

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-06

Дипломний проект

здобувача освіти денної форми навчання
РП.06.12.000.ДП

***КОЛЕСНІЧЕНКО
ЄВГЕНІЯ
ВІТАЛІЙОВИЧА***

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-06

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка серверної частини та проектування баз даних для веб-сайту ВСП «ОТФК ОНТУ»

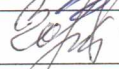
Проектний матеріал складається з пояснювальної записки на 70 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах).

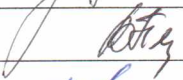
Дипломник  (Колесніченко Є.В.)

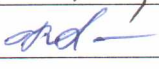
Керівник  (Шувалова І.О.)

Консультанти:

з економічної частини  (Копайгородська Т.Г.)

з охорони праці  (Чорновол Н.І.)

з дотримання вимог ЄСКД  (Петрашова В.І.)

старший консультант  (Кунуп Т.В.)

До захисту допущений

Голова циклової комісії  (Кривченко Ю.В.)

Завідувач відділення  (Скорнякова О.В.)

Захист « 23 » 06 2023 р.

Протокол ДКК № 2

Оцінка ДДК 5 (Відмінно)

Секретар ДДК 

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення комп'ютерних систем Комісія КТ та ПІ
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Розробка програмного забезпечення»

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР

Беркань І.В.

“ ” 2023 р.

ЗАВДАННЯ

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Колесніченко Євгенія Віталійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка серверної частини та проектування баз даних для сайту коледжа

затверджена наказом по коледжу від “ 17 ” жовтня 2023 р. № 235-А2-ОД

2. Термін здачі закінченого проекту (роботи) 09.06.2023

3. Вихідні данні до проекту (роботи)

1. Розробка серверної частини веб-сайту з використанням фреймворку ASP.NET 6

2. Використання OpenIdDict для аутентифікації та авторизації користувачів

3. Використання Entity Framework з модулем Npgsql для з'соднання та роботи з базою даних PostgreSQL

4. Використання OAuth для забезпечення авторизації користувачів

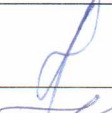
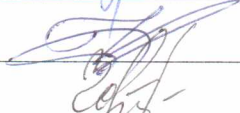
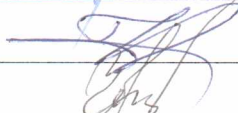
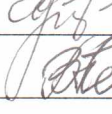
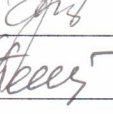
4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Опис предметної області. Огляд патернів для створення кодової бази. Загальний технічний опис вирішення задачі дипломного проекту. Огляд інструментів та програмних рішень. Проектування баз даних. Розробка серверної частини. Налаштування аутентифікації та авторизації. Розробка мікросервісів.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Титульний лист, Огляд проекту, Плюси обраної архітектури, Написання клієнтської частини на React.js + Next.js, Головна сторінка платформи, Написання мікросервісів на .NET та Python, Основні функціональні можливості, Переваги веб-додатку для коледжу, Ризики та виклики, Підсумок

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Технологічний розділ	Шувалова І.О.		
Економічна частина	Копайгородська Т.Г.		
Охорона праці	Чорновол В.І.		
Нормоконтроль	Петрашова В.І.		

7. Дата видачі завдання _____

Керівник

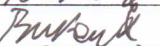
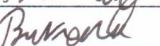
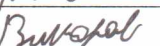
Шувалова І.О.


(підпис)

Завдання прийняв до виконання


(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	Вступ	15.05.2023	
2.	Аналіз предметної галузі	16.05.2023	
3.	Огляд інструментів та програмних рішень	17.05.2023	
4.	Проектування баз даних	23.05.2023	
5.	Розробка серверної частини	25.05.2023	
6.	Налаштування аутентиф. та авторизації	26.05.2023	
7.	Розробка мікросервісів	28.05.2023	
8.	Тестування	31.05.2023	
9.	Робота над Економічною частиною	04.06.2023	
10.	Робота над охороною праці	06.06.2023	
11.	Підготовка матеріалів для захисту	07.06.2023	

Дипломник


(підпис)

Керівник


(підпис)

[illegible]

ЗМІСТ

ВСТУП.....	6
1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ.....	7
1.1 Опис предметної області.....	7
1.2 Огляд паттернів для створення кодової бази.....	8
1.3 Загальний технічний опис вирішення задачі дипломного проекту.....	20
1.4 Огляд інструментів та програмних рішень.....	23
1.5 Проектування баз даних.....	28
1.6 Розробка серверної частини.....	34
1.7 Налаштування аутентифікації та авторизації.....	38
1.8 Розробка мікросервісів.....	42
2 ЕКОНОМІЧНА ЧАСТИНА.....	48
2.1 Резюме.....	48
2.2 Визначення трудомісткості розробки веб-додатку.....	48
3 ОХОРОНА ПРАЦІ.....	53
3.1 Гігієнічні вимоги до виробничого середовища.....	53
3.1.1 Вимоги до приміщення.....	53
3.1.2 Вимоги до організації робочого місця користувача ПК.....	54
3.2 Освітлення робочого місця.....	54
3.3 Мікроклімат.....	56
3.4 Іонізація повітря.....	56
3.5 Пожежна безпека.....	57
ВИСНОВКИ.....	58
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	59
ДОДАТОК А.....	61
ДОДАТОК Б.....	66

ВСТУП

Розробка серверної частини та проектування баз даних для сайту коледжу є невід'ємною складовою сучасного освітнього середовища. У цифрову еру, коли інформація та комунікація відіграють важливу роль у всіх сферах життя, важливо мати ефективні та зручні інструменти для забезпечення ефективної роботи коледжу, студентів та викладачів

Розробка серверної частини сайту коледжу передбачає створення програмного забезпечення, яке забезпечує його функціональність та доступність для користувачів. Це означає створення структурованої архітектури, використання мов програмування та фреймворків, що дозволяють обробляти запити, зберігати дані та забезпечувати безпеку інформації. Правильно розроблена серверна частина дозволяє забезпечити стабільну роботу сайту, швидку відповідь на запити користувачів та легке масштабування функціоналу в майбутньому.

Планування та розробка баз даних є важливим етапом в процесі розробки сайту коледжу. База даних забезпечує зберігання та організацію інформації про студентів, викладачів, навчальні плани, розклади занять та інші дані, необхідні для ефективної роботи коледжу. Вона дозволяє забезпечити швидкий доступ до інформації, виконувати складні запити та забезпечувати цілісність та безпеку даних. Ефективно спроектована база даних допомагає забезпечити організовану та систематичну роботу з інформацією, спрощує адміністрування та забезпечує точність та надійність даних.

Отже, розробка серверної частини та проектування баз даних для сайту коледжу є важливими кроками в створенні ефективного та функціонального освітнього середовища. Ці процеси допомагають забезпечити швидку та надійну роботу сайту, забезпечити доступ до необхідної інформації та покращити комунікацію між всіма учасниками навчального процесу.

					РП 06.12 000.00 ДП ПЗ	Арк.
						6
Змін.	Арк.	№ докум.	Підпис	Дата		

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Опис предметної області

У просторі теми дипломного проектування важливо охарактеризувати стандарти перебігу подій на стороні сервера і вибрати належні досягнення для їх виконання. Пошук відкритих відповідей для серверних додатків вже був випробуванням, але сьогодні доступні різні варіанти

Вирішення цих фундаментальних перспектив допомагає розрізнити, що з консервуванням може бути виконано з урахуванням обраних досягнень і передумов завдання. План інформаційних баз і методи вдосконалення є важливими для гарантування продуктивного накопичення і відновлення інформації. Загалом, основна увага в цьому проекті з розпізнавання зосереджена на покращенні корисності на стороні сервера та плануванні наборів даних для сайту, враховуючи стандарти перебігу подій на стороні сервера та використовуючи відповідні інновації. Покращення корисності серверної частини та планування наборів даних для шкільного сайту - це приголомшливе завдання, яке вимагає ніжного люблячого догляду та використання відповідних методів і пристроїв. Це завдання включає в себе планування інформаційної бази, яка буде працювати з інформаційною ємністю і асоціаціями. Удосконалення серверної частини включає в себе різні точки зору, наприклад, виконання обґрунтування сервера, задоволення запитів клієнтів, нагляд за інформацією та гарантування безпеки. Сюди входить вибір розумних діалектів програмування, структур і бібліотек, які забезпечать безперебійну роботу сайту. Код на стороні сервера буде займатися такими питаннями, як перевірка клієнтів, відновлення інформації, адміністрування контенту та співпраця між сайтом і клієнтами.

У зв'язку з розвитком подій на стороні сервера, планування структури набору даних має важливе значення для успішного розміщення та впорядкування інформації. Це включає в себе визначення субстанцій, зв'язків і властивостей, які будуть залучені до мапування інформаційної бази. Крім того, визначення відповідних типів інформації та обмежень гарантує достовірність

					РП 06.12 001.00 ДП ПЗ	Арк.
						7
Змін.	Арк.	№ докум.	Підпис	Дата		

інформації та її незмінну якість. План набору даних має сприяти кваліфікованому опитуванню та відновленню даних, а також забезпечувати адаптивність до майбутнього розвитку.

Завдяки ефективному підвищенню корисності на стороні сервера і плануванню професійного набору даних, веб сайт коледжу забезпечить послідовний клієнтський досвід, роботу з доступом до релевантних даних і підтримку фундаментальних регуляторних циклів. Поєднання цих компонентів гарантує надійну роботу виставки та довготривалу експлуатаційну придатність сайту.

1.2 Огляд паттернів для створення кодової бази

Перед написанням кодової бази потрібно визначитися з паттерном для кодової бази

Використання паттерну для кодової бази також сприяє полегшенню процесу розробки нових функцій або внесення змін до існуючого коду. Завдяки заздалегідь визначеній структурі, розробники можуть швидше орієнтуватися в кодовій базі і знаходити необхідні компоненти для роботи.

Крім того, використання паттерну дозволяє забезпечити більшу стійкість кодової бази до змін. Якщо розробники використовують спільні стандарти та шаблони, то внесення змін в одній частині коду не повинно негативно впливати на інші його компоненти. Це сприяє зменшенню кількості помилок та полегшує утримання кодової бази в робочому стані.

Визначення паттерну для кодової бази також допомагає новим розробникам швидше орієнтуватися у проекті. Вони можуть вивчати існуючі шаблони, принципи та підходи, що вже використовуються в проекті, і швидше внести свій внесок у розробку.

Узгодженість використання паттерну для кодової бази є основою для спільної роботи між розробниками. Вона створює єдиний мовний фон та спільне розуміння структури проекту. Це особливо важливо в командній розробці, де кілька розробників працюють над одним проектом одночасно.

Загалом, використання паттерну для кодової бази дозволяє забезпечити

					РП 06.12 001.00 ДП ПЗ	Арк.
						8
Змін.	Арк.	№ докум.	Підпис	Дата		

консистентність, структурованість та підтримуваність проекту, полегшуючи спільну роботу розробників та покращуючи якість розробленого коду.

Паттерн Фабричний метод дозволяє створювати об'єкти без прив'язки до конкретних класів. У цьому паттерні використовується абстрактний клас `FormAbstract`, який має абстрактний метод `CreateForm`. Підкласи `FormAbstract` успадковуються від нього і реалізують цей метод, повертаючи конкретний об'єкт типу `FormBuilder`.

Клас `FormAbstract` є абстрактним класом або інтерфейсом, який визначає загальні методи та властивості для всіх конкретних продуктів.

