ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

МАГІСТЕРСЬКА ДИПЛОМНА РОБОТА НА ТЕМУ

ОСОБЛИВОСТІ ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНОГО ПІДХОДУ ДО ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУНКІВ

ДОПОВІДАЧ

ІЖИКОВ А.І.

НАУКОВИЙ КЕРІВНИК

ЖЕРЕБЦОВ О.А.

LOG_IN →

00

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

HELLO!

мета : аналіз функціональної парадигми у галузі веб-розробки та дослідження механізмів і можливостей зміни програмної парадигми при інкапсуляції ФП у JavaScript задля покращення якості та продуктивності веб-додатків;

актуальність : функціональне програмування як підхід до створення кодової бази дозволяє створити загальний погляд на проект, що спрощує розробку та підтримку додатку;

наукову новизну роботи становлять : отримані у результаті проведеної роботи формалізовані вимоги до сучасного веб-додатку та обґрунтований набір методик та рекомендацій щодо якісної імплементації функціональної парадигми у веб-програмуванні задля досягнення цільового набору вимог;

практична цінність роботи : заснована на отриманих результатах і полягає у покращенні важливих показників веб-застосунку, зумовлених переводом веб-застосунку з класової до функціональної парадигми та впровадженням сучасних методик та практик;

апробація :

- міжнародна конференція ICISSE (International Conference on Innovative Solutions in Software Engineering), 29-30 листопада 2022 року, м. Івано-Франківськ, с. 162-169.
- XXIV науково-практична студентська конференція у Запорізькому Інституті Економіки та Інформаційних Технологій, 13 грудня 2022 року, м. Запоріжжя, с.72.

ADMIN_MODE_ON_

START_ →

00
: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS
PLEASE, WAIT_
LOADING
ADMIN_MODE_ON_

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULT

парадигма програмування — це набір концепцій, правил та абстракцій, що визначають стиль програмування.

Відповідно до них у кожній парадигмі закладено підхід до використання ключових конструкцій:

- об'єктно-орієнтоване програмування спирається на класи, об'єкти та взаємодію між ними;
- у функціональному програмуванні все тримається на використанні чистих функцій.

ПАРАДИГМА	КЛЮЧОВИЙ КОНЦЕПТ	ПРОГРАМА	РОБОТА ПРОГРАМИ	РЕЗУЛЬТАТ
Об'єктно-орієнтована	Об'єкт та клас	Набір класів та об'єктів	Обмін даними між об'єктами через виклик їх методів	Підсумковий стан об'єктів
Функціональна	Функція	Набір функцій	Обчислення функцій	Підсумкове значення головної функції

ADMIN_MODE_ON_

01

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULT

ПАРАДИГМИ ПРОГРАМУВАННЯ_ | ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ_ | ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ_

Дві найпоширеніші сьогодні парадигми це імперативна та декларативна.

імперативна парадигма визначає рішення так :

→ 1. купити філе курки
→ 2. купити помідори чері
→ 3. купити сіль та соняшникову олію
→ 4. нарізати філе
→ 5. поставити сковорідку на плиту
→ 6. закинути у сковорідку філе курки та помідори
→ 7. налити соняшникову олію та посолити
→ 8. увімкнути плиту на середньому вогні
→ 9. через 40 хвилин вимкнути плиту → 9....

декларативна парадигма :

0. хочу солоне тушковане філе курки з помідорами чері та олією.

У веб-розробці їх представники це ООП, яке стало розвитком імперативного підходу, та функціональне програмування як продовження декларативної моделі.

ADMIN_MODE_ON_

LOADING...

COMPLETED_07 %

NEXT →

Об'єктно-орієнтоване програмування розділяє навколишній світ на об'єкти та класи. Головні ідеї підходу було сформульовано ще у сімдесятих роках.

об'єкт – елементарна сутність, що має властивості (атрибути) та поведінку (методи, або процедури);

клас – тип, шаблон, що визначає структуру, набір атрибутів та методів для об'єктів одного типу - тобто екземплярів класу;

абстракція – використання лише тих характеристик об'єкта, які з достатньою точністю становлять його в даній системі;

поліморфізм – це механізм, що дозволяє використовувати одну й ту саму функцію чи метод для різних типів (класів);

інкапсуляція – механізм, що дозволяє приховувати деякі деталі реалізації усередині класу.

ФП теж про поділ обов'язків, і теж покликане структурувати кодову базу, але трохи по-іншому:
дані → функція1 → дані → функція2 → дані → функція3 → результат.

Алгоритм пошуку користувача у будь-якій соцмережі може виглядати так:
профілі → f1 → спільноти профілів → f2 → спільні спільноти → f3 → кількість учасників → f4 → спільноти більше 1000 → f5 → статистика спільнот → f6 → розбір за демографією → f7 → розбір за віком.

- послідовність неважлива;
- все рознесено окремо, і дані, і їх обробники;
- обробник - це функція, що приймає вхідні дані та повертає результат;
- у функціях прописується, яку інформацію вона має отримати як вхідне значення, а яку має віддати.
- функції можуть бути вкладеними, коли одна функція є аргументом іншої.

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

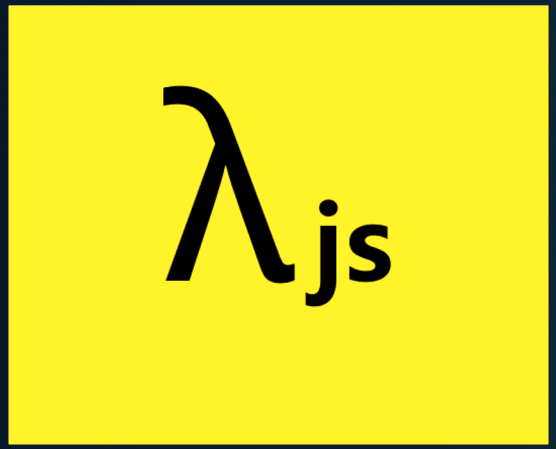
: RESULTS

JavaScript — мультипарадигмова мова програмування.

