React, { Dispatch, useEffect } from "react";

import { connect } from "react-redux";
import { setDataAction, setLoading } from "./redux/actions";
import { Action, CoinType, StateType } from "./redux/types/reducer.types";
import Loader from "react-loader-spinner";

import "react-loader-spinner/dist/loader/css/react-spinner-loader.css";
import "./App.css";
import List from "./components/List/List.component";
import Header from "./components/Header/Header.component";
import { fetchDataAndSet } from "./helpers/helpers";

type Props = {
  data: CoinType[];
  inputValue: string;
  isLoading: boolean;
  setLoading: (loading: boolean) => void;
```

```

    setData: (data: CoinType[]) => void;
  };

const App = (props: Props) => {
  useEffect(() => {
    fetchDataAndSet({
      url:
        "https://api.nomics.com/v1/currencies/ticker?key=1e911955aee02245962e1da42d7edb1528a05475&&interval=1d
        ,30d&convert=EUR&attributes=id,name,logo_url,price,price_timestamp&page=1&per-page=50",
      setState: props.setData,
      setLoading: props.setLoading,
    });
  }, []);
  return (
    <div className="app">
      {props.isLoading ? (
        <Loader type="ThreeDots" color="#000000" height={300} width={500} />
      ) : (
        <>
          <Header />
          <List data={props.data} inputValue={props.inputValue} />
        </>
      )}
    </div>
  );
};

const MapStateToProps = (state: StateType) => {
  return {
    data: state.data,
    inputValue: state.inputValue,
    isLoading: state.loading,
  };
};

const MapDispatchToProps = (dispatch: Dispatch<Action>) => {
  return {
    setData: (data: CoinType[]) => dispatch(setDataAction(data)),
    setLoading: (loading: boolean) => dispatch(setLoading(loading)),
  };
};

export default connect(MapStateToProps, MapDispatchToProps)(App);

```

Файл app.css

```

.app {
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
  scroll-behavior: smooth;
}

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;
}

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

```

Файл redux/reducer.ts

```
import { types } from "../actionTypes";
import { Action, StateType } from "../types/reducer.types";
const initialState: StateType = {
  data: [],
  inputValue: "",
  selectValue: "",
  loading: true,
};

export default function rootReducer(state = initialState, action: Action) {
  switch (action.type) {
    case types.SET_DATA:
      return { ...state, data: action.payload };
    case types.SET_LOADING:
      return { ...state, loading: action.payload };
    case types.SET_INPUT_VALUE:
      return { ...state, inputValue: action.payload };
    case types.SET_SELECT_VALUE:
      return {
        ...state,
        selectValue: action.payload.value,
      };
    default:
      return state;
  }
}
```

Файл redux/actions.ts

```
import { CoinType } from "../types/reducer.types";
import { types } from "../actionTypes";

export const setDataAction = (data: CoinType[]) => ({
  type: types.SET_DATA,
  payload: data,
});

export const SetInputValue = (value: string) => ({
  type: types.SET_INPUT_VALUE,
  payload: value,
});

export const SetSelectValue = (value: string) => ({
  type: types.SET_SELECT_VALUE,
  payload: value,
});

export const updateDashboard = (data: CoinType[]) => ({
  type: types.UPDATE_DASHBOARD,
  payload: { data },
});

export const setLoading = (loading: boolean) => ({
  type: types.SET_LOADING,
  payload: loading,
});
```

Файл reducer/actionTypes.ts

```
export const types = {
  SET_DATA: "SET_DATA",
  SET_INPUT_VALUE: "SET_INPUT_VALUE",
  SET_SELECT_VALUE: "SET_SELECT_VALUE",
  UPDATE_DASHBOARD: "UPDATE_DASHBOARD",
}
```

```
    SET_LOADING: "SET_LOADING",  
  } as const;
```

Файл reducer/types/reducer.types.ts

