Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту

бакалавр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: Дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів

Виконала: студентка 4 курсу, групи СПз-41

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва спеціальності (напряму підготовки))

Сельська Н.П.
(підпис) (прізвище та ініціали)

Керівник Стефанишин В.М.
(підпис) (прізвище та ініціали)

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

Рецензент Млинко Б.Б.
(підпис) (прізвище та ініціали)

м. 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 Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20_23__ р.

З А В Д А Н Н Я
НА ДИПЛОМНІЙ ПРОЄКТ СТУДЕНТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

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

студенту Сельська Наталія Петрівна
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів

Керівник роботи Стефанишин В.М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*08*» березня 2023 року № *4/7-282*

2. Термін подання студентом завершеної роботи 12 червня 2023

3. Вихідні дані до роботи Підсумок розробки веб-додатку для фітнес сайту з використанням TypeScript vtn

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ аналіз предметної області аналіз існуючих рішень в галузі SPA-розробки, Typescript як інструмент для створення веб-додатку,фітнес-сфери, розробка прототипу веб-додатку на прикладі сайту з занять фітнесом з авторизацією, висновки,додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Презентація зроблена в Canva 13 слайдів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності та охорона праці	Сенчишин .С.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Початок створення вступу та розподіл на частини проєкт	20.03	**виконано**
2	Створення першого розділу	03.04	**Виконано**
3	Створення другого розділу	12.04	**Виконано**
4	Створення третього розділу	28.04	**Виконано**
5	Створення четвертого розділу	05.05	**Виконано**
6	Створення п’ятого розділу та написання програми	10.05	**Виконано**
7	Підсумок зробленої роботи редагування тексту	16.05	**Виконано**
8	Оформлення висновків та списку джерел	10.06	**Виконано**
9	Безпека життєдіяльності та охорона праці	10.06	**Виконано**
10	Попередній захист	30.05	**Виконано**
11	Захист	14.06	**Виконано**

Студент _____ Сельська Н.П.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

«Дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів» // Дипломна робота ОР «Бакалавр» // Сельська Наталія Петрівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно- інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПз-41 // Тернопіль, 2023 // С. 63, табл. – 0, рис. – 7, додат. – 1.

Ключові слова: WEB- САЙТ, САЙТ ЗАНЯТТЯ ФІТНЕСОМ, HTML, CSS, TYPESCRIPT, АВТОРИЗАЦІЯ, SPA, ОДНОСТОРІНКОВИЙ САЙТ.

Мета дипломної роботи полягає у дослідженні перспектив розробки Single Page Applications (SPA) з використанням TypeScript на прикладі створення веб-додатку для занять фітнесом з авторизацією користувачів. Робота передбачає проведення аналізу існуючих рішень в галузі SPA-розробки, дослідження особливостей авторизації користувачів та розробку веб-додатку з використанням вищезазначених технологій та методологій. Основними завданнями роботи є визначення ефективності використання TypeScript для розробки SPA, дослідження можливостей використання SPA для веб-додатку фітнес-сфери та розробка прототипу веб-сайту з використанням TypeScript та SPA-архітектури з авторизацією користувачів.

У першому розділі було проаналізовано етапи предметної області, також аналіз потреб та вимог до користувачів. Також було проведено аналіз користувачів, клієнтів та дизайну сайту конкурентів. Їх плюси та мінуси. Для створення власного веб-додатку.

У другому розділі було проведено огляд на популярні Spa-фреймворки. Провели аналіз існуючих рішень в галузі SPA-розробки, було проведено

порівняння SPA-розробки з традиційними підходами. Також, проведено дослідження найефективніших підходів до розробки SPA-додатків.

У третьому розділі описується робота Typescript як інструмент для створення веб-додатку, його особливості у застосуванні. Плюси та мінуси Typescript в SPA-розробці. Та проведено кращі практики використання TypeScript в Spa розробці.

У четвертому розділі особливості авторизації користувачів у веб-додатках. Проведено загальний огляд реєстрації та авторизації. Бло показано переваги та недоліки різних видів авторизації.

У п’ятому розділі було розроблено прототип веб-додатку з викоритсанням Typescript на прикладі занять з фітнесом. Було описано вимоги до прототипу.Та також проведено опис сторінки автентифікації та головної сторінки.

У шостому розділі розкриваються питання з безпеки життєдіяльності та основи охорони праці.

У результаті підготовки дипломної роботи розроблено прототип веб-додатку для фітнес-сайту на прикладі SPA розробки.

ANNOTATION

"Investigation of the prospects of SPA development using TypeScript on the example of a site for fitness classes with user authorization" // Bachelor's thesis // Selska Nataliya Petrivna // Ivan Pulyuy Ternopil National Technical University, Faculty of Computer Information Systems and Software of Engineering, Department of Software Engineering, Group SPz-41 // Ternopil, 2023 // P. 63, tab. – 0, fig. – 7, add. - 1.

Keywords: WEB SITE, FITNESS SITE, HTML, CSS, TYPESCRIPT, AUTHORIZATION, SPA, SINGLE PAGE SITE.

The aim of the thesis is to investigate the prospects of developing Single Page Applications (SPA) using TypeScript on the example of creating a web application for fitness classes with user authorization. The work involves the analysis of existing solutions in the field of SPA development, the study of the features of user authorization, and the development of a web application using the above technologies and methodologies. The main tasks of the work are determining the effectiveness of using TypeScript for SPA development, researching the possibilities of using SPA for a fitness web application, and developing a prototype website using TypeScript and SPA architecture with user authorization. In the first section, the stages of the subject area were analyzed, as well as the analysis of needs and requirements for users. An analysis of users, customers and competitors' website design was also carried out. Their pros and cons. To create your own web application.

In the second section, an overview of popular Spa frameworks was conducted. An analysis of existing solutions in the field of SPA development was carried out, a comparison of SPA development with traditional approaches was carried out. Also, a study of the most effective approaches to the development of SPA applications was conducted.

The third chapter describes the work of Typescript as a tool for creating a web

application, its features in application. Pros and cons of Typescript in SPA development. And the best practices of using TypeScript in Spa development were carried out.

In the fourth section, features of user authorization in web applications. A general review of registration and authorization was conducted. Advantages and disadvantages of various types of authorization were shown.

In the fifth chapter, a prototype web application was developed using Typescript on the example of fitness classes. The requirements for the prototype were described. The authentication page and the main page were also described. In the sixth chapter, issues of life safety and the basics of labor protection are revealed.

As a result of the preparation of the thesis, a prototype of a web application for a fitness site was developed based on the example of SPA development.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

SPA - single page application

DOM - document Object Model

XSS - Cross-Site Scripting

AJAX – Asynchronous

CSRF - Cross-Site Request Forgery

SEO - Search Engine Optimization

CSRF - Cross-Site Request Forgery

XSS - Cross-Site Scripting

API - Application Prog ramming

HTTPS - HyperText Transfer Protocol Secure

CSRF - Cross-Site Request Forgery

JSON - JavaScript Object Notation

RBAC - Role Based Access Control

ВСТУП

Актуальність дослідження полягає в тому, що в сучасному світі веб-сайти стають все більш популярними, і вони є незаміною частиною бізнесу в багатьох галузях, включаючи фітнес-сферу. SPA-розробка є однією з найбільш популярних технологій розробки веб-додатки, оскільки вона забезпечує користувачам швидку та зручну роботу з додатком, зменшує час завантаження сторінок та забезпечує більш відчутну взаємодію з користувачами.

Крім того, TypeScript є однією з найбільш популярних мов програмування для веб-додатків на JavaScript, що дозволяє розробникам зменшити кількість помилок в коді, забезпечує більшу стабільність додатків та полегшує підтримку додатків в майбутньому.

Таким чином, дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів є актуальним і важливим, оскільки воно дозволить вивчити особливості розробки SPA на TypeScript, дослідити ефективність цієї технології в порівнянні з традиційними методами розробки, та дати рекомендації щодо подальшої розробки та використання веб-додатків у фітнес-сфері.

Мета і завдання дослідження. Метою дослідження є вивчення можливостей розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів, а також дослідження ефективності використання такої технології у порівнянні з традиційними методами розробки веб-додатків.

Задачі дослідження включають:

- Аналіз особливостей SPA-розробки та TypeScript як мови програмування.
- Розробка прототипу сайту для занять фітнесом з авторизацією користувачів з використанням SPA та TypeScript.
- Дослідження швидкості та ефективності роботи прототипу в порівнянні з

традиційними методами розробки.

- Вивчення особливостей взаємодії користувача зі створеним прототипом.
- Розробка рекомендацій щодо використання SPA з використанням TypeScript у розробці веб-додатків у фітнес-сфері.

Таким чином, завданням дослідження є проведення аналізу та вивчення можливостей розробки SPA з використанням TypeScript, розробка прототипу сайту для занять фітнесом з авторизацією користувачів на цій технології, а також дослідження його ефективності та взаємодії з користувачами. В результаті дослідження будуть сформульовані рекомендації щодо подальшого використання SPA з TypeScript у розробці веб-додатків у фітнес-сфері.

Об'єктом дослідження є розробка SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів. Дослідження передбачає аналіз технології SPA-розробки, вивчення можливостей TypeScript як мови програмування, а також дослідження ефективності використання цієї технології у порівнянні з традиційними методами розробки веб-додатків. В результаті дослідження будуть отримані рекомендації щодо використання SPA з TypeScript у розробці веб-додатків у фітнес-сфері, які можуть бути корисні для програмістів та розробників веб-додатків.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Етапи аналізів предметної області

Для створення власного сайту, потрібно проводити аналізи наступних етапів:

- Аналіз потреб та вимог користувачів. Важливо визначити, яку мету має створюваний сайт та яку аудиторію він буде обслуговувати.
- Аналіз конкурентів. Варто вивчити сайти конкурентів, їхні функції, дизайн, інтерфейс та інші аспекти, щоб знайти свої переваги та вигідно відрізнитись від них.
- Аналіз дизайну та інтерфейсу. Важливо визначити, як буде виглядати сайт, який дизайн буде використаний, які функції та інтерфейс будуть доступні користувачам.
- Аналіз контенту. Варто визначити, який контент буде розміщений на сайті, які матеріали будуть надаватися користувачам та як вони будуть структуровані.

Для проведення аналізів також використано різноманітні джерела, наприклад, онлайн-ресурси, книги, журнали, статистичні дослідження, аналізи даних, опитування користувачів тощо. Також можна звернутись до фахівців, які мають досвід у розробці сайтів та надати їм запит на проведення аналізів.

1.1.1 Аналіз потреб та вимог користувачів

Аналіз потреб та вимог користувачів для веб-сайту з занять фітнесом можна провести на основі два кроки:

- Вивчення цільової аудиторії - потенційних користувачів веб-сайту з заняттями фітнесом, їх вікових та статевих характеристик, основних потреб та мотивації для занять фітнесом.
- Опитування користувачів - проведення опитування серед потенційних

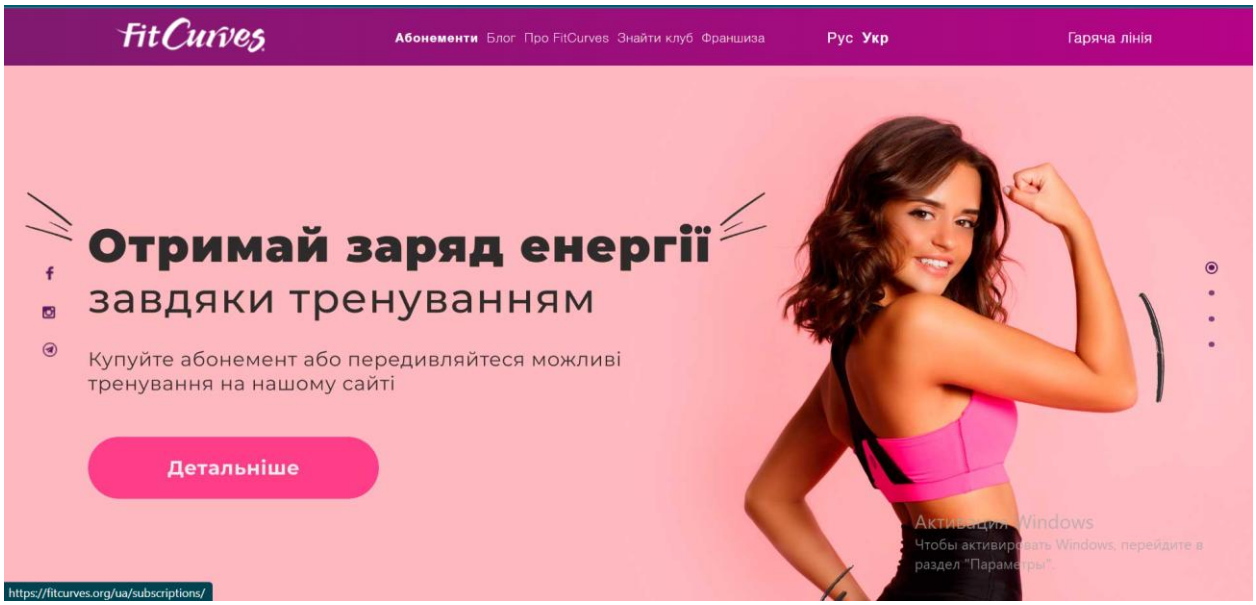
користувачів з метою вивчення їхніх потреб та вимог до веб-сайту з заняттями фітнесом.

На основі зібраних даних можна скласти аналіз потреб та вимог користувачів для веб-сайту з заняттями фітнесом.

