Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка банківського сервісу для перегляду стану рахунків та переказу коштів із застосуванням бібліотеки ReactJS

Виконав: студент IV курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Буртник І. Л.
(підпис) (прізвище та ініціали)

Керівник Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Буртник Іван Любомирович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка банківського сервісу для перегляду стану рахунків та переказу коштів із застосування бібліотеки ReactJS

Керівник роботи к.т.н., ст. Викладач каф. ПІ Стоянов Ю. М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2024 року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити): Вступ. Розділ 1. Теоретичні основи розробки банківських сервісів. Розділ 2. Проєктування банківського сервісу. Розділ 3. Реалізація та тестування банківського сервісу. Розділ 4. Безпека життєдіяльності та основи охорони праці. Висновок. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів): Зображення, зображення що доповнюють текст, діаграми, знімки екрану з проробленою роботою.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	к. т. н. доцент каф. МТ Лазарук В. В.		

7. Дата видачі завдання 14 лютого 2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.02.2024	*Виконано*
2.	Підбір джерел по темі кваліфікаційної роботи	15.02.2024-25.02.2024	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	26.02.2024-01.03.2024	*Виконано*
4.	Аналіз та планування	04.03.2024-08.03.2024	*Виконано*
5.	Розробка інтерфейсу користувача	11.03.2024-31.03.2024	*Виконано*
6.	Розробка сервісної частини	01.04.2024-28.04.2024	*Виконано*
7.	Тестування та доопрацювання	03.04.2024-28.04.2024	*Виконано*
8.	Виконання звіту по проробленій роботі	29.04.2024-05.05.2024	*Виконано*
9.	Оформлення кваліфікаційної роботи	11.05.2024-14.05.2024	*Виконано*
10.	Виконання завдання до підрозділу «Безпека життєдіяльності»	07.06.2024-07.06.2024	*Виконано*
11.	Виконання завдання до підрозділу «Основи хорони праці»	08.06.2024-08.06.2024	*Виконано*
12.	Нормоконтроль		*Виконано*
13.	Перевірка на плагіат		*Виконано*
14.	Попередній захист кваліфікаційної роботи		*Виконано*
15.	Захист кваліфікаційної роботи		*Виконано*

Студент _____ Буртник І. Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розробка банківського сервісу для перегляду стану рахунків та переказу коштів із використанням бібліотеки React JS // Кваліфікаційна робота освітнього рівня «Бакалавр» // Буртник Іван Любомирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2024 // С. – 63, рис. – 29, табл. – 0, кресл. – 0, додат. – 2, бібліогр. – 12.

Ключові слова: програмне рішення, фінанси, бюджет, облік, JavaScript, ReactJS, VS Code, Bootstrap5, React Router, JSON Server, Axios.

Кваліфікаційна робота присвячена аналізу, проєктуванню, розробці та тестуванню веб-сервісу, який дозволяє користувачам переглядати стан банківських рахунків та здійснювати переказ коштів. Для реалізації було використано сучасні технології та інструменти, зокрема ReactJS.

Перший розділ присвячений дослідженню предметної області, аналізу конкурентних рішень та визначенню вимог до розробки. У цьому розділі також розглядаються засоби та інструменти, необхідні для реалізації проєкту, що дозволяє ознайомитись з темою та актуальністю даної проблематики.

Другий розділ містить опис процесів вибору архітектури, проєктування та програмної реалізації сервісу. Цей розділ дозволяє зрозуміти, як здійснювалась безпосередня розробка на основі даних, отриманих у першому розділі.

Третій розділ присвячений тестуванню та верифікації розробленого програмного рішення. У ньому розглядаються інструменти та підходи до тестування, що дозволяє сформулювати кінцеву оцінку задовільності виконаної роботи.

ANNOTATION

Development of a banking service for viewing the status of accounts and transferring funds using the React JS library // Qualification work for the educational level "Bachelor" / Burtnyk Ivan Lyubomyrovych // Ternopil National Technical University named after Ivan Puluj, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41 // Ternopil, 2024 // P. — 63, fig. — 29, table – 0, drawings — 0, supplementary — 2, bibliography —12.

Keywords: software solution, finance, budget, accounting, JavaScript, ReactJS, VS Code, Bootstrap5, React Router, JSON Server, Axios.

The qualification work is devoted to the analysis, design, development and testing of a web service that allows users to view the status of bank accounts and transfer funds. Modern technologies and tools, including ReactJS, were used for the implementation.

The first section is devoted to the study of the subject area, analysis of competitive solutions, and definition of development requirements. This section also discusses the tools and instruments needed to implement the project, which allows you to get acquainted with the topic and relevance of this issue.

The second section describes the processes of choosing the architecture, designing, and programmatic implementation of the service. This section allows you to understand how the actual development was carried out based on the data obtained in the first section.

The third section is devoted to testing and verification of the developed software solution. It discusses tools and approaches to testing, which allows you to formulate a final assessment of the satisfactoriness of the work performed.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Фінанси — це галузь, що вивчає управління грошовими ресурсами та їхнім розподілом для досягнення фінансових цілей.

HTML (Hypertext Markup Language) — це мова розмітки, яка використовується для створення структури інтернет-сторінок.

CSS (Cascading Style Sheets) — це мова стилів, яка використовується для визначення вигляду інтернет-сторінок, розділяючи їхню презентацію від структури, яку визначає HTML.

JS (JavaScript) — це мова програмування, що використовується для надання інтерактивності веб-сторінкам.

ReactJS — це бібліотека JavaScript для розробки інтерфейсів користувача, яка дозволяє створювати динамічні веб-додатки, орієнтовані на компоненти.

Bootstrap — це відкрита платформа для розробки веб-сайтів і веб-додатків, яка містить набір заздалегідь написаних HTML, CSS і JavaScript компонентів, що спрощують створення сучасних і адаптивних інтерфейсів.

UI (User Interface) — це все, що взаємодіє з користувачем на екрані, включаючи кнопки, текстові поля, меню, інші елементи і їх організацію, з метою полегшення взаємодії користувача з програмою або веб-сайтом.

SPA (Single Page Application) — це тип веб-додатка, який взаємодіє з користувачем, перевантажуючи тільки частину сторінки, замість того, щоб завантажувати повну сторінку з сервера при кожному запиті.

IDE (Integrated Development Environment)— це програмне забезпечення, яке надає розробникам інструменти для написання, редагування, тестування і налагодження коду в одному зручному середовищі.

Веб-сервіс — це програмна система, яка надає можливість іншим програмам взаємодіяти з нею через мережу Інтернет, використовуючи стандартні веб-протоколи для обміну даними.

VS Code (Visual Studio Code) — це редактор коду, розроблений Microsoft, що надає розширені можливості для програмістів та розробників.

React Router — це бібліотека для навігації в односторінкових додатках React.

ЗМІСТ

ВСТУП

У сучасному світі, де фінансові операції стають все більш цифровими, зростає потреба у зручних і безпечних банківських сервісах, які дозволяють користувачам легко управляти своїми рахунками та здійснювати перекази коштів онлайн. Для досягнення цього використовуються новітні веб-технології, які забезпечують високу продуктивність, безпеку та зручність у користуванні.

Ця кваліфікаційна робота присвячена розробці банківського сервісу для перегляду стану рахунків та переказу коштів із використанням бібліотеки ReactJS. ReactJS є однією з провідних бібліотек для створення сучасних інтерфейсів користувача, завдяки своїй компонентній архітектурі, високій продуктивності та широкому набору інструментів для розробки.

Мета роботи – створення веб-сервісу, який надасть користувачам можливість швидко та зручно переглядати баланс своїх рахунків, історію транзакцій та здійснювати перекази коштів між рахунками. Для реалізації цього проєкту використовувались такі інструменти та технології: JavaScript, ReactJS, VS Code, Bootstrap5, React Router, JSON Server та Axios.

У першому розділі кваліфікаційної роботи досліджено предметну область, проаналізовано існуючі конкурентні рішення, визначено вимоги до розробки та обрано засоби й інструменти для реалізації проєкту. Другий розділ присвячено процесам вибору архітектури, проєктування та програмної реалізації веб-сервісу. Третій розділ зосереджений на тестуванні та верифікації розробленого програмного рішення, що дозволило оцінити його функціональність та відповідність вимогам.

Ця робота демонструє комплексний підхід до розробки веб-сервісу на основі ReactJS, підкреслюючи важливість використання сучасних технологій для створення ефективних фінансових додатків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБИ БАНКІВСЬКИХ СЕРВІСІВ

1.1 Основні концепції банківських сервісів

Банківські сервіси є невід'ємною частиною сучасного фінансового ринку, спрямовані на задоволення потреб клієнтів у фінансових послугах та операціях. Основні концепції банківських сервісів базуються на принципах доступності, зручності та безпеки, що забезпечують ефективність та надійність фінансових операцій.

Сьогодні важко уявити наш світ без банку. З кожним днем все більше і більше людей відкривають власний рахунок в банку, адже це дуже сильно спрощує життя. Погодьтеся, з появою банківських карток ми майже забули про готівкову оплату. Також за допомогою банків можна взяти кредит або поставити власні кошти в депозит.

Однією з ключових концепцій, через що люди використовують банківські сервіси є зручність. Банківські сервіси повинні бути легкими у використанні для всіх категорій клієнтів, незалежно від їхнього рівня технічної грамотності. Зручний інтерфейс та інтуїтивно зрозумілі функції дозволяють користувачам швидко та безпечно здійснювати фінансові операції. В основному це є основна причина, чому дехто може не використовувати ці сервіси. Основна причина чому «Monobank» став такий популярний – це зручність інтерфейсу (див. Рис. 1.1).

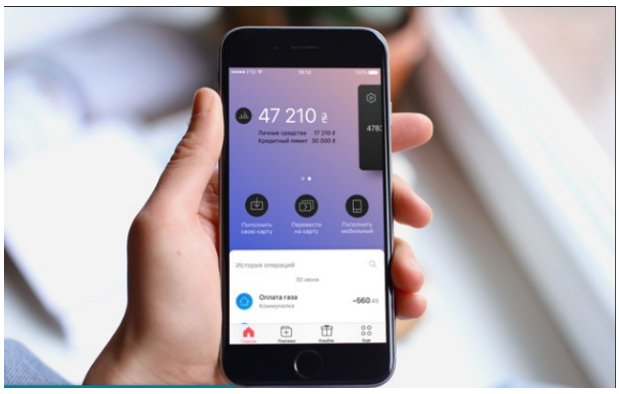

Рисунок 1.1 – Інтерфейс «Monobank»

Ще однією важливою концепцією є доступність. Банківські сервіси повинні бути доступні для клієнтів у будь-який час та з будь-якого пристрою з Інтернет-підключенням. Це означає, що клієнти можуть здійснювати фінансові операції в будь-який зручний для них час, не обмежуючись робочим часом банківських відділень.

