МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «**РОЗРОБКА WEB-ДОДАТКУ ДЛЯ ПРОЕКТУВАННЯ ДВОВИМІРНИХ ФОРМ»**

Виконав: студент 2 курсу, групи 8.1219
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.А. Недоля
(ініціали та прізвище)

Керівник професор кафедри програмної інженерії, доцент, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к. ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ”_____ 2020 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Недолі Дмитру Анатолійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка web-додатку для проектування двовимірних форм

Керівник роботи Чопоров Сергій Вікторович, д.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 р. № 576-с

2. Строк подання студентом роботи 30.11.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Презентація до захисту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 27.04.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1.	Розробка плану роботи.	10.06.2020	Виконано
2.	Збір вихідних даних.	15.07.2020	Виконано
3.	Обробка методичних та теоретичних джерел.	21.08.2020	Виконано
4.	Розробка першого та другого розділу.	11.09.2020	Виконано
5.	Розробка третього розділу.	15.11.2020	Виконано
6.	Оформлення і нормоконтроль	26.11.2020	Виконано
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	16.12.2020	Виконано

Студент _____ Д.А. Недоля
(підпис) (ініціали та прізвище)

Керівник проекту _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка web-додатку для проектування двовимірних форм»: 43 сторінок, 12 ілюстрацій, 5 позицій у переліку посилань.

ВЕБ-ЗАСТОСУНОК, ДВОВИМІРНА ГРАФІКА, CANVAS, HTML, JAVASCRIPT, REACT.

Об’єкт дослідження – бібліотека React, мова програмування JavaScript, HTML Canvas.

Мета дослідження – дослідити методи та засоби розробки веб-застосунків, що містять елементи двовимірної графіки, їх архітектуру та основні компоненти, розробити застосунок для проектування двовимірних форм на базі бібліотеки React.

Метод дослідження: аналітичний, емпіричний, порівняльний.

У кваліфікаційній роботі розглянуті сучасні методи розробки веб-застосунків із використанням елементів двовимірної графіки, спроектовано та розроблено веб-застосунок для проектування двовимірних форм.

SUMMARY

Master's Qualification Thesis «Development of the Web-Application for Two-Dimensional Shapes Designing»: 43 pages, 12 illustrations, 5 references.

CANVAS, HTML, JAVASCRIPT, REACT, WEB APPLICATION, 2D GRAPHICS.

The aim of the study is to explore methods and tools for the development of web applications with elements of 2d graphics, their architecture, and main components, develop a React based application for designing two-dimensional objects.

The object of the study is React library, JavaScript programming language, HTML Canvas.

The subject of the study is the process of developing a web application for designing two-dimensional objects.

The modern methods of developing web applications with elements of 2d graphics were reviewed in the qualification paper. The React based application for designing two-dimensional objects was designed and developed.

ЗМІСТ

ВСТУП

Розробка веб застосунків – це процес розробки застосунків, що виконуються у середовищі веб-браузеру. Ці програми можуть виконуватись на будь-якому пристрої при наявності на ньому веб-браузера з підтримкою базових веб-технологій.

Швидкий розвиток інтернету та пов’язаних із ним технологій, підключення великої кількості людей до мережі, привели до появи та розвитку такої галузі розробки програмного забезпечення як веб-розробка. У зв’язку із швидким зростанням інтернет покриття, а також поширенням бездротових технологій швидкої передачі даних (3G, LTE, 5G), почав розвиватись попит на створення різноманітних веб-застосунків, а відповідно зріс попит і на спеціалістів у галузі веб-розробки. Усі ці фактори і визначили напрямок дослідження.

Мета роботи – дослідити методи та засоби розробки веб-застосунків, що містять елементи двовимірної графіки, їх архітектуру та основні компоненти, розробити застосунок для проектування двовимірних форм на базі бібліотеки React.

Для досягнення поставленої мети були визначені такі завдання дослідження:

а) дослідити архітектуру та основні компоненти бібліотеки React;

б) розглянути та порівняти React із іншими засобами розробки web-застосунків;

в) розглянути засоби розробки із використанням двовимірної графіки;

г) розробити застосунок для проектування двовимірних форм, працюючий в браузері.

Об’єкт дослідження – бібліотека React, мова програмування JavaScript, HTML Canvas.

Предмет дослідження – процес розробки веб-застосунку для

проектування двовимірних форм.

Практичне значення отриманих результатів – матеріали роботи можуть бути використані при реалізації веб-застосунків, а також при підготовці до практичних занять з дисциплін «Веб-розробка», «Людино-машинний інтерфейс», «Мова програмування JavaScript».

1 СЕРЕДОВИЩЕ БРАУЗЕРА. ПІДХОДИ ДО РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

1.1 Мова програмування JavaScript

JavaScript (JS) — динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Мова JavaScript використовується для:

а) написання сценаріїв веб-сторінок для надання їм інтерактивності;

б) створення односторінкових веб-застосунків (ReactJS, AngularJS, Vue.js);

в) програмування на стороні сервера (Node.js);

г) стаціонарних застосунків (Electron, NW.js);

д) мобільних застосунків (React Native, Cordova);

е) сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite);

ж) всередині PDF-документів тощо.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними

мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманий «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. Але спочатку багато професіональних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-любителів. Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови, а подальші модифікації мови за стандартами ES2015 та ES2017 внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, – функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання (closures) — що додає мові додаткову гнучкість.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою C має такі корінні відмінності:

а) об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів

б) функції як об'єкти першого класу

в) обробка винятків

г) автоматичне приведення типів

д) автоматичне прибирання сміття

е) анонімні функції

JavaScript містить декілька вбудованих об'єктів: Global, Object, Error,

Function, Array, String, Boolean, Number, Math, Date, RegExp.

Крім того, JavaScript містить набір вбудованих операцій, які, строго кажучи, не обов'язково є функціями або методами, а також набір вбудованих операторів, що управляють логікою виконання програм. Синтаксис JavaScript в основному відповідає синтаксису мови Java (тобто, зрештою, успадкований від C), але спрощений порівняно з ним, щоб зробити мову сценаріїв легкою для вивчення. Так, приміром, декларація змінної не містить її типу, властивості також не мають типів, а декларація функції може стояти в тексті програми після неї.

1.2 Бібліотека React

React – бібліотека JavaScript для побудови користувацьких інтерфейсів. Вона підтримується Facebook, Instagram і спільнотою окремих розробників та корпорацій. React [1] дозволяє розробникам створювати великі веб-застосунки, які використовують дані і можуть змінюватися з часом без перезавантаження сторінки. Він спрямований перш за все на забезпечення швидкості, простоти та масштабованості. React обробляє тільки користувальницькі інтерфейси у застосунках. Це відповідає View у моделі Model-View-Controller (MVC) і може використовуватися в комбінації з іншими бібліотеками JavaScript або фреймворками MVC, такими як AngularJS.