- Цільова аудиторія це в більшості жінки віком від 18-40 років. Основною потребою являється схуднення та підтримка тіла у тонусі. Загалом, важливо зрозуміти, що аналіз потреб та вимог користувачів для веб-сайту з заняттями фітнесом є важливим етапом у розробці сайту, оскільки він допомагає врахувати потреби користувача.
- Зібрані дані показали, що користувач більш за все зацікавлені в онлайн-заняттях з фітнесу, бажають мати доступ до занять онлайн. Також можна визначити, що користувачі шукають сайт зі зручною навігацією, на якому можна швидко знайти інформацію про, опис послуг та професійність тренерів.

1.1.2 Аналіз конкерентів

Проведено аналіз сторінки конкурента. Конкурент “FitCurves”, який видно на рисунку 1.2.1.



“Рисунок 1.1 – “FitCurves” web-сторінка конкурента ”

Сайт що зображений на рисунку 1.1 пропонує жінкам послуги фітнес-клубу FitCurves, який спеціалізується на жіночому фітнесі. Оцінюючи сайт FitCurves з точки зору конкурентної стратегії, можна виділити наступні позитивні та негативні моменти:

Позитивні моменти:

- Зручність використання сайту: сайт має просту та зрозумілу навігацію, що дозволяє швидко знайти необхідну інформацію про клуб та його послуги.
- Сучасний дизайн: сайт має свіжий та яскравий дизайн, що привертає увагу та вражає відвідувачів.
- Привабливі пропозиції: сайт пропонує різні привабливі акції та знижки, що залучають клієнтів та стимулюють їх до реєстрації на тренування.

Негативні моменти:

- Високі ціни. Fitcurves може бути дорожчим за інші фітнес-клуби, що може зменшувати його привабливість для деяких клієнтів.
- Обмежені можливості тренувань. Fitcurves спеціалізується на жіночому фітнесі та може не мати багато різноманітних тренувань та устаткування, яке пропонують інші клуби.
- Обмежена гнучкість графіку. Fitcurves може мати обмежені години роботи, що може ускладнювати розклад занять.

Тому мета створити сучасний сайт який буде привертати увагу, також який повинен бути легким у навігації, з помірними цінами, з відкритим допуском у необмежений час та має містити тренуваня на різні види складності.

1.1.3 Аналіз дизайну та інтерфейсу

Загальний враження від дизайну та інтерфейсу сайту FitCurves “(див. рисунок 1. 2.1)” досить позитивне. Сайт має сучасний вигляд та звертає на себе увагу своїми яскравими кольорами, анімацією та зображеннями.

Кольорова палітра сайту використовує основні тони фіолетового та

рожевого кольорів, які відображають жіночність та енергію. Однак, варто зазначити, що дизайн сайту може бути трохи перевантаженим для деяких користувачів, особливо тих, які швидко втомлюються від великої кількості інформації та графічних елементів.

Щодо інтерфейсу, він в цілому є достатньо зрозумілим та простим для навігації. Меню навігації розташоване вгорі сторінки та містить основні категорії, такі як "Про нас", "Послуги", "Ціни", "Акції" та "Контакти". Додаткові посилання до інших розділів сайту також є на головній сторінці та внизу сторінки що досить облегшує перехід на наступний розділ.

Навігація сайту дуже зручна та інтуїтивно зрозуміла На сайті також присутній пошуковий рядок, що дозволяє швидко знайти потрібну інформацію.

Загалом, дизайн та інтерфейс сайту “Fitcurves” можна оцінити як зручний та функціональний, з певними недоліками в анімації та інтерактивності. Однак, в цілому сайт виконує свою основну функцію - надання інформації про компанію та її послуги та привернення клієнтів.

1.1.4 Аналіз контенту

Головна сторінка:

- На головній сторінці сайту знаходиться основний слайдер з фотографіями та текстом, який привертає увагу користувачів та описує основний напрямок діяльності сайту - створення кращих можливостей для заняття фітнесом.
- Далі йде секція з кроками як попасти на курси занять фітнесом, в яких описується переваги та ціни на кожен з них.
- На головній сторінці також є секція з клієнтськими відгуками, що додає довіри до сайту.

Сторінка "Про нас":

- На цій сторінці користувач може дізнатися більше про історію компанії та її місію.
- Також описуються переваги занять фітнесом у студії FitCurves, що

дозволяє користувачам зрозуміти, що саме компанія пропонує та які вигоди вона надає своїм клієнтам.

Сторінка "Розклад":

- На цій сторінці розміщений розклад занять фітнесом в студії FitCurves.
- Розклад описує різні типи занять, такі як заняття з аеробіки, заняття з силових вправ, танці та інші.
- Користувач може вибрати зручний для себе час та день для заняття.
- Сторінка "Контакти":
- На цій сторінці користувач може знайти адресу студії FitCurves, телефон та електронну пошту для зв'язку з представниками компанії.
- Також на сторінці знаходиться форма зворотнього зв'язку, яку можна використовувати для зв'язку зі студією.

У загальному, сайт містить інформацію про послуги, які надає студія FitCurves, а також детальний опис різних типів занять фітнесом, що доступні користувачам. Сайт містить інформацію про розклад занять, що дозволяє клієнтам легко знайти зручний для себе час та день для занять. Крім того, на сайті знаходяться відгуки клієнтів, які описують їхній досвід відвідування студії, що додає довіри до компанії.

Загалом, вміст сайту є достатньо інформативним та легко зрозумілим для користувачів. На сайті є достатньо інформації про компанію та її послуги, що дозволяє клієнтам зробити правильний вибір. Також, на сайті присутній зручний інтерфейс, який дозволяє користувачам легко знайти потрібну інформацію та зареєструватися на заняття. Отже, загалом контент на сайті “FitCurves”є добре структурованим та підходить для своєї мети - привернення клієнтів та надання їм інформації про компанію та її послуги.

1.2 Створення дизайну

Враховуючи усі проведені етапи аналізів таких як : аналіз потреб та вимог користувачів, аналіз конкурентів, аналіз дизайну та інтерфейсу, та аналіз контенту. Виконавши докладний аналіз усіх позитивних та негативних сторонах

конкурента створюємо дизайн власного сайту “KARMA YOGA”. Зображений на рисунку 1.2.

“Рисунок 1.2 – Головна сторінка сайту”

РОЗДІЛ 2
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ В ГАЛУЗІ SPA-РОБЗРОБКИ

2.1 Огляд популярних SPA-фреймворків

Фреймворк — це платформа розробки програмного забезпечення, яка забезпечує основу та структуру для створення програм. Зазвичай він включає набір бібліотек, інструментів і вказівок, які допомагають розробникам створювати додатки ефективніше, забезпечуючи попередньо визначену функціональність, багаторазові компоненти та стандартизовані процеси.

SPA фреймворки - це фреймворки, які дозволяють розробляти веб-додатки, які переважно працюють в браузері та не потребують перезавантаження сторінки при кожному запиті. Ось деякі з найпопулярніших SPA-фреймворків:

Angular: Angular є одним з найбільш популярних фреймворків для розробки SPA. Він був створений компанією Google та має велику підтримку та документацію. Angular має велику кількість функцій та можливостей, але водночас він може бути складним для вивчення для початківців.

Цей фреймворк, завдяки своїй чіткій і жорсткій структурі, проявляє стабільність і надійність, проте вимагає значних ресурсів для розробки застосунків, тому рідко використовується у невеликих проєктах. Однак, він прекрасно підходить для Enterprise-рішень.

React: React також є дуже популярним SPA-фреймворком, створеним компанією Facebook у 2013 році. Він має простий та легкий вивченні синтаксис, а також велику спільноту розробників та багато корисних бібліотек. Однак, React не має такої великої кількості функцій, як Angular.

Цей продукт став першим, який використовує Virtual DOM для оновлення структури, яка перетворюється в HTML і вимагає значних ресурсів). Його оптимізація полягає у впровадженні оновлення тільки тих даних, що зазнали змін,

і використанні спрощеної віртуальної структури для пошуку відмінностей.

React — це не фреймворк, а бібліотека JavaScript для створення інтерфейсів користувача. Його часто використовують як основну бібліотеку в поєднанні з іншими фреймворками або бібліотеками для створення повних фреймворків веб-додатків. React зосереджується на рівні інтерфейсу користувача програми, забезпечуючи декларативний та ефективний спосіб створення багаторазово використовуваних компонентів інтерфейсу користувача.

React дотримується компонентної архітектури, де інтерфейс користувача поділено на невеликі самодостатні компоненти. Ці компоненти інкапсулюють власний стан і поведінку, полегшуючи керування та повторне використання коду. React використовує віртуальну DOM (Document Object Model) для ефективного оновлення та відтворення компонентів, що забезпечує швидші та ефективніші оновлення інтерфейсу користувача.[1]

Vue.js — це прогресивний фреймворк JavaScript для створення інтерфейсів користувача. Її часто називають фреймворком, оскільки вона забезпечує комплексне рішення для розробки повних веб-додатків.

Vue.js має компонентну архітектуру, подібну до React, де інтерфейс користувача розділений на повторно використовувані та самодостатні компоненти. Ці компоненти інкапсулюють власну логіку, розмітку та стиль, що полегшує розробку та підтримку складних інтерфейсів користувача.[2]

Ember: Ember.js — це платформа JavaScript для створення амбітних веб-додатків. Він надає повний набір інструментів, угод і найкращих практик для полегшення розробки складних веб-додатків із сильним акцентом на продуктивності розробника.

Svelte: Svelte - це новий, швидкий та простий у використанні фреймворк для розробки SPA. Svelte має унікальний підхід до роботи зі статичними компонентами, що дозволяє зменшити розмір коду та підвищити продуктивність додатку. Це робить Svelte привабливим вибором для розробки додатків з обмеженими ресурсами та потужностями розробки SPA, який забезпечує швидку та ефективну розробку веб-додатків.

SvelteJS - досить новий фреймворк, який тільки починає привертати до себе увагу. Він публікувався у State of JavaScript 2018. Svelte має бути фреймворком, який насправді не є фреймворком. В його основі лежить інструмент для компіляції компонентів на етапі збирання, що дозволяє завантажити на сторінку лише необхідне для відображення вашої програми. Це означає, що немає віртуального програмного інтерфейсу.

У кожного з цих фреймворків є свої переваги та недоліки, і вибір конкретного фреймворку залежить від потреб проекту та рівня володіння технологією розробки веб-додатків. Незважаючи на це, SPA-фреймворки є популярними та ефективними інструментами для розробки високоякісних та швидких веб-додатків.

2.1 Порівняння SPA-розробки з традиційними підходами

SPA (Single Page Application) та традиційні підходи до розробки веб-додатків мають свої переваги та недоліки, і вибір конкретного підходу залежить від потреб проекту та його характеристик. Ось деякі переваги та недоліки SPA та традиційних підходів до розробки веб-додатків:

Переваги SPA:

- Швидкість та ефективність: SPA використовують запити для завантаження даних та динамічної зміни вмісту сторінки без перезавантаження сторінки, що дозволяє досягнути високої продуктивності та швидкості роботи додатку.
- Більш сучасний дизайн: SPA дозволяють розробляти веб-додатки з більш сучасним та інтерактивним дизайном, що робить додаток більш привабливим для користувачів.
- Зниження навантаження на сервер: завантаження даних в SPA відбувається асинхронно, що зменшує навантаження на сервер та дозволяє економити ресурси.

Недоліки SPA:

- Поганий пошуковий рейтинг: в зв'язку з тим, що SPA завантажують вміст динамічно, пошукові роботи не можуть індексувати всі сторінки додатку, що може погіршити його пошуковий рейтинг.
- Підвищена складність розробки: SPA мають складну структуру та вимагають багато JavaScript коду, що може зробити розробку більш складною та тривалою.
- Збільшення об'єму завантаження: завантаження великого об'єму JavaScript коду може зменшити швидкість завантаження додатку та спричинити погіршення користувацького досвіду.

Переваги традиційних підходів:

- Кращий пошуковий рейтинг
- Краща підтримка браузерів: традиційні веб-додатки мають кращу підтримку браузерами, оскільки вони не використовують сучасні функції JavaScript, які можуть бути не підтримані старішими браузерами.
- Простота розробки: розробка традиційних веб-додатків є менш складною, оскільки не вимагає великої кількості JavaScript коду та спеціалізованих знань.
- Краща безпека: традиційні веб-додатки мають кращу безпеку, оскільки не використовують сучасні функції JavaScript, які можуть бути вразливими до атак.

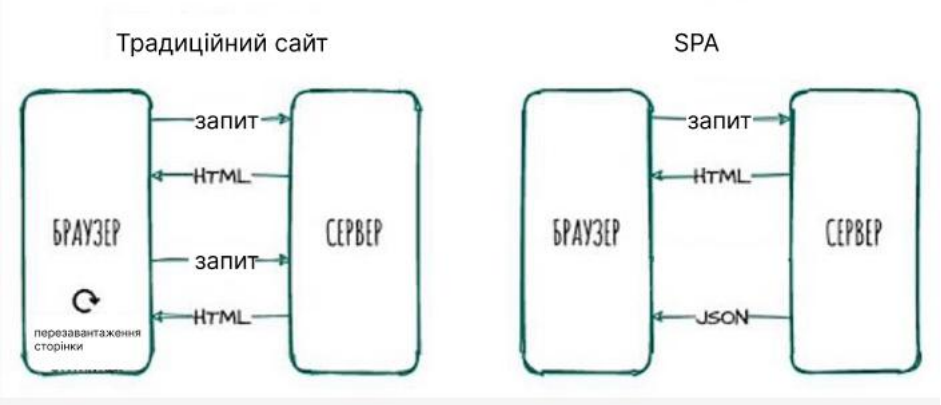
Недоліки традиційних підходів:

- Низька продуктивність: в традиційних веб-додатках при переході між сторінками відбувається повне перезавантаження сторінки, що знижує швидкість та продуктивність додатку.
- Обмежені можливості дизайну: традиційні веб-додатки мають обмежені можливості дизайну, оскільки користувач може бачити тільки те, що знаходиться на сторінці, яку він переглядає.
- Більше навантаження на сервер: у традиційних веб-додатках кожен запит на сервер призводить до повного перезавантаження сторінки, що збільшує навантаження на сервер та може зменшити продуктивність

додатку.

