КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Києво-Могилянська академія

Факультет Інформатики

Курсова робота

на тему:

Розробка веб-застосунку з використанням фреймворку React

студента III курсу

Спеціальності Комп’ютерні Науки

Грачова Олександра

Науковий керівник:

старший викладач, Борозенний

Сергій Олександрович

Завідувач кафедри

мультимедійних систем

доцент, кандидат наук. Жежерун

Олександр Петрович

Київ — 2023

Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Освітній ступінь бакалавр
Спеціальність 122 «Комп’ютерні науки»
Освітня програма бакалавр

ЗАТВЕРДЖУЮ
Завідувач кафедри мультимедіа
Жежерун О.П.
«13» вересня 2022 року

ЗАВДАННЯ
ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ
Грачову Олександру Віталійовичу

1. Тема роботи «**Розробка веб-застосунку з використанням фреймворку React**», керівник роботи Борозений Сергій Олександрович, старший викладач.
2. Строк подання студентом роботи «6» червня 2022.
3. План роботи:
   Анотація
   Вступ

   Розділ 1. Аналіз предметної області
   Що таке React і як все почалось
   Дослідження області

   Розділ 2. Аналіз предметної області
   Дослідження фреймворку React
   Дослідження інструментів розробки

   Розділ 3. Створення застосунку
   Написання гри
   Допоміжна функціональність

   Висновок
   Джерела

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№	Перелік робіт	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи	13 вересня			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	15 вересня 2022 – 1 жовтня 2022			
3.	Обрання розділів для курсової роботи	10 жовтня 2022 – 12 жовтня 2022			
4.	Написання кваліфікаційної роботи	15 вересня 2022 – 1 травня 2023			
	Розділ 1. Аналіз предметної області	23 листопада 2022			
	Розділ 2. Аналіз предметної області	3 січня 2023			
	Розділ 3. Створення застосунку	12 травня 2023			
5.	Завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами, подання науковому керівнику	15 травня 2023			
6.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності	15 травня 2023			
	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	Згідно з розкладом роботи ЕК			

Зміст

Анотація

React - це бібліотека для розробки інтерфейсів користувача веб-додатків, що створена Facebook. Його популярність зростає, оскільки він забезпечує простоту та швидкість розробки, підтримку компонентного підходу та переваги віртуального DOM.

У цій курсовій роботі ми досліджуватимемо основи React, включаючи створення компонентів, роботу з властивостями та станом, HOC та інше, що допоможе нам поглибитися в світ React та дізнатися, як створити веб-додатки, які будуть швидкими, масштабованими та добре організованими.

Вступ

Актуальність теми розробки веб-застосунку з використанням фреймворку React полягає у зростаючому попиті на модерні технології веб-розробки та зручності взаємодії з користувачами. Зокрема, у зв'язку з процесом повної цифровізації України, попит на сучасні веб-технології у нас зростає щодня. React - один з найбільш популярних та потужних фреймворків для створення сучасних веб-застосунків, що забезпечує високу продуктивність та ефективність розробки.

Однією з особливостей React є можливість створення компонентів, які дозволяють розбити веб-застосунок на невеликі, незалежні частини, що значно спрощує процес розробки та підтримки. Крім того, React забезпечує швидкий та зручний рендерінг веб-сторінок, що підвищує користувацький досвід.

Мета дослідження. Метою даної курсової роботи є розробка веб-застосунку з використанням фреймворку React та дослідження основних переваг та способів використання цього фреймворку.

Об’єкт дослідження. Об'єктом дослідження є процес створення веб-застосунку на базі React.

Предмет дослідження. Функціональні і класові компоненти, JSX, стани компонентів, хуки, Ajax, HOC.

Джерелами дослідження є електронні ресурси, які містять документацію та приклади використання фреймворку React, а також досвід використання цього фреймворку в сучасних веб-розробках.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Що таке React і як все почалось

React - це один з найпопулярніших JavaScript фронтенд фреймворків, який використовується для розробки веб-додатків та інтерфейсів користувача. React був представлений компанією Facebook у 2013 році, тоді як вони вирішували проблему зі складністю розробки великих додатків з великою кількістю динамічних елементів на сторінці. React пропонує новий підхід до розробки інтерфейсів, який базується на використанні компонентів, що дозволяє створювати складні інтерфейси з легкою перевикористовуваністю та підтримкою модульності.

Історія фреймворків JavaScript бере свій початок з середини 2000-х років, коли веб-розробка почала здійснюватися на клієнтській стороні за допомогою JavaScript. Початково, розробка веб-додатків здійснювалася на чистому JavaScript, що призводило до дублювання коду та погіршення підтримки.

У 2006 році був випущений фреймворк jQuery, який відіграв важливу роль у розвитку фреймворків JavaScript. Він був першим серйозним фреймворком JavaScript, який включав набір функцій для роботи з DOM та AJAX запитами, що значно спростило розробку веб-додатків.

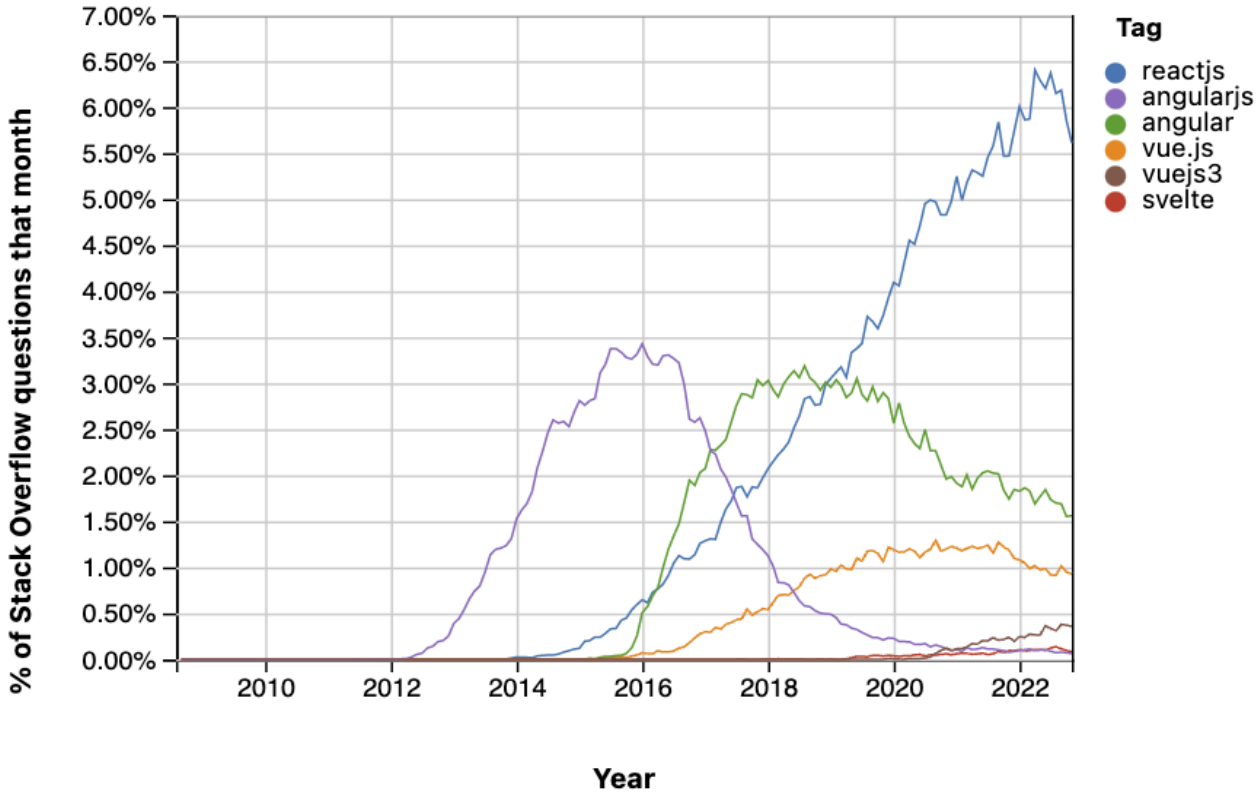
Після jQuery на ринку з'явилися інші фреймворки, такі як AngularJS від Google, Backbone.js та Ember.js. Вони надавали розробникам можливість створювати більш складні веб-додатки та робити розробку ефективнішою.