React був створений Йорданом Уолке, інженером програмного забезпечення з Facebook. На нього вплинув XHP, HTML-компонент для PHP. Вперше він був розгорнутий у новинній стрічці Facebook в 2011 році, а пізніше на Instagram.com в 2012 році. Вихідний код був відкритий на OJonf США в травні 2013 року.

Щоб застосувати React найефективніше, властивості (props), в ідеалі набір незмінних значень, передаються до функції render компонента. Компонент не повинен безпосередньо змінювати будь-які властивості, передані

йому, але повинен отримувати функції зворотного виклику, які, натомість, змінюють сховище, створюючи єдине джерело істини. Цей механізм описується як «properties flow down; actions flow up». Описаний механізм – це архітектура Flux. З моменту появи Flux було створено багато його варіантів, однак спільнота перейшла до Redux.

Ще однією помітною особливістю є використання «віртуальної моделі об'єктів документів» або «віртуального DOM». React створює кешовану копію даних у пам'яті, обчислює отримані відмінності, а потім ефективно оновлює відображений DOM браузера. Це дозволяє програмісту писати код так, ніби вся сторінка перемальовується на кожну зміну, а React відтворює тільки ті підкомпоненти, які дійсно змінюються.

Компоненти React зазвичай пишуться у JSX, розширенні синтаксису JavaScript, що дозволяє використовувати HTML та використовувати синтаксис HTML-тегів для відображення підкомпонентів. Це граматичне розширення, подібне до застарілого E4X. Синтаксис HTML перетворюється на виклики JavaScript методів React. Розробники також можуть писати на чистому JavaScript. JSX схожий на інший синтаксис розширення, створений Facebook для PHP, XHP. JSX виглядає як звичайний HTML.

1.3 Порівняння React та Angular

Основною відмінністю React і Angular вважається той факт, що перший використовує Virtual DOM. І цей факт відносять до переваг бібліотеки React.

Обидва фреймворка працюють з нативними і веб-застосунками:

а) у екосистемі React нативні застосунки розробляються за допомогою React Native;

б) у екосистемі Angular нативні застосунки розробляються за допомогою NativeScript та Ionic Framework.

При цьому у кожного з фреймворків свої слабкі і сильні сторони.

Зупинимося на цьому докладніше.

Переваги React:

а) легкий у вивченні. React набагато легше вчиться зважаючи на простоту його синтаксису. Інженери просто повинні згадати свої навички написання HTML і все на цьому. Немає потреби в глибокому вивченні TypeScript, як у випадку з Angular;

б) високий рівень гнучкості і максимальна швидкість;

в) віртуальна DOM (document object model), яка дозволяє впорядковувати документи форматів HTML, XHTML або XML в дерево, яке найкраще підходить веб-браузерам для аналізу різних елементів веб-застосунку;

г) у поєднанні з ES6/7 ReactJS може легко працювати при високих навантаженнях;

д) зв'язування даних від більших до менших. Це означає такий потік даних, при якому дочірні елементи не можуть впливати на батьківські дані;

е) JavaScript-бібліотека з відкритим вихідним кодом, яка отримує безліч щоденних оновлень і поліпшень відповідно до відгуків розробників з усього світу;

ж) неймовірно легка вага, оскільки дані, які виконуються на стороні користувача, можуть легко бути представлені на стороні сервера в той же самий час;

з) міграція між версіями, як правило, дуже проста. Також Facebook надає «codemods» для автоматизації більшої частини цього процесу;

Недоліки React:

а) брак офіційної документації. Надзвичайно швидка розробка ReactJS не залишає місця для правильної документації, яка зараз трохи хаотична, так як багато розробників вносять в неї індивідуальні зміни без будь-якого систематичного підходу;

б) React не має чіткої мети. Це означає, що розробники, іноді, мають занадто великий вибір;

в) довгий час для освоєння. ReactJS вимагає глибокого розуміння того, як інтегрувати користувальницький інтерфейс в структуру MVC.

Переваги Angular:

а) нові функції, такі як поліпшений RXJS, прискорена компіляція (менше 3 секунд) і новий лаунчер HttpClient;

б) детальна документація, яка дозволяє кожному розробнику отримати всю необхідну інформацію без звернення за допомогою до колег. Проте, навчання вимагає чималого часу;

в) двостороння прив'язка даних, яка забезпечує чудову поведінку застосунку, що мінімізує ризики можливих помилок;

г) MVVM (Model-View-ViewModel), яка дозволяє розробникам окремо працювати в одному розділі програми з використанням одного і того ж набору даних;

д) впровадження залежностей функцій пов'язаних з компонентами з модулями і модульності в цілому.

Недоліки Angular:

а) складний синтаксис, який виходить від першої версії Angular. Проте, Angular 5 використовує TypeScript, який вивчити не так вже й складно;

б) проблеми з міграцією, які можуть виникнути при переході від старої версії до нової.

1.4 Flux-архітектура

Flux-архітектура – це архітектурний підхід або набір шаблонів програмування для побудови користувацького інтерфейсу веб-застосунків, що поєднується з реактивним програмуванням і побудований на однодоступних потоках даних.

За задумом творців і незважаючи на те, що Facebook надала реалізацію Flux в додаток до ReactJS, Flux не є ще одним веб-фреймворком, а є

архітектурним рішенням.

Основною відмінною рисою Flux є одностороння спрямованість передачі даних між компонентами Flux-архітектури. Архітектура накладає обмеження на потік даних, зокрема, виключаючи можливість поновлення стану компонентів самими собою. Такий підхід робить потік даних передбачуваним і дозволяє легше простежити причини можливих помилок в програмному забезпеченні.

У мінімальному варіанті Flux-архітектура може містити три шари, які взаємодіють один по одному :

а) actions (дії);

б) stores (сховища);

в) views (подання).

Хоча зазвичай між діями і сховищами додають Dispatcher (диспетчер).

В першу чергу Flux працює з інформаційною архітектурою, яка потім відбивається в архітектурі програмного забезпечення, тому рівень уявлень слабо зачеплений з іншими рівнями системи

Дії (англ. Actions) – вираз подій (часто для дій використовуються просто імена – рядки, що містять деякий «дієслово») [4]. Диспетчери передають дії нижчих компонентів (сховищ) по одному. Нова дія не передається поки попередня повністю не оброблена компонентами. Дії через роботу джерела дії, наприклад, користувача, надходять асинхронно, але їх диспетчеризація є синхронним процесом. Крім імені (англ. Name), дії можуть мати корисне навантаження (англ. Payload), що містить пов'язані з дією дані.

