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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-календаря для створення нотаток-нагадувань на основі React JS з використанням TypeScript

Виконав: студент IV курсу, групи СТ-41

спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

_____ Бачинський А.В.
(підпис) (прізвище та ініціали)

Керівник _____ Приймак М.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Марценко С.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

Тернопіль
2023

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

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

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 126 Інформаційні системи та технології
(шифр і назва спеціальності)

Студенту Бачинському Артуру Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-календаря для створення нотаток-нагадувань на основі React JS з використанням TypeScript

Керівник роботи Приймак Микола Володимирович, доктор технічних наук, професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «7» лютого 2023 року № 4/7-135

2. Термін подання студентом завершеної роботи 14 червня 2023р.

3. Вихідні дані до роботи Документаційні інтернет-джерела щодо використання веб технологій

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Побудова файлової структури завдання та опис використаних технологій. 1.1 Вибір середовища написання коду. 1.2 Файлова структура проекту. 1.3 Історія та опис мови типізації TypeScript. 1.4 Сучасна бібліотека React JS. 1.5 Бібліотека для керування станом глобальних даних Redux. 1.6 Висновок до першого розділу. Розділ 2. Проектування та реалізація інтерфейсу веб-календаря. 2.1 Створення маршрутизації додатку. 2.2 Керування глобальними даними. 2.3 Реалізація посторінкової навігації. 2.4 Функціонал авторизації у додаток. 2.4.1 Валідація входу. 2.5 Реалізація інтерфейсу та логіки календаря. 2.5.1 Створення функціоналу для додавання нотаток. 2.5.2 Валідація створення нотаток. 2.6 Висновок до другого розділу. Розділ 3. Безпека життєдіяльності, основи хорони праці. 3.1 Шляхи підвищення життєдіяльності людини. 3.2 Санітарно-гігієнічні вимоги до умов праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.Титульна сторінка. 2. Мета роботи. 3. Актуальність обраних технологій. 4. Етап розробки авторизації. 5. Етап розробки календаря. 6. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Окіпний І.Б., к.т.н., доцент		

7. Дата видачі завдання 23 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	08.05.2023	*Виконано*
2.	Підбір джерел про середовище для розробки проекту	09.05.2023-11.05.2023	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	12.05.2023-15.05.2023	*Виконано*
4.	Виконання дослідження щодо технологій для	16.05.2023-22.05.2023	*Виконано*
	проектування та розробки календаря		
5.	Оформлення розділу «Побудова файлової структури	23.05.2023-25.05.2023	*Виконано*
	завдання та опис використаних технологій»		
6.	Оформлення розділу «Проектування та реалізація	26.05.2023-28.05.2023	*Виконано*
	інтерфейсу веб-календаря»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	29.05.2023-30.05.2023	*Виконано*
8.	Виконання завдання до підрозділу «Основи хорони	31.05.2023-01.06.2023	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.06.2023-06.06.2023	*Виконано*
10.	Нормоконтроль	07.06.2023-08.06.2023	*Виконано*
11.	Перевірка на плагіат	09.06.2023	*Виконано*
12.	Попередній захист кваліфікаційної роботи	12.06.2023	*Виконано*
13.	Захист кваліфікаційної роботи	23.06.2023	

Студент _____ Бачинський А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Приймак М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-календаря для створення нотаток-нагадувань на основі React JS з використанням TypeScript // Кваліфікаційна робота освітнього рівня «Бакалавр» // Бачинський Артур Васильович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СТ-41 // Тернопіль, 2023 // С. 53, рис. – 59, бібліогр. – 16.

Ключові слова: react, typescript, redux, веб-календар, webstorm, vscode, нотатки, редюсер.

Кваліфікаційна робота присвячена розробці веб-сторінки з календарем для створення нотаток-нагадувань з використанням бібліотеки React JS 18.2.0

В першому розділі кваліфікаційної роботи описано архітектуру проекту, та його файлову структуру. Висвітлено сучасні технології напрямку frontend. Розглянуто середовище написання коду WebStorm. Також проаналізовано чому сучасні користувацькі бібліотеки зручні у написанні проектів.

В другому розділі кваліфікаційної роботи досліджено спосіб авторизації користувача зі збереженням інформації про нього у локальному сховищі. Створено функціонал для додавання нотаток та можливість вибрати гостя до цієї нотатки. Був продемонстрований спосіб валідації для створення нотатки тільки у майбутньому часі.

В третьому розділі кваліфікаційної роботи описано які є шляхи для підвищення життєдіяльності людини, також було проаналізовано санітарно-гігієнічні вимови до умов праці людини.

ANNOTATION

Development of Web Calendar for Notes and Reminders Creation by Means of React.Js Using TypeScript // Qualification work of the educational level "Bachelor" // Bachynskiy Artur Vasyliovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group ST-41 // Ternopil, 2023 // P. 53, fig. – 59, references -16

Keywords: react, typescript, redux, web-calendar, webstorm, vscode, notes, reducer.

The qualification work is designed to the development of a web page with a calendar for creating reminder notes using the React JS 18.2.0 library

The first section of the qualification work describes the project architecture and file structure. Modern frontend technologies are highlighted. The WebStorm code writing environment is considered. It is also analyzed why modern UI libraries are convenient for writing projects.

In the second section of the qualification work, was investigated the method of authorizing the user with saving information about him in the local storage. Functionality for creating notes and the ability to select a guest for this note is provided. A validation method was demonstrated to create a note only in the future tense.

In the third section of the qualification work, the ways to improve the human activity are described, and the sanitary and hygienic requirements for human working conditions were also analyzed.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – JavaScript.

TS – TypeScript.

Redux – це інструмент для керування станом даних.

React – це JavaScript бібліотека для створення користувацьких інтерфейсів.

JSON – JavaScript object notation

Store – об’єкт з глобальними даними.

DOM – об'єктна модель документа.

WebStorm – редактор коду.

SRC – (англ. Source – джерело).

Стейт – (англ. State – Стан).

Пропс – (англ. Properties – дані які передаються між компонентами)

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі все більше поширюється використання веб-сторінок, різних додатків та сайтів, які полегшують життя. Вони стають невід'ємною частиною у житті людини. Тому ця галузь завжди буде актуальна, і мати великий попит на клієнтів та розробників.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи є розробка та модернізація веб-календаря. Багато інтернет-календарів зазвичай влаштовані на такій основі, що крім перегляду дати, вони нічого не можуть виконувати. Тому розробка даного проекту полягає в тому, щоб створити веб-календар для можливості записувати у ньому нотатки, та запрошувати до нотаток інших користувачів, така модернізація дозволить користувачам швидко та зручно працювати із календарем.

Для досягнення поставленої мети було виконано ряд наступних завдань:

– Побудувати архітектури проекту.

– Описати веб-технології для виконання завдання.

– Обрати та обґрунтувати середовище для розробки.

– Реалізувати інтерфейс клієнтської частини.

– Розробити функціонал для авторизації користувача, та модернізувати веб-сторінку для додавання гостів у нотатки.

Практичне значення одержаних результатів. Створений у результаті виконання кваліфікаційної роботи веб-календар, дозволяє користувачам додавати нотатки та запрошувати друзів. У подальшому буде легко піддаватись можливим змінам, створені бази даних та більш глибокій модернізації. Даний проект написаний за допомогою сучасної бібліотеки React з використанням мови типізації.

РОЗДІЛ 1. ПОБУДОВА ФАЙЛОВОЇ СТРУКТУРИ ЗАВДАННЯ ТА ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

1.1 Вибір середовища написання коду

Для написання коду існує багато різних цікавих та зручних середовищ, у своїй практиці я працював лише з двома текстовими редакторами, це VSCode, та WebStorm. Попрацювавши на обох редакторах, та порівнявши їх можливості я обрав для себе WebStorm, тому що цей редактор ідеально підходить для програміста який пише на JavaScript.

WebStorm - це інтегроване середовище для розробки на JavaScript та пов'язаних з ним технологій (React, Vue, Angular) і інші. WebStorm існує з 2010 року, він дозволяє автоматизувати рутинну роботу та легко справлятися зі складними завданнями, роблячи розробку більш зручною та цікавою. Засновниками даного продукту є компанія JetBrains.

Які є можливості даного редактора? Він добре розуміє структуру проектів та допомагає з будь-якими аспектами написання коду. Автодоповнення коду, безпечні внесення змін, постійний пошук потенційних проблем та підказки щодо їх виправлення завжди є під рукою (Див. рисунок 1.1).

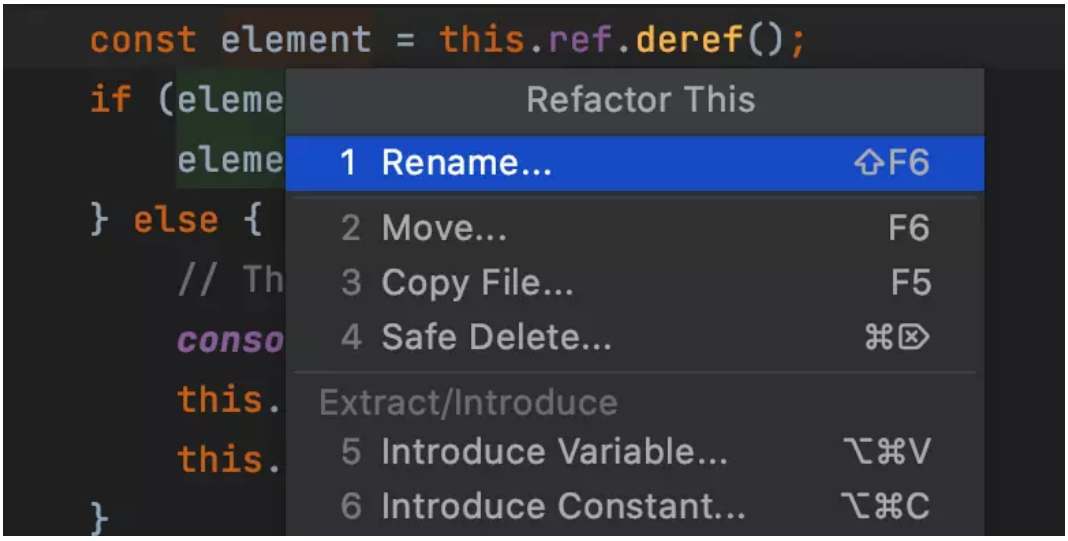

Рисунок 1.1 – Спосіб внесення змін

Швидка навігація та пошук по ключовим словам у коді є також важливою частиною у розробці, адже на пошуки тої чи іншої змінної можна витратити багато часу рукою (Див. рисунок 1.2).

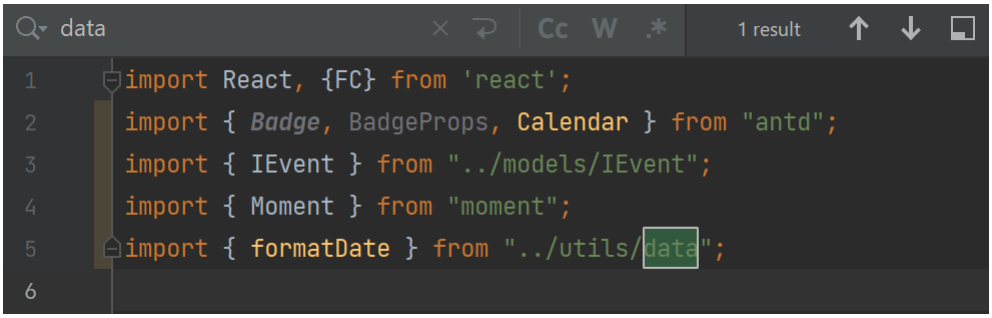

Рисунок 1.2 – Пошук по ключовому слову

Комбінація клавіш (cntrl + f) викликає поле вводу для пошуку необхідних слів чи символів.

1.2 Файлова структура проекту

Правильна побудова структури проекту є не менш важливою частиною у розробці, тому перед самою розробкою проекту я складаю структуру за якою буду йти далі і будувати веб-додаток.

Першим ділом я провів ініціалізацію React JS проекту, та описав усі необхідні папки для подальшої розробки (Див. рисунок 1.3).

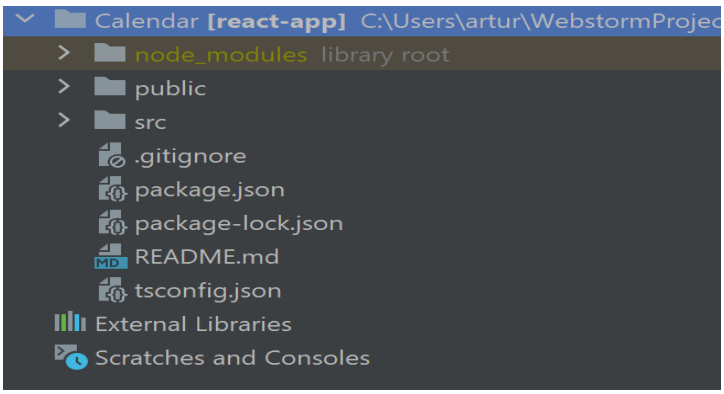

Рисунок 1.3 – Стандартний React проект

У корні проекту знаходиться головна папка “src” – від повного слова “source”, в якій знаходяться інші папки, кожна папка містить в собі файли які відносяться до неї (Див. рисунок 1.4).