Тому, як працюють звичайні сайти і односторінкові додатки (SPA), можна пояснити наступним чином. Традиційні сайти отримують статичний контент з сервера, після чого користувач може навігувати по різних сторінках, переходячи за посиланнями. Кожен раз, коли користувач переходить на нову сторінку, браузер завантажує цілу сторінку з сервера.

З іншого боку, односторінкові додатки SPA отримують динамічний контент з сервера, використовуючи технології, такі як web socket або AJAX. Замість завантаження нових сторінок, SPA відправляють AJAX-запити на сервер для отримання даних. Це дозволяє взаємодіяти з сервером, залишаючись на одній сторінці. У відповідь на ці запити сервер може повертати дані у форматі JSON, які потім можуть бути використані для оновлення змісту сторінки без необхідності перезавантаження сторінки, як показано на рисунку 2.1.[3]



“Рисунок 2.1 – Різниця між традиційним і SPA сайтами”

Розробка односторінкових програм (SPA) зазвичай включає кілька етапів, зокрема:

Планування та дизайн: на цьому етапі ви визначаєте вимоги до проекту, визначаєте цільову аудиторію та плануєте загальну архітектуру та досвід користувача програми. Це включає в себе розробку макета, каркасну структуру та створення макетів або прототипів.

Front-End розробка: на цьому етапі ви зосереджуєтеся на реалізації інтерфейсу користувача та функціональних можливостей програми. Для створення SPA ви використовуєте фреймворки HTML, CSS і JavaScript, такі як React, Angular або Vue.js. Це включає створення повторно використовуваних компонентів, обробку даних і керування станом, а також інтеграцію з API або серверними службами.

Розробка бекенда: Залежно від складності вашої програми, вам може знадобитися розробити бекенд-сервер для зберігання даних, бізнес-логіки та автентифікації. Цей етап передбачає вибір стека серверних технологій, налаштування серверної інфраструктури та впровадження API для обміну даними з інтерфейсом.

Тестування: Тестування є вирішальним етапом для забезпечення якості та функціональності SPA. Це включає модульне тестування, інтеграційне тестування та наскрізне тестування для перевірки окремих компонентів, їх взаємодії та загальної поведінки програми.Формат SPA найкраще підходить для проектів зі складним інтерфейсом та функціоналом, таких як соціальні мережі, корпоративний софт (CRM, ERP), SaaS-платформи та розділи в багатосторінкових сайтах.[4]

Отже, при виборі між SPA та традиційними підходами до розробки веб-додатків, необхідно враховувати конкретні потреби проекту та його характеристики. SPA найбільш підходять для великих та складних додатків, які потребують високої продуктивності та інтерактивного дизайну, тоді як тради ційні підходи можуть бути кращим вибором для менших та менш складних проектів, які не потребують високої продуктивності та інтерактивності, але можуть забезпечувати кращу підтримку браузерів та більшу безпеку.

Також варто зазначити, що SPA та традиційні підходи можуть бути комбіновані для досягнення найкращих результатів. Наприклад, SPA може використовуватись для частини додатку, яка потребує високої продуктивності та інтерактивності, а традиційні підходи можуть використовуватись для інших

частин додатку.

В кінці кінців, вибір між SPA та традиційними підходами залежить від конкретних потреб проекту та його характеристик, а також від ресурсів та знань, доступних для розробки додатку.

Крім того, SPA має свої особливості, які можуть бути недоліком в деяких випадках. Наприклад, SPA може бути менш доступним для пошукових систем, оскільки весь контент генерується динамічно на клієнтській стороні за допомогою JavaScript. Це може призвести до того, що пошукові системи не зможуть індексувати весь контент додатку, що може погіршити його SEO.

Крім того, SPA може мати проблеми з безпекою, особливо якщо не виконувати належні заходи для запобігання атакам, таким як XSS або CSRF. Загалом, SPA має високі вимоги до знань та досвіду розробників, що може зробити його менш доступним для початківців.

Однак, SPA має багато переваг, особливо для великих та складних проектів. Він може забезпечити більш високу продуктивність та інтерактивність, зменшити навантаження на сервер та зменшити кількість запитів між клієнтом та сервером. Крім того, SPA може забезпечити більш гладку та приємну для користувачів взаємодію з додатком, що може поліпшити користувацький досвід.

У кінці кінців, вибір між SPA та традиційними підходами залежить від конкретних потреб проекту та його характеристик, а також від ресурсів та знань, доступних для розробки додатку. В будь-якому випадку, це питання потребує обговорення та оцінки всіх переваг та недоліків перед тим, як здійснювати остаточний вибір.

Окрім того, SPA може бути складним в розробці та підтримці через його складну структуру та високий рівень абстракції. Це може вимагати великої кількості коду та спеціальних знань з використання фреймворків та інструментів, таких як React або Angular.

Крім того, SPA може бути менш безпечним через використання JavaScript на клієнтській стороні. Недостатня перевірка вхідних даних та виконання неналежної обробки на клієнтській стороні може призвести до злому безпеки

додатку.

Також, SPA може мати проблеми з оптимізацією для пошукових систем та соціальних мереж. Тому для поліпшення SEO необхідно забезпечити підтримку Server-Side Rendering (SSR) або використовувати інші техніки оптимізації для пошукових систем.

Нарешті, SPA може мати проблеми з підтримкою старих браузерів, оскільки він використовує багато сучасних технологій, таких як JavaScript, CSS3 та HTML5, які не підтримуються старими браузерами.

Узагальнюючи, SPA має свої переваги та недоліки, і вибір між SPA та традиційними підходами залежить від конкретних потреб проекту та його характеристик, а також від ресурсів та знань, доступних для розробки додатку.

2.3 Дослідження найефективніших підходів до розробки SPA-додатків

Розробка SPA-додатків відбувається на більш високому рівні в порівнянні з традиційними багатосторінковими додатками. Це пов'язано з тим, що SPA-додатки використовують JavaScript для завантаження даних, що дозволяє більш динамічно змінювати зміст сторінки без перезавантаження сторінки.

Основні підходи до розробки SPA-додатків:

Фреймворки та бібліотеки JavaScript дозволяють вам швидко розробляти SPA-додатки. React, Angular та Vue.js - найпопулярніші фреймворки, що використовуються для розробки SPA-додатків. Вони забезпечують широкий набір функцій та можливостей, що дозволяє розробникам ефективно працювати зі створенням SPA-додатків.

Статичні сайти - це додатки, які використовують статичні файли HTML, CSS та JavaScript. Зазвичай, ці файли знаходяться на сервері та віддаються клієнту як вони є. Найбільш популярний генератор статичних сайтів - Gatsby.js. Це дозволяє створювати SPA-додатки на основі статичних сайтів.

SPA-додатки можуть бути розроблені безпосередньо з використанням JavaScript та API. Це означає, що розробник створює всі функції та методи з нуля,

які забезпечують обмін даними між сервером та клієнтом.

Серверні фреймворки, такі як Node.js, можуть бути використані для розробки SPA-додатків. Ці фреймворки дозволяють розробникам розробляти бекен

Серверні фреймворки, такі як Node.js, можуть бути використані для розробки SPA-додатків. Ці фреймворки дозволяють розробникам розробляти бекенд частину додатка, яка забезпечує взаємодію з базою даних та API.

Існують різні схеми генерації SPA-додатків, які можуть допомогти розробникам створити SPA-додатки більш ефективно та швидко. Наприклад, Next.js - це фреймворк, який дозволяє розробникам створювати універсальні SPA-додатки з підтримкою рендерингу на сервері та на клієнті.

Узагальнюючи, найефективніші підходи до розробки SPA-додатків включають в себе використання фреймворків та бібліотек JavaScript, статичних сайтів, безпосереднього використання JavaScript та API, серверних фреймворків та схем генерації SPA-додатків. Вибір підходу залежить від потреб проекту та навичок розробника.

Зокрема, при розробці SPA-додатків також важливо дотримуватися певних принципів та підходів, які допоможуть підтримувати якість та розширюваність додатку:

- Компонентний підхід - SPA-додатки мають складну структуру, тому важливо використовувати компоненти для створення ієрархії елементів із застосуванням найкращих практик.
- Розділення логіки на клієнті та на сервері - SPA-додатки мають високу залежність від клієнтської сторони, тому важливо забезпечити належний рівень захисту даних та оптимізувати швидкість завантаження додатку.
- Використання тестування - тестування допомагає підтримувати якість та стабільність додатку, дозволяє зменшити ризик появи помилок під час розробки та зберігання якості продукту в майбутньому.
- Оптимізація завантаження додатку - SPA-додатки мають велику кількість JavaScript коду, тому важливо забезпечити оптимізацію швидкості завантаження та прискорити роботу додатку.

- Контроль якості - після завершення розробки важливо перевірити якість та стабільність додатку, переконатися, що він коректно функціонує на різних пристроях та браузерах.

Узагальнюючи, при розробці SPA-додатків важливо враховувати багато чинників, таких як вибір технологій, архітектура додатку, оптимізація процесу розробки та тестування. Також важливо слідкувати за актуальністю технологій та інструментів, що використовуються при розробці SPA-додатків, оскільки цей ринок постійно змінюється і розвивається.

Для розробки SPA-додатків часто використовуються такі технології та інструменти, як Angular, React, Vue.js, Webpack, Babel, Redux, MobX, TypeScript та інші. Кожна з цих технологій має свої переваги та недоліки, тому вибір конкретної технології залежить від потреб проекту та команди розробників.

Окрім технологій, важливим елементом розробки SPA-додатків є використання стандартів та протоколів, таких як HTTP, REST API, WebSockets, JSON, GraphQL та інші. Ці стандарти допомагають забезпечити зручний та безпечний обмін даними між клієнтом та сервером.

Також важливо пам'ятати про безпеку при розробці SPA-додатків. Використання SSL-шифрування, захисту від CSRF-атак, коректне зберігання даних на клієнтській стороні, перевірка вхідних даних на сервері та інші заходи можуть допомогти забезпечити безпеку додатку та його користувачів.

Узагальнюючи, розробка SPA-додатків є складним та багатогранним процесом, який потребує від розробників багато знань та навичок. Однак, застосування вірних підходів та технологій може допомогти створити високоякісний та функціональний додаток, який буде корисним та зручним для користувачів.

РОЗДІЛ 3

Typescript як інструмент для створення веб-додатку

3.1 Особливості Typescript

TypeScript - це мова програмування для розробки сучасних веб-застосунків, що розширює можливості вже традиційного JavaScript.

Відмінності TypeScript від JavaScript:

JavaScript створено для написання невеликих сценаріїв веб-сторінок. Чим більший додаток, тим громіздкішим стає код на JS.

Тому було створено TypeScript, який полегшує розробку широкомасштабних додатків.

Через несувору типізацію JavaScript багато помилок видно тільки під час виконання. Це незручно і, зрештою, неприємно. Доводиться витрачати багато часу на пошук таких помилок.

TypeScript усуває цей недолік: написаний на ньому сценарій не зможе випадково викликати змінну як функцію або вимагати значення поля, якщо воно не визначено в об'єкті. Ви побачите превентивне повідомлення про помилку вже під час набору коду.

Деякі з особливостей TypeScript:

- Статична типізація: TypeScript дозволяє заздалегідь визначити типи змінних та параметрів функцій, що дозволяє забезпечити більшу надійність та безпеку у програмі.
- Класи: TypeScript підтримує класи, що дозволяє зручно працювати з об'єктами та їх методами.
- Інтерфейси: TypeScript дозволяє визначати інтерфейси, що визначають типи даних та методів для об'єктів та класів.
- Модулі: TypeScript підтримує модулі, що дозволяє організувати код у

логічні блоки та забезпечує більшу гнучкість у структурі програми.

- Розширення типів: TypeScript дозволяє розширювати типи даних за допомогою інтерфейсів та класів, що дозволяє створювати більш складні типи даних.
- Підтримка ES6 та вище: TypeScript підтримує сучасні функції та функціональність, такі як стрілкові функції, розпакування та розширення об'єктів, асинхронний код та інші.
- Налагодження: TypeScript має підтримку налагодження коду, що дозволяє розробникам швидко виявляти та виправляти помилки.
- Підтримка VS Code: TypeScript має високу інтеграцію з редактором коду Visual Studio Code, що дозволяє розробникам швидко та зручно писати та налагоджувати код.
- Компіляція: TypeScript потребує компіляції у JavaScript, що дозволяє забезпечити оптимізацію та підтримку більш широкого діапазону браузерів та середовищ виконання.
- Наявність бібліотек та фреймворків: TypeScript має багато бібліотек та фреймворків, таких як Angular, React та Vue, що дозволяє розробникам швидко створювати веб-додатки.

TypeScript може бути використаний для розробки веб-додатків на будь-якому фреймворку або бібліотеці JavaScript, але варто звернути увагу на фреймворки, що вже мають підтримку TypeScript, такі як Angular та React.

Для розробки веб-додатків на Angular з TypeScript можна використовувати Angular CLI, що дозволяє швидко та зручно створювати нові компоненти, сервіси, директиви та інші елементи додатку. Angular також має велику кількість бібліотек та функцій, що дозволяє розробникам швидко створювати високоякісні веб-додатки.

Використання TypeScript може позитивно вплинути на продуктивність, надійність та безпеку програмного забезпечення. Деякі можливі позитивні впливи використання TypeScript на програмне забезпечення можуть включати наступне:

- Помилки під час розробки: TypeScript забезпечує статичну типізацію, що

дозволяє виявляти помилки ще на етапі розробки. Це допомагає уникнути масованій кількості помилок, які можуть виникнути під час виконання програмного забезпечення.