Диспетчер (англ. Dispatcher) призначений для передачі дій сховищ. У спрощеному варіанті диспетчер може взагалі не виділятися, як єдиний на весь застосунок. У диспетчері сховища реєструють свої функції зворотного виклику (callback) і залежності між сховищами.

Сховище (англ. Store) є місцем, де зосереджено стан (англ. State) застосунку. Інші компоненти, згідно Flux, не мають значного (з точки зору архітектури) стану. Зміна стану сховища відбувається строго на основі даних дії і старого стану сховища.

Уявлення (англ. View) – компонент, зазвичай відповідає за видачу інформації користувачеві. У Flux-архітектурі, яка може технічно не торкатися внутрішнього облаштування уявлень взагалі, це – кінцева точка потоків даних [4]. Для інформаційної архітектури важливо тільки, що дані потрапляють в систему (тобто, назад в сховища) тільки через дії.

1.5 Компонент HTML canvas

Canvas – елемент HTML призначений для створення графіки засобами JavaScript [2]. Для прикладу, його використовують для малювання графіків, створення композиції фотографій, створення анімацій, і навіть для обробки та рендерингу відео в реальному часі. Це низькорівнева процедурна модель, яка оновлює бітову карту і не має вбудованого графа сцен, але за допомогою WebGL дозволяє відображати 3D фігури та зображення. HTML5 Canvas також допомагає у створенні 2D-ігор.

Спочатку Canvas був представлений компанією Apple для використання в їх власному компоненті Mac OS X WebKit в 2004 році, забезпечуючи роботу таких програм, як віджети Dashboard та браузер Safari. Пізніше, у 2005 році, він був застосований у версії 1.8 браузерів Gecko та Opera у 2006 році. Стандартизований Web Hypertext Application Technology Working Group (WHATWG) згідно з новими пропонованими специфікаціями для веб-технологій наступного покоління.

Canvas складається з області для малювання, визначеної в HTML-коді з атрибутами висоти та ширини. JavaScript код може отримати доступ до області за допомогою набору функцій малювання, подібних до функцій інших типових 2D-інтерфейсів, таким чином дозволячи динамічно генерувати графіку. Деякі передбачувані способи використання canvas включають побудову графіків, анімацій, ігр та композицію зображень.

Наступний код створює елемент canvas на HTML сторінці:

```
<canvas id="example" width="200" height="200">
     Якщо ви бачите цей текст, ваш браузер не має підримки HTML5.
</canvas>
```

Використовуючи JavaScript можна створювати малюнки всередині canvas [2]:

```
let example = document.getElementById('example');
let context = example.getContext('2d');
context.fillStyle = 'red';
context.fillRect(30, 30, 50, 50);
```

Цей код відображує на екрані червоний прямокутник. Canvas API також надає методи save та restore для зберігання та відновлення усіх атрибутів контексту canvas.

1.6 Бібліотека react-konva

react-konva – це бібліотека JavaScript для малювання складної графіки на canvas за допомогою React. Вона надає декларативні та реактивні прив'язки до Konva Framework [3]. Це спроба змусити React працювати з бібліотекою canvas HTML5. Мета полягає в тому, щоб мати декларативну розмітку подібну до звичайного React, і мати подібну модель потоку даних.

Приклад декларативного інтерфейсу описаного за допомогою react-konva:

```
<Stage width={window.innerWidth} height={window.innerHeight}>
      <Layer>
        <Text text="Try to drag a star" />
        {stars.map((star) => (
          <Star
            key={star.id}
            id={star.id}
```

```
            x={star.x}
            y={star.y}
            numPoints={5}
            innerRadius={20}
            outerRadius={40}
            fill="#89b717"
            opacity={0.8}
            draggable
            rotation={star.rotation}
            shadowColor="black"
            shadowBlur={10}
            shadowOpacity={0.6}
            shadowOffsetX={star.isDragging ? 10 : 5}
            shadowOffsetY={star.isDragging ? 10 : 5}
            scaleX={star.isDragging ? 1.2 : 1}
            scaleY={star.isDragging ? 1.2 : 1}
            onDragStart={handleDragStart}
            onDragEnd={handleDragEnd}
          />
        ))}
      </Layer>
    </Stage>
```

Звичайний елемент canvas може бути швидшим за react-konva, через використання бібліотекою двох шарів абстракцій: фреймворку Konva та самого React.

Мета react-konva – зменшити складність побудови застосунку та використовувати загальноприйнятий декларативний спосіб для малювання на canvas. При цьому швидкодія бібліотеки залишається на рівні достатньому для більшості варіантів використання.

1.7 Середовище браузера

Мову JavaScript спочатку було створено для веб-браузерів. Але з тих пір вона значно еволюціонувала і перетворилася на багатоплатформову мову програмування для вирішення широкого кола завдань. Сьогодні JavaScript може використовуватися в браузері, на веб-сервері або в іншому середовищі, навіть в кавоварці. Кожне середовище надає свою функціональність, яку специфікація JavaScript називає середовищем. Середовище надає свої об'єкти і додаткові функції, на додаток базовим мовним. Браузери, наприклад, дають засоби для керування веб-сторінками. Node.js робить доступними серверні можливості.

На зображенні нижче показано, які компоненти доступні JavaScript у браузерному середовищі:

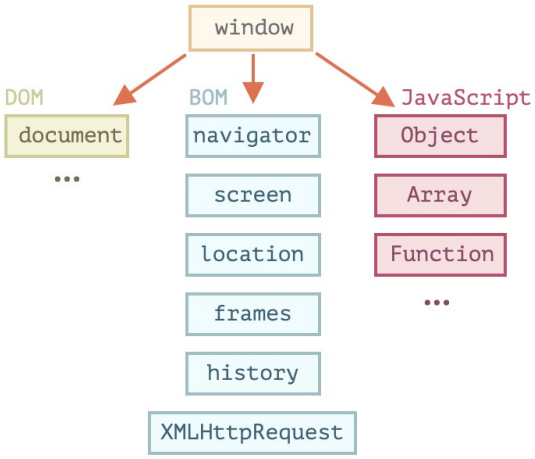

Рисунок 1.1 – Компоненти браузерного середовища

Як ми бачимо, є кореневий об'єкт window, який має в 2 ролі:

а) по-перше, це глобальний об'єкт для JavaScript-коду;

б) по-друге, він також є вікном браузера і має в своєму розпорядженні методи для керування ним.

Document Object Model (DOM) – об'єктна модель документа, яка представляє весь вміст сторінки у вигляді об'єктів, які можна змінювати. Об'єкт document – основна «точка входу». За його допомоги ми можемо щось створювати або змінювати на сторінці.