Також є ще безліч концепцій, таких як:

- Залучення депозитів. Це є однією з основних функцій банків, що полягає у прийомі коштів від фізичних та юридичних осіб на зберігання та управління;
- Кредитування. Також є однією з основних концепцій банківських сервісів, яка полягає у надані позик фізичним та юридичним особам;
- Платіжні послуги. Функція банків, яка охоплює різноманітні операції з фінансами, що дозволяє клієнтам здійснювати різні платежі та перекази;
- Інвестиційні послуги. Концепція, що дозволяє клієнтам вкладати власні кошти з метою отримання прибутку;
- Цифрові банківські послуги. Вони представляють собою широкий спектр фінансових сервісів, що надаються через онлайн та мобільні платформи.

  Це лише декілька з основних концепцій банківських сервісів, які в сполуці являють собою надійну систему, якій довіряють багато людей.

1.2 Аналіз сучасних тенденцій у банківській сфері

Сучасний банківський сектор постійно змінюється пристосовуючися до тенденцій та вимог ринку. Проаналізувавши тенденцій цього сектору, ми получимо розуміння напрямків, які визначатимуть подальші зміни та розвиток фінансових послуг та вплинуть на стратегії розвитку банків.

Одна з провідних тенденцій є цифровізація банків. У зв’язку з прогресією мобільних технологій та Інтернету, банки повинні розвиватися і створювати власні мобільні додатки, щоб забезпечити доступ до власних сервісів клієнтам по всьому світу. Банки активно інвестують у створення власних мобільних додатків. В Україні є безліч банківських сервісів, але є три які найпопулярніші, це «Monobank», «Privat24» та «ОщадБанк» (див. Рис. 1.2, 1.3, 1.4). Але навіть в тут найбільш популярний є вище згаданий «Monobank». Не маючи жодного відділення він дозволяє клієнтам відкрити власний рахунок лише за пару хвилин у власному телефоні.

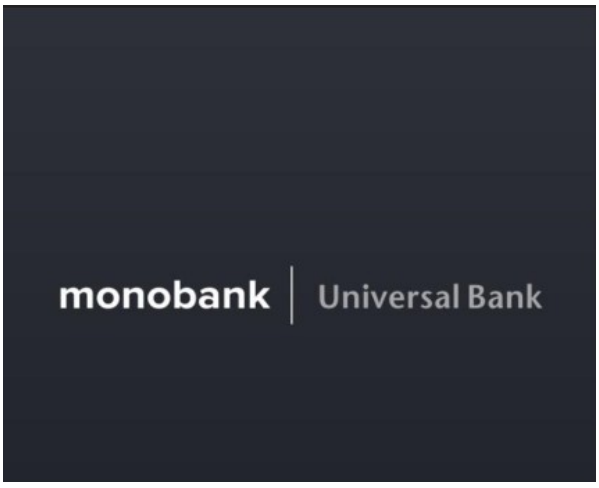

Рисунок 1.2 – «Monobank»

Рисунок 1.3 – «Privat24»

Рисунок 1.4 – «ОщадБанк»

Ще однією важливою тенденцією є підвищення уваги до кібербезпеки. У зв'язку з зростанням кількості кіберзагроз та інцидентів, пов'язаних із крадіжками фінансових даних, банки змушені приділяти особливу увагу захисту даних клієнтів. Вони впроваджують різноманітні заходи захисту, від шифрування даних до біометричної аутентифікації, щоб забезпечити безпеку фінансових операцій.

Також помітно збільшення ролі штучного інтелекту (ШІ) та аналітики даних у банківській сфері. Банки використовують ШІ для автоматизації процесів, покращення рішень у сфері кредитування, ризик-менеджменту та обслуговування клієнтів. Аналіз великих обсягів даних допомагає банкам зрозуміти потреби своїх клієнтів та надати їм персоналізовані послуги.

Не менш важливою тенденцією є розвиток безготівкових технологій. Зросла популярність мобільних платежів, криптовалют та інших електронних форм оплати, що призвело до зменшення використання готівки. Банки активно розвивають нові технології для забезпечення безпеки та зручності безготівкових операцій.

Загалом, аналіз сучасних тенденцій у банківській сфері свідчить про швидкий та постійний розвиток цієї галузі. Розуміння цих тенденцій є важливим для банків з метою адаптації до змін у фінансовому середовищі та забезпечення конкурентних переваг на ринку.

1.3 Використання цифрових технологій у банківських послугах

Одними з ключових факторів трансформації банківського сектора являється цифрові технології. Через значне прогресування цих технологій банки повинні були реформувати способи, якими клієнти взаємодіють з банками та отримують фінансові послуги. У цьому розділі буде проаналізовано, як саме банки використовують цифрові технології, як за допомогою них поліпшують свої послуги та забезпечують конкурентоспроможність.

Одна з основних технологій, яку активно застосовують в банківському секторі є інтернет-банкінг, концепцію якого в більшій мірі продемонстровано на рисунку 1.5. Ця технологія дозволяє клієнтам отримувати доступ до своїх рахунків через веб-сайт або мобільний додаток. За його допомогою можна перевірити стан рахунків, переказати кошти, оплачувати рахунки та керувати іншими фінансовими операціями з будь-якого пристрою з доступом до Інтернету.

Рисунок 1.5 – Концепція роботи інтернет-банкінгу

Також ще однією важливою цифровою технологією у банкінгу є аналітика даних. Банки збирають великі обсяги даних про своїх клієнтів та їхні фінансові операції, і за допомогою аналітики можуть отримати цінну інформацію щодо поведінки клієнтів, їхніх потреб та уподобань. Це дозволяє банкам надавати

персоналізовані послуги та рекомендації, що підвищує задоволеність клієнтів і збільшує їхню лояльність. За допомогою цієї технології банки можуть зрозуміти, чи можна буде надати вам який кредит, якщо ви побажаєте цього, і чи ви зможите потім його погасити.

Також зараз стало поширено використовувати штучний інтелект у банківській сфері. За останні роки штучний інтелект став ключовою технологією у банківському секторі. Від автоматизації клієнтського обслуговування до виявлення фінансових шахраїв, ШІ допомагає банкам покращити ефективність своїх операцій та знизити ризики.

Іншою важливою цифровою технологією є блокчейн. Блокчейн – це розподілений цифровий реєстр, що забезпечує безпечний, прозорий та незмінний спосіб запису транзакцій. Ця технологія може застосовуватися у банківській сфері для підтвердження транзакцій, обміну цифровими активами та забезпечення безпеки та недоторканості даних.

Використання цифрових технологій у банківських послугах не тільки сприяє покращенню якості обслуговування клієнтів, але й розширює можливості банків у відповідь на виклики, зміни та конкуренцію на ринку фінансових послуг.

1.4 Огляд бібліотеки ReactJS та її застосування у веб-розробці

Бібліотека ReactJS [1], створена компанією Facebook у 2013 році, є однією з найпопулярніших технологій для розробки інтерфейсів користувача веб-додатків. ReactJS [1] дозволяє розробникам створювати ефективні та динамічні веб-додатки, які забезпечують високу продуктивність та зручність користування. Цей розділ присвячений огляду основних особливостей бібліотеки ReactJS [1] та її застосування у веб-розробці.

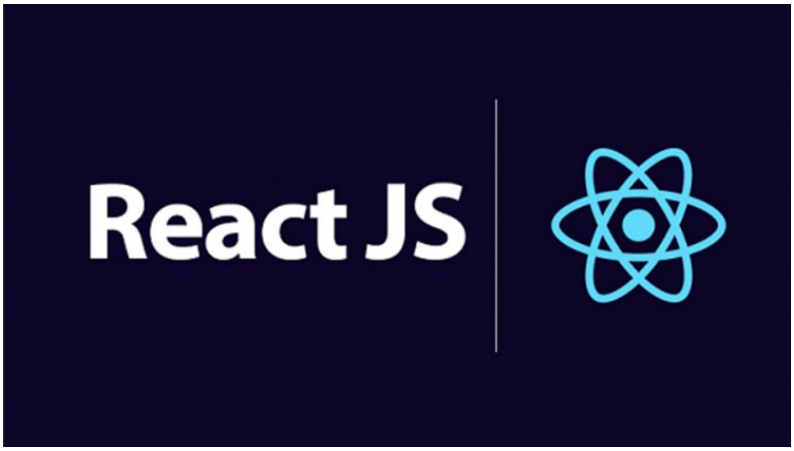

Рисунок 1.6 – Логотип бібліотеки

ReactJS [1] є бібліотекою для побудови користувацьких інтерфейсів, що використовує компонентний підхід. Компоненти в ReactJS [1] є незалежними та повторно використовуваними блоками коду, які інкапсулюють певну логіку та відображення. Це дозволяє розробникам легко створювати складні інтерфейси, складаючи їх із простих компонентів.

Однією з ключових особливостей ReactJS [1] є віртуальний DOM (Document Object Model). Віртуальний DOM є абстракцією реального DOM, яка дозволяє React ефективно оновлювати інтерфейс користувача. Коли стан додатку змінюється, React оновлює віртуальний DOM, порівнює його з попереднім станом і вносить мінімальні зміни до реального DOM. Це забезпечує високу продуктивність і швидкість роботи додатків.

Ще однією важливою особливістю є однонаправлений потік даних. У ReactJS [1] дані передаються від батьківських компонентів до дочірніх через властивості (props). Це забезпечує передбачуваність і простоту управління станом додатку. Крім того, для управління глобальним станом часто використовують такі бібліотеки як Redux або Context API.

ReactJS [1] підтримує JSX (JavaScript XML), що дозволяє писати компоненти з використанням синтаксису, схожого на HTML [8]. JSX спрощує розробку інтерфейсів, роблячи код більш читабельним та зрозумілим.

Ще однією перевагою ReactJS [1] є велика спільнота розробників та наявність багатьох інструментів та бібліотек. Наприклад, React Router [5] використовується для маршрутизації у додатках, а Create React App – для швидкого створення нових проектів з налаштованим середовищем розробки. Це дозволяє розробникам швидко починати нові проекти та ефективно працювати.

ReactJS [1] також відомий своєю гнучкістю та масштабованістю. Він може використовуватися для створення як невеликих компонентів, так і великих складних додатків. Завдяки компонентному підходу, розробники можуть легко підтримувати та розширювати свої додатки, додаючи нові функції без значних змін у коді.

Однією з причин популярності ReactJS [1] є його екосистема. Завдяки активній спільноті та постійному розвитку, ReactJS [1] інтегрується з багатьма іншими інструментами та фреймворками, що забезпечує гнучкість у виборі технологій для реалізації проектів.

У підсумку, ReactJS [1] є потужною бібліотекою для веб-розробки, яка дозволяє створювати ефективні, динамічні та масштабовані веб-додатки. Використання компонентного підходу, віртуального DOM та однонаправленого потоку даних робить ReactJS [1] одним із найефективніших інструментів для створення сучасних веб-інтерфейсів.

1.5 Технології та засоби розробки

У ході виконання поставленої задачі було підготовлено основні технології, які будуть використані для розробки надійного та сучасного веб-сервісу.

Мовою програмування було вибрано JavaScript [2]. JavaScript – це високорівнева, динамічна, об'єктно-орієнтована мова програмування, яка широко використовується для створення інтерактивних і динамічних веб-сторінок [2].