- Рефакторинг: TypeScript дозволяє здійснювати більш ефективний рефакторинг коду, що забезпечує більш швидкий та безпечний розвиток програмного забезпечення.
- Продуктивність: TypeScript може покращити продуктивність програмного забезпечення завдяки підказкам та автодоповненню коду.
- Надійність: TypeScript дозволяє забезпечити більшу надійність програмного забезпечення завдяки сильній типізації, що дозволяє виявляти помилки та зменшувати кількість вразливостей програмного забезпечення.
- Безпека: TypeScript дозволяє забезпечити більшу безпеку програмного забезпечення, оскільки він дозволяє використовувати сильну типізацію та забезпечує перевірку типів на етапі компіляції. Це допомагає зменшити кількість вразливостей та підвищити рівень безпеки програмного забезпечення.

Отже, використання TypeScript може допомогти забезпечити більшу продуктивність, надійність та безпеку програмного забезпечення. Таким чином, вивченя TypeScript може бути корисним для розробки великих та складних веб-додатків, де надійність та безпека дуже важливі. TypeScript забезпечує зручний та ефективний розвиток програмного забезпечення завдяки підказкам та автодоповненню коду, а також зменшує кількість помилок та покращує рефакторинг коду.

Для великих проектів зі складними структурами можна використовувати модулі, щоб зменшити залежності та зробити код більш структурованим. TypeScript також підтримує декілька стандартів JavaScript, що забезпечує його сумісність з більшістю існуючих бібліотек та фреймворків.

Крім того, TypeScript дозволяє зменшити кількість помилок, які можуть виникати в програмному забезпеченні, завдяки статичній типізації та перевірці

типів на етапі компіляції. Це може допомогти зменшити кількість вразливостей та підвищити рівень безпеки програмного забезпечення.

Загалом, використання TypeScript може позитивно вплинути на розвиток веб-додатків, забезпечуючи більшу надійність та безпеку програмного забезпечення, а також зменшуючи кількість помилок та полегшуючи розробку та рефакторинг коду.

3.2 Плюси та мінуси Typescript в SPA-розробці

TypeScript може бути корисним для розробки Single Page Applications, оскільки цей тип додатків зазвичай складніший та потребує більшої уваги до деталей. Ось декілька плюсів та мінусів використання TypeScript для розробки SPA:

Плюси:

- Надійність та безпека: TypeScript забезпечує статичну типізацію, що допомагає зменшити кількість помилок на етапі компіляції та забезпечує відслідковування типів даних під час розробки, що зменшує ризик виникнення вразливостей та помилок у програмному забезпеченні.
- Легша підтримка: TypeScript допомагає зробити код більш читабельним та структурованим, що полегшує його підтримку та розширення в майбутньому.
- Сумісність з JavaScript: TypeScript підтримує стандарти JavaScript, що дозволяє використовувати більшість існуючих бібліотек та фреймворків для SPA-розробки.
- Підвищення продуктивності: TypeScript зменшує кількість помилок, що може зекономити час на пошуки та виправлення помилок. Крім того, IDE може надавати підказки та автодоповнення коду, що допомагає збільшити продуктивність розробника.

Мінуси:

- Витрати на навчання: TypeScript має власні правила та особливості, які

можуть вимагати додаткового часу на навчання, якщо ви ще не працювали з TypeScript раніше.

- Додаткові витрати на розробку: Використання TypeScript може збільшити час на розробку, оскільки потрібно враховувати додаткові правила та вимоги під час написання коду.
- Необов'язковість: TypeScript є додатковим шаром над JavaScript, і він не є обов'язковим для розробки SPA. Деякі розробники можуть вважати, що використання TypeScript займає більше часу, ніж він вартий, тому вони можуть відмовитися від його використання. Крім того, TypeScript може стати непотрібним, якщо проект дуже малий або не вимагає великої кількості коду.
- Необхідність перетворення коду: TypeScript потребує компіляції в JavaScript, що може займати час на стадії розробки та тестування.

В цілому, використання TypeScript може бути корисним для розробки SPA, оскільки він забезпечує надійність та безпеку коду, сприяє підвищенню продуктивності та полегшує його підтримку. Однак, його використання може потребувати додаткового часу на навчання та розробку, тому варто ретельно розглянути його застосування у конкретному проекті.

Додатковими плюсами використання TypeScript в SPA-розробці можуть бути:

- Підтримка IDE: TypeScript має вбудовану підтримку в більшості популярних редакторів коду, таких як Visual Studio Code, WebStorm та інші. Це забезпечує зручну розробку та налагодження коду, що сприяє підвищенню продуктивності.
- Зручність роботи з бібліотеками та фреймворками: TypeScript дозволяє розробникам зручно працювати з бібліотеками та фреймворками, оскільки він забезпечує перевірку типів та підказки для їхніх методів та властивостей.

Незважаючи на ці плюси, використання TypeScript може мати деякі недоліки, зокрема:

- Необхідність навчання: Розробники, які не мають досвіду використання

TypeScript, можуть витрачати час на навчання цього інструменту. Це може стати проблемою для невеликих проектів або команд з обмеженим бюджетом.

- Збільшення обсягу коду: TypeScript може збільшити обсяг коду, оскільки потрібно визначити типи для кожної змінної та функції. Це може призвести до збільшення часу розробки та об'єму коду.

У загальному, використання TypeScript у SPA-розробці може забезпечити позитивний вплив на надійність, безпеку та продуктивність програмного забезпечення, однак, його використання вимагає додаткової роботи та навчання.

3.3 Кращі практики використання TypeScript в Spa розробці

TypeScript, як популярна відкрита мова програмування, є ідеальним вибором для сучасної розробки. Завдяки системі типів допомагає писати надійний код, який легше обслуговувати і масштабувати. Однак, для повного використання потенціалу цієї мови та створення високоякісних проектів, важливо ознайомитись з кращими практиками та дотримуватись їх.

У цьому списку представлено пару варіантів найкращих практик, які допоможуть вдосконалити вашу роботу з TypeScript. Ці практики охоплюють різноманітні теми і супроводжуються конкретними прикладами застосування у створенні реальних додатків.

Використання суворої перевірки типів дозволяє гарантувати, що всі змінні використовують задані типи. Наприклад, при оголошенні змінної з типом string у TypeScript буде перевірено, чи передане значення є дійсним рядком, а не числом. Це допомагає виявляти помилки на ранніх етапах розробки та забезпечувати коректну роботу коду. Режим суворої перевірки типів можна активувати, встановивши параметр "strict". Це дозволить TypeScript виконувати додаткові перевірки, які виявляють певні помилки, які інакше можуть залишитися непоміченими.

Також варто використовувати анотації типів для змінних та функцій у

TypeScript. Це дозволяє задати типи явно і забезпечити правильну типізацію та підказки для коду. TypeScript базується на явному вказанні типів, що означає, що вам потрібно вказувати типи всюди. Проте TypeScript також має функцію виведення типів, яка дозволяє компілятору автоматично визначити тип змінної на підставі її значення. Це зменшує необхідність в явному вказуванні типів при кожному оголошенні змінної. Замість цього компілятор аналізує присвоєне значення та встановлює відповідний тип. властивості та методи, якими об'єкт даного типу повинен володіти. Інтерфе

Використовуємо Generics для повторного використання коду: Generics дозволяють забезпечити повторне використання коду для різних типів даних. Використовуйте Generics, коли вам потрібно створити код, який можна використовувати з різними типами.

Використовуємо наслідування та інтерфейси: TypeScript надає можливість використовувати наслідування та інтерфейси для створення складніших структур даних та ієрархій компонентів. Використовуйте ці можливості, коли вам потрібно створити більш складний код. Інтерфейс у TypeScript визначає форму об'єкта, вказуючи його властивості та методи працювати з будь-яким типом данихякими об'єкт володіє. Інтерфейс також може використовуватися як тип для змінних. Це означає, що при присвоєнні об'єкту змінній з типом інтерфейсу, TypeScript перевірить, щоб цей об'єкт мав всі вказані у даному інтерфейсі властивості та методи.

Використовуємо декоратори: Декоратори дозволяють додавати метадані до класів, методів та властивостей. Використовуйте декоратори, коли вам потрібні додаткові можливості для ваших компонентів та сервісів.[5]

Використовуємо декларації типів для зовнішніх бібліотек: Якщо ви використовуєте зовнішні бібліотеки, створіть декларації типів для цих бібліотек. Це забезпечить правильну типізацію

Використовуємо інструменти аналізу коду: TypeScript дозволяє використовувати інструменти аналізу коду, такі як ESLint та TSLint, для забезпечення якості вашого коду та виявлення можливих проблем.

Використовуємо типи для API-запитів: Використовуйте типи, щоб описати структуру відповіді API та запитів. Це допоможе забезпечити правильну типізацію та підказки для вашого коду, коли ви працюєте з API.

Ці кращі практики допоможуть вам забезпечити максимальну безпеку та ефективність вашого коду, коли ви використовуєте TypeScript в SPA-розробці.

Використовуємо інтерфейси та типи для валідації вхідних даних: Використовуйте інтерфейси та типи для опису вхідних даних в компонентах SPA та для їх валідації. Це допоможе запобігти помилкам, пов'язаним з неправильним типом даних, та забезпечить чистоту та організованість вашого коду.

Використовуємо generics: Використання generics дозволяє створювати загальноприйняті функції та класи, які можуть працювати з різними типами вони часто йдуть рука об руку. Це допомагає зменшити дублювання коду та забезпечити більшу повторність та модульність.

Використовуємо декоратори: TypeScript дозволяє використовувати декоратори для зміни поведінки класів та їх методів. Використовуйте декоратори, щоб додавати нові функціональні можливості до вашого коду та робити його більш модульним та організованим.

Використовуємо імпорти з кореневої директорії: Використовуйте імпорти з кореневої директорії, щоб забезпечити більшу структуру та організованість вашого коду. Це допоможе зменшити дублювання коду та підвищити повторність.

Використовуємо асинхронні функції: TypeScript дозволяє використовувати асинхронні функції, що забезпечує більш ефективне та швидке виконання SPA. Використовуйте асинхронні функції, щоб зменшити затримки та збільшити продуктивність вашого коду.

Ці кращі практики допоможуть вам створити якісний та ефективний код при використанні TypeScript в SPA-розробці. Наступний крок - впровадження цих практик в ваш проект та роботя з деталями. Декілька порад для

ефективного використання TypeScript в SPA-розробці:

Увімкнути строгий режим: у TypeScript є строгий режим, який забезпечує суворіші правила перевірки типів. Увімкніть його у файлі tsconfig.json, встановивши "strict": true. Це допомагає вчасно виявляти потенційні помилки та сприяє безпечнішому кодуванню.

Використовуємо статичну типізацію: скористайтеся перевагами функцій статичної типізації TypeScript, щоб надати анотації типів для змінних, параметрів функцій і значень, що повертаються. Це допомагає покращити читабельність коду, зручність обслуговування та виявляє проблеми, пов’язані з типами під час розробки.

Використовуємо інтерфейси та псевдоніми типів: визначте інтерфейси або псевдоніми типів, щоб створити чіткі контракти для ваших структур даних. Це допомагає задокументувати форму ваших даних і забезпечує кращу навігацію кодом і автозавершення в сучасних IDE.

Використовуємо універсали: Універсали у TypeScript дозволяють писати багаторазовий і безпечний код. Вони забезпечують гнучкість при роботі з різними типами даних, колекціями та загальними алгоритмами. Розгляньте можливість використання генериків під час розробки службових функцій або компонентів, які можуть обробляти різні типи даних.

РОЗДІЛ 4

Особливості авторизації користувачів у веб-додатках фітнес-сфери

4.1 Загальний огляд реєстрації та авторизації

Реєстрація та авторизація є основними процесами, які використовуються в різних системах і платформах для ідентифікації та автентифікації користувачів. Хоча вони служать різним цілям, вони часто йдуть разом, щоб забезпечити безпеку та контроль доступу. Ось загальний огляд реєстрації та авторизації:

Реєстрація: Реєстрація — це початковий крок, на якому користувачі надають свою особисту інформацію для створення облікового запису або профілю в системі. Процес реєстрації зазвичай включає наступні кроки: Введення користувача: користувачі надають бажане ім’я користувача, адресу електронної пошти та іншу відповідну інформацію, яку вимагає система. Перевірка: система перевіряє введені дані на повноту, унікальність і дотримання попередньо визначених правил (наприклад, дійсний формат електронної пошти, складність пароля).

Створення облікового запису: коли введені дані проходять перевірку, система створює унікальний обліковий запис користувача та зберігає надану інформацію в базі даних. Перевірка: деякі системи можуть вимагати від користувачів підтвердити свою електронну адресу абономер телефону, щоб переконатися, що надані контактні дані дійсні. Умови: користувачам може знадобитися прийняти умови надання послуг або інші угоди перед завершенням процесу реєстрації.

Авторизація - це процес обмеження або надання доступу до ресурсів або

функцій на підставі ідентифікаційної інформації та дозволів користувача. Це забезпечує, що користувачі можуть виконувати тільки дозволені дії або отримувати доступ до відповідної інформації. Включає автентифікацію користувача, перевірку особи та контроль доступу на основі ролей (RBAC).):[6]

RBAC — це поширений підхід, коли користувачам призначаються ролі (наприклад, адміністратор, модератор, користувач), а дозволи пов’язуються з цими ролями. Це спрощує керування доступом, надаючи дозволи на основі ролей, а не окремо для кожного користувача. І реєстрація, і авторизація відіграють вирішальну роль у забезпеченні безпеки системи та конфіденційності користувачів.

Реєстрація встановлює унікальний ідентифікатор користувача, а авторизація контролює доступ до ресурсів і дії на основі привілеїв цього ідентифікатора. Ці процеси використовуються в різних контекстах, включаючи веб-сайти, програми, онлайн-сервіси та захищені системи.