При виклику методу `CreateForm` у підкласі `FormAbstract` він створює і повертає об'єкт типу `Product`. Таким чином, залежно від підкласу `FormAbstract`, який використовується, ми отримуємо конкретний об'єкт `Product`, але код, який його викликає, залишається незмінним.

При використанні паттерну Фабричний метод ми можемо забезпечити розширюваність і відділення логіки створення об'єктів від коду, який їх використовує. Це означає, що за необхідності додати новий тип об'єкта ми просто створюємо новий підклас `FormAbstract` і реалізуємо метод `CreateForm`, визначаючи, який конкретний об'єкт типу `Product` має бути створений. Це рішення дає змогу уникнути прив'язки до конкретних класів у клієнтському коді. Замість того, щоб безпосередньо створювати об'єкти, клієнтський код взаємодіє з абстрактним `FormAbstract` і викликає метод `CreateForm`. Потім конкретний `FormAbstract` визначає, який конкретний об'єкт типу `Product` потрібно створити і повертає його клієнту.

Цей паттерн також сприяє дотриманню принципу єдиної відповідальності. Логіка створення об'єктів відділяється від логіки, пов'язаної з використанням цих об'єктів. Кожен клас відповідає тільки за свою відповідальність, що полегшує розуміння і зміну коду. Загалом, паттерн дає змогу створювати об'єкти, відокремлюючи процес їх створення від клієнтського коду. Це забезпечує гнучкість, розширюваність і простоту змін у майбутньому.

					РП 06.12 001.00 ДП ПЗ	Арк.
						9
Змін.	Арк.	№ докум.	Підпис	Дата		

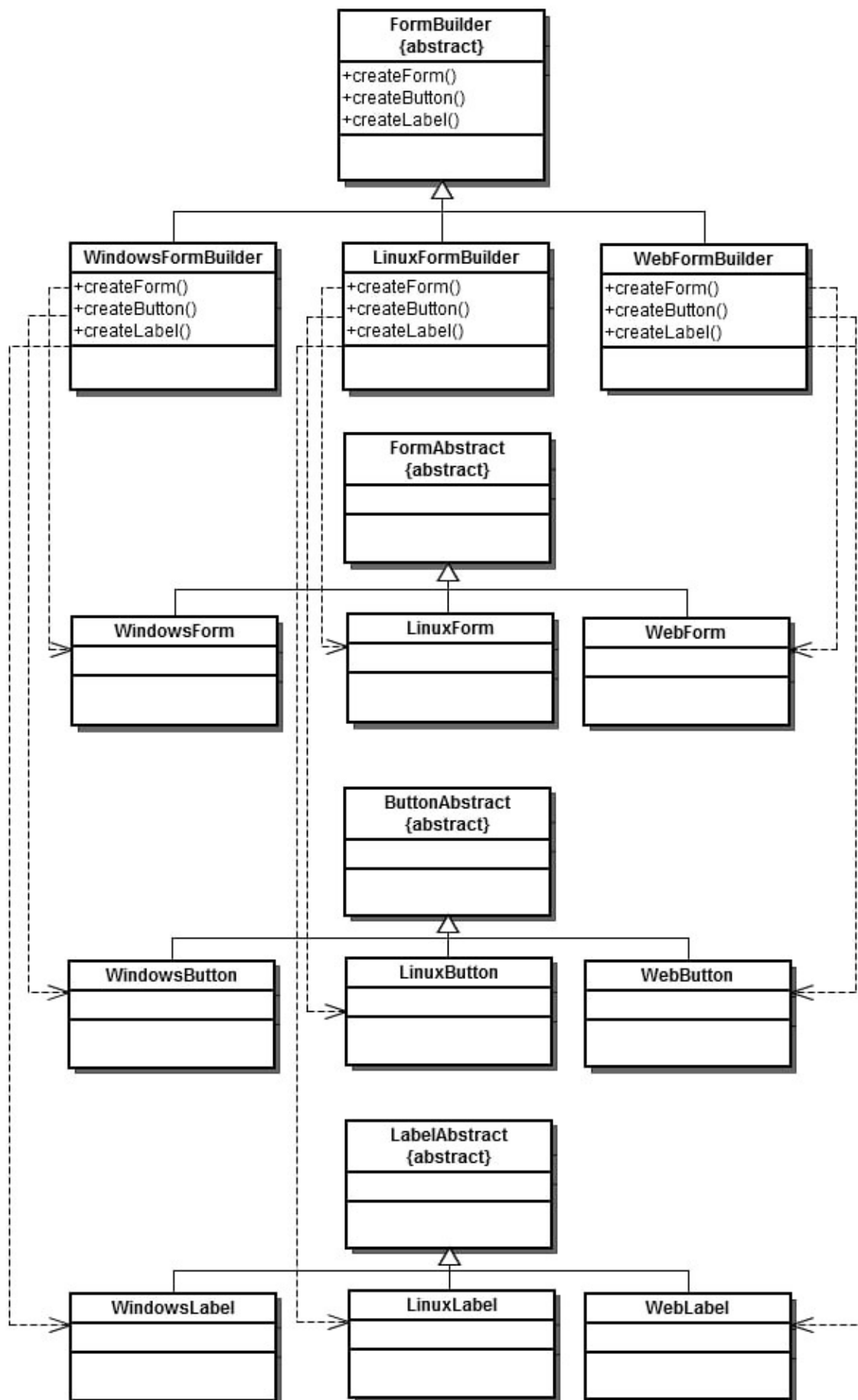


Рисунок 1.1 Схема паттерну “Фабричний метод”

Шаблонний метод є одним із поведінкових паттернів проектування, який можна використовувати в мові програмування С#. Він дозволяє визначити скелет алгоритму в базовому класі, переклавши деякі конкретні кроки до підкласів. Цей паттерн розв'язує проблему, коли потрібно мати спільний алгоритм, але з однією або декількома змінними частинами, які можуть відрізнятися у різних реалізаціях.

Структура шаблонного методу складається з двох ключових елементів: абстрактного базового класу і конкретних підкласів. Базовий клас містить основний алгоритм у вигляді методу зі специфічною послідовністю викликів. Деякі з цих методів можуть бути абстрактними, що дозволяє підкласам надавати свою реалізацію.

Паттерн шаблонного методу пропонує таке рішення: базовий клас має фіксований скелет алгоритму, але деякі кроки цього алгоритму відкладаються на підкласи. Конкретні підкласи наслідують базовий клас і надають свою реалізацію для цих відкладених кроків. В результаті, загальний алгоритм виконується у базовому класі, а деякі специфічні дії залежать від реалізацій у підкласах.

Шаблонний метод дозволяє зберігати загальний алгоритм у базовому класі, а одночасно дозволяє підкласам змінювати частини цього алгоритму за потреби. В результаті досягається гнучкість і легкість розширення системи без необхідності модифікації базового класу.

У С# можна застосовувати шаблонний метод для реалізації алгоритму обробки даних, де деякі конкретні кроки, такі як зчитування даних з різних джерел або обробка результатів, можуть відрізнятися в залежності від конкретного джерела або типу даних. Базовий клас містить загальну послідовність кроків алгоритму, а підкласи надають свою реалізацію для цих відкладених кроків, враховуючи специфіку своїх джерел даних (рисунок 1.2).

					РП 06.12 001.00 ДП ПЗ	Арк.
						11
Змін.	Арк.	№ докум.	Підпис	Дата		

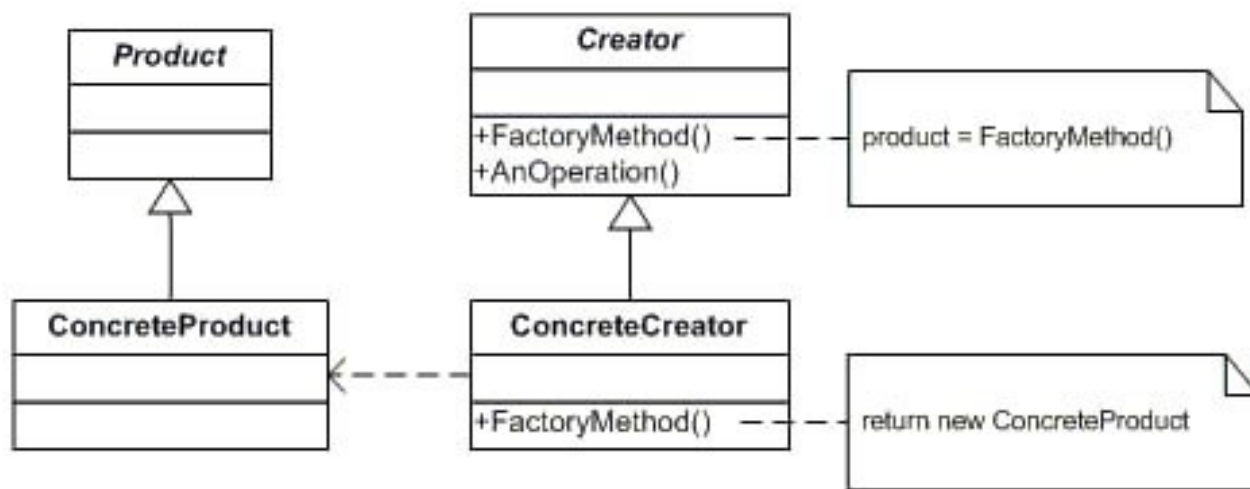


Рисунок 1.2 Схема паттерну Шаблоний метод

Паттерн Будівельник є паттерном проектування, який вирішує проблему складного процесу створення об'єкта і дозволяє розділити його конструкцію від представлення. Він надає можливість створювати складні об'єкти крок за кроком, незалежно від їхнього фінального вигляду.

Основна проблема, яку вирішує паттерн Будівельник, полягає в тому, що деякі об'єкти можуть мати складну конструкцію, включаючи багато кроків для їх створення. Це може бути особливо важко, якщо необхідно створити різні варіації об'єктів з однієї основної конструкції.

Паттерн Будівельник пропонує розділення процесу створення об'єкта на кілька окремих кроків (методів). Кожен крок відповідає за певну частину конструкції об'єкта. Ці кроки виконуються послідовно, але необов'язково в тому ж порядку для всіх об'єктів. В результаті ми отримуємо гнучкість в будівництві об'єктів і можемо легко створювати різні варіації за допомогою одного і того ж будівельника.

Структура паттерна Будівельник включає наступні основні компоненти:

- Будівельник (Builder): Визначає загальний інтерфейс для будівельників, які відповідають за створення різних частин об'єктів.
- Конкретний будівельник (ConcreteBuilder): Реалізує інтерфейс будівельника і забезпечує конкретну реалізацію кроків будівництва для створення об'єкта.
- Директор (Director): Керує процесом будівництва, використовуючи

будівельника.

– Об'єкт (Product): Представляє фінальний об'єкт, який будується. Може мати складну структуру, що складається з різних частин.

Застосування паттерна Будівельник дозволяє покращити модульність і розширюваність коду, оскільки кожен будівельник може мати власну реалізацію кроків будівництва. Цей підхід також полегшує створення нових варіацій об'єктів, дозволяючи змінювати порядок або додавати нові кроки будівництва без зміни коду клієнта.

В мові С# паттерн Будівельник можна реалізувати шляхом створення відповідних класів будівельників, конкретних будівельників і директора, які взаємодіють для створення об'єкта з різними варіаціями (рисунок 1.3).

