Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему:

Програмне забезпечення системи кібербезпеки Web-додатку з впровадженням e-service

КбПЗ_2024

Виконав здобувач вищої освіти
IV курсу, групи КІ-20
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____Ґрант Д.В.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук, доцент
Босько В.В
«____» _____ 2024 р.
Рецензент _____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань. 12 *"Інформаційні технології"*
Спеціальність *123 " Комп'ютерна інженерія "*
Освітньо-професійна (освітньо-наукова) програма *" Комп'ютерна інженерія "*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
"____" _____2024 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Ґранту Даніілу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки Web-додатку з впровадженням e-service*

2. Керівник роботи *Босько Віктор Васильвич, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 01.04.2024 року № 136-02

3. Строк подання роботи до захисту *20.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи. *Розробка програмного забезпечення системи кібербезпеки для web додатку*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію
6. Висновки.

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « » 2024р.

КОПІЯ 2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем керування	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.05.2024 р.	
8.	Попередній захист роботи	20.05.2024 р.	

КОПЗ_2024

Дата видачі завдання
«__»_____2024р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання

«__»_____2024р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Грант Д.В. Програмне забезпечення системи кібербезпеки Web-додатку з впровадженням e-service

123 Комп'ютерна Інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення інформаційної системи, яка буде спрощувати бізнес процеси спортивного закладу, шляхом надання сервіс-орієнтованих програмних засобів.

Метою роботи. Метою даної роботи є створення програмного забезпечення, яке надасть автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів, що дозволить контролювати навантаження закладу та збереже відвідувачів.

В ході роботи був розроблений сервіс-орієнтований веб-додаток який надає автоматизовані сервіси в сферу контролю та управління доступом спортивних комплексів.

Для вирішення завдань було визначено архітектуру розроблюваного додатку, а також проаналізовано сучасні підходи до побудови веб- застосунків з використанням мови програмування TypeScript та фреймворку Angular та перевірено їх ефективність на прикладі власного додатку.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів.

В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Ключові слова: кібербезпека, web, webpack, e-сервіс, GIT, Sass.

КПЗ_2024

ANNOTATION

Grant D.V. Web application cyber security software with e-service implementation

123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this qualifying bachelor's thesis, the software of the information system was developed, which will simplify the business processes of the sports institution by providing service-oriented software tools.

The purpose of the work. The purpose of this work is to create software that will provide automated services in the field of control and management of access to sports complexes, which will allow to control the load of the facility and save visitors.

During the work. A service-oriented web application was developed that provides automated services in the field of control and access management of sports complexes.

To solve the problems, the architecture of the developed application was determined, and modern approaches to building web applications using the TypeScript programming language and the Angular framework were analyzed, and their effectiveness was checked using the example of our own application.

In the process of working on the software model, an analysis of existing hardware and software was performed.

All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

Keywords: cyber security, web, webpack, e-service, GIT, Sass.

КОПЗ – 2024

ЗМІСТ

КОПІЯ 2024

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

REST – Representational State Transfer

MVC – Model-View-Controller

ПУ – Пристрої управління

MVC - Model-View-Controller

СКУД – системи контролю та управління доступом

RxJs – бібліотека для асинхронного програмування.

ОС - операційна система

XML - Extensible Markup Language

DI – Dependency Injection

ORM - Object-relational mapping

Sass - Syntactically Awesome Stylesheets

DOM - Document Object Model

EJS - Embedded JavaScript templating

КБПЗ_2024

ВСТУП

Актуальність теми. Спорт є корисним для здоров'я, проте іноді через графік, переповнений різними справами, люди не завжди можуть знайти час для зв'язку з адміністратором чи відвідування спортивного закладу, щоб записатися на тренування, повідомити про скасування або купити квитки заздалегідь. Це призводить до ситуацій, коли відвідувач, не бажаючи витрачати час на довгий процес, покидає заклад, а той, у свою чергу, втрачає клієнта. Також можуть виникати проблеми з неправильним розподілом відвідувачів, коли об'єкти або переповнені, або майже порожні.

Отже, проблема полягає у розробці інформаційної системи, яка б спростила бізнес-процеси спортивного закладу за допомогою сервіс-орієнтованих програмних рішень. Є актуальним створення програмного забезпечення, яке забезпечить автоматизовані сервіси для контролю та управління доступом у спортивні комплекси, дозволяючи керувати навантаженням на заклад та зберігати лояльність відвідувачів.

Для цього необхідна система відкритого спортивного майданчику, яка складається з веб-додатку, серверу та системи контролю і управління доступом. Веб-додаток дозволить інтегрувати сервіс-орієнтоване програмне забезпечення, яке надасть користувачам можливість з будь-якого пристрою з інтернетом виконувати онлайн-операції, такі як реєстрація, авторизація, перегляд розкладу занять, кількості вільних місць, деталів подій, бронювання та перегляд або скасування особистих занять.

Користувачами цієї системи можуть бути мешканці міста, де встановлено систему відкритого спортивного майданчику. На сьогоднішній день вже існують аналогічні рішення, такі як MyFit та GiftFitness, які надають інформацію про спортивні установи.

Ця система дозволить ефективно розподіляти та контролювати потік

КПЗ_2024

відвідувачів у спортивному закладі, використовуючи графік. Користувачі зможуть бронювати чи скасовувати заняття онлайн, не витрачаючи час на зв'язок з адміністратором спортивного майданчику.

Система також зберігатиме особисті дані користувачів, що значно спростить їм процес відвідування і дозволить економити час.

Таким чином, розроблювана система покращить управління потоком відвідувачів, оптимізує завантаженість майданчика і зробить процес взаємодії зі спортивним закладом простішим і зручнішим для користувача.

Об’єктом дослідженння є інформаційні системи для забезпечення бізнес процесів.

Мета роботи - створити програмне забезпечення, яке автоматизує сервіси управління та контролю доступу до спортивних комплексів, що дозволить контролювати навантаження та зручності для відвідувачів.

Для досягнення цієї мети були виконані наступні завдання:

- визначено архітектуру системи "Відкритого спортивного майданчику";
- проведено аналіз існуючих систем та рішень;
- розроблено сервіс-орієнтований веб-додаток для відкритих спортивних майданчиків, який швидко працює, має кросплатформенність та надає актуальну інформацію про заклад, а також дозволяє виконувати основні операції онлайн;
- розроблено концепцію проекту для його можливої комерціалізації;
- запропоновано варіанти подальшого вдосконалення та розширення системи.

КПЗ - 2024

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

У зв'язку з проблемами доступу відвідувачів до спортивних майданчиків, були розроблені інформаційні системи, які автоматизують контроль та управління доступом у спортивні комплекси. Було проведено аналіз існуючих рішень, на основі якого були визначені вимоги до системи, що розробляється. Також були окреслені основні завдання для даної кваліфікаційної роботи.

Основні завдання для інформаційних систем, що розробляються, можуть включати наступне:

Контроль доступу: Система повинна забезпечувати контроль доступу до спортивних майданчиків шляхом автоматичної ідентифікації відвідувачів та перевірки їхніх прав на доступ.

Управління правами доступу: Вона повинна надавати можливість налаштування рівнів доступу для різних користувачів, включаючи адміністраторів, персонал та відвідувачів.

Моніторинг відвідування: Система може збирати та аналізувати дані про відвідування, такі як час і дату відвідування, кількість відвідувачів тощо, щоб забезпечити ефективне управління та планування.

Інтеграція з іншими системами: Вона може інтегруватися з іншими системами управління, такими як системи бронювання, платіжні системи та системи відеоспостереження, для забезпечення повного циклу обслуговування.

Забезпечення безпеки: Система повинна мати вбудовані заходи безпеки для захисту від несанкціонованого доступу та зламів.

Забезпечення зручності користувачів: Важливою функцією є забезпечення зручності для користувачів, забезпечення швидкого та простого доступу до спортивних майданчиків.

Інформаційні системи для контролю та управління доступом у спортивні комплекси мають значний потенціал для полегшення управління і покращення

КПЗ_2024

досвіду відвідувачів. Вони можуть допомогти уникнути перевантажень, забезпечити безпеку та контроль, а також оптимізувати управління ресурсами.

1.1 Область застосування

Область застосування e-service включає в себе різноманітні сектори, такі як електронна комерція, електронні державні послуги, онлайн-банкінг, електронне здоров'я, електронна освіта, електронна торгівля, онлайн-туризм і багато інших.

Е-сервіси, або електронні сервіси, відіграють важливу роль у сучасному світі, забезпечуючи зручний доступ до послуг через Інтернет. Їх застосування охоплює широкий спектр секторів економіки і суспільства.

Електронна комерція (e-commerce): Е-сервіси в електронній комерції дозволяють споживачам здійснювати покупки онлайн, шукати товари та порівнювати ціни, забезпечуючи зручність та доступність. Платформи електронної комерції, такі як Amazon, eBay, а також онлайн-магазини брендів, надають широкий вибір товарів і послуг, забезпечуючи зручність покупцям у будь-який час та в будь-якому місці.

Електронні державні послуги (e-government): Уряди у всьому світі впроваджують е-сервіси для надання громадянам та бізнесу доступу до різних адміністративних послуг онлайн. Це включає в себе можливість подання електронних декларацій, заяв та запитів, оплату податків і штрафів, а також взаємодію з урядовими органами через веб-портали та мобільні додатки.

Онлайн-банкінг та фінансові послуги: Банки та фінансові установи надають клієнтам можливість виконувати фінансові операції через Інтернет, такі як переказ коштів, оплата рахунків, замовлення кредитів, купівля та продаж цінних паперів. Електронні платіжні системи, такі як PayPal, Stripe, також надають послуги з обробки платежів в Інтернеті, спрощуючи онлайн-торгівлю.

КПЗ_2024

Електронне здоров'я (e-health): В галузі охорони здоров'я е-сервіси використовуються для забезпечення пацієнтам доступу до медичної інформації, онлайн-консультацій з лікарем, запису на прийом до лікаря, замовлення медичних аналізів та ліків через Інтернет. Це сприяє підвищенню доступності та ефективності медичних послуг.

Електронна освіта (e-learning): У сфері освіти е-сервіси використовуються для навчання за допомогою відеоуроків, інтерактивних курсів, вебінарів та онлайн-платформ навчання. Вони дозволяють студентам отримувати освіту у будь-який час та в будь-якому місці, а також надають можливість персоналізованого навчання.

Електронна торгівля (e-tourism): У галузі туризму е-сервіси використовуються для бронювання готелів, квитків на транспорт, туристичних екскурсій та інших туристичних послуг через Інтернет. Онлайн-туристичні агентства та портали надають можливість клієнтам здійснювати планування подорожей та покупку послуг у зручний спосіб.

Е-сервіси мають великий потенціал для полегшення життя людей та покращення ефективності різних галузей. Вони стають все більш популярними та невід'ємною частиною нашого цифрового світу, надаючи доступ до послуг у будь-який час та з будь-якого пристрою з доступом до Інтернету.

КбПЗ-2024

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським рівнем вищої освіти)

На сьогодні існують системи, які надають інформацію про спортивні заклади, зокрема “MyFit” та “Gift Fitness”. Ці платформи дозволяють користувачам отримувати актуальні дані про розклади занять, доступність тренувань, а також бронювати та скасовувати візити до спортивних комплексів. (рисунок 2.1, 2.2).

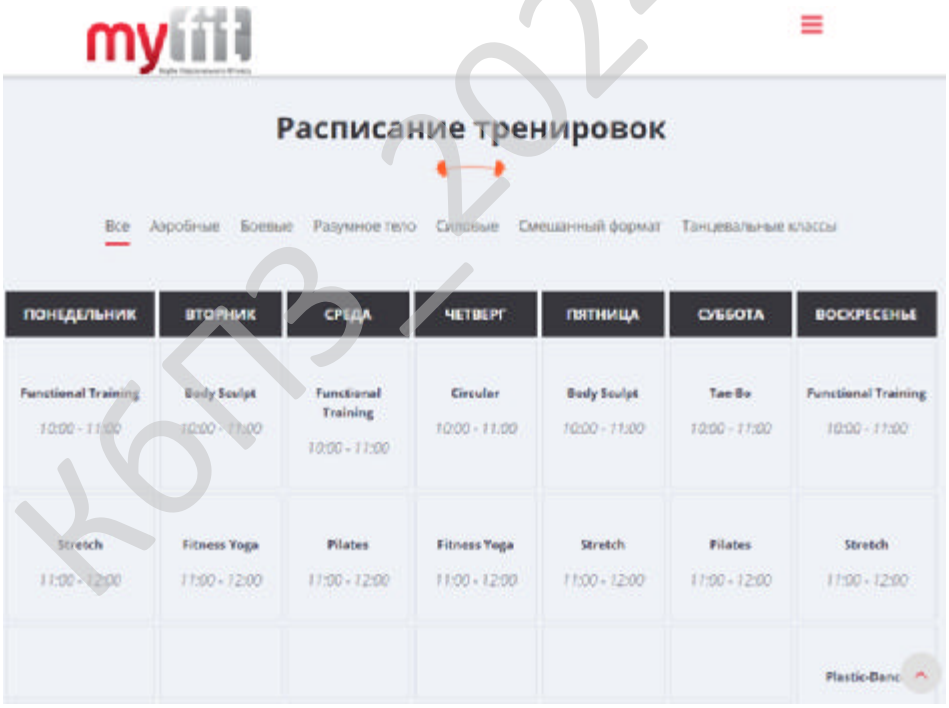

Рисунок 2.1 - Система “MyFit”

Наразі доступні системи, такі як "MyFit" та "Gift Fitness", дозволяють переглядати розклад та види спорту, які викладаються в спортивних закладах, але вони не надають можливості онлайн запису або скасування занять. Зазвичай, для запису на заняття потрібно звертатися до адміністратора телефоном. Крім того,

КОПІЯ 2024

заняття можуть бути перенесені лише обмежену кількість разів, і якщо заняття не перенесено, відвідувач втрачає як заняття, так і кошти, що були за нього сплачені.

Це може призвести до незадоволення відвідувачів і вибору ними іншого спортивного закладу.

Також користувачам часто не зручно телефонувати для відміни заняття, що може призводити до неправильного розрахунку кількості відвідувачів у спортивному майданчику в певний час. Це може спричинити переповнення або, навпаки, недостатню заповненість спортивного закладу.

Аналізуючи існуючі системи, було визначено базові вимоги до нової системи, яка має вирішити зазначені проблеми:

a) Можливість онлайн-запису на заняття: Користувачі мають мати змогу самостійно записуватися на заняття через інтернет без необхідності зв’язування з адміністратором.

b) Можливість онлайн-скасування заняття: Користувачі мають можливість скасувати заняття онлайн, що допоможе уникнути втрати коштів і заняття.

c) Автоматичне перенесення занять: Система має надавати гнучкість у перенесенні занять з автоматичним повідомленням відвідувачів.

d) Актуальність інформації про кількість відвідувачів: Система має автоматично оновлювати дані про кількість відвідувачів, щоб забезпечити оптимальне розподілення навантаження на спортивний заклад.

e) Інтуїтивно зрозумілий інтерфейс: Інтерфейс користувача має бути зрозумілим і зручним, щоб сприяти легкому взаємодії з системою.

КПЗ – 2024

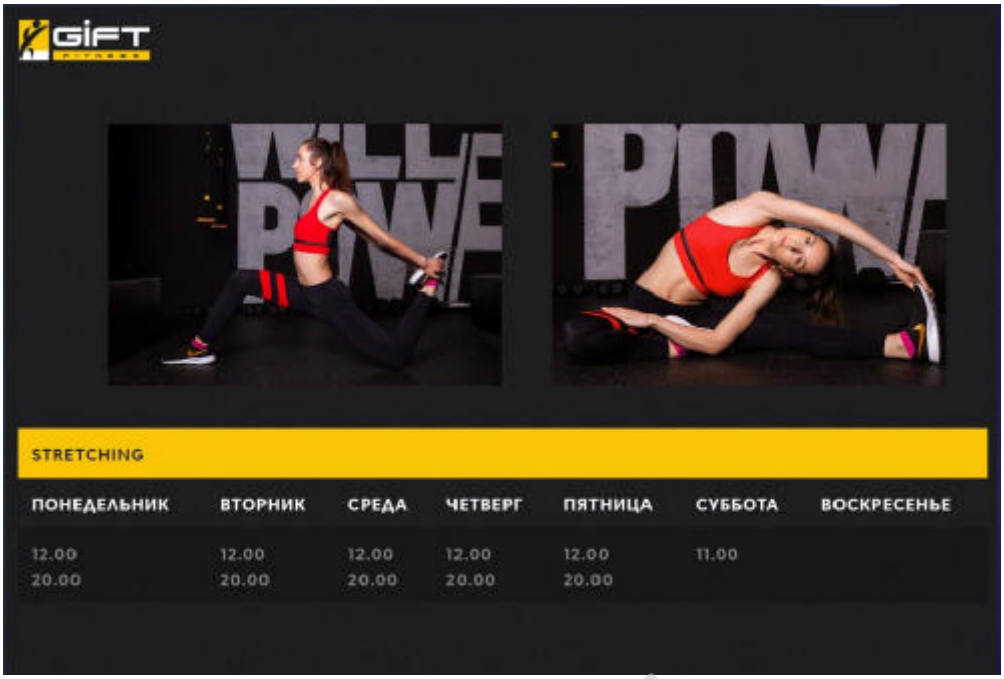

Рисунок 2.2 — Система “ Gift Fitness ”

Враховуючи, що на території міста існує багато спортивних майданчиків, які є відкритими та потребують оновлення та покращення, було вирішено створити систему "Відкритого спортивного майданчику". Ця система дозволить людям відвідувати майданчики в будь-який час, переглядати інформацію про заходи, наявність місць та записуватися на події чи скасовувати свій запис онлайн. Перший проект планується реалізувати на території студентського містечка.

Враховуючи потребу в гнучкості та можливості розширення системи, при розробці архітектури "Відкритого спортивного майданчику" було обрано сервіс-орієнтований підхід. Сучасні підприємства, установи та спортивні заклади потребують швидкої адаптації до змін у бізнес-цілях, гнучкої інтеграції, та забезпечення надійності в реальному часі. Сервіс-орієнтована архітектура (SOA) ідеально підходить для цих цілей.

Сервіс-орієнтована архітектура (SOA) розробляється на основі проектування та розробки сервісів, які є самостійними робочими або бізнес-функціями, призначеними для узгодженої роботи додатків. Це дозволяє системам легко адаптуватися до змін, оскільки модифікації в одному сервісі автоматично

КОПЗ_2024

розповсюджуються на всі пов'язані додатки без необхідності в ручному переписуванні коду.

Ось основні переваги використання SOA для "Відкритого спортивного майданчику":

1) Гнучкість та Масштабованість: Система може бути легко адаптована під змінні потреби користувачів і розширена новими функціональними можливостями без зміни основного коду.

2) Реюзабельність Компонентів: Сервіси можуть бути використані в різних частинах системи або навіть в інших системах, що знижує витрати на розробку.

3) Спрощення Інтеграції: SOA сприяє легшій інтеграції з іншими системами, оскільки сервіси мають чітко визначені інтерфейси.

4) Підтримка Реального Часу: Система може оперативно реагувати на зміни завдяки швидкому оновленню сервісів.

5) Оптимізація Взаємодії з Користувачем: Використання сервісів дозволяє забезпечити більш зручний і інтуїтивно зрозумілий інтерфейс для користувачів, що сприяє кращому взаємодії з системою.

Ця архітектурна підхід до "Відкритого спортивного майданчику" забезпечить стабільність, високу продуктивність та легку адаптацію до майбутніх вимог розвитку спортивних закладів і потреб відвідувачів, всі необхідні додатки [2].

Розробка високорівневої архітектури системи "Відкритого спортивного майданчику" на основі сервіс-орієнтованої архітектури (SOA) дозволяє створити гнучку та масштабовану систему. Розбиття системи на окремі сервіси згідно з їх бізнес-функціями забезпечує зручність управління та можливість легко адаптувати систему до змінюваних вимог бізнесу. Ось детальний аналіз та викладення цієї структури:

Сервіси системи "Відкритого Спортивного Майданчику".

Сервіс Контролю та Управління Доступом (СКУД): відповідає за

КБПЗ-2024

авторизацію та аутентифікацію користувачів, регулювання доступу до різних зон майданчика. Переваги Сервіс Контролю та Управління Доступом (СКУД) забезпечує контроль над тим, хто і коли може відвідувати спортивний майданчик, підвищуючи безпеку та ефективність використання ресурсів.

Платіжна Система: управління фінансовими транзакціями, включаючи оплату занять, абонементів та інших послуг. Спрощує процес оплати, підтримує різні платіжні методи, знижує адміністративні витрати.

Система Відеоспостереження: моніторинг території майданчика для забезпечення безпеки та контролю за дотриманням правил. Переваги Системи Відеоспостереження допомагає запобігати порушенням, забезпечує доказову базу у разі інцидентів.

Система суддійства: керування спортивними змаганнями, включаючи фіксацію результатів і вирішення спірних моментів. Переваги Системи Суддівства Автоматизації процесу суддівства - підвищує прозорість та об'єктивність рішень.

Система видачі спортивного інвентаря: управління інвентарем, забезпечення видачі та прийому обладнання. Переваги Системи Видачі Спортивного Інвентаря - оптимізація використання ресурсів, зменшення часу на обслуговування клієнтів.

Система енергопостачання: моніторинг та управління використанням енергії, забезпечення енергоефективності майданчика. Переваги Системи Енергопостачання - зниження витрат на енергію, підвищення сталості діяльності спортивного майданчика.

Система вуличного освітлення: автоматичне керування освітленням залежно від часу доби та погодних умов. Переваги Системи Вуличного Освітлення - забезпечення безпеки і комфорту відвідувачів у вечірній час, економія енергії.

Принципи та вигоди використання Сервіс-Орієнтованої Архітектури:

- Модульність: розбиття загальної системи на сервіси дозволяє розвивати і вдосконалювати кожен компонент незалежно, зменшуючи ризики і

КПЗ – 2024

витрати на впровадження змін.

- Гнучкість: при зміні бізнес-вимог, можна адаптувати або замінити окремі сервіси без необхідності переробки всієї системи.
- Масштабованість: легше масштабувати окремі сервіси відповідно до зростаючих навантажень чи змін у використанні.

Перспективи Розвитку

Система може бути введена в експлуатацію з базовим набором сервісів, а додаткові функції можуть бути реалізовані у міру необхідності, забезпечуючи поступове впровадження нових можливостей без зупинки діючих процесів. Це дозволяє спортивним майданчикам ставати більш привабливими для відвідувачів і ефективними в управлінні, забезпечуючи високий рівень задоволення користувачів та оптимальне використання ресурсів.

Цей підхід в архітектурі надає велику вартість для управління спортивними майданчиками, оскільки він забезпечує високий рівень адаптивності та підтримки різноманітних потреб користувачів і змінних бізнес-сценаріїв.

Визначивши проблему існуючих систем та проаналізувавши приклади застосунків було визначено вимоги до клієнтської частини. Такі як надання актуальної інформації користувачеві, надання можливості виконувати базові операції майданчику онлайн, кросплатформеність та швидкодія.

Відповідно до даних вимог було вирішене створити клієнтську частину як веб-застосунок, що вирішить проблему кросплатформеності та швидкодії, а для можливості виконання операцій онлайн та забезпечення актуальної інформації, веб-застосунок комунікуватиме з сервером, який пов’язаний з системою СКУД.