Рисунок 1.7 – Логотип мови програмування

Однак, щоб полегшити написання веб-сервісу було також вибрано бібліотеку ReactJS. ReactJS – це популярна бібліотека JavaScript [2] для створення користувацьких інтерфейсів [1].

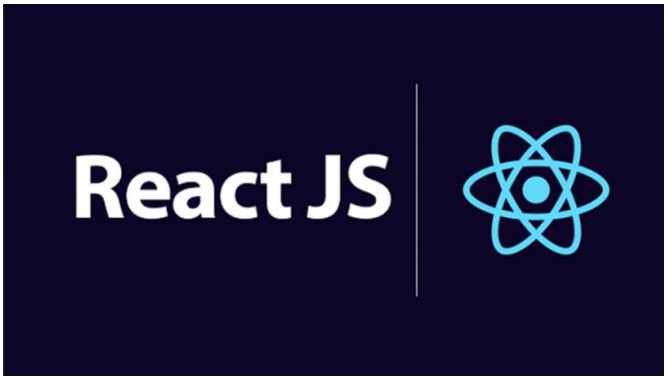

Рисунок 1.8 – Логотип бібліотеки

Також потрібно обрати IDE у якому буде писатися код для нашого сервісу. Існує багато IDE, які можна використати для досягнення нашої цілі, такі як NodePade++, SublimeText, WebStorm, Visual Studio Code і ще багато інших, але ми будемо використовувати Visual Studio Code (далі VS Code). VS Code – це безкоштовний редактор коду, який має відкритий вихідний код і який розроблено компанією Microsoft. Він має велику популярність серед розробників через свою функціональність, гнучкість та підтримку різних мов програмування [3].

Рисунок 1.9 – Логотип VS Code

Отже, роблячи висновки з цього розділу, ми зрозуміли що дана тема є доволі актуальною. Також ми підготували і розібрали технології, які ми будемо використовувати для реалізації даного проекту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ БАНКІВСЬКОГО СЕРВІСУ

2.1 Вимоги до системи

Вимоги до системи є невід'ємною складовою етапу проектування будь-якого програмного забезпечення, зокрема банківських сервісів. Визначення чітких вимог до системи забезпечує її відповідність очікуванням користувачів та бізнес-цілям, а також дозволяє уникнути проблем у процесі розробки та експлуатації. Вимоги до системи можна поділити на функціональні та нефункціональні.

Функціональні вимоги визначають конкретні можливості та функції, які система повинна надавати користувачам. Вони є основою для розробки та тестування програмного забезпечення, оскільки описують очікувану поведінку системи в різних сценаріях використання. Нефункціональні вимоги визначають критерії, яким повинна відповідати система для забезпечення її ефективності, надійності та зручності у використанні. Вони охоплюють аспекти продуктивності, безпеки, масштабованості, сумісності та зручності обслуговування, що є критичними для успішного функціонування банківського сервісу.

До функціональних вимог можна віднести наступні пункти:

- авторизація та аутентифікація користувачів;
- перегляд стану рахунків;
- переказ коштів;
- повідомлення та сповіщення;
- персоналізація інтерфейсу;
- підтримка кількох мов;
- доступність для людей з обмеженими можливостями.

Функціональні вимоги до банківського сервісу є ключовими для забезпечення його успішної роботи та відповідності потребам користувачів. Вони визначають конкретні можливості та функції системи, які повинні бути реалізовані для забезпечення її ефективної та безпечної експлуатації.

До нефункціональних вимог можна віднести наступні пункти:

- продуктивність. Продуктивність системи визначається її здатністю швидко реагувати на запити користувачів та ефективно обробляти великі обсяги даних. У контексті банківського сервісу продуктивність є особливо важливою, оскільки користувачі очікують миттєвого доступу до інформації про свої рахунки та швидкого виконання фінансових операцій;
- безпека. Безпека є однією з найважливіших нефункціональних вимог для банківських систем, оскільки вони обробляють конфіденційну фінансову інформацію користувачів. Високий рівень безпеки необхідний для захисту від кіберзагроз та забезпечення довіри користувачів;
- масштабованість. Масштабованість системи визначає її здатність адаптуватися до зростаючих навантажень і забезпечувати стабільну роботу при збільшенні кількості користувачів або обсягу даних. Це важливо для банківських систем, які повинні бути готові до розширення бізнесу та зростання клієнтської бази;
- сумісність. Сумісність визначає здатність системи коректно працювати на різних платформах, пристроях та з різними браузерами, що є важливим для забезпечення доступності банківських послуг для широкої аудиторії користувачів;
- зручність обслуговування. Зручність обслуговування визначає, наскільки легко система може бути підтримувана, оновлювана та розширювана. Це включає вимоги до документування, моніторингу та тестування.

Дотримання нефункціональних вимог є критично важливим для успішного функціонування банківського сервісу. Вони забезпечують надійність, продуктивність, безпеку та зручність використання системи, що є ключовими аспектами для задоволення потреб користувачів та бізнесу.

Ясна річ, що виконати ці всі вимоги за цей час не вийде, через те що над проектом буде працювати одна людина, тому нефункціональні вимоги майже не будуть зачіпатися і деякі з функціональних вимог також не будуть реалізовані,

адже ми робимо тестову версію нашої системи і в майбутньому вони будуть дороблятися.

2.2 Архітектура системи

Отже, щоб створити правильно працюючий веб-сервіс нам потрібно розібратися, як він повинен працювати із користувачем.

Найперше ми маємо користувача, який хоче увійти у власний акаунт. Отже, перше при вході на наш веб-сервіс, він попадає на сторінку авторизації. Авторизація – це процес визначення прав і привілеїв користувача після його автентифікації. Вона дозволяє системі контролювати доступ до ресурсів, даних і функцій на основі рівня дозволів, які має користувач. На цій сторінці повинні знаходитися поля вводу ID користувача і його PINкод, а також кнопка, яка при натисканні на неї повинна перевіряти правильність введення вище зазначених полів. Якщо все правильно введено і дані які користувач ввів зібгаються із даними, які знаходяться у базі сервіса, то сайт повинен переадресувати користувача на наступну сторінку, а якщо дані не правильні, то сервіс повинен вивести повідомлення про неправильність введеня якихось даних і потрібно перевірити дані щоб переадрисація пройшла вдало.

Якщо ж дані введено правильно, то користувача перекидає на сторінку, яка буде називатися «My Account». На цій сторінці повинно бути розміщено наступне:

- кнопка виходу із системи: при натискані на цю кнопку, сервіс повинен перекинути користувача на попередню сторінку сайту, тобто на сторінку автентифікації;
- наступне, що повинно бути розташовано на цій сторінці, інформація про користувача, а саме:

- ID користувача;
- номер акаунта;
- ім’я користувача;
- електронна адреса користувача;
- пароль до акаунта;
- контактний номер телефона користувача;
- тип акаунта.

- також, потрібно додати можливість редагувати дані акаунта, наприклад ім’я, електронну адресу, номер телефону і пароль;
- ще потрібно буде додати спосіб, який буде переадресовувати користувача на сторінку, на якій буде розміщено всі транзакції, які користувач здійснював, наприклад додати кнопку, яка буде виконувати ці поставлені дії;
- також, потрібно буде додати кнопку яка буде видаляти акаунт. При натисканні на неї буде вискакувати вікно сповіщення, яке буде запитувати, чи користувач дійсно хоче видалити акаунт, якщо він погодиться то акаунт буде видалено і користувача буде перекинуто на сторінку авторизацій, і дані про цей обліковий запис буде видалено із бази сервісу, якщо ж користувач все-таки відмовиться то нічого не відбудеться і користувач залишиться на цій сторінці, і зможе продовжувати робити те зо йому потрібно.

Отже реалізувавши вище наведені вимоги можна буде створити дану сторінку і переходити до реалізації наступної.

Наступна сторінка яку потрібно буде реалізувати – це сторінка яка буде показувати всі транзакції, які користувач робив. На цій сторінці буде знаходитися наступне:

- навігаційна панель із попередньої сторінки, на якій буде розташовано назву банка, а також кнопка виходу із облікового запису;
- список всіх транзакцій які робив користувач, а саме наступна інформація:

- ID транзакції: номер який присвоюється транзакції, щоб легше можна було її відшукати між рештою. Ця функція в більшості випадків буде використовуватися самим банком ніж користувачем;
- дата: коли була зроблена дана транзакція, також буде використовуватися для пошуку;
- тип: що саме зробив користувач, получив гроші чи витратив їх;
- сума: показує скільки грошових одиниць було витрачено чи получено в ході виконання транзакції.

- випадаючий список сортування. Сортування буде відбуватися за типом транзакції;
- спосіб переадресації на наступну сторінку для створення нових транзакцій;
- кнопка яка буде повертати на попередню сторінку.

Архітектура взаємодії користувача із сервером проілюстрована на рисунку 2.1.

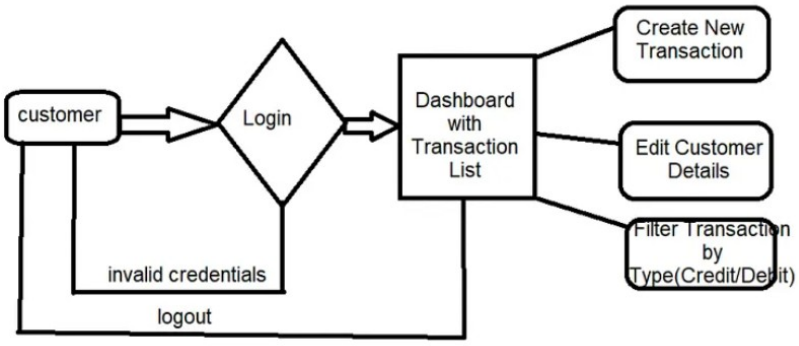

Рисунок 2.1 – Архітектура взаємодії користувача із банківським сервісом

За допомогою бібліотеки ReactJS [1] можна легко реалізувати цю архітектуру використовуючи модель SPA. SPA (Single Page Application) – це тип веб-додатку, який завантажує єдину HTML-сторінку [8] і динамічно оновлює її вміст у міру взаємодії користувача з додатком, без необхідності перезавантаження всієї сторінки.

Також, щоб реалізувати цю архітектуру, потрібно зробити компоненти, які будуть реалізовувати певну логіку сайту і поможуть в реалізації моделі SPA. Компоненти в ReactJS – це основні будівельні блоки для побудови інтерфейсів користувача. Вони є незалежними, самодостатніми частинами коду, які можуть бути повторно використані та об'єднані для створення складних користувацьких інтерфейсів.

Компонент LoginPage – призначений для авторизації користувача у наш сервіс. На цьому компоненті буде розміщено пору полів введення даних користувача, а також кнопки для перевірки правильності даних і можливості входу в систему.

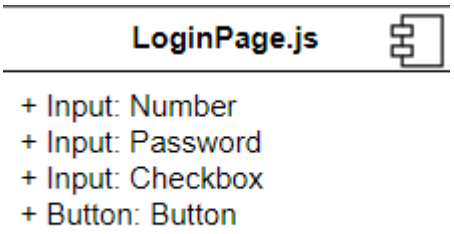

Рисунок 2.2 – Компонент LoginPage

Компонент UserPage – це основний компонент усього сервісу на якому буде розміщена вся інформація про користувача, а також спосіб переходу на наступні компоненти сайту. Дані які розміщені на цій сторінці можна буде редагувати.

