# ДОДАТОК А

## Перелік посилань відповідно до наукових досліджень кафедри

7. Ковалев Є., Лєсна Н. Побудова швидкодіючих клієнтських веб-застосувань за допомогою сучасних програмних засобів і технологій // Наука онлайн: міжнародний електронний науковий журнал. 2018. №12.

8. Фоменко О., Лєсна Н. Дослідження методів і моделей асинхронного оновлення даних для підвищення продуктивності web-систем // Наука онлайн: міжнародний електронний науковий журнал. 2019. №1.

15. Kuzochkina A., Z. Dudar, Shirokopetleva M. Analyzing and Comparison of NoSQL DBMS // International Scientific and Practical Conference «Problems of Infocommunications. Science and Technology» (PIC S&T`2018). 2018. Proceedings 8632133. pp. 560-565.

# ДОДАТОК Б

## Слайди презентації

Атестаційна робота магістра

ДОСЛІДЖЕННЯ ВПЛИВУ МЕТОДІВ ТА ЗАСОБІВ КОНТРОЛЮ ГЛОБАЛЬНОГО СТАНУ КОМПОНЕНТІВ НА ЕФЕКТИВНІСТЬ РОЗРОБКИ, ВИКОНАННЯ ТА СУПРОВОДУ REACT ДОДАТКІВ

Виконав:
ст. гр. ПЗСм-19-1 Пироженко С.С.

Керівник проекту проф. Лєсна Н.С.

2

## МЕТА РОБОТИ. ОБ'ЄКТ І ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Метою роботи** є дослідження впливу методів та засобів контролю глобального стану компонентів на ефективність розробки, виконання та супроводу React додатків.

**Об'єктом дослідження** є процес взаємодії React компонентів та глобальних сховищ.

**Предметом дослідження** є методи та засоби контролю глобального стану React додатків.

3

## АКТУАЛЬНІСТЬ ТЕМИ. НАУКОВА НОВИЗНА

**Актуальність** теми атестаційної роботи обумовлена наявністю проблеми вибору методів та засобів реалізації глобального сховища React додатків. Такий вибір постає перед розробниками на початку роботи над проектом і може вплинути як на саму розробку, її складність та тривалість цієї розробки, так і на подальшу роботу додатку, а саме: стабільність роботи, продуктивність, масштабування й супровід.

**Наукова новизна**. Вперше запропонована методика оцінювання ефективності методів та засобів контролю глобального стану React додатків, розроблені процедури моделювання продуктивності роботи компонентів внутрішнього устрою методів та засобів контролю глобального стану React додатку і продуктивності взаємодії React компонентів з глобальним сховищем.

4

## ПРЕДМЕТНА ОБЛАСТЬ. ІСНУЮЧІ МЕТОДИ ТА ЗАСОБИ

flux

Redux

Mobx

Apollo Link State

Unstated

Context API

5

## ПРЕДМЕТНА ОБЛАСТЬ. РЕЗУЛЬТАТИ АНАЛІЗУ. REDUX

| Переваги | Недоліки |
|---|---|
| ➢ жорстка архітектура, яка спрощує роботу у команді;<br>➢ verbosity – «багатослівність», яка спрощує тестування, супровід і масштабування додатку;<br>➢ широкі можливості по розширенню функціоналу за допомогою бібліотек;<br>➢ відмінний інструментарій для тестування та відладки прямо у браузері;<br>➢ можливість створення окремого шару для роботи з даними (DAL). | ➢ жорстка архітектура, яка обмежує розробника;<br>➢ verbosity – «багатослівність», яка збільшує час на розробку й є надмірною для невеликих додатків;<br>➢ не підтримує асинхронність «з коробки»;<br>➢ має погану продуктивність і потребує додаткового налаштування компонентів для оптимальної роботи з Redux сховищем. |

6

## ПРЕДМЕТНА ОБЛАСТЬ. РЕЗУЛЬТАТИ АНАЛІЗУ. MOBX

| Переваги | Недоліки |
|---|---|
| ➢ багато функціоналу працює автоматично;<br>➢ швидкість роботи «з коробки»;<br>➢ швидкість реалізації;<br>➢ мінімальна необхідність написання шаблонного коду;<br>➢ незначна можливість розширення за допомогою бібліотек. | ➢ багато функціоналу працює автоматично й недоступно для розробника, що може призводити до неочікуваних результатів і неможливості відстеження кожного кроку роботи;<br>➢ відсутність зручних інструментів для тестування;<br>➢ потрібні конфігурації IDE, а також TypeScript або Babel для використання декораторів. |

7

## ПРЕДМЕТНА ОБЛАСТЬ. РЕЗУЛЬТАТИ АНАЛІЗУ. CONTEXT API

| Переваги | Недоліки |
|---|---|
| ➢ не потрібно додаткових бібліотек;<br>➢ можливість створення багатьох сховищ зі своїми функціями-редьюсерами та діями. | ➢ погана продуктивність у порівняні з іншими засобами при частому оновленні даних;<br>➢ відсутність можливості розширення функціоналу;<br>➢ слабкі можливості з відладки та тестування. |

8

## ПОСТАНОВКА ЗАДАЧІ

- аналіз існуючих методів та засобів контролю глобального сховища;
- вибір найбільш перспективних методів та засобів для їх подальшого дослідження;
- визначення поняття ефективності та дослідження критеріїв ефективності;
- визначення метрик для оцінювання критеріїв ефективності;
- розробка методики оцінювання ефективності методів та засобів контролю глобального стану React додатків;
- розробка процедури моделювання продуктивності роботи компонентів внутрішнього устрою методів та засобів контролю глобального стану додатку;
- розробка процедури моделювання продуктивності взаємодії React компонентів з глобальним сховищем;
- розробка програмної системи для дослідження ефективності методів та засобів контролю глобального стану додатку;
- розробка рекомендацій по використанню методів та засобів контролю глобального стану React додатків за результатами оцінювання їх ефективності.

## КРИТЕРІЇ ЕФЕКТИВНОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

9

- масштабованість (scalability);
- можливості налагодження (debugging);
- сумісність (compatibility);
- крива навчання (learning curve);
- ширина інструментарію.

## КРИТЕРІЇ ЕФЕКТИВНОСТІ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

10

- вплив обраного методу/засобу на фінальний розмір додатку (Кб) при його збірці та подальшому розміщенні на сервері у виді статичних файлів;
- продуктивність роботи внутрішнього устрою елементів глобального сховища (мілісекунди), що визначає швидкість опрацювання дій користувача елементами глобального сховища;
- продуктивність взаємодії глобального сховища з React компонентами, що передбачає вимірювання: кількості оновлень викликаних зміною стану (одиниці), час оновлення кожного окремого компоненту (мілісекунди) та загальний час оновлення додатку (мілісекунди);
- латентність при виконанні високочастотних оновлень (мілісекунди), що визначає затримку при частому оновлені даних у глобальному сховищі та їх відображенні у підписаних на сховища компонентах у порівнянні з джерелом.

11

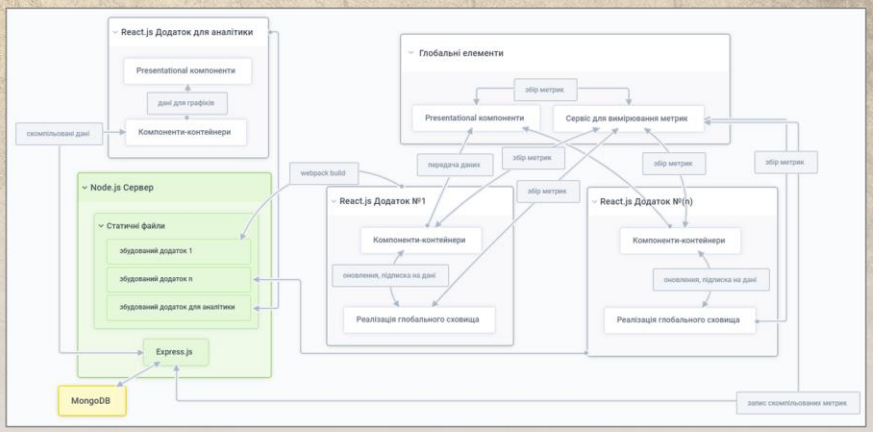
# МЕТОДИКА ОЦІНКИ ЕФЕКТИВНОСТІ

- моделювання продуктивності роботи елементів внутрішнього устрою обраних методів та засобів та формулювання результатів за метриками;
- моделювання продуктивності роботи при взаємодії глобального сховища з React компонентами та формулювання результатів за метриками;
- аналіз отриманого коду, вихідних файлів додатку та можливостей кожної з отриманих реалізацій з метою здійснення порівняння за критеріями ефективності розробки програмного забезпечення;
- розробка рекомендацій по використанню обраних методів та засобів;
- здійснення вибору найбільш ефективного методу, засобу контролю глобального стану React додатків.