Загальний підхід до реєстрації та аутентифікації. Для цілей транзакцій електронного уряду цей документ визначає рівні реєстрації та автентифікації, які підходять для різних класів транзакцій. Загалом неофіційні або транзакції меншої вартості притягнуть нижчі рівні реєстрації та автентифікації. Більше значення або юридично значимі транзакції вимагатимуть більш суворих вимог до реєстрації та автентифікації.

Слід зазначити, що для даної транзакції реєстрація та автентифікація можуть не мати однаковим акцентом і, отже, залучить різні рівні (тобто реєстрація рівня 1 не обов’язково передбачають вимогу до рівня автентифікації 1 і так далі). Як приклад, транзакція, така як псевдонімний доступ до медичного тестування потребує неоднакових рівнів реєстрації та автентифікації

оскільки справжня ідентифікація не потрібна, але потрібна надійна автентифікація, щоб гарантувати результати розкриваються лишеклієнт має правильну електронну ідентифікацію.

Відділи повинні призначити кожній електронній службі рівень реєстрації та автентифікації відповідно до вказівок, що містяться в цій структурі.

Для кожного рівня реєстрації уряд визначає профіль (набір вимог), що визначає механізм для досягнення необхідного ступеня впевненості в ідентичності реального світу (яка може бути у формі певної ролі, а не особистої ідентичності) і авторитет клієнта. Окремий профілі будуть визначені для бізнесу та громадян.

Подібним чином визначаються профілі для рівнів автентифікації, щоб встановити механізми досягнення правильний ступінь впевненості в електронній ідентичності клієнта.

Визнається, що це інфраструктура відкритих ключів (PKI), програми з підтримкою сертифікатів або маркери доступу (наприклад, смарт-картки) можуть бути недоступні з самого початку. У цьому випадку можуть бути інші механізми бути реалізовано спочатку, з наміромприйняти механізми PKI у належний час.

4.2 Переваги та недоліки різних видів авторизації

Різні типи механізмів авторизації пропонують різноманітні переваги та недоліки залежно від їх реалізації та контексту використання. Ось декілька поширених типів авторизації та пов’язані з ними плюси та мінуси: RBAC - це система управління доступом, яка заснована на ролях користувачів і визначає, які дії або ресурси їм доступні на основі призначених ролей.

Переваги:

- Простота: полегшує контроль доступу, групуючи користувачів. Це зменшує складність керування дозволами окремих користувачів.
- Масштабованість: він є масштабованим, оскільки дозволяє легко додавати або видаляти користувачів і ролі, не впливаючи на загальну структуру контролю доступу.
- Ефективне адміністрування: RBAC забезпечує централізоване адміністрування та спрощує процес надання та скасування дозволів.

Недоліки:

- Відсутність деталізації: RBAC може не мати детального керування,

оскільки дозволи призначаються на рівні ролі. Це може не дозволяти сценарії керування доступом, де потрібно керувати окремими дозволами.

- Рольовий вибух: У великих системах складні вимоги, кількість ролей може збільшуватися, що призведе до вибуху ролей і збільшення адміністративних витрат.
- Обмежена гнучкість: RBAC може погано обробляти динамічні або залежні від контексту вимоги контролю доступу, оскільки дозволи є статичними та прив’язані до ролей.

Основні правила та наданння доступу (RBAC):

Переваги:

- Гнучкість: дозволяє створювати власні правила, які визначають рішення щодо контролю доступу на основі конкретних умов і атрибутів.
- Виразність: системи на основі правил пропонують високий рівень виразності, дозволяючи специфікацію складної логіки авторизації та детальне керування.
- Контекстна обізнаність: системи на основі правил можуть враховувати контекстну інформацію для прийняття рішень щодо контролю доступу, забезпечуючи більш динамічну та адаптовану авторизацію.

Недоліки:

- Складність: системами, заснованими на правилах, може стати складніше керувати зі збільшенням кількості правил,потребує ретельної організації та обслуговування.
- Потенційні конфлікти: зі складними наборами правил можуть виникати конфлікти між правилами, що призведе до ненавмисних результатів контролю доступу.
- Вплив на продуктивність. Оцінка багатьох правил може вплинути на продуктивність системи, особливо під час роботи з великими наборами правил або в сценаріях прийняття рішень у реальному часі.

Варто зазначити, що вибір механізму авторизації залежить від конкретних вимог, складності та потреб безпеки системи чи програми. На практиці можна

використовувати комбінацію різних механізмів авторизації для досягнення бажаного рівня контролю доступу та гнучкості.

4.3 Вимоги до систем авторизації у веб-додатках

Системи авторизації в веб-додатках мають вимоги щодо контролю доступу та безпеки. Основні вимоги включають автентифікацію користувача, яка передбачає перевірку облікових даних, таких як імена користувачів і паролі, для перевірки їхньої ідентичності.

Точне керування доступом: на додаток до RBAC веб-програми часто потребують детального контролю доступу, щоб обмежити або дозволити доступ до певних ресурсів, функцій або дій. Це може включати визначення та керування окремими дозволами, щоб розглянути додаткові атрибути або контекстуальні фактори.

Безпечне зберігання облікових даних користувача: система авторизації повинна забезпечувати безпечне зберігання облікових даних користувача, наприклад паролів. Зазвичай це включає методи хешування та соління для захисту конфіденційної інформації.

Безпечне керування сеансами: система повинна безпечно керувати сеансами користувачів, щоб підтримувати автентифікований стан і запобігати неавторизованому доступу. Це включає такі функції, як закінчення терміну дії сеансу, перевірка сеансу та захист від викрадення сеансу або атак із фіксацією сеансу.

Безпечна передача: веб-додатки повинні використовувати безпечні протоколи зв’язку, такі як HTTPS, для шифрування даних, що передаються між клієнтом і сервером. Це запобігає несанкціонованому перехопленню або підробці конфіденційної інформації під час пересилання.

Аудит і журналювання: система авторизації повинна включати можливості аудиту та журналювання для відстеження та запису доступу користувачів, змін дозволів та інших відповідних дій. Це допомагає звідповідність, усунення

несправностей і аналіз безпеки.

Відкликання та деавторизація: система повинна дозволяти відкликати або вилучати права доступу користувача, коли це необхідно. Це може включати деактивацію облікового запису, відкликання дозволу або обробку змін ролі користувача.

Захищений інтерфейс адміністрування: інтерфейс адміністрування для керування ролями користувачів, дозволами та параметрами контролю доступу повинен мати відповідні заходи безпеки, такі як надійна автентифікація, обмеження доступу та журналювання адміністративних дій.

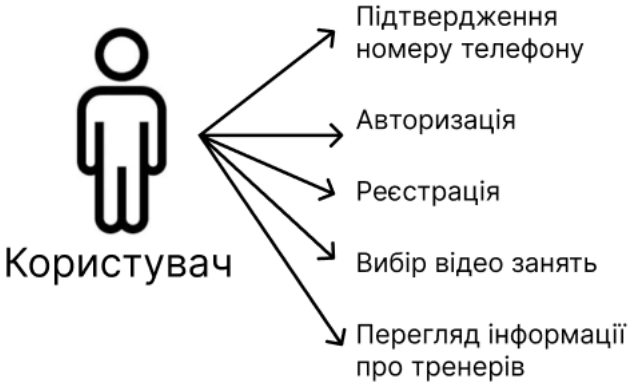
Відповідність нормативним вимогам. Залежно від характеру веб-програми та даних, які вона обробляє, може знадобитися дотримання відповідних норм і стандартів (наприклад, GDPR, HIPAA, PCI DSS). Авторизаціясистема повинна підтримувати впровадження необхідних заходів безпеки та конфіденційності.

Ці вимоги допомагають гарантувати, що веб-програми мають надійні та безпечні системи авторизації, захищають конфіденційні дані, запобігають несанкціонованому доступу та виконують нормативні зобов’язання та зобов’язання щодо відповідності. Реалізація цих вимог часто передбачає поєднання методів безпечного кодування, належної конфігурації та регулярних оцінок безпеки.[7]

РОЗДІЛ 5
Розробка прототипу веб-додатку на прикладі сайту з занять фітнесом з авторизацією користувачів

5.1 Опис вимог до прототипу

Зайшовши на сайт, ми одразу бачимо кнопку реєстрації. Під час переходу на реєстрацію користувач може вибрати один з двох варіантів. Якщо він має акаунт на сайті від проходить етап ауторизації, якщо не має користувач проходить етап реєстрації. KarmaFit повинен задовольняти вимоги які зображенні на рисунку 5.1.



“Рисунок 5.1 – Діаграма варіантів використання”

На рисунку 5.1 ви можете побачити діаграму варіантів використання. На ній зображено можливості реєстрації, авторизації, підтвердження пошти, вибір відео занять та перегляд інформації про тренерів.

Під час реєстрації користувач повинен ввести свої персональні дані такі як: ім’я, прізвище ,номер телефону та пароль.

Авторизація — це процес надання або отримання дозволу чи доступу до певного ресурсу, системи чи служби. Це передбачає перевірку особи та привілеїв фізичної чи юридичної особи, щоб визначити що буде дозволено виконувати користувачеві.

Під час авторизації потрібно ввести логін та пароль. Логіном користувача виступатиме його номер телефону. Пароль повинен скаладатися не менше аніж з вісьми символів які повинні містити як мінімум одну букву та одну цифру, також як мінімум одну букву у верхньому регістрі.

У контексті комп’ютерних систем авторизація є важливою частиною забезпечення безпеки та захисту конфіденційної інформації. Зазвичай це відбувається після автентифікації, яка підтверджує особу користувача за допомогою облікових даних, таких як імена користувачів і паролі. Після автентифікації користувача авторизація визначає, до яких ресурсів або дій він має право отримати доступ або виконувати на основі призначених їм дозволів або ролей.

Авторизація може базуватися на різних факторах, таких як ролі користувачів, списки контролю доступу (ACL) або спеціальні дозволи, призначені окремим користувачам або групам. Ці дозволи можна визначати та керувати ними через централізований сервер авторизації або в самій програмі чи системі.

Загалом авторизація відіграє важливу роль устворенні конфіденційності, цілісності та доступу даних і ресурсів у системі чи службі шляхом контролю доступу та застосування політик безпеки.

Звичайно! Ось деякі додаткові моменти щодо авторизації:

Рівні доступу: авторизація часто передбачає визначення різних рівнів доступу на основі ролі користувача або його обов’язків в організації. Наприклад, адміністратор може мати повний доступ до всіх ресурсів, тоді як звичайний користувач може мати обмежений доступ до певних функцій або даних.

Деталізація: авторизація може бути реалізована з різними рівнями деталізації. Він може бути таким широким, як надання доступу до всієї системи, або настільки детальним, як визначення доступу до окремих елементів даних або дій у програмі.

Примусове виконання: механізми авторизації зазвичай застосовуються системою або самою програмою. Коли користувач намагається отримати доступ, система перевіряє його облікові дані та дозволи на відповідність визначеним правилам авторизації, щоб визначити, чи дозволена запитана дія.

Динамічна авторизація. У деяких випадках рішення щодо авторизації можуть прийматися динамічно на основі мінливих факторів, таких як час доби, місцезнаходження користувача або стан системи. Механізми динамічної авторизації дозволяють контролювати доступ у реальному часі на основі цих контекстних факторів.

Аудит і ведення журналів: системи авторизації часто містять можливості аудиту та журналювання для відстеження спроб доступу та моніторингу дій користувачів. Це допомагає виявити потенційні порушення безпеки, розслідувати несанкціонований доступ і забезпечити відповідальність.

Єдиний вхід (SSO): авторизація тісно пов’язана з концепцією єдиного входу, коли облікові дані користувача перевіряються один раз, а потім ця автентифікація поширюється на інші системи чи програми, що дозволяє користувачеві отримувати доступ до кількох ресурсів без потрібно повторно автентифікувати.

OAuth і OpenID Connect: це широко використовувані структури авторизації для захисту веб-інтерфейсів API і надання делегованого доступу. OAuth дозволяє користувачеві надати сторонньому додатку обмежений доступ до своїх ресурсів, тоді як OpenID Connect забезпечує спосіб автентифікації та отримання ідентифікаційної інформації про користувача.
Ці пункти забезпечують ширше розуміння авторизації та її різних аспектів. Дайте мені знати, якщо ви хочете дізнатися щось конкретне або обговорити далі!