Це універсальна мова, що дозволяє програмувати у різних стилях та використовувати декілька парадигм.

Недоліком мультипарадигмового підходу є те, що задля забезпечення ефективної роботи з об'єктами та масивами імперативний і об'єктно-орієнтований стилі припускають мутації.

Об'єкти мають бути мінливими, щоб значення їх властивостей могло бути змінено за допомогою методів.



: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

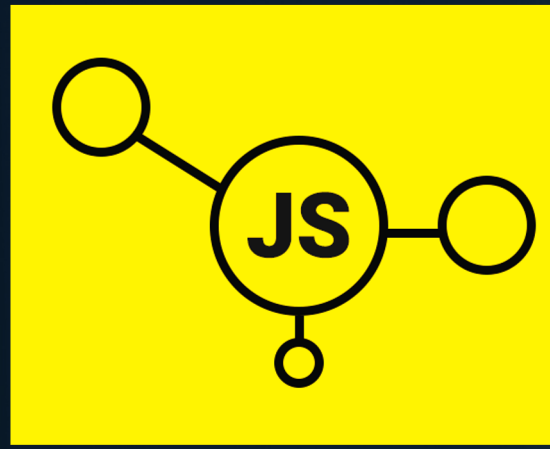
умовно наявне у JS від ФП :

- Функції першого класу
- Анонімні функції та лямбда-синтаксис
- Замикання

умовно немає, або реалізація вважається небезпечною :

- Чистота
- Незмінність, або імутабельність

До списку речей, які притаманні деяким функціональним мовам, і яких умовно немає в JavaScript, частіше за все відносять чистоту та незмінність. Більшість функціональних мов використовують спеціальні структури даних, типу префіксного дерева, де старий та новий об'єкти зберігають посилання на ті самі дані, якщо вони не змінювалися.



: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

JavaScript має чимало прийомів функціонального програмування : використання функції як даних, тобто функції першого класу, уможливлює існування функцій вищого порядку, що, у свою чергу, призводить до появи часткового застосування, карування та композиції.

Функції вищого порядку уможливлюють замикання. А завдяки замиканням з'являється мемоізація.

```
// Функції вищого порядку:

// map, filter, reduce и т.д.
[0, NaN, Infinity].filter( Boolean );
// обіцянки (проміси)
new Promise( ( res ) => setTimeout( res, 100 ) );
// обробники подій
document.addEventListener('keydown',({code, key}) => {
  console.log( code, key );
});



// мемоізація
const memo = (fn, cache = new Map) => (param) => {
  if (!cache.has(param)) {
    cache.set(param, fn(param));
  }
  return cache.get(param);
};
const f = memo((x) => x * Math.sin(1 / x));
f( 0.314 ); // обчислити
f( 0.314 ); // взяти з кешу
```

ADMIN_MODE_ON_

LOADING...

COMPLETED_31 %

NEXT →

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

чисті функції :

чисті функції завжди повертають той самий результат для тих самих параметрів.

Крім того, чисті функції мають прозорість посилань. Це ефект, який дозволяє замість виклику функції підставити результат роботи.

Переваги чистих функцій:

- простіше розібратися, як влаштована функція;
- їх можна запросто кешувати;
- легко тестувати;
- легко розпаралелювати.

найчистіша функція: (() => {})()

```
function impure (min, max) {
  return Math.floor(Math.random() * (max - min + 1)
  + min)
}
impure(1, 10) // 4
impure(1, 10) // 2

function pure (min, max, random = Math.random()) {
  return Math.floor(random * (max - min + 1) + min)
}
pure (1, 10, 0.42) // 5
pure (1, 10, 0.42) // 5



// прозорість посилань. Це ефект, який дозволяє
замість виклику функції без особливих труднощів
підставити результат роботи.

const refTransparency = () => Math.pow( 2, 5 ) +
Math.sqrt( 100 );
refTransparency(); // виклик функції
// або
Math.pow( 2, 5 ) + Math.sqrt( 100 ); // можна
розкрити і зрозуміти результат
32 + 10; // 42
```

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

найважливіші підходи ФП, які представлено в React :

- функції можуть бути присвоєні іншим змінним;
- декларативне оголошення функції;
- чиста функція без побічних ефектів;
- генерація функції;
- функція вищого порядку;
- комбінації функцій (compose) та керування (curry).

Створення обробника подій в Redux: `const middleware = store => next => action => {…}`

Суть процесу рендерінгу сторінки React – це вкладений процес виклику функцій. Компонент передається батьківським компонентом крізь props.

використання функціональної парадигми надає React таких корисних характеристик :

- швидкість розробки висока;
- функції, що часто використовуються, можуть постійно повторно використовувати логіку;
- можна легко зрозуміти корисність функції через ім'я, не знаючи внутрішньої реалізації;
- код більш зрозумілий;
- метод є чистою функцією і не впливає на зовнішні змінні;
- послідовність обробки може бути організована довільно.

ADMIN_MODE_ON_

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

переваги React-hooks :

- легше ділитися логікою стану (кастомні хуки);
- рятування від HOC і wrapper-hell;
- ефективніша мінімізація;
- не потрібно зберігати імена методів;
- не потрібно працювати з прототипами класів;
- не потрібно пам'ятати про this;
- спрощення логіки, пов'язаної із життєвим циклом;

Краще ментальне правило, яке застосовується до хуків, полягає в тому, що програмувати треба так, ніби будь-яке значення може змінитися в будь-який час.

Main hooks :

- useState
- useEffect
- useContext

Additional hooks :

- useCallback
- useMemo
- useReducer
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue
- useDeferredValue
- useTransition
- useId

Library hooks :

- useSyncExternalStore
- useInsertionEffect

ADMIN_MODE_ON_

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

Метод useState приймає один аргумент - значення за замовчуванням. Повертає кортеж (tuple), в якому є саме значення і метод для його зміни.