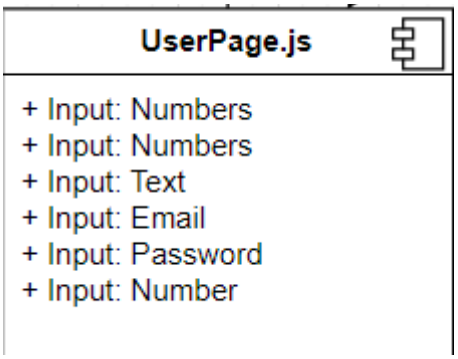

Рисунок 2.3 – Компонент UserPage

Компонент ViewTransaction – призначений для перегляду всіх транзакцій, які колись робив користувач, а також сортування їх за певними даними.

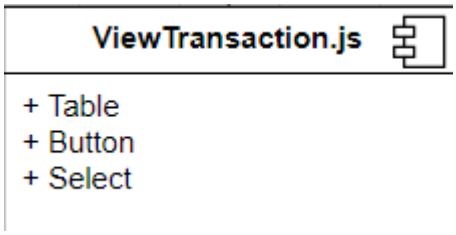

Рисунок 2.4 – Компонент ViewTransaction

Компонент Navbar – це компонент який буде знаходитися на кожному компоненту окрім LoginPage. Він буде використовуватися як навігаційна панель сайту. На нім буде розміщена кнопка і назва банку.

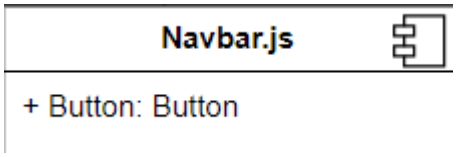

Рисунок 2.5 – Компонент Navbar

На рисунку 2.5 наведено діаграму компонентів, які використовуються для написання сервісу. Також на цій діаграмі вказано, які атрибути і типи атрибутів будуть використані при розробці даного компонента. А також ця діаграма вказує взаємодію компонентів між собою.

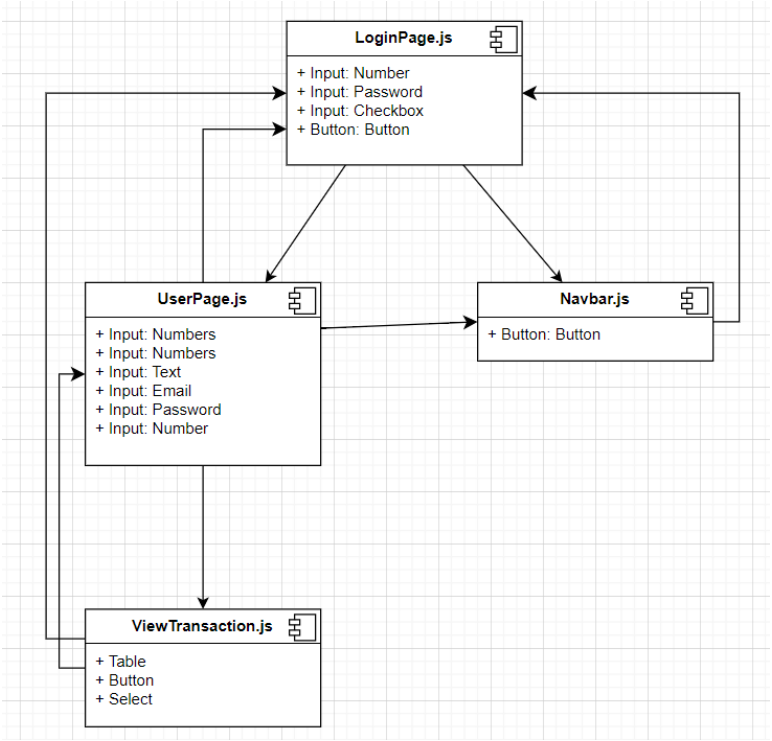

Рисунок 2.6 – Діаграма компонентів

Отже, розібравшись із архітектурою системи і її компонентами можна приступити до розробки інтерфейсу користувача, але перед цим потрібно буде ще обрати які технології підійдуть для виконання цього завдання, щоб все працювало правильно і швидко.

2.3 Розробка інтерфейсу користувача з використанням ReactJS

Розробка інтерфейсу користувача є невід'ємним етапом створення банківського сервісу. Використовуючи бібліотеку ReactJS [1] можна створити ефективні, інтерактивні та динамічні користувацькі інтерфейси, які забезпечують високий рівень зручності для користувачів.

ReactJS [1] являє собою бібліотеку для створення користувацьких інтерфейсів, що використовує компонентний підхід. В ReactJS [1] компоненти є незалежними блоками коду, які можна використовувати повторно, при цьому

вони інкапсулюють як логіку, так і відображення. Це дозволяє розробникам, складаючи із простих компонентів, створювати складні інтерфейси.

Отже, що ж таке компонентний підхід. Компонентний підхід в ReactJS – це методологія розробки інтерфейсів, яка передбачає створення інтерфейсу як набору невеликих, незалежних та повторно використовуваних компонентів. Кожен компонент відповідає за певну частину інтерфейсу і має власну логіку та стан.

Також компоненти можуть мати стан (state) і властивості (props). Стан використовується для збереження даних, які можуть змінюватися протягом всього життєвого шляху компонента. У свою чергу, властивості передаються від батьківських компонентів до дочірніх і є незмінними. Це все забезпечує однонаправлений потік даних, що робить процес управління додатком більш передбачуваним і зрозумілим.

Щоб керувати станом у складних додатках, часто використовують бібліотеки Redux [9] або Context Api [10] (див. Рис. 2.2 і 2.3). Ці бібліотеки дозволяють зберігати глобальний стан додатку і передавати його в будь-який компонент, що спрощує обробку даних і взаємодію між компонентами.

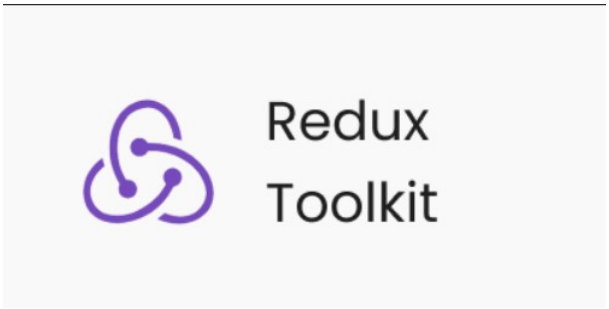

Рисунок 2.7 – Логотип бібліотеки Redux

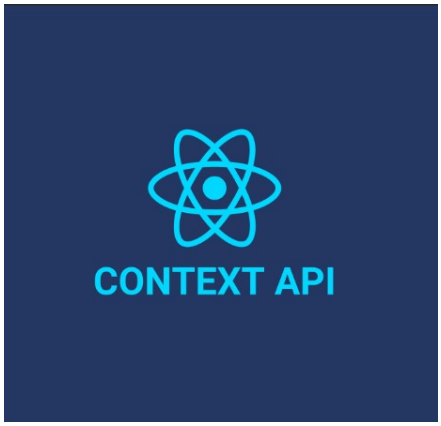

Рисунок 2.8 – Логотип бібліотеки Context Api

Ще одним плюсом використовувати бібліотеку ReactJS [1] – це змога використовувати JSX [1]. JSX [1] (JavaScript XML) є розширенням синтаксису JavaScript [2], яке дозволяє писати HTML [8]-подібний код у JavaScript [2]. Він робить код компонентів більш читабельним і зрозумілим.

Для створення багатосторінкових додатків у React [1] часто використовують бібліотеку React Router [5] (Рис. 2.5). React Router – це бібліотека для React, яка забезпечує маршрутизацію в односторінкових додатках [5]. За допомогою неї, можна визначити маршрути та керувати навігацією між різними сторінками або компонентами у додатках.

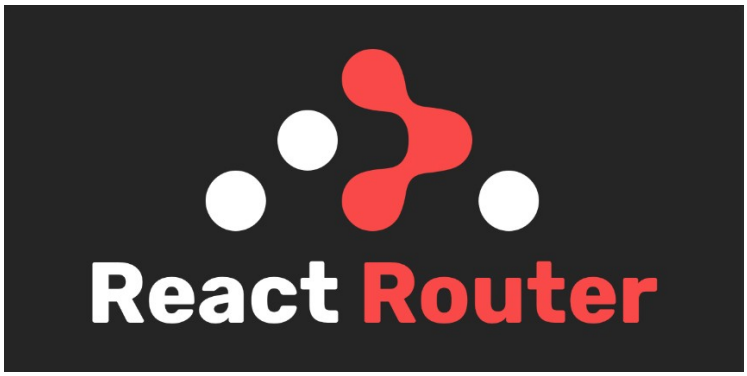

Рисунок 2.9 – Логотип бібліотеки React Router

Також під час реалізації буде застосовуватися така технологія як Axios [7]. Axios — це бібліотека для створення HTTP-запитів на JavaScript. Вона дозволяє

легко виконувати запити до сервера, обробляти відповіді та керувати асинхронними операціями [7]. Axios підтримує обіцянки, що робить її практичною для використання в сучасних JavaScript-додатках, в тому числі створених за допомогою React [1].

Рисунок 2.10 – Логотип бібліотеки Axios

Підводячи підсумки, можна сміло заявити, що розробка інтерфейсу користувача з використанням бібліотеки ReactJS [1] дозволяє створити ефективні та зручні банківські сервіси. Компонентний підхід, використання JSX [1], управління станом, роутинг та стилізація забезпечують гнучкість та масштабованість додатків. Завдяки цим інструментам та методам, ReactJS [1] є потужною бібліотекою для розробки сучасних веб-додатків, які відповідають високим вимогам користувачів та бізнесу.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ БАНКІВСЬКОГО СЕРВІСУ

3.1 Реалізація основних функцій сервісу на ReactJS

Що ж, розробивши архітектуру і визначивши вимоги для створення банківського сервісу, можна приступати до реалізації цього сервісу.

Перш-за-все створимо самий проект. Наступне що потрібно зробити це придумати назву для нашого веб-сервісу. Нехай він буде мати назву «TNTU Bank», це всього тестовий варіант і в майбутньому він зміниться. Після цього можна приступати до реалізації сервісу.

Першу сторінку, яку потрібно буде зробити – це сторінка авторизації. Що ж, для цього створюємо компонент і даємо йому назву «LoginPage». Наступним кроком потрібно буде додати бібліотеки і імпортувати із них, те що нам знадобиться для розробки даного компонента. Далі створюємо логіку самого компонента, тобто створюємо відповідні функції. І вкінці робимо рендеринг цього компонента, щоб він відображався на сервері. Реалізауію можна побачити на рисунку 3.1, а те як виглядає даний компонент можна побачити на рисунку 3.2.