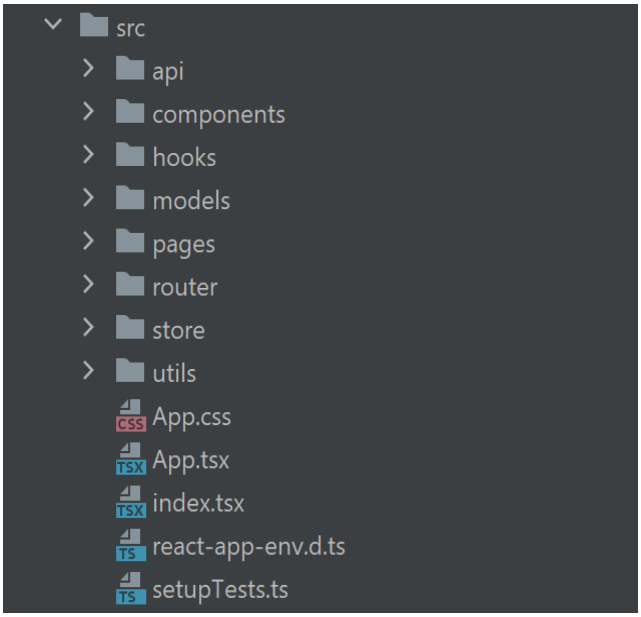

Рисунок 1.4 – Структура кореневої папки

Як можна замітити усі папки розміщені у алфавітному порядку, зверху до низу, також називаються відносно їхньої сутності (Див. рисунок 1.5).

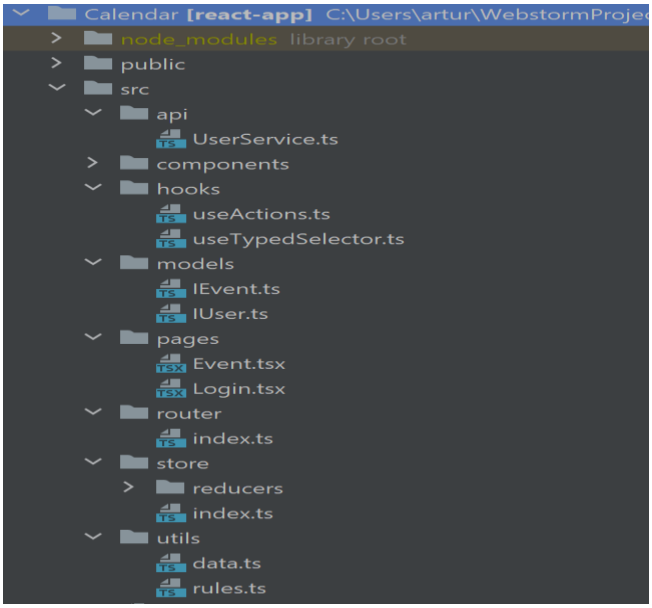

Рисунок 1.5 – Файлова структура

Такий підхід побудови файлової структури зменшує витрати на пошук файлів.

1.3 Історія та опис мови типізації TypeScript

TypeScript має незвичні стосунки з JavaScript. Він пропонує всі функції JavaScript і додатковий рівень на додаток до них: систему типів TypeScript.

Наприклад, JavaScript надає такі умовні примітиви, як рядок і число, але не перевіряє, чи ви їх постійно призначали. TypeScript це робить.

Це означає, що ваш існуючий робочий код написаний на js також є кодом TypeScript. Основна перевага типізованої мови полягає в тому, що вона може висвітлювати неочікувану поведінку у коді, знижуючи ймовірність помилок.

Розуміючи, як працює джава скрипт, TS може створити систему типів, яка приймає код JS, але має типи. Це надає систему типів без необхідності додавати додаткові символи, щоб зробити типи явними у коді. Нижче наведено приклад як працює типізація даних (Див. рисунок 1.6).

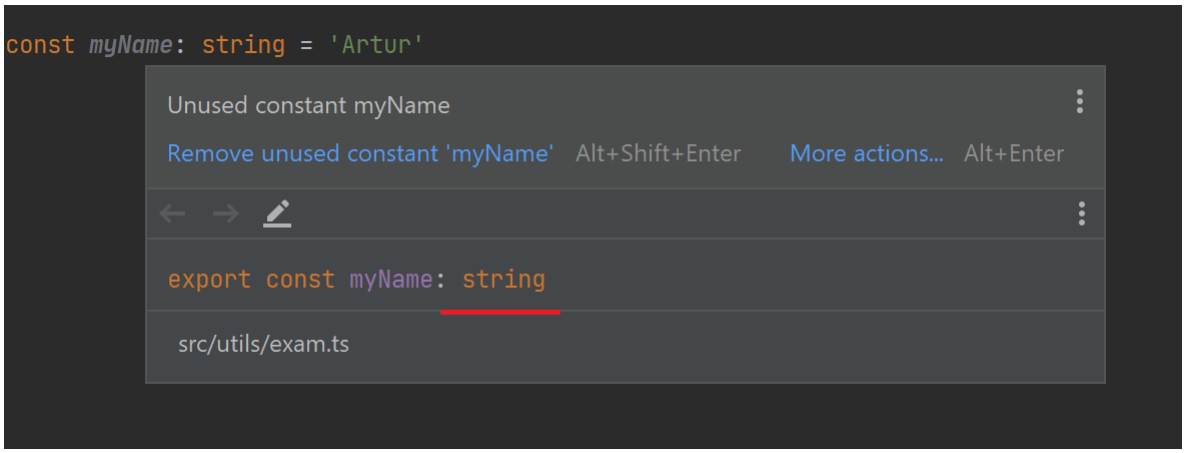

Рисунок 1.6 – Приклад типізованої змінної

Можна зауважити, що перед присвоєнням даних до змінної я надав їй тип “string”, що чітко вказує на те, що ця змінна може мати в собі тільки стрічку.

На рисунку 1.7 зображено як виглядав би такий самий код, але написаний на чистому JS без використання мови типізації (Див. рисунок 1.7).

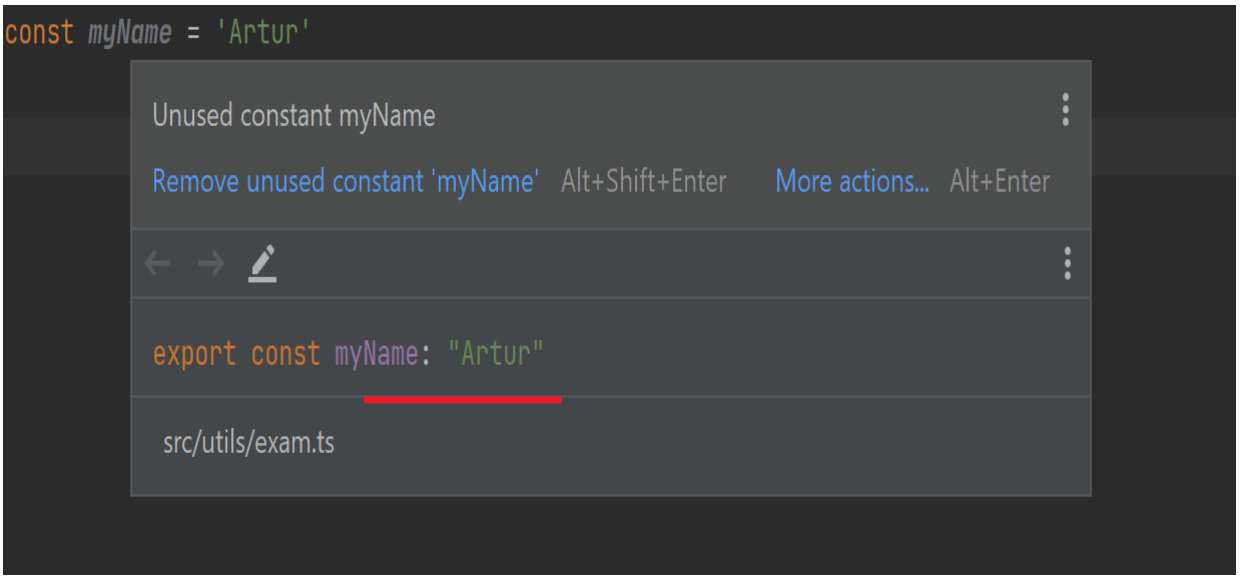

Рисунок 1.7 – Приклад нетипізованої змінної

Як видно на обох рисунках, змінні названі однаково, та містять одне і теж значення, але в першому випадку до змінної “myName” був присвоєний тип “string” тому при наведені курсором на цю змінну можна побачити який це тип і які дані містить змінна.

Безперечно використовувати типізацію у малих додатках немає сенсу, так як це тільки збільшить вміст коду та ускладнить його читання, але якщо мова йде про розробку великих та складних проектів, тоді TypeScript виконує свою важливу та корисну роль у цьому проекті, він мінімізує кількість помилок та допоможе зрозуміти де і по якій причині відбулась та чи інша помилка.

TypeScript був представлений компанією Microsoft в 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript. Розробником цієї мови є Андерс Хейлсберг.

За допомогою TypeScript можна створювати складні типи, поєднуючи прості. Є два популярних способи зробити це: за допомогою союзів і за допомогою дженериків. За допомогою об’єднання можна оголосити, що тип може бути одним із багатьох типів. Наприклад, можна описати логічний тип як істинний або хибний (Див. рисунок 1.8).

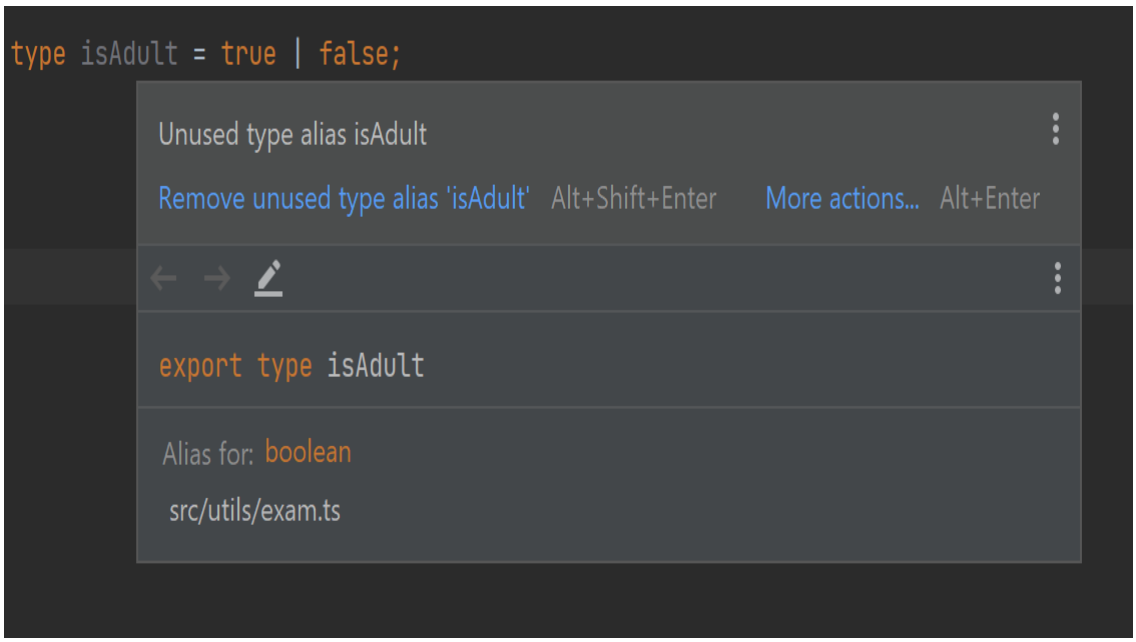

Рисунок 1.8 – Приклад об’єднання двох типів

Також існує популярний варіант використання об’єднаних типів, це опис набору рядкових або числових літералів. (Див. рисунок 1.9).

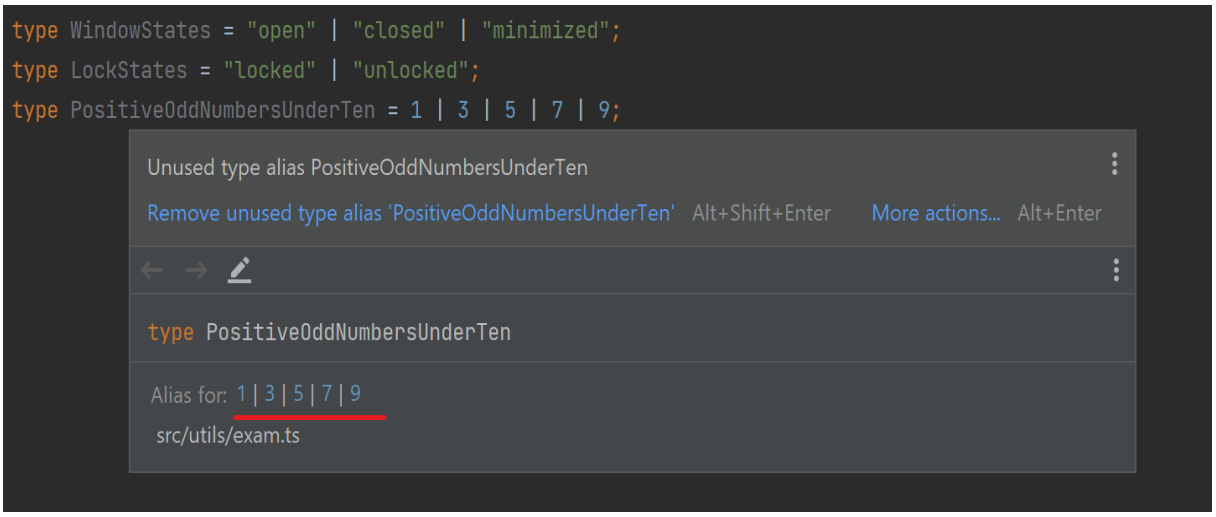

Рисунок 1.9 – Приклад об’єднання декількох типів

Приклад якого наведено вище.