КПЗ – 2024

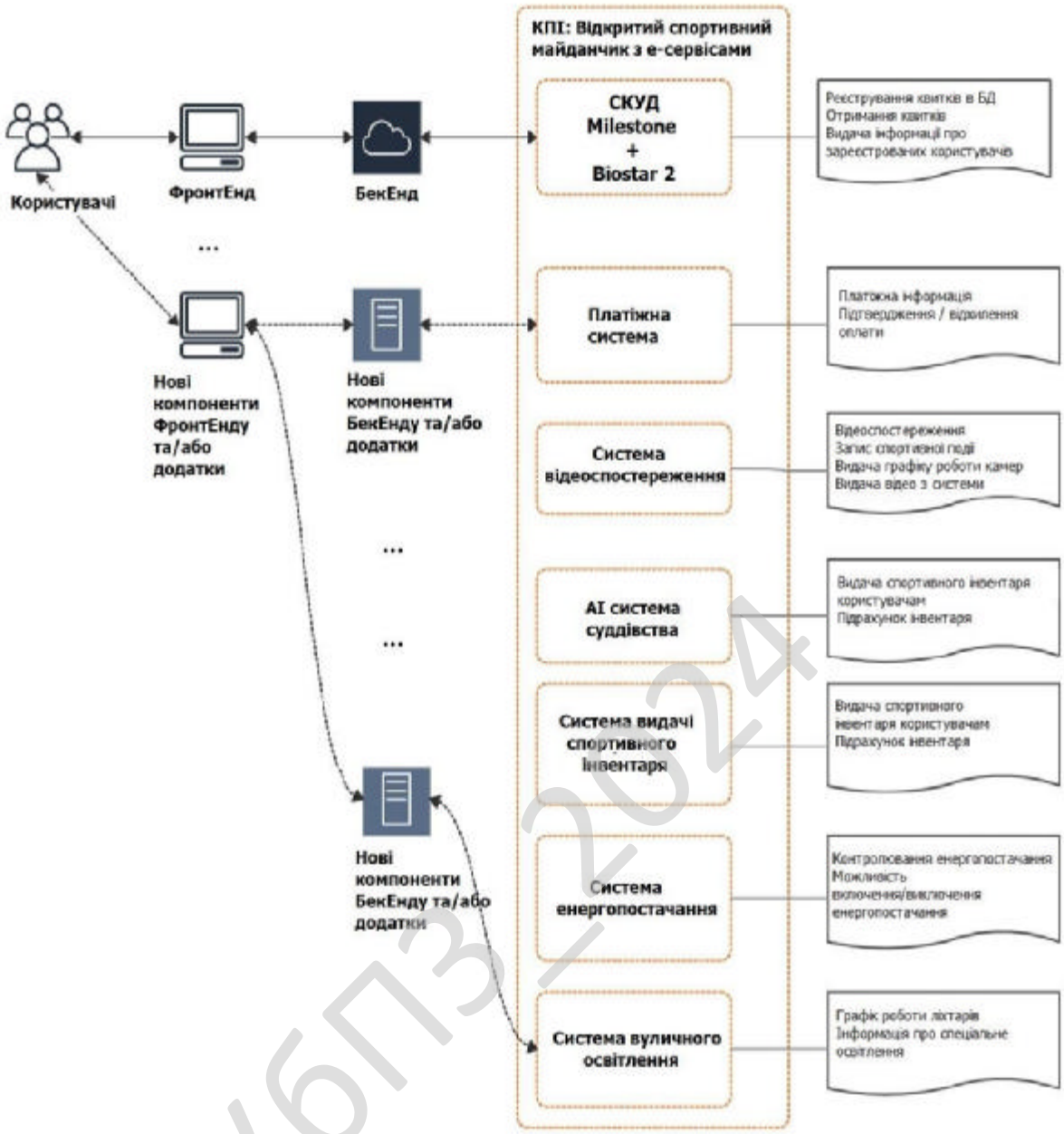

Рисунок 2.1 — Архітектура системи

Система контролю і управління доступом (СКУД), яка є частиною архітектури "Відкритого спортивного майданчику", дійсно включає три основні компоненти, які взаємодіють між собою для забезпечення ефективного та безпечного доступу до спортивного майданчику. Ось деталізація кожного з компонентів:

- Front-end Додаток (Angular) цей компонент створений на фреймворку Angular, який дозволяє розробляти динамічні та інтерактивні веб-інтерфейси.

КПЗ 2024

Angular забезпечує розробку односторінкових застосунків (SPA), що робить навігацію та взаємодію з користувачем швидкою і плавною.

- **Реєстрація та авторизація користувачів**: Управління обліковими записами користувачів, включаючи реєстрацію нових та вхід існуючих користувачів.

- **Бронювання і покупка квитків:** Інтерфейс для перегляду доступних занять, подій, та бронювання чи покупки квитків на них.

- **Перегляд інформації про заходи**: Відображення актуальної інформації про наявність місць, час проведення занять і спеціальних подій.

Back-end Додаток (Розгорнутий в Хмарі) серверна частина системи, яка обробляє логіку додатку, взаємодіє з базами даних і інтегрується з іншими сервісами. Використання хмарних платформ, як AWS, Azure або Google Cloud, дозволяє забезпечити масштабованість, високу доступність і надійність системи.

Основні функції:

- управління даними користувачів: Обробка даних про користувачів, включаючи їх реєстраційну інформацію та історію відвідувань;
- інтеграція з системою СКУД: Взаємодія з фізичними системами контролю доступу для верифікації прав користувачів;
- обробка платежів і бронювань**:** Управління фінансовими транзакціями та забезпечення коректного бронювання квитків.

Система Контролю та Управління Доступом (СКУД) фізична та програмна система, що контролює доступ до майданчику, включаючи вхідні ворота, турнікети, двері тощо. СКУД інтегрується з ідентифікаційними картками, RFID-мітками або біометричними системами.

Основні функції:

- моніторинг доступу: Контроль і запис всіх входів та виходів на територію майданчика з можливістю швидкого реагування на несанкціоновані дії;
- автоматизація процесів контролю: Автоматичне управління доступом на основі даних, отриманих від back-end системи;

КПЗ-2024

- взаємодія з базою даних: Підтримка актуальності даних про права доступу користувачів, їхні реєстрації та ідентифікації;

Шляхи Розвитку та Інтеграції - структура надає міцний фундамент для розвитку та інтеграції додаткових сервісів:

- інтеграція з платіжними системами для спрощення процесів купівлі квитків;
- розширення функціоналу для видачі спортивного інвентарю через автоматизовані системи видачі;
- введення системи відеонагляду для підвищення безпеки та контролю за порядком на майданчику;
- модернізація системи енергопостачання та освітлення для ефективного управління ресурсами;

Ці компоненти та шляхи їхнього розвитку забезпечують не тільки надійність і безпеку використання спортивного майданчику, але й гнучкість для адаптації під мінливі потреби користувачів та управління об'єктом.

Розробка базової інформаційної системи контролю і управління доступом для "Відкритого спортивного майданчику" видається як ключовий крок у напрямку створення гнучкої та ефективної платформи для керування спортивними заходами. Ця система, складаючись з Front-end, Back-end, та СКУД компонентів, вже виконує ряд важливих операцій і має потенціал для значного розширення.

Базова інформаційна система контролю і управління доступом вже забезпечує ключові функції для взаємодії користувачів зі спортивним майданчиком, але її потенціал для розвитку та інтеграції нових сервісів може значно розширити можливості та ефективність використання об'єкту. Крок за кроком, ця система може стати взірцем для сучасного управління спортивними об'єктами, відповідаючи на виклики часу та забезпечуючи високий рівень сервісу та безпеки.

КПЗ – 2024

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Метою даної роботи є створення програмного забезпечення для автоматизації сервісів у сфері контролю та управління доступом спортивних комплексів. Це забезпечить контроль навантаження закладу і збереже безпеку відвідувачів, забезпечуючи зручний і ефективний доступ до засобів моніторингу та управління.Додаток розроблений як веб-сайт, тому доступний користувачам при наявності мережі Інтернет з десктопу, планшету та мобільного пристрою.

Система розроблена з урахуванням потреб різних категорій користувачів:

- **Студенти:** Мають можливість використовувати студентські квитки для отримання спеціальних знижок або пріоритетного доступу.
- **Жителі Міста:** Люди, які проживають у місті, де розташований спортивний майданчик, можуть використовувати систему для планування та участі в заходах.

Для забезпечення правильної роботи системи використовуються наступні вхідні дані:

- **Особисті дані:** Користувачі реєструються, вводячи свою електронну пошту, ім’я та прізвище.
- **Студентський квиток:** Для студентів потрібно надати номер студентського квитка для підтвердження статусу і можливості використання особливих пропозицій.

На виході система надає користувачам інформацію:

- **Забронювані Заходи:** Деталі про події та заходи, на які користувач зарезервував місце, включаючи час, місце проведення, і статус бронювання.

Надання Актуальної Інформації:

- Система автоматично оновлює інформацію про заходи, розклади та доступність майданчика, гарантуючи, що всі дані є найновішими.

КПЗ_2024

Базові Онлайн Операції:

- Авторизація: перевірка правильності введених користувачем даних та надання доступу до його профілю.
- Реєстрація: введення і збереження особистих даних нових користувачів у системі.
- Перегляд Розкладу: можливість переглядати актуальний розклад заходів і доступність майданчика.
- Бронювання: зезервація місць на певні заходи або активності з можливістю вибору часу і дати.
- Скасування Бронювання: дозволяє користувачам скасувати своє бронювання, якщо змінили плани.

Кросплатформенність та Швидкодія:

- Система оптимізована для використання на різних пристроях, включаючи десктопи, планшети та мобільні телефони, забезпечуючи високу швидкість відгуку і легкість використання.

Відповідно до вимог та мети роботи було визначено основні завдання:

Визначення архітектури системи - розробка гнучкої та масштабованої архітектури, яка дозволить ефективно управляти доступом до спортивного майданчику та інтегрувати додаткові сервіси в майбутньому.

Розбиття на Модулі: Розділення системи на основні модулі та сервіси, такі як авторизація, реєстрація, бронювання, інтеграція з СКУД, та управління даними.

Сервіс-Орієнтований Підхід - використання принципів SOA (Service-Oriented Architecture) для забезпечення високого рівня гнучкості та можливості легкого додавання нових сервісів.

Контейнеризація та Хмарні Рішення - використання Docker, Kubernetes для оркестрації контейнерів та розгортання в хмарних сервісах для забезпечення масштабування та надійності.

Аналіз існуючих систем та рішень дослідження ринку та аналіз конкурентів для визначення найкращих практик і технологій.

КПЗ-2024

Вивчення Ринку - аналіз існуючих аналогів та визначення їхніх сильних та слабких сторін.

Технологічний Аудит - визначення технологій, які використовуються в успішних проектах, і адаптація їх для наших цілей.

Збір Вимог - взаємодія з потенційними користувачами для збору вимог та уточнення функціональності системи.

Розробка Веб-Додатку - створення ефективного, кросплатформенного веб-додатку, який надасть актуальну інформацію про майданчик та забезпечить виконання основних операцій онлайн.

Front-End Розробка - створення інтерактивного користувацького інтерфейсу на базі Angular, що підтримує різні пристрої та розміри екранів.

Back-End Розробка - розробка серверної частини, яка включає API для обробки запитів від клієнта, інтеграцію з БД та СКУД.

Безпека та Оптимізація - забезпечення високого рівня безпеки даних та швидкодії додатку.

Концепція Стартап-Проекту для Комерціалізації - розробка стратегії для комерціалізації проекту, включаючи маркетинг, продажі та стратегію виходу на ринок.

Бізнес Модель: Розробка бізнес-моделі, включаючи визначення основних джерел доходу, ціноутворення та партнерських програм.

Маркетинговий План: Планування та впровадження маркетингових стратегій для просування продукту.

Фінансовий План: Розробка фінансового плану, включаючи бюджет, прогнози доходів та аналіз рентабельності.

Подальше покращення та розширення системи - визначення шляхів для подальшого розвитку та вдосконалення системи, з урахуванням змінних потреб користувачів та технологій.

Скейлінг: Планування та реалізація можливостей для масштабування системи з урахуванням збільшення кількості користувачів та об'єктів.

КОПЗ-2024

Інтеграція Додаткових Сервісів: Додавання нових функцій, таких як система відеонагляду, енергопостачання, платіжні системи та ін.

Збір та Аналіз Зворотного Зв'язку: Регулярний збір зворотного зв'язку від користувачів для удосконалення функціоналу та користувацького досвіду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке продемонструє роботу і побудову системи e_service.

В процесі розробки роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран повідомлень про некоректні дії користувача та нестандартні ситуації;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) сформувати висновки про виконаний обсяг робіт та одержані результати.

КПЗ-2024

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

Відповідно до вимог, що передбачають отримання актуальної інформації та здійснення базових операцій на онлайн-майданчику, а також необхідності забезпечення кросплатформеності та високої швидкості обробки даних, було прийнято рішення створити клієнтську частину як веб-застосунок. Це дозволить забезпечити сумісність з різними платформами та оптимізувати швидкодію. Веб-застосунок буде взаємодіяти з сервером для виконання операцій онлайн та отримання актуальної інформації.

Для розробки веб-застосунку було обрано JavaScript та TypeScript як удосконалену версію JavaScript з додаванням типізації. Використано фреймворк Angular, який є ідеальним для створення масштабованих архітектурних застосунків, які легко адаптуються під зміни. Також застосовано NPM (Node Package Manager) для управління пакетами, що сприяє легкому встановленню та використанню необхідних бібліотек. Для компіляції модулів проекту в формат, зрозумілий браузерам, використовується збірник Webpack.

Додатково, для управління змінами в проекті та ефективного контролю версій, використовувалася система контролю версій Git. Це дозволило забезпечити відстеження та управління різними версіями коду, а також сприяло злагодженій роботі команди. Збереження копій проекту на віддаленому репозиторії було здійснено за допомогою GitHub, що надало додаткову надійність та можливість швидкого відновлення проекту у разі необхідності. Таким чином, використання Git та GitHub допомогло не тільки у версіонуванні та збереженні історії змін, але й у спільній роботі над проектом, забезпечуючи координацію роботи розробників. Система контролю версій (Version Control System, VCS) — це система, що записує зміни в файл або набір файлів і дозволяє повернутися пізніше до певної версії.

За своїм базовим можливостям Git схожий з Mercurial (і іншими VCS), але

КПЗ-2024

завдяки ряду переваг (висока швидкість роботи, можливість інтеграції з іншими VCS, зручний інтерфейс) Git вийшов в лідери ринку розподілених систем контролю версій[3]. Тому для даного проекту було також вибрано систему контролю версій Git.

Система контролю версій Git надає можливість відновлення файлів до попередніх станів, перегляду змін, ідентифікації авторів змін, та виявлення джерел проблем у проекті. Використання Git дозволяє легко виправляти проблеми, пов'язані з втратою файлів або пошкодженням даних.

Репозиторій проекту на GitHub служить основним сховищем, і за допомогою команди git clone можна отримати його локальну копію для роботи. В робочому процесі з Git файли перебувають у таких станах:

- Working Directory: тут знаходяться файлы, з якими ведеться робота (додавання, зміна, видалення).

- Staging Area: для переведення файлів із робочої директорії до цього проміжного стану використовується команда git add . або git add *.

- Git Repository: для фіксації змін в локальному репозиторії застосовується команда git commit –m "message". Щоб синхронізувати ці зміни з віддаленим репозиторієм на GitHub, використовується команда git push.

Під час створення застосунку було використано стратегію гілкування "feature branching". Відповідно до цієї стратегії:

Основна гілка master містить стабільну версію застосунку.

Для розробки нових функцій (feature) створюється окрема гілка, де ці функції розробляються та тестуються.

Після завершення розробки та тестування, гілка з новою функціональністю зливається з головною гілкою master.

Цей підхід дозволяє ефективно управляти розробкою та забезпечує високу якість остаточного продукту.

Приклад гілок проекту зображено на рисунку 3.1 .

Гілки були створені відповідно до вибраної стратегії гілкування "feature

КПЗ–2024

branching", де кожна гілка відповідає певній функціональності, яка розробляється. Ось основні гілки та їх призначення:

- feature-personal-info: призначена для розробки особистого кабінету користувача та сторінки, на якій можна редагувати інформацію про користувача. Ця функціональність забезпечує користувачам можливість управління своїми персональними даними;

- feature-schedule: використовується для розробки сторінки з розкладом заходів на майданчику. Ця гілка дозволяє користувачам переглядати актуальний розклад і планувати свій час;

- feature-shared: містить загальні компоненти, які використовуються в різних частинах додатку. Розробка такої гілки сприяє уникненню дублювання коду і полегшує подальше масштабування додатку;

- feature-authorization: створена для розробки механізмів авторизації та реєстрації користувачів. Ця гілка забезпечує безпеку додатку та ідентифікацію користувачів;

- feature-saving-personal-info: використовується для оновлення особистої інформації користувачів. Це дозволяє користувачам вносити зміни до своїх даних, що покращує їх досвід користування;

- feature-tickets: призначена для створення сторінки з квитками та реалізації функцій повернення квитків. Користувачі можуть керувати своїми квитками, включаючи їх купівлю та повернення.

Кожна з цих гілок розробляється незалежно, що дозволяє ефективно розподіляти роботу серед команди та мінімізувати конфлікти під час злиття коду. Після завершення розробки та тестування, зміни з кожної гілки зливаються з головною гілкою master, забезпечуючи постійне оновлення робочої версії застосунку.

КПЗ - 2024

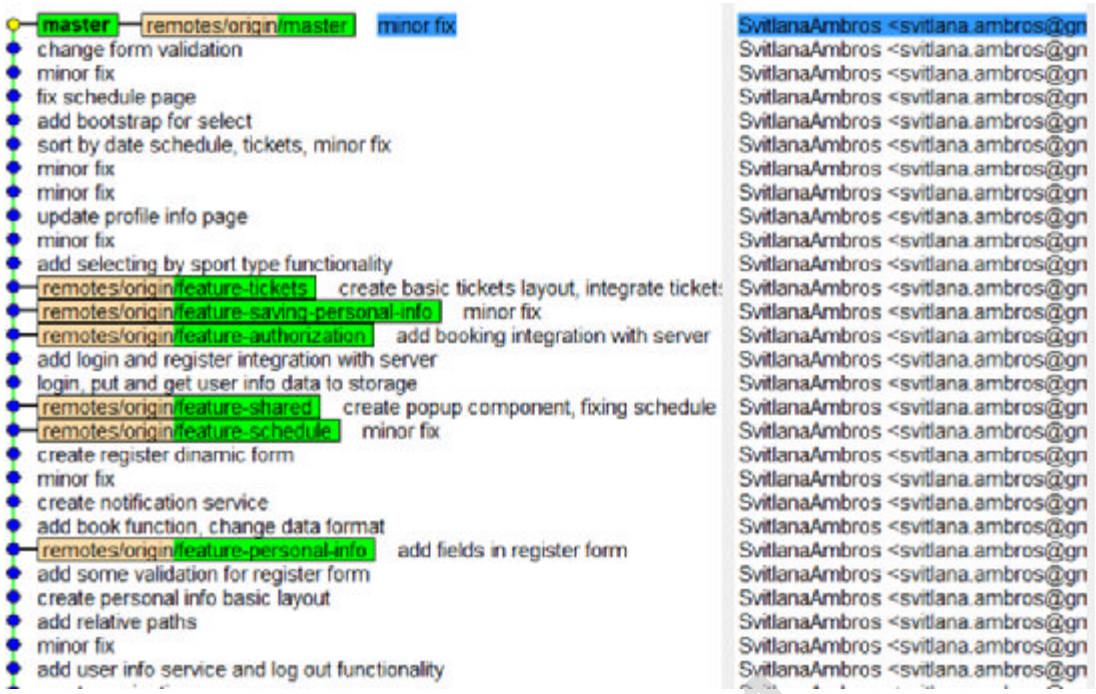

Рисунок 3.1 — Feature branching strategy

Менеджер пакетів NPM

NPM (Node Package Manager) використовується як менеджер пакетів для роботи з JavaScript і TypeScript, а також для інтеграції різних бібліотек, зокрема Angular. Цей інструмент застосовується для завантаження необхідних пакетів та локального запуску проектів.

Ось приклад команди, яка дозволяє глобально встановити пакети на комп'ютер:

npm install -g @angular/cli

Ця команда дозволяє встановити Angular CLI глобально, що полегшує створення та керування Angular-проектами.

Для додавання бібліотеки до проекту використовується команда з флагом --save-dev. Це додає залежність до розділу devDependencies у файлі package.json, забезпечуючи таким чином керування версіями бібліотек для розробки:

- npm install <назва_пакету> --save-dev

Після клонування проекту з репозиторію необхідно виконати команду:

- npm install

КОПІЯ_2024

Ця команда завантажує всі залежності, вказані в package.json, які необхідні для нормальної роботи проекту.

Для запуску проекту локально використовується:

- npm run start

Це команда запускає додаток на localhost:4200.

Додаткові скрипти можуть бути прописані у файлі package.json. Для запуску цих скриптів використовується команда:

- npm run <назва_команди>

Це особливо корисно, коли потрібно виконати складні команди з багатьма параметрами, уникнути повторення та спростити робочий процес.

Інші існуючі команди прописані також в package.json файлі та можуть доповнюватись. При цьому прописують назву команди та її опис, для її запуску потім використовується npm run <назва створеної команди>, що може бути зручним у випадку якщо ми додаємо безліч флагів та дозволяє не дублювати великі команди вручну в командному рядку.

Мова програмування TypeScript

JavaScript традиційно використовується для створення веб-додатків, але ця мова може стати важкою для управління в рамках більших проектів. Однією з основних проблем є відсутність статичної типізації, що ускладнює визначення типів даних, особливо при роботі з складними об'єктами і великими JSON-структурами. Це може призвести до складнощів у розумінні структури даних і відловлюванні помилок на ранніх етапах розробки.

TypeScript, як розширення JavaScript, вирішує ці проблеми, додаючи можливість статичного типування. Таке розширення дозволяє розробникам явно вказувати типи змінних, параметрів функцій та повернення значень, забезпечуючи більш строгу структуру і зрозумілість коду. TypeScript підтримує сучасні концепції об'єктно-орієнтованого програмування, включаючи класи і інтерфейси, що сприяє кращій організації коду і підтримці.

Компіляція TypeScript у JavaScript означає, що код на TypeScript можна

КПЗ-2024

запускати в будь-якому сучасному браузері або на серверній платформі Node.js. Це забезпечує гнучкість у використанні і розширює можливості інтеграції різних частин проекту.

Завдяки TypeScript, багато помилок виявляються і виправляються під час компіляції, що зменшує кількість помилок на виробництві і спрощує процес розширення проекту. Це робить код більш надійним і зрозумілим для нових розробників, сприяючи ефективній командній роботі та масштабуванню проекту.

Angular фреймворк

JavaScript є основною мовою для розробки веб-застосунків, але при розробці великих застосунків можуть виникнути труднощі через його низькорівневі можливості. Наприклад, для обробки подій у JavaScript потрібно визначити елемент, призначити йому обробник подій і тільки після цього вказати логіку реакції на ці події. Цей процес є повторюваним і часомістким, особливо коли розробникам необхідно зосередитись на більш важливих аспектах обробки подій.

Тому було вирішено використовувати Angular — сучасний фреймворк, який дозволяє побудувати інтерактивні та динамічні веб-додатки, значно спрощуючи вирішення згаданих проблем.

Angular використовує HTML як мову шаблонів, розширюючи його синтаксис для виразу компонентів програми. Це дозволяє значно зменшити кількість коду, який би довелося написати за допомогою чистого JavaScript, завдяки прив’язці даних і впровадженню залежностей.

Особливості Angular, які вплинули на вибір цього фреймворку:

1) Використання TypeScript як основи: Angular використовує TypeScript, який додає статичне типування і об'єктно-орієнтовані можливості, що покращує розробку і підтримку коду.

2) SPA-підхід (Single Page Application): Angular ідеально підходить для розробки SPA, де взаємодія з сервером і відображення результатів відбувається без перезавантаження сторінки, що підвищує швидкість і плавність роботи

КОПЗ-2024

застосунків.

3) Клієнтська MVC-інфраструктура: Angular надає структуру, яка дозволяє легко розділити представлення, бізнес-логіку, і моделі даних, сприяючи чистоті і масштабованості коду.