Рисунок 3.1 – Реалізація LoginPage

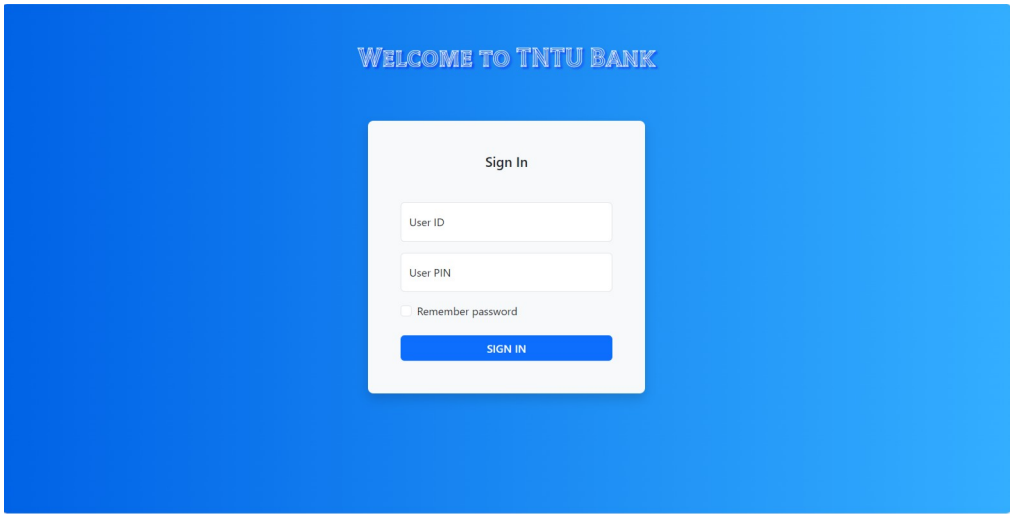

Рисунок 3.2 – Остаточний вигляд компонента LoginPage

Наступний компонент, який буде реалізовано – це основний компонент нашого веб сервісу UserPage. На ньому буде знаходитися уся інформація про користувача, а саме його ім’я, номер акаунта, ID, електронна адреса, номер телефона і пароль. За задумкою, всю цю інформацію можна буде змінити за бажанням користувача. Тому потрібно буде розмістити поля введення, на яких буде все це розміщено, а також потрібно добавити кнопку, щоб вона додавала змогу змінювати їх. Також потрібно додати спосіб переходу на наступні компоненти, а також змогу видаляти акаунт. Реалізація даного компонента зображена на рисунку 3.3 і остаточний вигляд на рисунку 3.4.

```jsx
import axios from "axios";
import React, { useEffect, useRef, useState } from "react";
import { Link, useNavigate, useParams } from "react-router-dom";

export default function UserPage() {
  const url = "http://localhost:3000/users";

  let navigate = useNavigate();

  const { index } = useParams();

  const inputRef = useRef(null);

  const [user, setUser] = useState({
    userId: "",
    accountNumber: "",
    accountHolder: "",
    accountType: "",
    contact: "",
    email: "",
    password: "",
  });

  const [readOnly, setReadOnly] = useState(true);

  const handleChange = (e) => {
    setUser({
      ...user,
      [e.target.name]: e.target.value,
    });
  };
```

Рисунок 3.3 – Реалізація компонента UserPage

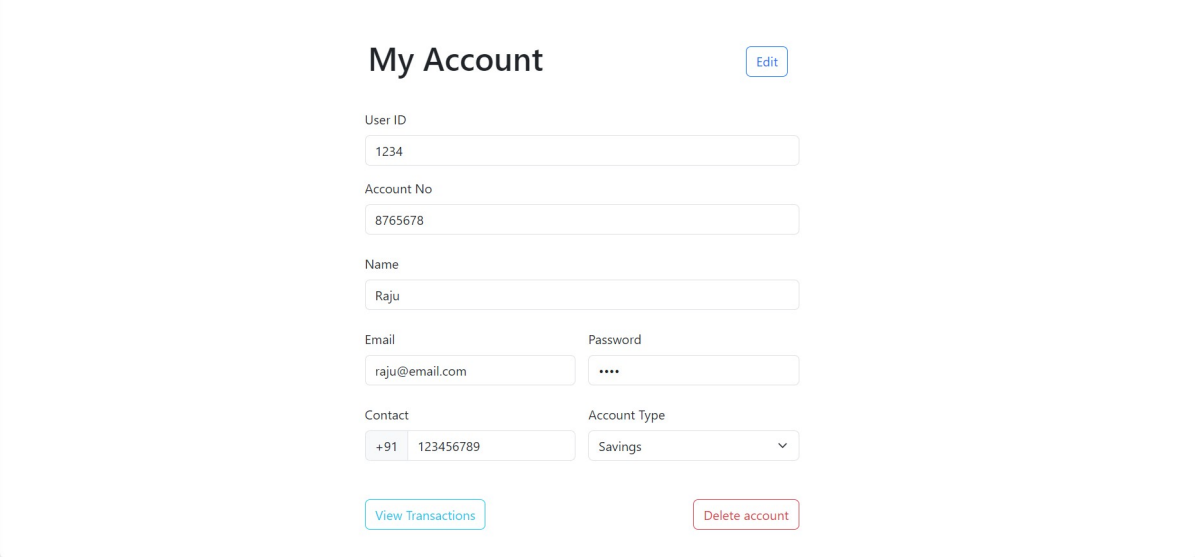

Рисунок 3.4 – Остаточний вигляд компонента UserPage

Після реалізації цього компоненту можна приступати до наступного, а саме ViewTransaction. Це компонент який буде показувати усі транзакції, тому для цього потрібно розмістити на ній таблицю, де будуть розміщені і додаватися нові транзакції. Також потрібно додати можливість фільтрування їх і добавити

можливість робити нові транзакції. Реалізацію можна побачити на рисунку 3.5, а остаточний вигляд на рисунку 3.6.

```
import axios from "axios";
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";

export default function ViewTransactions() {
  const url = "http://localhost:3000/users";

  let navigate = useNavigate();

  let { index } = useParams();

  const [filterType, setFilterType] = useState("none");

  const [transaction, setTransaction] = useState([]);

  const handleChange = (e) => {
    setFilterType(e.target.value);
  };

  const transactions = async () => {
    const datas = await axios.get(url);
    const userTransactions = datas.data[index].customerTransactions;
    setTransaction(userTransactions);
  };

  useEffect(() => {
    transactions();
  }, []);
```

Рисунок 3.5 – Реалізація ViewTransaction

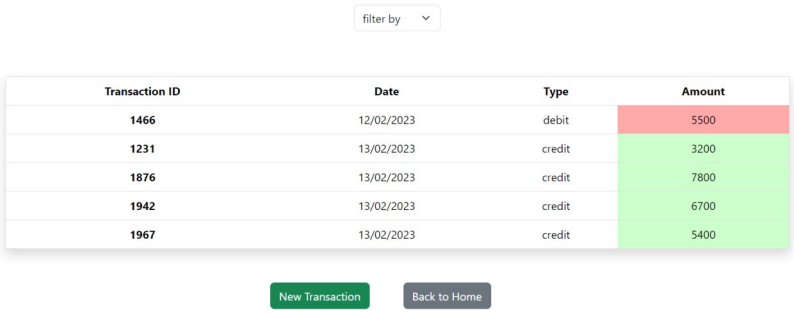

Рисунок 3.6 – Остаточний вигляд компонента ViewTransaction

І останній компонент який буде розроблено – це Navbar. Цей компонент не є якимось особливим, але він є невід'ємним для нормальної функціональності сайту. Він буде відповідати за навігацію на нашому сайті. Цей компонент буде

розміщений на всіх сторінках сайту окрім сторінки авторизації. На ньому буде розміщено назву нашого сервісу, а також кнопка виходу з акаунту. Те як реалізована цей компонент зображено на рисунку 3.7, а остаточний вишляд на рисунку 3.8.

```jsx
import React from "react";
import { Link } from "react-router-dom";

export default function Navbar() {
  return (
    <nav className="navbar navbar-dark bg-primary">
      <div className="container-fluid">
        <a className="navbar-brand mx-3 mx-lg-5">TNTU Bank</a>

        <form className="d-flex">
          <Link
            className="btn btn-outline-light mx-3 mx-lg-5"
            type="button"
            to={`/`}
          >
            Sign out
          </Link>
        </form>
      </div>
    </nav>
  );
}

```

Рисунок 3.7 – Реалізація Navbar

Рисунок 3.8 – Остаточний вигляд компонента Navbar

Отже, реалізувавши всі компоненти які ми хотіли можна приступати до тестування. Звичайно всі вони будуть допрацьовуватися у майбутньому і вцілому веб сервіс ну настільки функціональний , так як хотілося, але все буде допрацьовуватися. Також під час розробки цих компонентів могли бути якісь недопрацювання, які будуть виявлені під час тестування у наступному пункті цієї роботи.

3.2 Тестування функціональності та продуктивності системи

Тестування є невід'ємною частиною розробки програмного забезпечення, особливо коли мова йде про банківські сервіси, де важливі не лише функціональні можливості, але й продуктивність та безпека. Цей розділ присвячений методам тестування функціональності та продуктивності системи, розробленої з використанням React JS [1]. Для тестування веб сервісу було використано Unit-тестування, Google Lighthouse, тестування компонентів React та тестування інтерфейсу користувача.

- Використання Google Lighthouse: Google Lighthouse є потужним інструментом, розробленим Google, який допомагає аналізувати та оцінювати різні аспекти якості та продуктивності веб-додатків. Під час розробки банківського сервісу використання Google Lighthouse є важливим кроком для перевірки таких параметрів, як продуктивність завантаження, доступність, використання кешування, SEO та інші. Звіти, що надаються Google Lighthouse, містять рекомендації щодо покращення якості та продуктивності веб-додатку.

- Тестування дизайну: Дизайн є важливим аспектом банківського сервісу, оскільки він безпосередньо впливає на користувацький досвід. Тестування дизайну включає перевірку розташування елементів, кольорової палітри, типографіки, взаємодії та інших аспектів дизайну. Забезпечення відповідності дизайну до вимог, встановлених для банківського сервісу, допомагає забезпечити зручний та привабливий інтерфейс для користувачів.

- Unit-тестування: Unit-тестування є методом тестування, за яким окремі модулі та компоненти програмного забезпечення перевіряються на правильну роботу. У випадку розробки банківського сервісу unit-тестування включає перевірку функціональності окремих компонентів, реагування на події, валідацію введених даних та інші аспекти. Це допомагає виявити та виправити можливі

помилки та недоліки на ранніх етапах розробки, забезпечуючи стабільність та надійність сервісу.

- Тестування React компонентів: У розробці банківського сервісу я використовував фреймворк React для побудови користувацького інтерфейсу. Тестування у середовищі React включає в себе перевірку правильності рендерингу компонентів, взаємодії між компонентами та перевірку функціональності, яка залежить від специфічних особливостей React. Для цього використовуються спеціальні інструменти тестування React, такі як Enzyme або React Testing Library, які допомагають забезпечити якісне тестування компонентів у середовищі React.