1.4 Сучасна бібліотека React JS

Для розробки проекту було обрано бібліотеку React. Я завжди використовую цю бібліотеку для написання локальних веб-сторінок, тому що цей продукт має велику та багату документацію, і рахується одним із найпопулярніших фреймворків JavaScript.

Перший вихід бібліотеки на світ відбувся у 2013 році, її заснувала корпорація Meta Platforms, а саму бібліотеку створено Джорданом Волком.

Фреймворк був представлений як проект з відкритим початковим кодом на конференції розробників JSConf US, що проходила у Сполучених Штатах у травні 2013 року. На конференції React.js Conf, влаштовану Фейсбуком у березні 2015-го, проект було представлено як відкрите програмне забезпечення.

Особливості React полягають в тому, що властивості передаються в рендер компоненту, як властивості html тегу. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через функцію зворотнього виклику.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера.

Існує два види компонентів в реакті, це класові та функціональні, нижче буде наведено приклад як виглядає розгорнутий класовий компонент (Див. рисунок 1.10).

```
export class Component extends React.Component {
    render() {
        return (
            <div>

            </div>
        )
    }
}
```

Рисунок 1.10 – Приклад класового компонента

На сьогодні класовий підхід вже рахується застарілим методом написання проектів на реакті, тому більшість компаній та розробників пишуть свій код виключно на функціональних компонентах (Див. рисунок 1.11).

```
export const Component = () => {
    return (
        <div>

        </div>
    );
};
```

Рисунок 1.11 – Приклад функціонального компонента

Головна відмінність класового компонента від функціонального це можливість зберігати внутрішній стан. Класовий компонент повинен містити в собі метод render(), який повертає React-елемент.

Також у класових компонентах доступ до пропсів відбувається через ключове слово “this”.

Функціональні компоненти стали переважати над класовими з появленням хуків, вони стали більш лаконічними та дозволяли писати чистіший код, також у них немає свого зміненого стану, тому їх стало легше тестувати і повторно використовувати. Також варто згадати про JSX, це розширення javascript яке допомагає описувати html-подібні елементи за допомогою коду реакту, він нагадує мову шаблонів наділений силою javascript. Трансформація зазвичай включає в себе заміну коду всередині тегів на виклик функції “createElement”, якій в якості аргументів передається назва тега, властивості, та дочірні елементи, це розширення нагадує звичайну мову розмітки, але якщо в HTML це звичайний рядок, то в JSX це тільки об’єкт (Див. рисунок 1.12).

```jsx
export const Component = () => {
    return (
        <div>
            <h1>Hello World</h1>
            <button>Click</button>
            <ul>
                <li>1</li>
                <li>2</li>
                <li>3</li>
            </ul>
        </div>
    );
};
```

Рисунок 1.12 – Зразок JSX коду

Розширення JSX трансформується в джаву скрипт за допомогою таких елементів як babel.

1.5 Бібліотека для керування станом глобальних даних Redux

Redux — це інструмент для керування станом даних та інтерфейсом користувача у веб-додатках з великою кількістю сутностей. Назва читається як “Redux”, та складається з двох слів: reduce та flux.

Якщо якась частина програми змінилась, програміст про це не дізнається, а значить не зможе відреагувати, наприклад перемалювавши потрібну частину екрана. Redux вирішує цю проблему. Зміна даних всередині сховища породжує події, на які можна підписуватись і виконувати довільну логіку.

Redux дозволяє мені керувати станом моїх веб-застосунків, створених на будь-якому JavaScript фреймворку, наприклад, React, або Angular. Редакс виник з ідеї Flux – архітектурного патерну, створеного інженерами з Facebook.

Оскільки сховище Redux створюється заново при оновленні сторінки, дані цього типу повинні зберігатися десь ще: в базі даних на сервері або в

локальному сховищі в браузері. Це можуть бути любі дані, наприклад налаштування користувача. У своєму проекті веб-календаря я використовую редакс для зберігання та керування глобальними даними, саме махінації із створенням нотаток, та авторизацією (Див. рисунок 1.13).

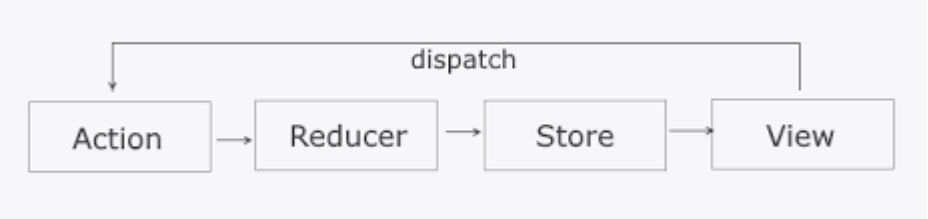

Рисунок 1.13 – Механізм роботи Redux

Компонент генерує дію (action), диспатчер повідомляє про це редюсеру, в свою чергу редюсер це проста функція яка розуміє action і змінює стан. Далі я отримую змінені дані які передаються в компонент.

Reducer це функція яка приймає два параметра: state, action. Нижче наведено зразок як виглядає функція редюсер (Див. рисунок 1.14).

```
function Reducer(state, action){
    switch (action.type) {
        case SOME_ACTION:
            return {...state, ...updated data}
        default:
            return state
    }
}
```

Рисунок 1.14 – Функція Reducer

В параметрах видно що є стейт і екшин, в залежності від типу екшина редюсер буде вносити ті чи інші зміни у стейті, на рисунку 1.14 видно конструкцію switch-case де обробляється кожен тип екшина.

Якщо ні один кейс не відпрацював, то редюсер поверне стандартний стейт без ніяких змін. Нижче зразок алгоритм дій у редаксі (Див. рисунок 1.15).

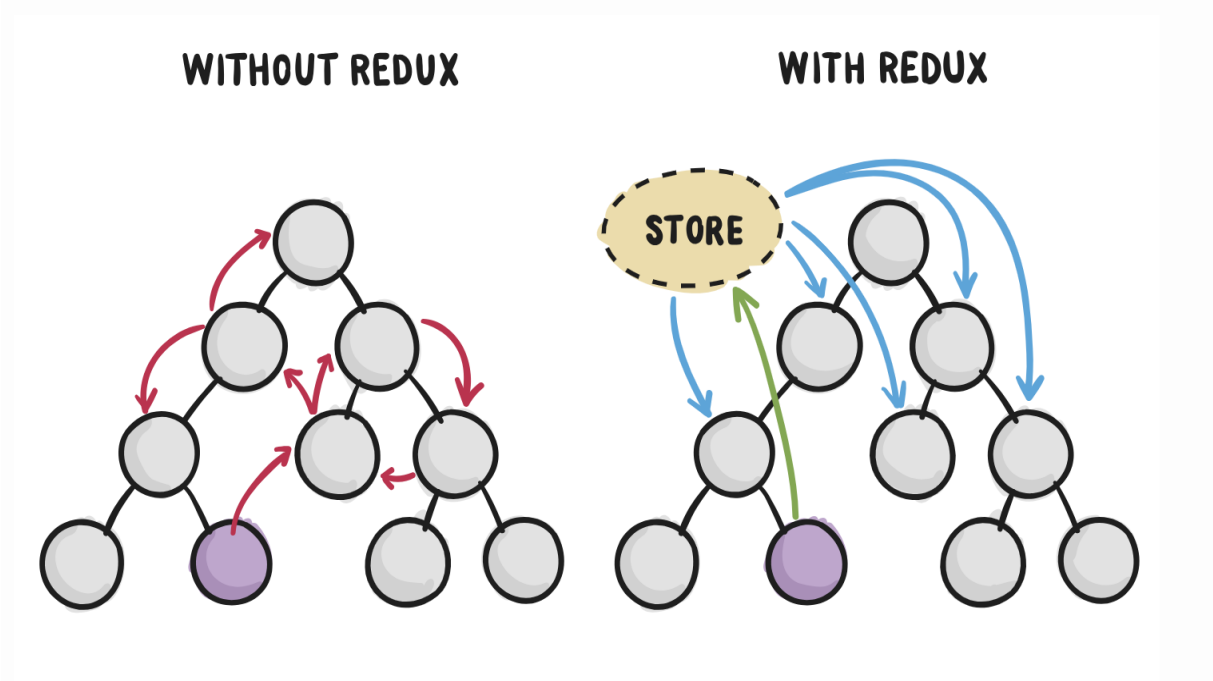

Рисунок 1.15 – Принцип роботи Redux

На рисунку 1.15 зліва приклад передачі даних між компонентами. Як можна замітити один компонент містить в собі якісь дані які передає в інші компоненти, і так відбувається по черзі з одного компонента в інший, аж поки дані не дійдуть до кінцевого компонента в які передаються. Так виглядає передача даних без використання Redux.

З права видно принцип за яким передаються дані за допомогою Редакс, фіолетовий кружечок це компонент який хоче поділитись своїми даними з іншими компонентами, він не буде кожному компоненту передавати по черзі якісь дані, а просто передасть їх в глобальний кошик store, і там вже будуть зберігатись дані які кожен компонент зможе використовувати в любому місці де йому це потрібно, при умові що цей компонент обгорнутий в цей стор, нижче наведено зразок як вигладяє огортання проекту в стор (Див. рисунок 1.16).

```
root.render(
    <Provider store={store}>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </Provider>
```

Рисунок 1.16 – Огортання головного компоненту в Store

В компонент “Provider” передається об’єкт в якому будуть міститись усі дані які будуть туди передаватись, і все що знаходиться у провайдері буде мати доступ до цих даних.

1.6 Висновок до першого розділу

В першому розділі кваліфікаційної роботи розглянуто та проаналізовано середовище написання коду в якому розроблялась веб-сторінка, висвітлені плюси використання даного середовища. Проведено порівняння технологій та аргументовано потребу їх використання, розглянуто, описано, та вибрано сучасні бібліотеки які найефективніші в розробці сучасних реакт-додатків. Продемонстровано та проаналізовано структури кореневої парки та файлову структуру проекту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ВЕБ-КАЛЕНДАРЯ

2.1 Створення маршрутизації додатку

Розробка проекту почалась зі створення маршрутизації, її було поділено на дві частини:

– Приватні маршрути.

– Публічні маршрути.

Такий метод використовується для того, щоб неавторизований користувач не зміг перейти до сторінки з календарем (Див. рисунок 2.1).

```
export interface IRoute {
    path: string
    element: React.ComponentType
}

export enum RouteNames {
    LOGIN = '/login',
    EVENT = '/'
}

export const publicRoutes: IRoute[] = [
    {path: RouteNames.LOGIN, element: Login }
]

export const privateRoutes: IRoute[] = [
    {path: RouteNames.EVENT, element: Event }
]
```

Рисунок 2.1 – Структура маршрутів

У масиві публічних маршрутів знаходиться один маршрут який веде на сторінку авторизації. У масиві приватних маршрутів суть та сама, окрім того, що приватний маршрут веде на сторінку календаря.

Для початку надано тип для цих маршрутів за допомогою тайп скрипта. Далі занесено у змінні назви посилань на сторінки, щоб застерегти себе від потенційних помилок, при повторному використанні цих маршрутів. Для роботи із маршрутизацією використано бібліотеку react-router-dom – це бібліотека яка встановлюється і підключається до реакт проекту, та вирішує маршрутизацію та переключення між сторінками.

Для того, щоб роутер працював необхідно імпортувати “BrowserRouter” з бібліотеки “react-router-dom” та огорнути в нього кореневий компонент (Див. рисунок 2.2).

```
import { BrowserRouter } from "react-router-dom";
```

Рисунок 2.2 – Встановлення BrowserRouter

На рисунку нижче буде наведено приклад як можна огорнути кореневий компонент та під’єднати усі сторінки що в ньому знаходяться до реакт-роутера (Див. рисунок 2.3).

```
<BrowserRouter>
    <App />
</BrowserRouter>
```

Рисунок 2.3 – Підключення роутера до кореневого компоненту

Усі компоненти які знаходяться в кореневому файлі будуть під’єднані до роутера.

2.2 Керування глобальними даними

За допомогою бібліотеки редакс проводилось керувавання глобальними даними. Створивши папку для роботи з редаксом, відбулось його налаштовування, спочатку була проведена ініціалізація об’єкта store за допомогою функції “createStore” яка використовується у редаксі (Див. рисунок 2.4).

Рисунок 2.4 – Ініціалізація store

При ініціалізації змінної функцією createStore, в неї передаються два параметри, редюсер та функція-посередник.

Оскільки у проекті задіяно більше одного редюсера, я заніс усі функції у кореневий редюсер, назвавши його “rootReducer” та передав його першим параметром, другим параметров передав функцію-посередник яка буде використовуватись при роботі з асинхронними функціями (Див. рисунок 2.5).

Рисунок 2.5 – Експорт усіх редюсерів одним об’єктом

Усі редюсери будуть експортовані одним об’єктом для зручності роботи.

2.3 Реалізація посторінкової навігації

Я використав дві кнопки у блоці header для входу і виходу (Див. рисунок 2.6).