Об'єктна модель браузера (Browser Object Model, BOM) – це додаткові об'єкти, що надаються браузером (середовищем), щоб працювати з усім, крім документа. Наприклад:

а) об'єкт navigator дає інформацію про самий браузер і операційну систему. Серед безлічі його властивостей найвідомішими є: navigator.userAgent – інформація про поточний браузер, і navigator.platform – інформація про платформу (може допомогти в розумінні того, в якій ОС відкритий браузер – Windows/Linux/Mac);

б) об'єкт location дозволяє отримати поточний URL і перенаправити браузер за новою адресою.

2 РОЗРОБКА ПРОЕКТУ ЗАСТОСУНКУ ДЛЯ ПРОЕКТУВАННЯ ДВОВИМІРНИХ ФОРМ

2.1 Технічне завдання

Дане технічне завдання поширюється на розробку web-застосунку, призначеного для створення та проектування двовимірних форм.

2.1.1 Найменування та область застосування

Програмний продукт, що розробляється, отримує найменування: «ms_web». Програма призначена для для створення та проектування двовимірних форм шляхом побудови геометричних фігур та здійснення операцій над ними.

2.1.2 Підстава для розробки

Програма розробляється у якості кваліфікаційної роботи спеціальності Інженерія програмного забезпечення».

2.1.3 Призначення розробки

Дана програма призначена для вирішення наступних завдань:

а) дана програма повинна надавати робочу область на якій розміститься двовимірна схема;

б) дана програма повинна надавати можливість додавати різні фігури у робочу область;

в) дана програма повинна надавати можливість переміщувати елементи в межах робочої області;

г) дана програма повинна надавати можливість виокремити елементи над якими буде виконуватись певна операція;

д) дана програма повинна надавати можливість проводити різні види операцій над елементами та відображати результати операцій у робочій області;

е) дана програма повинна надавати можливість експортувати та імпортувати результати роботи користувача у файл в одному з поширених форматів.

2.1.4 Технічні вимоги до програмного продукту

Технічні вимоги складаються з вимог до функціональних характеристик, організації даних, надійності, параметрам технічних засобів, програмної сумісності.

2.1.4.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати можливість виконання наступних функцій:

а) робоча область – формується за допомогою елементу HTML5 canvas, займає більшу частину екрану;

б) додавання елементів – здійснюється за допомогою меню вибору типу елемента, а також форми створення конкретного елемента шляхом вводу його параметрів;

в) переміщення елементів – здійснюється за методом «drag and drop»;

г) виокремлення елементів для здійснення операції над ними – здійснюється шляхом натискання на елемент;

д) проведення операцій над елементами – здійснюється за допомогою меню вибору типу операції, що стає видимим за умови виокремлення двох елементів, над якими буде проводитись операція;

е) експорт та імпорт – здійснюється шляхом натискання відповідних пунктів навігаційного меню. Дані зберігаються у форматі JSON.

2.1.4.2 Організація вхідних и вихідних даних

Вхідні дані:

а) параметри елементів створених користувачем, такі як: координати, ширина, висота, радіус тощо.

Вихідні дані:

а) відображаються на екрані: сформовані елементи та операції над ними;

б) сформоване дерево операцій у форматі JSON.

2.1.4.3 Вимоги до надійності

Передбачаються наступні вимоги:

а) передбачити контроль інформації, що вводиться у текстові поля;

б) забезпечити коректну роботу програми в усіх користувацьких сценаріях.

2.1.4.4 Вимоги до складу і параметрам технічних засобів

Програма повинна працювати на пристроях з підтримкою сучасних веб-браузерів.

Мінімальна конфігурація:

а) тип процесора: Intel Pentium 4 або новішої версії з підтримкою SSE3;

б) підтримка мережі Інтернет;

в) об’єм оперативного пристрою: 256 мб та більше;

г) об’єм вільного місця на жорсткому диску: 50 мб.

Рекомендована конфігурація:

а) об’єм оперативного пристрою: 1 гб та більше;

б) об’єм вільного місця на жорсткому диску: 100 мб та більше.

2.1.4.5 Вимоги до програмної сумісності

Програма повинна працювати у браузерному оточенні з підтримкою стандарту ECMAScript 6 або новіше

2.1.5 Стадії і етапи розробки

а) стадія «Технічного завдання на розробку програмного продукт»;

б) стадія «Ескізний проект»;

в) стадія «Технічний проект»;

г) стадія «Реалізація» або «Робочий проект»;

д) стадія «Тестування і Отладка».

Готовий програмний виріб повинен супроводжуватися наступною документацією:

а) керівництво користувача;

б) керівництво програміста.

2.2 Діаграма прецедентів

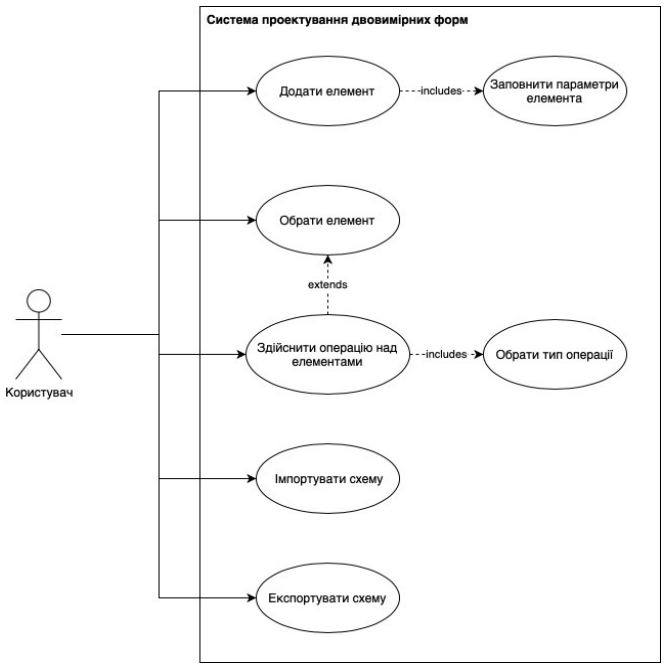

Рисунок 2.1 – Діаграма прецедентів

2.3 Структура даних

Дані про кожну схему у проекті представлені у вигляді деревоподібної структури (рис. 2.2).