12

# РОЗРОБЛЕНА ПРОГРАМНА СИСТЕМА

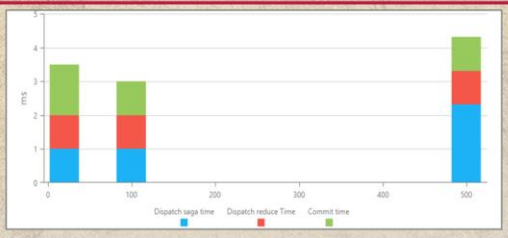
- додаток для кожної реалізації методу/засобу контролю глобального стану React додатку;
- компоненти інтерфейсу користувача, які використовуються у кожному з додатків;
- сервіс для збору метрик;
- сервер з базою даних для компіляції даних сервісу;
- додаток, який подає скомпільовані дані у графічному вигляді для їх аналізу;



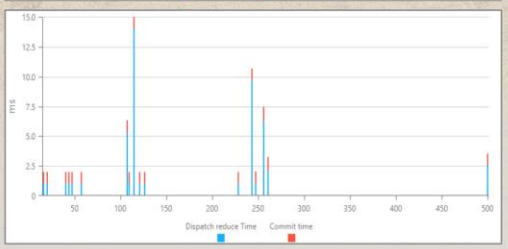
13

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. REDUX №1

Залежність швидкості завантаження від кількості елементів



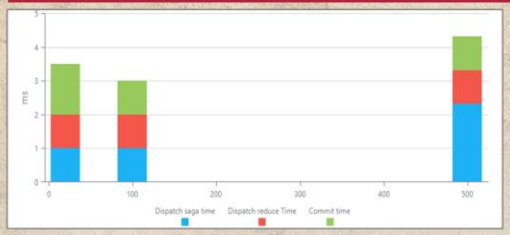

Залежність швидкості оновлення від кількості елементів

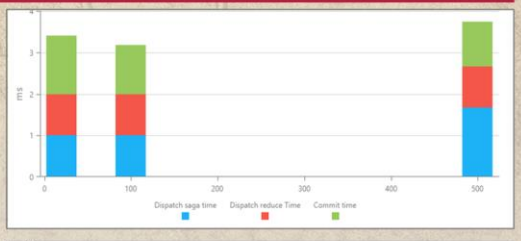


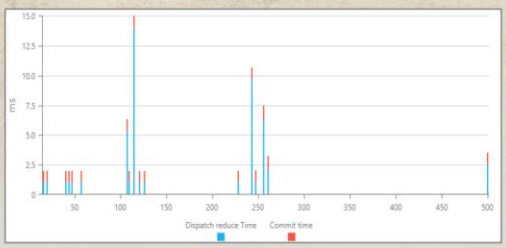

- не має залежності від розміру;
- можливість кешування дії, що впливає на час її ініціалізації;
- кешування недовгострокове та нестабільне;
- середня швидкість завантаження 3.5с;
- середня швидкість оновлення 5.5с;

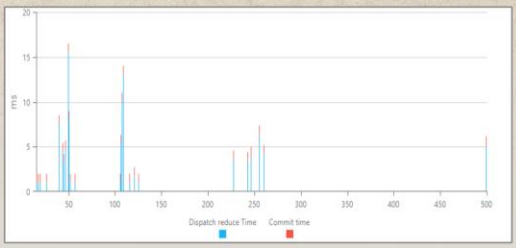
14

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. ПОРІВНЯННЯ REDUX №1 ТА REDUX №2



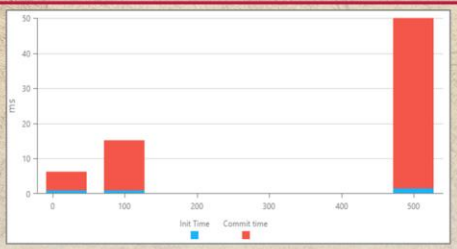



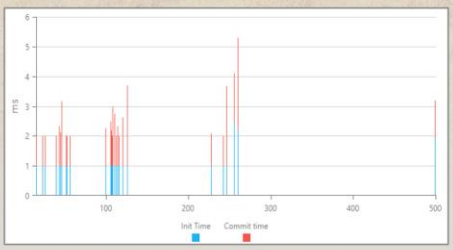
15

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. МОВХ №1

Залежність швидкості завантаження від кількості елементів



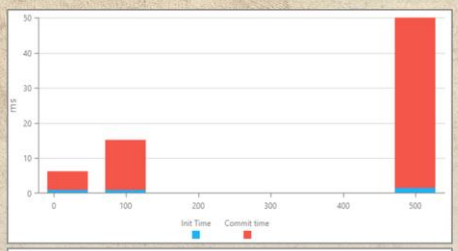

Залежність швидкості оновлення від кількості елементів

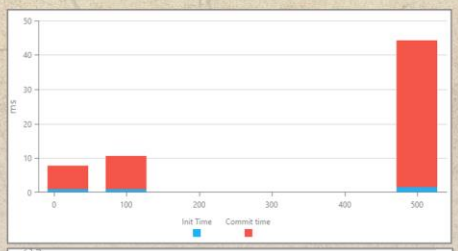


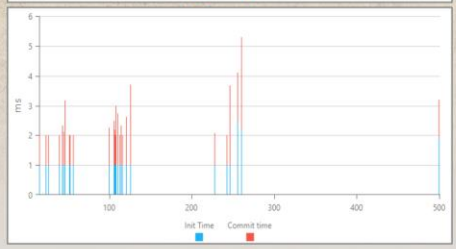

- час ініціалізації дії не залежить від даних, але залежить від навантаження на браузер;
- час фіксації дії має залежність від розміру даних.

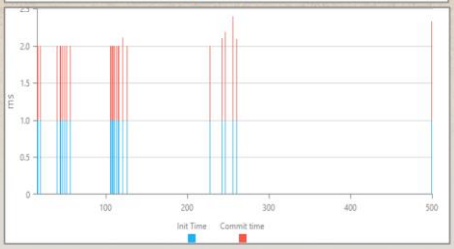
16

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. ПОРІВНЯННЯ МОВХ №1 ТА МОВХ №2



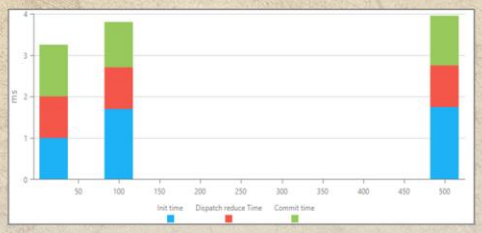



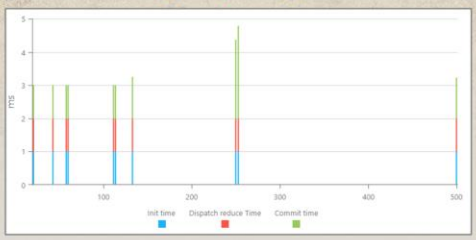
17

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. CONTEXT API

Залежність швидкості завантаження від кількості елементів




Залежність швидкості оновлення від кількості елементів




- майже не має залежності від розміру даних;
- позбавлений проблем кешування, які наявні у Redux.

18

# РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. ТРИВАЛІСТЬ РЕНДЕРИНГУ ОКРЕМИХ КОМПОНЕНТІВ ТА УСЬОГО ДОДАТКУ ПРИ 500 ЕЛЕМЕНТАХ ВИБІРКИ

| Засіб / Характеристика | Redux | MobX №1 | MobX №2 | Context |
|---|---|---|---|---|
| Мінімальний час (компонент) | 1.03 мс | 1.12 мс | 1.13 мс | 1.06 мс |
| **Середній час (компонент)** | **1.95 мс** | **2.21 мс** | **2.27 мс** | **2.02 мс** |
| Максимальний час (компонент) | 17.68 мс | 27.35 мс | 19.2 мс | 25.96 мс |
| Мінімальний повний час (додаток) | 1160.4 мс | 1243.5 мс | 1078.2 мс | 1238.7 мс |
| **Середній повний час (додаток)** | **1022.71 мс** | **1062.16 мс** | **934.06 мс** | **1015.64 мс** |
| Мінімальний повний час (додаток) | 747.5 мс | 845.9 мс | 753.3 мс | 843 мс |

19

## РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. ЛАТЕНТНІСТЬ ПРИ ВИСОКОЧАСТОТНОМУ ОНОВЛЕННІ

Порівняння оригінального відео та скріншоту (кадру) цього відео, який передається через глобальне сховище кожні 4мс


Redux


MobX


Context API

20

## РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. РОЗМІР ВИХІДНИХ ФАЙЛІВ У ЗАЛЕЖНОСТІ ВІД ОБРАНОГО ЗАСОБУ ТА ЙОГО РЕАЛІЗАЦІЇ

| Реалізація | Розмір |
|---|---|
| Redux №1 | 2.30 Кб |
| Redux №2 | 2.35 Кб |
| Redux тест високочастотного оновлення | 214 Кб |
| MobX №1 | 2.32 Мб |
| MobX №2 | 2.32 Мб |
| MobX тест високочастотного оновлення | 266 Кб |
| Context API + React Hooks | 2.26 Мб |
| Context API + React Hooks тест високочастотного оновлення | 267 Кб |

21

## РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. АНАЛІЗ КОДУ

| Характеристика / Реалізація | Кількість файлів | Кількість рядків коду | | |
|---|---|---|---|---|
| | | Вцілому | З них типізація | |
| | | | Вцілому | «inline» типізація |
| **Redux №1** | 8 | 240 | 82 | 26 |
| **Redux №2** | - | - | - | - |
| **Redux тест високочастотного оновлення** | 1 | 52 | 17 | 4 |
| **MobX №1** | 1 | 77 | 9 | 9 |
| **MobX №2** | 1 | 74 | 9 | 9 |
| **MobX тест високочастотного оновлення** | 1 | 13 | 2 | 2 |
| **Context API + React Hooks** | 4 | 164 | 53 | 12 |
| **Context API + React Hooks тест високочастотного оновлення** | 1 | 64 | 23 | 5 |

22

## РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ. ПОРІВНЯННЯ ЗАСОБІВ ЗА КРИТЕРІЯМИ ЕФЕКТИВНОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

| Засіб / Критерій (бали) | Redux | MobX | Context API |
|---|---|---|---|
| Масштабованість (scalability) | 3 | 2 | 1 |
| Можливості налагодження (debugging) | 3 | 2 | 1 |
| Сумісність (compatibility) | 2 | 1 | 3 |
| Крива навчання (learning curve) | 1 | 3 | 2 |
| Ширина інструментарію | 3 | 2 | 1 |
| **Сума** | **12** | **10** | **8** |

# АПРОБАЦІЯ РОБОТИ

- міжнародний молодіжний форум «Радіотехніка та молодь у XXI столітті», опубліковано тези доповіді «Критерії вибору стратегії керування глобальним станом React додатків»;
- стаття «Дослідження продуктивності методів та засобів контролю глобального стану компонентів React додатків» у міжнародному науковому журналі «Наука онлайн».