Рисунок 2.6 – Блок header неавторизованого користувача

Він розроблений простим чином, якщо користувач на сторінці авторизації то йому видно лише кнопку входу, якщо користувач авторизується у додаток, йому буде видно логін під яким він авторизований та кнопку виходу (Див. рисунок 2.7).

Рисунок 2.7 – Блок header авторизованого користувача

Для посторінкової навігації я використовую хук “useNavigate” який надає реакт, Він дозволяє швидко виконати ініціалізацію змінної (Див. рисунок 2.8).

```
const router = useNavigate()
```

Рисунок 2.8 – Виклик функції useNavigate для навігації по сторінках

Далі за допомогою змінної router я можу виконувати перехід з одної сторінки на іншу викликом однієї функції (Див. рисунок 2.9).

```
<Menu theme='dark' mode='horizontal' selectable={false}>
    <Menu.Item key={1}
               onClick={() => router(RouteNames.LOGIN)}>
        Вхід
    </Menu.Item>
</Menu>
```

Рисунок 2.9 – Виклик router для переключення на авторизацію

Після того як навігація готова, можна приступати до виконання логіки авторизації та перевірки на то, чи авторизований користувач, і в залежності від того авторизований він чи ні, перекидати його на ту чи іншу сторінку.

2.4 Функціонал авторизації у додаток

Дані про те, авторизований користувач чи ні, я зберігаю у редюсері, який приймає у себе стейт та екшин, нижче наведено стандарні дані стейту відносно авторизації користувача, та самого користувача (Див. рисунок 2.10).

```
const initialState: AuthState = {
    isAuth: false,
    error: '',
    isLoading: false,
    user: {} as IUser
}
```

Рисунок 2.10 – Стандартний стейт редюсера неавторизованого користувача

Поки людина не авторизується у додаток, значення isAuth буде хибним, помилка буде пустим рядком та загрузка буде відсутня, адже ніякі дії поки що не відбулись (Див. рисунок 2.11).

```
export default function authReducer(state : AuthState = initialState, action: AuthAction): AuthState {
    switch (action.type) {
        case AuthActionEnum.SET_AUTH:
            return {...state, isAuth: action.payload, isLoading: false}

        case AuthActionEnum.SET_USER:
            return {...state, user: action.payload}

        case AuthActionEnum.SET_ERROR:
            return {...state, error: action.payload, isLoading: false}

        case AuthActionEnum.SET_IS_LOADING:
            return {...state, isLoading: action.payload}
        default:
            return state
    }
}
```

Рисунок 2.11 – Функція AuthReducer

На рисунку вище показана функція яка приймає стейт та екшин і в залежності від типу екшинів буде виконувати той чи інший кейс.

У першому кейсі викликається екшин “Set Auth”, який повертає попередній стан та змінює тільки ті дані які повинні змінитись під час авторизації.

У другому кейсі логіка та сама як і в наступних, екшини всюди різні, та в залежності від їх типів будуть вноситись потрібні зміни (Див. рисунок 2.12).

```
export interface AuthState {
    isAuth: boolean
    user: IUser
    isLoading: boolean
    error: string
}

export enum AuthActionEnum {
    SET_AUTH = 'SET_AUTH',
    SET_ERROR = 'SET_ERROR',
    SET_USER = 'SET_USER',
    SET_IS_LOADING = 'SET_IS_LOADING'
}
```

Рисунок 2.12 – Типізація стандартного state та перечислення усіх actions

Їх є всього чотири, перший викликається при зміні стану користувача на авторизований або ні, другий при появлені помилки або її очищенню, третій міняє дані про користувача який на даний момент авторизувався, та четвертий який відповідає за загрузку веб-сторінки (Див. рисунок 2.13).

```
export interface SetAuthAction {
    type: AuthActionEnum.SET_AUTH ;
    payload: boolean
}
export interface SetErrorAction {
    type: AuthActionEnum.SET_ERROR ;
    payload: string
}
export interface SetUserAction {
    type: AuthActionEnum.SET_USER ;
    payload: IUser
}
export interface SetIsLoadingAction {
    type: AuthActionEnum.SET_IS_LOADING ;
    payload: boolean
}
```

Рисунок 2.13 – Код чотирьох екшинів

Перевірка на авторизацію відбувається у компоненті AppRouter, який за допомогою метода “map” пробігається по двом масивам (Див. рисунок 2.14).

```
const AppRouter: FC = () => {
    const {isAuth} = useTypedSelector( selector: state => state.auth)
    return (
        isAuth ?
            <Routes>
                {privateRoutes.map(route =>
                    <Route path={route.path} element={<route.element />} key={route.path} />
                )}
                <Route path='/*' element={<Navigate replace to={RouteNames.EVENT} />} />
            </Routes>
            :
            <Routes>
                {publicRoutes.map(route =>
                    <Route path={route.path} element={<route.element />} key={route.path} />
                    )}
                <Route path='/*' element={<Navigate replace to={RouteNames.LOGIN} />} />
            </Routes>
    );
};
```

Рисунок 2.14 – Код перевірки авторизації

Змінна isAuth містить в собі тільки логічне значення true або false, і в залежності від того, яке там значення, буде відображатись приватний або публічний компонент, перевірка яких відбувається в компоненті Routes.

Саму змінну isAuth я витягую з стейту авторизації, де редюсер повертає змінені дані в залежності від самих змін. За допомогою типізованого хука useTypedSelector в проекті можна дістатись до даних (Див. рисунок 2.15).

```
export const useTypedSelector: TypedUseSelectorHook<RootState> = useSelector
```

Рисунок 2.15 – Код хука useTypedSelector

Авторизація відбувається наступним чином, у папці Public є мокові дані з користувачами, яких я створив вручну, їх можна додавати скільки завгодно, на

момент розробки даного проекту я не маю достатнього досвіду для створення бази даних з повним функціоналом та сервером, тому я використав такий підхід. (Див. рисунок 2.16).

Рисунок 2.16 – Масив користувачів

Існує три умовних користувача, на рисунку вище показані їх логіни та паролі, цей масив записаний у файлі users.json який знаходиться в папці public (Див. рисунок 2.17).

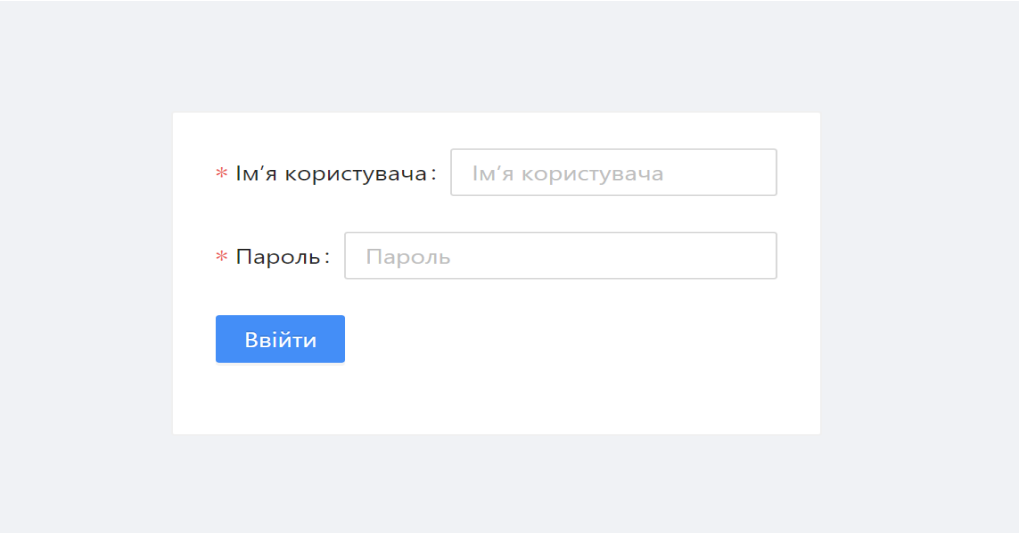

Рисунок 2.17 – Форма авторизації

Щоб увійти у календар, потрібно ввести логін одного з трьох користувачів та пароль. На рисунку нижче продемонстровано код форми логіна (Див. рисунок 2.18).

```
<Form onFinish={submit}>
    {error && <div style={{color: 'red'}}>{error}</div>}
    <Form.Item
        label="Ім'я користувача"
        name="username"
        rules={[rules.required('Введіть ім'я користувача')]}
    >
        <Input placeholder='Ім'я користувача' value={username} onChange={e => setUsername(e.target.value)} />
    </Form.Item>

    <Form.Item
        label="Пароль"
        name="password"
        rules={[rules.required('Введіть пароль')]}
    >
        <Input type='password' placeholder='Пароль' value={password} onChange={e => setPassword(e.target.value)} />
    </Form.Item>

    <Form.Item>
        <Button type="primary" htmlType="submit" loading={isLoading}>
            Ввійти
        </Button>
    </Form.Item>
</Form>
```

Рисунок 2.18 – Код блоку форми

Під час авторизації, якщо логін та пароль вірні кнопка “Ввійти” стає відключеною та на ній вимальовується значок завантаження який буде крутитись то тих пір поки не відбудеться вхід (Див. рисунок 2.19).

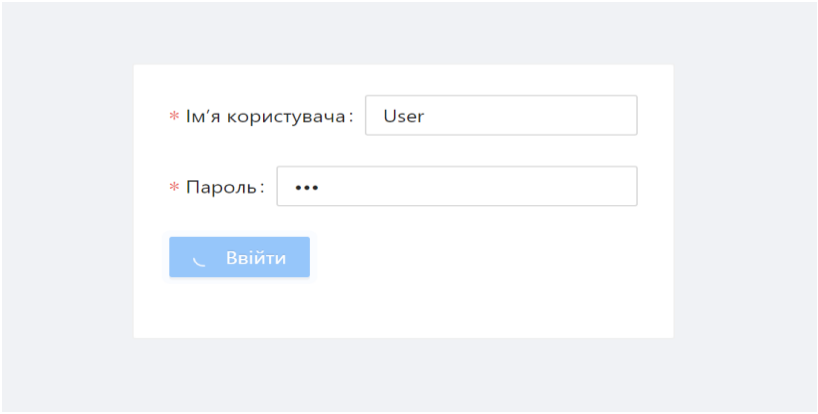

Рисунок 2.19 – Загрузка під час авторизації

Також, щоб поля вводу логіна та пароля були пов’язані, їх потрібно приєднувати до хуків useState, щоб даними які будуть внесені в ці поля, можна було керувати та проводити перевірки (Див. рисунок 2.20).

```
const [username, setUsername] = useState( initialState: '')
const [password, setPassword] = useState( initialState: '')
```

Рисунок 2.20 – Код стандартного стану логіна та пароля

Наступним чином після того як користувач вписав логін та пароль, відпрацьовує функція для перевірки даних (Див. рисунок 2.21).

```
login: (username: string, password: string) => async (dispatch: AppDispatch) => {
    try {
        dispatch(AuthActionCreators.setIsLoading(true))
        setTimeout( callback: async () => {
            const response = await UserService.getUsers()
            const mockUser = response.data.find(user => user.username === username && user.password === password)
            if(mockUser) {
                localStorage.setItem('auth', 'true')
                localStorage.setItem('username', mockUser.username)
                dispatch(AuthActionCreators.setUser(mockUser))
                dispatch(AuthActionCreators.setIsAuth(true))
            } else {
                dispatch(AuthActionCreators.setError('Невірний логін або пароль'))
            }
            dispatch(AuthActionCreators.setIsLoading(false))
        }, ms: 1000)
    } catch (e) {
        dispatch(AuthActionCreators.setError('Error'))
    }
},
```

Рисунок 2.21 – Функція авторизації

Першим ділом в диспатчер заноситься екшин для зміни значення загрузки, за ним спрацьовує асинхронна функція яка робить запис на масив з

користувачами, та порівнює ті дані які є, з тими які вписані у формі авторизації (Див. рисунок 2.22)

```
import axios, { AxiosPromise } from "axios";
import { IUser } from "../models/IUser";

export default class UserService {
    static async getUsers(): Promise<AxiosPromise<IUser[]>> {
        return axios.get<IUser[]>( url: './users.json')
    }
}
```

Рисунок 2.22 – Функція для запиту на файл з користувачами

У змінну mockUser зберігаються дані користувача, та якщо він є, у локальному сховищі я створюю ключ “auth” зі значенням “true”, також дістаю логін користувача і тим самим способом створюю у локальному сховищі ключ “username” і надаю йому ім’я користувача який був у логіні (Див. рисунок 2.23).

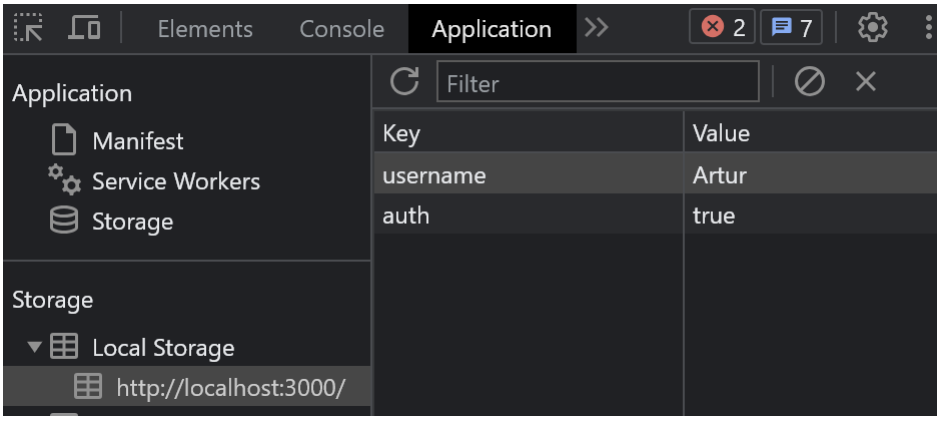

Рисунок 2.23 – Збереження даних у Local Storage

Під кінець в дані користувача підставляються ті дані які були внесені при авторизації та були перевірені, та в dispatch я прокидаю AuthActionCreator і міняю значення на true.