При створенні одразу декількох «стейтів» не потрібно думати про їх декомпозицію. Адже UseState розбиває стейт на логічні частини, щоб їх не змішувати в одному об'єкті, як у класах. І ці частини абсолютно повністю ізольовані, їх можна змінювати незалежно одна від одної.

```jsx
import { useState } from 'react';
import { declarative } from 'paradigms';

function ExampleWithManyStates() {

  // оголошення декількох змінних стану
  const [paradigm, setParadigm] = useState(declarative.fp);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{text: 'Hooks'}]);

  const addTodo = (todo) => {
    setTodos(prevState => {
      // Object.assign також працюватиме
      return {...prevState, todo};
    });
  }

  return (
    <div>…
    </div>
  );
}
```

На відміну від componentDidMount, що викликається один раз, useEffect викликається при кожному рендері.

useEffect виконується асинхронно, відразу після того, як відбувається зміна DOM. Це гарантує, що він буде виконаний після рендеру компонента, і може привести до наступного рендеру, якщо якісь значення зміняться.

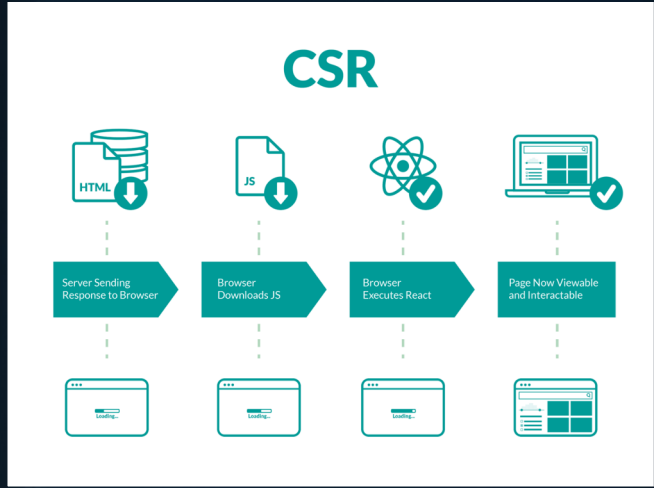
```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // відповідно componentDidMount и componentDidUpdate:
  useEffect(() => {
    // оновлюємо заголовок документу
    // за допомогою API браузера
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>{document.title}</p>
      <button onClick={ () => setCount( count + 1 ) }>
        Click me
      </button>
    </div>
  );
}
```

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS



плюси CSR :

- менше навантаження на сервер;
- відмінно підходить для веб-застосунків;
- швидкий рендер після першого завантаження;
- швидка навігація.

мінуси CSR :

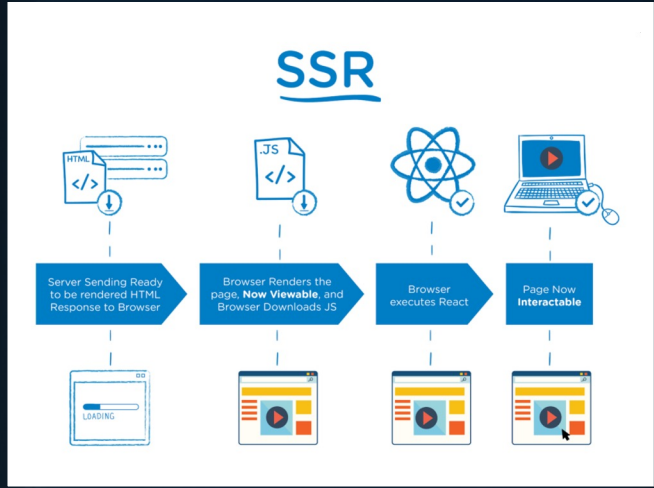
- повільне перше завантаження. «Повільне» - це досить відносний опис, але сьогодні навіть декілька секунд очікування користувачем можуть стати критичними;
- SEO - у разі неправильної індексації або наявності неправильної інформації у браузері, пошукові системи не зрозуміють сайт, і просто не виставлятимуть у топ пошуковика;

ADMIN_MODE_ON_

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

LOADING...

COMPLETED_59 %



плюси SSR :

- SEO. Як і у випадку з SSG, пошуковий робот отримує всю необхідну інформацію;
- First Contentful Paint. Завдяки SSR сторінки сайту швидше стають доступними для взаємодії. Маючи продуктивний сервер, SSR може дати чудову оцінку First Contentful Paint, що покращує UX і SEO сторінки.

мінуси SSR :

- уповільнюється час переходу між сторінками. Для початкового рендеру SSR швидше, ніж CSR, але при подальшій роботі з додатком доводиться рендерити дані двічі, один раз на сервері та один раз на клієнті.

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

ADMIN_MODE_ON_

LOADING...

COMPLETED_61 %

NEXT →

Задача :

розробити складний веб-додаток з динамічними даними, використовуючи сучасні інструменти, з налаштуванням SEO-оптимізації, з підтримкою Typescript, з передзавантаженням роутів, і, бажано, з мінімальними налаштуваннями конфігурації.

набір можливостей та властивостей, необхідний для сучасного додатка :

- швидкі та плавні переходи між сторінками, як у CSR;
- можливість роботи з SEO;
- швидкий First Contentful Paint;
- робота з динамічними даними;
- зменшення витрат, пов'язаних із вмістом сервера.

ADMIN_MODE_ON_

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

LOADING...

COMPLETED_63 %

плюси Next.js :

- завдяки вбудованому рендерінгу на стороні сервера, програми Next.js завантажуються значно швидше, ніж програми React;
- підтримує функцію експорту статичних сайтів;
- швидкий вхід для тих, хто вже працював із бібліотекою React.js;
- автоматичний поділ коду для сторінок;
- легко створювати внутрішні API-інтерфейси за допомогою вбудованих маршрутів API та створювати кінцеві точки API;
- вбудована підтримка маршрутизації сторінок, CSS, JSX та TypeScript;
- швидке додавання плагінів для налаштування Next.js відповідно до потреб сторінки;
- висока популярність.