4) Сумісність з різними браузерами: Angular компілює код з TypeScript в JavaScript, який може виконуватися в будь-якому сучасному браузері.

5) Проста маршрутизація: Angular має потужну і зрозумілу систему маршрутизації, що полегшує створення навігації в односторінкових застосунках.

6) Розширення HTML-синтаксису: Angular дозволяє легко створювати повторно використовувані компоненти, використовуючи директиви, що збагачує структуру HTML.

7) Підтримка препроцесорів стилів: З можливістю використання Sass та інших CSS-препроцесорів, Angular спрощує стилізацію застосунків.

Angular дозволяє швидко створювати високопродуктивні, масштабовані веб-додатки, які є простими в обслуговуванні та розширенні. Використання цього фреймворку значно покращує сприйняття і користувацький досвід веб-додатків.

Постачальник модулів Webpack

При розробці великих веб-додатків на TypeScript та Angular, ви часто маєте справу з багатьма модулями і залежностями. Ці модулі та залежності повинні бути коректно об'єднані та оптимізовані для браузера, адже веб-браузери не розуміють TypeScript та не можуть напряму імпортувати ES6 модулі з файлів.

Webpack є потужним інструментом (бандлером), який вирішує цю проблему, збираючи всі модулі та їхні залежності у єдиний файл (або декілька файлів, якщо це потрібно для оптимізації). Це спрощує завантаження та виконання вашого додатку в браузері.

Ключові Властивості Webpack

Автоматична Компіляція і Бандлінг: Webpack автоматично компілює TypeScript в JavaScript, обробляє інші ресурси (наприклад, CSS та зображення) і об'єднує їх у мінімізовані бандли.

КПЗ_2024

Управління Залежностями: Webpack визначає, які модулі та бібліотеки використовуються в вашому додатку, і включає їх у фінальний бандл у правильному порядку.

Loaders і Plugins: Ви можете використовувати різні loaders для трансформації файлів перед їхнім додаванням до бандлу. Наприклад, ts-loader або babel-loader для JavaScript. Plugins можуть бути використані для розширення функціональності Webpack, наприклад, для оптимізації бандлів або автоматичного створення HTML файлів.

Модульний Підхід: Webpack дозволяє розділити ваш код на кілька бандлів за допомогою Code Splitting, що може значно покращити час завантаження застосунку.

HMR (Hot Module Replacement): Webpack може замінювати, додавати або видаляти модулі, поки застосунок працює, без необхідності повного перезавантаження сторінки.

Оптимізація для Продакшн: При підготовці додатку до розгортання на продакшн, Webpack може мінімізувати та оптимізувати бандли для найкращої продуктивності (рисунок 3.2).

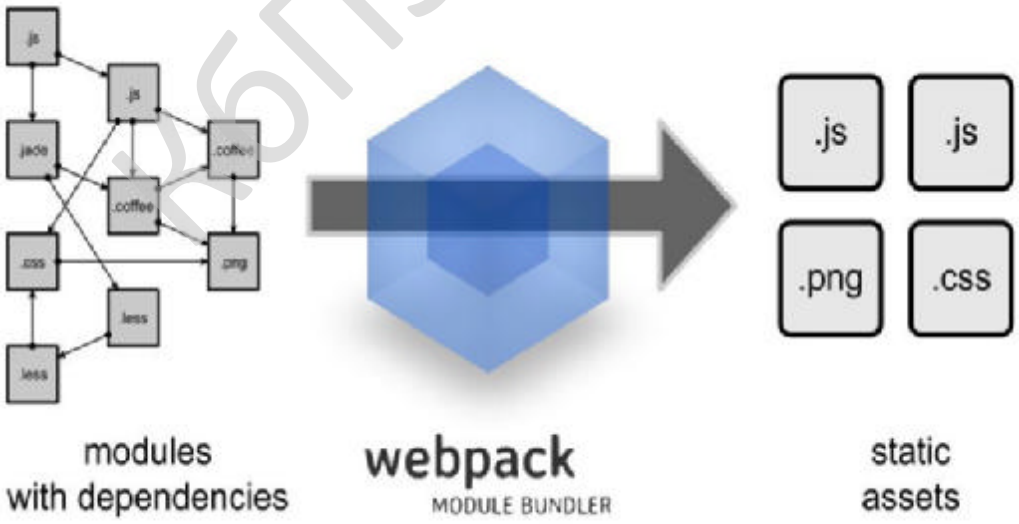

Рисунок 3.2 — Webpack

Webpack є надзвичайно потужним інструментом для модернізації та

ВКПЗ_2024

оптимізації веб-додатків. Цей інструмент допомагає автоматизувати багато рутинних процесів, забезпечуючи ефективнішу розробку та розгортання проектів.

Детальніше про процеси, які Webpack виконує з вашим додатком:

1) Переведення ES6 в ES5: Webpack використовує Babel (через babel-loader) для трансформації сучасного JavaScript (ES6+) в код, який сумісний зі старішими браузерами (ES5). Це гарантує, що ваші скрипти працюватимуть коректно навіть у старих версіях браузерів.

2) Переведення SCSS в CSS: За допомогою sass-loader (частина набору Webpack loaders), Webpack може перетворювати SCSS або SASS файли у звичайний CSS. Це спрощує роботу з стилями, дозволяючи використовувати зручніші для розробника функції SCSS.

3) Мініфікація файлів: Webpack може використовувати TerserPlugin, cssnano через css-minimizer-webpack-plugin, або інші плагіни для мініфікації JavaScript і CSS файлів. Це значно зменшує розмір вихідних файлів, покращуючи час завантаження додатку.

4) Видалення коментарів: В процесі мініфікації, Webpack також може видалити коментарі, що дозволяє додатково зменшити розмір файлів.

5) Додавання шрифтів і іконок: За допомогою лоадерів, як file-loader або url-loader, Webpack може інкорпорувати шрифти і іконки прямо в CSS або JavaScript файли, що зменшує кількість HTTP запитів до сервера.

6) Об'єднання CSS файлів: Використовуючи MiniCssExtractPlugin, Webpack може об'єднати всі CSS файли в один бандл, що зменшує кількість завантажень та покращує швидкість рендерингу сторінки.

7) Автоматичне додавання тегів <link> та <script>: За допомогою HtmlWebpackPlugin, Webpack може автоматично вставляти теги <link> та <script> у ваш HTML файл, вказуючи на створені бандли. Це виключає необхідність ручного оновлення HTML файлу при кожному зміненні версій файлів.

Після збірки проекту отримуємо єдиний js, css, html файл.

При використанні Webpack для створення додатка Angular створюється

КОПЗ-2024

package.json файл, який виглядає як на рисунку 3.3.

Рисунок 3.3 — Файл package.json

Файл tsconfig.json є важливим елементом у проектах, які використовують TypeScript. Він визначає, як компілятор TypeScript повинен обробляти код TypeScript і перетворювати його в JavaScript. Цей файл дає можливість тонко налаштовувати різні аспекти компіляції, включаючи цільову версію ECMAScript, модулі, строгість типізації та багато інших опцій.

Основні аспекти tsconfig.json

1) compilerOptions: Це основний розділ, де задаються налаштування компілятора. Тут можна вказати версію ECMAScript, модулі, опції строгого режиму, директорії виводу та інші важливі параметри.

2) include і exclude: Ці поля дозволяють вказати, які файли або директорії слід включити або виключити з процесу компіляції. Вони допомагають керувати тим, які частини проекту повинні оброблятися компілятором.

3) files: Якщо потрібно явно вказати окремі файли, які мають бути включені до компіляції, можна використати цей параметр.

4) extends: Використовується для наслідування конфігурації з іншого tsconfig.json файла. Це корисно в великих проектах зі спільними налаштуваннями.

КОПЗ_2024

5) typeRoots і types: Вказують де шукати типи і які типи включати. Це корисно для управління зовнішніми оголошеннями типів. (рисунок 3.4).

Рисунок 3.4 — Файл tsconfig.json

Файл package.json є ключовим у будь-якому проекті, що використовує Node.js та npm. Він містить метадані проекту, визначає залежності, а також містить скрипти для виконання різних задач.

BevDependencies: Ця секція містить пакети, які необхідні тільки для розробки і тестування. Ці пакети не будуть включені до продуктивного середовища. Приклади таких пакетів можуть включати typescript, webpack, і webpack-dev-server.

scripts: Це секція, де ви можете визначити скрипти для запуску різних задач.

Наприклад:

- "start": Цей скрипт зазвичай запускає локальний розробницький сервер, наприклад webpack-dev-server, що дозволяє розробникам побачити зміни в режимі реального часу;

- "build": Цей скрипт використовується для збірки проекту, тобто компіляції TypeScript коду в JavaScript, об'єднання модулів і мініфікації за

КПІВ_2024

допомогою Webpack;

- tsconfig.json: Файл конфігурації TypeScript, який вказує компілятору, як обробляти код TypeScript.

Важливі параметри включають:

- "baseUrl": Встановлює базовий шлях для відносних імпортів;
- "paths": Визначає аліаси для імпортів, дозволяючи вказувати коротші шляхи до модулів;
- "outDir": Вказує директорію, куди будуть поміщені вихідні файли після компіляції.

Препроцесор стилів Sass

При написанні CSS коду часто виникає ситуація, коли таблиці стилів стають надзвичайно великими та складними для управління. У таких випадках на допомогу приходить препроцесор Sass, який додає нові можливості до чистого CSS, такі як змінні, вкладені правила, міксини, спадкування та інші функції, які роблять процес розробки стилів більш ефективним і зручним.

Починаючи використовувати Sass, препроцесор обробляє файл Sass і перетворює його в стандартний CSS файл, який можна без проблем використовувати на будь-якому веб-сайті. Найпростіший спосіб здійснити це перетворення - використовувати командний рядок. Після інсталяції Sass, ви можете компілювати ваші Sass файли в CSS, використовуючи команду sass. Також можливо вказати директорії для моніторингу змін і збереження скомпільованих CSS файлів, використовуючи шляхи, розділені двокрапкою. Перетворення Sass в CSS забезпечується за допомогою webpack.

Змінні в Sass — це засіб для збереження інформації, яку можна повторно використовувати у всьому проекті. Наприклад, можна зберігати кольори, шрифти або будь-які CSS значення, які часто використовуються, в змінних. Щоб створити змінну в Sass, ви використовуєте символ $, який допомагає легко ідентифікувати і використовувати змінні у стилях [20].

Приклад використання змінних демонструється на рисунку 3.5, де змінні

КОПЗ-2024

для основних кольорів задаються на початку. Ці змінні потім використовуються по всьому додатку. Це надзвичайно зручно, адже у випадку необхідності змінити колір в дизайні, змінити його потрібно лише в одному місці — у файлі з змінними, замість того щоб переписувати цей колір в кожному місці стилів, де він використовується.

Цей підхід значно спрощує управління стилями і робить код більш читабельним і легшим для підтримки. Замість того, щоб розкидати кольори по різних селекторах, усі важливі кольори визначаються на початку у вигляді змінних. Якщо уявити, що ми хочемо змінити основний колір теми, достатньо змінити його значення в одній змінній, і ця зміна автоматично застосується до всіх стилів, де використовується ця змінна.

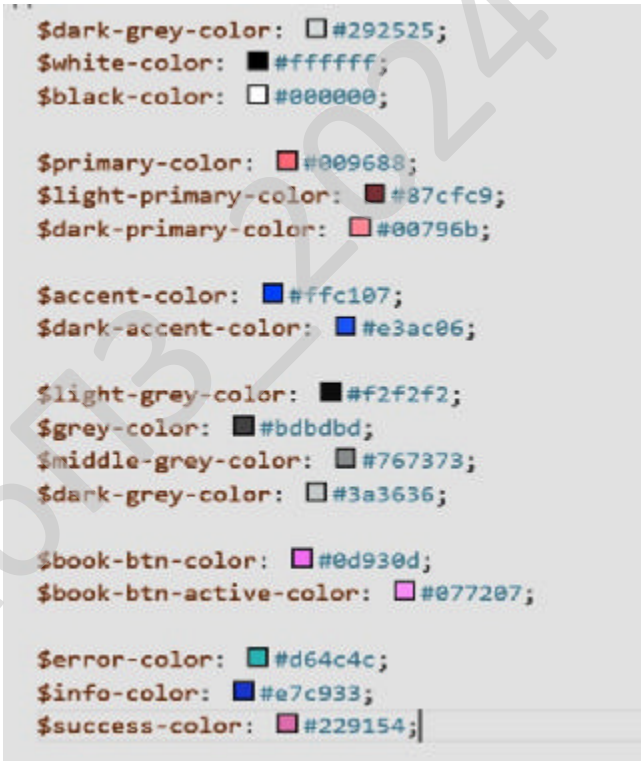

Рисунок 3.5— Файл tsconfig.json

Під час написання HTML коду можна спостерігати ясну вкладену і візуальну ієрархію, яка відображає структуру документа. Однак, в стандартному CSS подібна ієрархія втрачається, оскільки каскадність і специфічність селекторів не завжди дозволяють зберегти візуальну ієрархію.

Копія 2024

Sass вносить зміни у цю парадигму, дозволяючи вкладати CSS селектори в тому ж порядку, в якому вони вкладені у візуальній ієрархії HTML. Це означає, що ви можете використовувати вкладеність селекторів у Sass для мімікрії структури HTML, що робить стилі більш читабельними та логічно організованими.

Важливо пам'ятати, що хоча Sass дозволяє глибоко вкладати селектори, надмірне використання цієї можливості може зробити ваші стилі менш читабельними та важчими для підтримки. Ідеальною практикою є збереження помірної кількості вкладеності, що покращує структуру стилів без додавання зайвої складності.

На рисунку 3.6, можна побачити, як Sass використовується для демонстрації ієрархії та вкладеності стилів. Використання такої структури значно зменшує кількість дубльованого коду та робить стилі більш читабельними. Замість того, щоб описувати кожен селектор окремо, ви можете групувати селектори та їхні стилі за допомогою вкладеності.

При написанні HTML структури, очевидна вкладена та візуальна ієрархія елементів значно полегшує читання та розуміння коду. Втім, в чистому CSS така ієрархія часто втрачається, оскільки стилі описуються окремо і не завжди зберігають візуальну вкладеність. Sass пропонує рішення цієї проблеми, дозволяючи використовувати вкладеність селекторів, яка відображає візуальну ієрархію HTML.

Ця можливість Sass робить стилі більш організованими та інтуїтивно зрозумілими, адже вкладати стилі безпосередньо відповідно до структури HTML. Це значно спрощує процес розробки, оскільки зміни в одному місці стилів можуть автоматично відобразитись у всіх вкладених елементах, без потреби копіювати або повторювати код.

На рисунку 3.6 можна побачити як використовується ієрархія та вкладеність стилів за допомогою Sass. Використання таких вкладень допомагає зменшити загальний обсяг коду, уникаючи дублювання, і робить стилі більш

КПЗ - 2024

читабельними та легшими для підтримки.

З використанням Sass, замість того, щоб повторювати одні й ті ж стилі для кожного елемента, можна встановити змінні для кольорів, шрифтів чи інших CSS властивостей один раз, а потім застосовувати їх через всі стилі. Це не тільки підвищує ефективність розробки, але й робить процес змін більш швидким та менш схильним до помилок — змінюючи значення в одній змінній, ви автоматично змінюєте його у всіх місцях, де ця змінна використовується.

Це особливо корисно, коли потрібно зберегти єдиний стиль теми або палітри кольорів, що є частою потребою в багатьох проектах. Використання вкладеності в Sass дозволяє легко адаптувати і масштабувати ваші стилі без необхідності розширення коду і введення додаткової складності.

У CSS можна використовувати директиву @import для розділення стилів на кілька файлів, що спрощує організацію і підтримку коду. Однак, кожен @import у CSS створює додатковий HTTP-запит до сервера, що може знижувати швидкість завантаження сторінки, особливо при великій кількості імпортованих файлів.

Sass підходить до проблеми імпорту трохи інакше, що значно підвищує ефективність. Використовуючи ту саму директиву @import, Sass вставляє вміст імпортованих файлів прямо в той файл, де виконується імпортування, до того, як весь CSS буде компільований. Це означає, що у вихідному CSS-файлі вже міститься весь необхідний код, і додаткові HTTP-запити не потрібні.

Ця особливість Sass дозволяє розробникам використовувати модульний підхід у написанні стилів, зберігаючи організованість і читабельність коду, але без негативного впливу на продуктивність завантаження сторінки. Таким чином, Sass полегшує управління стилями і підвищує швидкість відображення веб-сторінок.

КПЗ - 2024

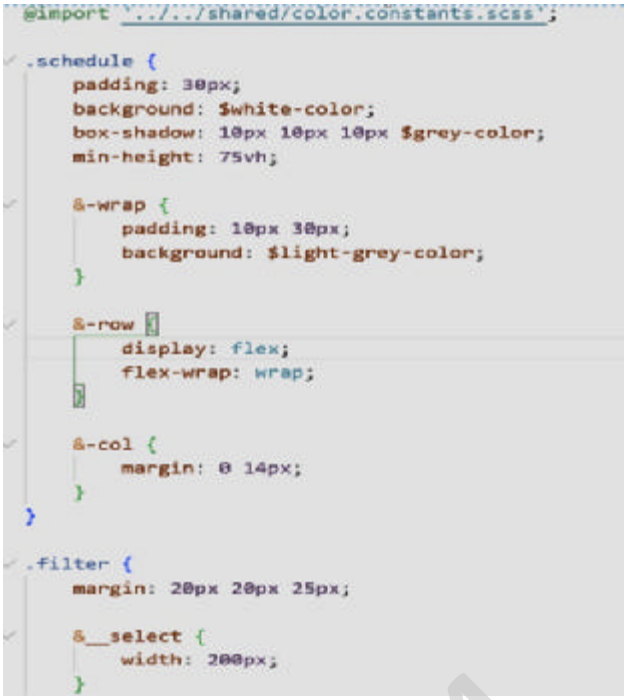

Рисунок 3.6— Стилі в Sass файлі

Використання Sass дозволяє зібрати стиль проекту у єдиний CSS-файл, скомпільований з різних частин. Це робиться за допомогою директиви @import, яка в Sass використовується для об'єднання різних файлів стилів у один загальний файл без створення додаткових HTTP-запитів. Цей підхід значно полегшує управління стилями і робить кінцевий CSS більш оптимізованим.

Коли імпортуємо змінні, міксини, чи інші фрагменти коду з різних файлів, Sass обробляє ці імпорти і включає весь необхідний код у фінальний скомпільований CSS-файл. Таким чином, виходить один CSS-файл, який містить усю інформацію з різних фрагментів, зібраних разом.

На рисунку 3.6, приклад імпорту ілюструє, як змінні з константами імпортуються для подальшого використання у файлі стилів. Це демонструє, як можна визначити основні стилі, кольори, шрифти, і інші параметри в окремих файлах, а потім легко імпортувати та використовувати їх у головних стилевих файлах.

Реактивне програмування RxJS

Реактивне програмування — це стиль програмування, який працює з

КПЗ-2024

асинхронними потоками даних, забезпечуючи високий рівень абстракції. Цей підхід дозволяє розробникам зосередитися на взаємозв'язках між подіями, які формують бізнес-логіку, замість витрачання часу на керування складними деталями імплементації. Реактивне програмування також спрощує код, роблячи його більш стислим та зручним для читання.

У контексті розробки веб-додатків, реактивне програмування ефективно вирішує проблему обробки різних UI-подій. Наприклад, натискання на кнопку може ініціювати цілий ланцюг дій: відправку запиту на сервер, отримання даних, їх фільтрацію за певними критеріями, та виведення результатів користувачеві з опцією бронювання події. Визначення і виконання такої послідовності дій, особливо після отримання даних з сервера, стає значно простішим з реактивним програмуванням. Без асинхронності такі події могли б заблокувати інтерфейс користувача, погіршуючи користувацький досвід.

Для роботи з асинхронними та подієзорієнтованими програмами використовується бібліотека RxJS. Ця бібліотека використовує основний тип Observable, а також допоміжні типи, такі як Observer, Schedulers, Subjects. Вона також надає оператори для роботи з подіями аналогічно до методів роботи з колекціями в JavaScript, такі як map, filter, reduce, every та інші.

Observable — це тип, який дозволяє створювати спостережувані об'єкти, за якими можна слідкувати відповідно до шаблону "Спостерігач". Він виступає як об'єкт, що може виробляти множинні значення або події, на відміну від Promise, який обробляє лише одне значення. Функції, які визначаються під час підписки на Observable, називаються спостерігачами. Вони описують, як програма має реагувати на зміни в Observable і записуються через метод subscribe (рисунок 3.7).

КПЗ - 2024

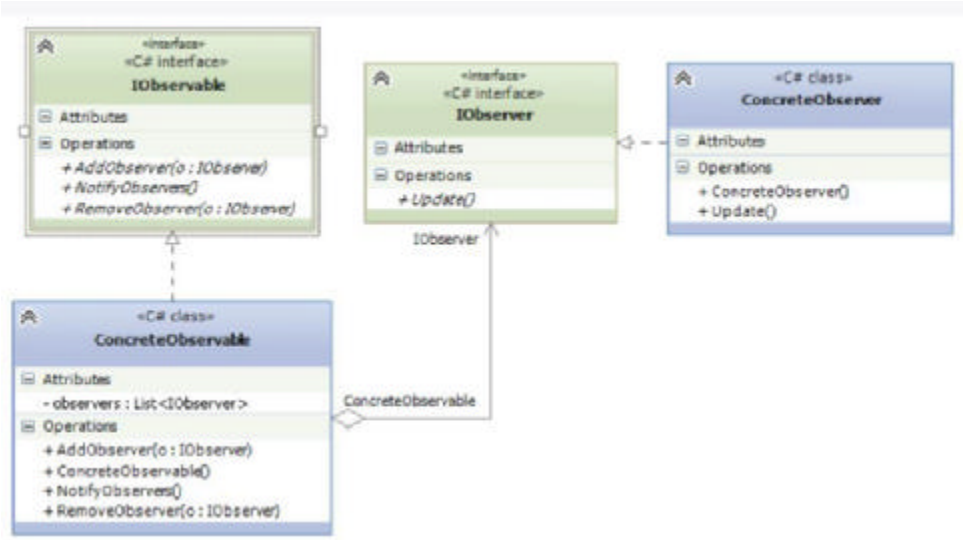

Рисунок 3.7— Патерн “Спостерігач”

Функція підписки в реактивному програмуванні має три параметри: перший — це функція, яка виконується при отриманні даних, другий — функція для обробки помилок, і третій — функція, яка викликається при завершенні потоку даних. Ця структура дозволяє чітко визначити, як програма повинна реагувати на різні ситуації при роботі з асинхронними даними.

На рисунку 3.8 показано приклад реактивного програмування у веб-додатку. Використовуючи HttpClient, який є частиною ангулярівського фреймворку, можна робити запити до сервера. HttpClient повертає об'єкт Observable, на який можна підписатися, щоб реагувати на зміни даних, отримані в результаті запиту.

Оператори, такі як map, filter, та sort, використовуються для трансформації потоку даних після їх отримання. Вони працюють подібно до методів масивів у JavaScript, дозволяючи виконувати операції над кожним елементом потоку і повертати трансформований потік даних для подальшої обробки або відображення перетворювати потік після отримання результатів після чого повертають його.

КОПЗ_2024

```
constructor(private http: HttpClient) { }

public getScheduleForPeriod(sportType: string): Observable<ScheduleDay[]> {
  if (sportType !== 'All') {
    return this.http.get<ScheduleDay[]>(GET_SCHEDULE_FOR_PERIOD).pipe(
      map((data: ScheduleDay[]) => {
        return data.map((item: ScheduleDay) => {
          return { ...item, dateUI: this.getDateFromString(item.date) };
        })
          .sort((a: ScheduleDay, b: ScheduleDay) => a.dateUI.getTime() - b.dateUI.getTime())
          .map((day: ScheduleDay) => {
            return {
              ...day,
              events: day.events
                .filter((event: ScheduleEvent) => event.sportType === sportType)
            };
          })
          .filter((day: ScheduleDay) => day.events.length > 0);
      })
    );
  }
```

Рисунок 3.8— Використання RxJS

У цьому прикладі HttpClient повертає об'єкт Observable, який асинхронно отримує дані від сервера. Отриманий потік даних перетворюється та фільтрується відповідно до вибраного виду спорту, і сортується за датами, після чого формується новий потік даних. Якщо вид спорту не вибрано, то користувачеві повертається повний розклад, як він прийшов від сервера.