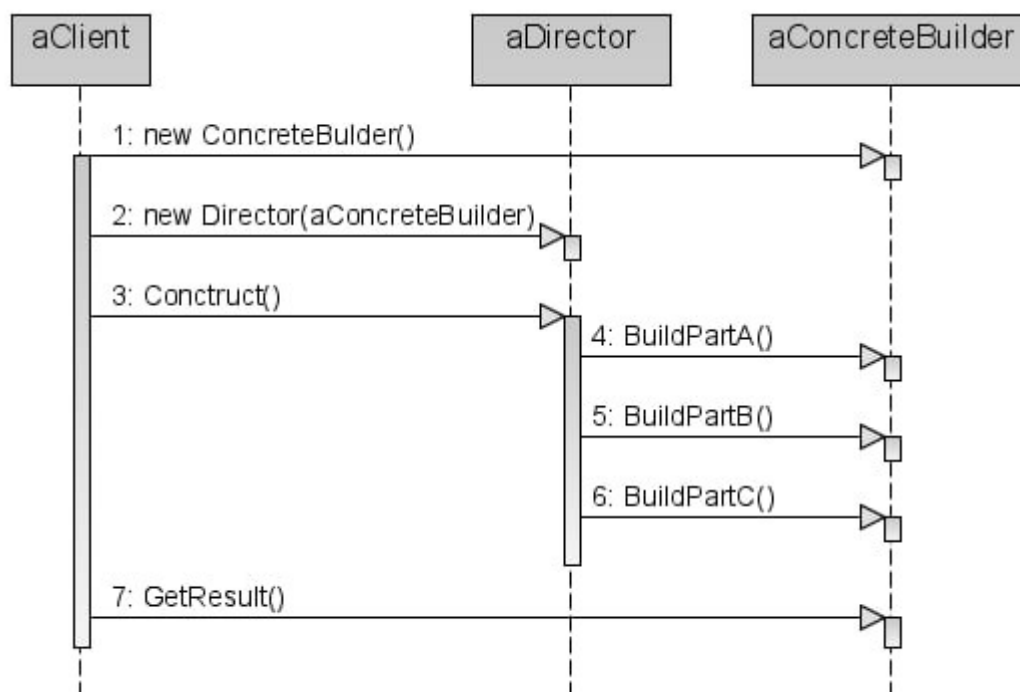


Рисунок 1.3 Схема паттерну Будівельник

Паттерн Спостерігач є одним з поведінкових паттернів проектування, який використовується для забезпечення взаємодії між об'єктами. У цьому паттерні деякі об'єкти, які називаються спостерігачами, спостерігають і реагують на зміни стану іншого об'єкта, який називається суб'єктом.

Проблема, яку вирішує паттерн Спостерігач, полягає в потребі у встановленні зв'язку між об'єктами таким чином, щоб зміна стану одного об'єкта автоматично спричиняла зміну стану інших об'єктів, які на нього спостерігають.

При цьому паттерн забезпечує слабку залежність між спостерігачами і суб'єктом, щоб уникнути прямих залежностей між ними.

Рішення, запропоноване паттерном Спостерігач, полягає у тому, що спостерігачі реєструються в суб'єкті і отримують повідомлення про зміни його стану. При зміні стану суб'єкта, він повідомляє всіх своїх зареєстрованих спостерігачів про зміну, і вони можуть відповідно реагувати на неї.

Структура паттерна Спостерігач має такі основні елементи:

- Суб'єкт (Subject): Це об'єкт, стан якого може змінюватися, і який публікує повідомлення про зміну свого стану спостерігачам.

- Спостерігач (Observer): Це інтерфейс або абстрактний клас, який описує методи, які використовуються спостерігачами для отримання повідомлень про зміну стану суб'єкта.

- Конкретний спостерігач (Concrete Observer): Це реалізація інтерфейсу або підклас абстрактного класу спостерігача, який виконує конкретні дії при отриманні повідомлень про зміну стану суб'єкта.

Реєстрація та видалення спостерігачів: Суб'єкт надає методи для реєстрації і видалення спостерігачів, щоб вони могли підписатися на або відписатися від отримання повідомлень (рисунок 1.4).

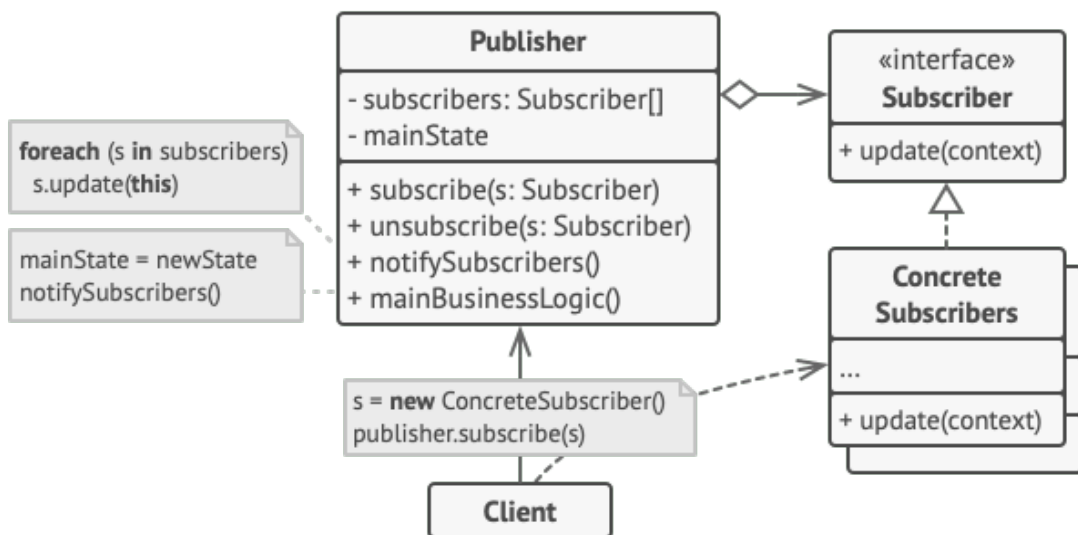


Рисунок 1.4 Схема паттерну Спостерігач

Паттерн Декоратор є одним з структурних паттернів проектування, який використовується для динамічного додавання нової поведінки до існуючих

об'єктів. Він дозволяє розширювати функціональність об'єктів, додаючи до них додаткові властивості і методи.

Проблема, яку вирішує паттерн Декоратор, полягає в необхідності додавати функціональність до об'єктів без зміни їх основної структури і без використання підкласів для кожної комбінації функцій. Він дозволяє динамічно надавати об'єктам нові можливості, що сприяє збереженню прозорості і змістовності коду.

Рішення, яке пропонує паттерн Декоратор, полягає в тому, що він використовує структуру композиції, де об'єкт-декоратор обгортає базовий об'єкт (компонент) і додає до нього додаткову функціональність. Кожен декоратор має такий самий інтерфейс, що дозволяє послідовно додавати нові декоратори та комбінувати їх, утворюючи ланцюжок декораторів.

Структура паттерна Декоратор має наступні основні елементи:

- Компонент (Component): Це базовий об'єкт або інтерфейс, який визначає загальний інтерфейс для базового об'єкта та декораторів.

- Конкретний компонент (Concrete Component): Це реалізація базового об'єкта або інтерфейсу компонента, до якого можна додавати додаткову функціональність.

- Декоратор (Decorator): Це абстрактний клас або інтерфейс, який визначає загальний інтерфейс для всіх декораторів. Він містить посилання на об'єкт-компонент та дозволяє додавати додаткову функціональність.

- Конкретний декоратор (Concrete Decorator): Це реалізація декоратора, який додає конкретну функціональність до базового об'єкта. Він обгортає базовий об'єкт і розширює його функціональність шляхом додавання власних методів або зміни поведінки.

Таким чином, паттерн Декоратор дозволяє динамічно додавати функціональність до об'єктів шляхом обгортання їх у декоратори. Це забезпечує гнучкість, розширюваність і змістовність коду, оскільки нові можливості можна додавати без зміни вже існуючих класів (рисунок 1.5).

					РП 06.12 001.00 ДП ПЗ	Арк.
						15
Змін.	Арк.	№ докум.	Підпис	Дата		

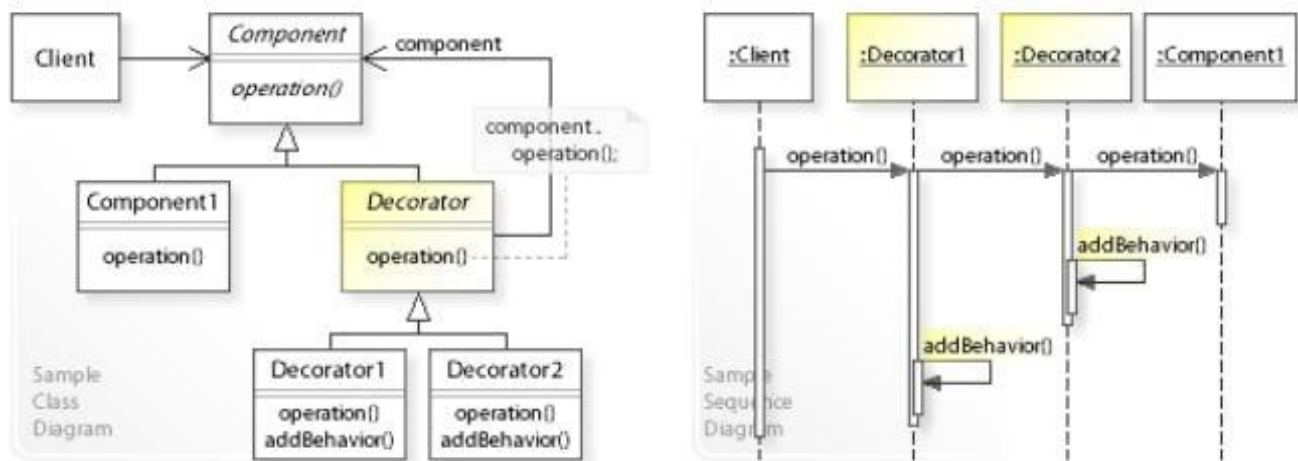


Рисунок 1.5 Схема паттерну Декоратор

Ітератор (Iterator) - це паттерн проектування, що використовується в об'єктно-орієнтованому програмуванні для отримання послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Він дозволяє перебирати елементи колекції без прив'язки до конкретної реалізації колекції.

Проблема, яку вирішує паттерн Ітератор, полягає в тому, що інкапсулюється спосіб доступу до елементів колекції та дозволяє ітерувати їх без знання про їхню конкретну структуру або реалізацію. Це дозволяє розділити код, який відповідає за ітерацію по колекції, від коду, який відповідає за саму колекцію.

Рішення, запропоноване паттерном Ітератор, полягає в тому, що він визначає загальний інтерфейс для ітераторів, що дозволяє послідовно отримувати доступ до елементів колекції. Класи, що реалізують цей інтерфейс, повинні забезпечувати методи для переходу до наступного елемента, перевірки наявності наступного елемента та отримання поточного елемента.

Структура паттерна Ітератор включає наступні компоненти:

- Інтерфейс Ітератор (Iterator): Визначає загальні методи для ітераторів, такі як MoveNext (перехід до наступного елемента), Current (отримання поточного елемента) та інші.
- Конкретний Ітератор (ConcreteIterator): Реалізує інтерфейс Ітератор та виконує конкретну ітерацію по колекції.

- Колекція (Collection): Визначає інтерфейс для отримання ітератора, який використовується для ітерації по колекції.
- Конкретна Колекція (ConcreteCollection): Реалізує інтерфейс Колекція та повертає конкретний Ітератор для ітерації по своїх елементах.
- Така структура дозволяє клієнту ітерувати елементи колекції без прив'язки до конкретної реалізації колекції або способу ітерації. Це забезпечує гнучкість та розділення відповідальностей між клієнтом і колекцією.