```
export type CoinType = {  
  name: string;  
  id: string;  
  logo_url: string;  
  price: number;  
  rank: number;  
  status: string;  
  "1d": {  
    price_change: string;  
  };  
};  
  
export type ExportedCoinType = Record<  
  string,  
  {  
    name: string;  
    logo_url: string;  
    price: number;  
    rank: number;  
    status: string;  
    "1d": {  
      price_change: string;  
    };  
  }  
>;  
  
export type StateType = {  
  data: CoinType[];  
  inputValue: string;  
  selectValue: string;  
  loading: boolean;  
};  
  
export type Action = {  
  type: string;  
  payload: any;  
};
```

Файл helpers/helpers.ts

```
import { CoinType, ExportedCoinType } from "../../redux/types/reducer.types";  
  
export type fetchDataParams = {  
  url: string;  
  setToState: (data: CoinType[]) => void;  
  setLoading: (bool: boolean) => void;  
};  
  
export const fetchDataAndSet = async (params: fetchDataParams) => {  
  params.setLoading(true);  
  fetch(params.url)  
    .then((res) => res.json())  
    .then((parsedRes) => {  
      params.setToState(parsedRes);  
      params.setLoading(false);  
    });  
};  
  
export const downloadData = (  
  exportObj: CoinType[] | null,  
  exportName: string  
) => {  
  if (!exportObj) {
```

```

    console.log("no data provided");
}
const reducedCoins = exportObj
  ?.map(({ name, id, logo_url, price, rank, status, ...rest }) => ({
    name,
    id,
    logo_url,
    price,
    rank,
    status,
    "1d": {
      price_change: rest["1d"].price_change,
    },
  }))
  .reduce((acc, item) => {
    const { id, ...rest } = item;
    acc[id] = { ...rest };
    return acc;
  }, {} as ExportedCoinType);

const dataJson =
  "data:text/json;charset=utf-8," +
  encodeURIComponent(JSON.stringify(reducedCoins));

const downloadTriggerNode = document.createElement("a");
downloadTriggerNode.setAttribute("href", dataJson);
downloadTriggerNode.setAttribute("download", exportName + ".json");
document.body.appendChild(downloadTriggerNode); // required for firefox
downloadTriggerNode.click();
downloadTriggerNode.remove();
};

const swap = (items: CoinType[], firstIndex: number, secondIndex: number) => {
  const temp = items[firstIndex];
  items[firstIndex] = items[secondIndex];
  items[secondIndex] = temp;
};

const partition = (arr: CoinType[], left: number, right: number) => {
  let center: number =
    +arr[Math.floor((right + left) / 2)]["1d"]["price_change"],
    i = left,
    j = right;
  while (i <= j) {
    while (+arr[i]["1d"]["price_change"] < center) {
      i++;
    }
    while (+arr[j]["1d"]["price_change"] > center) {
      j--;
    }
    if (i <= j) {
      swap(arr, i, j);
      i++;
      j--;
    }
  }
  return i;
};

let quickCount = 0;
const quickSortStart = (arr: CoinType[], left: number, right: number) => {
  let index;
  quickCount++;
  if (arr.length > 1) {
    index = partition(arr, left, right);
    if (left < index - 1) {
      quickSort(arr, left, index - 1);
    }
  }

```

```

        if (index < right) {
            quickSort(arr, index, right);
        }
    }
    return arr;
};

export const quickSort = (arr: CoinType[], left: number, right: number) => {
    const sortedData = quickSortStart(arr, left, right);
    return { sortedData, quickCount };
};

export const shellSort = (
    arr: CoinType[]
): { sortedData: CoinType[]; shellCount: number } => {
    let shellCount = 0;
    let n = arr.length;
    for (let gap = Math.floor(n / 2); gap > 0; gap = Math.floor(gap / 2)) {
        for (let i = gap; i < n; i += 1) {
            let temp = JSON.parse(JSON.stringify(arr[i]));
            let j;
            for (j = i; j >= gap && +arr[j - gap].price < +temp.price; j -= gap) {
                shellCount++;
                arr[j] = JSON.parse(JSON.stringify(arr[j - gap]));
            }
            arr[j] = JSON.parse(JSON.stringify(temp));
        }
    }

    return { sortedData: arr, shellCount };
};

```

Файл components/button/button.tsx