МІЖНАРОДНИЙ ЕЛЕКТРОННИЙ НАУКОВИЙ ЖУРНАЛ "НАУКА ОНЛАЙН"
INTERNATIONAL ELECTRONIC SCIENTIFIC JOURNAL "SCIENCE ONLINE"
(Свідоцтво про реєстрацію: КВ № 22285-12185Р)

**СВІДОЦТВО**
**про публікацію**

Серія: НО № 02056

*засвідчує, що стаття*

**Пироженка**
**Сергія Станіславовича**

*на тему:*

**«ДОСЛІДЖЕННЯ ПРОДУКТИВНОСТІ МЕТОДІВ ТА ЗАСОБІВ КОНТРОЛЮ ГЛОБАЛЬНОГО СТАНУ КОМПОНЕНТІВ REACT ДОДАТКІВ»**

*опублікована в*

*Міжнародному електронному науковому журналі "Наука онлайн"*
*ВИПУСК 10/2020*

*Web-адреса публікації:*
*https://nauka-online.com/publications/informatsionnye-tehnologii/2020/10/doslidzhennya-produktivnosti-metodiv-ta-zasobiv-kontrolyu-globalnogo-stanu-komponentiv-react-dodatkiv/*

*/ Головний редактор* *Т. Е. Беляпов*



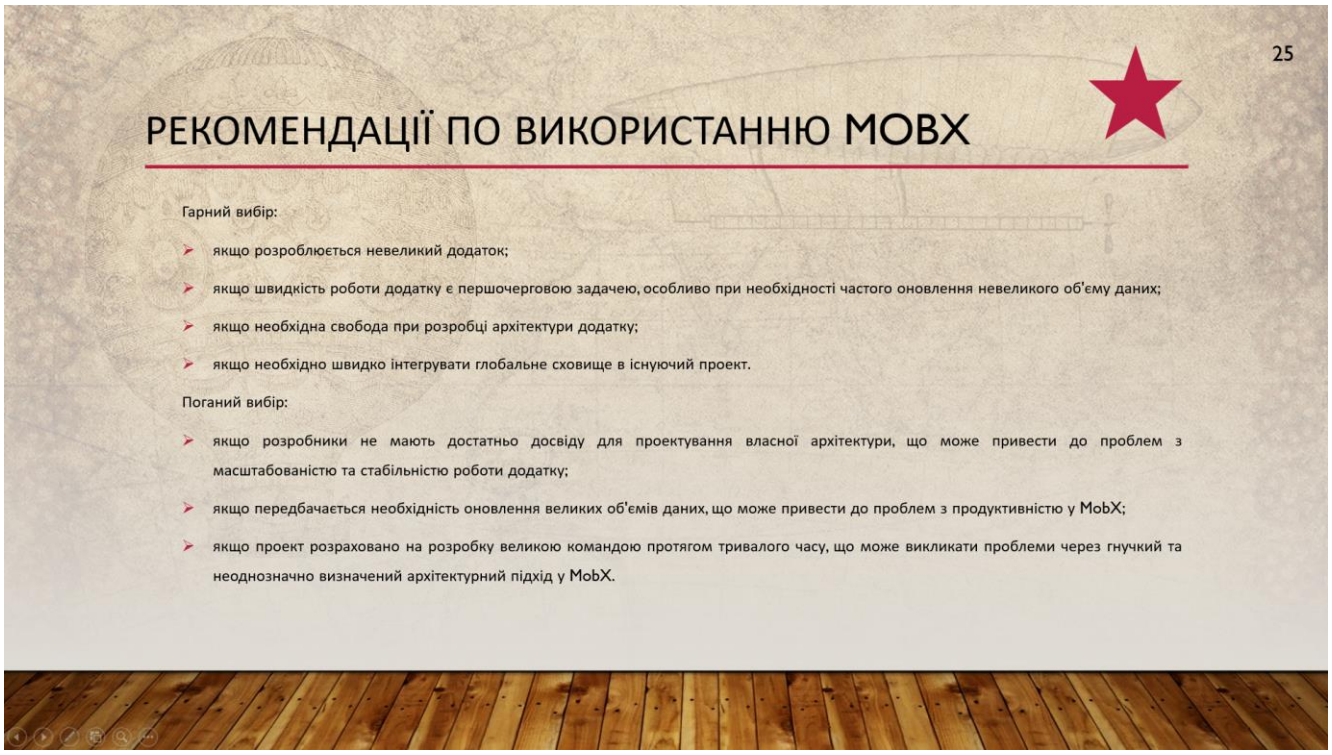
# РЕКОМЕНДАЦІЇ ПО ВИКОРИСТАННЮ REDUX

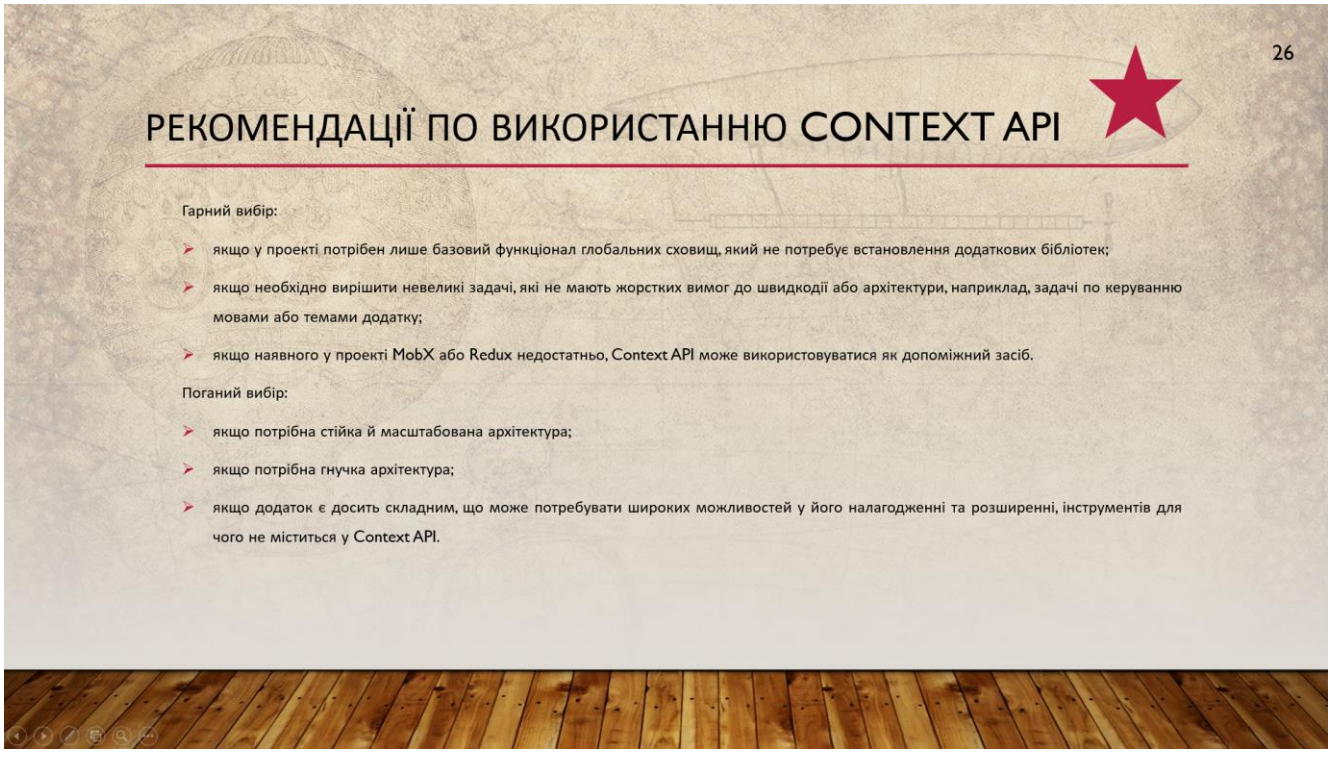
Гарний вибір:

- якщо метою є створення супроводжуємої та масштабованої архітектури, яка може показати достатню продуктивність;
- якщо передбачається робота з даними великого розміру, які при цьому можуть часто оновлюватися;
- якщо метою є створення великого додатку, який потребує широкого інструментарію по відлагодженню та розширенню для задоволення вимог навіть після року розробки.

Поганий вибір:

- якщо розробляється невеликий проект, при роботі над яким витрата часу на написання великої кількість шаблонного коду та дотримання строгих правил архітектури Redux може привести до непотрібних затримок у релізі;
- якщо проект вже має певну архітектуру, яка не зовсім співпадає або навіть конфліктує з досить громіздкими архітектурними рішеннями Redux;
- якщо швидкодія додатку є першочерговою задачею та при цьому не передбачається роботи з великими об'ємами даних;


25
РЕКОМЕНДАЦІЇ ПО ВИКОРИСТАННЮ MOBX
Гарний вибір:
якщо розроблюється невеликий додаток;
якщо швидкість роботи додатку є першочерговою задачею, особливо при необхідності частого оновлення невеликого об'єму даних;
якщо необхідна свобода при розробці архітектури додатку;
якщо необхідно швидко інтегрувати глобальне сховище в існуючий проект.
Поганий вибір:
якщо розробники не мають достатньо досвіду для проектування власної архітектури, що може привести до проблем з масштабованістю та стабільністю роботи додатку;
якщо передбачається необхідність оновлення великих об'ємів даних, що може привести до проблем з продуктивністю у MobX;
якщо проект розраховано на розробку великою командою протягом тривалого часу, що може викликати проблеми через гнучкий та неоднозначно визначений архітектурний підхід у MobX.


26
РЕКОМЕНДАЦІЇ ПО ВИКОРИСТАННЮ CONTEXT API
Гарний вибір:
якщо у проекті потрібен лише базовий функціонал глобальних сховищ, який не потребує встановлення додаткових бібліотек;
якщо необхідно вирішити невеликі задачі, які не мають жорстких вимог до швидкодії або архітектури, наприклад, задачі по керуванню мовами або темами додатку;
якщо наявного у проекті MobX або Redux недостатньо, Context API може використовуватися як допоміжний засіб.
Поганий вибір:
якщо потрібна стійка й масштабована архітектура;
якщо потрібна гнучка архітектура;
якщо додаток є досить складним, що може потребувати широких можливостей у його налагодженні та розширенні, інструментів для чого не міститься у Context API.

# ВИСНОВКИ

Проведений аналіз предметної області.

Досліджені й обгрунтовані критерії оцінювання ефективності розробки React додатків та їх роботи.