В мові програмування С# паттерн Ітератор можна реалізувати шляхом використання інтерфейсу `IEnumerator<T>`, який визначає методи для ітерації по колекції. Класи, які реалізують цей інтерфейс, забезпечують методи `MoveNext()` (перехід до наступного елемента) та `Current` (отримання поточного елемента) для ітерації по колекції типу `T` (рисунок 1.6).

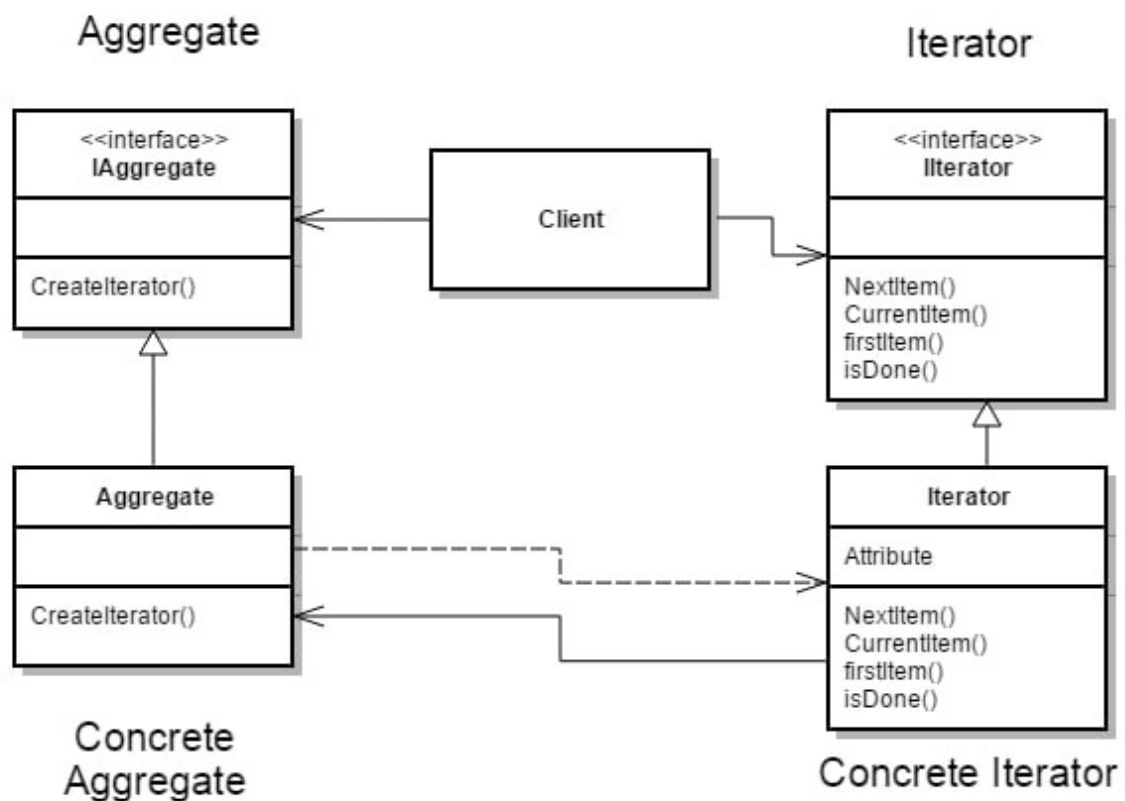


Рисунок 1.6 Схема патерну Ітератор

Шаблон одиночки (Singleton) є одним з популярних шаблонів проектування в мові програмування С#. Його основна ідея полягає в тому, щоб гарантувати існування лише одного екземпляра певного класу та забезпечити

глобальний доступ до цього екземпляра. Порівняно з іншими шаблонами проектування, такими як Фабричний метод, Шаблонний метод, Будівельник, Спостерігач, Декоратор та Ітератор, Одинак має кілька переваг:

- Унікальність екземпляра. Шаблон Singleton гарантує, що в системі буде існувати лише один екземпляр певного класу. Це особливо корисно, коли потрібно мати контроль над загальними ресурсами, наприклад, базою даних або файловою системою. Шляхом обмеження кількості екземплярів до одного, Singleton забезпечує впевненість, що всі об'єкти та модулі в системі використовують одну і ту саму інстанцію цього класу.

- Глобальний доступ до єдиного екземпляра. Це означає, що будь-який об'єкт в системі може отримати доступ до цього Singleton та використовувати його методи та властивості. Глобальний доступ забезпечує зручну можливість обміну даними та взаємодії з цим єдиним екземпляром усередині програми. Наприклад, можна використовувати Singleton для доступу до центрального об'єкта, який зберігає загальні дані чи конфігураційні параметри.

- Ліниве створення єдиного екземпляра. Це означає, що екземпляр класу Singleton буде створений лише в разі потреби, а не при старті програми. Це може бути корисно, особливо якщо створення екземпляра класу вимагає значних ресурсів або займає багато часу. Ліниве створення Singleton дозволяє економити ресурси системи та покращує продуктивність програми, оскільки екземпляр класу створюється лише тоді, коли це дійсно необхідно.

- Збереження стану об'єкта протягом тривалого часу. Завдяки тому, що існує лише один екземпляр класу Singleton, його стан залишається незмінним. Це дозволяє зберігати певну інформацію або стан, які можуть бути використані у різних частинах програми. Наприклад, Singleton може використовуватися для збереження історії подій або спільного стану, який потрібно використовувати в багатьох модулях програми.

Використання Singleton також має свої обмеження і потенційні проблеми. Наприклад, цей шаблон може призводити до нездатності одночасного використання багатьох екземплярів класу, що може бути необхідно у деяких

сценаріях. Крім того, використання Singleton може ускладнити тестування програми через його глобальний стан.

Узагалі, шаблон одиночки (Singleton) є корисним інструментом для створення єдиного екземпляра класу з глобальним доступом. Він гарантує унікальність екземпляра, забезпечує зручний глобальний доступ, можливість лінійного створення та збереження стану об'єкта. Проте, перед використанням Singleton потрібно уважно проаналізувати вимоги проекту і врахувати можливі обмеження та проблеми. Кожен шаблон проектування має свої особливості, і вибір найбільш підходящого залежить від конкретного контексту та вимог проекту (рисунки 1.7 і 1.8).

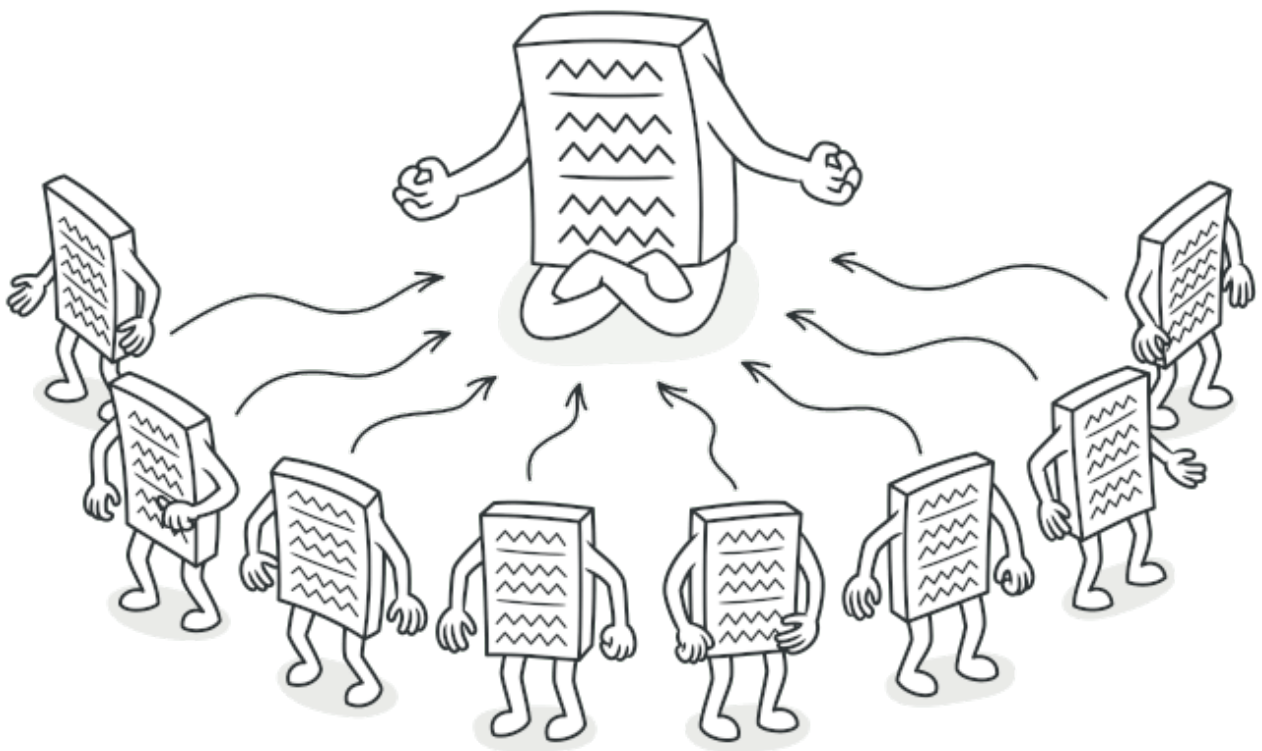


Рисунок 1.7 Ілюстрація паттерну Одинак

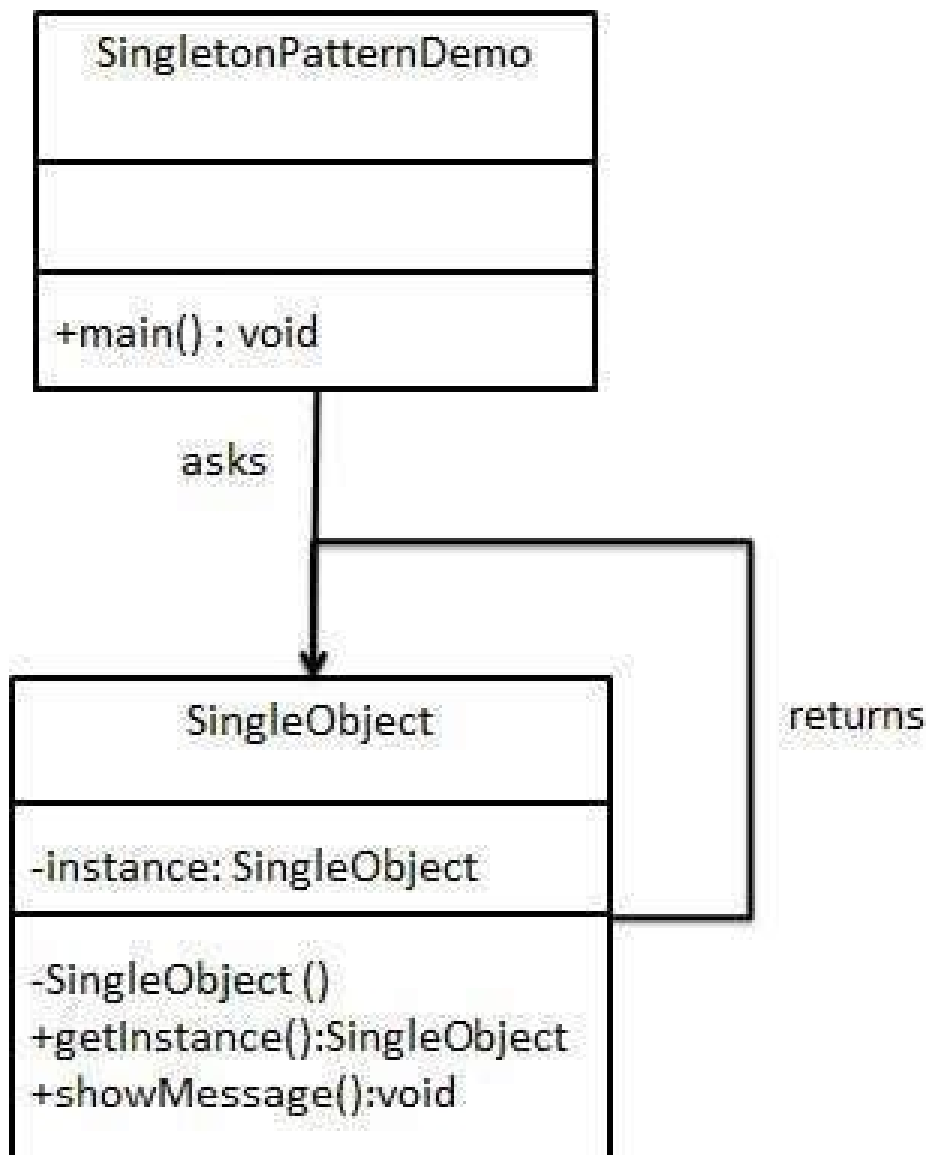


Рисунок 1.8 Схема патерну Одинак

1.3 Загальний технічний опис вирішення задачі дипломного проекту

Для розробки серверної частини та проектування баз даних для веб-сайту коледжу будуть використані такі технології:

ASP.NET 6 є останньою версією популярного веб-фреймворку, розробленого компанією Microsoft для створення потужних і масштабованих веб-додатків. Цей фреймворк базується на платформі .NET і надає розширені можливості для розробки веб-сайтів, веб-служб, API та інших сучасних веб-рішень.