2.4.1 Валідація входу

Після кліку на кнопку “Ввійти” спрацює перевірка на те, чи є щось у цих полях, якщо імені користувача або пароля не було прописане у полі, тоді помилка повідомить що саме було пропущене (Див. рисунок 2.24).

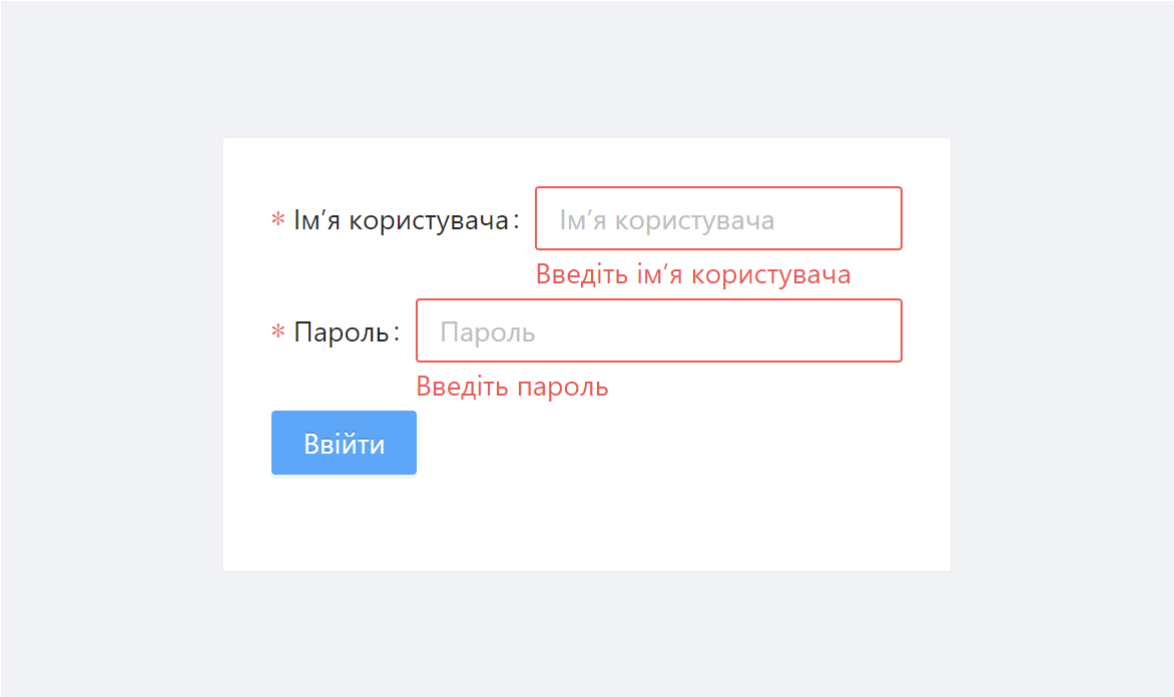

Рисунок 2.24 – Перевірка на заповнення полів авторизації

Для того, щоб відбулась перевірка правильності написання логіна та пароля, я передаю в поле вводу логіна та пароля правила, за якими вони повинні працювати, у випадок якщо трапиться суперечка в полі вводу для логіна, вискочить помилка тільки для нього, і аналогічно те ж саме з паролем.

В пропс “rules” я передаю об’єкт який в свою чергу викликає функцію “required”, та вже в саму функцію я передаю текст з повідомленням про помилку. На рисунку нижче буде продемонстровано як це описується в коді. (Див. рисунок 2.25)

```
<Form.Item
    label="Ім’я користувача"
    name="username"
    rules={[rules.required('Введіть ім’я користувача')]}
>
    <Input placeholder='Ім’я користувача' value={username} onChange={e => setUsername(e.target.value)} />
</Form.Item>

<Form.Item
    label="Пароль"
    name="password"
    rules={[rules.required('Введіть пароль')]}
>
    <Input type='password' placeholder='Пароль' value={password} onChange={e => setPassword(e.target.value)} />
```

Рисунок 2.25 – Виклик функції required

Функція “required”, сама в собі містить стандартне повідомлення яке вона поверне у випадку виклику цієї функції у конкретній частині коду, але вона поверне своє стандартне повідомлення тільки в тому випадку якщо при виклику функції в неї не передати повідомлення про конкретну помилку, наприклад як з повідомленням про поле логіна чи пароля. Нижче буде показано функцію з стандартним повідомленням (Див. рисунок 2.26).

```
required: (message: string = 'Обовязкове поле') => ({
    required: true, message
}),
```

Рисунок 2.26 – Функція required з стандартним повідомленням

Як можна замітити, ця функція приймає один параметр стрічкового типу, і до цього параметру одразу присвоєно значення “Обов’язкове поле”. Така махінація потрібна для того, якщо я забуду для поля логіна прописати текст з помилкою, буде висвітлено текст з стандартним повідомленням.

Після перевірки на заповнення полів буде відпрацьовувати перевірка на сам логін та пароль. Натискаючи на кнопку “Ввійти” буде викликатись функція

“submit” у неї я передаю функцію “login” в яку передаю дані логіна і пароля які вписує користувач (Див. рисунок 2.27).

```
const [username, setUsername] = useState( initialState: '')
const [password, setPassword] = useState( initialState: '')

const submit = () => {
    login(username, password)
}
```

Рисунок 2.27 – Виклик функції авторизації

Після того як функція буде викликана і відбудеться перевірка, користувача або перекине на сторінку календаря якщо логін та пароль були вірними, або виникне помилка з повідомленням “Невірний логін або пароль”, якщо користувач введе невірні дані (Див. рисунок 2.28).

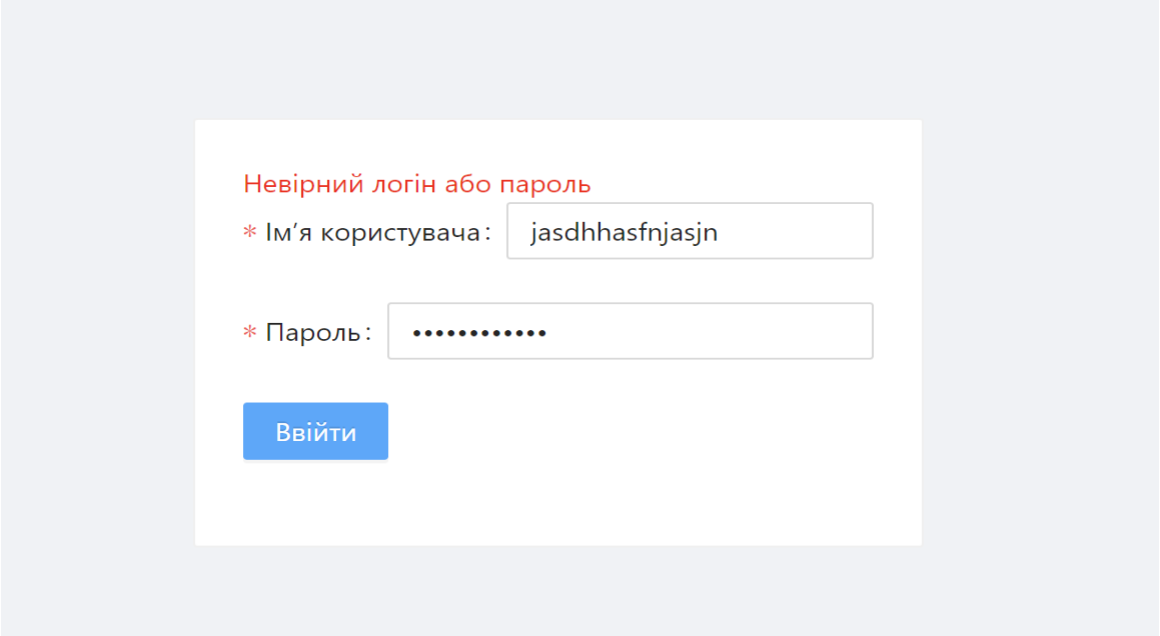

Рисунок 2.28 – Перевірка на вірність логіна і пароля

Наступним кроком після авторизації відкривається сторінка з календарем.

2.5 Реалізація інтерфейсу та логіки календаря

Після успішної авторизації у додаток, відкривається друга сторінка, це сторінка з календарем (Див. Рисунок 2.29).

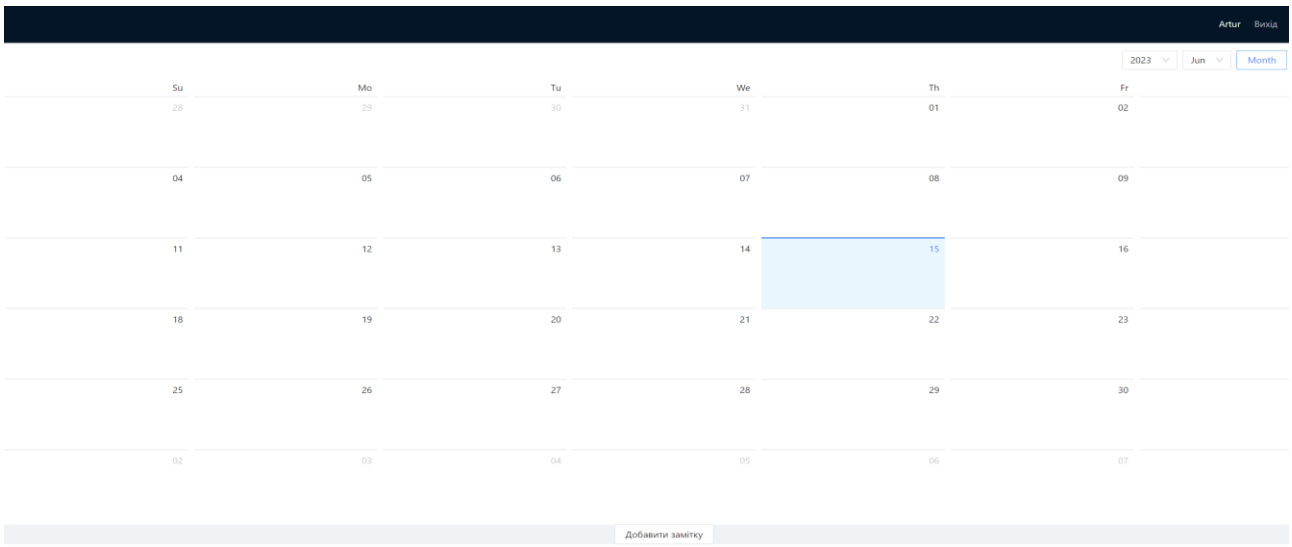

Рисунок 2.29 – Сторінка календаря

В самому верху є навігаційна панель, з кнопкою щоб вийти, та іменем під яким авторизований користувач. Далі на всю сторінку розміщений календар, його я взяв з бібліотеки “Antd design”, це гнучка бібліотека компонентів, розроблена для допомоги розробникам при створенні різних інтерфейсів.

На рисунку нижче зображено вихідний код календаря (Див. рисунок 2.30).

```
return (
    <Layout>
        <EventCalendar events={events} />
        <Row justify='center'>
            <Button onClick={() => setModalVisible( value: true)}>Добавити замітку</Button>
        </Row>
        <Modal
            title='Добавити замітку'
            visible={modalVisible}
            footer={null}
            onCancel={() => setModalVisible( value: false)}
        >
            <EventForm guests={guests} submit={addNewEvent} />
        </Modal>
    </Layout>
);
```

Рисунок 2.30 – Блок коду календаря

В правому верхньому кутку календаря є колонки, зі списком вибору, місяця, чи року (Див. рисунок 2.31).

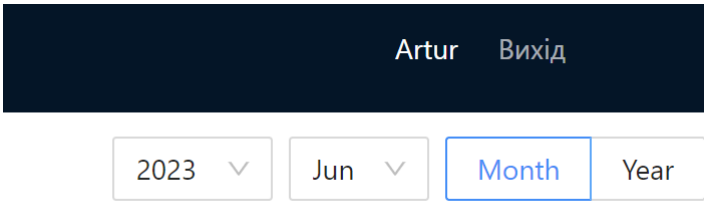

Рисунок 2.31 – Колонки з вибором дати

Календар автоматично налаштований так, що буде підсвічувати день синім кольором, та сам заповнює місяць та рік у колонках зверху. Можна натиснути на праву крайню кнопку “Year” і календар покаже список усіх місяців, щоб повернутись на список днів потрібно натиснути назад на кнопку “Month”, яка розташована лівіше. (Див. рисунок 2.32).