Запропонована методика оцінювання ефективності методів та засобів контролю глобального стану React додатків, розроблені процедури моделювання продуктивності роботи компонентів внутрішнього устрою методів та засобів контролю глобального стану React додатку і продуктивності взаємодії React компонентів з глобальним сховищем.

Розроблена програмна система для дослідження ефективності обраних методів та засобів контролю глобального стану.

За допомогою розробленої програмної системи проведено дослідження:

- продуктивності роботи компонентів внутрішнього устрою методів та засобів контролю глобального стану додатку;
- продуктивності взаємодії React компонентів додатку з глобальним сховищем;
- ефективності методів та засобів контролю глобального стану React додатків за критеріями ефективності розробки програмного забезпечення.

Розроблені рекомендації по використанню методів та засобів контролю глобального стану React додатків.

Результати оцінювання ефективності та розроблені рекомендації щодо вибору методів та засобів контролю глобального стану React додатків дозволять пришвидшити розробку React додатків, зменшивши її складність та збільшити продуктивність і стабільність роботи додатків, а також зменшити затрати на їх супровід.

# ДЯКУЮ ЗА УВАГУ!
# ВАШІ ЗАПИТАННЯ?

# ДОДАТОК В

## Лістинг коду програми

Далі наведений код глобального сховища, реалізованого за допомогою бібліотеки MobX.

```
import { createContext } from "react";
import { observable, runInAction } from "mobx";

import { fetchBlogList } from "../../../axios/blogs";
import { generateFilters, isBlogPasses } from "../../../utils/filters";

import { IBlog } from "../../../interfaces/Blog";
import {
  IBooleanFilter, IValueFilter, IRangeFilter, isBooleanFilter,
  isValueFilter, isRangeFilter,
} from "../../../interfaces/Filter";

import { TrackerService } from "../../../services/tracker";
import { TrackerActions, TrackerSources, TrackerPositions } from
"../../../../shared/interfaces";

type FilterCallback = (item: IBlog) => boolean;

const mutationFilter = (arr: Array<IBlog>, callback: FilterCallback) => {
  for (let i = arr.length - 1; i >= 0; i -= 1) {
    if (!callback(arr[i])) arr.splice(i, 1);
  }
};

class BlogListState {
  @observable defaultBlogs: Array<IBlog> = [];

  @observable blogs: Array<IBlog> = [];

  @observable defaultFilters:
  Array<IBooleanFilter|IValueFilter|IRangeFilter> = [];

  @observable filters: Array<IBooleanFilter|IValueFilter|IRangeFilter>=[];

  @observable loading = false;

  @observable error: string | null = null;

  fetchBlogList = async (limit: number) => {
    TrackerService.setTimeStamps({
      source: TrackerSources.MOBX_V1,
      position: TrackerPositions.MOBX_ACTION_INIT,
      action: TrackerActions.FETCH_BLOG_LIST,
      state: "finished",
      time: Date.now(),
    });
```

```
    this.loading = true;
    try {
      const blogs = await fetchBlogList({ limit });
      const filters = generateFilters(blogs);

      TrackerService.setTimeStamps({
        source: TrackerSources.MOBX_V1,
        position: TrackerPositions.MOBX_ACTION_COMMIT,
        action: TrackerActions.FETCH_BLOG_LIST,
        state: "started",
        time: Date.now(),
        affectedItems: blogs.length,
      });

      runInAction(() => {
        this.defaultFilters = filters;
        this.filters = filters;
        this.defaultBlogs = blogs;
        this.blogs = blogs;
        this.loading = false;
        this.error = null;
      });
    } catch (err) {
      this.loading = false;
      this.error = err;
    }
  }

  updateFilters = (title: string, value: boolean | number, secondValue:
number) => {
    TrackerService.setTimeStamps({
      source: TrackerSources.MOBX_V1,
      position: TrackerPositions.MOBX_ACTION_INIT,
      action: TrackerActions.CHECKBOX_FILTER,
      state: "finished",
      time: Date.now(),
    });

    runInAction(() => {
      const filters = this.filters.map((filter) => {
        if (filter.title !== title) return filter;
        const updated = filter;

        if (isBooleanFilter(updated)) {
          updated.value = !updated.value;
        } else if (isValueFilter(updated)) {
          updated.value = Number(value);
        } else if (isRangeFilter(updated)) {
          updated.min = Number(value);
          updated.max = Number(secondValue);
        }

        return updated;
      });

      if (value) {
        // we've added a new filter, can use current blogs
```

```
        TrackerService.setTimeStamps({
          source: TrackerSources.MOBX_V1,
          position: TrackerPositions.MOBX_ACTION_COMMIT,
          action: TrackerActions.CHECKBOX_FILTER,
          state: "started",
          time: Date.now(),
        });

        mutationFilter(this.blogs, (blog) => isBlogPasses(blog, filters));

        this.filters = filters;
      } else {
        // we've removed some of the filters. Have to use default blogs
        const blogs = this.defaultBlogs.map(
          (blog) => (isBlogPasses(blog, filters) ? blog : null),
        ).filter(Boolean);
        TrackerService.setTimeStamps({
          source: TrackerSources.MOBX_V1,
          position: TrackerPositions.MOBX_ACTION_COMMIT,
          action: TrackerActions.CHECKBOX_FILTER,
          state: "started",
          time: Date.now(),
        });
        this.blogs = blogs;
        this.filters = filters;
      }
    });
  }
}

export const blogListState = createContext(new BlogListState());
```

У коді вище також наведений приклад використання сервісу для вимірів часу оновлення й об’єму задіяних даних. Сам код сервісу наведений нижче.

```
import {
  OperationPartTimestamp,
  OperationPart,
  ReduxOperation,
  MobxOperation,
  ReduxSagaOperation,
  TrackerSources,
  TrackerPositions,
  ContextOperation,
} from "../../shared/interfaces";

import {
  sendMobxOperationTrackerInfo,
  sendReduxOperationTrackerInfo,
  sendReduxSagaOperationTrackerInfo,
  sendContextOperationTrackerInfo,
} from "../axios/tracker";
const searchOperationPartTimestamp = (
  x: OperationPartTimestamp,
```

```
  y: OperationPartTimestamp,
): boolean => x.source === y.source && x.action === y.action && x.position
=== y.position && x.state === "started";

class TrackerServiceClass {
  operationPartTimestampList: Array<OperationPartTimestamp> = [];

  reduxOperation: ReduxOperation = null;

  reduxSagaOperation: ReduxSagaOperation = null;

  mobxOperation: MobxOperation = null;

  contextOperation: ContextOperation = null;

  public setTimeStamps = (operationPartTimestamp: OperationPartTimestamp)
=> {
    console.log(operationPartTimestamp);

    const index = this.operationPartTimestampList.findIndex(
      (x) => searchOperationPartTimestamp(x, operationPartTimestamp),
    );

    const started = this.operationPartTimestampList[index];

    if (!started) {
      this.operationPartTimestampList.push(operationPartTimestamp);
      return;
    }

    const operationPartTime = operationPartTimestamp.time - started.time;

    this.setOperationPart({
      source: started.source,
      position: started.position,
      action: started.action,
      time: operationPartTime === 0 ? 1 : operationPartTime,
      affectedItems: operationPartTimestamp.affectedItems ||
started.affectedItems,
    });

    this.operationPartTimestampList.splice(index, 1);
  }


  private setOperationPart = (operationPart: OperationPart) => {
    switch (operationPart.source) {
      case TrackerSources.MOBX_V1:
      case TrackerSources.MOBX_V2:
        if (!this.mobxOperation && operationPart.position ===
TrackerPositions.MOBX_ACTION_INIT) {
          this.mobxOperation = {
            source: operationPart.source,
            action: operationPart.action,
            initTime: operationPart.time,
            commitTime: null,
            affectedItems: operationPart.affectedItems || null,
```

```
          };
        } else if (operationPart.position ===
TrackerPositions.MOBX_ACTION_COMMIT) {
          const operation = this.mobxOperation;
          operation.commitTime = operationPart.time;
          operation.affectedItems = operationPart.affectedItems;
          sendMobxOperationTrackerInfo(operation);
          this.mobxOperation = null;
        }
        break;
      case TrackerSources.REDUX_V1:
      case TrackerSources.REDUX_V2:
        if (!this.reduxSagaOperation && operationPart.position ===
TrackerPositions.REDUX_DISPATCH_SAGA) {
          this.reduxSagaOperation = {
            source: operationPart.source,
            action: operationPart.action,
            dispatchSagaTime: operationPart.time,
            dispatchReducerTime: null,
            commitTime: null,
            affectedItems: operationPart.affectedItems || null,
          };
          return;
        }
        if (this.reduxSagaOperation) {
          switch (operationPart.position) {
            case TrackerPositions.REDUX_DISPATCH_REDUCER:
              this.reduxSagaOperation.dispatchReducerTime =
operationPart.time;
              break;
            case TrackerPositions.REDUX_COMMIT:
              sendReduxSagaOperationTrackerInfo({
                ...this.reduxSagaOperation,
                commitTime: operationPart.time,
                affectedItems: operationPart.affectedItems,

              });
              this.reduxSagaOperation = null;
              break;
            default:
              break;
          }
          return;
        }
        if (!this.reduxOperation && operationPart.position ===
TrackerPositions.REDUX_DISPATCH_REDUCER) {
          this.reduxOperation = {
            source: operationPart.source,
            action: operationPart.action,
            dispatchReducerTime: operationPart.time,
            commitTime: null,
            affectedItems: operationPart.affectedItems || null,
          };
          return;
        }
        if (this.reduxOperation && operationPart.position ===
TrackerPositions.REDUX_COMMIT) {
```

```
          this.reduxOperation.commitTime = operationPart.time;
          this.reduxOperation.affectedItems = operationPart.affectedItems;
          sendReduxOperationTrackerInfo(this.reduxOperation);
          this.reduxOperation = null;
        }
        break;
      case TrackerSources.CONTEXT_V1:
        if (!this.contextOperation && operationPart.position ===
TrackerPositions.CONTEXT_INIT) {
          this.contextOperation = {
            source: operationPart.source,
            action: operationPart.action,
            initTime: operationPart.time,
            dispatchReducerTime: null,
            commitTime: null,
            affectedItems: operationPart.affectedItems || null,
          };
        } else if (this.contextOperation && operationPart.position ===
TrackerPositions.CONTEXT_DISPATCH_REDUCER) {
          this.contextOperation.dispatchReducerTime = operationPart.time;
        } else if (this.contextOperation && operationPart.position ===
TrackerPositions.CONTEXT_COMMIT) {
          const operation = this.contextOperation;
          operation.commitTime = operationPart.time;
          operation.affectedItems = operationPart.affectedItems;
          sendContextOperationTrackerInfo(operation);
          this.contextOperation = null;
        }
        break;
      default:
        break;
    }
  }
}

export const TrackerService = new TrackerServiceClass();
```