ASP.NET 6 включає в себе багато нових функцій і покращень, які роблять його ще більш гнучким та ефективним для розробки веб-додатків. Одним з ключових змін у версії 6 є перехід до моделі відправлення даних (data-driven) в розробці. Зараз розробникам надається можливість використовувати будь-яку комбінацію фреймворків та бібліотек для створення веб-додатків. Наприклад, ви можете використовувати Razor Pages, Blazor, або MVC для створення різних частин додатку.

OpenIdDict - це бібліотека аутентифікації та авторизації, яка надає механізми для забезпечення безпечного доступу користувачів до веб-додатків та захисту їхніх конфіденційних даних. Вона базується на протоколах OpenID Connect і OAuth 2.0, які є стандартами для реалізації механізмів аутентифікації та авторизації.

Використання OpenIdDict дозволяє розробникам легко додавати функціональність аутентифікації та авторизації до своїх веб-додатків. Вона підтримує різні сценарії, такі як аутентифікація з використанням імен користувачів та паролів, використання зовнішніх постачальників ідентифікації (наприклад, Google або Facebook), а також використання токенів доступу для захисту API.

Entity Framework - це об'єктно-орієнтований фреймворк для роботи з базами даних у середовищі .NET. Він надає розробникам високорівневий доступ до даних та автоматично відображає об'єкти додатку на таблиці бази даних.

Одним із головних переваг Entity Framework є його здатність працювати з реляційними базами даних, такими як Microsoft SQL Server, Oracle, MySQL, а також з нереляційними базами даних, такими як MongoDB. Він дозволяє розробникам працювати з даними у вигляді об'єктів та здійснювати операції з ними без прямої роботи з SQL-запитами.

PostgreSQL є потужною та розширюваною об'єктно-реляційною системою управління базами даних (СУБД). Вона використовується для зберігання та керування великими обсягами даних у надійний та ефективний

					РП 06.12 001.00 ДП ПЗ	Арк.
						21
Змін.	Арк.	№ докум.	Підпис	Дата		

спосіб. PostgreSQL є однією з найпопулярніших відкритих СУБД та має широкий набір функціональних можливостей.

PostgreSQL підтримує реляційну модель даних, що дозволяє розробникам організовувати дані у вигляді таблиць зі зв'язками між ними. Вона підтримує мову SQL для взаємодії з базою даних і надає розширення, такі як збережені процедури, тригери та функції. PostgreSQL також підтримує геопросторові дані, що дозволяє зберігати та обробляти інформацію про місцезнаходження.

Що стосується Npgsql, це .NET-провайдер даних, який забезпечує зв'язок між програмами, написаними на платформі .NET, та базою даних PostgreSQL. Npgsql надає розробникам зручний спосіб взаємодії з PostgreSQL, дозволяючи виконувати SQL-запити, отримувати та зберігати дані у базі даних, а також використовувати транзакції та інші функціональні можливості PostgreSQL. Це дозволяє розробникам ефективно використовувати PostgreSQL у своїх програмах на платформі .NETASP.NET 6 буде використано як основна платформа для розробки серверної частини веб-сайту. За допомогою ASP.NET 6 будуть створені різні компоненти, що відповідатимуть за обробку запитів користувачів та взаємодію з базою даних.

Для забезпечення механізмів аутентифікації та авторизації користувачів буде використано OpenIDDict. OpenIDDict буде налаштовано як провайдера аутентифікації у серверному додатку, що дозволить користувачам зручно авторизуватись за допомогою своїх існуючих облікових записів з популярних платформ.

Entity Framework буде використано для роботи з базою даних. За допомогою Entity Framework буде налаштовано зв'язок з PostgreSQL базою даних за допомогою драйвера Npgsql. Це дозволить зберігати та керувати даними аутентифікації користувачів у базі даних серверного додатку.

Технологія мікросервісів буде використана для розділення функціональності серверної частини на окремі компоненти. Кожен мікросервіс буде відповідати за свою конкретну функціональність, що спростить розробку та підтримку системи.

					РП 06.12 001.00 ДП ПЗ	Арк.
						22
Змін.	Арк.	№ докум.	Підпис	Дата		

Для забезпечення безпеки та захисту будуть використані різні механізми, включаючи механізми шифрування, хешування та перевірки достовірності даних.

Використання цих технологій у поєднанні з ASP.NET 6 дозволить розробити потужну та безпечну серверну частину веб-сайту коледжу. Буде забезпечено зручну аутентифікацію користувачів та ефективну роботу з базою даних, забезпечуючи надійну та швидку взаємодію з користувачами та їхніми даними.

1.4 Огляд інструментів та програмних рішень

Для виконання дипломного проекту було обрано такі інструменти:

WebStorm - це інтегроване середовище розробки (Integrated Development Environment, IDE), призначене для веб-розробки. Воно розроблено компанією JetBrains і підтримується на різних операційних системах, включаючи Windows, macOS і Linux (рисунок 1.9).

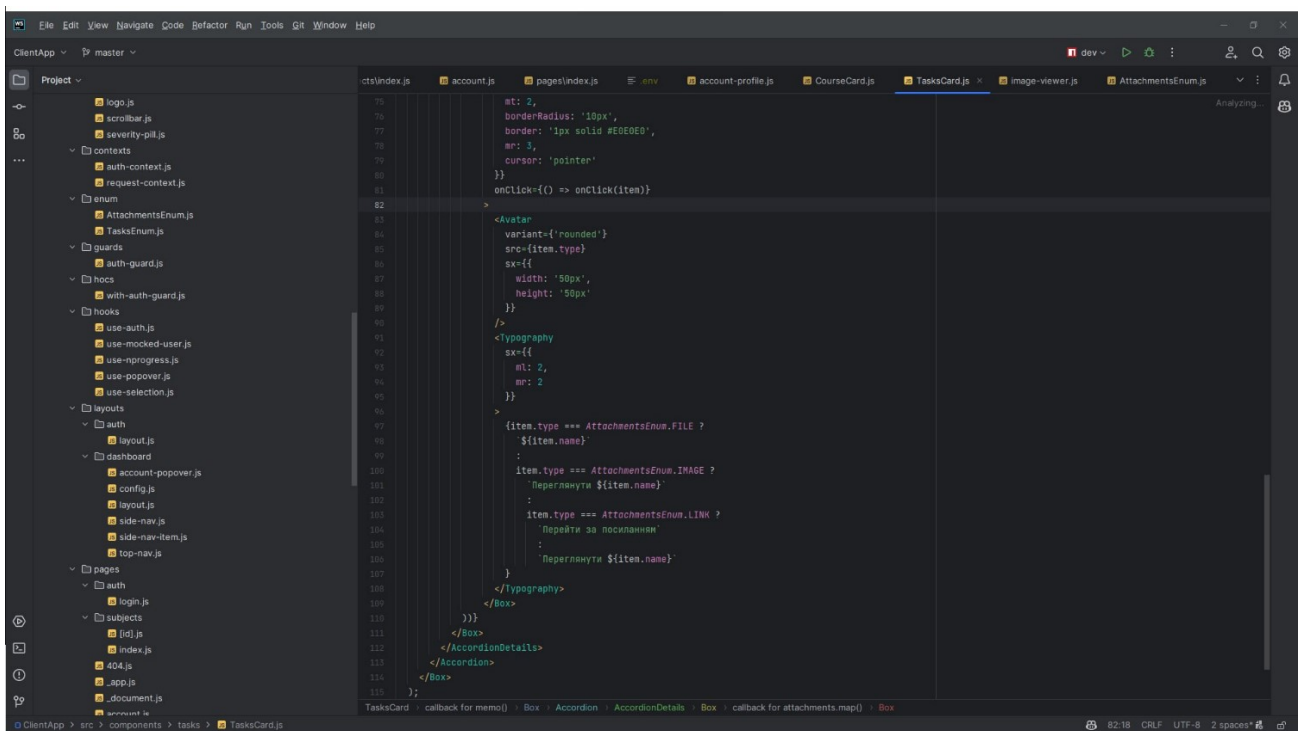


Рисунок 1.9 Інтерфейс WebStorm

Основні особливості WebStorm включають:

— Редактор коду: WebStorm надає потужний редактор коду з

					РП 06.12 001.00 ДП ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		23

підсвічуванням синтаксису, автодоповненням, рефакторингом, форматуванням коду та багатьма іншими корисними функціями. Він підтримує різні мови програмування, такі як JavaScript, HTML, CSS, TypeScript, React і багато інших.

– Налаштовувач (Debugger): WebStorm має вбудований налагоджувач, який дозволяє розробникам відстежувати та аналізувати виконання свого коду. Він надає можливість встановлювати точки зупину, переглядати значення змінних, крокувати по коду та інші налагоджувальні операції.

– Управління проектами: WebStorm дозволяє зручно керувати проектами. Користувач може створювати, видаляти, переміщувати та перейменовувати файли і папки, а також виконувати операції копіювання та переміщення. Він також інтегрований з системами контролю версій, такими як Git, що дозволяє вам працювати зі своїми репозиторіями безпосередньо в середовищі розробки.

– Засоби автоматизації: WebStorm підтримує інструменти автоматизації задач, такі як Gulp, Grunt і npm scripts. Користувач може налаштувати скрипти для виконання певних дій, таких як компіляція, збірка або запуск тестів, і запускати їх безпосередньо з WebStorm.