Отже приступимо тестування із Google Lighthouse. Використовуючи цей інструмент можна перевірити наступні компоненти сервісу: продуктивність, швидкість завантаження та доступність. На рисунку 3.9 можна побачити те що всі ці перевірки пройшли вдало.

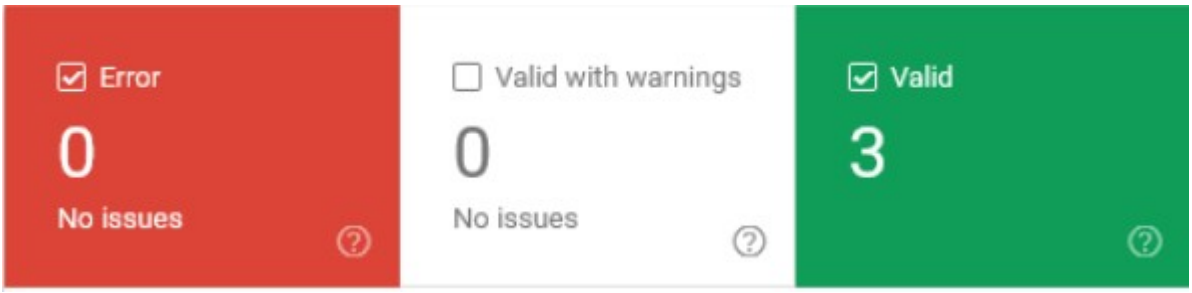

Рисунок 3.9 – Висновок тесту у Lighthouse

Тестування дизайну допомагає зрозуміти, чи буде комфортно користувачу знаходитися на нашім сервісі, чи кольорова палітра не буде дуже неприємною для них і чи буде інтерфейс інтуїтивним. Отже почнемо із кольорової палітри. Вона має бути не дуже відразливих кольорів. Для сервісу я обрав наступні кольори: білий фон і сині кнопки із навігаційною панеллю. Це нейтральні кольори які у своєму поєднанні не є занадто яскравими. Також всі елементи є доволі інтуїтивними, щоб у них можна було з легкістю розібратися. Приклад наведений на рисунку 3.10.

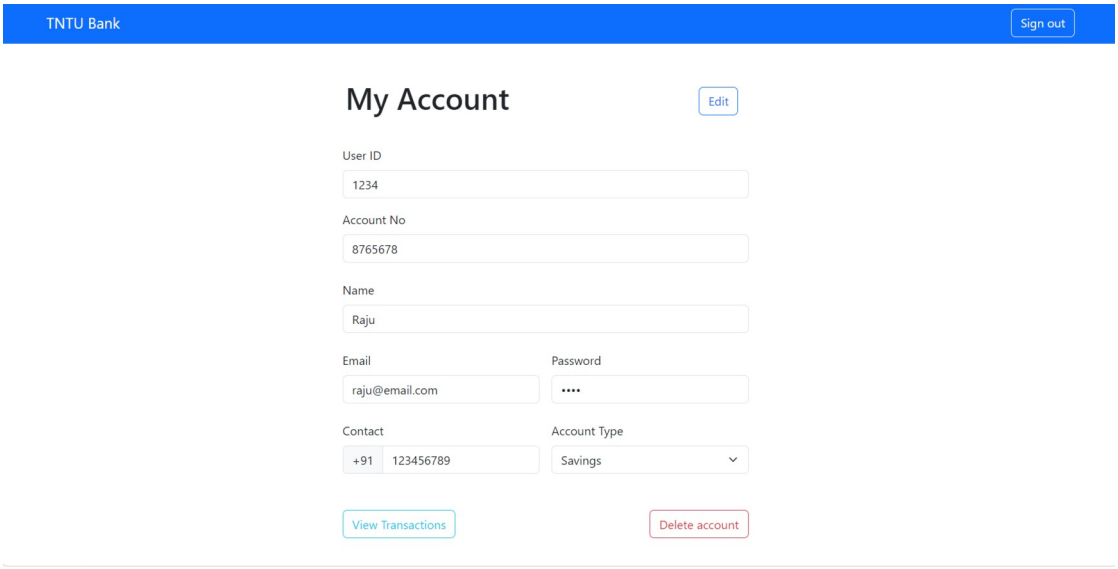

Рисунок 3.10 – Вигляд банківського сервісу

Також два пункти тестування я з’єднав у один і протестував компоненти за допомогою unit-тестів. Під час виконання цих тестів все пройшло вдало і компоненти виконують поставлені вказівки.

Отже, підводячи підсумки, ми розробили доволі вдалий продукт, який буде правильно функціонувати. Звичайно він буде підтримуватися і оновлюватися, щоб у майбутньому він став конкурентно спроможнім на глобальному ринку.

3.3 Оцінка ефективності та зручності використання розробленого сервісу

Ефективність і зручність використання банківського сервісу для перегляду стану рахунків та переказу коштів з використанням бібліотеки ReactJS є критичними аспектами, які визначають успіх проекту. У цьому розділі розглянемо ключові параметри, за якими можна оцінити даний сервіс, зокрема: продуктивність, користувацький досвід, надійність, безпека та відповідність сучасним стандартам.

Продуктивність додатка є одним з основних критеріїв оцінки ефективності. Вона визначається швидкістю завантаження сторінок, відгуком на дії користувачів та загальною швидкодією системи.

Завдяки використанню технології SPA, всі необхідні ресурси завантажуються одноразово при першому відвідуванні сторінки. Це значно скорочує час завантаження при наступних взаємодіях користувача з додатком. Локальне зберігання даних (кешування) дозволяє миттєво оновлювати інформацію на сторінці, що робить роботу з сервісом більш швидкою.

Завдяки ReactJS та його віртуальному DOM (Virtual DOM) зміни на сторінці здійснюються швидко та ефективно. Це забезпечує миттєву реакцію на дії користувачів, що важливо для таких функцій, як перегляд стану рахунку та здійснення переказів.

Користувацький досвід включає в себе легкість у використанні, інтуїтивність інтерфейсу та задоволення користувачів від взаємодії з додатком.

Інтерфейс користувача, розроблений за допомогою ReactJS, дозволяє створювати компоненти, які легко повторно використовувати та оновлювати. Простий і зрозумілий дизайн інтерфейсу дозволяє користувачам швидко знаходити необхідну інформацію та виконувати фінансові операції.

Регулярний збір зворотного зв'язку від користувачів через опитування та аналітичні інструменти (наприклад, Google Analytics) допомагає визначити сильні та слабкі сторони додатка. На основі цієї інформації можна проводити коригування для покращення користувацького досвіду.

Надійність додатка визначається його здатністю працювати без збоїв, обробляти помилки та забезпечувати стабільну роботу під навантаженням.

Використання таких бібліотек, як Sentry, дозволяє відслідковувати та обробляти помилки в реальному часі. Це допомагає швидко виявляти та виправляти проблеми, що виникають у користувачів.

Регресійне тестування, включаючи автоматизовані тести з використанням Jest і React Testing Library, забезпечує стабільну роботу додатка після внесення

змін. Це зменшує ймовірність виникнення помилок та підвищує надійність сервісу.

Безпека банківського додатка є пріоритетом, оскільки він обробляє конфіденційну фінансову інформацію користувачів.

Використання безпечних методів аутентифікації (наприклад, OAuth 2.0, JWT) гарантує, що доступ до додатка мають тільки авторизовані користувачі. Двофакторна аутентифікація (2FA) додає додатковий рівень захисту.

Всі дані, що передаються між клієнтом та сервером, шифруються за допомогою SSL/TLS. Це запобігає перехопленню даних під час передачі.

Використання сучасних методів захисту від таких атак, як XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery), забезпечує безпеку додатка. ReactJS вбудовано містить захист від XSS, що робить його додатково безпечним для розробки інтерфейсів.

Відповідність сучасним веб-стандартам та вимогам є важливим аспектом, який впливає на зручність використання та подальший розвиток додатка.

Додаток розроблений з урахуванням принципів адаптивного дизайну, що дозволяє йому коректно відображатися на різних пристроях (десктопах, планшетах, смартфонах). Це забезпечує доступність сервісу для широкого кола користувачів.

Використання принципів доступності, таких як WCAG (Web Content Accessibility Guidelines), гарантує, що додаток буде зручним для користувачів з обмеженими можливостями.

Для кількісної оцінки ефективності та зручності використання розробленого сервісу можуть використовуватись такі показники:

1. Час завантаження сторінок: Оптимальний час завантаження не перевищує 2-3 секунд.

2. Рівень задоволеності користувачів: Визначається через опитування та відгуки, цільовий рівень – не менше 85% позитивних відгуків.

3. Частота виникнення помилок: Кількість критичних помилок на 1000 запитів не повинна перевищувати 0,1%.

4. Конверсія користувачів: Відсоток користувачів, які здійснюють перекази або інші фінансові операції, від загальної кількості зареєстрованих користувачів.

Розроблений банківський сервіс для перегляду стану рахунків та переказу коштів з використанням ReactJS є ефективним та зручним для користувачів завдяки високій продуктивності, інтуїтивному інтерфейсу, надійності та безпеці. Відповідність сучасним стандартам та використання передових технологій забезпечують високий рівень задоволеності користувачів та ефективність роботи сервісу. Регулярний моніторинг та оптимізація роботи додатка дозволять підтримувати високу якість обслуговування та адаптувати сервіс до змінних потреб користувачів.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Категорійно-понятійний апарат з безпеки життєдіяльності

Категорійно-понятійний апарат з безпеки життєдіяльності охоплює низку ключових понять та категорій, що забезпечують систематизацію знань у цій сфері. Наведені нижче терміни та поняття є фундаментальними для розуміння і вивчення безпеки життєдіяльності.

Безпека життєдіяльності – це стан захищеності особи та суспільства від загроз, що виникають у повсякденному житті та діяльності, і забезпечення умов для їхнього здорового та гармонійного розвитку. Розрізняють такі складові: фізична, соціальна, економічна, екологічна безпека [11].

Загроза – це потенційно небезпечна подія або дія, яка може призвести до негативних наслідків для здоров’я, життя людини, матеріальних та культурних цінностей. Розрізняють такі типи загроз: природні, технологічні, соціальні та екологічні .

Ризик – це імовірність виникнення небажаних подій та величина їх можливих наслідків. Існують такі компоненти ризиків: оцінка ризику, управління ризиком та прийнятний ризик.

Безпека – стан, за якого відсутній неприпустимий ризик завдання шкоди. Є такі види безпеки: особиста безпека, суспільна, національна та міжнародна безпека .

Небезпека – це джерело потенційної шкоди для життя, здоров’я людини, майна та навколишнього середовища. Розрізняють наступні категорії небезпек: фізичні, хімічні, біологічні та психологічні небезпеки.

Система безпеки життєдіяльності – це сукупність організаційних, технічних, соціально-економічних і правових заходів, спрямованих на забезпечення безпеки особистості, суспільства та держави. Існують наступні

елементи: органи управління, засоби і методи контролю, нормативно-правова база [11].