Нижче наведено код, який використовується для виводу агрегованих даних тестувань у вигляді графіків, які наведені у третьому розділі.

```
import * as React from "react";
import { Spin, Select, InputNumber } from "antd";
import { useLocation, useHistory } from "react-router-dom";
import { parse, stringify } from "query-string";
import {
  Chart, CommonSeriesSettings, Legend, ValueAxis, Title, Export, Tooltip,
} from "devextreme-react/chart";

import { TrackerActions, TrackerSources } from
"../../../../shared/interfaces";
import { getTrackers } from "../../../axios/tracker";

import { PresentationTracker } from "./interafaces";
import { PresentationWrapper, SelectGroup } from "./styled";
```

```
import { prettifyKey, renderSeries } from "./utils";

const DEFAULT_LIMIT = 10;
const { useEffect, useState, useMemo } = React;
const { Option } = Select;

interface PageProps {
  id: number;
}

export const ChartPage = ({ id }: PageProps): JSX.Element => {
  const { search } = useLocation();
  const history = useHistory();

  const searchParams = useMemo(() => parse(search), [search]);

  const [trackers, setTrackers] = useState<Array<PresentationTracker>>([]);

  const [source, setSource] = useState<TrackerSources>(null);
  const [action, setAction] = useState<TrackerActions>(null);
  const [limit, setLimit] = useState<number>(null);

  useEffect(() => {
    const sourceParam = searchParams[`source${id}`];
    const actionParam = searchParams[`action${id}`];
    const limitParam = searchParams[`limit${id}`];

    setSource(sourceParam ? Number(sourceParam) : TrackerSources.REDUX_V1);
    setAction(actionParam ? Number(actionParam) :
TrackerActions.CHECKBOX_FILTER);
    setLimit(limitParam ? Number(limitParam) : DEFAULT_LIMIT);
  }, [search]);

  const fetchTrackers = async () => {
    try {
      const response = await getTrackers({ source, action, limit });

      setTrackers(response.map<PresentationTracker>((t) => ({ ...t,
...t.averageTime })));
    } catch (err) {
      console.log(err);
    }
  };

  useEffect(() => {
    if (source && action && limit) {
      fetchTrackers();
    }
  }, [source, action, limit]);

  const updateParam = (type: "action" | "source" | "limit") => (value:
TrackerSources | TrackerActions | number) => {
    switch (type) {
      case "action":
        setAction(Number(value));
        searchParams[`action${id}`] = value.toString();
        history.push({
```

```
          search: stringify(searchParams),
        });
        break;
      case "source":
        setSource(Number(value));
        searchParams[`source${id}`] = value.toString();
        history.push({
          search: stringify(searchParams),
        });
        break;
      case "limit":
        setLimit(Number(value));
        searchParams[`limit${id}`] = value.toString();
        history.push({
          search: stringify(searchParams),
        });
        break;
      default:
        break;
    }
  };

  if (!trackers) {
    return <Spin />;
  }

  return (
    <PresentationWrapper>
      <SelectGroup>
        <Select id="source-select" value={source}
onChange={updateParam("source")}>
          {Object.entries(TrackerSources)
            .filter(([, value]) => typeof value === "number")
            .map(([key, value]) => <Option key={value}
value={value}>{prettifyKey(key)}</Option>)}
        </Select>
        <Select id="action-select" value={action}
onChange={updateParam("action")}>
          {Object.entries(TrackerActions)
            .filter(([, value]) => typeof value === "number")
            .map(([key, value]) => <Option key={value}
value={value}>{prettifyKey(key)}</Option>)}
        </Select>
        <InputNumber min={DEFAULT_LIMIT} max={50} value={limit}
onChange={updateParam("limit")} />
      </SelectGroup>
      <Chart
        id="chart"
        title="Update time to affected objects"
        dataSource={trackers}
      >
        <CommonSeriesSettings argumentField="affectedItems"
type="stackedBar" ignoreEmptyPoints />
        {renderSeries(source, action)}
        <ValueAxis position="left">
          <Title text="ms" />
        </ValueAxis>
```

```
        <Legend
          verticalAlignment="bottom"
          horizontalAlignment="center"
          itemTextPosition="top"
        />
        <Export enabled />
        <Tooltip
          enabled
          location="edge"
          customizeTooltip={(arg: any) => ({ text:
`[${TrackerSources[source].toLowerCase()}] \n
[${TrackerActions[action].toLowerCase()}] \n ${arg.seriesName}:
${arg.valueText}ms` })}
        />
      </Chart>
    </PresentationWrapper>
  );
};
```

# ДОДАТОК Г

## Наукові публікації

### Г.1 Стаття у міжнародному науковому журналі Наука Онлайн

*International Electronic Scientific Journal "Science Online" http://nauka-online.com/*

Інформаційні технології

УДК 004.02

**Пироженко Сергій Станіславович**
*студент*
*Харківського національного університету радіоелектроніки*

**Лєсна Наталя Совєтівна**
*кандидат технічних наук,*
*професор кафедри програмної інженерії*
*Харківський національний університет радіоелектроніки*

**ДОСЛІДЖЕННЯ ПРОДУКТИВНОСТІ МЕТОДІВ ТА ЗАСОБІВ КОНТРОЛЮ ГЛОБАЛЬНОГО СТАНУ КОМПОНЕНТІВ REACT ДОДАТКІВ**

***Анотація.*** *У дослідженні була розглянута продуктивність роботи внутрішніх функцій Redux, MobX та Context API, які, на поточний час, є основними засобами контролю глобального стану компонентів у React додатках.*

**Ключові слова:** *стан React додатку, глобальний стан, компонент, Redux, MobX, Context API, незмінність даних, редьюсер.*

**Проблема і актуальність дослідження.** Актуальність досліження методів та засобів контролю глобального стану React додатків зумововлена тим, що технології сучасної веб-розробки дуже стрімко розвиваються, постійно привносячи нові підходи, бібліотеки та фремворки. Втім, на сьогоднішній день, найпопулярнішим вибором серед фронт-енд розробників

усього світу є React.js. React дозволяє розробляти веб-сайти будь-якої складності й розміру завдяки створенню незалежних компонентів із можливістю повторного використання. При зростанні кількості останніх, React ставить питання щодо контролю передачі, оновлення та розподілення між ними доступу до даних. Зазвичай, це питання має бути вирішено на стадії проектування архітектури й помилкове рішення може привести до недостатньої продуктивності додатку, неправильної реалізації критичних моментів або зайвих витрат часу. У сучасному світі продуктивність веб-сайтів є критичним моментом, особливо при використанні їх на мобільних платформах, тому важливо використовувати підходи, які є найбільш ефективними.

**Огляд поточного стану об'єкта дослідження.** На сьогодні, основними підходами до контролю глобального стану, що конкурують, є бібліотека Redux, бібліотека MobX, а також вбудоване у React рішення – Context API.

Бібліотека Redux основана на архітектурі Flux та є найпопулярнішою серед конкурентів і за даними менеджеру пакетів npm використовується у більше ніж половині проектів. Бібліотека заохочує розробників використовувати певні архітектурні рішення, що веде до можливості контролю кожного кроку виконання й дозволяє швидке налагодження коду, навіть у проектах, які розробляють не перший рік. Це було достягнуто шляхом незмінності даних у глобальиих сховищах, що означає неможливість змінити будь-яке поле об'єкту за бажанням, а лише замінити весь об'єкт повністю. Втім, через такий підхід, багато розробників скаржаться на потребу написання великої кількості шаблонного коду, а також поступове погіршення продуктивності додатку з його ростом, що й послужило підставою для проведення даного дослідження.

Бібліотека MobX у свою чергу надає розробнику повний контроль за створенням архітектури додатку, а також можливість змінювати об'єкти та їх частини за своїм бажанням. Це приводить до зменшення розміру коду й, у теорії, підвищення продуктивності роботи React компонентів. Через такий підхід, у останні роки дана бібліотека постійно збільшує свою аудиторію.

Context API є вбудованим рішенням, яке не передбачає конкретного підходу до вирішення проблеми з глобальним станом компонентів, а лише надає інструменти для реалізації (деякі з яких використовуються в Redux та MobX).

Більшість проаналізованих досліджень, наприклад, [1] та [2], були націлені на взаємодію засобу контролю глобального стану з компонентами React, а не на продуктивність роботи самого засобу/методу.

**Об'єктом дослідження** є глобальне сховище React додатків.

**Предметом дослідження** є методи та засоби контролю глобального стану React додатків.

**Метою** даного досліджнення є аналіз продуктивності методів та засобів контролю глобального стану React додатків.

**Дослідження продуктивності внутрнішнього устрою Redux, MobX та Context API засобів.** За критерій продуктивності даного дослідження був взятий час у мілісекундах.

Для тестування були розроблені:

- node.js сервер з базою даних для агрегації та зберігання проміжної інформації тестів, а також для отримання тестових даних для компонентів у React клієнті;

- п'ять клієнтських додатків React для проведення тестування, у двух з яких була реалізована бібліотека Redux, ще у двох – MobX і в останньому –

Context API. Хоча була можлива реалізація усіх засобів у одному клієнському додатку, це може вплинути на точність результатів;

- клієнстський додаток для відображення й порівняння результатів тестів.

Кожен з клієнтських додатків для тестування використовує однакові дані й компоненти, у які були вбудовані різні засоби з контролю глобального стану відповідно до документації, що наведена у [3 - 5]. У кожному з додатків для тестування наявні: компонент для відображення списку, компонент для відображення окремих елементів списку та компонент з фільтрами для списку.