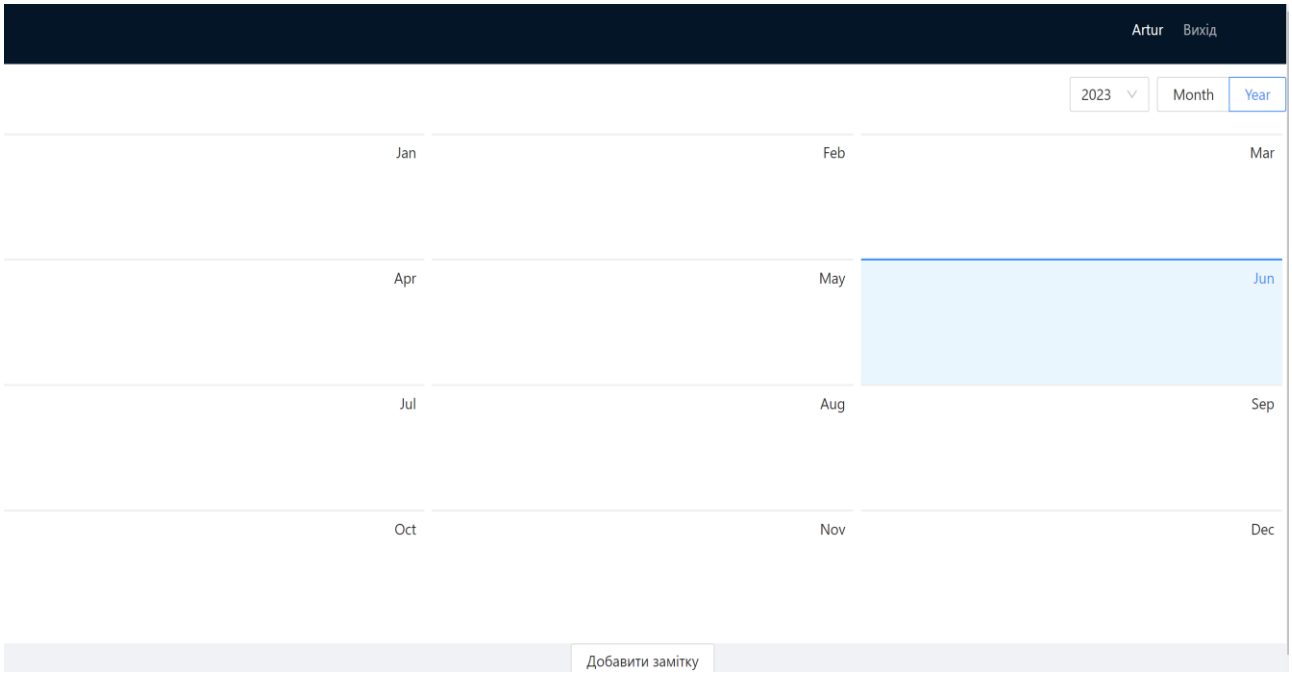

Рисунок 2.32 – Сторінка з місяцями

Місяць так само як і день, автоматично підсвічується при відкриті сторінки, тобто якщо сьогодні червень, то буде активним блок червня.

2.5.1 Створення функціоналу для додавання нотаток

Внизу календаря є кнопка “Добавити замітку”, натискаючи на неї буде випливати модальне вікно з формою для створення нотатки. Щоб створити нотатку потрібно буде заповнити наступні дані:

– Текст нотатки.
– Дата, на який день буде записана нотатка.
– Гість, який буде запрошений до цієї нотатки.

У розділі з вибором дати події, буде можливість вибрати дату за допомогою маленького влаштованого календаря, замість того, щоб писати дату вручну, це допомагає уникнути помилки при виборі дня чи місяця. Заповнивши всі поля, можна буде створити нотатку натиснувши на кнопку “Створити”, і замітка буде автоматично додана до блоку вибраного дня, який знаходиться на сторінці календаря.

На рисунку нижче зображено як виглядає модальне вікно, з формою для заповнення та створення нотатки (Див. рисунок 2.33).

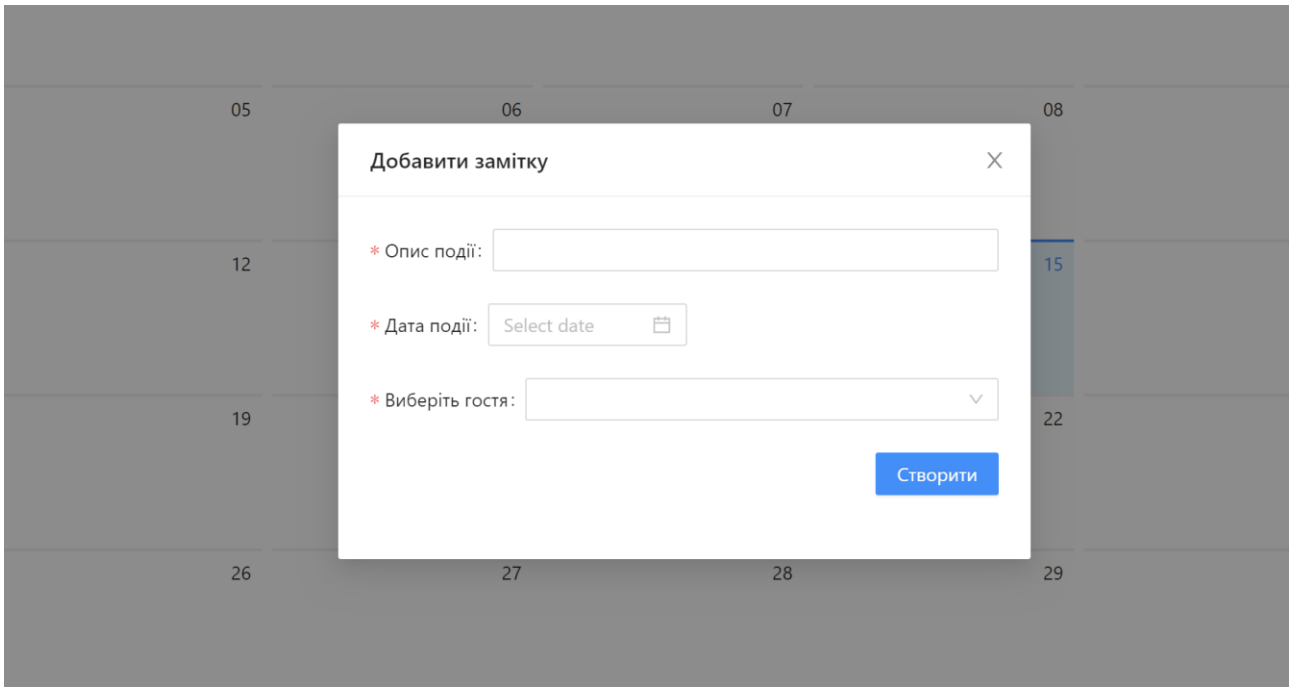

Рисунок 2.33 – Форма створення нотатки

Запрошення гостя до нотатки відбувається таким чином, що умовний користувач може створювати нотатку яку буде бачити не тільки він, а і той кого він добавить в якості гостя, список гостів береться звідти, де є вручну створені користувачі у файлі users про який я писав раніше. Дані про створену нотатку, передаються в функцію створення нотатки, після цього функція приймає дані в параметрах та заносить їх у локальне сховище.

Дані про нотатку передаються як об’єкт JSON формату, тому перед збереженням їх у список всіх нотаток, я викликаю функцію json.parse, після того як цей об’єкт перетворився на звичайний javascript об’єкт я добавляю його до масиву методом push, та передаю у диспатч. Для того щоб цю нотатку зберегти у локальному сховищі, адже якщо її там не зберегти, вона пропаде після першого оновлення сторінки. Отже для того щоб її зберегти, до розпарсеного джава скрипт об’єкту викликається функція “stringify()” яка поверне назад цей об’єкт у json формат.

Першим параметром надано назву, другим параметром нотатку. На рисунку нижче буде продемонстрована функція, яка відповідає за обробку та збереження нотатки у локальне сховище (Див. рисунок 2.34).

```
createEvent: (event: IEvent) => async (dispatch: AppDispatch) => {
    try {
        const events = localStorage.getItem( key: 'events') || '[]'
        const json = JSON.parse(events) as IEvent[]
        json.push(event)
        dispatch(EventActionCreators.setEvents(json))
        localStorage.setItem('events', JSON.stringify(json))
    } catch (e) {
        console.log(e)
    }
},
```

Рисунок 2.34 – Функція створення нотатки

У асинхронних функціях я завжди використовую метод відловлювання помилок “try-catch”, щоб в разі чогось, зразу можна було знайти місце та причину помилки.

У редюсері для створення нотаток знаходиться тільки два кейси, один відповідає за створення нотатки, другий за запрошення гостя. На початку це два масиви які по стандарту пусті (Див. рисунок 2.35).

```
const initialState: EventState = {
    events: [],
    guests: []
}

export default function EventReducer(state : EventState  = initialState, action: EventAction): EventState {
    switch (action.type) {
        case EventActionEnum.SET_GUESTS:
            return {...state, guests: action.payload}
        case EventActionEnum.SET_EVENTS:
            return {...state, events: action.payload}
        default:
            return state
    }
}
```

Рисунок 2.35 – Функція EventReducer

Після написання функцій для створення нотатки та написання інтерфейсу, можна перевіряти чи зберігаються дані у локальне сховище, та робити перевірку на те, чи є доступ у гостя до замітки, в яку він був доданий. Це відбувається наступним чином, умовно я авторизувався під іменем Admin та створюю якусь замітку, вибираю день та місяць, запрошую до цієї замітки іншого користувача під іменем Artur.

Отже коли гість авторизується під своїм іменем, то він буде бачити вже готову нотатку яку створив якийсь інший користувач, та запросив його до неї. Інші користувачі які не були запрошені до будь-яких нотаток, не будуть бачити ніяких сповіщень.

Дані зберігаються у локальному сховищі та не пропадуть якщо буде перезавантажено веб-сторінку, та будуть збережені після виходу і повторної авторизації.

Нижче буде зображений рисунок на якому буде створення нотатки від імені Admin та запрошення до цієї події користувача під іменем Artur (Див. рисунок 2.36).

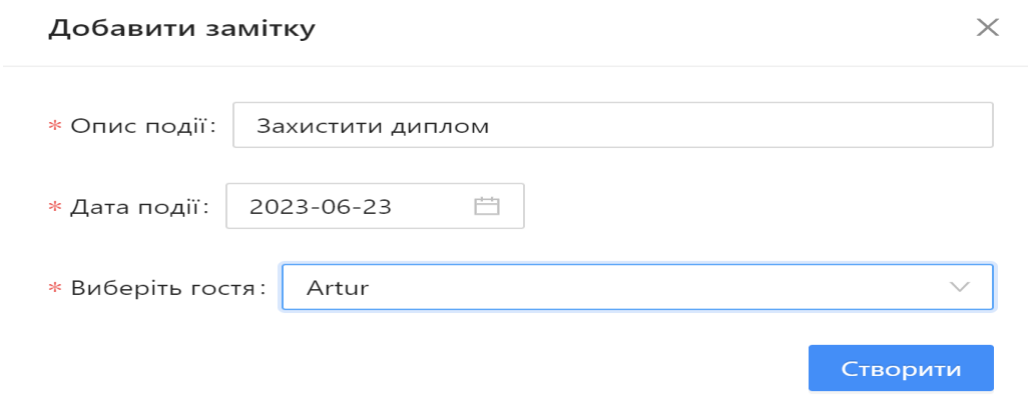

Рисунок 2.36 – Створення нотатки та запрошення гостя

Після підтвердження, натиснувши на кнопку створення, замітка буде додана до календаря, та буде доступна для користувача Artur (Див. рисунок 2.37).

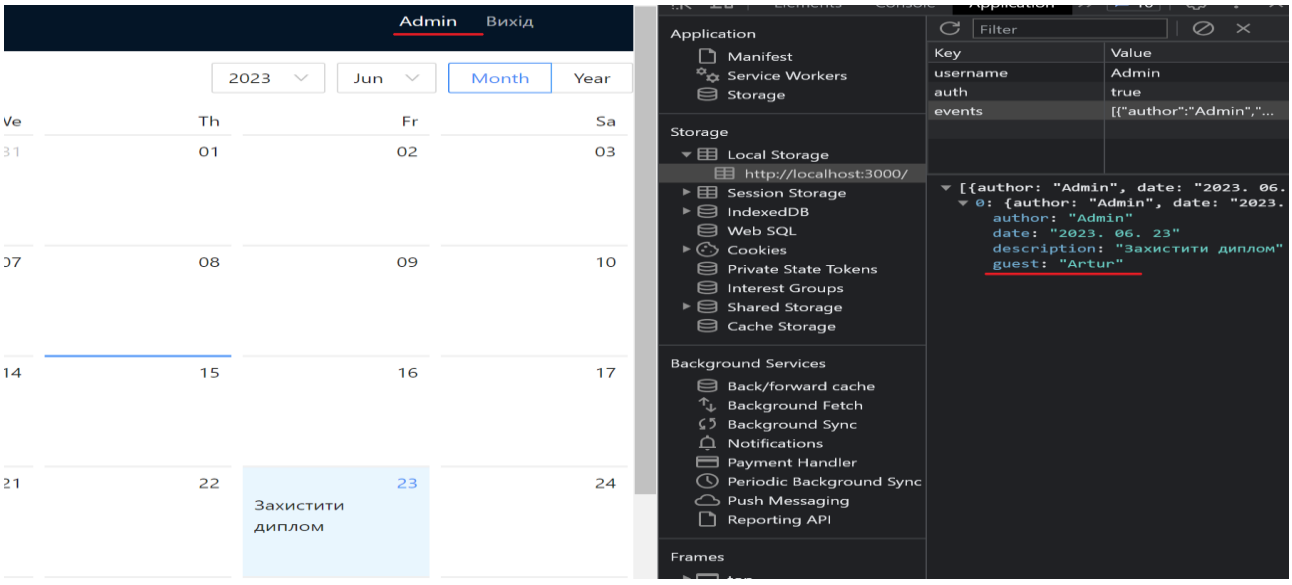

Рисунок 2.37 – Вигляд створеної нотатки

Як можна замітити, усе ідеально працює, нотатка створилась на той день який був вибраний, також у локальному сховищі можна побачити, повний об’єкт який зберігається після створення нотатки, вона була створена автором під іменем Admin, нижче видно дату на яку створювалась замітка, опис самої замітки, та у полі “guest” користувач який був запрошений.