Використовуючи цей реактивний підхід, у сервісі ми повертаємо Observable і це дозволяє компонентам підписуватися на цей потік за допомогою методу subscribe і отримувати дані вже відсортовані та перетворені, готові до відображення у користувацькому інтерфейсі.

Додавання асинхронності через Observable дозволяє підписуватися на потік даних без блокування головного потоку виконання, що робить додаток більш відгуковим і забезпечує можливість продовжувати роботу з інтерфейсом, поки дані завантажуються. Коли результати будуть отримані, спрацьовує callback-функція, яка обробляє ці дані. В методі subscribe також можна обробляти помилки та виводити відповідні повідомлення для користувача.

Підхід з реактивним програмуванням значно спрощує управління потоками даних, робить код менш складним і більш зрозумілим, а також покращує взаємодію користувача з додатком, роблячи його інтерфейс більш відгуковим і зручним для використання.

КПЗ-2024

3.1 Опис функціонування системи

Система e-service є інформаційною системою, яка надає можливість використання різноманітних послуг через Інтернет.

Основними складовими функціонування такої системи є платформа, інтерфейс користувача, база даних та модулі реєстрація та авторизація користувачів.

Перед використанням сервісів користувачі мають зареєструватися в системі, заповнивши необхідні дані. Після цього вони можуть авторизуватися на платформі за допомогою своїх облікових записів.

Пошук та вибір послуг. Система надає користувачам можливість шукати та вибирати потрібні послуги з веб-інтерфейсу. Це може бути різноманітні послуги, такі як покупка товарів, бронювання подорожей, оплата рахунків, отримання інформації тощо.

Оформлення замовлення та оплата. Користувачі можуть оформляти замовлення на обрані послуги через систему, вказуючи необхідні дані та параметри. Після цього вони можуть здійснити оплату за допомогою різних платіжних методів, які підтримує система.

Виконання послуги або доставка товару. Після отримання замовлення, відповідні дії виконуються системою або її партнерами. Це може включати доставку товару, надання послуги, опрацювання запиту або будь-яку іншу відповідну дію.

Відстеження статусу замовлення. Користувачі мають можливість відстежувати статус свого замовлення через особистий кабінет на платформі. Вони можуть переглядати інформацію про стан обробки замовлення, очікуваний час доставки та інші деталі.

Зворотній зв'язок та обслуговування користувачів. Система надає можливість зв'язку зі службою підтримки або операторами через онлайн-чат, електронну пошту або телефон. Це дозволяє користувачам отримувати допомогу

КПЗ-2024

та вирішувати свої питання.

Збір та аналіз даних. Система збирає дані про користувачів та їхні замовлення для подальшого аналізу та використання в різних цілях, таких як покращення обслуговування, рекламні кампанії та ін. бробки даних.

3.2 Розробка структурної схеми

Структурна схема системи e-service складається з різних компонентів, які спільно працюють для надання користувачам можливості використання різноманітних послуг через Інтернет.

Користувацький інтерфейс (User Interface)

Користувацький інтерфейс є основним інтерфейсом між користувачем і системою e-service. Він має за задачу забезпечити зручну та інтуїтивно зрозумілу взаємодію користувача з системою. Користувацький інтерфейс може бути реалізований у вигляді веб-сайту, мобільного додатка або десктопної програми. Він повинен бути добре структурованим та забезпечувати простий доступ до різних функцій та послуг системи.

База даних (Database)

База даних використовується для зберігання і організації різних видів інформації, яка використовується в системі e-service. Це може включати дані про користувачів, їхні замовлення, історію транзакцій, каталог товарів або послуг, налаштування системи та інше. База даних може бути реалізована з використанням різних технологій, таких як реляційні СУБД (наприклад, MySQL, PostgreSQL) або нереляційні СУБД (наприклад, MongoDB).

Бізнес-логіка (Business Logic)

Бізнес-логіка системи e-service визначає правила та процедури, за якими працює система. Вона відповідає за контроль логіки обробки запитів користувачів, валідацію даних, проведення бізнес-процесів та взаємодію з іншими компонентами системи. Бізнес-логіка може бути реалізована у вигляді програмного коду, який

КПЗ_2024

виконується на серверній стороні системи.

Система управління користувачами (User Management System)

Ця система відповідає за управління користувачами системи e-service. Вона забезпечує функціонал для реєстрації нових користувачів, авторизації, аутентифікації, керування правами доступу, відновлення паролів та інші операції, пов'язані з управлінням обліковими записами користувачів.

Кожен з цих компонентів відіграє ключову роль у функціонуванні системи e-service, забезпечуючи її стабільну та ефективну роботу, а також зручність для користувачів. Їхнє взаємодія дозволяє системі надавати різноманітні послуги через Інтернет у найефективніший спосіб.

У структурній схемі (рисунок 3.9) позначені зовнішні специфікації програми, які дають розуміння функціональній частині програми, за що вони відповідають, та як вони взаємодіють з вхідними та вихідними даними. Особливо важливо розуміти тип структури даних.

КбПЗ_2024

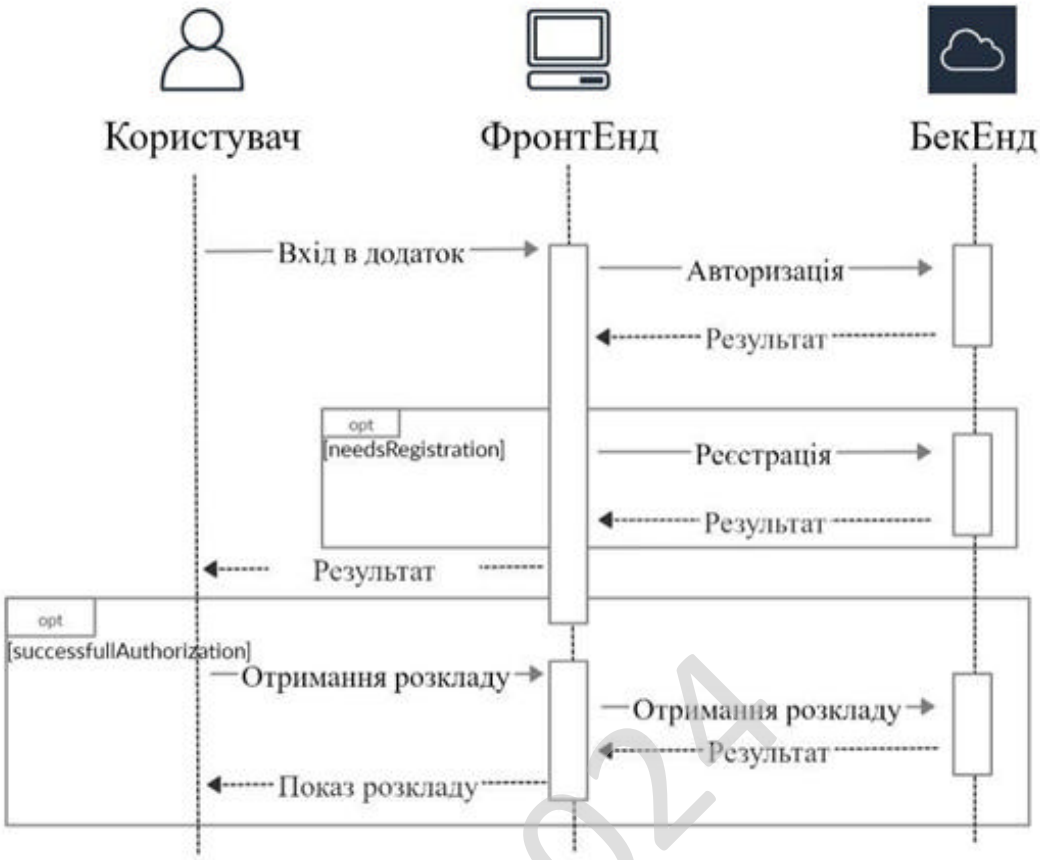

Рисунок 3.9 - Структурна схема

3.3 Розробка функціональної схеми

Функціональна схема системи e-service відображає основні функції та процеси, які відбуваються в системі під час надання користувачам доступу до різних послуг через Інтернет.

Реєстрація та авторизація

Функція: Користувачі можуть створювати облікові записи, вказуючи необхідну інформацію, таку як ім'я, електронна адреса, пароль тощо. Після реєстрації користувачі мають можливість авторизуватися в системі, використовуючи свої облікові дані.

Взаємозв'язки: Після реєстрації користувач має доступ до свого облікового запису, де може здійснювати різні дії, такі як перегляд історії замовлень, зміна особистих даних тощо.

Пошук та вибір послуг

Функція: Користувачі можуть шукати та переглядати доступні послуги

КПЗ_2024

через систему e-service за допомогою різних критеріїв, таких як категорія, ключові слова, ціновий діапазон тощо.

Взаємозв'язки: Користувач може вибрати бажану послугу та перейти до процесу оформлення замовлення.

Оформлення замовлення та оплата

Функція: Після вибору послуги користувач може оформити замовлення, вказавши необхідні деталі, такі як кількість, адреса доставки тощо. Після цього він може здійснити оплату за допомогою різних платіжних методів.

Взаємозв'язки: Після успішної оплати користувач отримує підтвердження замовлення та інформацію про дату та спосіб доставки.

Виконання послуги або доставка товару

Функція: Після отримання замовлення від користувача система e-service організує виконання послуги або доставку товару за вказаною адресою.

Взаємозв'язки: Після виконання послуги або доставки товару користувач отримує сповіщення про завершення операції.

Зворотній зв'язок та обслуговування користувачів

Функція: Система надає можливість зв'язку зі службою підтримки або операторами через онлайн-чат, електронну пошту або телефон. Це дозволяє **користувачам отримувати допомогу та вирішувати свої питання.**

Взаємозв'язки: Користувачі можуть звертатися за допомогою та отримувати відповіді на свої запитання через різні канали зв'язку.

КПЗ 2024

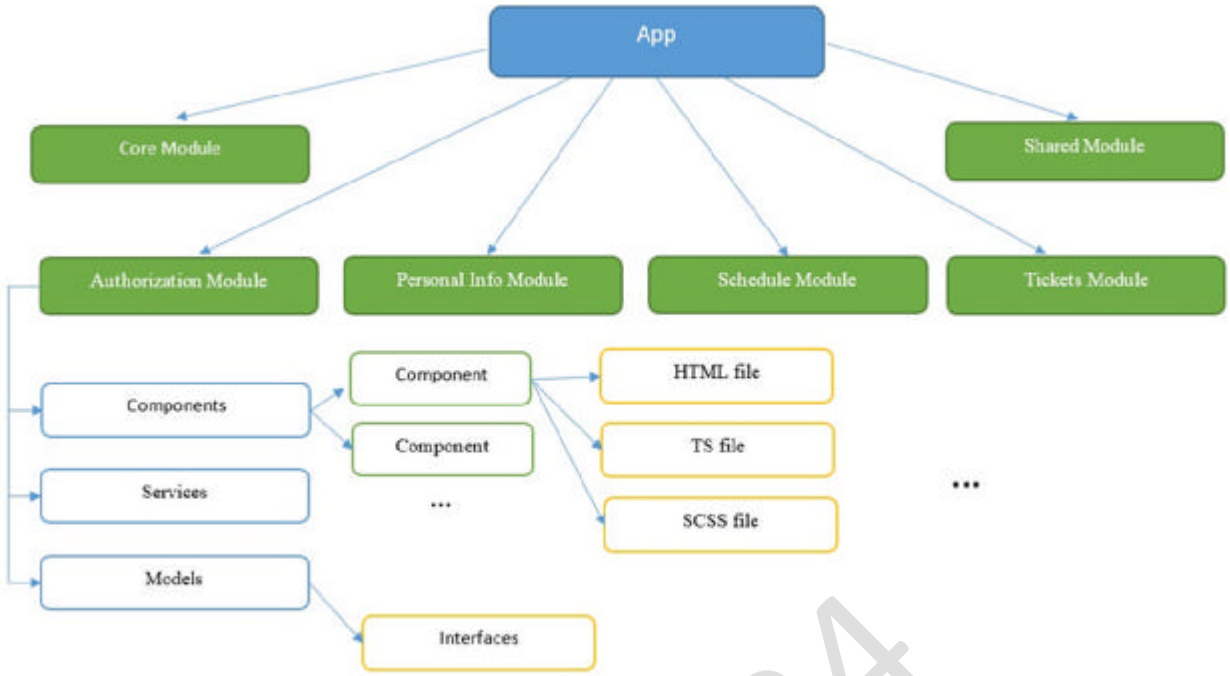

Рисунок 3.10 - Функціональна схема системи

3.4 Розробка діаграми процесів

Є різні методи побудови діаграм, які допомагають відстежити логіку та послідовність дії в процесах, але найкраще себе зарекомендував метод погрокового алгоритму побудови діаграм.

Діаграма процесів системи e-service

Початок процесу: Користувач запускає додаток або відкриває веб-сайт системи e-service.

Реєстрація / Авторизація: Користувач має можливість увійти в свій обліковий запис або зареєструватися, якщо він новий користувач.

Пошук послуги: Користувач шукає потрібну послугу за допомогою пошукового фільтра або категорій.

Вибір послуги: Користувач вибирає певну послугу зі списку результатів пошуку або каталогу.

КПЗ_2024

Оформлення замовлення: Користувач оформляє замовлення, вказуючи деталі, такі як кількість, адреса доставки тощо.

Оплата: Користувач здійснює оплату за допомогою обраного методу платежу (карта, електронний гаманець тощо).

Обробка замовлення: Система обробляє замовлення, перевіряє наявність товару або доступність послуги, виконує операції з оплатою.

Виконання послуги / доставка товару: Якщо потрібно, система організовує виконання послуги або доставку товару до зазначеної адреси.

Підтвердження завершення: Користувач отримує підтвердження про успішне завершення замовлення.

Зворотній зв'язок та підтримка: Користувач може звернутися до служби підтримки в разі потреби або залишити відгук щодо отриманих послуг.

Завершення процесу: Процес завершується після успішного оформлення та отримання послуги або товару користувачем.

КБПЗ_2024

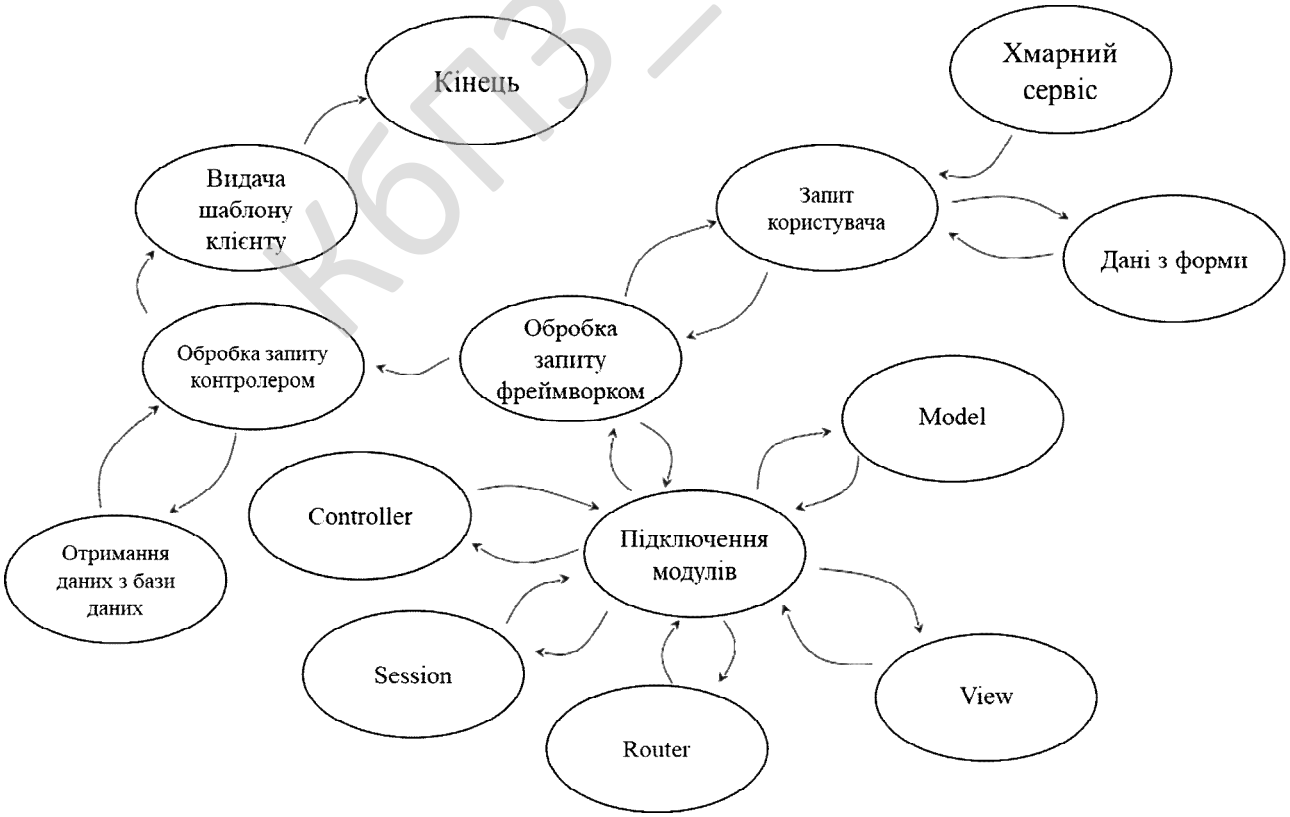

Рисунок 3.11 – Діаграма процесів системи

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

Програмна реалізація

Під час створення веб-додатку була обрана базова архітектура, яка включає модулі, компоненти та сервіси. Ці елементи є критично важливими для забезпечення чистоти коду та його ефективності. Щоб взаємодіяти з сервером, у проекті було вирішено використовувати архітектурний шаблон REST. Такий підхід дозволяє створювати легкозрозумілі та ефективні API для обміну даними між клієнтом та сервером.

Клієнт-серверна архітектура REST

Однією з ключових задач при створенні веб-додатку для спортивного закладу було забезпечення надання актуальної інформації користувачам. Для цього було обрано клієнт-серверну архітектуру, як показано на рисунку 4.1. У такій архітектурі стан ресурсу зберігається на сервері, а клієнти, підключаючись до сервера, отримують та зберігають локальний стан додатку, що дозволяє їм завжди мати доступ до останніх даних.

Для спілкування між клієнтом та сервером були розглянуті протокол та стандарт SOAP, а також архітектура REST. Після аналізу цих підходів було вибрано архітектуру REST з наступних причин:

- Незалежність від мови та платформи: REST може бути використаний з будь-якою мовою програмування та на будь-якій платформі, що забезпечує велику гнучкість у виборі технологій для клієнта та сервера.

- Повне використання можливостей HTTP: REST базується на стандартних методах HTTP, таких як GET, POST, PUT, DELETE тощо, використовуючи протокол HTTP як із його рідними можливостями, що підвищує ефективність і

КПЗ_2024

читабельність API.

- Простота і швидкість: В той час як SOAP використовує XML, що може сповільнити обробку даних, REST часто використовує більш легкі формати, такі як JSON, що робить обмін даними швидшим і спрощує роботу з отриманими даними.

- CRUD операції: REST надає чітко визначені оперції CRUD (create, read, update, delete), які дозволяють ефективно управляти ресурсами, роблячи архітектуру інтуїтивно зрозумілою та легкою для використання.

- Робота з ресурсами через URI: В архітектурі REST кожен ресурс є ідентифікованим за допомогою універсального ідентифікатора (URI), що робить веб-посилання натуральним способом адресації ресурсів.

- Визначеність дій над ресурсами: REST використовує обмежений набір чітко визначених операцій, що дозволяє легко визначити потрібні дії для створення, читання, оновлення, або видалення ресурсів.

Приклад Архітектури на REST

При створенні веб-додатку для спортивного закладу за допомогою архітектури REST, взаємодія між клієнтом і сервером може бути організована наступним чином:

- Запити до сервера: Клієнт виконує HTTP запити до сервера для отримання, створення, оновлення або видалення інформації про спортивні події. Наприклад, клієнт може виконати GET запит до /api/sport-events для отримання списку всіх подій.

- Обробка запитів на сервері: Сервер обробляє запит, взаємодіючи з базою даних або іншими сервісами, і повертає відповідь. Наприклад, сервер може повернути JSON масив з об'єктами подій, кожен з яких містить інформацію про одну подію.

- Відображення даних: Клієнт отримує відповідь від сервера і використовує ці дані для оновлення інтерфейсу користувача. Наприклад, додаток може відобразити отримані події у формі таблиці або списку на веб-сторінці.

КБПЗ-2024

Ця архітектура дозволяє розробникам створювати веб-додатки, які ефективно взаємодіють з сервером, забезпечуючи користувачам актуальну інформацію та високий рівень взаємодії з додатком.

Для розробки ефективних веб-служб з використанням передачі стану, важливо глибоко розуміти основи протоколу передачі гіпертексту (HTTP) і унікальних ідентифікаторів ресурсів (URI). Це дозволяє створити чіткі і оптимізовані інтерфейси для взаємодії з сервером, що є ключовим аспектом для успішного застосування архітектурного стилю REST.

HTTP Методи

У RESTful архітектурі кожен ресурс ідентифікується через URI, а взаємодія з цими ресурсами відбувається через стандартні HTTP методи. Ось основні методи, які я буду використовувати:

- GET: Використовується для отримання даних від сервера. GET запити повинні бути ідемпотентними, тобто їхнє повторне виконання не повинно впливати на стан сервера.
- POST: Використовується для створення нових ресурсів на сервері. POST запити можуть змінювати стан сервера, додаючи нові ресурси.
- PUT: Використовується для оновлення існуючих ресурсів. PUT запит повинен містити повну інформацію про ресурс, яка замінює поточний стан ресурсу на сервері.
- PATCH: Використовується для часткового оновлення ресурсу. Відмінно від PUT, PATCH змінює лише ті атрибути ресурсу, які були включені у запит.
- DELETE: Використовується для видалення ресурсів. DELETE запити видаляють ресурси з сервера.

У веб-додатку використання основних HTTP методів, таких як GET, POST, і PUT, є стандартним підходом для реалізації RESTful архітектури. Кожен з цих методів відіграє ключову роль у взаємодії з сервером та управлінні станом ресурсів. За допомогою Swagger документації ефективно інтегрувати клієнтську

КБПЗ-2024

та серверну частини додатку.

GET: метод використовується для запиту інформації з сервера. Він ідемпотентний, що означає, що повторні запити за тим же URL повернуть однакові результати без зміни стану сервера. Ваш додаток може використовувати GET для отримання даних, наприклад, списку спортивних подій, деталей про конкретну подію, або інших ресурсів.

POST: метод використовується для створення нового ресурсу на сервері. POST не є ідемпотентним, що означає, що кожний запит може змінювати стан сервера, додаючи новий ресурс. Ваш додаток може використовувати POST для створення нової спортивної події або інших сутностей.

PUT: метод використовується для оновлення існуючих ресурсів. PUT є ідемпотентним, а тому повторні запити з однаковими даними не змінюватимуть результат. Використовується для оновлення даних про існуючі спортивні події або інші ресурси.