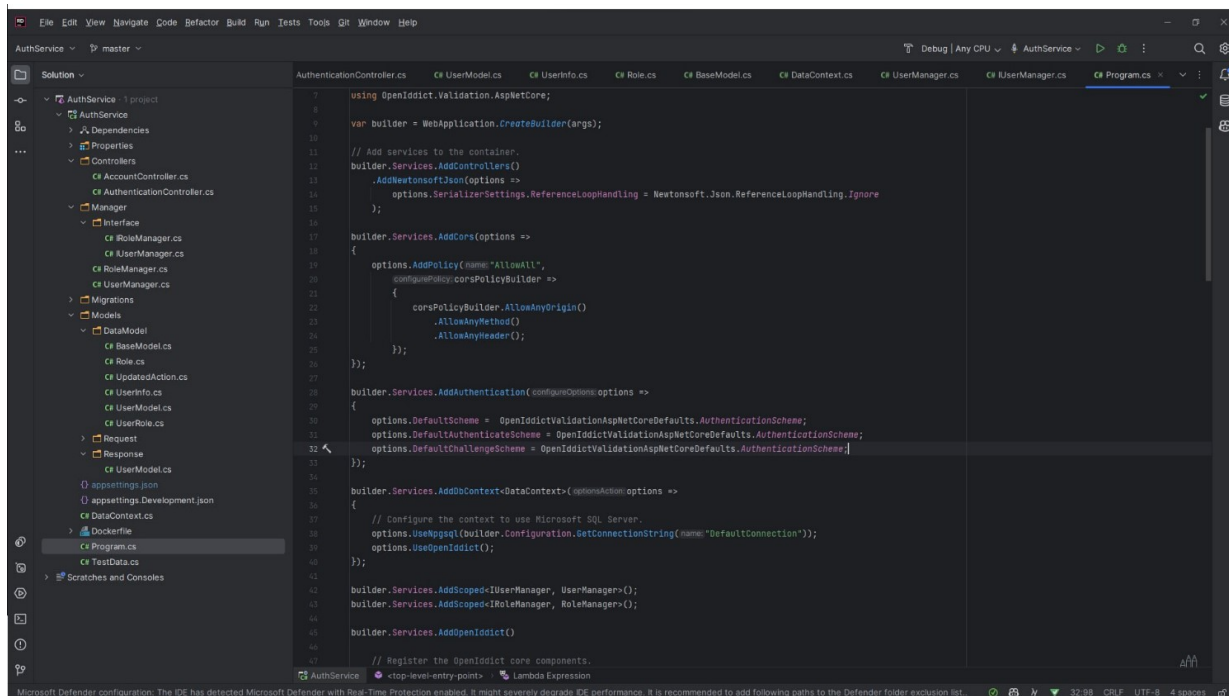
– Інтеграція з іншими інструментами: WebStorm може легко інтегруватися з іншими популярними інструментами розробки, такими як інспектори коду, системи тестування, лінери, сервери розробки та багато інших. Це дозволяє зручно працювати з різними інструментами в межах одного середовища.

– Підтримка фреймворків: WebStorm має вбудовану підтримку популярних фреймворків, таких як Angular, React, Vue.js, Node.js і багато інших. Це дозволяє розробникам легко створювати проекти з використанням цих фреймворків і отримувати підказки та інструменти, специфічні для кожного фреймворку.

– Це лише деякі основні особливості WebStorm. Інструмент має багато інших функцій і можливостей, які допомагають зробити процес розробки веб-додатків більш продуктивним і зручним.

					РП 06.12 001.00 ДП ПЗ	Арк.
						24
Змін.	Арк.	№ докум.	Підпис	Дата		

Rider - це інтегроване середовище розробки (Integrated Development Environment, IDE), розроблене компанією JetBrains, спеціалізоване на розробці програмного забезпечення на мові програмування C# і .NET. Rider є мультиплатформовим і підтримується на різних операційних системах, включаючи Windows, macOS і Linux (рисуюнок 1.10).



Рисуюнок 1.10 Інтерфейс Rider

Rider є редактором коду з багатьма потужними функціями, які полегшують процес розробки програмного забезпечення. Він підтримує мови програмування, такі як C#, JavaScript, TypeScript, HTML, CSS і SQL. Редактор має функції підсвічування синтаксису, автодоповнення, рефакторинг та багато інших, що допомагають писати код більш продуктивно.

Окрім редактора коду, Rider надає вбудований налагоджувач, який дозволяє відстежувати та аналізувати виконання коду. Він підтримує налагодження крок за кроком, встановлення точок зупину, перегляд значень змінних і багато інших функцій, що полегшують відлагодження програм.

Rider також забезпечує зручне управління проектами. Ви можете створювати, видаляти, переміщувати та перейменовувати файли і папки, керувати залежностями проекту і виконувати інші операції, пов'язані з управлінням проектом. Він також інтегрований з системами контролю версій,

					РП 06.12 001.00 ДП ПЗ	Арк.
						25
Змін.	Арк.	№ докум.	Підпис	Дата		

такими як Git, що дозволяє вам працювати зі своїми репозиторіями безпосередньо в середовищі розробки.

Rider має вбудовані інструменти аналізу коду, які допомагають виявляти потенційні проблеми в коді та надавати рекомендації щодо виправлення. Це включає інспектор коду, лінери, перевірку правил оформлення коду і багато інших інструментів, що сприяють підтримці якості коду.

Окрім того, Rider підтримує розробку на платформі .NET і фреймворку ASP.NET. Він надає можливості розробки, тестування і налагодження .NET-додатків, включаючи ASP.NET Core, ASP.NET MVC, Xamarin і багато інших.

Rider також може легко інтегруватися з іншими інструментами розробки, такими як системи збирання проєктів, тестувальні фреймворки, контроль якості коду і засоби CI/CD. Це дозволяє вам використовувати ваші улюблені інструменти розробки в поєднанні з Rider. Загалом, Rider надає потужні функції і інтеграцію з іншими інструментами, що полегшують процес розробки програмного забезпечення.

pgAdmin - це безкоштовне інструментарій для адміністрування та керування базами даних PostgreSQL. Він надає графічний інтерфейс користувача, який дозволяє легко створювати, керувати та моніторити бази даних PostgreSQL (рисунок 1.11).

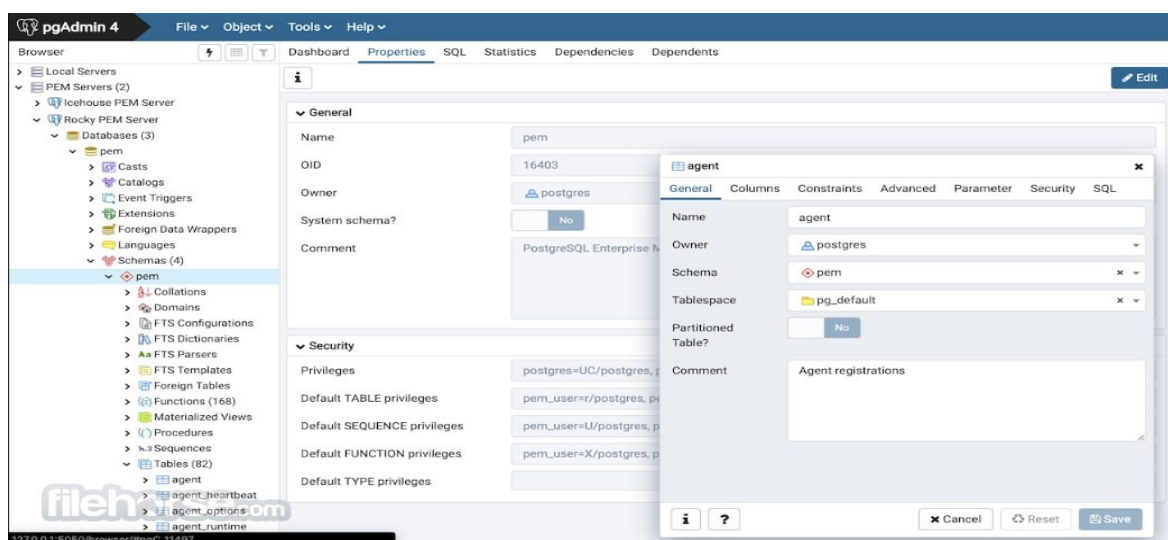


Рисунок 1.11 Інтерфейс pgAdmin

pgAdmin є потужним інструментом для керування та адміністрування баз даних PostgreSQL. Він надає інтуїтивний інтерфейс для створення, зміни та

					РП 06.12 001.00 ДП ПЗ	Арк.
						26
Змін.	Арк.	№ докум.	Підпис	Дата		

видалення баз даних, схем, таблиць, індексів, функцій та інших об'єктів PostgreSQL. Користувач може легко виконувати операції збереження, перейменування та зміни структури об'єктів бази даних.

Він також надає інструменти для виконання SQL-запитів до бази даних PostgreSQL. Це включає SQL-редактор, вікно запитів та консоль запитів. Користувач може виконувати запити, переглядати та редагувати результати запитів, використовувати автодоповнення та отримувати пояснення виконання запитів.

Окрім того, має функції моніторингу бази даних. Ви можете відстежувати активність запитів, використання ресурсів, статистику запитів та іншу інформацію про стан бази даних. Ви також можете переглядати історію запитів, аналізувати профілювання запитів та виявляти потенційні проблеми продуктивності.

Цей інструмент забезпечує можливість створення резервних копій баз даних PostgreSQL та їх відновлення з резервних копій. Ви можете налаштовувати регулярне резервне копіювання, вибирати об'єкти для включення до резервних копій та виконувати операції відновлення.

Також він дозволяє керувати ролями користувачів, групами та дозволами доступу до бази даних. Ви можете створювати та керувати ролями, встановлювати дозволи доступу до об'єктів бази даних, контролювати безпеку та обмеження.

Крім основних функцій, має й інші розширені можливості, такі як відладка процедур бази даних, керування розширеннями та використання візуальних засобів для роботи з географічними даними.

Переваги використання включають підтримку на різних операційних системах, інтуїтивний графічний інтерфейс, широкий набір інструментів для адміністрування та розробки, а також активну спільноту користувачів і підтримку.

pgAdmin є надійним і потужним інструментом, який допомагає вам ефективно керувати та адмініструвати бази даних PostgreSQL зручним способом.

					РП 06.12 001.00 ДП ПЗ	Арк.
						27
Змін.	Арк.	№ докум.	Підпис	Дата		

npm (Node Package Manager) є одним з найпопулярніших інструментів для управління пакетами в екосистемі JavaScript та Node.js. Він має кілька ключових переваг, які роблять його популярним серед розробників.

Найбільший репозиторій пакетів: npm має найбільшу кількість доступних пакетів для JavaScript і Node.js. Ви знайдете тисячі готових пакетів та модулів, які можна використовувати в своїх проектах. Це дозволяє прискорити розробку, використовуючи готові рішення, замість написання коду з нуля.

Легкість встановлення та використання: npm встановлюється разом з Node.js, що робить його легко доступним для використання. Встановлення пакетів здійснюється за допомогою простих команд командного рядка, що дозволяє швидко включати нові функціональності у ваш проект.

Керування залежностями та версіонування: npm надає потужні засоби для керування залежностями вашого проекту і версіонування пакетів. Ви можете легко встановлювати конкретні версії пакетів, вирішувати конфлікти залежностей, оновлювати пакети і т.д. Це допомагає забезпечити стабільність та сумісність залежностей вашого проекту.

Розширюваність та спільнота: npm має велику та активну спільноту розробників, яка надає підтримку, допомогу та сприяє внеску в розвиток пакетів та інструментів. Ви можете знайти багато готових пакетів, модулів та рішень, які можна використовувати в своїх проектах, а також знайти відповіді на свої запитання та проблеми у спільноті.

Набір функцій npm робить його потужним інструментом для управління пакетами в екосистемі JavaScript та Node.js. Використання npm полегшує розробку, сприяє використанню готових рішень та сприяє спільноті розробників.

1.5 Проектування баз даних

UML (Unified Modeling Language) - це стандартна мова моделювання, яка використовується для візуального представлення та документування різних аспектів системи, включаючи бази даних.