У 2013 році був представлений фреймворк React, який заснував новий підхід до розробки веб-додатків та інтерфейсів користувача використовуючи віртуальний DOM та компоненти, що дозволяє створювати більш ефективні та перевикористовувані інтерфейси користувача.

З часом на ринку з'явилися також інші фреймворки, такі як Vue.js, Svelte та інші, які продовжують розвиватися та створювати нові можливості для розробки веб-додатків та інтерфейсів користувача.

Дослідження області

Зацікавленість спільноти

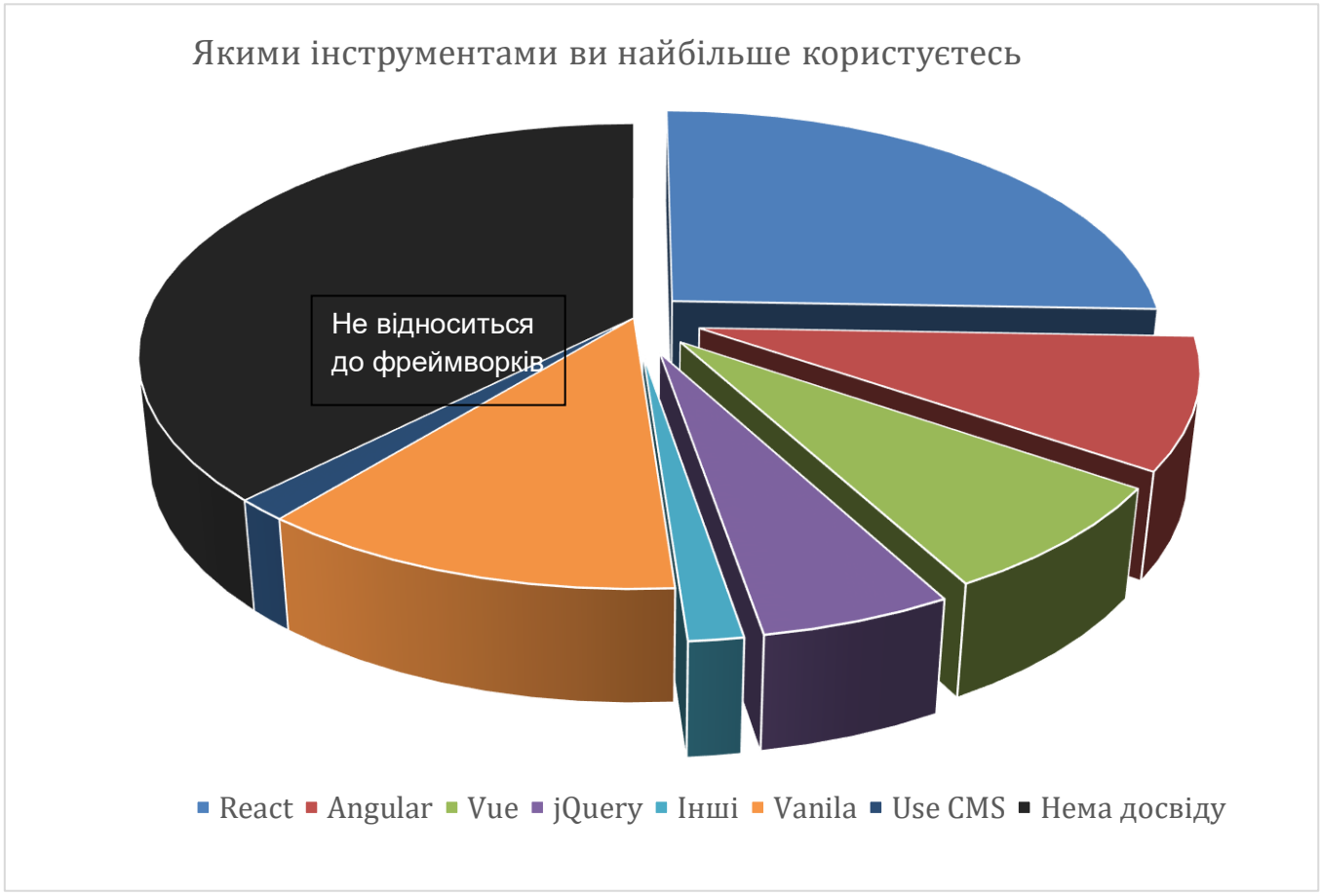


https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

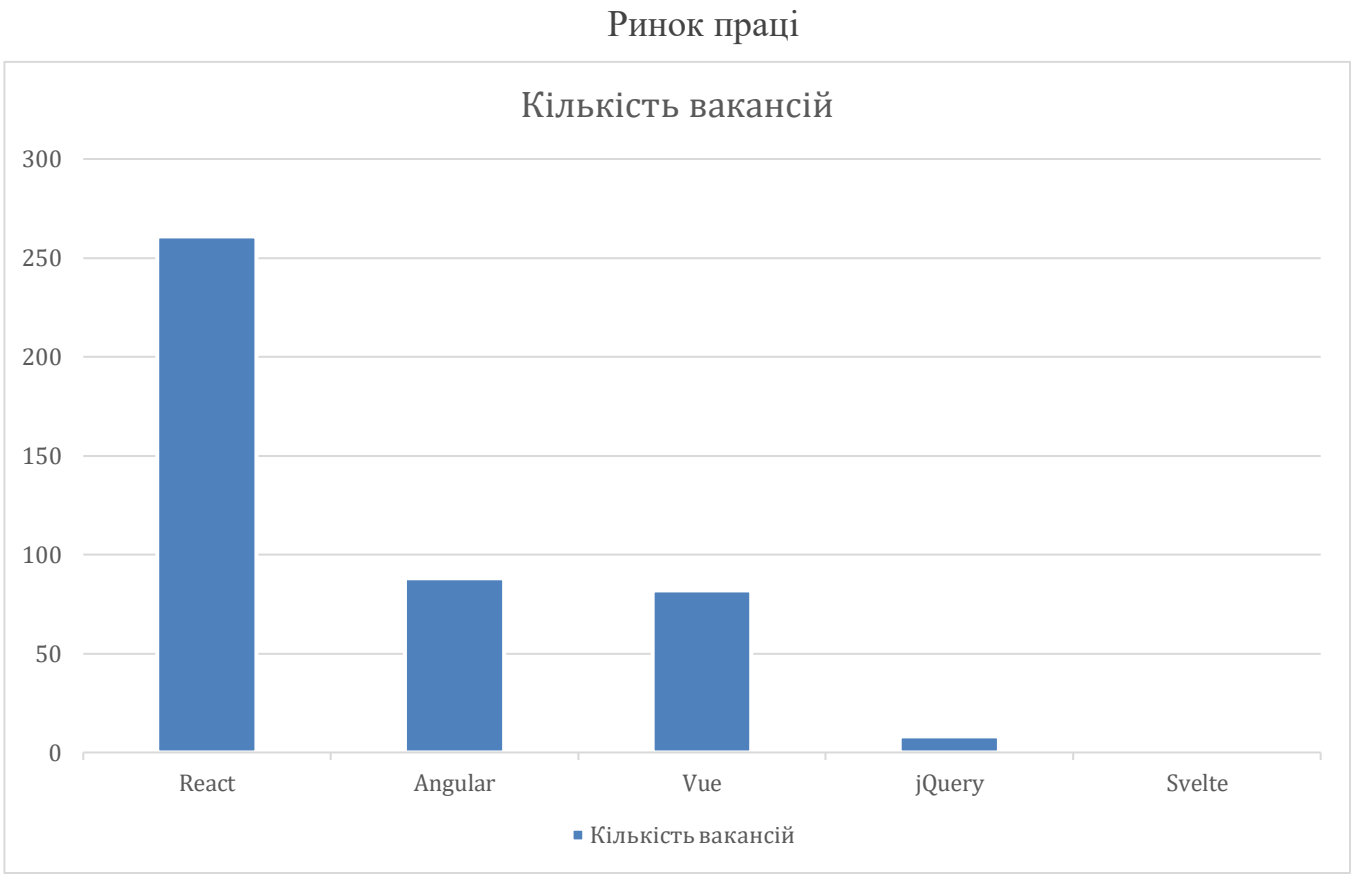
Зсилаючись на графік кількості питань stackoverflow по різних фреймворках, можна сказати, що, починаючи з 2014 року, реакт зазнає стрімкого зростання популярності. З роками React.js став вибором багатьох програмістів та компаній для створення сайтів різного масштабу. Він все ще залишається беззаперечним лідером у світі js фреймворків.

За даними опитування stackoverflow, в червні 2022 року React.js має показник популярності 42.62%, випереджаючи Angular, Vue.js, Angular.js та Svelte. Щодо улюбленості, React.js лідируючі позиції серед розробників, які віддають перевагу цій технології.

Порівнюючи з даними минулих років, можна помітити зростання популярності React.js. Наприклад, за минулий рік цей показник дорівнює 42.62%, а ще в 2019 році був 31.3% . Проте улюбленість React.js майже не змінилась за цей рік, а в лідерство вийшов відносно новий фреймворк Svelte.



Крім цього, мною було проведено власне опитування студентів факультету інформатики НаУКМА. Результати цього дослідження виявили, що серед студентів університету найбільшою популярністю користуються Реакт, Ангуляр та Вью. Ці фреймворки виявилися лідерами у відповідних категоріях, отримавши високий рівень визнання та зацікавленості серед студентської спільноти. Такі результати підтверджують, що ці фреймворки є важливими інструментами у сфері розробки програмного забезпечення та є домінуючими гравцями на ринку веб-технологій.



Більше того, після аналізу ринку праці на відомому сайті для пошуку роботи djinni.co я прийшов до висновку, що React має найбільшу кількість вакансій серед популярних фронтенд фреймворків. Це може свідчити про те, що React є одним з найбільш популярних фреймворків серед роботодавців, а

також про високу потребу в програмістах, які володіють навичками розробки за допомогою цього фреймворку.

У порівнянні з React, Angular та Vue мають більш менш подібну кількість вакансій. Ці два фреймворки також популярні серед роботодавців та можуть бути цікавими виборами для програмістів, які шукають роботу в області фронтенд розробки.

jQuery та Svelte мають найменшу кількість вакансій, що може свідчити про те, що ці фреймворки менш популярні серед роботодавців порівняно з React, Angular та Vue. Однак, для програмістів, які володіють навичками розробки за допомогою цих фреймворків, можуть бути цікавими вакансії, які попри невелику кількість можуть бути доступними.