Щоб перевірити, чи користувачу Artur видно цю замітку, я виконую авторизацію під його іменем, та одразу виконаю перевірку чи змінився ключ авторизації у локальному сховищі (Див. рисунок 2.38).

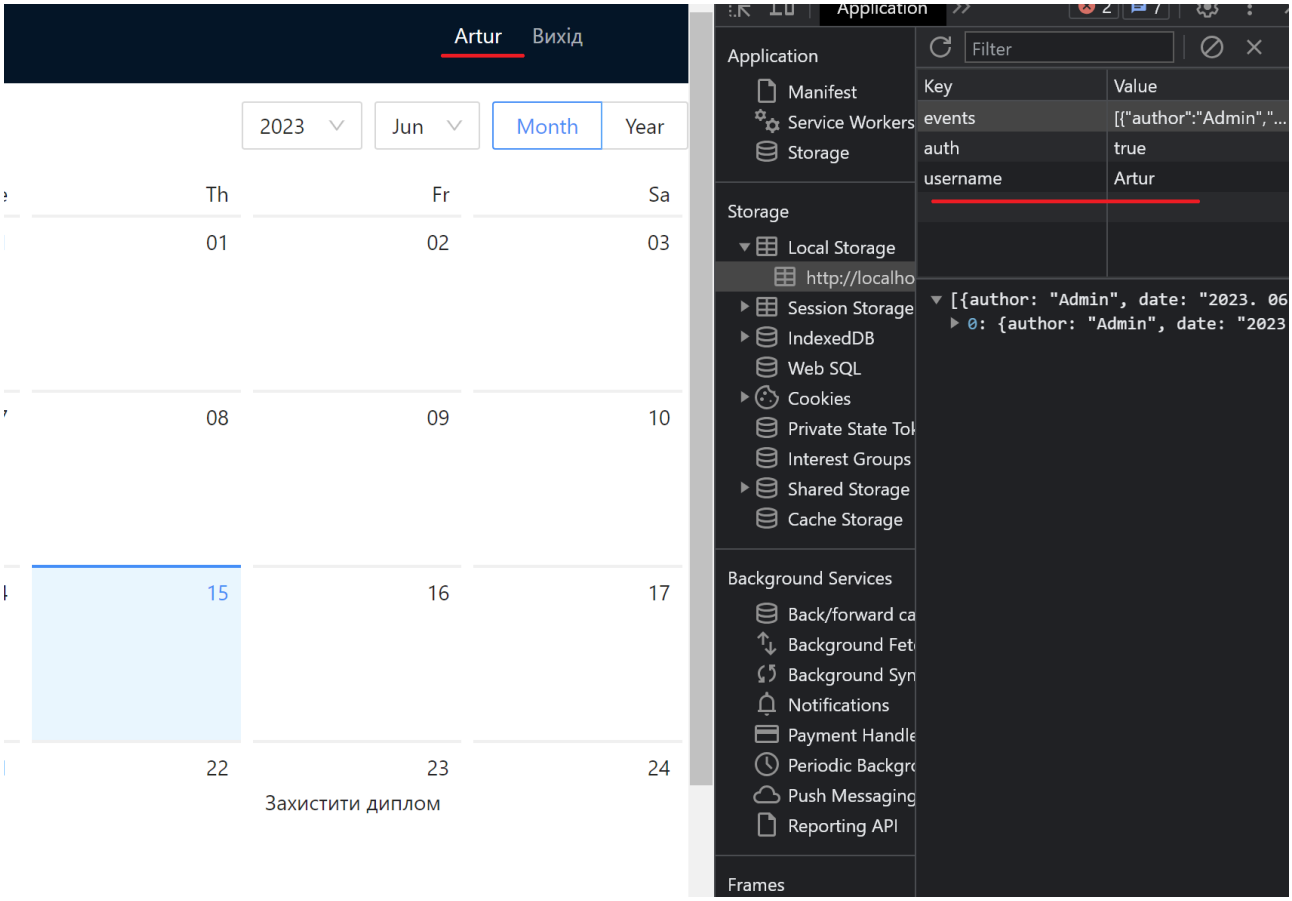

Рисунок 2.38 – Авторизація від мені користувача Artur

Отже, на рисунку вище можна побачити, що нотатка доступна для користувача Artur, який виконує роль гостя, також видно у локальному сховищі, що користувач дійсно інший, та нотатка яку створив Admin також там зберігається зі всіма даними. Як виглядає сам код блоку форми для створення нотаток буде наведено на рисунку нижче (Див. рисунок 2.39).

```
<Form onFinish={submitForm}>
    <Form.Item label="Опис події" name="description" rules={[rules.required()]}>
        <Input
            value={event.description}
            onChange={e => setEvent( value: {...event, description: e.target.value})}
        />
    </Form.Item>
    <Form.Item label="Дата події" name="date"
        rules={[rules.required(), rules.isDateAfter('Замітки можна створювати лише в майбутньому часі')]}>
        <DatePicker onChange={(date) => selectDate(date)}/>
    </Form.Item>
    <Form.Item label="Виберіть гостя" name="guest" rules={[rules.required()]}>
        <Select onChange={(guest: string) => setEvent( value: {...event, guest})}>
            {props.guests.map(guest =>
                <Select.Option key={guest.username} value={guest.username}>
                    {guest.username}
                </Select.Option>
            )}
        </Select>
    </Form.Item>
    <Row justify='end'>
        <Form.Item>
            <Button type="primary" htmlType="submit">Створити</Button>
        </Form.Item>
    </Row>
</Form>
```

Рисунок 2.39 – Блок форми створення нотатки

Для роботи з датою, я використовував компонент “DatePicker”, та передавав у нього вибрану користувачем дату. Для того щоб правильно форматувалась дата, я також передавав вибрану дату у функцію “formatDate” для форматування дати, в залежності від місяця, року, чи дня.

Було створено три змінних для форматування дати в роках, місяцях та днях, усі три змінні були повернуті у вигляді стрічки (Див. Рисунок 2.40).

```
export const formatDate = (date: Date): string => {
    const year = date.getFullYear()
    const month = date.getMonth() < 9 ? `0${date.getMonth() + 1}` : date.getMonth() + 1
    const day = date.getDate() < 10 ? `0${date.getDate()}` : date.getDate()
    return `${year}. ${month}. ${day}`
}
```

Рисунок 2.40 – Форматування дати

У випадку з місяцями у даті потрібно ще перевіряти чи порядковий номер місяця менше дев’яти, якщо менше, то перед цим номером добавляю нуль, а якщо порядковий номер місяця більше дев’яти, додаю один. У результаті я повертаю рядок з датою склеївши їх за допомогою шаблонних літералів і через крапку отримую відформатовану дату.

2.5.2 Валідація створення нотаток

Для валідації форми створення нотатки я використав схожі правила, я з формою авторизації окрім того що виконую перевірку на те, чи дата яку обрано для замітки в майбутньому часі, я розробив створення нотаток таким чином, щоб їх неможливо було записувати у минулий час, наприклад на вчорашній день. Тому при спробі створити нотатку на час який в минулому, виникне помилка з повідомленням.

Також обов’язковими рядками є текст нотатки, та гість якого було запрошено до нотатки. Нижче продемонстрована функція яка перевіряє дату та приймає параметром повідомлення помилки (Див. рисунок 2.41).

```
isDateAfter: (message: string) => () => ({
    validator(_:any, value: Moment) {
        if(value.isSameOrAfter(moment())) {
            return Promise.resolve()
        }
        return Promise.reject(new Error(message))
    }
})
```

Рисунок 2.41 – Функція для перевірки дати на валідність

При спробі створити нотатку та не заповнити обов’язкові поля, або ввести дату на минулий час, користувач побачить повідомлення з помилкою (Див. рисунок 2.42).

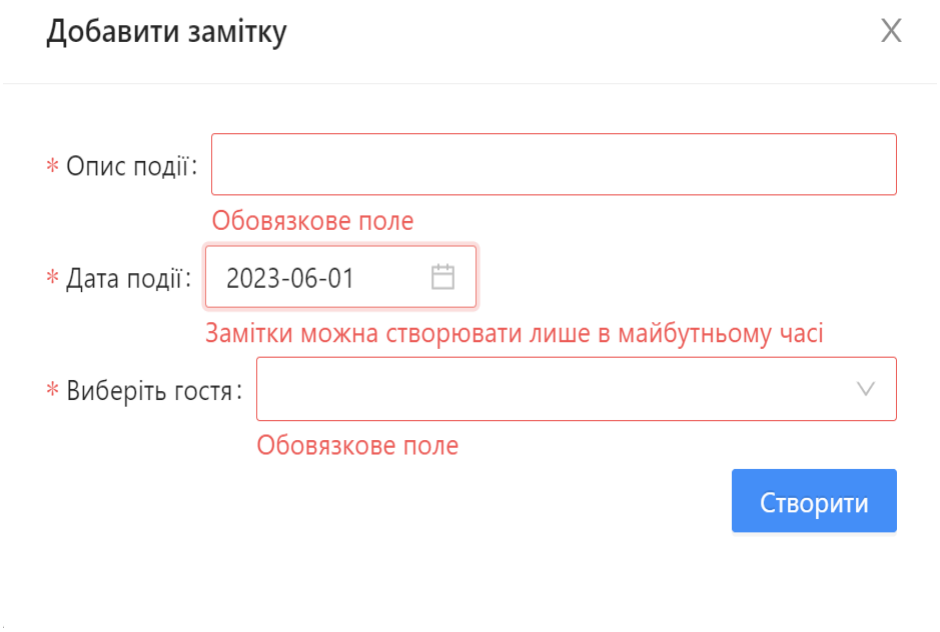

Рисунок 2.42 – Перевірка на валідність форми створення нотатки

Повідомлення у випадку першого та останнього поля, беруться з стандартного параметра про який я писав вище. (Див. рисунок 2.43).

```
<Form.Item
    label="Дата події"
    name="date"
    rules={[rules.required(), rules.isDateAfter('Замітки можна створювати лише в майбутньому часі')]}>
    <DatePicker
        onChange={(date) => selectDate(date)}
    />
</Form.Item>
```

Рисунок 2.43 – Передача повідомлення у функцію перевірки дати

Повідомлення про неправильне заповнення дати я передаю у функцію яка обробляє валідність дати, та повертає повідомлення про помилку.

2.6 Висновок до другого розділу

В другому розділі кваліфікаційної роботи відбулось завершення розробки веб-календаря. Реалізовано інтерфейс проекту. Створено масив користувачів для тестування функціоналу додатка. Проведена робота із керуванням станом глобальних даних. Було розроблено сторінку авторизації з перевіркою на логін і пароль, та сторінку календаря. Реалізовано функціонал для створення нотаток з повною перевіркою на валідність. Також була проведена робота зі збереженням даних у локальне сховище, та робота з датою.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

3.1 Шляхи підвищення життєдіяльності людини

Проблема збереження життєдіяльності людини має глобальний характер, тому кожна людина повинна зробити свій внесок у її вирішення як заради сьогоднішнього суспільства, так й для добробуту прийдешніх поколінь.

Здоров'я людини ґрунтується на основі генетичних факторів, способу життя, свідомого ставлення людини до себе.

Здоров'я людини – стан повного соціального, біологічного і соціологічного комфорту, коли функції усіх органів і систем організму зрівноважені з природним і соціальним середовищем, відсутні будь-які захворювання, хворобливі стани та фізичні дефекти.

Критерій здоров'я визначається комплексом показників, а саме у розумному ставленні до свого здоров'я, фізичній та психічній культурі, загартовуванні організму, вмілій організації праці та відпочинку.

Але суттєві зміни побутових та соціально-культурних умов життя населення, передусім значним зменшенням його фізичної активності призвели до поширення епідемії неінфекційних захворювань (зокрема серцево-судинних) людини, тому одним із шляхів підвищення життєдіяльності людини є оздоровча фізична культура.

У сучасних умовах в Україні склалася критична ситуація із станом здоров'я населення. Різко зросла захворюваність, у тому числі на гіпертонію – у три рази, стенокардію – у 2,4 рази, інфаркт міокарду – на 30 відсотків. Несприятливі природні умови стали причиною зростання кількості хворих дітей. У молодого покоління різко прогресують хронічні хвороби серця, неврози, артрити, сколіози, ожиріння тощо. Причиною цього є зростаюча популярність у дитячому та молодіжному середовищі привабливих видів нефізичної діяльності. За даними державної статистики, 13% наших співгромадян займаються фізичною культурою та спортом. Це в чотири рази

менше, ніж у Фінляндії, утричі – ніж у Канаді та вдвічі порівняно з Німеччиною, Японією, США і Францією.

Основним критерієм фізичного здоров'я варто вважати енергопотенціал біосистеми, оскільки життєдіяльність будь-якого живого організму залежить від акумуляції і мобілізації енергії для забезпечення фізичних функцій. Отже, стан фізичного розвитку людини характеризується його антропометричними даними.