Рисунок 4.1 — Приклад документаці серверних методів

При розробці веб-додатків, правильний вибір HTTP методів для різних операцій є ключовим для забезпечення безпеки, ефективності та інтуїтивно

КПЗ-2024

зрозумілого API. Ось як можна використовувати ці методи в контексті вашого додатку:

Метод POST для аутентифікації та реєстрації

- POST /login та /registration: застосування: ці ендпойнти використовуються для аутентифікації та реєстрації користувачів.
- Безпека: Використання POST для цих операцій дозволяє відправляти особисті дані (як-от логін і пароль) у тілі запиту (body), відокремлено від URL, що підвищує безпеку порівняно з GET, де дані були б видимими у URL.

Метод GET для отримання інформації

GET /events: застосування: використовується для отримання інформації про розклади або спортивні події. цей метод ідеально підходить для отримання даних, оскільки він не змінює стан сервера.

Метод PUT для оновлення інформації

- PUT /users: застосування: використовується для оновлення особистих даних користувача. PUT метод є ідемпотентним, що означає повторні запити з однаковими даними не змінюють результат.

Діаграма послідовностей відображає взаємодію між клієнтом та сервером в процесі використання вищезазначених методів. На рисунку 3.2 можна побачити, як клієнтська частина взаємодіє з сервером, використовуючи різні HTTP методи для реалізації бізнес-логіки:

- POST /login: Клієнт відправляє дані користувача для логіну.
- Сервер перевіряє дані та повертає результат аутентифікації.
- POST /registration: Клієнт відправляє дані для реєстрації нового користувача.
- Сервер обробляє ці дані та створює новий запис користувача.
- GET /events: Клієнт запитує інформацію про події.
- Сервер відправляє дані про всі доступні події.
- PUT /users: Клієнт відправляє оновлену інформацію про користувача.

КПЗ-2024

- Сервер оновлює дані користувача у базі даних та повертає оновлений профіль.

Ця послідовність дій демонструє, як різні частини системи взаємодіють для обробки даних користувача та подій. Використання Swagger документації дозволяє всім розробникам чітко розуміти, як взаємодіяти з кожним з цих ендпойнтів, що підвищує ефективність розробки та інтеграції клієнтської та серверної частини додатку.

Отже, після введення та підтвердження входу чи реєстрації, додаток надсилає дані з використанням серверного метода POST login чи registration, після чого отримує результат входу з серверу. В випадку успішної операції входу, користувач потрапляє в особистий кабінет, де за замовчуванням відкривається вкладка з розкладом, а відповідно відправляється запит на сервер для отримання розкладу GET events. В випадку неуспішного входу, користувачеві необхідно повторити введення особистих даних. За таким самим принципом відбувається виклик методів серверу для отримання квитків користувача, бронювання події чи скасування бронювання події, а також методу оновлення особистої інформації користувача.

КПЗ_2024

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Архітектура веб-додатку

Основні елементи Angular-додатку включають модулі, компоненти та сервіси (див. рисунок 4.2). Відповідно, додаток був організований у ключові логічні блоки — модулі. Зокрема:

- Core: цей модуль включає основні сталі компоненти додатку, які використовуються як singleton.

- Shared: містить елементи, що застосовуються у різних сегментах додатку, а отже, в множині модулів.

- Schedule: модуль, що містить розклад занять та їх деталі.

- PersonalInfo: модуль з даними користувача.

- Authorization: модуль, який забезпечує процеси авторизації та реєстрації.

- Tickets: модуль з актуальними квитками.

Кожен модуль був структурований наступним чином: він містить компоненти, сервіси та моделі. Модель включає інтерфейси, які задають структуру даних. Сервіси реалізують бізнес-логіку додатку, наприклад, завантажують та обробляють дані. Компоненти, у свою чергу, є графічними представленнями, що виводяться на екран і складаються з файлу розмітки, файлу стилів, та TypeScript файлу, який дозволяє інтегрувати логіку в інтерфейс, зокрема підключення до сервісів для завантаження даних.

Згідно з архітектурною схемою, представленою на рисунку 4.1, можна побачити поділ проекту на класи, які ілюстровані на діаграмі класів (рисунок 4.1). На цій діаграмі зображено часткову діаграму класів, яка включає головні модулі та деталізований опис одного з модулів. Інші модулі розроблені за аналогічним принципом.

Ключовими та базовими елементами в проектах Angular є app.component.ts та app.module.ts. Файл app.module.ts є початковим модулем, тоді як app.component.ts є базовим компонентом, який, у свою чергу, завантажує наступні компоненти. Цей модуль використовується в Router модулі, який додає можливості навігації у проекті, здійснюючи переходи відповідно до визначених маршрутів та включає інші основні модулі, такі як Core, Shared, Schedule, PersonalInfo, Authorization, та дозволяє навігувати користувача до цих модулів.

На діаграмі класів подано детальний розподіл модуля Authorization, який включає в себе наступні ключові компоненти:

- Компонент authorization: Цей компонент є основним і включає в себе компоненти register та login. Він відображає відповідний інтерфейс в залежності від дії користувача — реєстрації або входу в систему.

- Сервіс authorization.service: Цей сервіс забезпечує зв'язок між компонентами та сервером, виконуючи основні запити на сервер для авторизації та

КПЗ_2024

реєстрації користувача. У разі успішного входу в систему, сервіс використовує webstorage-service для збереження особистих даних користувача в localstorage. Це дозволяє зберігати дані в системі, якщо користувач не виконав вихід з системи. Сервіс також надсилає результат входу до компоненту.

- Компонентна поведінка: Залежно від результату входу, компонент або активує роутер для навігації та відкриття особистого кабінету користувача, або залишає користувача на сторінці авторизації чи реєстрації для повторного введення даних.

- Модель user-info.model: Цей інтерфейс описує формат зберігання інформації про користувача і використовується у компонентах та сервісі даного модуля для забезпечення типізації та структурування даних.

Таким чином, модуль Authorization є важливою частиною додатку, забезпечуючи інтеграцію з серверною частиною для авторизації та реєстрації, а також відповідне відображення в залежності від стану користувача.

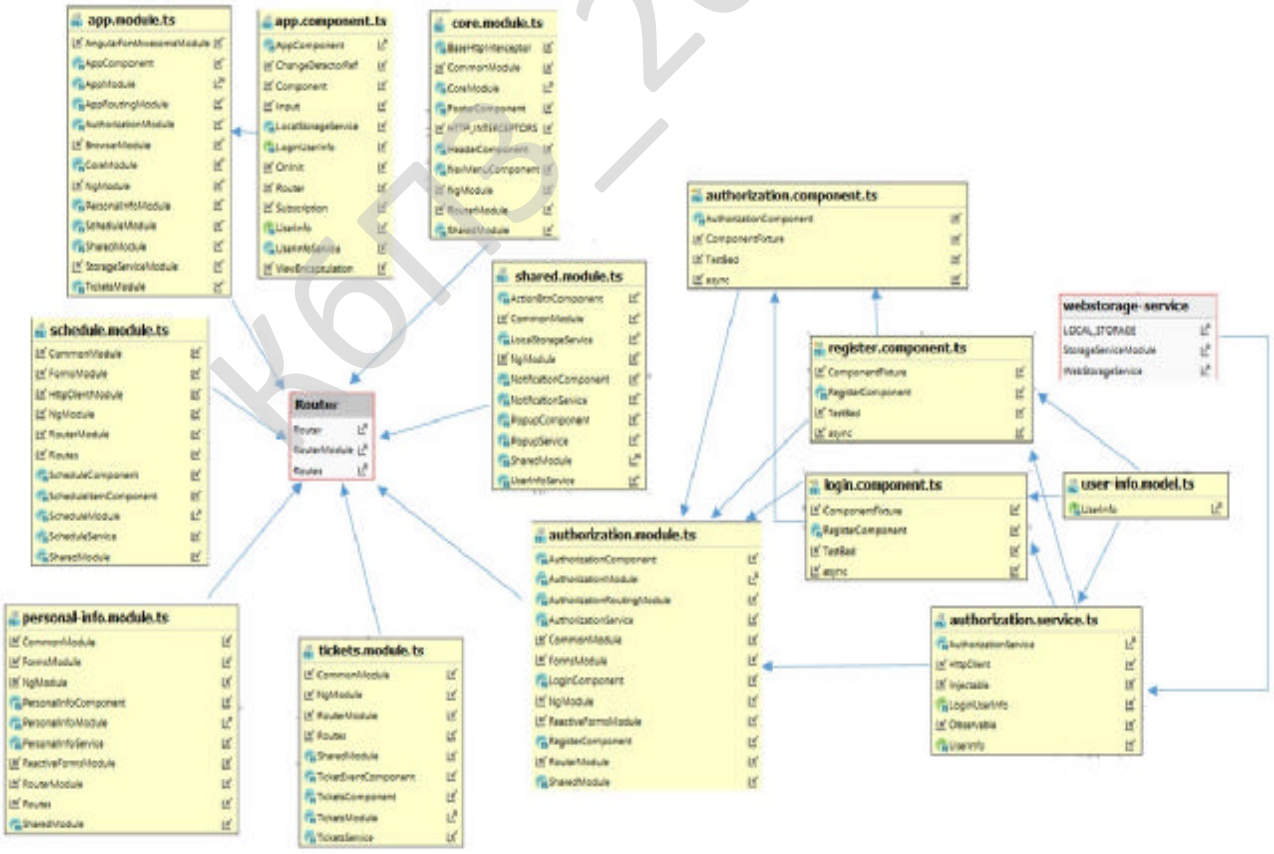

Рисунок 4.2 — Архітектура веб-додатку

КПІ_2024

Інші модулі в архітектурі Angular-додатку розроблені за аналогічним принципом, як і модуль Authorization. Кожен з цих модулів має свою унікальну структуру, яка складається з наступних основних частин.

1) Компоненти: Вони є основою для графічного представлення модулів. Компоненти визначають розмітку (HTML) і стилі (CSS), які характеризують візуальний аспект модулю. Вони також можуть включати додаткову логіку на TypeScript, яка допомагає взаємодіяти з даними і сервісами. Компоненти використовуються для формування інтерфейсу користувача, який забезпечує відображення і взаємодію з моделлю даних:

- Core Module: Може включати компоненти як core-layout, navbar, footer, які є загальними для всього додатку.

- Shared Module: Містить такі компоненти, як buttons, inputs, dialogs, які можуть бути використані в різних частинах додатку.

- Schedule Module: Включає специфічні компоненти, наприклад, calendar, appointment-details, які відображають графік занять.

- PersonalInfo Module: Має компоненти як profile-details, edit-profile, які дозволяють переглядати та редагувати особисті дані користувача.

- Tickets Module: Включає компоненти, які управляють відображенням інформації про квитки, такі як ticket-list, ticket-detail.

2) Сервіси: Це класи, які містять бізнес-логіку та взаємодію з сервером. Сервіси відповідають за обробку даних, виконання запитів на сервер, і обробку відповідей. Вони забезпечують необхідну логіку для роботи компонентів і дозволяють виконувати такі операції, як завантаження даних, авторизація користувачів, редагування профілю, та інші дії.

Core Module: Може містити сервіси як auth-service, user-service, які обробляють загальні задачі на рівні всього додатку.

Schedule Module: Може включати schedule-service, appointment-service, які забезпечують логіку для роботи з графіком занять.

КПЗ-2024

PersonalInfo Module: Містить такі сервіси, як profile-service, які допомагають обробляти інформацію про користувача.

3) Модель (Model): Це інтерфейси або класи, які визначають формат даних, що використовуються в додатку. Моделі допомагають структурувати дані, які пересилаються між клієнтом і сервером, та забезпечують типізацію даних у TypeScript. Вони використовуються для опису даних, що входять і виходять з компонентів і сервісів.

Core Module: Може включати моделі як User, AuthData, які описують основні дані користувача і дані авторизації.

Schedule Module: Містить моделі як Schedule, Appointment, які використовуються для роботи з графіком.

PersonalInfo Module: Використовує моделі як UserProfile, UserSettings, для опису інформації користувача.

Tickets Module: Може містити моделі як Ticket, Event, для опису квитків і подій.

Цей підхід забезпечує єдність архітектури і спрощує розширення та утримання Angular-додатку, оскільки кожен модуль має чітко визначені компоненти, сервіси та моделі, які взаємодіють між собою згідно з зазначеною структурою.

КбПЗ_2024

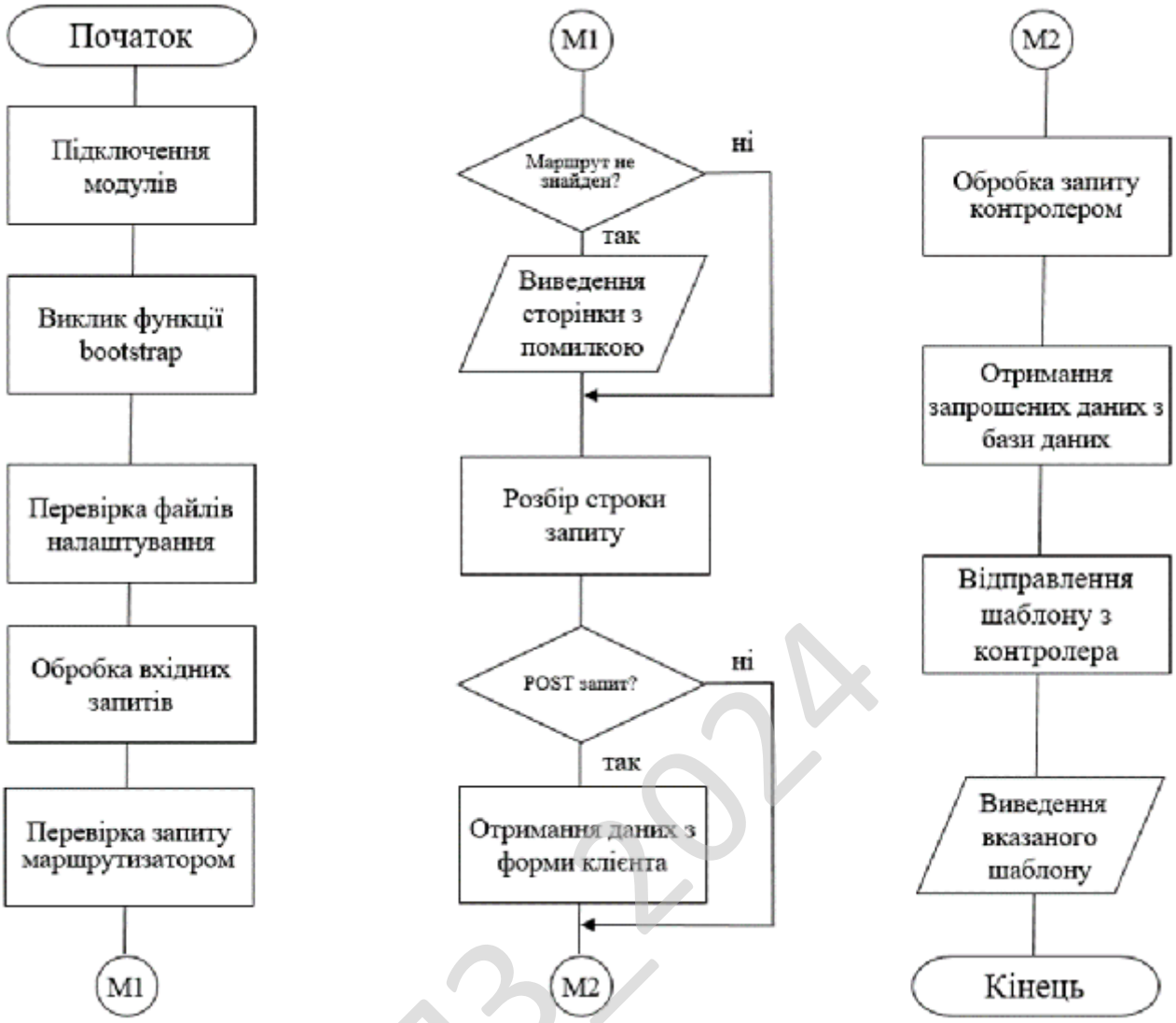

Рисунок 4.3 - Блок-схема алгоритму роботи основної програми

КОПЗ 2024

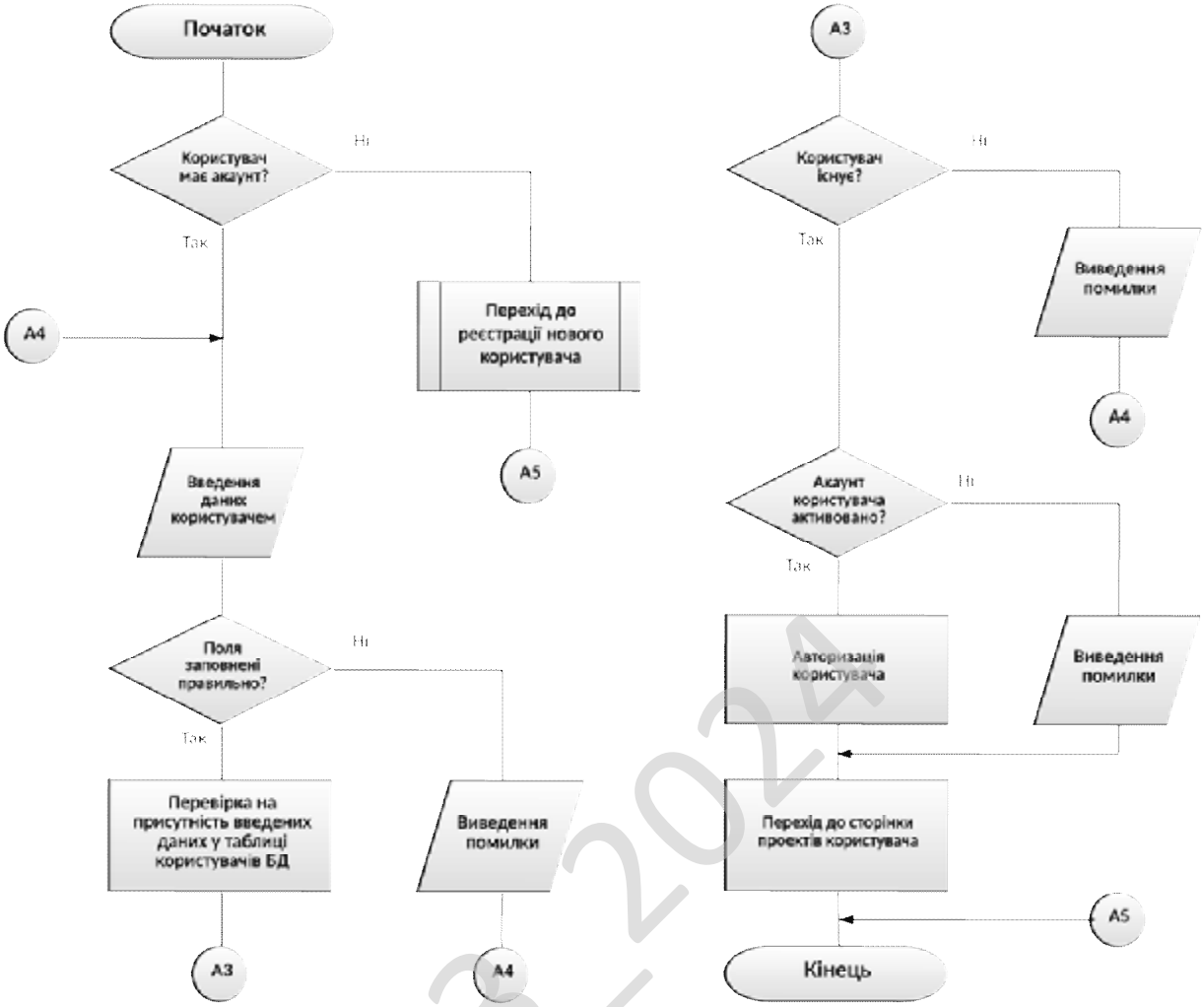

Рисунок 4.4 - Блок-схема роботи підпрограми реєстарції в додатку

КбПЗ_2024

4.2 Захист розробленого програмного забезпечення

Перш за все, рекомендується провести перевірку розробленого додатку за допомогою автоматизованих інструментів, які здатні ідентифікувати найпоширеніші помилки. Ці інструменти аналізують не лише код, написаний розробниками, але й використовувані бібліотеки та модулі, які часто є причиною вразливостей. Останні перевірки показують, що більше ніж 50% npm модулів не оновлювались протягом кількох років.

Є декілька методів для перевірки модулів перед їхнім використанням. Першочергово слід оглянути їх код та визначити, чи відповідає модуль потребам

розробника. Важливо також перевірити, які інші бібліотеки використовує даний модуль і коли вони були оновлені востаннє; краще, коли модуль має менше залежностей.

Великі компанії часто мають спеціалізованих співробітників, які проводять аудит пакетів перед їх використанням та складають спеціальні білі списки дозволених модулів. Npm стурбована тим, як часто в бібліотеки впроваджуються шкідливі програми, такі як майнери та віруси для збору приватних даних. У відповідь на це було випущено спеціальний інструмент npm audit, який сканує встановлені модулі в проекті і порівнює їх з чорним списком модулів, що містять вразливості.

Навіть команда npm install тепер попереджає про наявність вразливостей у модулях. Команда npm audit fix пропонує заміну вразливих пакетів на більш захищені версії. Проте, цей метод допомагає виявити лише відомі вразливості, про які вже повідомлено користувачами та розробниками. Також, написання тестів є важливим аспектом перевірки безпеки додатку.

Розробники на AngularJS часто використовують концепцію Jasmine's Behavior-Driven Development (BDD). Jasmine — це фреймворк для тестування JavaScript коду, що може бути використаний на будь-якій платформі і відрізняється зручним інтерфейсом та зрозумілим налаштуванням.

Для тестування одиничних тестів часто застосовують Karma, що дозволяє привести тестування додатку до умов продакшену. Важливо також використовувати протокол HTTPS, який є більш безпечним за HTTP. HyperText Transfer Protocol Secure (HTTPS) забезпечує шифрування даних і надійно захищає користувацькі дані при передачі в Інтернеті.

Більшість сучасних сайтів використовують цей протокол, особливо для передачі приватних даних, наприклад, паролів або інформації кредитних карт. Також потрібно звернути увагу на cookie файли під час авторизації, оскільки зловмисники можуть їх перехопити. Використання HTTPS протоколу допоможе уникнути цих проблем. Необхідно враховувати загрози CORS та CSRF атак.

КОПЗ-2024

CSRF-атака використовує непомітні форми для відправлення запитів з приватними даними користувача. Якщо авторизація на сайті здійснюється тільки через cookies, така атака може бути успішною.

Щоб запобігти цьому, використовуються спеціальні токени. Під час підпису XMLHttpRequest, токен зберігається в cookies, а JavaScript може прочитати його з домену і додати в заголовок запиту, а сервер перевіряє наявність коректного токену.

При крос-доменних запитах, браузер додає заголовок Origin, який містить домен запиту. Сервер повинен перевірити цей домен і відповісти спеціальним заголовком.

Якщо сервер дозволяє доступ, він надсилає заголовок Access-Control-Allow-Origin з доменом запиту. У випадку відсутності цього заголовка, сервер завершує обробку запиту з помилкою. Під час таких запитів куки і HTTP-авторизація не передаються.

КбПЗ_2024

5 ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Під час створення веб-додатку була обрана базова архітектура, яка включає модулі, компоненти та сервіси. Ці елементи є критично важливими для забезпечення чистоти коду та його ефективності. Щоб взаємодіяти з сервером, у проекті було вирішено використовувати архітектурний шаблон REST. Такий підхід дозволяє створювати легкозрозумілі та ефективні API для обміну даними між клієнтом та сервером.

Клієнт-серверна архітектура REST

Однією з ключових задач при створенні веб-додатку для спортивного закладу було забезпечення надання актуальної інформації користувачам. Для цього було обрано клієнт-серверну архітектуру, як показано на рисунку 4.1. У такій архітектурі стан ресурсу зберігається на сервері, а клієнти, підключаючись до сервера, отримують та зберігають локальний стан додатку, що дозволяє їм завжди мати доступ до останніх даних.