Next.js by Vercel :

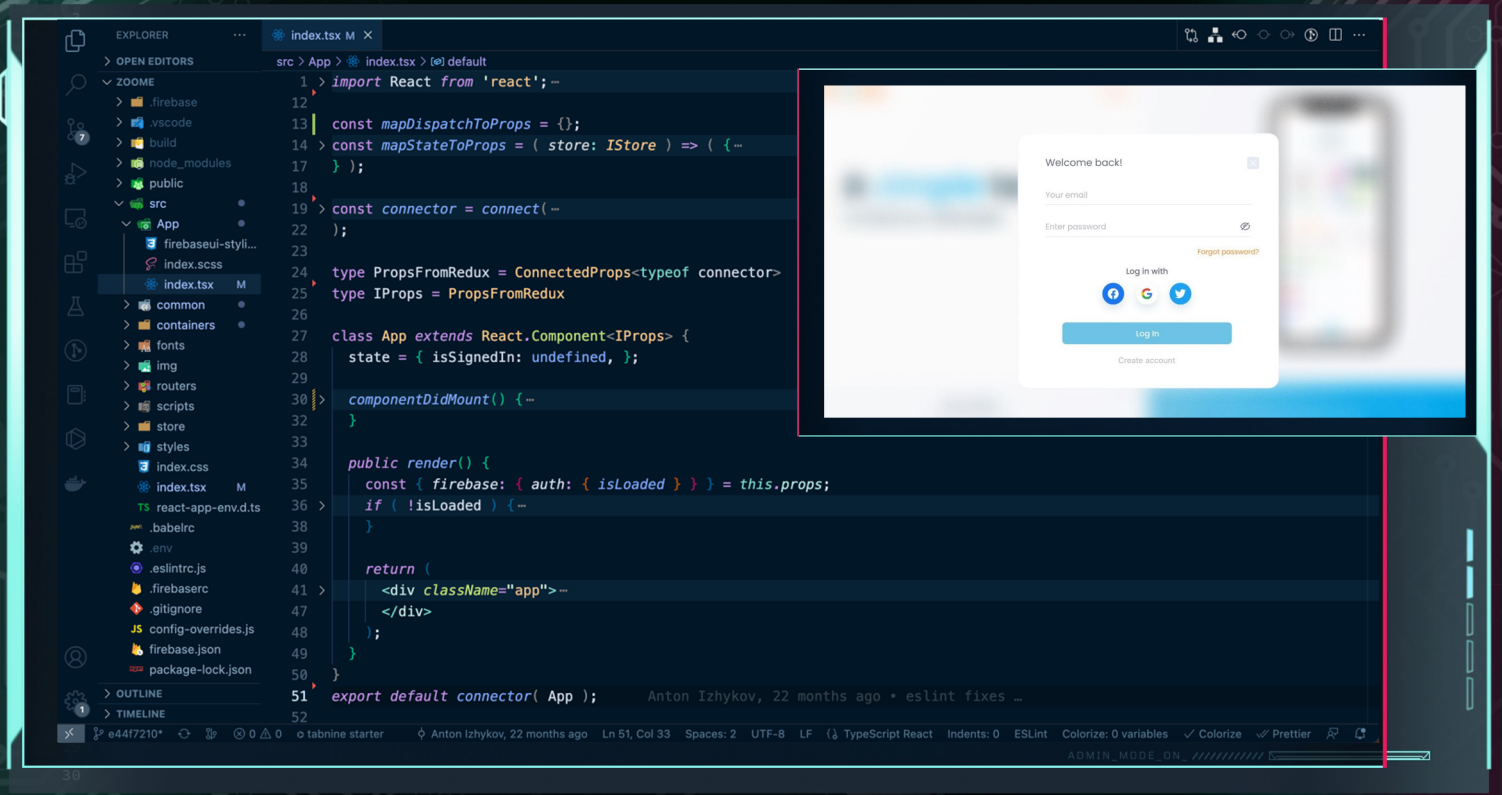
- робота без налаштування. Використання файлової системи як API
- лише JavaScript. Все є функціями
- автоматичний Server Side Rendering та code-splitting
- механізм отримання даних визначається розробником
- передзавантаження для збільшення продуктивності
- простий деплой та розгортання

ADMIN_MODE_ON_

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

LOADING...

COMPLETED_69 %



: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

LOADING...