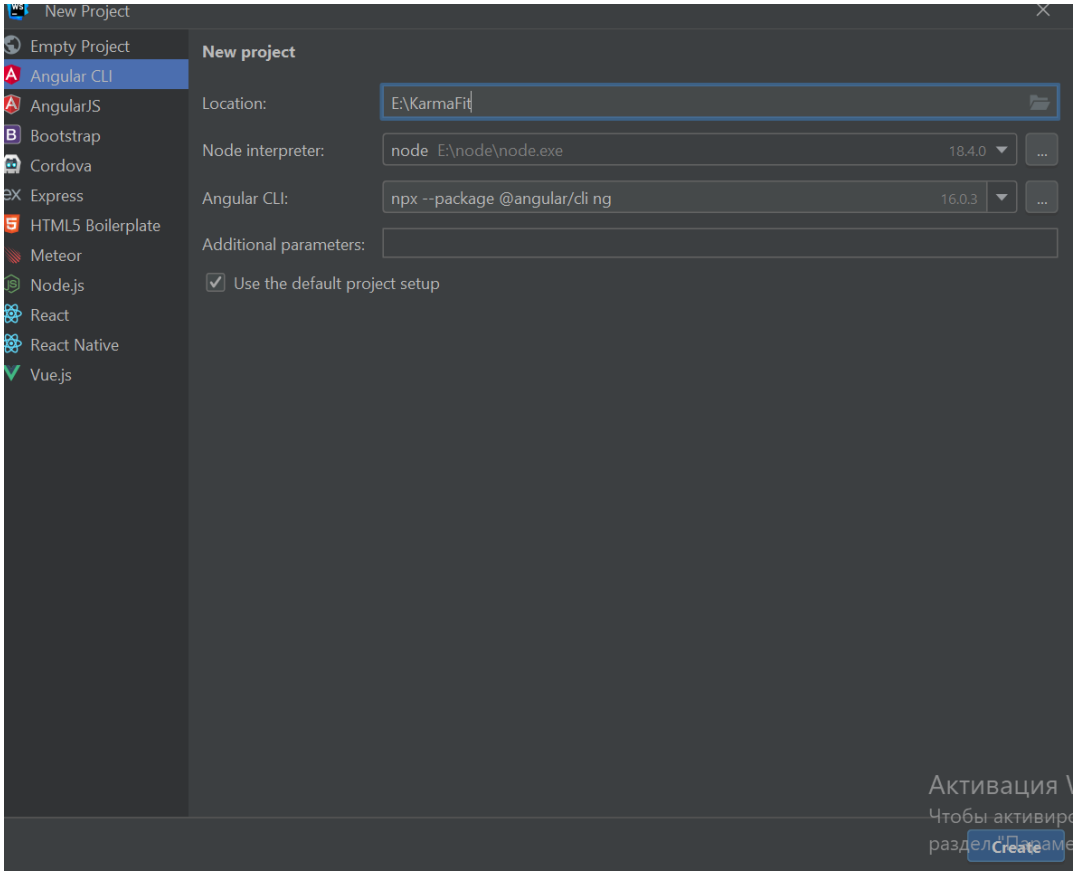
Після активації акаунту кодом який прийшов на номер користувача його

акаунт стає активним. Після чого його перенаправляє на головну сторінку.

Прогортаючи сторінку вниз, користувач може ознайомитись з умовами графіку, повною інформацією про тренерів «KarmaFit», та переглядати відео тренувань які надає сайт.

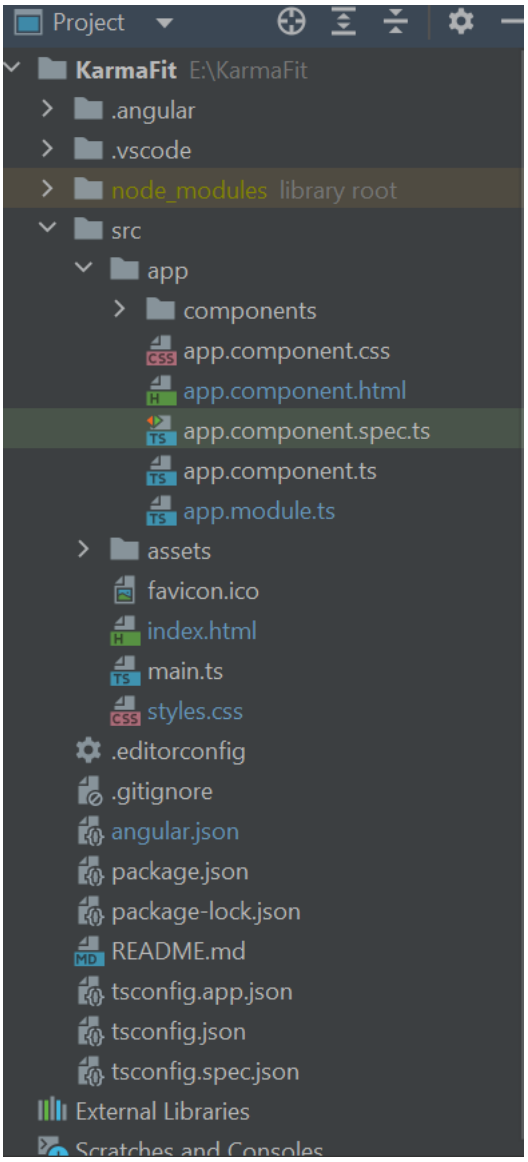
5.2 Реалізація веб-додатку

Створюємо проєкт Angular, та за допомогою обраного редактора коду Web Storm (рисунок 5.2) .



“Рисунок 5.2 – Створення проекту”

На рисунку 5.1 зображено створення проекту Angular. Щоб створити проект необхідно створити його назву. Після чого проєкт та структура його файлів та папок буде створено автоматично. Саму структуру проєкту можна побачити на рисунку 5.3.



“Рисунок 5.3 – Структура проєкту”

На рисунку 5.3 зображено структуру проєкту яку Angular Initializer створив автоматично. Під час налаштування структури проекту Angular існує кілька поширених підходів. Одна широко поширена структура базується на створеному проекті Angular CLI (інтерфейс командного рядка), який має модульну архітектуру. Далі буде описано типову структуру проєкту та деякі потенційні підводні камені, про які слід знати.

Кореневий каталог містить конфігураційні файли, такі як package.json, angular.json і tsconfig.json. Він також містить інші файли, пов’язані з проектом, такі як README.md, .gitignore та .editorconfig. у каталозі src знаходиться більшість коду проекту. Каталог програми містить основні компоненти програми,

модулі, служби та інші файли, що стосуються програми. Каталог активів зберігає статичні файли, які використовуються програмою, наприклад зображення, шрифти та файли конфігурації.

Каталог середовищ містить файли конфігурації для середовища, включаючи environment.ts і environment.prod.ts. Каталог styles містить глобальні таблиці стилів, такі як файли CSS або Sass, які визначають загальний візуальний стиль програми. Файл index.html є точкою входу для програми та містить селектор кореневого компонента. У каталозі app відбувається більша частина роботи з розробки. Зазвичай він складається з таких підкаталогів і файлів. Каталог компонентів містить повторно використовувані компоненти, які використовуються в різних модулях або частинах програми.Каталог modules містить модулі функцій, кожен з яких відповідає за певний набір пов’язаних функцій.

Каталог послуг містить класи послуг, які надають дані або функціональність іншим частинам програми. Каталог models зберігає інтерфейси або класи TypeScript, які визначають структури даних, що використовуються в програмі. Файл app-routing.module.ts визначає конфігурацію маршрутизації на рівні програми. Файл app.component.ts представляє кореневий компонент програми. Файл app.module.ts є точкою входу для програми та визначає кореневий модуль. Файл app.component.html містить шаблон для кореневого компонента.

Файли тестування та конфігурації структура проекту також може включати каталоги, такі як e2e для наскрізних тестів і node_modules для встановлених залежностей. Крім того, можуть існувати такі файли конфігурації, як .eslintrc.json для правил linting або karma.conf.js для тестування конфігурації.

Потенційні підводні камені, про які слід знати під час структурування проекту Angular Надмірне ускладнення структури: дуже важливо знайти баланс між добре організованою структурою та надмірною складністю. Зберігайте структуру простою та модульною, уникаючи непотрібного вкладення каталогів.

Неузгоджені угоди про іменування: Послідовність у угодах про іменування допомагає підтримувати читабельність коду та зменшує плутанину. Визначте

угоду про іменування файлів, каталогів і імен класів і послідовно дотримуйтесь її протягом усього проекту.

Погана організація модулів: уникайте створення монолітних модулів, які обробляють кілька непов’язаних функцій. Натомість націлюйтеся на невеликі цілеспрямовані функціональні модулі з чітко визначеними обов’язками.

Не розділяти проблеми: Забезпечте належний розподіл проблем, організовуючи компоненти, служби та моделі у відповідних каталогах. Це допомагає підтримувати чітку та модульну кодову базу.

Відсутність масштабованості: розгляньте довгострокову масштабованість структури проекту. Плануйте зростання та уникайте тісного зв’язку між компонентами та модулями, що полегшить додавання або зміну функцій пізніше.

Дотримуючись найкращих практик і уникаючи цих пасток, ви можете створити добре структурований проект Angular, який сприяє зручності обслуговування, масштабованості та повторному використанню коду.

5.3 Опис сторінки автентифікації та реєстрації

Автентифікація та реєстрація є важливими компонентами багатьох веб- і мобільних програм. Ось основні причини, чому вони важливі автентифікація та реєстрація допомагають застосувати заходи безпеки в програмі. Вимагаючи від користувачів автентифікації, наприклад, за допомогою імен користувачів і паролів, ви можете контролювати доступ до конфіденційної інформації та захистити дані користувачів від несанкціонованого доступу. Ідентифікація користувача автентифікація дозволяє ідентифікувати та перевіряти особу окремих користувачів. Це вкрай важливо для додатків, які вимагають персоналізованого досвіду, даних або дій користувача. За допомогою автентифікації користувачів ви можете пов’язати їхні дії та дані з їхньою унікальною ідентифікацією, уможливлюючи персоналізацію та налаштування.

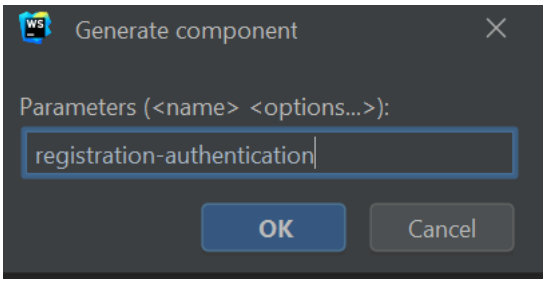
Після автентифікації користувачів ви можете призначати та застосовувати різні рівні авторизації або дозволів на основі їхніх ролей або привілеїв. Це гарантує, що користувачі мають належний доступ до різних функцій, ресурсів або

функцій у програмі. Авторизація допомагає підтримувати конфіденційність даних і запобігає неавторизованим діям або змінам. Реєстрація відіграє вирішальну роль у управлінні користувачами. Це дозволяє новим користувачам створювати облікові записи та отримувати доступ до програми. За допомогою реєстрації користувача ви можете збирати необхідну інформацію, таку як адреси електронної пошти, імена користувачів і паролі, і зберігати їх у безпечному стані.

Керування користувачами також дає змогу скидати пароль, оновлювати та видаляти облікові записи. Автентифікація та реєстрація дозволяють персоналізувати роботу користувачів. Після автентифікації користувачі можуть налаштовувати параметри, уподобання та персоналізувати вміст на основі своїх індивідуальних профілів. Це покращує залучення користувачів, їхнє задоволення та загальну взаємодію з ними.

Автентифікація забезпечує відстежуваний запис дій користувача в програмі. Це забезпечує аудит і підзвітність, дозволяючи відстежувати, хто та коли виконав певні дії. Це особливо важливо для програм, де необхідна відповідність нормативним вимогам або аудит даних.

Щоб створити сторінку аутентифікації та реєстрації спочатк буде створенно компонент який буде містити компоненти форми реєстрації та аутентифікації. Щоб це зробити використаємо вбудований механізм WebStorm приклад чого зображено на рисунку 5.4.



“Рисунок 5.4– Створення компоненту “

На рисунку 5.4 наведено приклад створення компоненту registration-authentication який буде містити форми аутентифікації та реєстрації. В Angular ви можете створити спеціальний компонент під назвою «registration-authentication», щоб

інкапсулювати комбіновані функції реєстрації та автентифікації користувача. Цей компонент забезпечить безперебійну роботу користувача для обох процесів. Ось опис того, як можна структурувати цей настроюваний компонент. Шаблон компонента міститиме необхідні елементи HTML для відображення форм реєстрації та автентифікації та будь-які додаткові елементи інтерфейсу користувача, необхідні для комбінованої функціональності. Приклад шаблону компоненту наведено у лістингу 5.1 додатку А.

Він включатиме елементи керування формами для збору інформації про користувача під час реєстрації, наприклад імені користувача, номер телефону, пароля та будь-яких інших відповідних полів.

Він також включатиме елементи керування формою для процесу входу, такі як ім’я користувача/телефон та пароль. Залежно від ваших конкретних вимог ви можете розробити макет і стиль форм і включити додаткові елементи, як-от прапорці «Запам’ятати мене» або параметри входу через соціальні мережі.

Клас компонентів міститиме логіку та функціональні можливості для процесів реєстрації та автентифікації. Він включатиме такі елементи визначено властивості для зберігання даних форми, повідомлень про помилки, станів завантаження або будь-якої іншої відповідної інформації, необхідної для реєстрації та автентифікації. Реалізовано методи для обробки надсилання форм, зв’язку зі службою автентифікації, виконання перевірки форми та керування потоком користувачів між реєстрацією та автентифікацією. Приклад коду можна побачити у лістингу 5.2 додатку А.

Щоб стилізувати компонент «реєстрація-автентифікація» в Angular, ви можете створити окремий файл стилю, який відповідає компоненту. Ось опис того, як ви можете структурувати та застосовувати стилі до цього спеціального компонента створено файл стилю спеціально для компонента «реєстрація-автентифікація». За угодою ви можете назвати його "registration-authentication.component.scss" або використати іншу умову іменування, яка підходить для вашого проекту. Почато із націлювання на селектори компонентів у файлі стилів. Селектори можуть містити кореневий елемент компонента, імена

класів або селектори атрибутів, пов’язані з елементами компонента. Приклад коду з файлу стилів можна побачити в лістингу 5.3 додатку А.

Створивши спеціальний компонент «реєстрація-автентифікація», ви можете інкапсулювати логіку та користувальницький інтерфейс для обох процесів, забезпечуючи безперебійний досвід для ваших користувачів. Цей підхід допомагає підтримувати багаторазове використання коду, модульність і чіткий розподіл проблем у вашому проекті Angular.

Далі розглянемо реалізацію форми реєстрації. ЇЇ було створено як компонент таким самим шляхом як і вище згаданий компонент. Їх структура ідентична. Приклад файлу який містить шаблон форми наведено в лістингу 5.4 додатку А.