Знання антропометричних характеристик людини необхідно враховувати при вирішенні багатьох питань життєдіяльності. З врахуванням антропометричних характеристик здійснюється життя і діяльність людини в навколишньому середовищі.

Організація умов предметної діяльності людини, без врахування антропометричних показників, ускладнюють роботу, підвищують напруженість усього організму, викликають швидку втому, що призводить до зниження працездатності та проблем безпеки.

Дослідження показують, що під впливом фізичних тренувань суттєво покращуються функції основних органів і систем людини. Добре тренована людина на протязі восьмі годин може витримувати навантаження в межах 50%, а нетренована – лише 25% максимальної аеробної здатності. Аеробна здатність організму, а тому й витримування фізичних навантажень, залежить від стану систем транспортування кисню.

Основною метою фізичних тренувань є поліпшення стану серцево-судинної, дихальної, м'язової, а також інших систем організму шляхом максимальної активізації їх функціональних резервів.

Для того, щоб досягти фізичного стану, навантаження під час тренування повинні бути досить інтенсивними і тривалими. Важливим фактором високої ефективності і безпеки фізичних тренувань є поступове збільшення навантаження і суворий лікарський контроль.

Спостереження вчених показують, що здоров'я людини на 45% – 50% визначається способом життя, на 20% – впливом навколишнього середовища, на 20% – спадковістю, на 8-10% – охороною здоров'я, харчуванням.

Фізичні вправи діють як природний фактор, який знищує надлишки гормонів і допомагає організму вернутися до стану гармонії.

В окремо взятій фізичній вправі або комплексі фізичних вправ виділяють зовнішній і внутрішній аспекти. Зовнішній аспект фізичної вправи – це ритмічні характеристики руху: час, амплітуда, швидкість, зовнішні зусилля, прискорення, положення тіла в просторі.

Оздоровча фізична культура є важливим фактором збільшення фізичних можливостей і продовження життя людини, так як людина, як і будь-який живий організм, активно взаємодіє зі всіма компонентами середовища, в якому перебуває. Для підвищення ефективності трудової діяльності працююча людина повинна знати динаміку працездатності, її різні фази.

Фаза розвитку - організм пристосовується, налаштовується до режиму роботи. Поліпшується координація, точність і швидкість руху, вимальовується оптимальна робоча поза, встановлюється економний режим дихання і кровообігу. Фаза стійкого робочого стану, відрізняється найвищою індивідуальною ефективністю роботи. Через 3-4 години працездатність знижується. До цього часу передбачений перерву, після закінчення якого настає фаза розвитку, але коротша, ніж на початку трудового дня. Сталий робочий стан триває менше і переходить у фазу часткового стомлення, що вимагає мобілізації ресурсів організму.

Велике значення має період поступового «входження в роботу». Треба починати її без зайвої стрімкості і квапливості. «Галоп з місця» шкідливий не тільки у фізичному, а й у розумовій праці. Будь-яка нова функція повинна урівноважитися з системою вже наявних функцій і навичок. Добре продумана і відпрацьована звична послідовність і певна система у праці, планова діяльність завжди більш продуктивні.

3.2 Санітарно-гігієнічні вимоги до умов праці

Гігієна праці розглядає питання, пов'язані з умовами роботи і їхнім впливом на людський організм; розробляє гігієнічні і лікувально-профілактичні заходи, спрямовані на поліпшення і збереження здоров'я працівників, підвищення працездатності і продуктивності праці. Діяльність людини, залежно від умов реалізації і особливостей технологічних процесів, може супроводжуватись суттєвим відхиленням параметрів виробничого середовища від їх природного значення, бажаного для забезпечення нормального функціонування організму людини.

Уникнути небажаного впливу техногенної діяльності людини на стан виробничого середовища і довкілля в цілому практично нереально. Тому метою гігієни праці є встановлення таких граничних відхилень від природних фізіологічних норм для людини, допустимих навантажень на організм людини, які не будуть викликати негативних змін у функціонуванні організму людини і окремих його систем.

На сучасному етапі розвитку гігієни праці як науки, гігієністи при вирішенні питань охорони здоров'я працюючих дотримуються так званого порогового принципу: фактичне відхилення окремого чинника виробничого середовища від природної фізіологічної норми до певної межі не спричиняє небажаних змін в організмі людини і не призведе до негативних наслідків.

Особливе значення має оцінювання умов праці, яке є основою для вжиття заходів, необхідних для запобігання небезпекам або зведення їх до мінімуму.

Шкідливі фактори виробничого середовища відповідно до Гігієнічної класифікації поділяються на фізичні, хімічні, біологічні та фактори трудового процесу.

Умови праці значною мірою залежать від стану виробничого середовища, яке характеризується мікрокліматом. Комфортні параметри виробничого мікроклімату (температура, відносна вологість, швидкість руху повітря) для кожного конкретного випадку визначаються в нормативному документі -

системі стандартів безпеки праці, і є обов'язковим для всіх виробництв і для виробництв у різних географічних розташуваннях.

В основу принципу нормування метеорологічних умов виробничого середовища покладена диференційна характеристика оптимальних і допустимих метеорологічних умов в робочому середовищі залежно від теплової характеристики виробничого приміщення, категоріїробіт і періоду року.

Під оптимальними мікрокліматичними умовами розуміють такі співвідношення параметрів мікроклімату, котрі при дії на людину забезпечують нормальний функціональний тепловий стан організму без залучення механізму терморегуляції. Внаслідок цього забезпечується тепловий комфорт, що значною мірою впливає на працездатність.

Висока температура також негативно діє на організм працюючого: зростає навантаження на серцево-судинну систему, частішим стає дихання, підвищується температура тіла, збільшується виділення поту, що порушує водно-сольовий і вітамінний баланс в організмі. Все це викликає швидку втому, а отже, й різке зниження продуктивності праці. Особливу турботу про мікрокліматичний режим слід проявляти у тваринницьких приміщеннях. Відхилення від встановлених норм не тільки погіршує стан робітника, а й знижує продуктивність тварин.

Тому обладнання приміщень опалювальними пристроями, вентиляційними системами, дозволяє одночасно здійснювати обмін повітря і регулювати температурний режим, підвищувати продуктивність праці, зберігати здоров'я працівників і збільшувати вихід тваринницької продукції.

Основними засобами захисту, які зменшують загазованість повітряного середовища в робочій зоні, є встановлення в кабінах тракторів кондиціонерів, систематичний догляд за двигунами та їх регулювання, усунення несправностей, а також застосування індивідуальних засобів захисту. Боротьбі із загазованістю в робочих приміщеннях сприяє вентиляційне і калориферне обладнання. Це значно зменшує вологість повітря, бактеріальну забрудненість,

особливо в тваринницьких приміщеннях, що поліпшує умови праці робітників і зменшує витрати кормів для тварин.

Між санітарно-гігієнічними, психофізіологічними й естетичними умовами праці існує тісний органічний зв'язок. Відомо, що освітлення належить до санітарно-гігієнічних умов, а колір - до естетичних. Але якщо виробничі приміщення погано освітлені, то невтішним є й раціональне кольорове оформлення. Блакитне здається сірим, зелене - брудно-сірим тощо. Дослідженнями наукових установ встановлено вплив кольорової гами на працюючу людину. Так, червоний колір швидко притягує увагу; синій - дещо сприяє зняттю втоми; зелений - створює оптимістичний настрій, знижує зорове напруження, підвищує працездатність рук; оранжевий - підвищує настрій; брунатний, чорний - суворий, тяжкий, гнітюче діє на психіку людини. Отже, на основі зазначеної дії кольорів на людину слід і застосовувати кольорове оформлення робочих місць та знарядь.

Переважна більшість працездатного населення є офісними працівниками. Причому їх кількість у порівнянні з представниками інших галузевих професій, щороку постійно зростає. площа приміщення повинна бути не менше 6,0 м² на 1 робоче місце; робочі місця повинні бути розташовані на відстані не менше ніж 1 м від стіни з вікном, і 1,4 м від звичайної стіни; відстань між бічними поверхнями комп‘ютерів має бути не меншою за 1,2 м; відстань між тильною поверхнею одного комп‘ютера та екраном іншого не повинна бути меншою 2,5м. Відповідні робочі місця заборонено облаштовувати у підвальних або цокольних приміщеннях будинків.

У приміщеннях, де здійснюється робота з комп’ютерами, щодня має проводитися вологе прибирання з метою недопущення запиленості підлоги та меблів. Крім того, має бути обладнана кімната психологічного розвантаження.

Конструкція робочого столу та крісла користувача персонального комп’ютера має забезпечити підтримання оптимальної робочої пози та забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання і документів.

3.3 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи описано основні шляхи підвищення життєдіяльності людини, та проаналізовано причини погіршення життєдіяльності. Проаналізовано у кого із населення інтенсивно знижується рівень життєдіяльності, та описано поради, щодо підвищення рівня фізичного та психологічного стану людини.

Також розглянуто основні санітарно-гігієнічні вимоги до умов праці людини у різних сферах діяльності. Проаналізовано чому висока температура погано діє на умови працюючого, та наслідки праці в таких умовах. Було розглянуто умови праці в офісних приміщеннях у роботі з комп’ютерами. Проаналізовано основні засоби захисту повітря в робочій зоні.

ВИСНОВКИ

Отже, модернізація та оновлення будь-яких веб-сторінок, є важливим аспектом у сучасному світі. Виконуючи кваліфікаційну роботу було розроблено веб-календар за допомогою JavaScript бібліотеки – React JS із взаємодією з мовою типізації TypeScript. Умовно проектування завдання можна поділити на три етапи. Це побудова архітектури, створення інтерфейсу, та реалізація функціоналу.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Розглянуто найкраще та найсучасніше середовище для написання JavaScript коду та проаналізовано його переваги.

– Подано загальну інформацію про архітектуру проекту, та його файлову структуру.

– Висвітлено та проаналізовано сучасні веб-технології у фронтенді для створення одно-сторінкових додатків.

– Обґрунтовано важливість використання типізації у складних проектах, та використання бібліотек для керування глобальними даними.

В другому розділі кваліфікаційної роботи:

– Розроблено одно-сторінковий веб-додаток календаря, для створення нотаток у реальному часі, із збереженням даних у локальне сховище.

– Реалізовано авторизацію користувача у додаток, за допомогою логіна та пароля, заданих вручну у масиві усіх користувачів.

– Описано процедуру створення нотатки, перевірки її на валідність, та запрошення інших користувачів.

У розділі «Безпека життєдіяльності, основи хорони праці» Було висвітлено основні проблеми життєдіяльності людини у сучасному світі, та шляхи для її підвищення. Також описано основні санітарно-гігієнічні вимоги до умов праці людини у різних сферах діяльності.

ПЕРЕЛІК ДЖЕРЕЛ

1. Документація щодо роботи з React JS [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn.

2. Документація щодо роботи з TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

3. Документація щодо роботи з Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/tutorials/essentials/part-1-overview-concepts.

4. Використання бібліотеки Antd design [Електронний ресурс] – Режим доступу до ресурсу: https://ant.design/components/overview/?theme=dark.

5. Документація щодо роботи з Local Storage [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

6. Веб-технології та веб-дизайн [Електронний ресурс] – Режим доступу: http://victoria.lviv.ua/html/wp/index.html – Дата доступу: 30.05.2023. – Загол. с екрана.

7. Моделі потоків даних (DFD-моделі): призначення, місце застосування в системному аналізі, правила побудови, приклади [Електронний ресурс] – Режим доступу: http://victoria.lviv.ua/html/wp/index.html – Дата доступа: 15.05.2023.

8. Основні методи аналізу виробничого травматизму [Електронний ресурс] – Режим доступу: http://pidruchniki.com/15341220/bzhd/ – Дата доступу: 12.04.2023. – Загол. з екрану.

9. Ергономічні вимоги до організації робочих місць [Електронний ресурс] – Режим доступу: http:// pidruchniki.com/14821111/bzhd – Дата доступу: 12.04.2023. – Загол. з екрану.

10. Охорона праці користувачів персональних комп'ютерів (ПК) [Електронний ресурс] – Режим доступу: http://library.if.ua/book/9/1016.html – Дата доступу: 20.04.2023.

11. Загальні вимоги до виробничих приміщень для експлуатації ВДТ ЕОМ та ПЕОМ [Електронний ресурс] – Режим доступу: http://mybiblioteka.su/11-115960.html – Дата доступу: 20.04.2023.

12. Перша долікарська допомога [Електронний ресурс] – Режим доступу: http://ua-referat.com/Перша_долікарська _допомога – Дата доступу: 22.04.2023.

13. Надання першої допомоги потерпілим [Електронний ресурс] – Режим доступу: http://referatu.net.ua/newreferats/448/187221 – Дата доступу: 20.04.2023.

14. Основи охорони праці [Електронний ресурс] – Режим доступу: http://ua.textreferat.com/referat-2127-1.html – Дата доступу: 20.04.2023.

15. Гандзюк, М. П. Основи охорони праці [Текст] : підручник / М. П. Гандзюк, Є. П. Желібо, М. О. Халімовський ; за ред. М. П. Гандзюка ; МОН України. – 4-е видання. – К. : Каравела, 2013. – 384 с. – ISBN966-8019-01-6.

16. Гогіташвілі, Г. Г. Основи охорони праці [Текст] : навчальний посібник / Г. Г. Гогіташвілі, В. М. Лапін ; 4-те вид. випр. і доп. – К. : Знання, 2013. – 302 с. – ISBN 978-966-346-400-8.