Профілактика небезпек – це комплекс заходів, спрямованих на запобігання виникненню небезпечних ситуацій і мінімізацію їх наслідків. Визначають такі стратегії профілактики: інформування, навчання, технічні заходи та організаційні заходи .

Захист від небезпек – це сукупність дій і засобів, спрямованих на усунення або зменшення впливу небезпек на людину та суспільство. Існують наступні засоби захисту: індивідуальні засоби захисту, колективні засоби захисту та евакуаційні заходи.

Також, у категорійно-понятійний апарат входять наступні принципи безпеки життєдіяльності.

Принцип системного підходу – це принцип, який несе за собою суть розгляду безпеки як комплексної системи, що включає різні компоненти, які взаємодіють між собою.

Принцип превентивності – принцип, який має за мету запобіганню небезпекам на стадії їх зародження, до того як вони стануть реальністю.

Принцип мінімізації наслідків – створює умови, за яких можливі негативні наслідки від небезпечних ситуацій будуть мінімальними.

Принцип адаптивності – це принцип, який забезпечує здатність системи безпеки до гнучкого реагування на зміни в зовнішньому середовищі та швидкого пристосування до нових умов.

Принцип відповідальності – чітко визначає відповідальності за забезпечення безпеки на різних рівнях: особистому, організаційному, державному.

Особиста безпека – це заходи та дії, що здійснюються кожною окремою особою для забезпечення власної безпеки. Є наступні компоненти: самозахист, здоровий спосіб життя та навички безпечної поведінки.

Суспільна безпека – це система заходів, спрямованих на забезпечення безпеки груп людей, суспільних утворень та всього суспільства. Компоненти: охорона громадського порядку, соціальний захист, система охорони здоров'я [11].

Національна безпека – це захист суверенітету, територіальної цілісності та інтересів держави. Існують наступні компоненти: військова безпека, економічна безпека та інформаційна безпека.

Міжнародна безпека – це система заходів і механізмів, спрямованих на забезпечення безпеки у міжнародному масштабі.

Також, потрібно додати сюди нормативно-правове забезпечення БЖД, такі як:

1. міжнародні документи;
2. національні закони та акти;
3. стандарти та нормативи.

Щоб закінчити категорійно-понятійний апарат потрібно додати ще основні підходи до управління безпекою життєдіяльності. Сюди потрібно віднести наступні підходи:

1. Організаційний підхід. Суть цього підходу полягає в розробці та впроваджені організаційних заходів, планів та інструкцій для забезпечення безпеки;
2. Технічний підхід. Використання технічних засобів та інновацій для попередження та ліквідації небезпек;
3. Психологічний підхід. Суть: формування психологічної готовності та стресостійкості у населення для адекватного реагування на небезпеки.
4. Освітній підхід. Вся суть цього підходу являє собою проведення навчально-інформаційних заходів для підвищення обізнаності та компетентності в питаннях безпеки.
5. Науково-дослідний підхід. Проведення наукових досліджень для виявлення нових загроз і розробки ефективних методів їхнього попередження.

Ці поняття та категорії утворюють основу для систематизованого вивчення і розуміння безпеки життєдіяльності, дозволяючи ефективно управляти ризиками та забезпечувати захист особи, суспільства і держави.

4.2 Розрахунок економічної ефективності заходів щодо поліпшення санітарно-гігієнічних умов

Впровадження заходів з охорони праці забезпечує соціально-економічні результати, які відображаються в соціально-економічному ефекті [12].

Взаємозв'язок економічних і соціальних ефектів визначається їх спільною природою: економічний ефект є матеріальною основою соціального, а соціальний ефект створює кращі умови праці, сприяючи зростанню економічного ефекту.

Більшість заходів, що створюють соціальний ефект, сприяють підвищенню рівня виробництва, зниженню трудомісткості, тобто покращують показники, які характеризують економічний ефект.

Оцінка ефективності заходів щодо покращення умов та охорони праці здійснюється за такими групами показників:

- зміни стану умов праці;
- соціальні;
- соціально-економічні;
- економічні.

Зміни стану умов праці оцінюються підвищенням рівня безпеки, покращенням санітарно-гігієнічних, психофізіологічних і естетичних показників.

Підвищення рівня безпеки визначається збільшенням кількості технологій, механізмів, машин, приміщень та інших об'єктів, які приведені у відповідність до вимог нормативних документів [12].

Показник числа робочих місць, приведених у відповідність санітарно-гігієнічним вимогам, визначають за формулою:

Ксч = Есрм/Чрм

де Есрм – приріст числа робочих місць, на яких поліпшені умови праці після впровадження заходів;

Чрм – загальне число робочих місць.

Покращення санітарно-гігієнічних показників пов'язане зі зниженням рівня вібрації, шуму, випромінювань тощо [12].

Скорочення числа робітників, умови праці яких не відповідають санітарно-гігієнічним нормативам, визначається за показниками поліпшення умов праці на робочому місці:

Чсл = Еу/Ч

де Еу – приріст робітників, у яких поліпшені умови праці;

Ч – загальне число працюючих, чол.

Будь-який соціальний результат може бути оцінений економічно, з певним ступенем достовірності.

Результати досліджень свідчать про те, що підвищення продуктивності праці досягається за рахунок раціональної організації робочого місця на 20%, правильної організації освітлення робочих місць – на 6-13%, раціонального пофарбування приміщень – на 25% [12].

Водночас продуктивність праці знижується при невиконанні вимог охорони праці (наприклад, при температурі 20-30° становить менше 20-50% від необхідного рівня).

Залежно від виду заходів та їх кінцевої мети проводять комплексну оцінку соціальних та економічних результатів.

До основних показників економічної ефективності належать: збільшення продуктивності праці; річний економічний ефект; коефіцієнт економічної ефективності; строк окупності [12].

Збільшення кількості та питомої ваги об'єктів (машин, механізмів, виробничих приміщень), приведених у відповідність до вимог стандартів безпеки та інших нормативів, визначається за формулою:

Коб = Ео/Чо

де Ео – приріст числа машин, механізмів, які відповідають вимогам безпеки;

Чо – загальне число об'єктів, од.

Показники захворюваності та виробничого травматизму розраховують у такий спосіб:

Ксз = Ез/Ч, Ктз = Етз/Ч, Кч = Еч/Псз, Кт = Ет/Пт

де:

- Ксз – показник зниження числа випадків захворювань із тимчасовою втратою працездатності через несприятливі умови праці в розрахунку на одну людину;

- Ктз – показник тривалості захворювань;

- Кч – показник частоти травматизму;

- Кт – показник тяжкості травматизму;

- Ез, Етз – відповідно зменшення числа випадків захворювань та тривалість захворювань;

- Ч – загальне число працюючих;

- Псз – число випадків захворювань у поточному році;

- Еч, Ет – відповідно різниця числа випадків та тяжкості травматизму до і після впровадження заходів;

- Пт – число травм у звітному році.

При плануванні заходів проводять короткострокове прогнозування травматизму або захворюваності на основі динаміки за три роки або планують повне усунення травматизму та захворюваності в результаті впровадження заходів.

ВИСНОВКИ

У ході виконання бакалаврської роботи було розглянуто теоретичні та практичні аспекти розробки банківських сервісів, зокрема, створення системи для перегляду стану рахунків. Основні результати та досягнення роботи можна підсумувати наступним чином:

1. Теоретичний аналіз: У роботі проведено глибокий аналіз сучасних тенденцій у банківській сфері, зокрема використання цифрових технологій для покращення якості обслуговування клієнтів. Визначено основні концепції банківських сервісів, які включають автоматизацію процесів, забезпечення безпеки та зручність користування.

2. Технологічні рішення: Вибір технологій для реалізації проєкту було обґрунтовано аналізом їхніх можливостей та відповідності вимогам завдання. Основним інструментом для розробки користувацького інтерфейсу обрано бібліотеку ReactJS, яка забезпечує високу продуктивність та зручність у використанні. Використання Visual Studio Code як основного середовища розробки сприяло підвищенню ефективності роботи завдяки його функціональності та гнучкості.

3. Проектування та реалізація: Було розроблено архітектуру системи, яка включає клієнтську та серверну частини, а також базу даних для зберігання інформації про рахунки клієнтів. Основні функції сервісу, такі як перегляд стану рахунків, виконання транзакцій та генерація звітів, були реалізовані за допомогою ReactJS та Node.js.

4. Тестування та оцінка: Проведено комплексне тестування функціональності та продуктивності системи, яке включало як автоматизовані, так і ручні тести. За результатами тестування було виявлено та усунуто декілька недоліків, що дозволило забезпечити стабільну роботу системи. Крім того, проведено оцінку зручності використання сервісу шляхом опитування

потенційних користувачів, що дозволило внести деякі покращення в інтерфейс користувача.

5. Безпека та охорона праці: Особлива увага приділялася питанням безпеки життєдіяльності та охорони праці розробників програмного забезпечення. Було проведено аналіз умов праці, організовано робочі місця відповідно до санітарно-гігієнічних норм та розроблено рекомендації щодо запобігання професійним захворюванням.

6. Економічна ефективність: Оцінка економічної ефективності впроваджених заходів з охорони праці показала позитивний соціально-економічний ефект. Впровадження сучасних технологій та покращення умов праці сприяло підвищенню продуктивності праці, зниженню трудомісткості та покращенню загальних показників економічної ефективності підприємства.

Таким чином, бакалаврська робота досягла поставлених цілей, забезпечивши всебічний аналіз та реалізацію сучасного банківського сервісу. Результати дослідження можуть бути використані як основа для подальшого розвитку та вдосконалення банківських послуг, а також для поліпшення умов праці розробників програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Quick Start [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn.

2. The Modern JavaScript Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.info/.

3. Visual Studio Code Docs [Електронний ресурс] – Режим доступу до ресурсу:https://code.visualstudio.com/Docs.

4. Bootstrap Docs [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

5. React Router Tutorial[Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/en/main/start/tutorial.

6. JSON Server [Електронний ресурс] – Режим доступу до ресурсу: https://json-server.dev/.

7. Axios Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro.

8. HTML Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/html/.

9. Redux Essentials [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/tutorials/essentials/part-1-overview-concepts.

10. How to Use the React Context API in Your Projects [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/context-api-in-react/.

11. Лєськів Г. З., Верескля М. Р. (2018). Безпека Життєдіяльності Та Охорона Праці . Львів.

12. Сидоренко, М. П. (2020). Аналіз економічної ефективності інвестицій у здоров'я працівників. Харків: ХНАДУ.