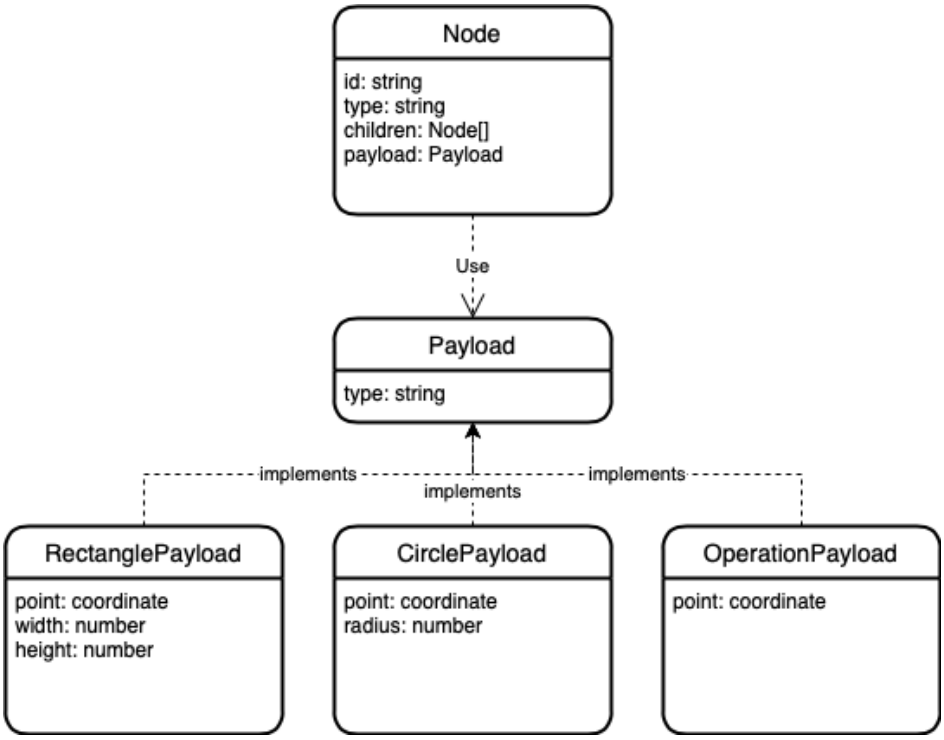

Рисунок 2.2 – Структура даних застосунку

2.4 Проектування web-застосунку

При проектуванні було вирішено використати стандартну для React застосунків Flux-подібну архітектуру. Такий підхід забезпечує розділення представлень та бізнес-логіки.

Тека src містить основну частину коду проекту (рис. 2.3). Тека components містить у собі компоненти користувацького інтерфейсу застосунку. У теці lib розташовані файли що містять логіку пов’язану із взаємодією з сторонніми API, складними обчисленнями тощо. Файл index.js є кореневим файлом усього застосунку. У файлі context.js міститься логіка використання React Context API для перевикористання стану у різних компонентах.

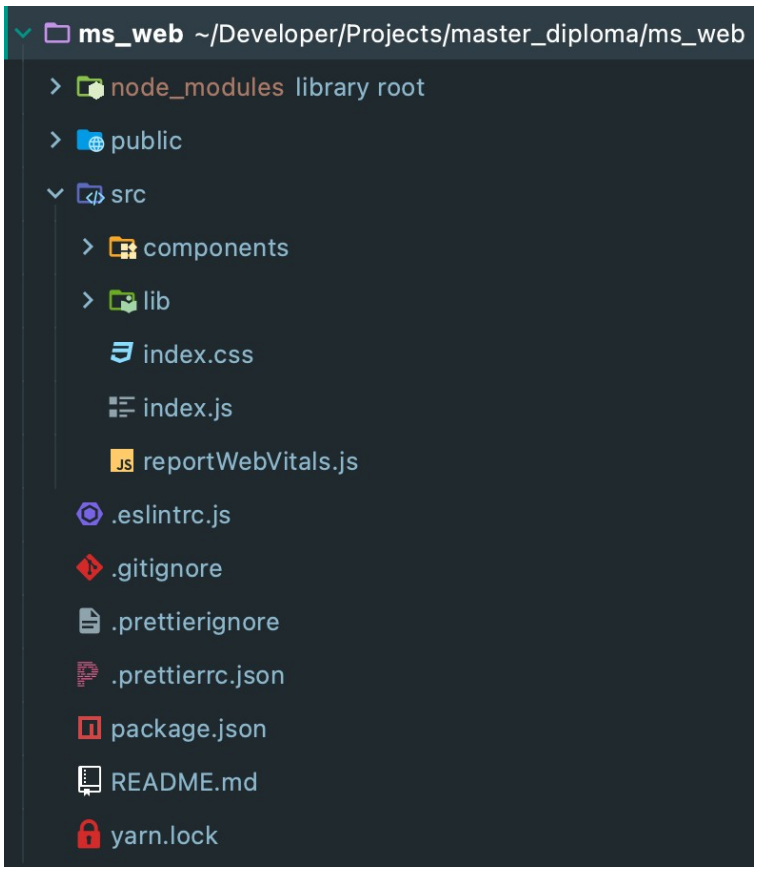

Рисунок 2.3 – Структура тек у JS проекті.

У типовому додатку React дані передаються зверху вниз (від батьківських компонентів до дітей) через props, але цей спосіб може бути занадто громіздким для певних типів даних (наприклад, локальні налаштування, тема інтерфейсу), які потрібні багатьом компонентам програми.

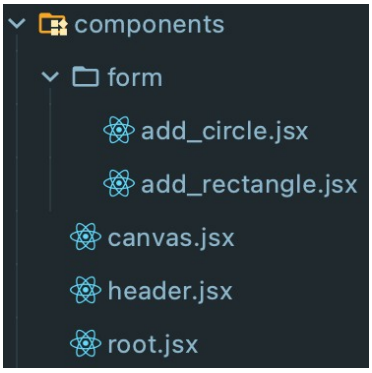

Рисунок 2.4 – Структура підтеки components

Рисунок 2.5 – Структура підтеки lib

Context API забезпечує спосіб обміну подібними даними між компонентами, не здійснюючи явного передавання props через кожен рівень дерева.

2.5 Макет інтерфейса додатку

Нижче наведені макети інтерфейсу застосунку:

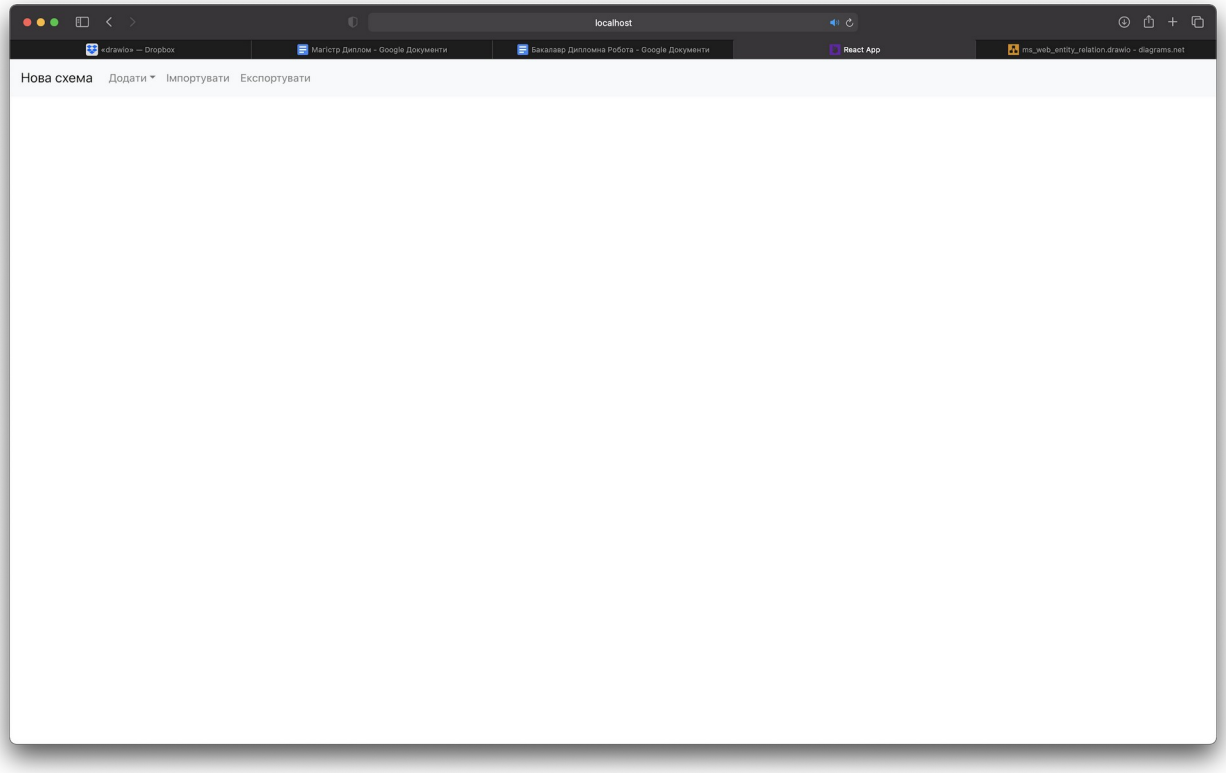

Рисунок 2.6 – Початковий екран застосунку

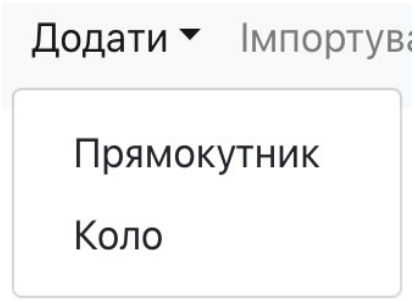

Рисунок 2.7 – Контекстне меню вибору типу елемента

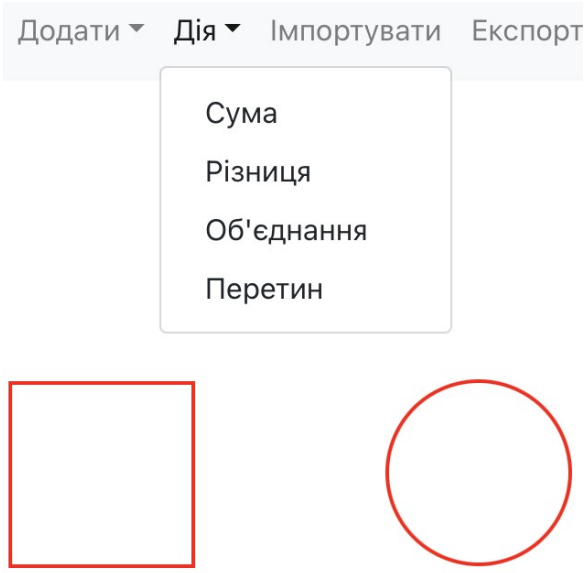

Рисунок 2.8 – Меню вибору типу операції після вибору двох елементів

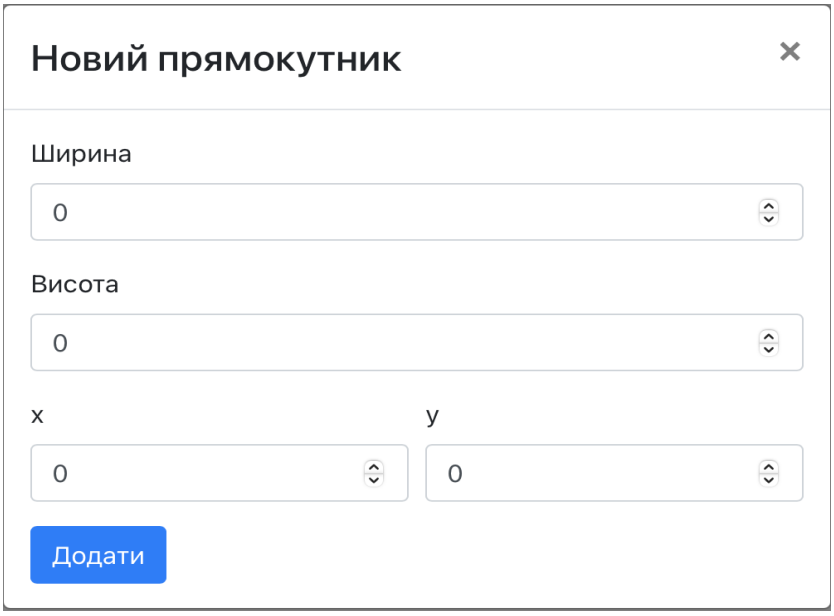

Рисунок 2.9 – Форма створення нового прямокутника

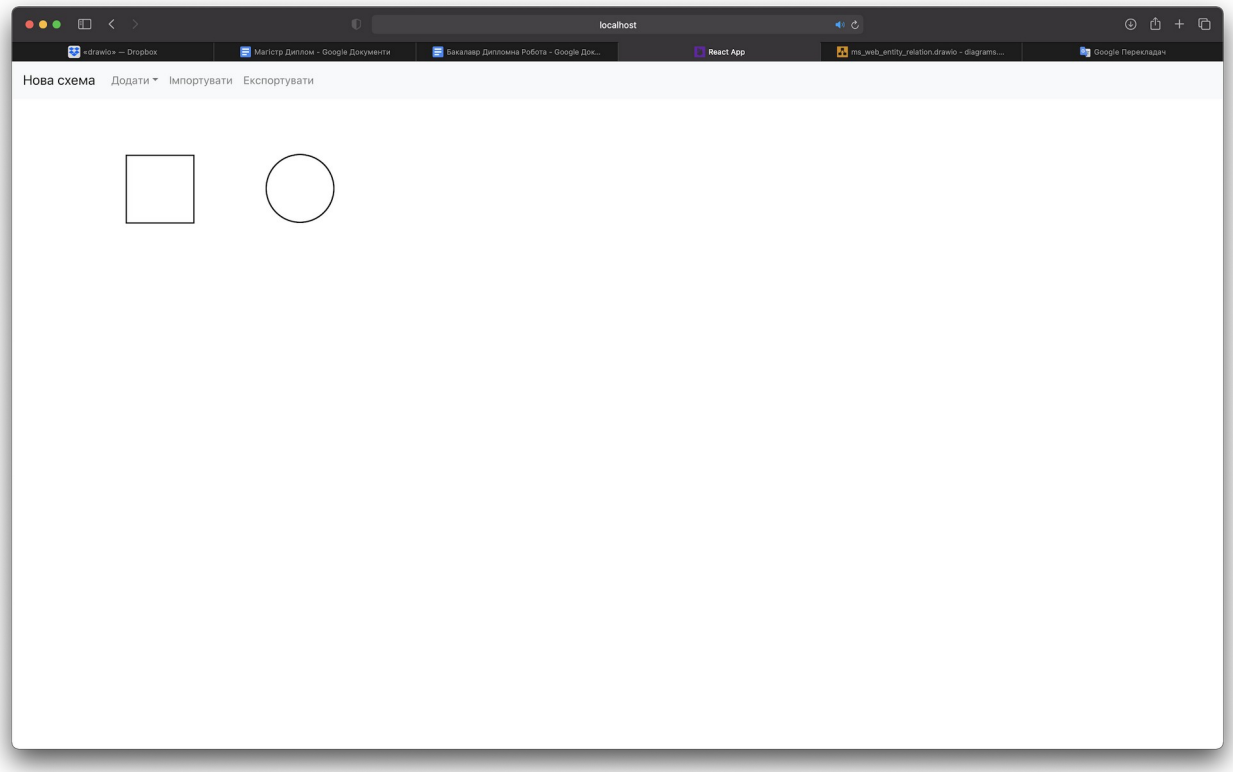

Рисунок 2.10 Екран застосунку після створення різних типів елементів.

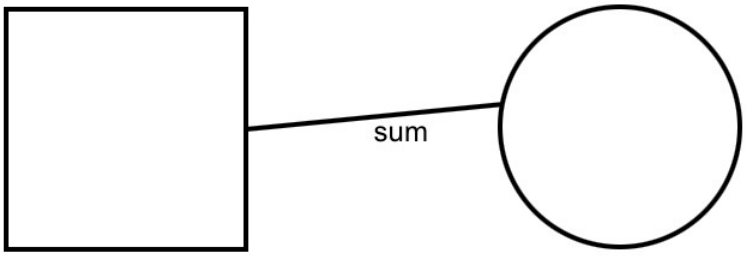

Рисунок 2.11 – Результат операції

Таким чином було отримано базовий макет web-застосунку.

3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Визначення набору інструментів

В якості IDE було обрано JetBrains WebStorm. Для ефективного виконання поставленої задачі були використані сторонні бібліотеки, такі як bootstrap для швидкої розробки інтерфейсу, react-konva для роботи з canvas, formik для форм та ін.

```
"dependencies": {
    "bootstrap": "^4.5.3",
    "formik": "^2.2.5",
    "konva": "^7.1.7",
    "prop-types": "^15.7.2",
    "react": "^17.0.1",
    "react-bootstrap": "^1.4.0",
    "react-dom": "^17.0.1",
    "react-konva": "^17.0.0-0",
    "react-scripts": "4.0.0",
    "uuid": "^8.3.1",
    "web-vitals": "^0.2.4",
    "yup": "^0.29.3"
  },
  "devDependencies": {
    "eslint": "^7.13.0",
    "eslint-config-prettier": "^6.15.0",
    "eslint-plugin-react": "^7.21.5",
    "prettier": "2.1.2"
  }
```

3.2 Визначення архітектури проекту

При розробці застосунку було використано базову архітектуру простого React застосунку. Оскільки глобальний стан застосунку обмежений деревом елементів та операцій над ними, не є доцільним ускладнювати архітектуру використанням стороннього рішення для управління станом. Тому було обрано підхід згідно з яким стан застосунку зберігається у кореневому компоненті, звідки він розповсюджується до інших компонентів за допомогою Context API.

```
const Root = () => {
  const [tree, setTree] = useState([]);
  const [selected, setSelected] = useState([]);

  const onAddOperation = (type) => {
    const connectors = getConnectorPoints(
      selected[0].payload,
      selected[1].payload
    );
    const node = {
      id: uuidv4(),
      type: "operation",
      children: [...selected],
      payload: {
        type,
        point: {
          x: connectors[0],
          y: connectors[1],
          x2: connectors[2],
          y2: connectors[3],
        },
      },
    };
    const updated = tree.filter(
```

```
      (n) => !node.children.some((s) => s.id === n.id)
    );
    updated.push(node);
    setTree(updated);
    setSelected([]);