Для спілкування між клієнтом та сервером були розглянуті протокол та стандарт SOAP, а також архітектура REST. Після аналізу цих підходів було вибрано архітектуру REST з наступних причин:

- Незалежність від мови та платформи: REST може бути використаний з будь-якою мовою програмування та на будь-якій платформі, що забезпечує велику гнучкість у виборі технологій для клієнта та сервера.
- Повне використання можливостей HTTP: REST базується на стандартних методах HTTP, таких як GET, POST, PUT, DELETE тощо, використовуючи протокол HTTP як із його рідними можливостями, що підвищує ефективність і читабельність API.
- Простота і швидкість: В той час як SOAP використовує XML, що може сповільнити обробку даних, REST часто використовує більш легкі формати,

КПЗ_2024

такі як JSON, що робить обмін даними швидшим і спрощує роботу з отриманими даними.

- CRUD операції: REST надає чітко визначені оперції CRUD (create, read, update, delete), які дозволяють ефективно управляти ресурсами, роблячи архітектуру інтуїтивно зрозумілою та легкою для використання.
- Робота з ресурсами через URI: В архітектурі REST кожен ресурс є ідентифікованим за допомогою універсального ідентифікатора (URI), що робить веб-посилання натуральним способом адресації ресурсів.
- Визначеність дій над ресурсами: REST використовує обмежений набір чітко визначених операцій, що дозволяє легко визначити потрібні дії для створення, читання, оновлення, або видалення ресурсів.

Приклад Архітектури на REST

При створенні веб-додатку для спортивного закладу за допомогою архітектури REST, взаємодія між клієнтом і сервером може бути організована наступним чином:

- Запити до сервера: Клієнт виконує HTTP запити до сервера для отримання, створення, оновлення або видалення інформації про спортивні події.
- Наприклад, клієнт може виконати GET запит до /api/sport-events для отримання списку всіх подій.
- Обробка запитів на сервері: Сервер обробляє запит, взаємодіючи з базою даних або іншими сервісами, і повертає відповідь.
- Наприклад, сервер може повернути JSON масив з об'єктами подій, кожен з яких містить інформацію про одну подію.
- Відображення даних: Клієнт отримує відповідь від сервера і використовує ці дані для оновлення інтерфейсу користувача.
- Наприклад, додаток може відобразити отримані події у формі таблиці або списку на веб-сторінці.

Ця архітектура дозволяє розробникам створювати веб-додатки, які ефективно взаємодіють з сервером, забезпечуючи користувачам актуальну

КПЗ-2024

інформацію та високий рівень взаємодії з додатком.

Для розробки ефективних веб-служб з використанням передачі стану, важливо глибоко розуміти основи протоколу передачі гіпертексту (HTTP) і унікальних ідентифікаторів ресурсів (URI). Це дозволяє створити чіткі і оптимізовані інтерфейси для взаємодії з сервером, що є ключовим аспектом для успішного застосування архітектурного стилю REST.

HTTP Методи

У RESTful архітектурі кожен ресурс ідентифікується через URI, а взаємодія з цими ресурсами відбувається через стандартні HTTP методи. Ось основні методи, які ви використовуєте:

- **GET**: Використовується для отримання даних від сервера. GET запити повинні бути ідемпотентними, тобто їхнє повторне виконання не повинно впливати на стан сервера.
- **POST**: Використовується для створення нових ресурсів на сервері. POST запити можуть змінювати стан сервера, додаючи нові ресурси.
- **PUT**: Використовується для оновлення існуючих ресурсів. PUT запит повинен містити повну інформацію про ресурс, яка замінює поточний стан ресурсу на сервері.
- **PATCH**: Використовується для часткового оновлення ресурсу. Відмінно від PUT, PATCH змінює лише ті атрибути ресурсу, які були включені у запит.
- **DELETE**: Використовується для видалення ресурсів. DELETE запити видаляють ресурси з сервера.

У веб-додатку використання основних HTTP методів, таких як GET, POST, і PUT, є стандартним підходом для реалізації RESTful архітектури. Кожен з цих методів відіграє ключову роль у взаємодії з сервером та управлінні станом ресурсів. За допомогою Swagger документації ефективно інтегрувати клієнтську та серверну частини додатку.

GET: метод використовується для запиту інформації з сервера. Він

КОПЗ_2024

ідемпотентний, що означає, що повторні запити за тим же URL повернуть однакові результати без зміни стану сервера. Ваш додаток може використовувати GET для отримання даних, наприклад, списку спортивних подій, деталей про конкретну подію, або інших ресурсів.

POST: метод використовується для створення нового ресурсу на сервері. POST не є ідемпотентним, що означає, що кожний запит може змінювати стан сервера, додаючи новий ресурс. Ваш додаток може використовувати POST для створення нової спортивної події або інших сутностей.

PUT: метод використовується для оновлення існуючих ресурсів. PUT є ідемпотентним, а тому повторні запити з однаковими даними не змінюватимуть результат. Використовується для оновлення даних про існуючі спортивні події або інші ресурси.

Методи серверу для роботи з ресурсами було відображено на Swagger документації та надано для інтеграції клієнтської частини з серверною частиною (рисунок 5.1).

Рисунок 5.1 — Приклад документаці серверних методів

При розробці веб-додатків, правильний вибір HTTP методів для різних операцій є ключовим для забезпечення безпеки, ефективності та інтуїтивно

КПЗ 2024

зрозумілого API. Ось як можна використовувати ці методи в контексті вашого додатку:

Метод POST для Аутентифікації та Реєстрації:

- POST /login та /registration: застосування: ці ендпойнти використовуються для аутентифікації та реєстрації користувачів.
- Безпека: Використання POST для цих операцій дозволяє відправляти особисті дані (як-от логін і пароль) у тілі запиту (body), відокремлено від URL, що підвищує безпеку порівняно з GET, де дані були б видимими у URL.

Метод GET для Отримання Інформації

GET /events: застосування: використовується для отримання інформації про розклади або спортивні події. цей метод ідеально підходить для отримання даних, оскільки він не змінює стан сервера.

Метод PUT для Оновлення Інформації

- PUT /users: застосування: використовується для оновлення особистих даних користувача. PUT метод є ідемпотентним, що означає повторні запити з однаковими даними не змінюють результат.

Діаграма послідовностей відображає взаємодію між клієнтом та сервером в процесі використання вищезазначених методів. На рисунку 3.2 можна побачити, як клієнтська частина взаємодіє з сервером, використовуючи різні HTTP методи для реалізації бізнес-логіки.

POST /login: Клієнт відправляє дані користувача для логіну.

Сервер перевіряє дані та повертає результат аутентифікації.

POST /registration: Клієнт відправляє дані для реєстрації нового користувача.

Сервер обробляє ці дані та створює новий запис користувача.

GET /events: Клієнт запитує інформацію про події.

Сервер відправляє дані про всі доступні події.

PUT /users: Клієнт відправляє оновлену інформацію про користувача.

Сервер оновлює дані користувача у базі даних та повертає оновлений

КОПЗ_2024

профіль.

Ця послідовність дій демонструє, як різні частини системи взаємодіють для обробки даних користувача та подій. Використання Swagger документації дозволяє всім розробникам чітко розуміти, як взаємодіяти з кожним з цих ендпойнтів, що підвищує ефективність розробки та інтеграції клієнтської та серверної частини додатку.

Отже, після введення та підтвердження входу чи реєстрації, додаток надсилає дані з використанням серверного метода POST login чи registration, після чого отримує результат входу з серверу. В випадку успішної операції входу, користувач потрапляє в особистий кабінет, де за замовчуванням відкривається вкладка з розкладом, а відповідно відправляється запит на сервер для отримання розкладу GET events. В випадку неуспішного входу, користувачеві необхідно повторити введення особистих даних. За таким самим принципом відбувається виклик методів серверу для отримання квитків користувача, бронювання події чи скасування бронювання події, а також методу оновлення особистої інформації користувача.

Кожен модуль був структурований наступним чином: він містить компоненти, сервіси та моделі. Модель включає інтерфейси, які задають структуру даних. Сервіси реалізують бізнес-логіку додатку, наприклад, завантажують та обробляють дані. Компоненти, у свою чергу, є графічними представленнями, що виводяться на екран і складаються з файлу розмітки, файлу стилів, та TypeScript файлу, який дозволяє інтегрувати логіку в інтерфейс, зокрема підключення до сервісів для завантаження даних.

Згідно з архітектурною схемою, представленою на рисунку 3.3, можна побачити поділ проекту на класи, які ілюстровані на діаграмі класів (рисунок 3.4). На цій діаграмі зображено часткову діаграму класів, яка включає головні модулі та деталізований опис одного з модулів. Інші модулі розроблені за аналогічним принципом.

Ключовими та базовими елементами в проектах Angular є app.component.ts

КПЗ-2024

та app.module.ts. Файл app.module.ts є початковим модулем, тоді як app.component.ts є базовим компонентом, який, у свою чергу, завантажує наступні компоненти. Цей модуль використовується в Router модулі, який додає можливості навігації у проекті, здійснюючи переходи відповідно до визначених маршрутів та включає інші основні модулі, такі як Core, Shared, Schedule, PersonalInfo, Authorization, та дозволяє навігувати користувача до цих модулів.

На діаграмі класів подано детальний розподіл модуля Authorization, який включає в себе наступні ключові компоненти.

Компонент authorization: Цей компонент є основним і включає в себе компоненти register та login. Він відображає відповідний інтерфейс в залежності від дії користувача — реєстрації або входу в систему.

Сервіс authorization.service: Цей сервіс забезпечує зв'язок між компонентами та сервером, виконуючи основні запити на сервер для авторизації та реєстрації користувача. У разі успішного входу в систему, сервіс використовує webstorage-service для збереження особистих даних користувача в localstorage. Це дозволяє зберігати дані в системі, якщо користувач не виконав вихід з системи. Сервіс також надсилає результат входу до компоненту.

Компонентна поведінка: Залежно від результату входу, компонент або активує роутер для навігації та відкриття особистого кабінету користувача, або залишає користувача на сторінці авторизації чи реєстрації для повторного введення даних.

Модель user-info.model: Цей інтерфейс описує формат зберігання інформації про користувача і використовується у компонентах та сервісі даного модуля для забезпечення типізації та структурування даних.

Таким чином, модуль Authorization є важливою частиною додатку, забезпечуючи інтеграцію з серверною частиною для авторизації та реєстрації, а також відповідне відображення в залежності від стану користувача.

КПЗ_2024

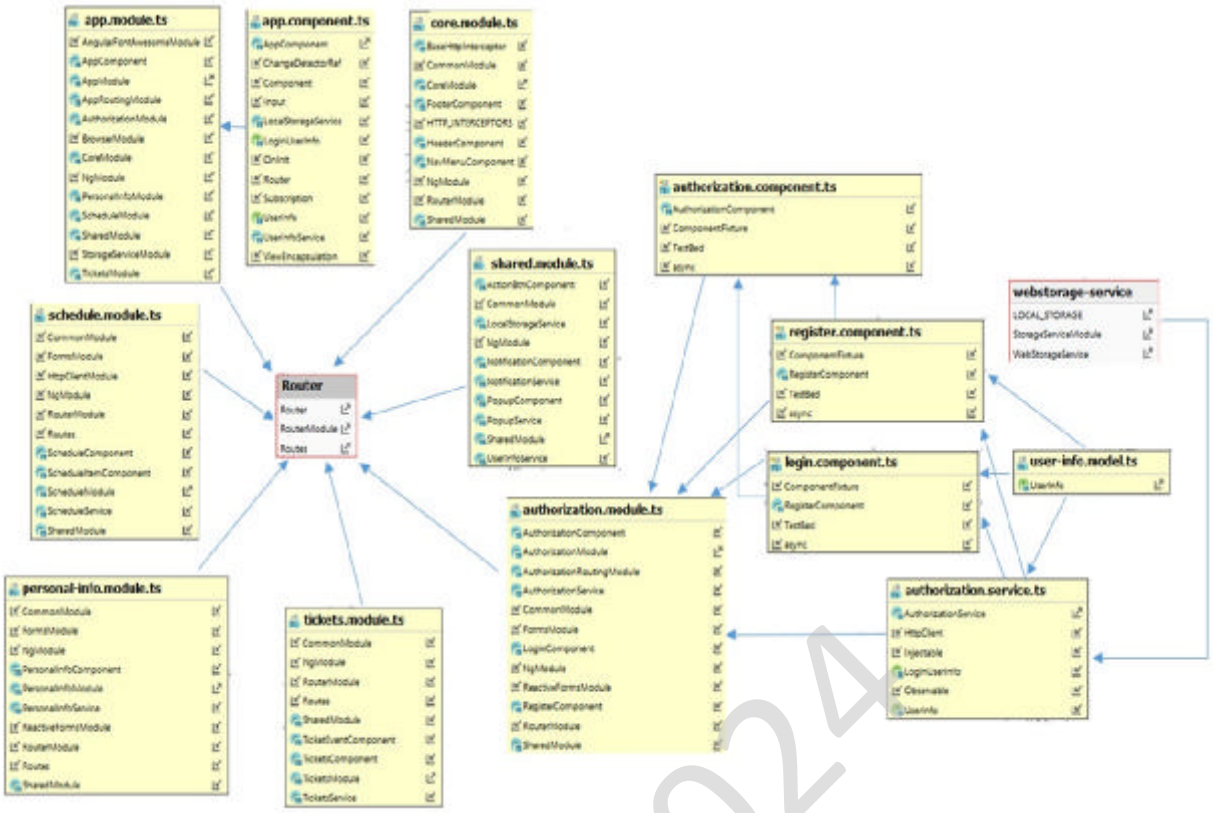

Рисунок 5.2 — Діаграма класів веб-додатку

Інші модулі в архітектурі Angular-додатку розроблені за аналогічним принципом, як і модуль Authorization. Кожен з цих модулів має свою унікальну структуру, яка складається з наступних основних частин:

Компоненти: Вони є основою для графічного представлення модулів. Компоненти визначають розмітку (HTML) і стилі (CSS), які характеризують візуальний аспект модулю. Вони також можуть включати додаткову логіку на TypeScript, яка допомагає взаємодіяти з даними і сервісами. Компоненти використовуються для формування інтерфейсу користувача, який забезпечує відображення і взаємодію з моделлю даних.

Core Module: Може включати компоненти як core-layout, navbar, footer, які є загальними для всього додатку.

Shared Module: Містить такі компоненти, як buttons, inputs, dialogs, які можуть бути використані в різних частинах додатку.

Schedule Module: Включає специфічні компоненти, наприклад, calendar,

КОПЗ_2024

appointment-details, які відображають графік занять.

PersonalInfo Module: Має компоненти як profile-details, edit-profile, які дозволяють переглядати та редагувати особисті дані користувача.

Tickets Module: Включає компоненти, які управляють відображенням інформації про квитки, такі як ticket-list, ticket-detail.

Сервіси: Це класи, які містять бізнес-логіку та взаємодію з сервером. Сервіси відповідають за обробку даних, виконання запитів на сервер, і обробку відповідей. Вони забезпечують необхідну логіку для роботи компонентів і дозволяють виконувати такі операції, як завантаження даних, авторизація користувачів, редагування профілю, та інші дії.

Core Module: Може містити сервіси як auth-service, user-service, які обробляють загальні задачі на рівні всього додатку.

Schedule Module: Може включати schedule-service, appointment-service, які забезпечують логіку для роботи з графіком занять.

PersonalInfo Module: Містить такі сервіси, як profile-service, які допомагають обробляти інформацію про користувача.

Модулі веб-додатку

Angular-додатки організовані за модульним принципом, і в центрі цієї архітектури знаходиться система NgModules. NgModules виконує роль контейнерів, які згуртовують в собі блоки коду, пов'язані з певним доменом програми, робочим процесом, або набором функціональностей.

Основні Властивості NgModules

declarations: Включає компоненти, директиви, та пайпи, які належать до даного модулю. Тут зазначаються усі класи, які прямо взаємодіють із представленнями (views) у даному модулі, та які модуль сам обробляє.

Exports: Визначає підмножину декларацій, які повинні бути доступними в шаблонах компонентів інших модулів. Це означає, що елементи, оголошені тут, можуть бути використані у компонентах інших NgModules.

Imports: Включає інші модулі, чиї експортовані класи необхідні шаблонам

КПЗ_2024

компонентів, оголошеним у цьому NgModule. Тобто, тут модуль вказує, які інші модулі він використовує для своєї роботи.

Providers: Описує сервіси, які NgModule додає до глобальної колекції сервісів, роблячи їх доступними в усіх частинах програми. Також можна вказати провайдерів на рівні компонентів для більш локалізованого використання.

Bootstrap: Вказує основний компонент програми, який є кореневим компонентом, та який розміщує всі інші представлення програми. Ця властивість встановлюється тільки у кореневому NgModule.

Кожен Angular додаток має хоча б один NgModule клас, зазвичай названий AppModule, який розташований у файлі app.module.ts. Цей кореневий модуль є стартовим пунктом для Angular, який ініціалізує завантаження додатку:

Ця структура забезпечує чітку модульну організацію, де AppModule інтегрує функціональні модулі, такі як CoreModule, SharedModule, TicketsModule, PersonalInfoModule, та AuthorizationModule. Кожен з цих модулів, в свою чергу, містить свої компоненти, сервіси, та інші класи, що сприяє легкому масштабуванню та утриманню додатку.

КПІ, 2024

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from '@angular/router';

import { SharedModule } from '@shared/shared.module';
import { TicketsComponent } from '@tickets/tickets/tickets.component';
import { TicketsService } from '@tickets/services/tickets.service';
import { TicketEventComponent } from '@tickets/ticket-event/ticket-event.component'

const routes: Routes = [
  {
    path: 'tickets',
    component: TicketsComponent
  }
];

@NgModule({
  imports: [
    CommonModule,
    SharedModule,
    RouterModule.forRoot(routes)
  ],
  declarations: [
    TicketsComponent,
    TicketEventComponent
  ],
  providers: [
    TicketsService
  ],
  exports: [
    TicketsComponent,
    RouterModule
  ]
})
```

Рисунок 5.3 — Базова структура модулю

Ця архітектура забезпечує ясну та ефективну модульну структуру, де AppModule служить основним інтеграційним центром для всіх функціональних модулів, таких як CoreModule, SharedModule, TicketsModule, PersonalInfoModule, і AuthorizationModule. Кожен з цих модулів містить свої власні компоненти, сервіси та інші класи, що дозволяє з легкістю розширювати та підтримувати додаток, сприяючи чіткому розділенню відповідальності та покращенню управління кодом.

Важливість модульної організації в Angular.

1) Сприяння Чіткому Розділенню Відповідальності: Кожен модуль фокусується на конкретній функціональності або домені, зменшуючи складність та покращуючи читабельність коду. Це розділення допомагає розробникам швидше знаходити, виправляти помилки та додавати нові функції.

2) Підвищення Перевикористання Коду: Завдяки SharedModule можна легко перевикористовувати загальні компоненти, директиви, та пайпи в різних частинах додатку, що сприяє ефективності та зниженню дублювання коду.

3) Спрощення Тестування: Модульна структура робить юніт-тестування більш організованим та ефективним, оскільки можна ізолювати тестування для конкретних модулів без необхідності запускати весь додаток.

4) Легкість Масштабування: Модулі можуть бути розроблені та вдосконалені незалежно, що дозволяє легко масштабувати додаток, додаючи нові функціональні модулі або розширюючи існуючі без значного впливу на інші частини системи.

5) Оптимізація Завантаження: Angular підтримує ліниве завантаження (lazy loading) модулів, що дозволяє покращити час завантаження додатків, завантажуючи модулі по мірі необхідності, замість завантаження всього додатку одразу.

Цей приклад демонструє як основний AppModule використовує вбудовані та власні модулі для створення згуртованого та масштабованого додатку. Кожен модуль забезпечує специфічні для себе компоненти та сервіси, вносячи свій вклад в загальну функціональність та структуру програми.

КБПЗ-2024

Сервіси веб-додатку

Сервіси в Angular — це класи з чітко визначеними завданнями, які спрямовані на виконання конкретних функцій в додатку. Ці класи відіграють ключову роль у забезпеченні перевикористання коду, зниженні зв'язності між різними частинами додатку та покращенні тестування компонентів.

Основні Риси Сервісів в Angular

Визначення Цілі та Функцій: Сервіси в Angular мають вузьку, добре визначену мету. Вони зосереджені на виконанні певної задачі або набору задач, наприклад, управління даними, взаємодія з сервером, кешування, аутентифікація користувачів та інше.

Сприяння Модульності та Перевикористанню: Сервіси підвищують модульність додатку, дозволяючи різним компонентам використовувати загальні функції без дублювання коду. Це сприяє легкій зміні та масштабуванню додатку.

Відділення Відповідальностей: Сервіси допомагають відділити логіку обробки даних від UI логіки, яка знаходиться у компонентах. Це робить компоненти більш чистими, оскільки вони не виконують надмірну обробку даних, а лише відображають дані користувачу.

Тестування: Сервіси легше тестувати окремо від компонентів, оскільки вони не залежать від DOM або Angular-специфічних шаблонів. Це дозволяє писати більш ефективні юніт-тести.

Dependency Injection (DI): Angular використовує DI для надання сервісів компонентам. Це означає, що компоненти можуть декларативно вказати, які залежності їм потрібні, замість того, щоб створювати їх вручну, що покращує гнучкість та зменшує зв'язаність коду.

Ролі Компонентів та Сервісів в Angular

Angular використовує архітектурну структуру, яка розділяє компоненти та сервіси, щоб збільшити модульність та можливість повторного використання коду. Це розділення відіграє ключову роль у забезпеченні гнучкості та ефективності в додатках, створених з Angular.

КПЗ_2024

@Injectable (), та щоб надати метадані, які дозволяють Angular вводити його в компонент як залежність.