Висновок

З огляду на наведені дані можна зробити висновок, що React.js є одним з найбільш популярних фронтенд фреймворків на сьогоднішній день, який зазнає стрімкого зростання популярності від 2014 року. Він займає перше місце за показником популярності серед інших фреймворків, таких як Angular, Vue.js, Angular.js та Svelte, а також займає перші місця у списках улюблених фреймворків серед програмістів.

Водночас, дані опитування студентів факультету інформатики НаУКМА підтверджують популярність React, Angular та Vue, серед студентської спільноти, а також високий рівень визнання та зацікавленості серед молодих розробників.

Дослідження ринку праці, зокрема вакансій на сайті djinni.co, підтверджують високу потребу в програмістах, які володіють навичками розробки за допомогою React.js, і свідчать про те, що цей фреймворк є одним з найбільш популярних фреймворків серед роботодавців.

Отже, на цей момент React.js можна вважати лідером на ринку фронтенд розробки, що виявляється в його популярності серед програмістів та роботодавців, високій кількості вакансій і визнанні від молодих розробників.

Однак, Angular та Vue.js також залишаються популярними виборами для розробки веб-додатків.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТЕХНІЧНОЇ ЧАСТИНИ

Переваги і недоліки

Давайте розглянемо переваги і недоліки Реакту:

Переваги:

- Ефективність: React працює з віртуальним DOM, що дозволяє зменшити кількість звернень до реального DOM і покращити продуктивність додатка.
- Компонентна архітектура: React побудований на основі компонентів, що дозволяє легко організувати код та реорганізовувати його.
- Масштабованість: React дозволяє легко масштабувати додатки за допомогою створення більш складних компонентів.
- Розширюваність: React дозволяє використовувати більшість сторонніх бібліотек та пакетів.
- Підтримка: React має широку спільноту розробників та багато ресурсів для вивчення.

Недоліки:

- Навчання: React може бути складним для вивчення, особливо для початківців, які не мають досвіду у роботі з JavaScript.
- Несумісність зі старим кодом: React не завжди працює зі старим кодом, що може вимагати переписування деяких частин додатку.
- Великий розмір: Розмір додатку на React може бути більшим, ніж на інших фреймворках, що може призвести до погіршення продуктивності.

Дослідження фреймворку React

Компоненти

Основою фреймворку ReactJS є компоненти. Це основні будівельні блоки веб-інтерфейсу, які дозволяють розбити великі та складні UI (користувацькі інтерфейси) на більш малі та керовані частини. Компоненти - це незалежні

вузли, які мають властивості (props) та стан (state), і можуть бути вкладені один в одного для створення складних ієрархій інтерфейсів.