COMPLETED_72 %

NEXT →

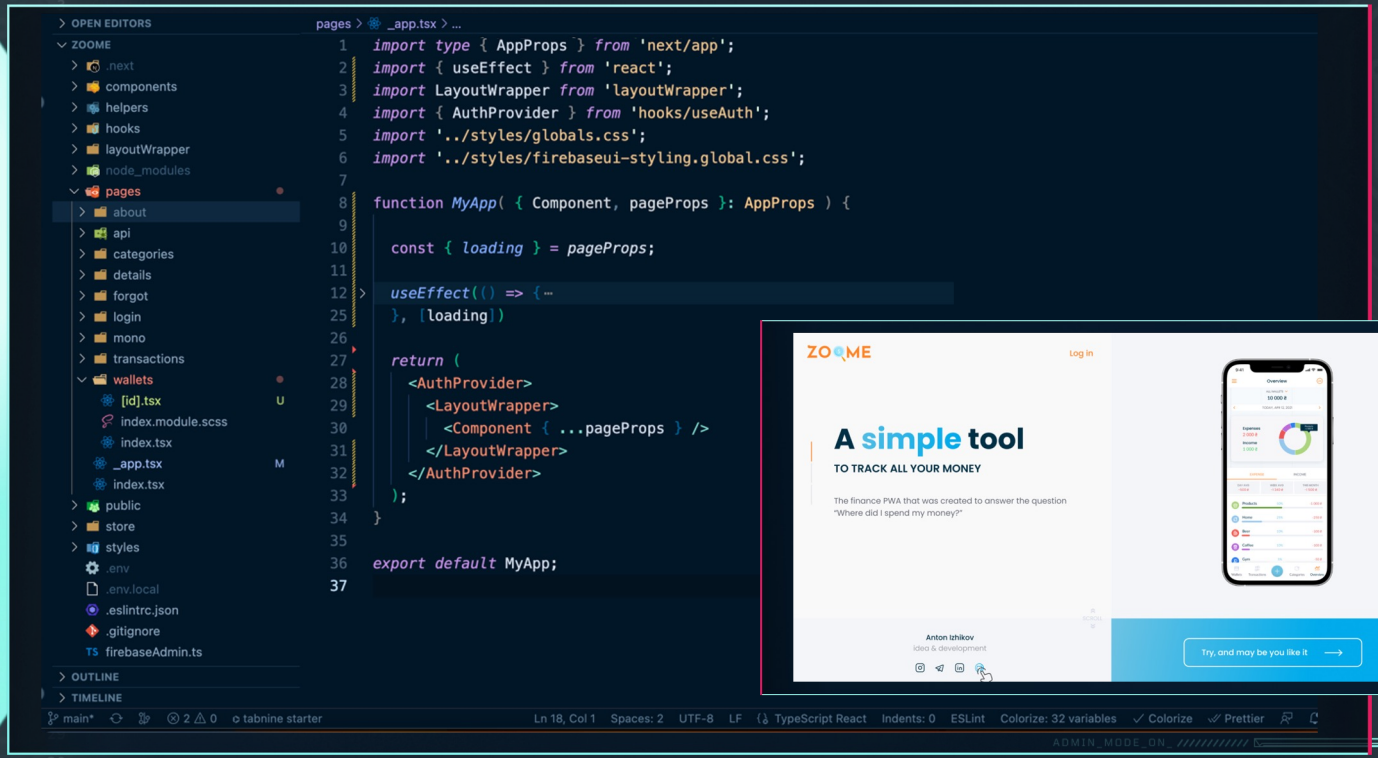
: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS



LOADING...

COMPLETED_75 %

NEXT

```tsx
const LayoutWrapper = ( { children, title, description, keywords, ...rest }: ILayoutProps ) => {

  return (
    <div className={ styles.wrapper }>
      <Head>
        <title>{title}</title>
        <meta name="description" content={ description } />
        <meta name="keywords" content={ keywords } />
      </Head>
      <Header classname={ styles.header }/>
      <Sidebar classname={ styles.sidebar }/>
      <div className={ styles.body }>
        {children}
      </div>
      <Footer classname={ styles.footer } />
    </div>
  );
};

LayoutWrapper.defaultProps = {
  title: 'Zoome',
  description: 'Personal finansial assistant',
  keywords: 'money, savings, assistant',
};
export default LayoutWrapper;
```

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

ADMIN_MODE_ON_

LOADING...

COMPLETED_77 %

NEXT →

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

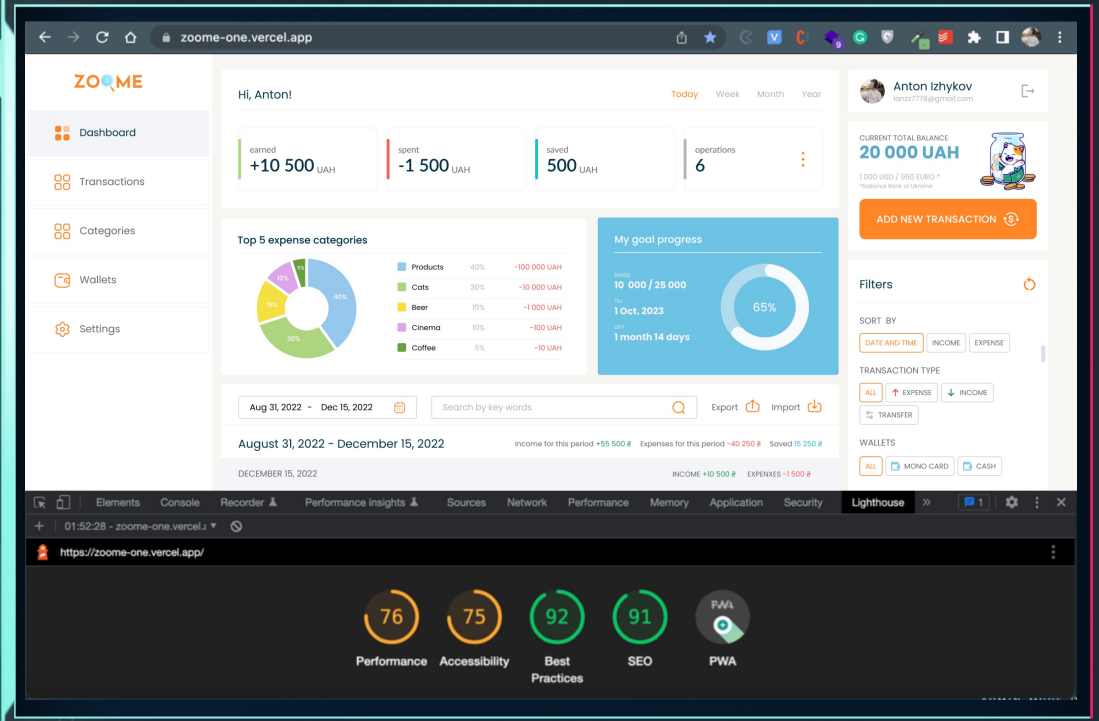
: RESULTS

```
19- const connector = connect(mapStateToProps, mapDispatchToProps);
20-
21- type PropsFromRedux = ConnectedProps<typeof connector>;
22-
23- type IProps = PropsFromRedux & WithFirestoreProps;
24- interface IState {
25-   currentType: IWalletType;
26- }
27- const composer = compose(
28-   firestoreConnect(() => {
29-     return [
30-       {
31-         collection: "users",
32-         doc: `${getUid()}`,
33-         subcollections: [{ collection: "wallets" }],
34-         storeAs: "wallets",
35-       },
36-     ];
37-   }),
38-   connector
39- );
40-
41- class Wallets extends React.Component<IProps, IState> {
42-   state = { currentType: "all" };
43-
44-   updateType = (type: IWalletType) => {
45-     this.setState({ currentType: type });
46-   };
47-
48-   public render() {
49-     const { currentType } = this.state;
50-     return (
51-       <div className="wallets">
52-         <div className="upperPart">
53-           <TopBalance />
54-           <TypeSwitcher
55-             currentType={currentType}
```

```
58+       </LayoutWrapper>
59+     );
60+ };
61+
62+ export const getServerSideProps = async ( ctx: GetServerSidePropsContext ) => {
63+   try {
64+     const cookies = nookies.get( ctx );
65+     const token = await firebaseAdmin.auth().verifyIdToken( cookies.token );
66+     const { uid, email } = token;
67+     const wallets: IWallet[] = getArrayFromCollection(
68+       await firebaseClient
69+         .firestore()
70+         .collection( 'users' )
71+         .doc( uid )
72+         .collection( 'wallets' )
73+         .get()
74+     ) as IWallet[];
75+     return {
76+       props: {
77+         message: `Your email is ${email} and your UID is ${uid}.`,
78+         wallets: wallets,
79+       },
80+     };
81+   } catch ( err ) {
82+     return {
83+       redirect: {
84+         permanent: false,
85+         destination: '/login',
86+       },
87+       props: {} as never,
88+     };
89+   }
90+ };
91
92+ export default Wallets;
```