  };

  const onExportData = () => {
    exportJSON(tree);
  };

  const onImportData = () => {
    importJSON((data) => {
      setTree(data);
    });
  };

  return (
    <StateContext.Provider value={{ tree, setTree }}>
      <div>
        <Header
          selected={selected}
          onAddOperation={onAddOperation}
          onExportData={onExportData}
          onImportData={onImportData}
        />
        <Container fluid>
          <Canvas selected={selected} setSelected={setSelected} />
        </Container>
      </div>
    </StateContext.Provider>
  );
};

export default Root;
```

Усі компоненти є функціональними та використовують hooks API, відповідно до останніх кращих практик екосистеми React. Окремі більш складні частини логіки, як наприклад геометричні обчислення, або взаємодія із браузерними API винесені в окремі модулі в теці lib. Для швидкої розробки користувацького інтерфейсу було обрано UI-фреймворк Bootstrap у зв'язці з бібліотекою компонентів React Bootstrap. Це дозволяє швидко розробити базовий макет інтерфейсу не витрачаючи час на написання CSS стилів.

3.3 Визначення структури даних застосунку

Основною функцією застосунку є створення двовимірних елементів та виконання операцій над ними. Для відображення цієї функціональності у коді було обрано структуру даних бінарне дерево. У найпростішому випадку коли є два елементи та операція над ними, елементи виступають в якості листя дерева, батьківський елемент яких – операція.

Нижче наведено приклад елемента що містить у собі параметри фігури, в даному випадку прямокутника.

```
{
        id: “4ed92f-s39vke-3f9s03”,
        type: "element",
        children: [],
        payload: {
            type: "rectangle",
            point: {
                 x: 100,
                 y: 100,
            },
            width: 150,
            height: 120,
        },
```

```
}
```

Кожен елемент дерева має подібну структуру, із деякими відмінностями в залежності від типу елемента.

3.4 Визначення логіки відображення даних

Завдяки деревоподібній структурі даних логіка відображення зводиться до рекурсивного обходу дерева та відображення кожного елементу.

```
const renderTree = (tree, rootId) => {
    const left = tree.children[0];
    const right = tree.children[1];
    let leftRender, rightRender;
    if (left) {
      leftRender = renderTree(left, rootId);
    }
    if (right) {
      rightRender = renderTree(right, rootId);
    }
    return (
      <>
        {leftRender}
        {rightRender}
        {renderNode(tree, rootId)}
      </>
    );
};
```

Наведений алгоритм дозволяє легко відображати декілька дерев на одній робочій області.