```

import React from "react";
import { Button as ButtonMaterial } from "@material-ui/core";

import "./button.css";

type Props = {
    title: string;
    onClickHandler: any;
};

const Button = (props: Props) => {
    return (
        <div>
            <ButtonMaterial onClick={() => props.onClickHandler()}>
                {props.title}
            </ButtonMaterial>
        </div>
    );
};

export default Button;

```

Файл components/header/header.tsx

```

import React, { Dispatch } from "react";
import Input from "../Input/Input.component";
import Logo from "../Logo/Logo.component";
import Select from "../Select/Select.component";
import Button from "../Button/Button.component";

import "./header.css";
import { Action } from "redux";
import {

```



```

    downloadData,
    fetchDataAndSet,
    quickSort,
    shellSort,
  } from "../../helpers/helpers";
import { setDataAction, setLoading } from "../../redux/actions";
import { CoinType, StateType } from "../../redux/types/reducer.types";
import { connect } from "react-redux";

const sortOptions = [
  {
    value: "quick",
    optionName: "sort by prise change (quick)",
  },
  {
    value: "shell",
    optionName: "sort by price (shell)",
  },
];

type Props = {
  data: CoinType[];
  setLoading: (loading: boolean) => void;
  setData: (data: CoinType[]) => void;
};

const Header = (props: Props) => {
  const updateOnClickHandler = () => {
    fetchDataAndSet({
      url:
        "https://api.nomics.com/v1/currencies/ticker?key=1e911955aee02245962e1da42d7edb1528a05475&interval=1d,30d&convert=EUR&attributes=id,name,logo_url,price,price_timestamp&page=1&per-page=50",
      setState: props.setData,
      setLoading: props.setLoading,
    });
  };

  const onChangeHandler = (e: React.ChangeEvent<HTMLSelectElement>) => {
    let performance1, performance2;
    if (e.target.value === "quick") {
      performance1 = performance.now();
      const { sortedData, quickCount } = quickSort(
        props.data,
        0,
        props.data.length - 1
      );
      performance2 = performance.now();
      console.log(
        "%cperformance of quick sort & count of quickSort's iterations: ",
        "color: green",
        performance2 - performance1,
        quickCount
      );

      props.setLoading(true);
      props.setData(sortedData);
      props.setLoading(false);
    } else if (e.target.value === "default") {
      performance1 = performance.now();
      const sortedData = props.data.sort(
        (itemOne, itemTwo) => itemOne.rank - itemTwo.rank
      );
      performance2 = performance.now();
      console.log(
        "%cperformance of default sort: ",
        "color: blue",
        performance2 - performance1
      );
      props.setLoading(true);
    }
  };

```

```

        props.setData(sortedData);
        props.setLoading(false);
    } else {
        performance1 = performance.now();
        const { sortedData, shellCount } = shellSort(props.data);
        performance2 = performance.now();
        console.log(
            "%cperformance of shell's sort & count of shell's iterations: ",
            "color: orange",
            performance2 - performance1,
            shellCount
        );
        props.setLoading(true);
        props.setData(sortedData);
        props.setLoading(false);
    }
};

const downloadDataHandler = () =>
    downloadData(props.data || null, "cryptoDashboardData");
return (
    <div className="header">
        <Logo title="crypto dashboard" />
        <div className="wrapper">
            <Select
                name="sort"
                options={sortOptions}
                onChangeHandler={onChangeHandler}
            />
            <Input />
            <div className="button_group">
                <Button
                    title="Update dashboard"
                    onClickHandler={updateOnClickHandler}
                />
                <Button title="Download data" onClickHandler={downloadDataHandler} />
            </div>
        </div>
    </div>
);
};

const MapStateToProps = (state: StateType) => {
    return {
        data: state.data,
    };
};

const MapDispatchToProps = (dispatch: Dispatch<Action>) => {
    return {
        setData: (data: CoinType[]) => dispatch(setDataAction(data)),
        setLoading: (loading: boolean) => dispatch(setLoading(loading)),
    };
};

export default connect(MapStateToProps, MapDispatchToProps)(Header);

```

Файл components/header/header.css