LOADING...

COMPLETED_80 %



Google Lighthouse — це автоматизований інструмент з відкритим кодом для вимірювання якості веб-сторінок. Його можна запустити на будь-якій веб-сторінці, яка є загальнодоступною або потребує аутентифікації.

Google Lighthouse перевіряє продуктивність, доступність та пошукову оптимізацію веб-сторінок.

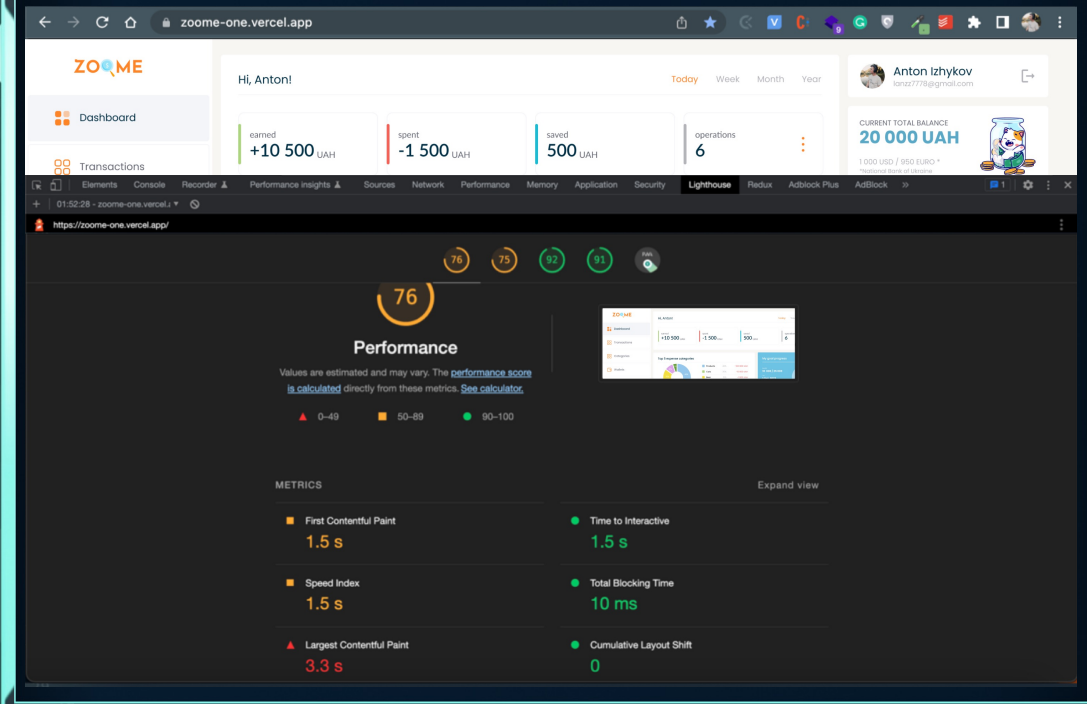
: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

ADMIN_MODE_ON_

LOADING...

COMPLETED_84 %

NEXT →



Lighthouse аналізує чотири показники: продуктивність, доступність, SEO та найкращі практики. Для прогресивних веб-застосунків додається п'ятий параметр — PWA.

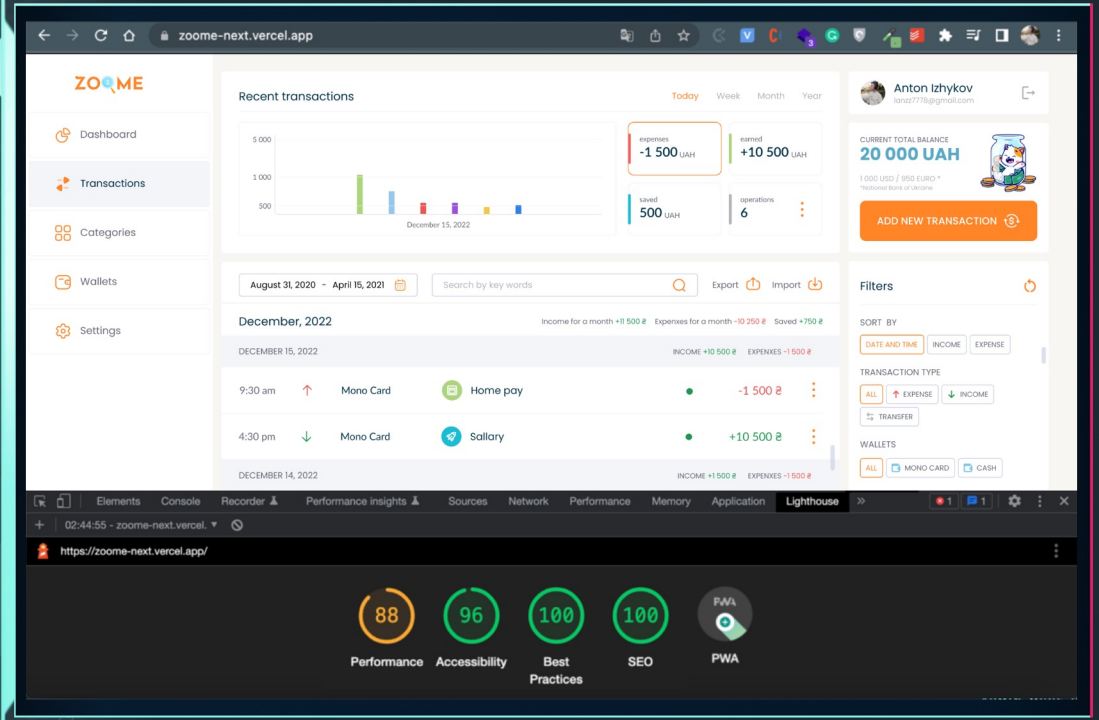
Performance – продуктивність. Аналізує швидкість завантаження сайту. На цю оцінку впливає час блокування, відтворення стилів, завантаження інтерактивних елементів, шрифтів та контенту.