					РП 06.12 001.00 ДП ПЗ	Арк.
						28
Змін.	Арк.	№ докум.	Підпис	Дата		

UML-діаграми баз даних забезпечують зручний спосіб візуалізації складних структур та взаємозв'язків в базі даних. Вони дозволяють моделювати сутності, атрибути, зв'язки між таблицями, а також основні правила цілісності та обмеження бази даних. За допомогою UML-діаграм баз даних розробники можуть краще розуміти та комунікувати структуру бази даних з іншими учасниками проекту, що сприяє покращенню процесу розробки та підтримки системи (рисунк 1.12).

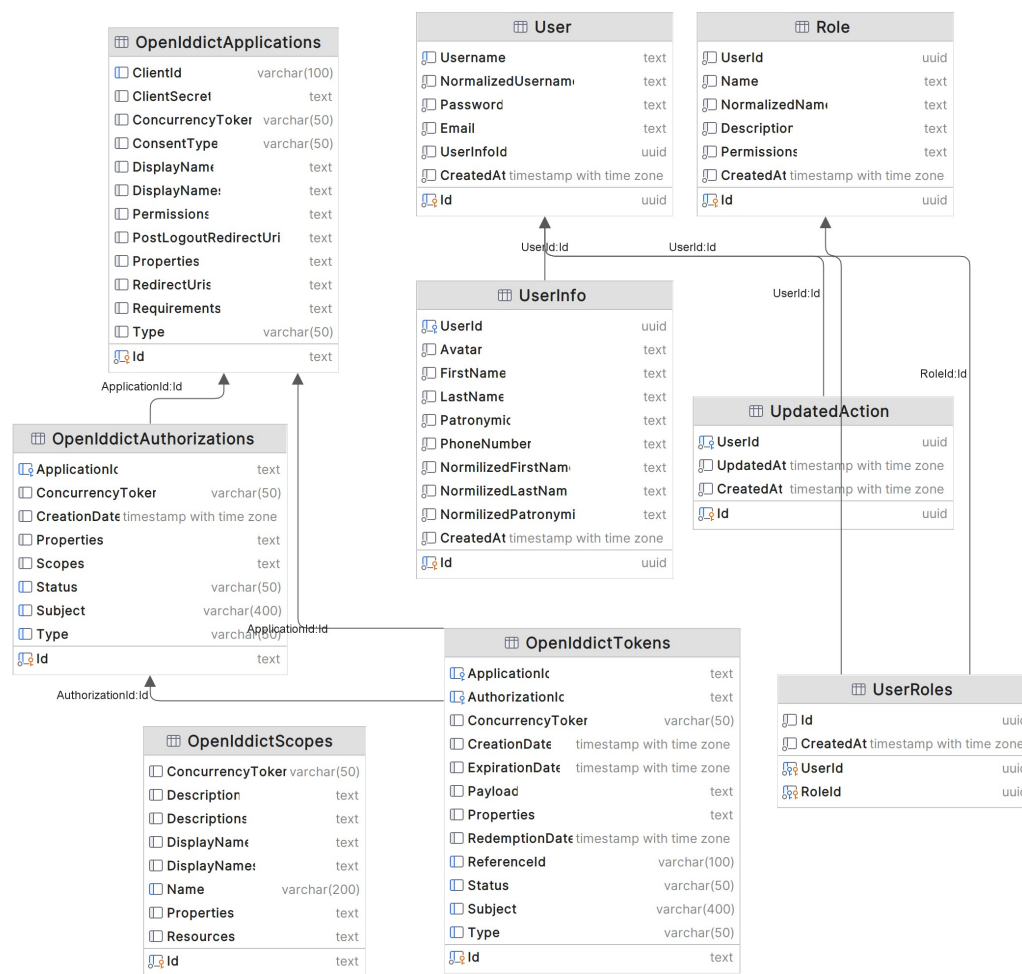


Рисунок 1.12 UML-діаграма бази даних для авторизації

"Базова модель" є класом, що використовується як основний шаблон для інших моделей даних у цьому додатку. Він містить деякі поля та функціональність, які можуть бути спільними для всіх моделей (рисунк 1.12).

Характеристики: клас містить конструктор `public BaseModel()`, який викликається при створенні нового об'єкта класу "Базова модель";

У конструкторі виконується ініціалізація деяких полів. Наприклад, поле "Id" отримує нове значення типу Guid, яке є унікальним ідентифікатором об'єкта. Поле "CreatedAt" отримує поточну дату та час в універсальному форматі.

Поле "Id" використовується як первинний ключ для таблиці бази даних, що відповідає цьому класу. Це означає, що кожен об'єкт цього класу буде мати унікальний ідентифікатор.

Клас містить властивість Id, яка забезпечує доступ до значення поля "Id".

Клас також має властивість CreatedAt, яка забезпечує доступ до значення поля "CreatedAt".

Основна мета класу "Базова модель" полягає в тому, щоб надати загальну структуру та функціональність для інших моделей даних у цьому додатку.



Рисунок 1.13 Схема BaseModel

Таблиця бази даних має такі стовпці:

UserId - це стовпець типу Guid, що відповідає ідентифікатору користувача. Його призначення - зберігати ідентифікатор користувача, який пов'язаний з цією роллю.

Name - це стовпець типу string, який зберігає ім'я ролі. Наприклад, "Адміністратор", "Модератор" тощо.

NormalizedName - це стовпець типу string, який зберігає нормалізоване ім'я ролі. Нормалізація використовується для стандартизації тексту, наприклад, у нижньому регістрі та без пробілів. Це допомагає уніфікувати роботу з ролями.

Description - це стовпець типу string, який містить опис ролі. Це може бути короткий опис того, що роль робить або які має повноваження.

Permissions - це стовпець типу string, який зберігає дозволи або права доступу, пов'язані з цією роллю. Це можуть бути різні дозволи на виконання певних операцій або доступ до певних ресурсів (рисунок 1.13).

					РП 06.12 001.00 ДП ПЗ	Арк.
						30
Змін.	Арк.	№ докум.	Підпис	Дата		

Role	
UserId	uuid
Name	text
NormalizedName	text
Description	text
Permissions	text
CreatedAt	timestamp with time zone
Id	uuid

Рисунок 1.14 Схема таблиці Role

Таблиця бази даних має рядки, кожен з яких представлятиме роль зі відповідними значеннями для кожного поля. За допомогою цієї таблиці ви зможете зберігати ролі користувачів та пов'язані з ними дані в базі даних.

Цей код представляє клас UpdatedAction, який, відображає таблицю бази даних. Давайте розглянемо кожне поле класу, щоб зрозуміти, як воно відповідає таблиці бази даних.

UserId - це стовпець типу Guid, який відповідає ідентифікатору користувача. Він використовується, для зберігання ідентифікатора користувача, пов'язаного з цією дією оновлення.

CreatedAt і Id – це поля з базової моделі.

UpdatedAt - це стовпець типу DateTime, який відображає дату та час оновлення. Це поле використовується для зберігання часової мітки, коли була зроблена дія оновлення (рисунок 1.15).

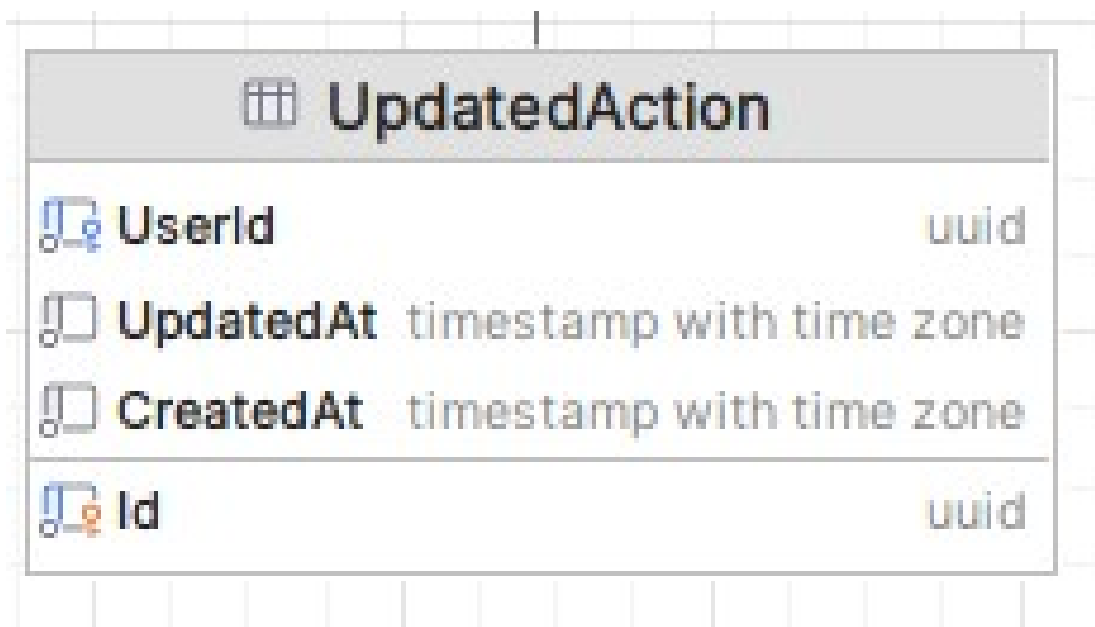


Рисунок 1.15 Схема моделі UpdatedAction

Тут є клас UserInfo, який, відображає таблицю бази даних. Давайте розглянемо кожне поле класу, щоб зрозуміти, як воно відповідає таблиці бази даних.

UserId - це стовпець типу Guid, який відповідає ідентифікатору користувача. Він використовується, для зберігання ідентифікатора користувача, пов'язаного з цими даними профілю користувача.

Avatar - це поле типу string, яке містить посилання або шлях до аватару користувача. Воно використовується для зберігання інформації про зображення, яке представляє профіль користувача.

FirstName - це поле типу string, яке містить ім'я користувача.

LastName - це поле типу string, яке містить прізвище користувача.

Patronymic - це поле типу string, яке містить по-батькові користувача.

PhoneNumber - це поле типу string, яке містить номер телефону користувача.

NormilizedFirstName - це поле типу string, яке містить нормалізоване значення імені користувача. Нормалізація може включати перетворення регістру чи видалення непотрібних символів. Це поле допомагає уніфікувати роботу з іменами користувачів.

NormalizedLastName - це поле типу string, яке містить нормалізоване значення прізвища користувача.

NormalizedPatronymic - це поле типу string, яке містить нормалізоване значення по-батькові користувача (рисунок 1.16).