Компоненти в Реакті можуть бути класовими або функціональними. Класові компоненти використовують класи ES6 для оголошення та управління компонентом, в той час як функціональні компоненти - це просто функції JavaScript, які приймають властивості в якості аргументів та повертають віртуальний DOM (Virtual DOM) для рендерингу.

Компоненти в Реакті дозволяють створювати ефективний, модульний та перевикористовуваний код, який полегшує розробку складних веб-додатків. Вони дозволяють розділити UI на окремі компоненти, керувати їх станом, передавати властивості між компонентами та оновлювати інтерфейс відповідно до змін в стані. Компоненти є основним поняттям в Реакті та дозволяють розробникам побудувати потужні та гнучкі веб-додатки.

Життєвий цикл компоненту зазвичай поділяють на 3 фази:

1. Монтування
2. Оновлення
3. Демонтування

Монтування є етапом створення компоненту. Розглянемо цей процес:

1) При створенні компоненту спочатку викликається метод *constructor()*, що встановлює стани та успадковує батьківські методи.

2) Далі йде необов’язковий метод *getDerivedStateFromProps()*, що встновлює змінні стану в залежності від “пропсів”.

3) Після цього активується обов’язковий метод *render()*, що просто повертає HTML для рендерингу на сторінці.

4) І в кінці викликається теж необов’язковий метод *componentDidMount()*, що викликає код, який потребує, щоб компонент вже був промальований (наприклад, зміна кольору через певний час).

Оновлення відбувається при кожній зміні props або state. Розглянемо процес оновлення:

1) Спочатку (якщо є) викликається метод *getDerivedStateFromProps()*, який може оновлювати стани компоненту.

2) Після цього йде *shouldComponentUpdate()*, який повертає булеан значення - треба чи не треба рендерити компонент (по стандарту - *true*).

3) Йде перемальовування компоненту методом *render()* (якщо минулий метод повернув *true*).

4) Далі відпрацьовує метод *getSnapshotBeforeUpdate()*, де можна побачити та використати значення стану та пропсів ще до оновлення, але при цьому обов'язково треба визначити метод *componentDidUpdate()*.

5) *componentDidUpdate()* викликається наприкінці оновлення зі станом та "пропсами" вже оновленого компоненту.

Демонтування є процессом видалення (із виду) компонента.

1) Цей процес включає себе лише небов'язковий 1 метод - *componentWillUnmount()*, в якому можна визначити якийсь додатковий функціонал.

Функціональні та класові компоненти

Як я вже сказав, реалізація компоненту в основному поділяється на 2 типи: функціональну(функційну) та класову. Вважається, що використання класів є легасі та вже майже не використовується. Розглянемо обидва варіанти.

Класові компоненти є старішим підходом до створення компонентів в Реакті та використовують класи ES6 для оголошення. Ми маємо наслідуватися від *React.Component* та таким чином отримуємо доступ до функцій класу *Component*, а також це зобов'язує нас використати функцію *render()*, що відповідає за відображення компонента на віртуальному DOM.

Приклад коду:

```
import React, { Component } from 'react';

class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
```

```
    count: 0
  };
 }

 incrementCount = () => {
  this.setState(prevState => ({ count: prevState.count + 1 }));
 }

 render() {
  const { count } = this.state;
  return (
   <div>
    <h1>Мій компонент</h1>
    <p>Лічильник: {count}</p>
    <button onClick={this.incrementCount}>Збільшити</button>
   </div>
  );
 }
}


export default MyComponent;
```

У цьому прикладі ми створили класовий компонент MyComponent, який розширює базовий клас Component із бібліотеки React. Він має внутрішній стан з полем count, яке ініціалізується значенням 0 у конструкторі. Компонент також має метод incrementCount, який збільшує значення count на 1 при кожному кліку на кнопку. Значення count відображається в рендері компонента, а кнопка має обробник події onClick, який викликає метод incrementCount. Коли стан компонента змінюється за допомогою setState, React автоматично оновлює відображення компонента на сторінці.

Функціональні компоненти є новішим підходом в Реакті та використовують прості функції JavaScript для оголошення компонента. Вони приймають властивості (props) в якості аргументів та повертають віртуальний DOM. Для цього методу треба набагато менше місця в коді і він легше для сприйняття. Хоча і раніше з цим підходом ми позбавлялись певного функціоналу класових компонентів, проте в останніх версіях додали хуки в обхід цього недоліку, що я розгляну далі.

Приклад функціонального компоненту:

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

const MyComponent = () => {
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(prevCount => prevCount + 1);
  }

  return (
    <div>
      <h1>Мій компонент</h1>
      <p>Лічильник: {count}</p>
      <button onClick={incrementCount}>Збільшити</button>
    </div>
  );
}

export default MyComponent;
```

У цьому прикладі ми використали функціональний компонент MyComponent замість класового компонента, і використали хук useState для встановлення та управління станом компонента. Замість конструктора, ми використовуємо деструктуризацію масиву, щоб отримати поточне значення count та функцію setCount для оновлення стану. Метод incrementCount залишився незмінним, як і кнопка з обробником події onClick. Коли функція setCount викликається з новим значенням, React автоматично оновлює відображення компонента з новим значенням count.

Оптимізація

Коли мова йде про оптимізацію додатку в Реакті, використання класових компонентів може бути менш ефективним порівняно з функціональними компонентами з введенням хуків (hooks) в React 16.8+.

Ось кілька причин:

- Надмірна складність: Класові компоненти мають більше синтаксичного цукру та більше коду, що може зробити їх більш складними для розуміння та підтримки. Вони вимагають визначення конструктора, використання this в методах, та можуть мати більшу кількість життєвих циклів, що може бути складним у використанні та налагодженні.
- Загальна продуктивність: Функціональні компоненти з введенням хуків (hooks) можуть мати кращу продуктивність порівняно з класовими компонентами. Це через те, що вони мають менше внутрішніх операцій та легше оптимізовані відтворення (re-rendering) компонентів, що може впливати на продуктивність додатку.
- Що стосується "kept alive" (також відомого як "memoization") - це підхід, який дозволяє зберігати кеш (пам'ять) результатів обчислень для попередньо виконаних операцій, що може допомогти знизити виконання зайвих обчислень та оптимізувати продуктивність додатку. З введенням хуків в React 16.8+, можна використовувати useMemo та useCallback, які дозволяють досягнути подібного ефекту в функціональних компонентах.

Отже, функціональні компоненти з введенням хуків вважаються більш сучасним та рекомендованим підходом в Реакті, який може допомогти досягти кращої продуктивності та оптимізації додатку порівня

JSX

Також важливо розглянути jsx (надробку над javascript для зручного опису UI) та процес переходу jsx в html, який оброблюється та промальовується браузером.

Jsx виглядає як мова розмітки, проте вона має повний функціонал чистого javascript. Приклад:

```
const name = 'Alexander ';
const surname = 'Hrachov';
const element = <div><h1>Hello, {name+surname}</h1></div>;
```

Як ми бачимо, теги jsx можуть містити дочірні елементи всередині себе, а також ми можемо вставляти будь-який js код, що потім буде обраховуватися при компіляції.

Під капотом

JSX використовується в бібліотеці React для опису відображення користувацького інтерфейсу в компонентах. Проте, браузер не може розуміти JSX, оскільки це не валідний JavaScript. Тому, перед тим як бути виконаним, код JSX потребує транспіляції в стандартний JavaScript. React використовує внутрішній процесор Babel, який автоматично транспілює код JSX на еквівалентний JavaScript. Під капотом, JSX використовується для створення віртуального DOM (Virtual DOM) - це внутрішнє представлення DOM-структури, яке використовується React для оптимізації рендерингу компонентів. Коли компонент React, що містить JSX, рендериться в браузері, Babel транспілює код JSX на валідний JavaScript, що використовується браузером. Приклад компонента з JSX:

```jsx
const MyComponent = () => {
  const name = 'John';
  const age = 30;

  return (
    <div>
      <h1>Привіт, {name}!</h1>
      <p>Тобі {age} років.</p>
    </div>
  );
```

```
}
```

Транспілюється в наступний валідний JavaScript:

```
const MyComponent = () => {
  const name = 'John';
  const age = 30;

  return React.createElement(
    'div',
    null,
    React.createElement(
      'h1',
      null,
      'Привіт, ',
      name,
      '!'
    ),
    React.createElement(
      'p',
      null,
      'Тобі ',
      age,
      ' років.'
    )
  );
}
```

Цей код створює еквівалентний віртуальний DOM за допомогою React.createElement і рендерить його в реальний DOM у браузері. Цей процес транспіляції та створення віртуального DOM відбувається автоматично за кулісами React, що дозволяє розробникам використовувати JSX для зручного опису відображення компонентів.

Стани компонентів

Стани (states) є одним з важливих понять в React і використовуються для збереження та управління даними в компонентах. Стан (state) - це об'єкт, який містить дані, які можуть змінюватися в процесі виконання компонента. Кожен компонент може мати свій власний стан, що дозволяє керувати локальними

даними та їх змінами в межах компонента без впливу на інші компоненти. Для використання стану в компонентах React, спочатку потрібно визначити початковий стан за допомогою useState хука або класового компонента.

Присвоювання нового значення стану відбувається через метод *this.setState*. Палкою в колеса є їх асинхронне оновлення. Тобто, при використанні стейту під час обрахування нового, нам варто застосувати аргументи функції: *this.setState((state, props) =>...* Таким чином ми отримуємо минуле значення станів та на основі них обраховуємо нові.

Приклад використання стану за допомогою useState хука:

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

const MyComponent = () => {
  // Визначаємо початковий стан за допомогою useState хука
  const [count, setCount] = useState(0);

  // Функція-обробник події для зміни стану
  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Лічильник: {count}</p>
      <button onClick={increment}>Збільшити</button>
    </div>
  );
};
```

У цьому прикладі ми використовуємо useState хук для визначення стану count зі значенням 0 та функцією setCount, яка дозволяє змінювати значення стану. Потім ми використовуємо цей стан в рендері компонента та встановлюємо функцію increment як обробник події для зміни стану при кліку на кнопку. Стани дозволяють компонентам React реагувати на зміни в даних та перерендерюватися відповідно до нових значень стану. Вони дуже корисні для управління динамічними даними в компонентах, такими як форми, взаємодія з користувачем, асинхронні запити тощо.

Хуки

Хуки (hooks) є функціями, які введені в React з появою версії 16.8 і дозволяють використовувати стан і функціональні можливості React в функціональних компонентах без використання класових компонентів. Хуки використовуються для взаємодії з життєвим циклом компонента, управління станом, роботи з контекстом, та виконання побічних ефектів. Декілька найпоширеніших хуків в React:

- useState: дозволяє використовувати стан в функціональних компонентах. Він приймає початкове значення стану і повертає масив з поточним значенням стану та функцією для його оновлення
- useEffect: дозволяє виконувати побічні ефекти, такі як запити до API, підписки на події, оновлення DOM, після рендерингу компонента. Він виконується після того, як компонент був відображений на екрані та при зміні залежностей, вказаних в масиві залежностей
- useContext дозволяє отримувати значення з контексту в компонентах, які вкладені в провайдер контексту. Він приймає контекст, створений за допомогою React.createContext, і повертає поточне значення контексту
- useReducer дозволяє використовувати підхід на основі reducer для управління станом в компонентах. Він приймає функцію reducer, початковий стан та функцію диспетчера (dispatch), яка дозволяє оновлювати стан за допомогою дій
- Хуки useMemo та useCallback дозволяють оптимізувати рендеринг компонентів шляхом кешування результатів обчислень або функцій, що передаються в пропси компоненту. Вони повертають

Більш того, є інші хуки для використання частини функціоналу Redux, оптимізації роботи додатку та інших дрібниц, проте вони є менш поширеними та використовуваними в багатьох проектах, тому включення їх в дослідження є необов'язковим

Ajax

Ajax (**A**synchronous **J**avaScript **A**nd **X**ML) - підхід до побудови веб-додатків з фоновим обміном даних з веб-сервером для . Зазвичай для *Ajax* в *React* використовується асинхронний http-клієнт *axios*.

Для створення запиту використовується метод класу axios *get\post\put\...*

Приклад: *axios.get(`https://my-api/users`)* як результат ми отримаємо об’єкт відповіді з такими полями:

data: {} - тіло запиту (тобто основні дані, що повернув сервер)

status: 200 - статус відповіді

statusText: 'OK' - текст статусу відповіді

headers: {} - хедери, що надіслав сервер с запитом

config: {} - конфіг axios, що був використаний при запиті

request: {} - початковий запит, що отримав від нас сервер

Далі, після отримання відповіді, ми можемо запустити componentDidUpdate з вже оновленими даними і реакт перемалює певні місця DOM-дерева, не перезавантажуючи сторінку

HOC

Higher-Order Components (HOC) - це високорівневий патерн в React, який дозволяє повторно використовувати логіку компонентів та забезпечує зручний спосіб для впровадження взаємодії між компонентами без використання вкладених компонентів.

HOC - це функції, які приймають компонент React як аргумент і повертають новий компонент. Новий компонент може мати додаткові властивості, стани або методи, які додаються через логіку, визначену в HOC.

Приклад:

```
const withLogger = (WrappedComponent) => {
  return class extends React.Component {
    componentDidMount() {
      console.log('Component has been rendered:', WrappedComponent.name);
    }

    render() {
      return <WrappedComponent {...this.props} />;
    }
  };
};

const MyComponent = (props) => {
  // Вміст компоненту
  return (
    <div>
      {/* Розмітка компоненту */}
    </div>
  );
};

// Використання HOC
const MyComponentWithLogger = withLogger(MyComponent);
```

Основні переваги використання HOC включають:

- Реюзабельність: ми можемо використовувати одну й ту ж логіку на кількох компонентах. Це дозволяє уникнути дублювання коду та забезпечує більшу реюзабельність логіки.

- Розширюваність: HOC дозволяє легко розширити функціональність компонентів, додаючи додаткові властивості, стани або методи до компонентів, не змінюючи їхньої реалізації.
- Розділення відповідності: можна відокремити відповідність різних аспектів компонента, таких як логіка взаємодії зі сторонніми сервісами, логіка авторизації або робота з даними, від логіки відображення компонента. Це допомагає зберегти код чистим та зручним для тестування.