3.5 Визначення логіки відображення форм

Як вже було зазначено, для відображення форм у застосунку було обрано популярну бібліотеку Formik. Formik надає зручний декларативний інтерфейс для визначення форм майже будь-якої складності, а також бере на себе повторювані та надокучливі речі – відстеження значень, помилки, відвідані поля, організовує перевірку полей та обробляє результати вводу форми. Це означає, що ви витрачаєте менше часу на розробку стану та обробників змін і маєте більше часу на розробку своєї бізнес-логіки.

```
const AddCircleForm = ({ onSuccess }) => {
  const { tree, setTree } = useTreeState();
  return (
    <Formik
      initialValues={{ width: 0, height: 0, x: 0, y: 0 }}
      validationSchema={yup.object().shape({
        radius: yup
          .number()
          .min(1)
          .max(window.innerWidth / 2)
          .required(),
        x: yup.number().min(1).max(window.innerWidth).required(),
        y: yup.number().min(1).max(window.innerHeight).required(),
      })}
      onSubmit={onSubmit}
    >
      {({ values, touched, errors, handleChange }) => (
        <FormikForm>
          <Form.Row>
            <Form.Group as={Col}>
              <Form.Label>Радіус</Form.Label>
              <Form.Control
                type="number"
                name="radius"
```

```
              value={values.radius}
              onChange={handleChange}
              isInvalid={touched.radius && !!errors.radius}
              isValid={touched.radius && !errors.radius}
            />
          </Form.Group>
        </Form.Row>
        <Form.Row>
          <Form.Group as={Col}>
            <Form.Label>x</Form.Label>
            <Form.Control
              type="number"
              name="x"
              value={values.x}
              onChange={handleChange}
              isInvalid={touched.x && !!errors.x}
              isValid={touched.x && !errors.x}
            />
          </Form.Group>
          <Form.Group as={Col}>
            <Form.Label>y</Form.Label>
            <Form.Control
              type="number"
              name="y"
              value={values.y}
              onChange={handleChange}
              isInvalid={touched.y && !!errors.y}
              isValid={touched.y && !errors.y}
            />
          </Form.Group>
        </Form.Row>
        <Button type="submit">Додати</Button>
      </FormikForm>
    )}
  </Formik>
);
```

```
};
```

На листингу вище можна побачити легкість визначення форми створення нового елемента, в даному випадку круга, за допомогою Formik. Також у наведеному прикладі демонструється здатність Formik легко інтегруватися із сторонніми бібліотеками компонентів, такими як React Bootstrap.

3.6 Визначення логіки експорту результатів роботи застосунку

Як було зазначено під час проектування та аналізу вимог, застосунок повинен мати можливість експорту результатів роботи у файл формату JSON.

Для реалізації даної функціональності було використано можливості браузерного оточення.

```
export const exportJSON = (data) => {
  const content = JSON.stringify(data, null, "\t");
  const a = document.createElement("a");
  const file = new Blob([content], { type: "application/json" });
  a.href = URL.createObjectURL(file);
  a.download = "scheme.json";
  a.click();
};
```

Наведена функція приймає дані у вигляді стандартного об’єкту JavaScript, конвертує їх у JSON та записує у файл, що користувач може завантажити. Отриманий файл має наступний вигляд:

```
[
    {
        "id": "02b0b3c1-2b0e-4614-962e-e2328ee15eec",
        "type": "operation",
```

```
          "children": [
               {
                    "id": "7c46575c-101f-4aee-a61c-6012ea7db7b6",
                    "type": "operation",
                    "children": [
                         {
                              "id":       "082ce4d1-4341-4628-831d-
0c747f2ad07b",
                              "type": "element",
                              "children": [],
                              "payload": {
                                   "type": "rectangle",
                                   "point": {
                                        "x": 961,
                                        "y": 314
                                   },
                                   "width": 100,
                                   "height": 100
                              }
                         },
                         {
                              "id":       "f8580d66-15bd-4f14-ab39-
1959e4e10032",
                              "type": "element",
                              "children": [],
                              "payload": {
                                   "type": "rectangle",
                                   "point": {
                                        "x": 1168,
                                        "y": 311
                                   },
                                   "width": 100,
                                   "height": 100
                              }
                         }
                    ],
```

```
                    "payload": {
                         "type": "sum",
                         "point": {
                              "x": 1061,
                              "y": 347,
                              "x2": 1168,
                              "y2": 352
                         }
                    }
               },
               {
                    "id": "0a7dc364-f1d9-4a6f-b889-4f3c965cda0a",
                    "type": "element",
                    "children": [],
                    "payload": {
                         "type": "circle",
                         "point": {
                              "x": 1115,
                              "y": 554
                         },
                         "radius": 50
                    }
               }
          ],
          "payload": {
               "type": "intersection",
               "point": {
                    "x": 1114.5,
                    "y": 349.5,
                    "x2": 1102.3789051426497,
                    "y2": 505.6191363801578
               }
          }
     }
]
```

Таким чином отриманий файл можна використати для імпорту у застосунок або відкрити у будь-якому текстовому редакторі для аналізу.

ВИСНОВКИ

Отже, в процесі виконання кваліфікаційної роботи було досягнуто мети – досліджено методи та засоби розробки веб-застосунків, що містять елементи двовимірної графіки, їх архітектуру та основні компоненти, розроблено застосунок для проектування двовимірних форм на базі бібліотеки React.

При цьому було виконано наступні завдання:

а) досліджено архітектуру та основні компоненти бібліотеки React;

б) розглянуто та порівняно React із іншими засобами розробки web-застосунків;

в) розглянуто засоби розробки із використанням двовимірної графіки;

г) розроблено застосунок для проектування двовимірних форм, працюючий в браузері;

д) оформлено звітну документацію проекту.

Web-застосунок було розроблено мовою програмування JavaScript з використанням бібліотеки React.

ПЕРЕЛІК ПОСИЛАНЬ

1. React documentation. Facebook. URL : https://reactjs.org/ (дата звернення: 24.11.2020).
2. Canvas API – Web APIs | MDN. Mozilla Foundation. URL : https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (дата звернення: 24.11.2020)
3. Getting started with react and canvas via Konva | Konva. URL : https://konvajs.org/docs/react/index.html (дата звернення: 24.11.2020).
4. Freeman E., Robson E., Bates B., Sierra K. Head First Design Patterns. Sebastopol : O`Reilly Media, 2004. 694 p.
5. Simpson K. You Don't Know JS: ES6 & Beyond. Sebastopol : O`Reilly Media, 2015. 278 p.