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

ADMIN_MODE_ON_

LOADING...

COMPLETED_88 %



: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

Best Practices - найкращі практики. Перевіряє безпеку сайту та використання сучасних стандартів веб-розробки.

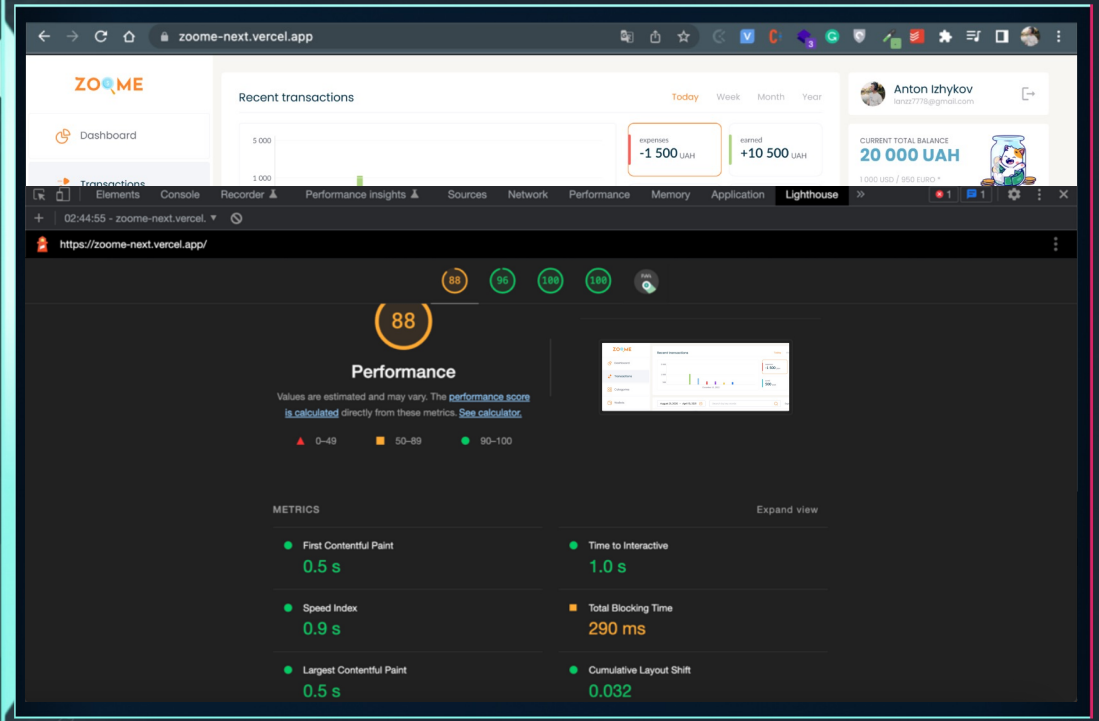
Оцінка залежить від того, чи використовується HTTPS, застарілі API, правильне кодування та інші параметри

ADMIN_MODE_ON_

LOADING...

COMPLETED_91 %

NEXT →



Accessibility – доступність. Перевіряє, чи всі користувачі можуть отримувати доступ до контенту і ефективно переміщатися по сайту.

Ця оцінка залежить від зрозумілості та сприймання контенту, можливості керувати інтерфейсом і пересуватися вмістом без допомоги миші.

: FP VS OOP
: JAVASCRIPT
: REACT
: NEXT
: RESULTS

ADMIN_MODE_ON_

LOADING...