Щоб користувач отримав хороший досвід з нашою користувацькою формою додамо стилі які показані у лістингу 5.5 додатку А. Даному лістингу можна побачити що даний Scss файл посилається на інший. Це файл зі стилями які містить загальні стилі для форми реєстрації та авторизації. Код якого зображено в лістингу 5.6 додатку А.

Щоб надати нашій формі необхідний функціонал нам необхідно реалізувати його за допомогою файлу typescript який відповідає за даний компонент приклад коду наведено в лістингу 5.7 додатку А. В даному класі наведено приклад реактивної форми яка контролює форму наведено в лістингу 5.5 додатку А.

Перейдемо до форми авторизації. ЇЇ також буде створено інструментів WebStorm. Його структура є ідентичною до вище загаданих компонентів. Html шаблон цієї формули наведено у лістингу 5.8 додатку А.

Також щоб стилізувати нашу форму нам необхідно створити файл подібний файлу компоненту згаданого вище. Приклад файлу стилів який відноситься до даного компоненту наведено в лістингу 5.9 додатку А. У даному файлі можна побачити імпортування загального файлу стилів для форм авторизації та реєстрації.

Щоб додати логіку обробки нашої форми нам необхідно реалізувати її у файлі Typescript який відповідає за даний компонент приклад цього коду можна

побачити в лістингу 5.10 додатку А.

5.4 Опис головної сторінки

Головна сторінка, яку часто називають цільовою сторінкою або домашньою сторінкою, є важливим компонентом веб-програми або веб-сайту. Він служить початковою точкою взаємодії для користувачів і відіграє важливу роль у формуванні їх загального досвіду. Ось кілька причин важливості головної сторінки. Головна сторінка – це перше, що бачать користувачі, коли відвідують вашу програму чи веб-сайт. Це створює перше враження та задає тон подальшій взаємодії. Добре розроблена та приваблива головна сторінка може привернути увагу користувачів, викликати інтерес і спонукати їх досліджувати далі. Навігація та інформаційна ієрархія: головна сторінка містить огляд програми або веб-сайту, структуру, вміст і параметри навігації. Зазвичай він містить чітке та інтуїтивно зрозуміле меню або систему навігації, яка допомагає швидко та легко знаходити інформацію чи функції, які вони шукають. Ефективно впорядковуючи та пріоритезуючи інформацію, головна сторінка спрямовує користувачів до відповідних розділів і покращує їхній загальний досвід навігації.

На головній сторінці часто демонструється ідентичність бренду, включаючи логотипи, кольори та візуальні елементи, щоб створити постійну та впізнавану присутність бренду. Це дозволяє користувачам ідентифікувати бренд і підключатися до нього, зміцнюючи довіру та довіру. Добре брендована головна сторінка може передати мету та цінності програми чи веб-сайту, залишаючи незабутнє враження на користувачів.

Головна сторінка може містити ключовий вміст, наприклад рекомендовані продукти, важливі оголошення або виділені статті, щоб привернути увагу користувачів і забезпечити швидкий доступ до відповідної інформації. Він діє як шлюз для демонстрації найціннішого або актуального вмісту, заохочуючи користувачів до подальшої взаємодії та глибшого вивчення програми чи веб-сайту.

Конверсія та заклик до дії: головна сторінка часто містить елементи заклику до дії, такі як форми реєстрації, кнопки завантаження або підказки щодо покупки, щоб стимулювати конверсію користувачів. Він спрямований на те, щоб скеровувати користувачів до певних дій, будь то створення облікового запису, підписка на інформаційний бюлетень або здійснення покупки. Оптимізація макета головної сторінки та дизайну для перетворення може значно вплинути на успіх вашої програми чи веб-сайту. Продуктивність і час завантаження: головна сторінка створює основу для загальної продуктивності вашої програми або веб-сайту. Він має бути оптимізований для швидкого завантаження, забезпечуючи плавну та безперебійну роботу користувача. Повільне завантаження головних сторінок може призвести до розчарування та залишення користувачів, негативно впливаючи на залученість користувачів і коефіцієнт конверсії.

Головна сторінка відіграє вирішальну роль у пошуковій оптимізації. Використовуючи релевантні ключові слова, мета-теги та структурований вміст, головна сторінка може покращити видимість програми або веб-сайту в рейтингах пошукових систем. Це може збільшити органічний трафік і залучити більшу аудиторію до вашої програми або веб-сайту.

Таким чином, головна сторінка життєво важлива для справляння позитивного першого враження, забезпечення ефективної навігації та інформаційної ієрархії, створення бренду та ідентичності, виділення важливого вмісту, стимулювання переходів, забезпечення ефективності та покращення видимості в пошукових системах. Продумано розробляючи та оптимізуючи головну сторінку, ви можете створити привабливу та зручну точку входу для своєї програми чи веб-сайту.

Щоб створити дану частину сайту буде створено новий компонент з назвою content. Даний компонент буде мати ідентичну структуру з поередніми компонентами.

Щоб встановити розмітку сайту додамо необхідний код до html файлу який відповідає за даний компонент. Приклад можна побачити в лістингу 5.11 додаток А. Зі структури даного документу можна зрозуміти що сайт складатиметься з

декількох секцій.

Перша секція міститиме банер з кнопкою реєстрації яка перенапрявлятиме на відповідну форму. Друга секція містить кроки як попасти на заняття з фітнесу. Третя частина інформація та біографія тренерів.

Та четверта секція це меню навігації.

Щоб надати html сторінці приємного для користувача вигляду необхідно додати зміни в scss файлу який відповідає за даний документ та відповідно за html документ. Приклад лістингу можна побачити в 5.12 додатку А.

РОЗДІЛ 6
Безпеки життєдіяльності та основа охорони праці

6.1 Ризики як кількісна оцінка небезпек у досліжені перспектив розробки SPA з використанням TypeScript

Кількісна оцінка ризиків у напрямку дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів може включати наступні аспекти:

- Вимоги до безпеки даних: Цей аспект оцінює ризики пов'язані з безпекою інформації, таких як втрата, викрадення або неправомірний доступ до особистих даних користувачів. Оцінка ризику може враховувати типи даних, які обробляються, рівень чутливості цих даних та заходи безпеки, які будуть реалізовані.
- Недостатня автентифікація та авторизація: Цей ризик оцінюється відносно можливостей зловмисників обійти механізми автентифікації та авторизації, що може призвести до незаконного доступу до функцій та даних сайту.
- Вразливості в програмному забезпеченні: Оцінка ризику пов'язана з можливістю наявності програмних дефектів, вразливостей або слабких місць у коді, які можуть бути використані для атак з боку зловмисників. Це може включати оцінку потенційних загроз, таких як перехоплення даних, введення шкідливого коду або виконання атак типу "cross-site scripting" (XSS) та "cross-site request forgery" (CSRF).
- Недостатня стійкість до навантаження: Цей ризик оцінюється відносно можливості веб-додатку не витримувати великі навантаження або стресові ситуації. Він може включати оцінку ризику втрати продуктивності, аварійного завершення роботи або неадекватної реакції системи на великий потік запитів.
- Сумісність з браузерами та пристроями: Оцінка ризику пов'язана з можливими проблемами, які можуть виникнути в результаті несумісності програмного забезпечення з різними браузерами та пристроями. Це може

включати відсутність підтримки певних функцій, неправильне відображення контенту або проблеми зі стабільністю.

- Використання сторонніх бібліотек та залежностей: Цей аспект оцінює ризики, пов'язані з використанням сторонніх бібліотек та залежностей у розробці програмного забезпечення. Оцінка може включати оцінку надійності та безпеки цих компонентів, а також можливість виникнення конфліктів версій або проблем з управлінням цими залежностями.
- Недостатня тестування та валідація: Оцінка ризику пов'язана з недостатнім тестуванням та валідацією розробленого програмного забезпечення. Це може включати ризик помилок у функціональності, невідповідності до вимог клієнта або проблем з безпекою.
- Обмеження ресурсів: Цей ризик оцінюється відносно обмежень ресурсів, таких як людські, фінансові або часові ресурси, необхідні для розробки та підтримки програмного забезпечення. Оцінка ризику може включати визначення можливостей виконання проекту в умовах обмежених ресурсів та врахування потенційних наслідків недостатку ресурсів.

Важливо врахувати, що оцінка ризиків має бути здійснена в контексті конкретного проекту та може варіюватись в залежності від його обсягу, складності та умов розробки.

Зазначені ризики є лише загальними прикладами і не вичерпують повного переліку потенційних проблем, пов'язаних з розробкою SPA з використанням TypeScript для сайту з фітнесом та авторизацією користувачів. Конкретні ризики можуть варіюватися в залежності від багатьох факторів, таких як обсяг проекту, рівень досвіду команди розробників, вимоги до безпеки та специфіка фітнес-індустрії.[8]

Для виконання кількісної оцінки ризиків, можна використати різні методики, такі як аналіз SWOT (аналіз сильних і слабких сторін, можливостей і загроз), методики аналізу ризиків , або інші стандартні підходи до оцінки ризиків в інженерії програмного забезпечення.

Для зменшення ризиків та забезпечення успішного розвитку проекту,

рекомендується прийняти наступні заходи:

- Визначення чітких вимог до безпеки даних і розробка політик безпеки. Це включає розуміння типів даних, що обробляються, встановлення механізмів шифрування, захисту від несанкціонованого доступу та відповідність вимогам регулювання (якщо є такі).
- Ретельна розробка та тестування механізмів автентифікації та авторизації користувачів. Це включає використання сильних паролів, багаторівневої авторизації, валідацію вхідних даних та запобігання атакам типу "brute force" (грубої сили).
- Використання надійних і протестованих бібліотек та залежностей. Перевірте репутацію та безпеку сторонніх компонентів, переконайтеся, що вони актуальні та використовуються відповідно до кращих практик безпеки.
- Проведення ретельного тестування, включаючи функціональне та безпекове тестування, а також навантажувальне тестування для оцінки продуктивності та стабільності системи.
- Постійний моніторинг та підтримка системи після впровадження. Забезпечення регулярних оновлень, патчів безпеки та відстеження нових загроз, щоб забезпечити безпеку сайту та захист від нових атак.

Важливо враховувати, що оцінка ризиків та прийняття заходів щодо їх зменшення повинні бути постійним процесом протягом усього життєвого циклу проекту.[9]

6.2 Загальні вимоги безпеки з охорони праці для користувачів

Загальні вимоги безпеки з охорони праці для користувачів ПК у контексті дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів можуть включати наступні аспекти:

- Ергономіка робочого місця: Забезпечення комфорту та безпеки користувача під час роботи з ПК, включаючи належне розташування робочого столу, стільця, монітора, клавіатури та миші. Також слід враховувати освітлення приміщення та регулярні перерви для відпочинку.

- Попередження перенапруження: Рекомендації щодо правильної організації робочого часу та уникнення тривалої неперервної роботи перед ПК. Важливо включати регулярні перерви, фізичні вправи, а також рекомендації щодо використання захисних окулярів для запобігання втомі очей.
- Безпека даних та конфіденційність: Забезпечення захисту особистих даних користувачів, включаючи механізми автентифікації та авторизації, шифрування даних, заходи безпеки мережі та захист від несанкціонованого доступу до інформації.
- Захист від шкідливих програм та вірусів: Встановлення та оновлення антивірусного програмного забезпечення, брандмауера та інших засобів захисту, які допоможуть запобігти вторгненням, інфікуванням ПК шкідливими програмами та втратою даних.
- Освіта та навчання: Проведення навчання користувачів щодо безпеки використання ПК, включаючи правила безпеки в мережі, виявлення шкідливих програм, обізнаність про схеми шахрайства та фішингу, та інші аспекти, які можуть вплинути на безпеку та конфіденційність даних.

Ці загальні вимоги безпеки з охорони праці слід адаптувати та дотримуватися згідно специфічних вимог вашого проекту розробки SPA з використанням TypeScript для сайту з фітнесом та авторизацією користувачів.

Для ефективної організації безпеки з охорони праці для користувачів ПК у контексті дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів, рекомендується врахувати наступні аспекти:

- Резервне копіювання та відновлення даних: Регулярне створення резервних копій важливої інформації та наявність процедур відновлення даних у разі випадкового видалення або втрати даних. Додатково, використання системи контролю версій може сприяти збереженню попередніх версій коду та відновленню до попередніх станів.
- Регулярні оновлення та патчі: Правильне та своєчасне оновлення програмного забезпечення, включаючи операційну систему, браузери,

плагіни та інші компоненти, що використовуються на сайті. Це допоможе запобігти використанню вразливостей, які можуть бути використані зловмисниками для вторгнення та компрометації системи.

- Фізична безпека: Забезпечення фізичної безпеки обладнання, на якому працюють користувачі, включаючи захист від крадіжок, випадкового пошкодження та несанкціонованого доступу до ПК та інших пристроїв.
- Захист від втрати даних: Регулярне резервне копіювання даних на зовнішні носії або хмарні сервіси, щоб запобігти втраті інформації у разі несправності апаратного забезпечення або небажаної події.
- Правила використання: Розробка та впровадження правил використання ПК для користувачів, включаючи вимоги до паролів, обмеження доступу до недозволених джерел та ресурсів, заборону використання приватного програмного забезпечення та регулярну навчання користувачів щодо правил безпеки та розумного використання ПК.

Загальні вимоги безпеки з охорони праці для користувачів ПК слід враховувати на всіх етапах розробки SPA сайту для занять фітнесом з авторизацією користувачів, починаючи від проектування та розробки до експлуатації та підтримки. Запровадження цих вимог допоможе забезпечити безпеку та конфіденційність інформації, а також знизити ризик вразливостей та атак на систему.[10]

ВИСНОВКИ

У роботі було проведено дослідження перспектив розробки односторінкового сайту (SPA) та порівняно з традиційними методами розробки веб-додатку з використанням TypeScript на прикладі створення сайту для занять фітнесом з авторизацією користувачів.