```
@Injectable()
export class ScheduleService {

  constructor(private http: HttpClient) { }

  public getScheduleForPeriod(sportType: string): Observable<ScheduleDay[]> {
    if (sportType !== 'All') {
      return this.http.get<ScheduleDay[]>(GET_SCHEDULE_FOR_PERIOD).pipe(
        map((data: ScheduleDay[]) => {
          return data.map((item: ScheduleDay) => {
            return { ...item, dateUI: this.getDateFromString(item.date) };
          })
            .sort((a: ScheduleDay, b: ScheduleDay) => a.dateUI.getTime() - b.dateUI.getTime())
            .map((day: ScheduleDay) => {
              return {
                ...day,
                events: day.events
                  .filter((event: ScheduleEvent) => event.sportType === sportType)
              };
            })
            .filter((day: ScheduleDay) => day.events.length > 0);
```

Рисунок 5.4 — Сервіси

Розуміння Dependency Injection в Angular

В Angular, Dependency Injection (DI) є ключовим аспектом для досягнення високої модульності та гнучкості. DI дозволяє компонентам, сервісам, пайпам і навіть модулям отримувати свої залежності, не створюючи їх напряму. Це сприяє більшій тестованості, зниженню зв'язності та підвищенню гнучкості програми.

Angular's DI система є фундаментальним інструментом для створення ефективних, модульних та легко тестованих додатків. Вона дозволяє розробникам зосередитись на бізнес-логіці, звільняючи їх від необхідності ручного управління залежностями та створення екземплярів. Використання цього підходу в вашому Angular додатку може значно покращити його архітектуру та спростити майбутнє супроводження зображено на рисунку 5.5.

КОПЗ_2024

```
export class ScheduleItemComponent implements OnInit {
    public popupControls: PopupControls;
    public currentEvent: ScheduleEvent;

    public isActionPerformed = false;

    @Input() data: ScheduleDay;
    @Output() updateScheduleInfo = new EventEmitter();

    constructor(private scheduleService: ScheduleService,
        private notification: NotificationService,
        private localStorageService: LocalStorageService,
        private popupService: PopupService) { }

    ngOnInit() {
```

Рисунок 5.5 — Приклад DI

Методика роботи користувача з програмою

Для того, щоб скористатися веб-додатком, перше, що потрібно зробити — це забезпечити підключення до мережі Інтернет і перейти на адресу **localhost:4200** у браузері. Це адреса, на якій локально розгорнуто веб-додаток, зазвичай через Angular CLI або інший веб-сервер розробки.

Процес входу та реєстрації у веб-додатку

Початковий екран

Коли користувач запускає веб-додаток, він спочатку побачить екран авторизації та реєстрації. Цей екран дозволяє користувачам ввійти в систему, якщо вони вже зареєстровані, або зареєструватися, якщо вони нові користувачі.

Реєстрація нового користувача

Якщо користувач ще не зареєстрований, йому необхідно перейти на вкладку "Registration".

На цій вкладці користувачу потрібно заповнити всі обов'язкові поля форми реєстрації. Це можуть бути поля, як-от ім'я, електронна адреса, логін, пароль тощо.

Після заповнення форми користувач натискає кнопку "Register", щоб

КПІВ_2024

завершити процес реєстрації.

Вхід для зареєстрованих користувачів

Зареєстровані користувачі вибирають вкладку "Authorization".

На цій вкладці потрібно ввести логін та пароль, які були вказані під час реєстрації.

Після введення цих даних користувач натискає "Log In", щоб увійти в систему і отримати доступ до функціоналу веб-додатку.

Важливі моменти

Забезпечення безпеки: Забезпечте безпечне зберігання та передачу паролів, використовуючи хешування та безпечні протоколи передачі даних.

Валідація даних: Переконайтеся, що всі поля форм реєстрації та авторизації належним чином валідуються, щоб уникнути помилок та потенційних атак.

Користувацький інтерфейс: Забезпечте, щоб користувацький інтерфейс був інтуїтивно зрозумілим та зручним, з чіткими інструкціями та повідомленнями про помилки.

Процес дозволяє забезпечити, що користувачі можуть легко та безпечно входити в систему або реєструватися, підтримуючи гладкий користувацький досвід і забезпечуючи доступ до важливих функцій вашого веб-додатку.

КбПЗ_2024

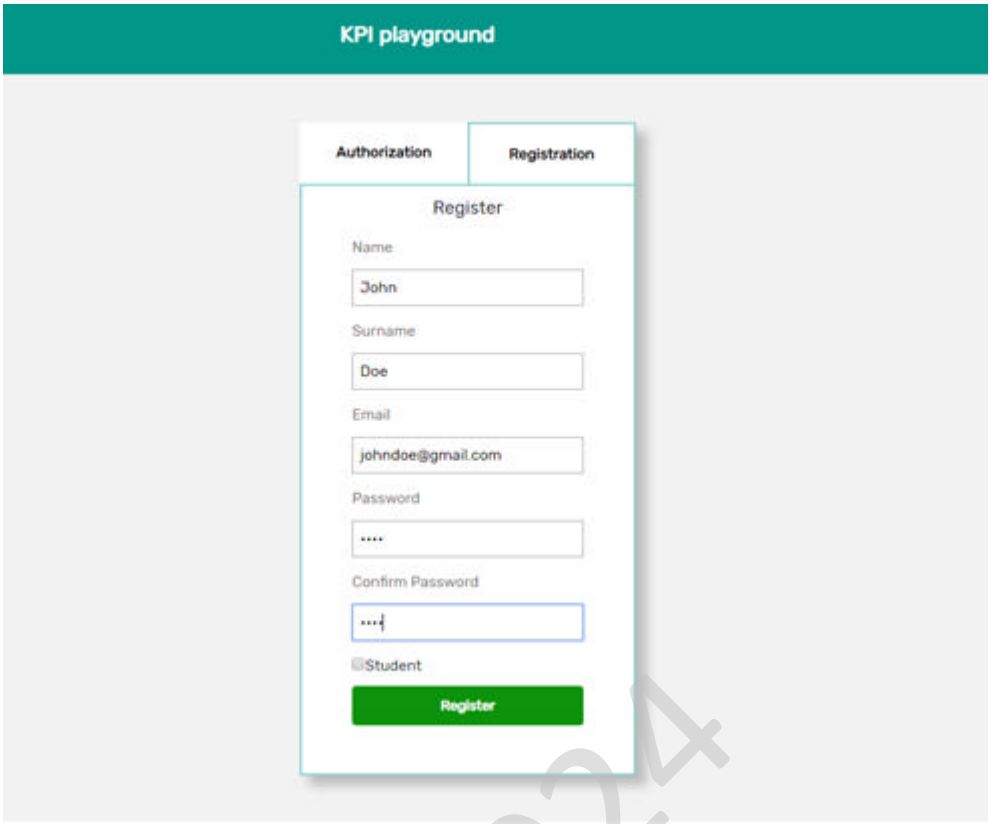

Рисунок 5.6 — Екран реєстрації

У випадку неправильного заповнення форми або незаповнення обов'язкових полів, веб-додаток має належним чином інформувати користувача про помилки та забезпечити, щоб дії, які вимагають коректного заповнення форми, були заблоковані до моменту виправлення помилок. Ось як це може бути організовано:

Повідомлення про помилки

Візуальні індикатори помилок

Поля, що містять помилки, можуть бути виділені, наприклад, червоним кольором або підкресленням.

Додатково, поряд з кожним полем може з'являтися текстове повідомлення, яке конкретно описує помилку (наприклад, "Це поле є обов'язковим", "Електронна адреса недійсна", тощо).

Загальні повідомлення

Над формою може розміщуватися загальне повідомлення про помилки, яке

КПІ_2024

вказує на те, що деякі поля заповнені неправильно або залишені **порожніми.**

Керування активністю кнопок

Неактивна кнопка "Register" або "Log In":

Кнопка реєстрації ("Register") або авторизації ("Log In") повинна залишатися неактивною (disabled), поки не будуть коректно заповнені всі обов'язкові поля.

Перехід в Особистий Кабінет

Після успішного входу або реєстрації користувач автоматично потрапляє в особистий кабінет, де за замовчуванням відкрита вкладка "Schedule". Це може бути реалізовано через роутинг в Angular: структура допомагає забезпечити зручність та ефективність інтерфейсу користувача, динамічно адаптуючись до введення даних та валідації форми, і забезпечуючи користувачам легкий доступ до основних функцій веб-додатку (риисунок 5.7).

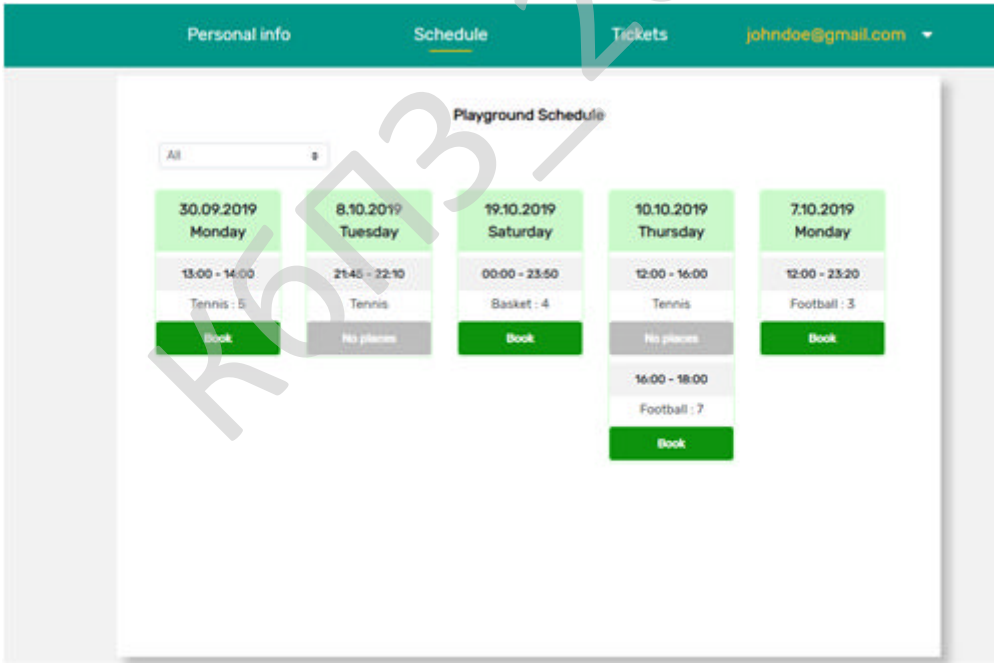

Рисунок 5.7 — Розклад

На вкладці "Schedule" веб-додатку представлено актуальний розклад занять, де користувачі мають можливість переглядати та фільтрувати заняття за обраним

КБПЗ-2024

видом спорту. Інформація про заняття систематизована за днями, і кожен день включає список заходів з детальною інформацією про кожен з них, зокрема про наявність вільних місць.

КБПЗ_2024

6 ОСНОВНІ ВИСНОВКИ

Відтак, ідея впровадження автоматизованих сервісів для контролю та управління доступом до спортивних комплексів є вкрай актуальною на сучасному етапі. З метою забезпечення ефективного контролю за навантаженням закладів та збереженням безпеки відвідувачів, була створена система "Відкритого спортивного майданчику з е-сервісами".

Під час побудови архітектури цієї системи я обрав сервіс-орієнтований підхід, який дозволяє кожній складовій системи відповідати за окремі бізнес-функції без взаємовпливу на інші підсистеми.

На цьому етапі розробки була створена базова частина — система контролю та управління доступом на спортивний майданчик, яка включає веб-застосунок, сервер та систему контролю доступу (СКУД). В майбутньому, система може бути розширена за рахунок введення систем суддівства, видачі інвентаря, освітлення та інших елементів.

Ключовим інструментом, який забезпечує керування доступом на майданчик, є клієнтська частина, що взаємодіє з серверним застосунком. В процесі роботи було аналізовано існуючі застосунки спортивних закладів, їхні переваги та обмеження, і визначено вимоги до розроблюваної системи.

Головними задачами системи є:

- забезпечення кросплатформеності та швидкодії: для досягнення цієї мети було обрано веб-застосунок, що дозволяє користувачам отримувати доступ до системи з будь-якого пристрою через Інтернет;
- забезпечення актуальності інформації та онлайн виконання операцій: для цього створено комунікацію веб-додатку з серверним застосунком через власний API.

Щодо швидкодії, було вибрано фреймворк Angular, який дозволяє створювати динамічні веб-застосунки, які швидко реагують на дії користувача. Для підтримки асинхронного програмування використано бібліотеку RxJS, що

КПЗ - 2024

додатково покращує продуктивність. Крім того, використано TypeScript для розширення можливостей JavaScript.

Архітектура веб-додатку визначена відповідно до ключових компонентів Angular, включно з модулями, компонентами та сервісами.

Було також розглянуто можливість комерціалізації проекту, і зроблено висновок, що він має потенціал для успішного впровадження на сучасному ринку. Основними обмеженнями є орієнтація системи на студентів та локальний запуск застосунку, що є першочерговими напрямками для подальшого поліпшення.

Методика розробки дозволяє створювати масштабовані веб-застосунки, готові до розширення в майбутньому. У разі необхідності доповнення застосунку, створюється нова гілка в системі контролю версій, додаються нові модулі з необхідними компонентами та сервісами в Angular проекті. Після розробки та тестування нова функція може бути інтегрована в основну версію проекту.

КбПЗ_2024

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мова програмування JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/

2. Introduction to Node.js [Електронний ресурс] – Режим доступу до ресурсу:https://nodejs.dev/

3. Серверное программирование веб-сайтов [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side

4. Node.js v14.0.0 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/api/

5. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих // “Node.js в действии”.

6. Янг А., Мек Б., Кантелон М. // “Node.js в действии. 2-е издание”.

7. John Resig, Bear Bibeault, Josip Maras // “Secrets of theJavaScript Ninja”.

8. Express [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/

9. Фреймворк AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AngularJS

10. AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/

11. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/

12. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://https://uk.reactjs.org/

13. AngularJS MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

14. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

КБПЗ_2024

15. Angular 2 [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

16. Односторінковий застосунок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Односторінковий_застосунок

17. Что такое MVC [Електронний ресурс] – Режим доступу до ресурсу: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami

18. Build Node.js Apps [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

19. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

20. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

21.Тренди веб-разробки [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/451572/

22.Веб додаток [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/

23.Мова розмітки гіпертексту [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML

24. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/HTML

25. Каскадні таблиці стилів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS

26. Стек MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MEAN_(веброзробка)

27. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB

28. Веб-технології для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web

29. CORS, XSS [Електронний ресурс] – Режим доступу до ресурсу:

КОПЗ-2024

https://dev.to/maleta/cors-xss-and-csrf-with-examples-in-10-minutes-35k3

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX

31. Fetch API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API

32. AngularJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/angular/default.asp

33. jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

34. Основи Web-технологій [Електронний ресурс] – Режим доступу до ресурсу: https://pidruchniki.com/1243020547796/informatika/web-tehnologiyi_pidpriyemstvah

35. npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/

36. Install MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/manual/administration/install-on-linux/

37. Как установить Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/node-js-ubuntu-18-04-ru

38. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux

39. npm-audit [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/cli/audit

40. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

41. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

42. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass

43. Postman [Електронний ресурс] – Режим доступу до ресурсу:

КБПЗ-2024

https://www.postman.com/

44. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

45. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA

55. The Top JavaScript Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/

56. JavaScript Frameworks 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/migrate-mongo

57. Web Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/web-technology/

КБПЗ_2024

Додаток А
(обов’язковий)

Технічне завдання

Зміст

КПЗ-2024

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи кібербезпеки Web-додатку з впровадженням e-service. Підставою для розробки служить завдання на випускну кваліфікаційну роботу, видане на кафедрі програмування та захисту інформації (нак. №132-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної бакалаврської роботи є розробка програмна реалізація ІС Web-додатку з впровадженням e-service та системою кіберзахисту.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської дипломної роботи є відносна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

КПЗ_2024

5.2 Показники призначення

Система повинна забезпечувати:

- розробку додатку;
- систему підключення та тестування баз даних ;
- цілісність даних у процесі роботи та при зберіганні;
- простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

КПЗ_2024

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

При розробці ПЗ потрібно використовувати наступні технології та мови програмування: фреймворк Angular та СУБД MySQL.

Середовище програмування – мова програмування TypeScript та фреймворку Angular

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

КОПЗ – 2024

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи керування – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок схема алгоритму програми – 1 аркуш.
– Блок-схема підпрограми – 1 аркуш.
– Пояснювальна записка – 84 аркуша.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи.

Постановка задачі на виконання кваліфікаційної роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

КПЗ_2024

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання кваліфікаційної роботи на попередній захист 20.05.2024 р.

9.2 Подання кваліфікаційної роботи на захист 05.06.2024 р.

КбПЗ_2024

Додаток Б
(обов'язковий)

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник кваліфікаційної бакалаврської роботи
_____ Грант Д.В

Програмне забезпечення системи кібербезпеки навігації техніки на базі Arduino

Лістинг програми

Код документу 12
Носій: CD/DVD-диск

Загальна кількість аркушів: 15

Літера: РП

КбПЗ_2024

Кропивницький – 2024 року

Файл основного файлу програми для завантаження AngularJS

```
<!DOCTYPE html>
<html ng-app="app">
<head>
    <meta charset="utf-8" />
    <title>AngularJS</title>
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" />
    <link href="app-content/app.css" rel="stylesheet" />
</head>
<body>
    <div class="jumbotron2">
        <div class="container">
            <div class="col-sm-8 col-sm-offset-2">
                <div ng-class="{ 'alert': flash, 'alert-success':
flash.type === 'success', 'alert-danger': flash.type === 'error' }" ng-
if="flash" ng-bind="flash.message"></div>
                <div ng-view></div>
            </div>
        </div>
    </div>

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
route.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
cookies.min.js"></script>

    <script src="app.js"></script>
    <script src="app-services/authentication.service.js"></script>
    <script src="app-services/flash.service.js"></script>

    <script src="app-services/user.service.local-storage.js"></script>

    <script src="home/home.controller.js"></script>
    <script src="login/login.controller.js"></script>
    <script src="register/register.controller.js"></script>
    <script src="sidebar/sidebar.controller.js"></script>
    <script src="chat/chat.controller.js"></script>
    <script src="technologies/technologies.controller.js"></script>
    <script src="company/company.controller.js"></script>
    <script src="settings/settings.controller.js"></script>
</body>
```

КОПІЮ 2024

```
</html>
```

Файл який відповідає за маршрутизацію додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app', ['ngRoute', 'ngCookies'])
        .config(config)
        .run(run);

    config.$inject = ['$routeProvider', '$locationProvider'];
    function config($routeProvider, $locationProvider) {
        $routeProvider
            .when('/', {
                controller: 'HomeController',
                templateUrl: 'home/home.view.html',
                controllerAs: 'vm'
            })
            .when('/login', {
                controller: 'LoginController',
                templateUrl: 'login/login.view.html',
                controllerAs: 'vm'
            })
            .when('/register', {
                controller: 'RegisterController',
                templateUrl: 'register/register.view.html',
                controllerAs: 'vm'
            })
            .when('/sidebar', {
                controller: 'RegisterController',
                templateUrl: 'sidebar/sidebar.view.html',
                controllerAs: 'vm'
            })
            .when('/technologies', {
                controller: 'TechnologiesController',
                templateUrl: 'technologies/technologies.view.html',
                controllerAs: 'vm'
            })
            .when('/settings', {
                controller: 'SettingsController',
                templateUrl: 'settings/settings.view.html',
                controllerAs: 'vm'
            })
            .when('/company', {
```

КбПЗ_2024

```
                    controller: 'CompanyController',
                    templateUrl: 'company/company.view.html',
                    controllerAs: 'vm'
                })
                .when('/chat', {
                    controller: 'ChatController',
                    templateUrl: 'chat/chat.view.html',
                    controllerAs: 'vm'
                })
                .otherwise({ redirectTo: '/login' });
        }

        run.$inject = ['$rootScope', '$location', '$cookies', '$http'];
        function run($rootScope, $location, $cookies, $http) {
            $rootScope.globals = $cookies.getObject('globals') || {};
            if ($rootScope.globals.currentUser) {
                $http.defaults.headers.common['Authorization'] = 'Basic ' +
$rootScope.globals.currentUser.authdata;
            }
        }

    })();
```

Файл який відповідає логіку реєстрації додатка на AngularJS

```
    (function () {
        'use strict';

        angular
            .module('app')
            .controller('RegisterController', RegisterController);

        RegisterController.$inject = ['UserService', '$location',
'$rootScope', 'FlashService'];
        function RegisterController(UserService, $location, $rootScope,
FlashService) {
            var vm = this;

            vm.register = register;

            function register() {
                vm.dataLoading = true;
                UserService.Create(vm.user)
                    .then(function (response) {
                        if (response.success) {
                            FlashService.Success('Registration successful',
true);
```

КБПЗ_2024

```
                            $location.path('/login');
                        } else {
                            FlashService.Error(response.message);
                            vm.dataLoading = false;
                        }
                    });
            }
        }

    })();
```

Файл який відповідає за представлення реєстрації додатка на AngularJS

```
    <div class="col-md-6 col-md-offset-3">
        <h2>Register</h2>
        <form name="form" ng-submit="vm.register()" role="form">
            <div class="form-group" ng-class="{ 'has-error':
form.firstName.$dirty && form.firstName.$error.required }">
                <label for="username">First name</label>
                <input type="text" name="firstName" id="firstName"
class="form-control" ng-model="vm.user.firstName" required />
                <span ng-show="form.firstName.$dirty &&
form.firstName.$error.required" class="help-block">First name is required</span>
            </div>
            <div class="form-group" ng-class="{ 'has-error':
form.lastName.$dirty && form.lastName.$error.required }">
                <label for="username">Last name</label>
                <input type="text" name="lastName" id="Text1" class="form-
control" ng-model="vm.user.lastName" required />
                <span ng-show="form.lastName.$dirty &&
form.lastName.$error.required" class="help-block">Last name is required</span>
            </div>
            <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                <label for="username">Username</label>
                <input type="text" name="username" id="username" class="form-
control" ng-model="vm.user.username" required />
                <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
            </div>
            <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                <label for="password">Password</label>
                <input type="password" name="password" id="password"
class="form-control" ng-model="vm.user.password" required />
```

КОПЗ_2024

```
                <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
            </div>
            <div class="form-actions">
                <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Register</button>
                <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                <a href="#!/login" class="btn btn-link">Cancel</a>
            </div>
        </form>
    </div>
```

Файл який відповідає за представлення логіна додатка на AngularJS

```
    <div class="col-md-6 col-md-offset-3">
        <h2>Login</h2>
        <form name="form" ng-submit="vm.login()" role="form">
            <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                <label for="username">Username</label>
                <input type="text" name="username" id="username" class="form-
control" ng-model="vm.username" required />
                <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
            </div>
            <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                <label for="password">Password</label>
                <input type="password" name="password" id="password"
class="form-control" ng-model="vm.password" required />
```

КбПЗ_2024

```
                <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
            </div>
            <div class="form-actions">
                <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Login</button>
                <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                <a href="#!/register" class="btn btn-link">Register</a>
            </div>
        </form>
    </div>

    <div ng-controller="SidebarController">
        <div ng-class="'test'">
            {{phone.name}}
        </div>
    </div>
```

Файл який відповідає за логіку логіна додатка на AngularJS

```
    (function () {
        'use strict';

        angular
            .module('app')
            .controller('LoginController', LoginController);

        LoginController.$inject = ['$location', 'AuthenticationService',
'FlashService'];
        function LoginController($location, AuthenticationService,
FlashService) {
            var vm = this;

            vm.login = login;
```

КОПІЯ_2024

```
            (function initController() {
                // reset login status
                AuthenticationService.ClearCredentials();
            })();

            function login() {
                vm.dataLoading = true;
                AuthenticationService.Login(vm.username, vm.password, function
(response) {
                    if (response.success) {
                        AuthenticationService.SetCredentials(vm.username,
vm.password);
                        $location.path('/');
                    } else {
                        FlashService.Error(response.message);
                        vm.dataLoading = false;
                    }
                });
            };
        }

    })();
```

Файл для завантаження сервера main.ts

```
import 'module-alias/register';

import { NestFactory } from '@nestjs/core';
import { ValidationPipe, ValidationError } from '@nestjs/common';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
import ValidationExceptions from './exceptions/validation.exceptions';

import AppModule from './routes/app/app.module';

import AllExceptionsFilter from './filters/all-exceptions.filter';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe({
    exceptionFactory: (errors: ValidationError[]) => new
ValidationExceptions(errors),
  }));
  app.useGlobalFilters(new AllExceptionsFilter());

  const port = process.env.SERVER_PORT || 3000;
```

КПЗ_2024

```
    const options = new DocumentBuilder()
      .setTitle('Api v1')
      .setDescription('The boilerplate API for nestjs devs')
      .setVersion('1.0')
      .addBearerAuth({ in: 'header', type: 'http' })
      .build();
    const document = SwaggerModule.createDocument(app, options);

    SwaggerModule.setup('api', app, document);

    await app.listen(port, async () => {
      console.log(`The server is running on ${port} port:
http://localhost:${port}/api`);
    });
  }
  bootstrap();
```

Файл контролер для реєстрації та длгіну на сервері

```
  import {
    Body,
    Controller,
    HttpCode,
    Get,
    Post,
    Delete,
    Param,
    Request,
    UnauthorizedException,
    UseGuards,
    NotFoundException,
    ForbiddenException,
    HttpStatus,
    UseInterceptors,
  } from '@nestjs/common';
  import {
    ApiTags,
    ApiBody,
    ApiOkResponse,
    ApiInternalServerErrorResponse,
    ApiUnauthorizedResponse,
    ApiBearerAuth,
    ApiNotFoundResponse,
    ApiBadRequestResponse,
    ApiConflictResponse,
    ApiNoContentResponse,
```