```

.header {
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    padding: 25px 0;
    margin-bottom: 50px;
    position: relative;
}

```

```

}

.header::after {
  content: '';
  position: absolute;
  bottom: -25%;
  left: 50%;
  transform: translateX(-50%);
  width: 95%;
  border-radius: 10px;
  background-color: #000;
  height: 1px;
}

.wrapper {
  box-sizing: border-box;
  padding: 0 75px;
  margin-top: 15px;
  display: flex;
  justify-content: space-between;
  /* justify-content: center; */
  width: 100%;
}

.button_group {
  display: flex;
}

.button_group button {
  border-bottom: 1px solid #000;
  background: rgba(237, 234, 234, 0.809);
  box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
  backdrop-filter: blur(6px);
  -webkit-backdrop-filter: blur(6px);
  border-radius: 10px;
  border: 1px solid rgba(255, 255, 255, 0.18);
  cursor: pointer;
  transition: 1s all;
  margin: 0 15px;
}

.button_group button:hover {
  background: rgba(24, 203, 223, 0.25);
  box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
  backdrop-filter: blur(6px);
  -webkit-backdrop-filter: blur(6px);
  border-radius: 10px;
  border: 1px solid rgba(0, 0, 0, 0.6);
  transform: scale(1.1);
}

```

Файл components/input/input.tsx

```

import React, { Dispatch } from "react";
import { Input } from "@material-ui/core";

import "./input.css";
import { connect } from "react-redux";
import { Action, StateType } from "../../redux/types/reducer.types";
import { SetInputValue } from "../../redux/actions";

const InputCmp = (props: {
  inputValue: string;
  setInputValue: (val: string) => void;
}) => {
  return (
    <div className="input_wrapper">
      <Input

```

```

        className="input"
        placeholder="Enter currency name"
        value={props.inputValue}
        onChange={(e) => props.setInputValue(e.target.value)}
      />
    </div>
  );
};

const mapStateToProps = (state: StateType) => ({
  inputValue: state.inputValue,
});

const mapDispatchToProps = (dispatch: Dispatch<Action>) => ({
  setInputValue: (val: string) => dispatch(SetInputValue(val)),
});

export default connect(mapStateToProps, mapDispatchToProps)(InputCmp);

```

Файл components/input/input.css

```

.input_wrapper {
  width: 30%;
}

.input {
  width: 100%;
}

.input input {
  text-align: center;
}

```

Файл components/list/list.tsx

```

import React from "react";
import { CoinType } from "../../redux/types/reducer.types";
import ListItem from "../../ListItem/ListItem.component";

import "../List.css";

type Props = {
  data: CoinType[];
  inputValue: string;
};

const List = (props: Props) => {
  const filteredList = props.data.filter((el) =>
    el.name.toLowerCase().includes(props.inputValue)
  );
  return (
    <div className="list">
      <ul>
        {filteredList.map((item) => (
          <ListItem listItem={item} key={item.id} />
        ))}
      </ul>
    </div>
  );
};

export default List;

```

Файл components/list/list.css

```

.list {
  width: 75%;
}

```

Файл components/listItem/listItem.tsx

```
import React from "react";
import { CoinType } from "../../redux/types/reducer.types";
import "./listItem.css";

type Props = {
  listItem: CoinType;
};

const ListItem = (props: Props) => {
  const { logo_url, name, rank, status, price } = props.listItem;
  return (
    <div className="list-item">
      <div className="left">
        <div className="img_wrapper">
          <img src={logo_url} alt="" className="list-item_img" />
        </div>

        <span className="coinParam name">{name}</span>

        <span className="coinParam price">{price}</span>
      </div>

      <div className="right">
        <span className="price-change">
          {parseFloat(props.listItem["1d"].price_change).toFixed(2)}
        </span>
        {parseFloat(props.listItem["1d"].price_change) < 0 ? (
          <i className="arrow down"></i>
        ) : (
          <i className="arrow up"></i>
        )}
        <span className="coinParam rank">rank: {rank}</span>
        <span className="coinParam status">{status}</span>
      </div>
    </div>
  );
};

export default ListItem;
```

Файл components/listItem/listItem.css

```
.list-item {
  width: 100%;
  height: 50px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  border: 1px solid #000;
  margin: 25px 0;
  padding: 3px 10px;
  background: rgba(255, 255, 255, 0.2);
  box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
  backdrop-filter: blur(6px);
  -webkit-backdrop-filter: blur(6px);
  border-radius: 10px;
  border: 1px solid rgba(255, 255, 255, 0.18);
  cursor: zoom-in;
  transition: 1s all;
}