Тестуванню підлягають дві наступні дії:

- початкове завантаження даних з серверу з подальшим збереженням їх у глобальному сховищі та передання до компонентів. Ініціація дії виконується з головного компонента додатку після його відмальовування на сторінці;

- фільтрація даних, які знаходяться у глобальному сховищі. Ініціація дії виконується з компоненту фільтрів для списку.

Для тестування внутрішнього устрою засобів були виділені критичні точки для кожного з них. Для бібліотеки MobX – це час ініціації дії від компоненту та час фіксації дії (час, потрібний для отримання оновлених даних у компонентах). Для бібліотеки Redux при асинхронному завантаженні даних з серверу – це час від ініціалізації дії, викликаної компонентом, до обробки цієї дії бібліотекою Redux-Saga [6], яка використовується для асинхронних дій; час передачі дії від Redux-Saga до Redux редьюсера [7]; час від фіксації оновлених даних до отримання їх компонентом. Для бібліотеки Redux для синхронного оновлення даних, ми не використовуємо Redux-saga, тому вимірюються лише ініціація дії від компонента до Redux редьюсера та

час фіксації дії у зворотньому напряму. Для Context API вимірюємо час ініціалізації, час ініціалізації дії у Redux-подібній функції-редьюсері та час фіксації. Таким чином вимірюється лише час роботи інструментів самого засобу, виключаючи усі додаткові розрахунки й запити в мережі.

Для вимірювання часу виконання був розроблений сервіс, який отримує від компонентів та засобів контролю наступні дані: поточний засіб, критична точка, дія, стан (початок або завершення дії), а також час і розмір (кількість) заторкнутих даних. Час, який нижчий за 1мс округлюється до 1мс.

Для тестування початкового завантаження були протестовані вибірки з 20, 100 та 500 елементів. Кожен тестовий випадок було виконано 10 разів та отримано середнє значення. Тестування оновлення кількості елементів буде залежати від набору фільтрів. Кількість початково збережених елементів при оновленні буде дорівнювати 500 елементам.

На рисунку 1 наведено тестування базової реалізації Redux.

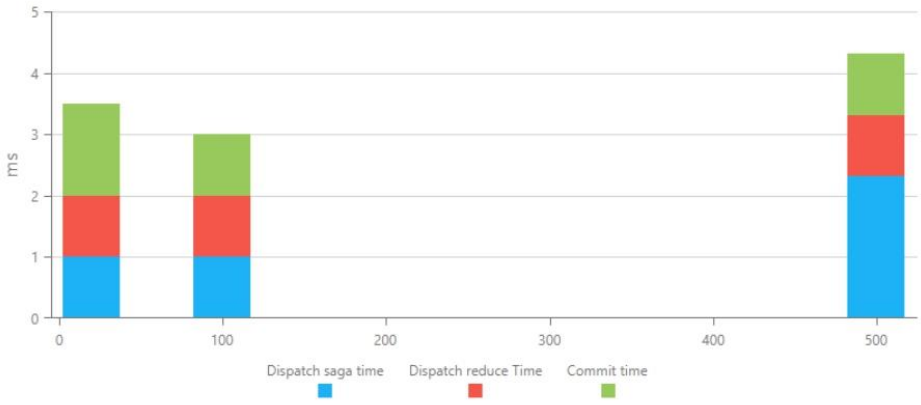



**Рис. 1. Тестування реалізації Redux №1. Завантаження даних**

Як видно з рисунку 1, при початковому завантаженні даних для Redux не має значення їх розмір. Після проведення 10 тестів для 20/100/500

елементів завантаження даних у Redux займає 3.5мс у середньому, що є несуттєвим значенням.

На рисунку 2 наведено результати тестування оновлення даних у базовій реалізації Redux.

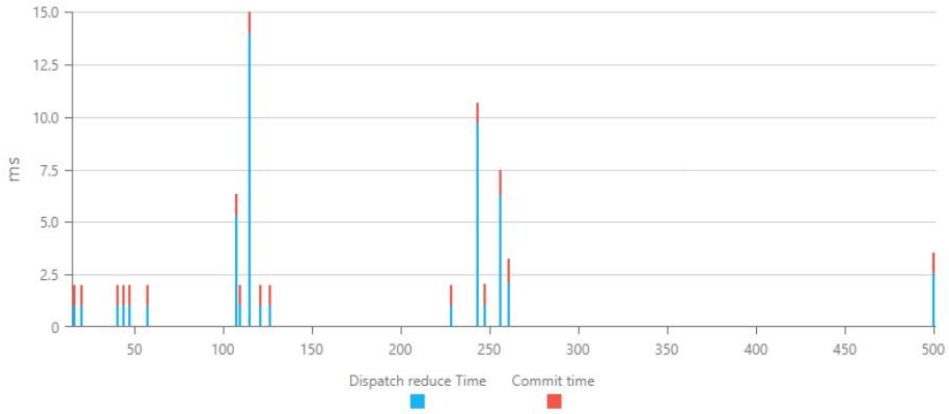



**Рис. 2. Тестування реалізації Redux №1. Оновлення даних**

Як видно з рисунку 2, у даному випадку залежності між кількістю елементів та часом виконання немає. Різні середні значення Dispatch обумовлені тим, що Redux вміє кешувати ініціювання однакових дій протягом деякого часу. Таким чином, якщо дія знаходиться у кешу, її виконання займає близько 1мс, у протилежному разі – від 10 до 18мс. Різниця у часі між різними вибірками пов'язана з різним інтервалом між їх виконанням.

Реалізація Redux №2 полягає у додаванні великої кількості редьюсерів, що ,за багатьма скаргами розробників, може знижувати швидкість роботи. Окрім редьюсерів для обробки завантаження та оновлення даних у реалізацію було додано 180 пустих, що, за словами авторів бібліотеки, не повинно вплинути на роботу.

Результати тестування при навантаженні та оновлені даних наведені на рисунках 3 и 4 відповідно.

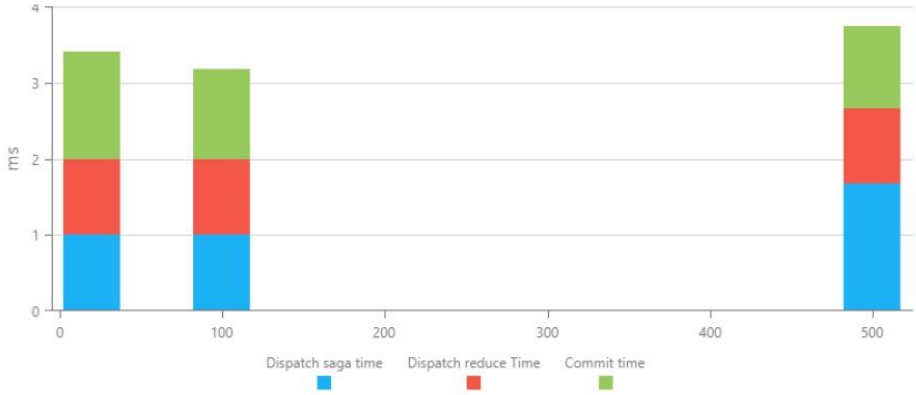



**Рис. 3. Тестування реалізації Redux №2. Завантаження даних**

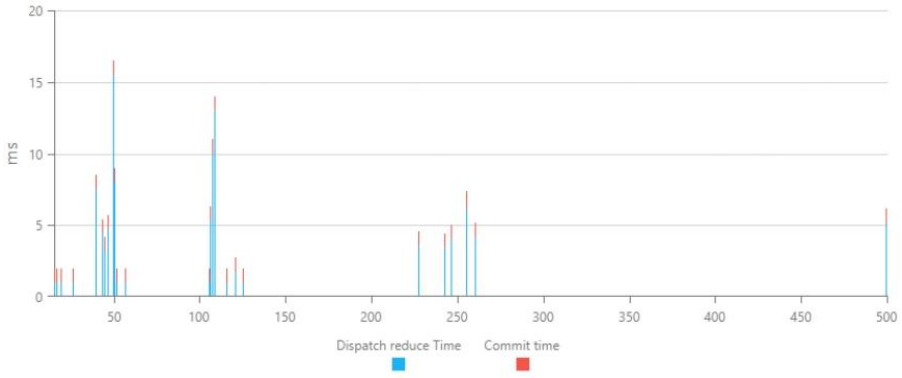



**Рис. 4. Тестування реалізації Redux №2. Оновлення даних**

Як видно з графіків, додавання великої кількості редьюсерів у проект не привело до суттєвих змін.

При реалізації сховища за допомогою бібліотеки MobX ініціювання дій є простим викликом функцій, тому час ініціації повинен залишатися постійним. Графіки тестування MobX при завантаженні та оновленні даних наведено на рисунках 5 та 6 відповідно.

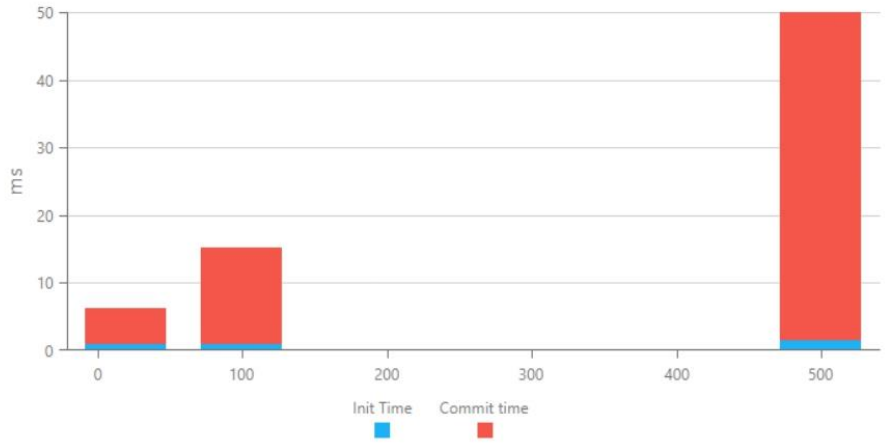



**Рис. 5. Тестування реалізації MobX №1. Завантаження даних**

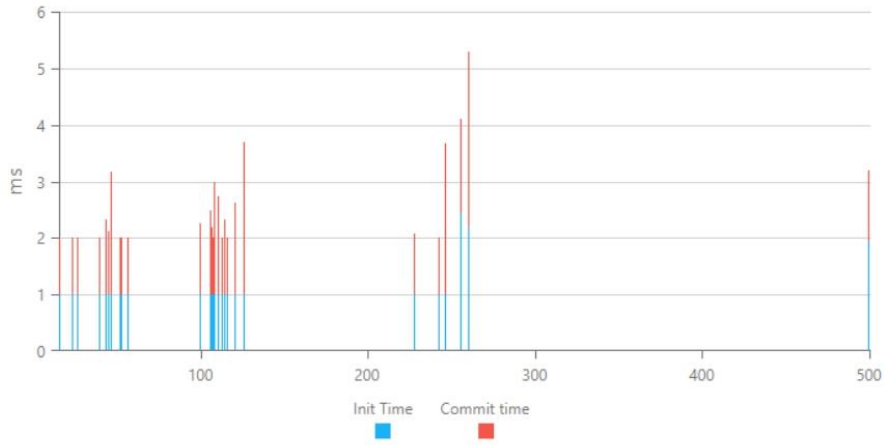



**Рис. 6. Тестування реалізації MobX №1. Оновлення даних**

Як видно з рисунку 5, при завантаженні даних час ініціалізації залишається незмінним, але час, що необхідний для передачі даних від сховища до компонента, значно зростає зі збільшенням кількості цих даних і досягає 50мс у середньому для 500 елементів списку, що може бути відчутною затримкою.