Недоліки використання HOC в React:

- Загромадження коду: Використання HOC може призвести до зайвого загромадження коду, особливо якщо використовується кілька HOC одночасно.
- Проблеми зі зрозумілістю коду: Використання HOC може зробити код складнішим для розуміння, особливо для новачків у React.
- Проблеми з передачею пропсів: Деякі проблеми можуть виникнути при передачі пропсів через кілька рівнів HOC, що може зробити код складним для налагодження та підтримки.

Отже, хоча HOC і дарує набір корисних можливостей для написання кращого коду, проте варто враховувати, HOC має свої обмеження та може призвести до зайвої вкладеності компонентів, що може бути важко для розуміння та підтримки коду. Окрім того, з випуском React Hooks, деякі сценарії використання HOC можуть бути замінені на використання Hooks, таких як useEffect та useContext та, не дивлячись на усі її плюси, вона вважається застарілою.

Дослідження інструментів розробки

Редактору коду

Вибір редактора коду для написання коду на React залежить від вашого особистого вподобання та потреб. Мій вибір пав на Visual studio code (VS code). Плюси VSCode для розробки на React:

- Підтримка: VSCode має вбудовану підтримку для React, включаючи автодоповнення, перевірку синтаксису, розпізнавання JSX та багато інших корисних функцій, що допомагають спростити розробку в React.
- Розширення та плагіни: В редакторі є велика кількість розширень та плагінів, що можуть бути встановлені для розширення функціональності редактора. Наприклад, розширення "Reactjs code snippets" надає набір коротких кодових фрагментів для React, що спрощує написання коду.
- Інтеграція з Git: Застосунок має вбудовану підтримку Git, що дозволяє зручно керувати версіями коду та співпрацювати з іншими розробниками у командному проекті.
- Зручність: VSCode має чистий та зручний інтерфейс, з багатьма налаштуваннями та можливостями налаштування розробчого середовища під мої потреби.
- Багатомовність: Наявність підтримки багатьох мов програмування та вбудовані інструменти для розробки в різних мовах, що дозволяє зручно працювати з багатомовними проектами.
- Швидкість: Редактор вважається lightweight. Тобто усі команди, гарячі клавіши, переходи працюють миттєво. Це особливо помітно, коли знову починаєш користуватись IDE від JetBrains або Visual Studio (не Code)

Npm

Говорячи про Javascript, важко обійти стороною npm (**N**ode **P**ackage **M**anager). Це є стандартний менеджер пакетів для Node.js (як не очевидно). Воно використовується для управління залежностями та установки сторонніх пакетів в проектах JavaScript за допомогою команди npm install <назва бібліотеки>. Залежності поділяються на devDependencies та dependencies (при завантаженні першим потрібен флаг -dev і вони необхідні саме під час разробки). Їх списки та їхні версії зберігаються в файлі package.json. В той же час, самі файли бібліотек зберігаються в директорії node_modules та мають структуру дерева (одні бібліотеки можуть мати node_modules зі своїми залежностями).
Звісно, свої власні проекти також можна завантажувати в хмару за допомогою команди npm publish.

Плюси використання npm:

- Велика кількість пакетів: npm має найбільшу екосистему пакетів серед менеджерів пакетів JavaScript. Можна знайти велику кількість готових пакетів, які можна використовувати в своєму проекті React, таких як різні бібліотеки, компоненти, розширення та інструменти.
- Простота використання: npm має простий та зрозумілий інтерфейс командного рядка, що дозволяє легко встановлювати, оновлювати та видаляти пакети. Ви також можете використовувати npm для керування версіями пакетів, що спрощує роботу зі залежностями вашого проекту.
- Інтеграція з Node.js: Як я вже казав, npm є стандартним менеджером пакетів для Node.js, що означає, що він ідеально підходить для розробки на React, який базується на Node.js. Ви можете використовувати npm для встановлення та управління пакетами, які використовуються на серверному боці та на клієнтському боці вашого додатку React.

Мінуси npm:

- Версійні конфлікти: При використанні багатьох пакетів можуть виникати конфлікти версій, особливо коли одні пакети вимагають різних версій одного й того ж другого пакету. Це може створити складності та проблеми в розробці та підтримці проекту.
- Безпека: Використання сторонніх пакетів може вносити певний рівень ризику з точки зору безпеки. Не всі пакети є добре забезпеченими та актуальними, тому слід ретельно вибирати пакети та періодично оновлювати їх, щоб уникнути потенційних вразливостей у вашому проекті.
- Залежності: Використання багатьох пакетів може призвести до великої кількості залежностей в вашому проекті, що може збільшити розмір та складність проекту. Керування залежностями може бути складним, особливо в великих проектах, тому потрібно це враховувати.
- Якість пакетів: Не всі пакети в npm мають високу якість та добре покриття тестами. Деякі пакети можуть бути менш популярними або неактивно підтримуватися, що може призвести до проблем з функціональністю або надійністю вашого проекту.
- Неоднорідність: npm має велику кількість пакетів різних авторів, що може викликати неоднорідність в структурі, якості та стилях коду в пакетах. При використанні багатьох різних пакетів, ваш код може мати різні стилі та конвенції, що може вплинути на читабельність та обслуговування проекту

Загалом, npm є потужним та широко використовуваним менеджером пакетів для розробки на React та інших JavaScript-проектах. Проте, варто усвідомлювати певні ризики та використовувати його з розумінням, ретельно вибираючи та оновлюючи пакети, а також дотримуючись найкращих практик безпеки та якості

Typescript

Також не можна не згадати і typescript (типізований javascript). Використання цієї мови програмування дає всі плюси типізованих мов програмування та майже не обмежує самого програміста, якщо він все ще хоче писати чистим javascript (зворотна сумісність). Його встановлення відбувається за допомогою npm команди *npm install --save typescript @types/node @types/react @types/react-dom @types/jest.* Також усі .js файли мають бути переіменовані на .ts . Після вивчення та використання тайпскрипту в професійній та навчальній діяльності я можу зазначити такі переваги його використання:

- Статична типізація: TypeScript дозволяє використовувати статичну типізацію, що допомагає виявляти помилки на етапі компіляції, забезпечує більшу безпеку та надійність коду. Можна визначати типи для змінних, функцій, параметрів, повернутих значень, та інших елементів коду.
- Гнучкість: Можливість поступового переходу з JS на TS за допомогою гнучкого налаштування конфігу для typescript
- Підтримка нових функцій JS: TypeScript є розширенням JavaScript, тому він підтримує всі його стандартні функції, а також нові функції, які ще не ввійшли в стандарт JavaScript. Ви можете використовувати класи, стрілкові функції, асинхронні функції, декоратори та багато інших функцій, що допомагають покращити продуктивність та ефективність розробки.
- Засоби рефакторингу: Присутній великий набір інструментів, таких як автодоповнення, перейменування, вилучення, вилучення параметра та багато інших, що допомагає забезпечити більшу продуктивність та точність при розробці.
- Велика спільнота: TypeScript має велику та активну спільноту розробників, що забезпечує багату ексосистему.
- Інтеграція з інструментами розробки: TypeScript має добру інтеграцію з рядом інструментів розробки, таких як Visual Studio Code, WebStorm,

Sublime Text та інші, що допомагає забезпечити зручний та продуктивний розробний процес.