.arrow {
  border: solid black;
  border-width: 0 3px 3px 0;
```

```

        display: inline-block;
        padding: 3px;
    }

    .down {
        transform: rotate(45deg);
        border-color: rgba(247, 20, 20, 0.967);
    }

    .price-change {
        margin-right: 5px;
    }

    .up {
        transform: rotate(-135deg);
        border-color: rgba(58, 247, 20, 0.967);
    }

    .dot {
        height: 25px;
        width: 25px;
        background-color: #000;
        border-radius: 50%;
        display: inline-block;
    }

    .list-item:hover {
        background: rgba(108, 236, 65, 0.25);
        box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
        backdrop-filter: blur(6px);
        -webkit-backdrop-filter: blur(6px);
        border-radius: 10px;
        border: 1px solid rgba(0, 0, 0, 0.6);
        transform: scale(1.1);
    }

    .name {
        font-size: 16px;
        margin-left: 15px;
        font-weight: 600;
    }

    .price {
        font-size: 16px;
        margin-left: 15px;
    }

    .status {
        font-size: 12px;
        margin-left: 15px;
        text-transform: uppercase;
    }

    .rank {
        text-transform: uppercase;
        margin-left: 14px;
    }

    .img_wrapper {
        width: 50px;
        height: 50px;
    }

    .left {
        height: 100%;
        display: flex;
        align-items: center;
    }

```

```
.list-item_img {
  height: 100%;
  object-fit: cover;
}
```

Файл components/logo/logo.tsx

```
import React from "react";
import "../logo.css";

const Logo = (props: { title: string }) => {
  return <h1 className="title">{props.title}</h1>;
};

export default Logo;
```

Файл components/logo/logo.css

```
.title {
  font-size: 42px;
  font-weight: 700;
  text-transform: uppercase;
  text-shadow: rgb(36, 35, 34) 1px 0 10px;
}
```

Файл components/select/select.tsx

```
import React, { Dispatch } from "react";
import { NativeSelect } from "@material-ui/core";

import "../select.css";
import { connect } from "react-redux";
import { SetSelectValue } from "../../redux/actions";
import { Action, StateType } from "../../redux/types/reducer.types";

type Props = {
  name: "sort" | "second";
  options: { value: string; optionName: string }[];
  selectValue: string;
  setSelectValue: (val: string) => void;
  onChangeHandler: (e: React.ChangeEvent<HTMLSelectElement>) => void;
};

const Select = (props: Props) => {
  return (
    <div className="select_wrapper">
      <NativeSelect
        value={props.selectValue}
        onChange={(e) => {
          props.setSelectValue(e.target.value);
          props.onChangeHandler(e);
        }}
        inputProps={{
          name: props.name,
          id: props.name + "_id",
        }}
        className="select"
      >
        <option value="default">by rank (default)</option>
        {props.options.map(({ optionName, value }) => (
          <option value={value} key={value + "_key">
            {optionName}
          </option>
        ))}
      </NativeSelect>
    </div>
```

```
    );  
};  
  
const mapStateToProps = (state: StateType) => ({  
  selectValue: state.selectValue,  
});  
  
const mapDispatchToProps = (dispatch: Dispatch<Action>) => ({  
  setSelectValue: (val: string) => dispatch(SetSelectValue(val)),  
});  
  
export default connect(mapStateToProps, mapDispatchToProps)(Select);
```

Файл components/select/select.css

```
.select_wrapper {  
  width: 18%;  
}  
  
.select {  
  width: 100%;  
}  
  
.select select {  
  border-radius: 5px;  
  width: 100%;  
  text-align: center;  
  text-transform: uppercase;  
}
```