При оновленні даних, різниця у розмірі даних відрізняється не так суттєво, бо оновлюється лише один список даних в той час, коли при початковому завантаженні оновлювалися чотири списки (два для елементів списку і два для фільтрів). Утім, якщо будуть виконуватися послідовні оновлення з великою кількістю даних, ця різниця буде відчутною в порівнянні з Redux, який може кешувати дії і не залежить від розміру зберігаємих даних.

Реалізація MobX №2 полягає в переході від прямого оновлення списку за допомогою @observable [8] до підрахунку значення за допомогою @computed [9], який ініційований оновленням залежностей (фільтрів). Результати завантаження зображені на рисунку 7.

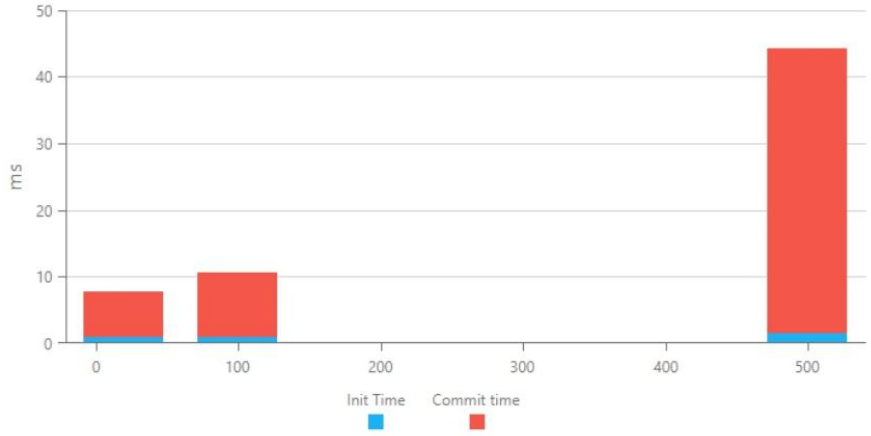



**Рис. 7. Тестування реалізації MobX №2. Завантаження даних**

Як видно з рисунку 7, у порівнянні зі звичайною реалізацією, ми маємо приблизно 15% прискорення, починаючи зі 100 елементів. Дане прискорення пов'язано з тим, що при використанні директиви @computed, ми маємо дві порівняно невеликі операції, замість однієї операції при якій усі необхідні дані оновлюються одразу.

Оновлення даних з директивою @computed зображено на рисунку 8.

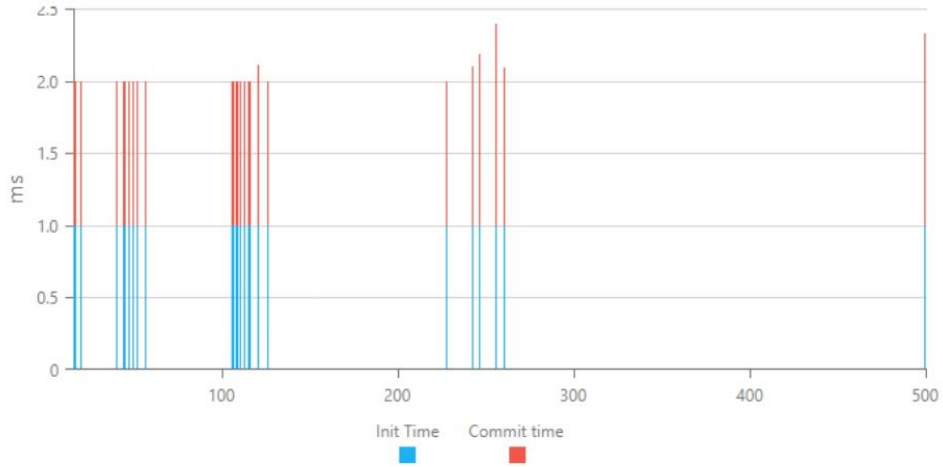



**Рис. 8. Тестування реалізації MobX №2. Оновлення даних**

З результатів, наведених на рисунку 8 можна зробити висновок, що перехід до @computed значень покращив продуктивність до двох разів. У більшості випадків, оновлення займає мінімальний можливий у тестуванні час – 1мс.

Слід зазначити, що при використанні директиви @computed особливу увагу потрібно приділити взаємодії з компонентом, який використовує дані.

Результати проведеного тестування при завантаженні даних в Context API наведено на рисунку 9.

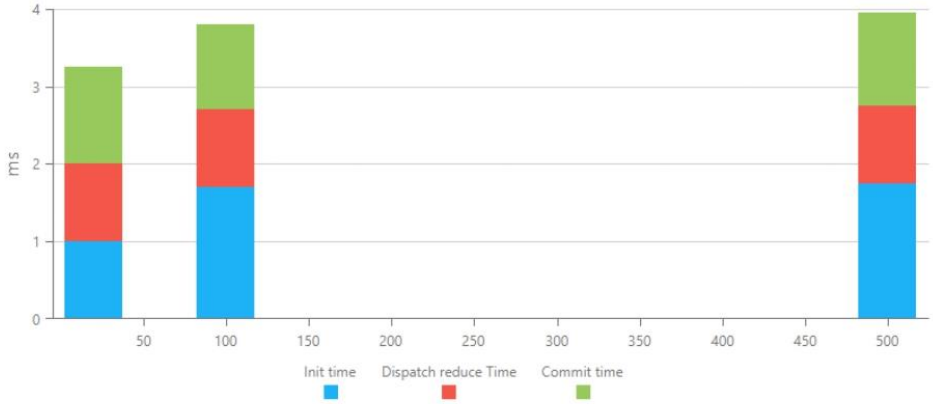



**Рис. 9. Тестування реалізації Context API. Завантаження даних**

Як видно з рисунку 9, поведінка Context API при завантаженні даних схожа на Redux й майже не залежить від розміру цих даних.

Результати тестування оновлення даних на Context зображено на рисунку 10.

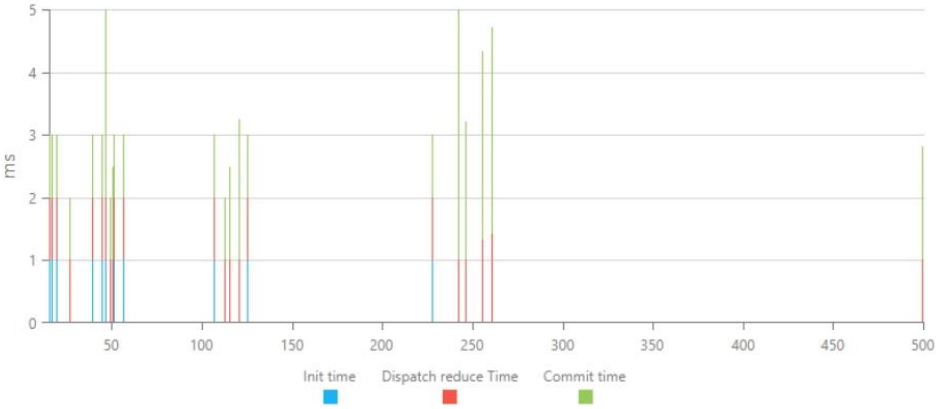



**Рис. 10. Тестування реалізації Context API. Оновлення даних**

Швидкість оновлення за допомогою вбудованого у React Context API близька до MobX реалізації й не погіршується при збільшенні розміру даних. Додатковою перевагою Context API є можливість створювати декілька сховищ з глобальними даними та унікальними для них функціями-

редьюсерами. Таке рішення дозволяє уникнути проблему бібліотеки Redux, яка має одне сховище та безліч редьюсерів у ньому, що приводить до постійної необхідності їх пошуку для обробки дії від компоненту.

**Висновки.** За результатами проведеного тестування, можна зробити висновок, що за продуктивністю внутрішнього устрою найкращим засобом є Context API. Це можна пояснити тим, що він є частиною самого React та не має жодних додаткових інструментів та розширень, які можуть сповільнювати роботу.

Бібліотека MobX у свою чергу, хоча й має хорошу продуктивність при роботі з невеликими об'ємами даних, особливо з @computed реалізацією, має суттєві проблеми при оновленні великих об'ємів інформації.

Бібліотека Redux, напроти, може добре працювати з будь-якими об'ємами інформації й чудово підходить до високочастотних оновлень, може мати проблеми при виклику різних функцій-редьюсерів, які ще не були закешовані. Але продуктивність, у такому разі, навряд чи буде критичною.

Слід зазначити, що проведені тестування показують лише роботу внутрішнього устрою засобів і не торкаються проблем взаємодії компонентів та глобальних сховищ, що потребує додаткових досліджень, тому Context API, який за своєю продуктивністю має значну перевагу, на практиці може бути недоречним рішенням для конкретного додатку або вимагатиме додаткових налаштувань компонентів.