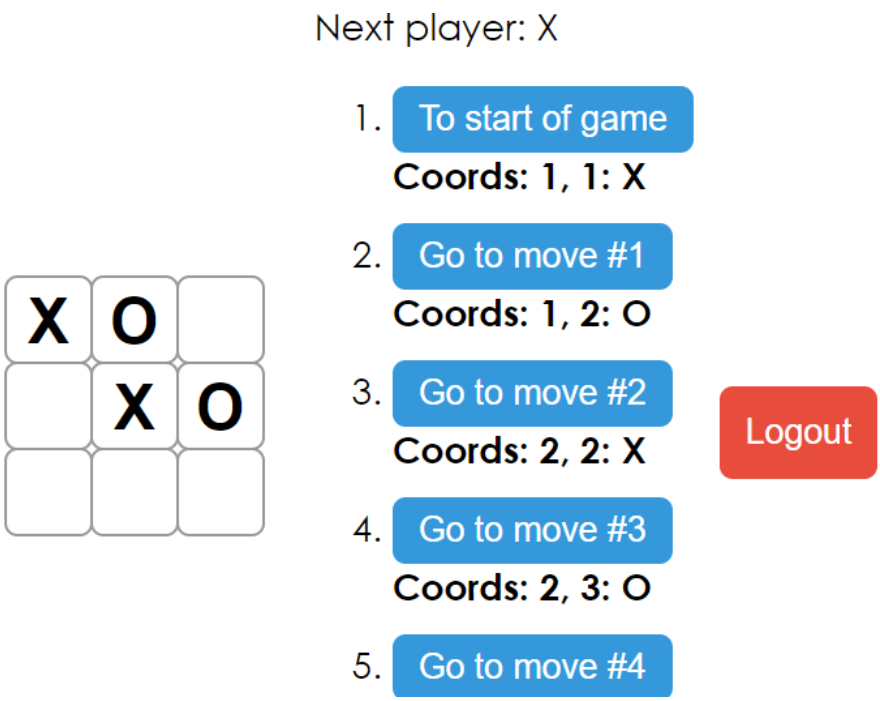
Щодо мінусів:

- Вивчення кривої: Хоча TypeScript базується на JavaScript, він також має свої власні функції та синтаксис, що може потребувати додаткового вивчення, особливо для початкових розробників, які знайомі тільки з JavaScript.
- Додатковий крок компіляції: TypeScript код потребує компіляції в JavaScript перед виконанням в браузері, що додає незручний крок в розробний процес.
- Обмежена підтримка сторонніх бібліотек: Хоча TypeScript має широку підтримку стандартних JavaScript бібліотек, вона може бути обмежена для деяких сторонніх бібліотек, особливо тих, які не оновлюються регулярно.
- Додатковий обсяг коду: Використання статичної типізації та інших функцій TypeScript може призвести до більшої кількості коду, проте більш читабильного.

Отже, перед використанням TypeScript в проекті варто ретельно взважити його переваги та недоліки в контексті вимог\ потреб проекту, рівня знань розробників та вимог до швидкості розробки. TypeScript може бути потужним інструментом для розробки сучасних веб-додатків, але вимагає відповідної оцінки вигод та жертв.

РОЗДІЛ 3. СТВОРЕННЯ ЗАСТОСУНКУ

Написання гри



Під час роботи над курсовою я вирішив написати невеличкий застосунок-гру «крестики нулики» для демонстрації деяких інструментів цього фреймворку. За основу був взятий код з офіційного гайду React. В процесі створення застосунку я опустив моменти з дизайном та CSS верстkою, що не є важливою частиною мого дослідження.

Почнемо з основного компоненту: Game

```
export function TicTacToe(props: TicTacToeProps) {
  const [history, setHistory] = useState<HistoryItem[]>([{ squares: Array(9).fill(null) }]);
  const [coordHistory, setCoordHistory] = useState<string[]>([]);
  const [stepNumber, setStepNumber] = useState<number>(0);
  const [xIsNext, setXIsNext] = useState<boolean>(true);

  const currentState = history[stepNumber];
  const winner = calculateWinner(currentState.squares);
  const status = winner ? `Winner is ${winner}` : `Next player: ${xIsNext ? CROSS : CIRCLE}`;

  const moves = history.map((step, move) => {
    const desc = move ? `Go to move #${move}` : 'To start of game';
    return (
      <li key={move}>
        <button className='button-history' onClick={() => jumpTo(move)}>{desc}</button>
```

```
      <h4>Coords: {coordHistory[move]}</h4>
    </li>
  );
 });

 const jumpTo = (step: number) => {
  setStepNumber(step);
  setXIsNext(step % 2 === 0);
 };

 const handleClick = (i: number) => {
  const currentHistory = history.slice(0, stepNumber + 1);
  const currentState = currentHistory[currentHistory.length - 1];
  const squares = [...currentState.squares];

  const currentCoordHistory = coordHistory.slice(0, stepNumber + 1);
  if (squares[i] || calculateWinner(squares)) return;

  squares[i] = xIsNext ? CROSS : CIRCLE;
  setHistory([...currentHistory, { squares }]);
  setCoordHistory([
    ...currentCoordHistory,
    `${getRow(i)}, ${getCol(i)}: ${squares[i]}`,
  ]);
  setStepNumber(currentHistory.length);
  setXIsNext(!xIsNext);
 };

 const handleLogout = () => {
  localStorage.removeItem("userData");
  window.location.replace("/login");
 }

 return (
  <div className="game">
    <div className="game-board">
      <Board squares={currentState.squares} onClick={(i: number) => handleClick(i)} />
    </div>
    <div className="game-info">
      <div>{status}</div>
      <ol>{moves}</ol>
    </div>
    <button className='button-logout' onClick={handleLogout}>Logout</button>
  </div>
 );
};
```

Пройдемось по основним місцям функційного компоненту Game:

- Використаний набір хуків useState для зберігання інформації про стан гри, зокрема історії ходів, номеру поточного ходу, та того, хто є наступним гравцем
- handleClick - викликається при кліку на певну клітинку на дошці гри. Ця функція перевіряє, чи може гравець зробити хід у вибрану клітинку, та змінює стан гри відповідно до зробленого ходу
- jumpTo - викликається при натисканні на кнопку переходу до певного ходу у історії. Ця функція змінює стан гри, щоб відображати поточний стан гри на обраний момент часу
- handleLogout - викликається при натисканні на кнопку виходу
- Компонент Game також використовує допоміжні функції з модулю utils, кастомні типи для точнішої типізації, а також інший компоненти застосунку (Board), що відображає дошку гри

Board:

```
export function Board(props: BoardProps) {
 const renderCell = (i: number) => {
  return (
   <Cell
    value={props.squares[i]}
    onClick={() => props.onClick(i)}
   />
  );
 }

 return (
  <div>
   <div className="board-row">
    {renderCell(0)}
    {renderCell(1)}
    {renderCell(2)}
   </div>
   <div className="board-row">
    {renderCell(3)}
    {renderCell(4)}
    {renderCell(5)}
   </div>
   <div className="board-row">
    {renderCell(6)}
    {renderCell(7)}
    {renderCell(8)}
```

```
    </div>
   </div>
  );
}
```

- Цей код відповідає за відображення інтерактивної дошки для гри. Функційний компонент Board приймає props, що включає масив значень квадратів на дошці та функцію зворотнього виклику onClick.
- Функція renderSquare приймає індекс і та повертає компонент Square з відповідним значенням та функцією зворотнього виклику onClick.
- Компонент Board повертає компонент дошки з трьома рядками і трьома колонками із відповідними квадратами за допомогою функції renderSquare. Кожен квадрат відображається з використанням компонента Square, який може бути заповненим значенням "X", "O" або залишитися порожнім, залежно від значення відповідного елемента масиву squares.