COMPLETED_95 %

NEXT →

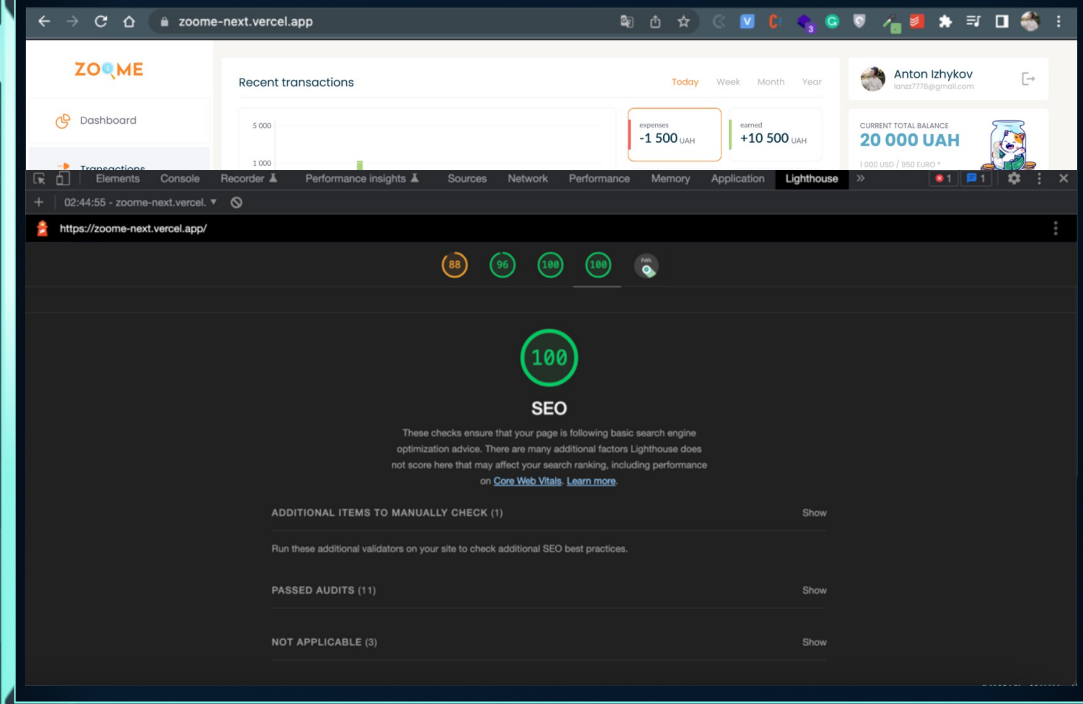
: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS



SEO – оцінює відповідність сторінки порадам Google з пошукової оптимізації.

Тут перевіряється використання метатегів, доступ до індексації та переобходу роботами, наявність атрибутів alt у зображень, адаптованість до мобільних екранів та інші характеристики.

ADMIN_MODE_ON_

LOADING...

COMPLETED_99 %

NEXT →

05

ВИСНОВКИ_

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

```tsx
import type { AppProps } from 'next/app';
import { useEffect } from 'react';
import LayoutWrapper from 'layoutWrapper';
import { AuthProvider } from 'hooks/useAuth';
import '../styles/globals.css';
import '../styles/firebaseui-styling.global.css';

function MyApp( { Component, pageProps }: AppProps ) {

  const { completed } = pageProps;

  useEffect( () => {…
  }, [completed] );

  return (
    <AuthProvider>
      <LayoutWrapper>
        <Component { ...pageProps } />
      </LayoutWrapper>
    </AuthProvider>
  );
}

export default MyApp;
```

- В ході даної роботи проведено аналіз функціональної парадигми у галузі веб-розробки та дослідження механізмів зміни програмної парадигми при інкапсуляції функціонального підходу у JavaScript для покращення якості та продуктивності веб-додатків.
- Проаналізовано розвиток та становлення мови JavaScript з огляду функціонального підходу. JS – мультипарадигмова мова програмування і дозволяє користуватися механізмами та можливостями функціонального стилю.
- У процесі дослідження використано такі методи та підходи функціональної парадигми: функції першого класу, функції вищого порядку, замикання, часткове застосування та карування, чисті функції.

ADMIN_MODE_ON_

LOADING...

COMPLETED_100%

NEXT →

05

ВИСНОВКИ_

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

```
function MyApp( { Component, pageProps }: AppProps ) {

  const { completed } = pageProps;

  useEffect( () => {
    if ( completed === 100 ) {
      const { FP, OOP } = getInfo();
      const reactInstance = getReactJs();
      const nextInstance = getNextJs();

      const results = {
        isFPbetter: compare( FP, OOP ),
        isJsGoodEnoughForWeb:
          FP.includesJS( reactInstance ) &&
          FP.includesJS( nextInstance ),
        isAppImproved: true,
      };

      return log( results );
    }
  }, [completed] );

  return (
    <AuthProvider>
      <LayoutWrapper>
```

- Проведено огляд використання функціональної парадигми у роботі з бібліотекою React. З появою хуків мінімізації та useMemo, функціональні компоненти піддаються мемоізації.
- На основі аналізу переваг та недоліків видів рендерінгу сторінок, поєднавши плюси кожного з підходів та позбавитись мінусів, формалізовано набір вимог до сучасного додатка.
- Знайдено та перевірено інструмент, який надає можливостей досягти цільових показників. Основна перевага Next.js - вбудована підтримка SSR для підвищення продуктивності. Next також дозволяє редагувати тег <head> сайту. Тег <head> є основною частиною метаданих веб-сторінки та сприяє підвищенню SEO-рейтингу сайту.

ADMIN_MODE_ON_

LOADING...

COMPLETED_100%

NEXT →

05

: FP VS OOP

: JAVASCRIPT

: REACT

: NEXT

: RESULTS

ВИСНОВКИ_

Функціональний підхід надає великих можливостей. Написання «функціонального» коду дозволяє поглянути на проблему з іншого боку, де розробка рішення може бути більш ефективною.

В ході дослідження була отримана теоретична інформація та практика використання сучасних підходів функціонального програмування у веб-розробці. В результаті проведеної роботи отримано обгрунтований та перевірений інструмент для покращення показників продуктивності, розширюваності та модульності веб-застосунків, а також створення нових додатків з дотриманням сучасних вимог.

Результатом даної роботи є покращення показників веб-застосунку завдяки переводу з класової до функціональної парадигми та використання сучасних методик та практик.

Продуктивність – покращено на 12 пунктів - з 76 до 88 відсотків;
Доступність – додатку стала краще на 21 відсоток - з 75 до 96;
Best practice та SEO – виросли з 92 та 91 до 100 відсотків.

Функціональне програмування може змінити стиль написання коду на краще. Якщо існуюча кодова база вже не дає необхідного рівня управління складністю бізнес-області, код засмічено непотрібними залишками, а імплементація нового функціоналу призводить до рефакторингу та втрати часу, перехід на функціональне програмування надає значних переваг.

ADMIN_MODE_ON_

LOADING...

COMPLETED_100%

FINISH_ →

ДЯКУЮ ЗА УВАГУ_