**Література**


1. Carl Vitullo. Performance Profiling a Redux App. URL: https://dev.to/vcarl/performance-profiling-a-reduxapp-1k3b (дата звернення: 05.10.2020).
2. Steve Armstrong. React + Redux Performance and the Benchmarks to Prove It. URL: https://tech.smartling.com/react-redux-performance-and-the-benchmarks-to-prove-it-79b0bc9f25a4 (дата звернення: 06.10.2020).
3. Dan Abramov. React-Redux. Документація. URL: https://react-redux.js.org/ (дата звернення: 03.10.2020).
4. MobX. Документація. URL: https://mobx.js.org/README.html (дата звернення: 04.10.2020).
5. Context. Документація. URL: https://reactjs.org/docs/context.html (дата звернення 04.10.2020).
6. Redux-Saga. Документація. URL: https://redux-saga.js.org/ (дата звернення 11.10.2020)
7. Dan Abramov. Reducers. URL: https://redux.js.org/basics/reducers (дата звернення 06.10.2020).
8. MobX. Creating observable state. URL: https://mobx.js.org/observable-state.html (дата звернення 06.10.2020).
9. MobX. Deriving information with computeds. URL: https://mobx.js.org/observable-state.html (дата звернення 06.10.2020).

Г.2 Тези доповіді на міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті»

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ
XXIV МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

**«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ**
**У XXI СТОЛІТТІ»**

7 – 9 квітня 2020 р.

Том 6

**КОНФЕРЕНЦІЯ**
**«ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ»**

Харків 2020

Кудрявцева М. С. ..... 32, 34, 36
Кузьма В.Д. ..... 280
Кузьма Є.А. ..... 48
Кузьміна О.Г. ..... 132
Кулишова Н.Е. ..... 313, 327, 335
Купчик О.О. ..... 138
Курач А.І. ..... 58, 110
Куренков Б. М. ..... 46
Куценко О. С. ..... 345

**Л**

Лаврик М.О. ..... 106
Левикін В. М. ..... 62
Левикін В.М. ..... 60, 100, 122
Левикін І.В. ..... 319
Лещинська І.О. ..... 217, 221, 231
Лещинський В.О. ..... 227, 229, 233
Лєсна Н.С. ..... 193
Літвінов О.О. ..... 70
Ліхачов С.О. ..... 357
Логвінова К.В. ..... 23
Луніна К.О. ..... 355
Люліна К. П. ..... 207

**М**

Маврова К.Г. ..... 21
Мазурова О.О. ..... 153, 155, 157, 159
161, 163, 169, 173
Макаренко Г.М. ..... 403
Малькова И. А. ..... 50, 56
Малькова И.А. ..... 52
Малькова І.А. ..... 58, 66
Мальований Т.В. ..... 34
Мамонтов Ю.В. ..... 134
Мироненко М.Ю. ..... 64
Мирошниченко А.А. ..... 50
Михневич Т.К. ..... 153
Мищеряков Ю.В. ..... 280
Міхнов Д.К. ..... 112
Міхнова А.В. ..... 48, 110, 132
Міщеряков Ю.В. ..... 278, 294

**Н**

Незовибатько О. І. ..... 130
Непомняща І. В. ..... 157
Нерубацький В.В. ..... 357
Несміян Д.М. ..... 215
Новіков Ю.С. ..... 205, 207
Норець Д.А. ..... 144
Норець Ден. А. ..... 136
Носик А.М. ..... 253
Носик К.А. ..... 253

**О**

Ольховська В.О. ..... 68
Онищенко К.Г. ..... 175
Орленко К.А. ..... 359
Орлов О.К. ..... 237
Останіна В.Д. ..... 102
Остапенко О.О ..... 13

**П**

П'янов Р.В. ..... 185
Павленко М. Ю. ..... 116
Панфьорова І. Ю. ..... 78, 80, 118, 120, 144
Пашаєва С.М. ..... 173
Перетятко М.В. ..... 177
Петрова Р.В. ..... 290
Пильгуй К.А. ..... 266
Пироженко С.С. ..... 193
Погуляєв Ю. С. ..... 205
Полторацький А.О. ..... 165
Пономаренко А.С. ..... 361
Пономарьов С.К. ..... 175
Пономарьова О. В. ..... 409
Пономарьова С.В. ..... 409
Попович І.Д. ..... 169
Постельняк О.С. ..... 363
Потапов Г.І. ..... 303
Потєхін С.В. ..... 84
Проценко В.П. ..... 365

**Р**

Ревенчук І.А. ..... 9, 167
Редін Д.В. ..... 104
Редюхин Н.Н. ..... 290
Різниченко О.А. ..... 140
Рогач В.Д. ..... 32
Русаков К.О. ..... 367
Рябова Н.В. ..... 11, 15
Рябоконь Т.О. ..... 257

**С**

Садовников Б.І. ..... 161
Саенко В.И. ..... 116
Самофалов Л. Д. ..... 185, 247, 249, 251
Свиридова Ю.В. ..... 371
Свіргодська Т.В. ..... 94
Світенко Г. М. ..... 209

# КРИТЕРІЇ ВИБОРУ СТРАТЕГІЇ КЕРУВАННЯ ГЛОБАЛЬНИМ СТАНОМ REACT ДОДАТКІВ

Пироженко С.С.
Науковий керівник – к.т.н., проф. Лєсна Н.С.
Харківський національний університет радіоелектроніки
61183, Харків, вул. Роднікова 9, тел. (063)-473-14-83
Email: serhii.pyrozhenko@nure.ua

This paper discusses the problem of choice of the strategies of the global state management of the React application. The problem appears due to various architectures and libraries that can be used. Their choice directly affects both developing and further runtime of the website or mobile application. Furthermore, it's hard to shift from one library to another because it's tightly integrated with every React component and module making it time-consuming to fully change. That is why we need some criteria to follow to choose an appropriate library for the current project.

На сьогодні великого розвитку набули технології розробки веб-сайтів у вигляді single page application за допомогою мови програмування JavaScript. Це призвело до відсутності перезавантаження при переході між сторінками, але все ж таки має недолік - браузер отримує лише html без даних. На теперішній час, однією з найбільш популярних технологій, які використовують single page application підхід, є бібліотека React. Дана бібліотека використовує компонентний підхід розробки, що означає велику кількість незалежних компонентів, які можуть працювати разом, вбудовуватися один в один й мають різні функції, призначення та дані. Окрім розробки самих компонентів, перед розробником постає також й задача передачі даних між ними. Таку передачу необхідно виконувати у дві сторони – як від "батька" до "нащадка", так й у іншу сторону. Вбудована реалізація першого виконана дуже якісно, передача ж від "нащадка" до "батька" має в собі деякі недоліки, які призводять до написання великої кількості коду й у деяких випадках ведуть до втрати продуктивності й погіршення тестування додатку.

Через вищезазначені проблеми була створена значна кількість архітектур та підходів, таких як Flux, Redux, Mobx, Context API, Unstated. Усі вони спрощують передачу даних між компонентами й іноді надають інші переваги. Хоча кожна з бібліотек має свої переваги, їх використання не позбавлене проблем. Ці проблеми пов'язанні зі способами внутрішньої реалізації зберігання, зміни й передачі даних. Також слід зазначити, що перехід від одного підходу до іншого на пізніх стадіях розробки проекту майже неможливий, бо потребує великої кількості часу через необхідність зміни усіх компонентів, які використовують глобальні дані.

Щоб уникнути необхідність переходу від одного підходу до іншого на пізніх етапах розробки, є сенс роботи цей вибір більш свідомо, базуючись на деяких критеріях. Дані критерії часто пов'язані з нефункціональними

вимогами до програмного продукту. До основних критеріїв, які необхідно розглянути на старті проекту, слід віднести наступні:

– performance – вплив використання підходу на загальну продуктивність додатку або на продуктивність його окремих частин. В залежності від інструментів React та JavaScript, які були використані під час розробки підходу, його продуктивність може значно відрізнятися. Тому у багатьох випадках цей критерій стає основним під час вибору підходу, але слід також зазначити, що не для всіх типів додатків висока продуктивність є ключовою вимогою, у такому разі є сенс звернути увагу на інші моменти. Також на продуктивність бібліотеки впливають дані, їх тип, структура, яка використовується для їх зберігання, а також спосіб реалізації бібліотеки. Так знання бібліотеки й її правильна реалізація можуть компенсувати проблеми у продуктивності;

– scalability – вплив на масштабованість додатку під час його тривалої розробки й підтримки. Деякі з бібліотек, наприклад, Redux, передбачають використання задокументованих архітектурних підходів, в той час як багато інших, наприклад, MobX, дозволяють розробнику самому вирішувати як використовувати бібліотеку. Перший спосіб використовує більш суворий підхід, який хоча й накладає багато обмежень, сприяє поліпшенню масштабованості додатку при розробці великою командою протягом тривалого часу;

– testability – можливість як автоматичного так і ручного тестування додатку. В залежності від внутрішнього устрою бібліотеки й інструментів, які вона надає, можливість покриття додатку тестами, а також відловлювання помилок виконання суттєво відрізняється;

– compatibility – можливість використання з різними версіями React та іншими популярними бібліотеками. На цей критерій у першу чергу впливає команда розробки, яка супроводжує й оновлює бібліотеку. Так, при недостатніх ресурсах для підтримки бібліотеки, може втратитися стабільна робота у нових версіях React й з'явитися конфлікти з іншими бібліотеками у додатку;

– зручність використання з точки зору програміста. Даний критерій не має прямого впливу на роботу додатку, але значно впливає на комфорт розробки та її швидкість. Прикладами можуть бути: необхідність притримуватися правил, написання шаблонного коду та інше;

– інструменти розробки та розширення для бібліотеки.

Список літератури

1. Jonathan Saring. State management of React 2019 – URL: https://blog.bitsrc.io/state-of-react-state-management-in-2019-779647206bbc (Дата звернення 09.02.2020);

2. Sidath Asiri. Flux and Redux – URL: https://medium.com/@sidathasiri/flux-and-redux-f6c9560997d7. (Дата звернення 04.02.2020).

# ДОДАТОК Д

Електронні матеріали