На цьому написання логіки самої гри закінчується

Допоміжна функціональність

Окрім самої гри, було реалізовано авторизацію на сайті за допомогою JWT та HTTP-запитів на сервер. Цей підхід забезпечує безпеку передачі даних та можливість відстеження стану користувача (якщо користувач неавторизований – не допускати до гри).

Розглянемо основні моменти. Для початку, я додав роутер для створення можливості переходу на різні сторінки і створення «захищених» сторінок:

Роутинг

```
import { Route, Routes } from 'react-router-dom';
import { Page404 } from './pages/Page404';
import { HomePage } from './pages/Home';
import { TicTacToePage } from './pages/TicTacToe';
import { LoginPage } from './pages/Login';
import { ProtectedRoute } from './components/ProtectedRoute';

export function UseRoutes (props: any) {
 return(
  <Routes>
```

```
    <Route path='/'             element={<HomePage />}/>

    <Route path='/game'         element={<ProtectedRoute />}>
      <Route path='/game'         element={<TicTacToePage />}/>
    </Route>

    <Route path='/login'        element={<LoginPage />}/>
    <Route path='*'             element={<Page404 />}/>
   </Routes>
  )
}
```

```
import { Navigate, Outlet } from 'react-router-dom';

export function ProtectedRoute() {
  const token = localStorage.getItem('userData');

  return token ? <Outlet /> : <Navigate to="/login" />;
};
```

Як можна побачити, у компоненті UseRoutes ми визначаємо «роути» для нашого додатку за допомогою компоненту Routes з бібліотеки react-router-dom. Першим маршрутом визначено шлях '/' і пов'язаний з компонентом HomePage. Далі визначається маршрут '/game', який потребує авторизації. Для цього маршруту ми використовуємо компонент ProtectedRoute, який буде перевіряти чи існує локально збережений токен користувача, щоб дозволити доступ до компоненту GamePage. Якщо токен відсутній, користувач буде перенаправлений на сторінку входу (LoginPage). Маршрут '*' означає всі інші невизначені шляхи на сайті і рендерить компонент Page404.

Компонент ProtectedRoute перевіряє, чи існує токен у локальному сховищі. Якщо він існує, то відбувається відображення компоненту в середині захищеного маршруту. Якщо токен не знайдено, користувач буде перенаправлений на сторінку входу.

Таким чином, цей код допомагає забезпечити захист доступу до певних сторінок додатку з використанням токенів, які зберігаються в локальному

сховищі, і перенаправлення користувачів на сторінку входу в разі відсутності токенів.

Сторінка авторизації

```
export function Login(props: FormProps) {

  const refForm = useRef(null);
  const navigate = useNavigate();

  const [email, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleUsernameChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setUsername(event.target.value);
  };

  const handlePasswordChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setPassword(event.target.value);
  };

  const handleOnLogin = (event: MouseEvent<HTMLButtonElement>) => {
    axios.post(`${api_url}/auth`, { email, password })
      .then(response => {
        if (response.status === 200) {
          localStorage.setItem("userData", response.data.token);
          navigate('/');
        }
      })
      .catch(error => {
        setError('Invalid username or password');
      });
  };

  return (
    <div className='main-block-auth'>
      <form className='form-auth' ref={refForm}>
        <div>
          <label className='label-email' htmlFor="email">Username:</label>
          <input className="input-email"type="text" value={email} onChange={handleUsernameChange}/>
        </div>
        <div>
          <label className='label-email' htmlFor="password">Password:</label>
          <input className="input-password" type="password" value={password}
onChange={handlePasswordChange}/>
        </div>
        <button className='button-login' type="button" onClick={handleOnLogin}>Login</button>
        {error && <p>{error}</p>}
      </form>
```

```
    </div>
  );
}
```

Цей компонент має стан з трьома змінними: email, password та error. email та password зберігають значення, які вводить користувач, а error зберігає повідомлення про помилку, якщо авторизація була неуспішною.

Коли користувач вводить ім'я користувача та пароль, вони зберігаються у стані компонента за допомогою функцій handleUsernameChange та handlePasswordChange.

Після того, як користувач натискає на кнопку "Login", відбувається запит на сервер за допомогою бібліотеки Axios. Якщо авторизація пройшла успішно (статус відповіді 200), токен, отриманий від сервера, зберігається в локальному сховищі браузера, щоб мати доступ до авторизованого розділу сайту. У протилежному випадку, з'являється повідомлення про помилку на екрані.

Висновок

Під час виконання даної курсової роботи було проведено дослідження фреймворку React, що є одним з найпопулярніших інструментів для розробки веб-додатків в сучасному програмуванні. В ході роботи було детально вивчено предметну область, основні поняття, принципи роботи та можливості фреймворку, його архітектуру та компоненти. Він дозволяє розбити код на малі інкрементальні компоненти, що робить його більш простим для розробки, тестування та розширення.

Було проаналізовано переваги та недоліки використання React у порівнянні з іншими інструментами розробки та долю його використання серед усіх фронтенд фреймворків.

В цілому, результати дослідження підтверджують важливість фреймворку React для розробки веб-додатків, а також його значний вплив на сучасний розвиток програмування. Рекомендую продовжувати дослідження даного інструменту та використання його для створення високоякісних та продуктивних застосунків.

Джерела

- React.js офіційна документація [Електронний ресурс] Режим доступу до ресурсу: https://reactjs.org/docs/getting-started.html
- Differences between Functional Components and Class Components in React [Електронний ресурс] Режим доступу до ресурсу: https://www.geeksforgeeks.org/differences-between-functional-components-and-class-components-in-react/
- React lifecycle [Електронний ресурс] Режим доступу до ресурсу: https://www.w3schools.com/react/react_lifecycle.asp#:~:text=Lifecycle%20of%20Components,Mounting%2C%20Updating%2C%20and%20Unmounting
- Axios офіційна документація [Електронний ресурс] Режим доступу до ресурсу: https://axios-http.com/docs/res_schema
- How JSX (React) Works Under the Hood [Електронний ресурс] Режим доступу до ресурсу: https://www.telerik.com/blogs/how-jsx-react-works-under-hood
- Front-end frameworks popularity (React, Vue, Angular and Svelte) [Електронний ресурс] Режим доступу до ресурсу: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190
- History of front-end frameworks [Електронний ресурс] Режим доступу до ресурсу: https://blog.logrocket.com/history-of-frontend-frameworks/