Застосування SPA підходу дозволило досягти кращого користувацького досвіду шляхом зниження часу перезавантаження сторінок та поліпшення швидкодії за рахунок динамічного завантаження контенту. Користувачі отримали плавну та реактивну взаємодію зі сторінкою, що значно покращило їхнє сприйняття та задоволення від використання сайту для занять фітнесом.

Використання TypeScript як мови програмування забезпечило перевагу типізації, що сприяло виявленню помилок на ранніх етапах розробки, зрозумілому та структурованому коду, а також полегшенню процесу розробки та підтримки проекту. TypeScript дозволив покращити надійність програмного продукту та зменшити ймовірність виникнення помилок.

Результати дослідження підтвердили, що розробка SPA з використанням TypeScript є перспективним напрямком для створення сучасних та функціональних веб-додатків. Отримані переваги у вигляді поліпшеного користувацького досвіду, кращої швидкодії, зручної розробки та підтримки дозволяють рекомендувати використання SPA з TypeScript для розробки сайтів з авторизацією користувачів у сфері фітнесу та інших подібних галузях.

Результати роботи можуть надати продовження як підґрунтя для подальшого розширення функціональності сайту, оптимізації продукту та вдосконалення його архітектури. Додатково, розглянуті аспекти SPA та TypeScript можуть бути застосовані в інших проектах, що спрямовані на розробку веб-додатків з високими вимогами до швидкодії та користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. REACT.JS. Створення і розробка інтернет-магазину Харків, Київ, створення і розробка інтернет магазинів під ключ - Brander. URL: https://brander.ua/technologies/reactjs (date of access: 09.06.2023).
2. Mell, O. (2021, September 14). Огляд фреймворків JavaScript. Що, для чого і коли використовувати. DOU. https://dou.ua/forums/topic/34739/
3. SPA в програмуванні: розбираємося з одностранічніками. FoxmindEd. URL: https://foxminded.ua/spa-u-prohramuvanni/ (дата звернення: 09.06.2023).
4. Розробка односторінкових додатків SPA | WebCase. Webcase. URL: https://webcase.com.ua/uk/blog/razrabotka-odnostranichnyh-prilozhenij-spa-webcase/#id6 (дата звернення: 09.06.2023).
5. TypeScript, найкращі практики при написанні коду. MarkupUA. URL: https://markup-ua.com/typescript-najkrashhi-praktiki-pri-napisanni-kodu/ (дата звернення: 09.06.2023).
6. Аутентифікація і авторизація: що це і в чому відмінність. QualityAssuranceGroup. URL: https://qagroup.com.ua/publications/autentyfikatciia-i-avtoryzatciia/ (дата звернення: 09.06.2023).
7. Прототипування сайту - brainlab. brainlab. URL: https://brainlab.com.ua/uk/blog-uk/prototipuvannya-sajtu (дата звернення: 09.06.2023).
8. Ризик як оцінка небезпеки. Прогнозування небезпек та захист від їхньої дії. - Studies. Studies. URL: https://studies.in.ua/bjd-atamachuk/1019-133-rizik-yak-ocnka-nebezpeki-prognozuvannya-nebezpek-ta-zahist-vd-yihnoyi-dyi.html
9. Ризик як кількісна оцінка небезпек. Освіта та самоосвіта. URL: https://referatss.com.ua/work/rizik-jak-kilkisna-ocinka-nebezpek/
10. Інструкція з охорони праці при роботі з комп'ютером Інструкції для навчальних закладів України | Інструкції з охорони праці, техніки безпеки і пожежної безпеки. URL: https://osvita-docs.com/node/41

ДОДАТОК А

Лістинг 5.1

```
<h1>Registration</h1>

<form [formGroup]="form" class="authorization_form">

  <input formControlName="firstName" class="form_input" type="text"
placeholder="First Name">
  <input formControlName="lastName" class="form_input" type="text"
placeholder="Last Name">
  <input formControlName="phone" class="form_input" type="phone "
placeholder="Phone">
  <input formControlName="password" class="form_input" type="password"
placeholder="Password">

  <button [disabled]="!form.valid"
          class="form_button">Register</button>
  <a class="form_link" routerLink="/authorization/authentication">Do you
have an account?</a>
</form>
```

Лістинг 5.2

```
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";

@Component({
  selector: 'app-authentication',
  templateUrl: './authentication.component.html',
  styleUrls: ['./authentication.component.scss']
})
export class AuthenticationComponent implements OnInit {

  form!: FormGroup;
```

```
  constructor() {

    this.form = new FormGroup({
      phone: new FormControl('', [Validators.required]),
      password: new FormControl('', [Validators.required]),
    });
  }

  ngOnInit(): void {

  }

}
```

Лістинг 5.3

```
@import      'src/app/components/registration-authentication/authorization-general.scss';

:host {
  display: flex;
  flex-direction: column;
}
```

Лістинг 5.4

```
<h1>Registration</h1>

<form [formGroup]="form" class="authorization_form">

  <input    formControlName="firstName"    class="form_input"    type="text" placeholder="First Name">
  <input    formControlName="lastName"    class="form_input"    type="text" placeholder="Last Name">
  <input    formControlName="phone"    class="form_input"    type="phone    " placeholder="Phone">
  <input   formControlName="password"   class="form_input"   type="password"
```

```
placeholder="Password">

  <button [disabled]="!form.valid"
          class="form_button">Register</button>
  <a  class="form_link"  routerLink="/authorization/authentication">Do  you
have an account?</a>
</form>
```

Лістинг 5.5

```
@import      'src/app/components/registration-authentication/authorization-
general.scss';

:host {
  display: flex;
  flex-direction: column;
}
```

Лістинг 5.6

```
.authorization_form {
  display: flex;
  flex-direction: column;

  .form_input {

    background-color: #CDC4BC;
    padding: 10px 20px;
    border-radius: 10px;

    & + .form_input {

      margin-top: 20px;
    }
  }

  .form_button {
    margin-top: 30px;
```

```
    background-color: #664C3A;
    border-radius: 10px;
    padding: 5px;
    color: #fff;
    cursor: pointer;

    &:disabled {
      border: 1px solid #999999;
      background-color: #cccccc;
      color: #7a7a7a;
    }
  }

  .form_link {
    margin-top: 10px;
    color: #664C3A;
  }
}
```

Лістинг 5.7

```
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.scss']
})
export class RegistrationComponent implements OnInit {

  form!: FormGroup;


  constructor() {

    this.form = new FormGroup({
      firstName: new FormControl('', [Validators.required]),
      lastName: new FormControl('', [Validators.required]),
```

```
      phone: new FormControl('', [Validators.required]),
      password:       new       FormControl('',       [Validators.required,
Validators.min(8)]),
    });
  }

  ngOnInit(): void {

  }

}
```

Лістинг 5.8

```
<h1>Authentication</h1>

<form [formGroup]="form" class="authorization_form">

  <input     formControlName="phone"     class="form_input"     type="text"
placeholder="Phone">
  <input    formControlName="password"    class="form_input"    type="text"
placeholder="Password">

  <button [disabled]="!form.valid"
    class="form_button">Authenticate</button>

  <a class="form_link" routerLink="/authorization/registration">Don't have
an account?</a>
</form>
```

Лістинг 5.9

```
@import      'src/app/components/registration-authentication/authorization-
general.scss';

:host {
  display: flex;
  flex-direction: column;
```

```
}
```

Лістинг 5.10

```
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";

@Component({
  selector: 'app-authentication',
  templateUrl: './authentication.component.html',
  styleUrls: ['./authentication.component.scss']
})
export class AuthenticationComponent implements OnInit {

  form!: FormGroup;

  constructor() {

    this.form = new FormGroup({
      phone: new FormControl('', [Validators.required]),
      password: new FormControl('', [Validators.required]),
    });
  }

  ngOnInit(): void {

  }

}
```

Лістинг 5.11

```
<section class="top">
  <header>
    <img class="logo" src="./assets/images/logo.png" alt="logo">
    <nav>
      <ul>
        <li>
```

```
          <a>Hole</a>
        </li>
        <li>
          <a>Equipment</a>
        </li>
        <li>
          <a>About us</a>
        </li>
      </ul>
    </nav>
  </header>


  <img class="woman_img" src="./assets/images/Woman_main%201.png" alt="">

  <div class="register_block">
    <h1>Feel yourself and your body</h1>
    <h2>Yoga will help you in this</h2>
    <a routerLink="/authorization/registration">Register</a>
  </div>

  <button class="go_down">\/</button>

</section>
<section class="statistic">
  <ul>
    <li>+500 satisfied customers</li>
    <li>+220 comments</li>
    <li>+3 000 conducted classes</li>
  </ul>
</section>
<section class="how_to_get">

  <h2>How to get to us</h2>

  <div class="block left_block">
    <div class="sub_block">
      <span class="text">Fill out the questionnaire</span>
    </div>
```

```
    <span class="number">01</span>
  </div>
  <div class="block right_block">
    <span class="number">02</span>
    <div class="sub_block">
      <span class="text">Choose a convenient date and time</span>
    </div>
  </div>
  <div class="block left_block">
    <div class="sub_block">
      <span class="text">Watch video training</span>
    </div>
    <span class="number">03</span>
  </div>

</section>
<section class="our_trainers">

  <h2>Our trainers</h2>

  <div class="section">
    <div class="description">
      <p class="title">trainer <span class="name">Olena</span></p>
      <p class="experience">Experience in classes 6 years</p>
      <p>Olena is a personal yoga trainer. It will help you gain control over your body and mind. He conducts personal training so that your training is as comfortable as possible</p>
    </div>
    <div class="photo">
      <img src="./assets/images/olena.png" alt="">
    </div>
  </div>
  <div class="section">
    <div class="photo">
      <img src="./assets/images/natalia.png" alt="">
    </div>
    <div class="description">
      <p class="title">trainer <span class="name">Natalia</span></p>
      <p class="experience">Experience in classes 4 years</p>
```

```
      <p>Natalia conducts group yoga classes. It will help you feel light,
flexible and full of strength</p>
    </div>
  </div>

</section>
<footer>
  <div class="block">
    <img src="./assets/images/logo.png" alt="">
    <ul>
      <li>Occupation</li>
      <li>Implement</li>
      <li>About us</li>
    </ul>
  </div>
  <div class="block">
    <input type="text" placeholder="Search...">
    <ul>
      <li>0-800-123-789</li>
      <li>karma_yoga@gmail.com</li>
    </ul>
  </div>
  <div class="block">
    <span>Actions:</span>
    <ul>
      <li>Clothing</li>
      <li>Sneakers</li>
      <li>Vitamins</li>
      <li>Inventar</li>
    </ul>
  </div>
</footer>
```

Лістинг 5.12

```
.top {

  position: relative;
```

```
height: 100vh;

header {
  display: flex;
  align-items: center;
  padding: 0 20px;
}

ul {
  display: flex;
  margin-left: 100px;
}

li {
  display: flex;
}

li + li {

  margin-left: 20px;
}

.woman_img {
  position: absolute;
  right: 0;
  top: 0;
  height: 100vh;
}

.register_block {

  border-top: 2px solid #664C3A;
  border-bottom: 2px solid #664C3A;
  width: 300px;
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  padding: 20px 20px 20px 50px;
```

```
    color: #664C3A;

    h2 {
      font-size: 16px;
    }

    a {
      padding: 10px 15px;
      border-radius: 5px;
      background-color: #664C3A;
      color: white;
      margin-top: 20px;
      display: inline-block;
    }
  }

  .go_down {
    background-color: #CDC4BC;
    color: #664C3A;
    height: 50px;
    width: 50px;
    border-radius: 100%;
    position: absolute;
    bottom: 10px;
    right: 50%;
    transform: translateY(-50%);
  }
}

.statistic {
  background-color: #CDC5BC;
  color: white;

  ul {
    padding: 30px 0;
    margin: 0;
    font-size: 30px;

    display: flex;
```

```
    justify-content: space-around;
  }

  li {
    display: flex;
  }
}

.how_to_get {
  height: 100vh;
  text-align: center;

  h2 {
    color: #664C3A;
  }

  .block {
    height: 30%;
    display: flex;
    justify-content: space-between;
    align-items: center;

    .text {
      font-size: 40px;
      color: #664C3A;
    }

    .number {
      color: white;
      font-weight: 700;
      font-size: 128px;
    }

    .sub_block {
      padding: 20px;
      border-top: 1px solid #664C3A;
      border-bottom: 1px solid #664C3A;
    }
  }
```

```
  .left_block {

    .sub_block {

      border-right: 1px solid #664C3A;
      border-radius: 0 50px 50px 0;
    }

    .number {

      margin-right: 100px;
    }
  }

  .right_block {

    .sub_block {

      border-left: 1px solid #664C3A;
      border-radius: 50px 0 0 50px;
    }

    .number {

      margin-left: 100px;
    }
  }
}

.our_trainers {
  height: 100vh;
  background-color: #CDC4BC;
  padding: 20px 0;

  .section {
    height: 40%;
    display: flex;
  }
```

```
  h2 {
    color: #664C3A;
  }

  .description {
    width: 50%;
    border: 1px solid #664C3A;
    background-color: #DAD7D4;
    color: #664C3A;
    padding: 10px;
  }

  .photo {
    width: 50%;
    align-items: center;
    text-align: center;
  }

  .name {
    font-weight: bold;
  }

  img {
    height: 100%;
  }

  .experience {
    color: white;
  }
}

footer {

  padding: 20px 0;
  background-color: #CDC4BC;
  border-top: 1px solid #664C3A;
  display: flex;
  justify-content: space-around;
```

```
  .block {
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
  }

  ul {
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
  }

  li {
    display: flex;

    & + li {
      margin-top: 10px;
    }
  }

  input {
    padding: 5px 10px;
    border-radius: 20px;
  }
}
```