ДОДАТКИ

Лістинг 1 – код LoginPage

```
import axios from "axios";
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";

export default function LoginPage() {
  const url = "http://localhost:3000/users";
  const [login, setLogin] = useState({
    userId: "",
    password: "",
  });

  const [invalid, setInvalid] = useState("d-none");

  let navigate = useNavigate();

  const handleChange = (e) => {
    setLogin({
      ...login,
      [e.target.name]: e.target.value,
    });
    setInvalid("d-none");
  };

  const checkLogin = async () => {
    const loginDetails = await axios
      .get(url)
      .then((res) => {
        let userIndex;
        const allDatas = res.data;
        const userData = allDatas.find((user, index) => {
          userIndex = index;
          return (
            user.customerId == login.userId &&
            user.customerPin == login.password
          );
        });
        console.log(userData);
        if (userData) {
          navigate(`/account/${userIndex}`);
        } else {
          setInvalid("d-block");
```

```
            }
          })
          .catch((err) => console.log(err));
      };

      return (
        <div
          style={{
            background: "#007bff",
            background: "linear-gradient(to right, #0062E6, #33AEFF)",
            width: "100%",
            height: "100vh",
            overflow: "hidden",
          }}
        >
          <p
            className="fs-1 text-light text-center mt-5"
             style={{ fontFamily: "Tilt Prism", textShadow: "3px 2px 3px
blue" }}
          >
            Welcome to TNTU Bank
          </p>
          <div className="row">
              <div  className="col-sm-9  col-md-7  col-lg-6  col-xl-4  m-auto
px-5">
                <div  className="card  border-0  shadow  rounded-3  my-5  bg-
light">
                <div className="card-body p-4 p-sm-5">
                  <h5 className="card-title text-center mb-5 fw-light fs-
5 fw-semibold">
                    Sign In
                  </h5>
                  <form>
                    <div className="form-floating mb-3">
                      <input
                        type="number"
                        className="form-control"
                        id="floatingInput"
                        placeholder="Email address"
                        value={login.userId}
                        onChange={handleChange}
                        name="userId"
                      />
                      <label htmlFor="floatingInput">User ID</label>
                    </div>
                    <div className="form-floating mb-3">
                      <input
                        type="password"
                        className="form-control"
```

```
                        id="floatingPassword"
                        placeholder="Password"
                        autoComplete="true"
                        value={login.password}
                        onChange={handleChange}
                        name="password"
                      />
                      <label htmlFor="floatingPassword">User PIN</label>
                    </div>
                         <p  className={`text-danger  fw-semibold  ms-2  ${invalid}`}>
                      Invalid ID or PIN
                    </p>
                    <div className="form-check mb-3">
                      <input
                        className="form-check-input"
                        type="checkbox"
                        value=""
                        id="rememberPasswordCheck"
                      />
                      <label
                        className="form-check-label"
                        htmlFor="rememberPasswordCheck"
                      >
                        Remember password
                      </label>
                    </div>
                    <div className="d-grid mt-4">
                      <button
                            className="btn  btn-primary  text-uppercase  fw-semibold"
                        type="button"
                        onClick={checkLogin}
                      >
                        Sign in
                      </button>
                    </div>
                  </form>
                </div>
              </div>
            </div>
          </div>
        );
      }
```

Лістинг 2 – код UserPage

```
import axios from "axios";
import React, { useEffect, useRef, useState } from "react";
import { Link, useNavigate, useParams } from "react-router-dom";

export default function UserPage() {
  const url = "http://localhost:3000/users";

  let navigate = useNavigate();

  const { index } = useParams();

  const inputRef = useRef(null);

  const [user, setUser] = useState({
    userId: "",
    accountNumber: "",
    accountHolder: "",
    accountType: "",
    contact: "",
    email: "",
    password: "",
  });

  const [readOnly, setReadOnly] = useState(true);

  const handleChange = (e) => {
    setUser({
      ...user,
      [e.target.name]: e.target.value,
    });
  };

  const loadData = async () => {
    const users = await axios.get(url).then((res) => {
      const mainUser = res.data[index];

      setUser({
        ...user,
        accountNumber: mainUser.customerAccountNo,
        accountHolder: mainUser.customerName,
        accountType: mainUser.customerAccountType,
        contact: mainUser.customerContact,
        email: mainUser.customerEmail,
        password: mainUser.customerPin,
        userId: mainUser.customerId,
      });
    });
  };
```

```
        useEffect(() => {
          loadData();
        }, []);

        const handleClick = (e) => {
          setReadOnly(false);
          inputRef.current.focus();
        };

        const handleDelete = async () => {
          alert(
               "Are you sure you want to delete with account no " +
user.accountNumber
          );
          navigate("/");
        };

        return (
           <div className="container m-auto row mt-5 d-flex align-items-
center col-lg-5">
            <div className="col-8">
              <h1 className="text-start">My Account</h1>
            </div>
            <div className="col-3">
              <button
                className="btn btn-outline-primary float-end"
                onClick={handleClick}
              >
                Edit
              </button>
            </div>

            <div id="about-areas" className="row col-auto">
              <form className="row g-3">
                <div className="col-12 mt-lg-3">
                  <label htmlFor="inputAddress3" className="form-label">
                    User ID
                  </label>
                  <input
                    type="number"
                    className="form-control"
                    id="inputAddress3"
                    name="accountNumber"
                    value={user.userId}
                    onChange={handleChange}
                    readOnly={true}
                    onWheel={(e) => e.target.blur()}
                  />
                </div>
```

```
<div className="col-12 mt-lg-3">
  <label htmlFor="inputAddress" className="form-label">
    Account No
  </label>
  <input
    type="number"
    className="form-control"
    id="inputAddress"
    name="accountNumber"
    value={user.accountNumber}
    onChange={handleChange}
    readOnly={true}
    onWheel={(e) => e.target.blur()}
  />
</div>
<div className="col-12 mt-lg-4">
  <label htmlFor="inputAddress2" className="form-label">
    Name
  </label>
  <input
    type="text"
    className="form-control"
    id="inputAddress2"
    name="accountHolder"
    value={user.accountHolder}
    onChange={handleChange}
    readOnly={readOnly}
    ref={inputRef}
  />
</div>
<div className="col-md-6 mt-lg-4">
  <label htmlFor="inputEmail4" className="form-label">
    Email
  </label>
  <input
    type="email"
    className="form-control"
    id="inputEmail4"
    name="email"
    value={user.email}
    onChange={handleChange}
    readOnly={readOnly}
  />
</div>
<div className="col-md-6 mt-lg-4">
  <label htmlFor="inputPassword4" className="form-label">
    Password
  </label>
  <input
```

```
    type="password"
    className="form-control"
    id="inputPassword4"
    name="password"
    value={user.password}
    onChange={handleChange}
    autoComplete="true"
    readOnly={readOnly}
  />
</div>

<div className="col-md-6 mt-lg-4">
  <div className="row ">
    <label htmlFor="contact" className="form-label">
      Contact
    </label>
    <div className="input-group">
      <span className="input-group-text" id="basic-addon1">
        +91
      </span>
      <input
        type="number"
        className="form-control"
        aria-label="Username"
        aria-describedby="basic-addon1"
        id="contact"
        name="contact"
        value={user.contact}
        readOnly={readOnly}
        onChange={handleChange}
        onWheel={(e) => e.target.blur()}
      />
    </div>
  </div>
</div>
<div className="col-md-6 mt-lg-4">
  <label htmlFor="inputState" className="form-label">
    Account Type
  </label>
  <select
    id="inputState"
    className="form-select"
    name="accountType"
    onChange={handleChange}
    value={user.accountType}
  >
    <option value="current" disabled={readOnly}>
      Current
    </option>
```

```
                    <option value="savings" disabled={readOnly}>
                      Savings
                    </option>
                  </select>
                </div>

                <div className="col-12 my-4 mt-lg-5 d-flex justify-content-
between">
                  <Link
                    type="button"
                    className="btn btn-outline-info"
                    to={`/transactions/${index}`}
                  >
                    View Transactions
                  </Link>
                  <button
                    type="button"
                    className="btn btn-outline-danger"
                    onClick={handleDelete}
                  >
                    Delete account
                  </button>
                </div>
              </form>
            </div>
          </div>
        );
      }
```

Лістинг 3 – код ViewTransaction

```
import axios from "axios";
import React, { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";

export default function ViewTransactions() {
  const url = "http://localhost:3000/users";

  let navigate = useNavigate();

  let { index } = useParams();

  const [filterType, setFilterType] = useState("none");

  const [transaction, setTransaction] = useState([]);

  const handleChange = (e) => {
    setFilterType(e.target.value);
```

```
        };

        const transactions = async () => {
          const datas = await axios.get(url);
          const userTransactions = datas.data[index].customerTransactions;
          setTransaction(userTransactions);
        };

        useEffect(() => {
          transactions();
        }, []);

        return (
           <div className="d-flex flex-column justify-content-center align-
items-center">
            <div className="col-3 col-lg-1 m-5">
                        <select   name=""   id=""   className="form-select"
onChange={handleChange}>
              <option hidden={true}>filter by</option>
              <option value="credit">credit</option>
              <option value="debit">debit</option>
              <option value="none">none</option>
            </select>
            </div>
             <table  className="table  text-center  w-75  m-auto  mt-3  shadow
border ">
              <thead className="bg-light">
                <tr>
                  <th scope="col">Transaction ID</th>
                  <th scope="col">Date</th>

                  <th scope="col">Type</th>
                  <th scope="col">Amount</th>
                </tr>
              </thead>
              <tbody>
                {transaction.map((data, index) => (
                                  <TableDatas    data={data}    key={index}
filter={filterType} />
                ))}
              </tbody>
            </table>
            <div>
              <button className="btn btn-success ">New Transaction</button>
              <button
                className="btn btn-secondary m-5"
                onClick={() => navigate(`/account/${index}`)}
              >
                Back to Home
```

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

const TableDatas = ({ data, filter }) => {
  return filter == data.transactionType ? (
    <tr>
      <th scope="row">{data.transactionId}</th>
      <td>{data.transactionDate}</td>
      <td>{data.transactionType}</td>
      <td
        style={
          data.transactionType == "credit"
            ? { backgroundColor: "#ccffcc" }
            : { backgroundColor: "#ffaaaa" }
        }
      >
        {data.transactionAmount}
      </td>
    </tr>
  ) : filter == "none" ? (
    <tr>
      <th scope="row">{data.transactionId}</th>
      <td>{data.transactionDate}</td>
      <td>{data.transactionType}</td>
      <td
        style={
          data.transactionType == "credit"
            ? { backgroundColor: "#ccffcc" }
            : { backgroundColor: "#ffaaaa" }
        }
      >
        {data.transactionAmount}
      </td>
    </tr>
  ) : (
    ""
  );
};
```

Лістинг 4 – код Navbar

```
import React from "react";
import { Link } from "react-router-dom";
```

```
export default function Navbar() {
  return (
    <nav className="navbar navbar-dark bg-primary">
      <div className="container-fluid">
        <a className="navbar-brand mx-3 mx-lg-5">TNTU Bank</a>

        <form className="d-flex">
          <Link
            className="btn btn-outline-light mx-3 mx-lg-5"
            type="button"
            to={`/`}
          >
            Sign out
          </Link>
        </form>
      </div>
    </nav>
  );
}
```

Лістинг 5 – код App

```
import "bootstrap/dist/css/bootstrap.min.css";
import LoginPage from "./components/LoginPage";
import { BrowserRouter as Router, Route, Routes } from "react-router-
dom";
import MainLayout from "./MainLayout";

function App() {
  return (
    <div>
      <Router>
        <Routes>
          <Route index path="/" element={<LoginPage />} />
          <Route path="*" element={<MainLayout />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```

Додаток Б – Диск