КбПЗ_2024

```
  ApiExtraModels,
  getSchemaPath,
} from '@nestjs/swagger';
import { JwtService } from '@nestjs/jwt';
import { Request as ExpressRequest } from 'express';
import { MailerService } from '@nestjs-modules/mailer';

import UsersService from '@v1/users/users.service';
import JwtAccessGuard from '@guards/jwt-access.guard';
import RolesGuard from '@guards/roles.guard';
import { User } from '@v1/users/schemas/users.schema';
import WrapResponseInterceptor from '@interceptors/wrap-
response.interceptor';
import AuthBearer from '@decorators/auth-bearer.decorator';
import { Roles, RolesEnum } from '@decorators/roles.decorator';
import authConstants from '@v1/auth/auth-constants';
import { DecodedUser } from './interfaces/decoded-user.interface';
import LocalAuthGuard from './guards/local-auth.guard';
import AuthService from './auth.service';
import RefreshTokenDto from './dto/refresh-token.dto';
import SignInDto from './dto/sign-in.dto';
import SignUpDto from './dto/sign-up.dto';
import JwtTokensDto from './dto/jwt-tokens.dto';
import UsersEntity from '@v1/users/entity/user.entity';

@ApiTags('Auth')
@UseInterceptors(WrapResponseInterceptor)
@ApiExtraModels(JwtTokensDto)
@Controller()
export default class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly jwtService: JwtService,
    private readonly usersService: UsersService,
    private readonly mailerService: MailerService,
  ) {}

  @ApiBody({ type: SignInDto })
  @ApiOkResponse({
    schema: {
      type: 'object',
      properties: {
        data: {
          $ref: getSchemaPath(JwtTokensDto),
        },
      },
```

КОПЗ_2024

```
  },
  description: 'Returns jwt tokens',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
      message: [
        {
          target: {
            email: 'string',
            password: 'string',
          },
          value: 'string',
          property: 'string',
          children: [],
          constraints: {},
        },
      ],
      error: 'Bad Request',
    },
  },
  description: '400. ValidationException',
})
@ApiInternalServerErrorResponse({
  schema: {
    type: 'object',
    example: {
      message: 'string',
      details: {},
    },
  },
  description: '500. InternalServerError',
})
@ApiBearerAuth()
@HttpCode(HttpStatus.OK)
@UseGuards(LocalAuthGuard)
@Post('sign-in')
async signIn(@Request() req: ExpressRequest): Promise<JwtTokensDto> {
  const user = req.user as User;

  return this.authService.login(user);
}

@ApiBody({ type: SignUpDto })
@ApiOkResponse({
```

КОПЗ_2024

```
  description: '201, Success',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
      message: [
        {
          target: {
            email: 'string',
            password: 'string',
          },
          value: 'string',
          property: 'string',
          children: [],
          constraints: {},
        },
      ],
      error: 'Bad Request',
    },
  },
  description: '400. ValidationException',
})
@ApiConflictResponse({
  schema: {
    type: 'object',
    example: {
      message: 'string',
    },
  },
  description: '409. ConflictResponse',
})
@ApiInternalServerErrorResponse({
  schema: {
    type: 'object',
    example: {
      message: 'string',
      details: {},
    },
  },
  description: '500. InternalServerError',
})
@HttpCode(HttpStatus.CREATED)
@Post('sign-up')
async signUp(@Body() user: SignUpDto): Promise<any> {
```

КОПЗ_2024

```
    const { _id, email } = await this.usersService.create(user) as
UsersEntity;

    const token = this.authService.createVerifyToken(_id);

    await this.mailerService.sendMail({
      to: email,
      from: process.env.MAILER_FROM_EMAIL,
      subject: authConstants.mailer.verifyEmail.subject,
      template: `${process.cwd()}/src/templates/verify-password`,
      context: {
        token,
        email,
        host: process.env.SERVER_HOST,
      },
    });

    return { message: 'Success! please verify your email' };
  }

  @ApiOkResponse({
    schema: {
      type: 'object',
      properties: {
        data: {
          $ref: getSchemaPath(JwtTokensDto),
        },
      },
    },
    description: '200, returns new jwt tokens',
  })
  @ApiUnauthorizedResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
      },
    },
    description: '401. Token has been expired',
  })
  @ApiInternalServerErrorResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
        details: {},
```

КБПЗ_2024

```
          },
        },
        description: '500. InternalServerError ',
      })
      @ApiBearerAuth()
      @Post('refresh-token')
      async refreshToken(
        @Body() refreshTokenDto: RefreshTokenDto,
      ): Promise<JwtTokensDto | never>  {
        const decodedUser = this.jwtService.decode(
          refreshTokenDto.refreshToken,
        ) as DecodedUser;

        if (!decodedUser) {
          throw new ForbiddenException('Incorrect token');
        }

        const oldRefreshToken:
          | string
          | null = await
this.authService.getRefreshTokenByEmail(decodedUser.email);

        // if the old refresh token is not equal to request refresh token then
this user is unauthorized
        if (!oldRefreshToken || oldRefreshToken !==
refreshTokenDto.refreshToken) {
          throw new UnauthorizedException(
            'Authentication credentials were missing or incorrect',
          );
        }

        const payload = {
          _id: decodedUser._id,
          email: decodedUser.email,
          role: decodedUser.role,
        };

        return this.authService.login(payload);
      }

      @ApiNoContentResponse({
        description: 'No content. 204',
      })
      @ApiNotFoundResponse({
        schema: {
          type: 'object',
```

КОПІЯ 2024

```
            example: {
              message: 'string',
              error: 'Not Found',
            },
          },
          description: 'User was not found',
        })
        @HttpCode(HttpStatus.NO_CONTENT)
        @Get('verify/:token')
        async verifyUser(@Param('token') token: string): Promise<User | null> {
          const { id } = await this.authService.verifyEmailVerToken(
            token,
            authConstants.jwt.secrets.accessToken,
          );
          const foundUser = await this.usersService.getUnverifiedUserById(id) as
UsersEntity;

          if (!foundUser) {
            throw new NotFoundException('The user does not exist');
          }

          return this.usersService.update(foundUser._id, { verified: true });
        }

        @ApiNoContentResponse({
          description: 'no content',
        })
        @ApiUnauthorizedResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
            },
          },
          description: 'Token has been expired',
        })
        @ApiInternalServerErrorResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              details: {},
            },
          },
          description: 'InternalServerError',
        })
```

КОПЗ_2024

```
  @ApiBearerAuth()
  @UseGuards(JwtAccessGuard)
  @Delete('logout/:token')
  @HttpCode(HttpStatus.NO_CONTENT)
  async logout(@Param('token') token: string): Promise<{} | never> {
    const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
      token,
      authConstants.jwt.secrets.accessToken,
    );

    if (!decodedUser) {
      throw new ForbiddenException('Incorrect token');
    }

    const deletedUsersCount = await this.authService.deleteTokenByEmail(
      decodedUser.email,
    );

    if (deletedUsersCount === 0) {
      throw new NotFoundException();
    }
    return {};
  }

  @ApiNoContentResponse({
    description: 'no content',
  })
  @ApiInternalServerErrorResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
        details: {},
      },
    },
    description: '500. InternalServerError',
  })
  @ApiBearerAuth()
  @Delete('logout-all')
  @UseGuards(RolesGuard)
  @Roles(RolesEnum.admin)
  @HttpCode(HttpStatus.NO_CONTENT)
  async logoutAll(): Promise<{}> {
    return this.authService.deleteAllTokens();
  }
```

КОПЗ_2024

```
    @ApiOkResponse({
      type: User,
      description: '200, returns a decoded user from access token',
    })
    @ApiUnauthorizedResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
        },
      },
      description: '403, says you Unauthorized',
    })
    @ApiInternalServerErrorResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
          details: {},
        },
      },
      description: '500. InternalServerError',
    })
    @ApiBearerAuth()
    @UseGuards(JwtAccessGuard)
    @Get('token')
    async getUserByAccessToken(
      @AuthBearer() token: string,
    ): Promise<DecodedUser | never> {
      const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
        token,
        authConstants.jwt.secrets.accessToken,
      );

      if (!decodedUser) {
        throw new ForbiddenException('Incorrect token');
      }

      const { exp, iat, ...user } = decodedUser;

      return user;
    }
  }
```

Файл сервіс для реєстрації та лгіну на сервері

КБПЗ_2024

```
import * as bcrypt from 'bcrypt';

import { Injectable, NotFoundException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Types } from 'mongoose';

import UsersRepository from '@v1/users/users.repository';
import { UserInterface } from '@v1/users/interfaces/user.interface';
import { DecodedUser } from './interfaces/decoded-user.interface';
import JwtTokensDto from './dto/jwt-tokens.dto';
import { LoginPayload } from './interfaces/login-payload.interface';

import authConstants from './auth-constants';
import AuthRepository from './auth.repository';
import UsersEntity from '@v1/users/entity/user.entity';

@Injectable()
export default class AuthService {
  constructor(
    private readonly jwtService: JwtService,
    private readonly usersRepository: UsersRepository,
    private readonly authRepository: AuthRepository,
  ) {}

  public async validateUser(
    email: string,
    password: string,
  ): Promise<null | UserInterface> {
    const user = await this.usersRepository.getVerifiedUserByEmail(email)
as UsersEntity;

    if (!user) {
      throw new NotFoundException('The item does not exist');
    }

    const passwordCompared = await bcrypt.compare(password,
user.password);

    if (passwordCompared) {
      return {
        _id: user._id,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
```

КПЗ_2024

```
    refreshToken,
  );

  return {
    accessToken,
    refreshToken,
  };
}

public getRefreshTokenByEmail(email: string): Promise<string | null> {
  return this.authRepository.getToken(email);
}

public deleteTokenByEmail(email: string): Promise<number> {
  return this.authRepository.removeToken(email);
}

public deleteAllTokens(): Promise<string> {
  return this.authRepository.removeAllTokens();
}

public createVerifyToken(id: Types.ObjectId): string {
  return this.jwtService.sign(
    { id },
    {
      expiresIn: authConstants.jwt.expirationTime.accessToken,
      secret: authConstants.jwt.secrets.accessToken,
    },
  );
}

public verifyEmailVerToken(token: string, secret: string) {
  return this.jwtService.verifyAsync(token, { secret });
}

public async verifyToken(
  token: string,
  secret: string,
): Promise<DecodedUser | null> {
  try {
    const user = (await this.jwtService.verifyAsync(token, {
      secret,
    })) as DecodedUser | null;

    return user;
  } catch (error) {
```

КбПЗ_2024

```
      return null;
    }
  }
}     email: user.email,
        role: user.role,
      };
    }

    return null;
  }

  public async login(data: LoginPayload): Promise<JwtTokensDto> {
    const payload: LoginPayload = {
      _id: data._id,
      email: data.email,
      role: data.role,
    };

    const accessToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.accessToken,
      secret: authConstants.jwt.secrets.accessToken,
    });
    const refreshToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.refreshToken,
      secret: authConstants.jwt.secrets.refreshToken,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
      refreshToken,
    );

    return {
      accessToken,
      refreshToken,
    };
  }

  public getRefreshTokenByEmail(email: string): Promise<string | null> {
    return this.authRepository.getToken(email);
  }

  public deleteTokenByEmail(email: string): Promise<number> {
    return this.authRepository.removeToken(email);
  }
```

КПЗ_2024

```
  public deleteAllTokens(): Promise<string> {
    return this.authRepository.removeAllTokens();
  }

  public createVerifyToken(id: Types.ObjectId): string {
    return this.jwtService.sign(
      { id },
      {
        expiresIn: authConstants.jwt.expirationTime.accessToken,
        secret: authConstants.jwt.secrets.accessToken,
      },
    );
  }

  public verifyEmailVerToken(token: string, secret: string) {
    return this.jwtService.verifyAsync(token, { secret });
  }

  public async verifyToken(
    token: string,
    secret: string,
  ): Promise<DecodedUser | null> {
    try {
      const user = (await this.jwtService.verifyAsync(token, {
        secret,
      })) as DecodedUser | null;

      return user;
    } catch (error) {
      return null;
    }
  }
}
```

Файл репозиторій для реєстрації та лгіну на сервері

```
import * as Redis from 'ioredis';
import { Injectable } from '@nestjs/common';

import { RedisService } from 'nestjs-redis';
import authConstants from './auth-constants';

@Injectable()
export default class AuthRepository {
  private readonly redisClient: Redis.Redis;
```

КПЗ_2024

```
        constructor(private readonly redisService: RedisService) {
          this.redisClient = redisService.getClient();
        }

        public async addRefreshToken(userEmail: string, refreshToken: string):
Promise<void> {
          await this.redisClient.set(
            userEmail,
            refreshToken,
            'EX',
            authConstants.redis.expirationTime.jwt.refreshToken,
          );
        }

        public getToken(key: string): Promise<string | null> {
          return this.redisClient.get(key);
        }

        public removeToken(key: string): Promise<number> {
          return this.redisClient.del(key);
        }

        public removeAllTokens(): Promise<string> {
          return this.redisClient.flushall();
        }
      }
```

Файл модуль для маршрутизації в Angular

```
      import {NO_ERRORS_SCHEMA, NgModule} from '@angular/core';
      import { Routes, RouterModule } from '@angular/router';

      import { HomeComponent } from './home';
      import { AuthGuard } from './_helpers';

      const accountModule = () => import('./account/account.module').then(x =>
x.AccountModule);
      const usersModule = () => import('./users/users.module').then(x =>
x.UsersModule);
      const profileModule = () => import('./profile/profile.module').then(x =>
x.ProfileModule);
      const documentModule = () => import('./document/document.model').then(x =>
x.DocumentModule);
      const documentChat = () => import('./chat/chat.model').then(x =>
x.ChatModule);
      const documentCompany = () => import('./company/company.model').then(x =>
x.CompanyModule);
```

КБПЗ_2024

```
    const documentTechnologies = () =>
import('./technologies/technologies.model').then(x => x.TechnologiesModule);
    const documentSettings = () => import('./settings/settings.model').then(x
=> x.SettingsModule);

    const routes: Routes = [
        { path: '', component: HomeComponent, canActivate: [AuthGuard] },
        { path: 'users', loadChildren: usersModule, canActivate: [AuthGuard]
},
        { path: 'account', loadChildren: accountModule },
        { path: 'profile', loadChildren: profileModule },
        { path: 'document', loadChildren: documentModule },
        { path: 'chat', loadChildren: documentChat },
        { path: 'company', loadChildren: documentCompany },
        { path: 'technologies', loadChildren: documentTechnologies },
        { path: 'settings', loadChildren: documentSettings },

        { path: '**', redirectTo: '' }
    ];

    @NgModule({
        imports: [RouterModule.forRoot(routes)],
        exports: [RouterModule],
        schemas: [NO_ERRORS_SCHEMA]
    })
    export class AppRoutingModule { }
```

Файл який відповідає за логіку логіна додатка на Angular

```
    import { Component, OnInit } from '@angular/core';
    import { Router, ActivatedRoute } from '@angular/router';
    import { FormBuilder, FormGroup, Validators } from '@angular/forms';
    import { first } from 'rxjs/operators';

    import { AccountService, AlertService } from '@app/_services';

    @Component({ templateUrl: 'login.component.html' })
    export class LoginComponent implements OnInit {
        form: FormGroup;
        loading = false;
        submitted = false;
        returnUrl: string;

        constructor(
            private formBuilder: FormBuilder,
            private route: ActivatedRoute,
```

КПЗ_2024

```
            private router: Router,
            private accountService: AccountService,
            private alertService: AlertService
        ) { }

        ngOnInit() {
            this.form = this.formBuilder.group({
                username: ['', Validators.required],
                password: ['', Validators.required]
            });

            this.returnUrl = this.route.snapshot.queryParams['returnUrl'] ||
'/';
        }

        get f() { return this.form.controls; }

        onSubmit() {
            this.submitted = true;

            this.alertService.clear();

            if (this.form.invalid) {
                return;
            }

            this.loading = true;
            this.accountService.login(this.f.username.value,
this.f.password.value)
                .pipe(first())
                .subscribe(
                    data => {
                        this.router.navigate([this.returnUrl]);
                    },
                    error => {
                        this.alertService.error(error);
                        this.loading = false;
                    });
        }
    }
```

Файл який відповідає за представлення логіна додатка на Angular

```
<div class="card">
    <h4 class="card-header">Login</h4>
    <div class="card-body">
```

КОПЗ_2024

```
            <form [formGroup]="form" (ngSubmit)="onSubmit()">
                <div class="form-group">
                    <label for="username">Username</label>
                    <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                    <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                        <div *ngIf="f.username.errors.required">Username is
required</div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="password">Password</label>
                    <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors
}" />
                    <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                        <div *ngIf="f.password.errors.required">Password is
required</div>
                    </div>
                </div>
                <div class="form-group">
                    <button [disabled]="loading" class="btn btn-primary">
                        <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                        Login
                    </button>
                    <a routerLink="../register" class="btn btn-
link">Register</a>
                </div>
            </form>
        </div>
    </div>
```

Файл який відповідає за логіку реєстрації додатка на Angular

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'register.component.html' })
export class RegisterComponent implements OnInit {
```

КбПЗ_2024

```
    form: FormGroup;
    loading = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            firstName: ['', Validators.required],
            lastName: ['', Validators.required],
            username: ['', Validators.required],
            password: ['', [Validators.required, Validators.minLength(6)]]
        });
    }

    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        this.alertService.clear();

        if (this.form.invalid) {
            return;
        }

        this.loading = true;
        this.accountService.register(this.form.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.alertService.success('Registration successful', {
keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo:
this.route });
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
```

КбПЗ_2024

```
        }
    }
```

Файл який відповідає за представлення реєстрації додатка на Angular

```
    <div class="card">
        <h4 class="card-header">Register</h4>
        <div class="card-body">
            <form [formGroup]="form" (ngSubmit)="onSubmit()">
                <div class="form-group">
                    <label for="firstName">First Name</label>
                    <input type="text" formControlName="firstName"
class="form-control" [ngClass]="{ 'is-invalid': submitted && f.firstName.errors
}" />
                    <div *ngIf="submitted && f.firstName.errors"
class="invalid-feedback">
                        <div *ngIf="f.firstName.errors.required">First Name is
required</div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="lastName">Last Name</label>
                    <input type="text" formControlName="lastName" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.lastName.errors }" />
                    <div *ngIf="submitted && f.lastName.errors"
class="invalid-feedback">
                        <div *ngIf="f.lastName.errors.required">Last Name is
required</div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="username">Username</label>
                    <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                    <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                        <div *ngIf="f.username.errors.required">Username is
required</div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="password">Password</label>
                    <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors
}" />
```

КБПЗ_2024

```
                    <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                        <div *ngIf="f.password.errors.required">Password is
required</div>
                        <div *ngIf="f.password.errors.minlength">Password must
be at least 6 characters</div>
                    </div>
                </div>
                <div class="form-group">
                    <button [disabled]="loading" class="btn btn-primary">
                        <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                        Register
                    </button>
                    <a routerLink="../login" class="btn btn-link">Cancel</a>
                </div>
            </form>
        </div>
    </div>
```

Файл сервіс який відповідає за юзера додатка на Angular

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { environment } from '@environments/environment';
import { User } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class AccountService {
    private userSubject: BehaviorSubject<User>;
    public user: Observable<User>;

    constructor(
        private router: Router,
        private http: HttpClient
    ) {
        this.userSubject = new
BehaviorSubject<User>(JSON.parse(localStorage.getItem('user')));
        this.user = this.userSubject.asObservable();
    }

    public get userValue(): User {
```

КОПІЯ 2024

```
            return this.userSubject.value;
        }

        login(username, password) {
            return
this.http.post<User>(`${environment.apiUrl}/users/authenticate`, { username,
password })
                .pipe(map(user => {
                    localStorage.setItem('user', JSON.stringify(user));
                    this.userSubject.next(user);
                    return user;
                }));
        }

        logout() {
            localStorage.removeItem('user');
            this.userSubject.next(null);
            this.router.navigate(['/account/login']);
        }

        register(user: User) {
            return this.http.post(`${environment.apiUrl}/users/register`,
user);
        }

        getAll() {
            return this.http.get<User[]>(`${environment.apiUrl}/users`);
        }

        getById(id: string) {
            return this.http.get<User>(`${environment.apiUrl}/users/${id}`);
        }

        update(id, params) {
            return this.http.put(`${environment.apiUrl}/users/${id}`, params)
                .pipe(map(x => {
                    if (id == this.userValue.id) {
                        const user = { ...this.userValue, ...params };
                        localStorage.setItem('user', JSON.stringify(user));

                        this.userSubject.next(user);
                    }
                    return x;
                }));
        }
```

КбПЗ_2024

```
    delete(id: string) {
        return this.http.delete(`${environment.apiUrl}/users/${id}`)
            .pipe(map(x => {
                if (id == this.userValue.id) {
                    this.logout();
                }
                return x;
            }));
    }
}
```

Файл схеми юзера на сервері

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

import { RolesEnum } from '@decorators/roles.decorator';

@Schema()
export class User {
  @Prop({
    required: true,
    unique: true,
    type: String,
  })
  email: string = '';

  @Prop({
    required: true,
    type: String,
  })
  password: string = '';

  @Prop({
    required: true,
    type: Boolean,
  })
  verified: boolean = false;

  @Prop({
    type: RolesEnum,
    required: false,
    default: RolesEnum.user,
  })
  role: RolesEnum = RolesEnum.user;
}
```

КбПЗ_2024

```
export type UserDocument = User & Document;

export const UserSchema =
SchemaFactory.createForClass(User).set('versionKey', false);
```

Файл головного модуля на сервері

```
import { Module } from '@nestjs/common';
import { RedisModule } from 'nestjs-redis';
import { ConfigModule } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { MailerModule } from '@nestjs-modules/mailer';
import { HandlebarsAdapter } from '@nestjs-
modules/mailer/dist/adapters/handlebars.adapter';
import V1Module from '@v1/v1.module';
import AppController from './app.controller';
import AppService from './app.service';
import AppGateway from './app.gateway';
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    MongooseModule.forRoot(process.env.MONGODB_URL as string, {
      autoReconnect: true,
      useCreateIndex: true,
      reconnectTries: Number.MAX_VALUE,
      reconnectInterval: 1000,
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }),
    RedisModule.register({
      url: process.env.REDIS_URL,
      onClientReady: async (client): Promise<void> => {
        client.on('error', console.error);
        client.on('ready', () => {
          console.log('redis is running on 6379 port');
        });
        client.on('restart', () => {
          console.log('attempt to restart the redis server');
        });
      },
      reconnectOnError: (): boolean => true,
    }),
    MailerModule.forRoot({
      transport: {
```

КОПЗ_2024

```
        host: process.env.MAILER_HOST,
        port: Number(process.env.MAILER_PORT),
        secure: false,
        auth: {
          user: process.env.MAILER_USERNAME,
          pass: process.env.MAILER_PASSWORD,
        },
      },
      defaults: {
        from: process.env.MAILER_FROM_EMAIL,
      },
      template: {
        adapter: new HandlebarsAdapter(),
        options: {
          strict: true,
        },
      },
    }),
    V1Module,
  ],
  controllers: [AppController],
  providers: [AppService, AppGateway],
})
export default class AppModule {}
```

Файл домашньої сторінки AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('HomeController', HomeController);

    HomeController.$inject = ['UserService', '$rootScope'];
    function HomeController(UserService, $rootScope) {
        var vm = this;

        vm.user = null;
        vm.allUsers = [];
        vm.deleteUser = deleteUser;

        initController();

        function initController() {
            loadCurrentUser();
            loadAllUsers();
```

КбПЗ_2024

```
            }

            function loadCurrentUser() {

UserService.GetByUsername($rootScope.globals.currentUser.username)
                    .then(function (user) {
                        vm.user = user;
                    });
            }

            function loadAllUsers() {
                UserService.GetAll()
                    .then(function (users) {
                        vm.allUsers = users;
                    });
            }

            function deleteUser(id) {
                UserService.Delete(id)
                .then(function () {
                    loadAllUsers();
                });
            }
        }

    })();
```

КбПЗ_2024