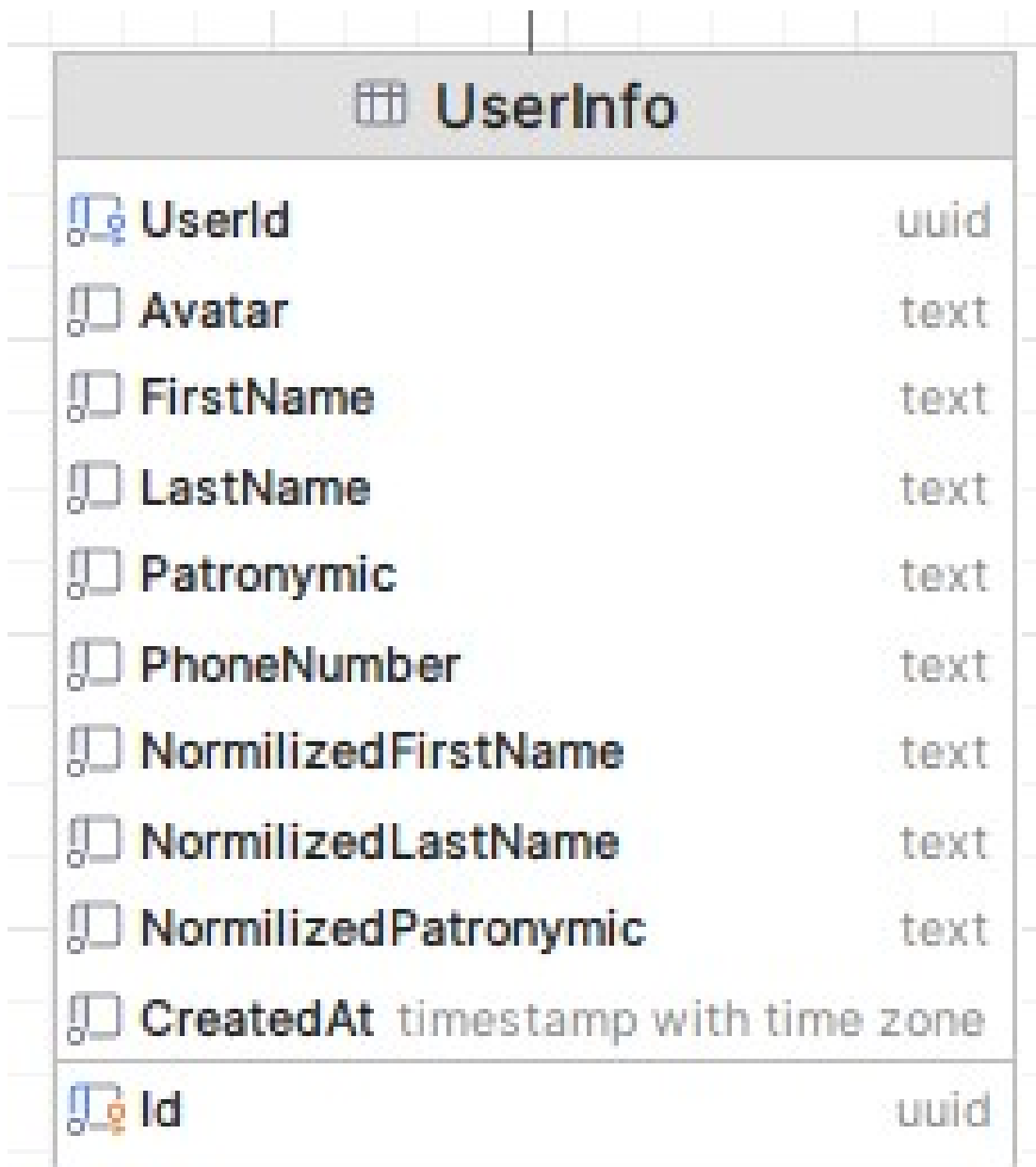


Рисунок 1.16 Схема моделі UserInfo

Клас "UserRole" є моделлю даних, яка відповідає таблиці в базі даних. Він успадковується від класу "BaseModel".

"UserId" - це стовпець типу "Guid", який використовується для зберігання ідентифікатора користувача, пов'язаного з цією роллю.

"RoleId" - це стовпець типу "Guid", який використовується для зберігання ідентифікатора ролі, пов'язаної з цією роллю користувача.

"User" - це стовпець, який використовується для встановлення зв'язку з таблицею "User". Він містить посилання на користувача, пов'язаного з цією роллю. Це дозволяє отримати додаткову інформацію про користувача, наприклад, ім'я, адресу електронної пошти тощо.

"Role" - це стовпець, який використовується для встановлення зв'язку з таблицею "Role". Він містить посилання на роль, пов'язану з цією роллю користувача. Це дозволяє отримати додаткову інформацію про роль, наприклад, назву ролі, дозволи тощо.

Отже, клас "UserRole" визначає структуру таблиці бази даних, яка містить дані про зв'язки між користувачами та їх ролями. Кожен рядок у цій таблиці містить ідентифікатори користувача та ролі, а також зв'язки з відповідними таблицями "User" та "Role", що надають додаткову інформацію про користувача та роль, пов'язану з цією роллю користувача (рисунок 1.17).

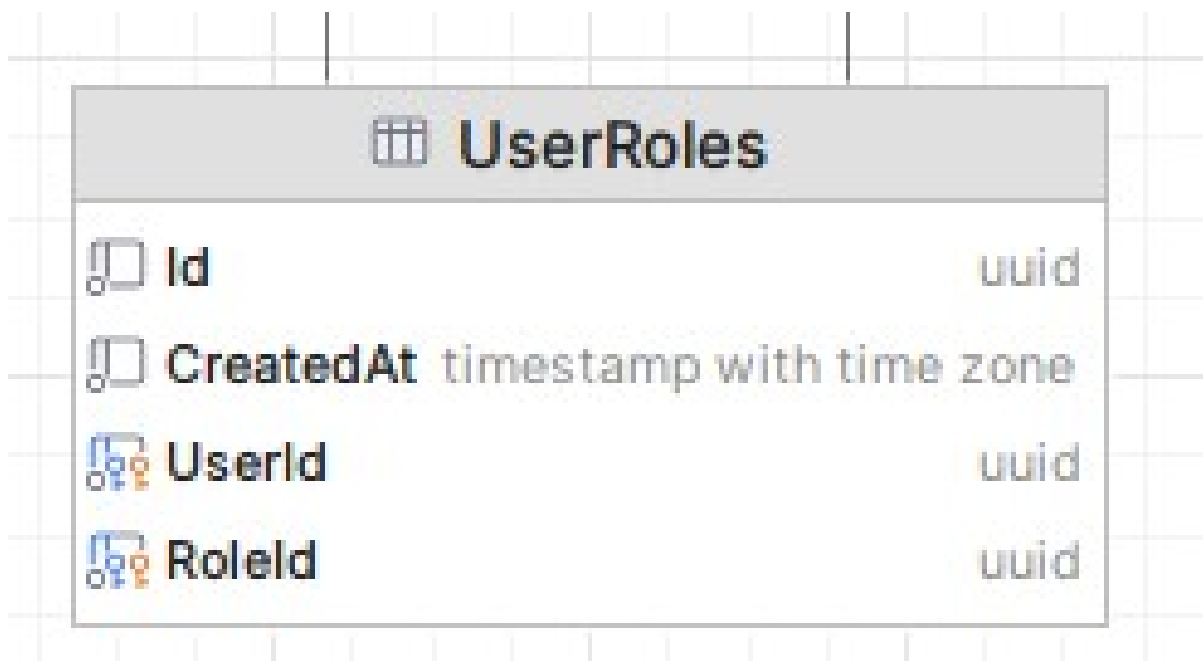


Рисунок 1.17 Схема таблиці зв'язків UserRoles

1.6 Розробка серверної частини

Принцип Singleton використовується для створення класу, що може мати лише один єдиний екземпляр. Цей принцип дозволяє забезпечити глобальний доступ до цього екземпляра та уникнути створення дублікатів.

У цьому випадку, є два інтерфейси - IUserManager і IRoleManager, які реалізовані в класах UserManager і RoleManager відповідно. Ці класи мають виконувати певні функції пов'язані з керуванням користувачами та ролями в системі.

Щоб використати ці реалізації в DI (Dependency Injection) контейнері, реєструю їх за допомогою методів builder.Services.AddScoped<IUserManager, UserManager>() та builder.Services.AddScoped<IRoleManager, RoleManager>(). Це означає, що в DI контейнері будуть створені екземпляри класів UserManager і RoleManager, і при кожному запиті на ці сервіси буде повертатися один і той самий екземпляр.

Зареєстровані реалізації інтерфейсів в DI контейнері дозволяють забезпечити зв'язування залежностей між класами в системі. Замість того, щоб створювати екземпляри цих класів безпосередньо в коді, вказано DI контейнеру, які конкретні реалізації використовувати при потребі.

Це дозволяє забезпечити велику гнучкість та розширюваність системи. Якщо потрібно буде змінити реалізацію IUserManager або IRoleManager, потрібно буде змінити лише реєстрацію в DI контейнері, а не весь код, що використовує ці сервіси.

Принцип Singleton дозволяє забезпечити тільки один екземпляр сервісу на протязі життєвого циклу програми. Це може бути корисно, наприклад, якщо є об'єкт, який зберігає загальний стан для всієї системи. Використання одного екземпляра дозволяє забезпечити консистентність даних і уникнути проблем з одночасним доступом до ресурсів.

Однак, використання Singleton може мати певні недоліки. Наприклад, він може ускладнити тестування, оскільки один єдиний екземпляр може зберігати стан, який може впливати на різні тести. Також, якщо Singleton має багато залежностей, це може призвести до збільшення складності коду та залежностей між класами.

У цьому випадку, використання принципу Singleton дозволяє мати один єдиний екземпляр IUserManager і IRoleManager в системі, який можна

					РП 06.12 001.00 ДП ПЗ	Арк.
						35
Змін.	Арк.	№ докум.	Підпис	Дата		

використовувати для керування користувачами та ролями. Це спрощує доступ до цих сервісів у будь-якій частині вашої програми, а також забезпечує їх консистентність та унікальність.

Застосування принципу Singleton дозволяє забезпечити централізоване управління ресурсами. Завдяки єдиному екземпляру IUserManager і IRoleManager, є можливість контролювати доступ до базових операцій, пов'язаних з користувачами і ролями, і забезпечити їх узгодженість та безпеку.

Один з сценаріїв застосування Singleton - це кешування даних. Наприклад, якщо є потреба у збереженні певних даних, що вимагають значних обчислень або запитів до бази даних, Singleton використовується для збереження цих даних та їх многоразового використання. Це покращує продуктивність системи та зменшує навантаження на базу даних.

Додатково, використання DI контейнера дозволяє забезпечити легкість розширення та заміни реалізацій. Можна легко змінити реалізацію IUserManager або IRoleManager, замінивши її в DI контейнері, без необхідності внесення змін у весь код, що використовує ці сервіси. Це сприяє зменшенню залежностей і полегшує тестування та розробку.

Принцип Singleton у поєднанні з DI контейнером дозволяє створити гнучку та легко розширювану архітектуру для керування користувачами і ролями в системі. Ви можете легко отримувати доступ до цих сервісів у будь-якій частині програми, контролювати стан та уникати непотрібного дублювання екземплярів. Це сприяє поліпшенню підтримки коду, його ясності та розумінню.

Використання принципу Singleton у вашому контексті також може сприяти забезпеченню консистентності та уніфікації даних. Завдяки єдиному екземпляру IUserManager і IRoleManager, ви маєте гарантію, що будь-яка операція, яку ви виконуєте з цими сервісами, буде впливати на одну і ту ж множину даних. Це дозволяє уникнути розбіжностей і неконсистентності, що можуть виникнути при роботі з декількома незалежними екземплярами сервісів.

Використання Singleton полегшує спільний доступ до ресурсів, таких як база даних або зовнішні сервіси. За допомогою єдиного екземпляру, можна

					РП 06.12 001.00 ДП ПЗ	Арк.
						36
Змін.	Арк.	№ докум.	Підпис	Дата		

ефективно керувати з'єднаннями, управляти кешем або встановлювати правила доступу до зовнішніх ресурсів. Це спрощує керування системними ресурсами і зменшує навантаження на зовнішні системи.

Принцип Singleton може допомогти уникнути зайвого використання пам'яті та ресурсів. Оскільки ви створюєте лише один екземпляр класу, ви економите ресурси, які були б витрачені на створення кожного нового екземпляра. Це особливо важливо, коли використовуються великі або ресурсоємні об'єкти.

Застосування Singleton також дозволяє зберігати стан і контекст між викликами. Оскільки ви маєте один єдиний екземпляр класу, ви можете зберігати стан, який залишається стійким протягом довшого періоду часу. Це корисно, наприклад, коли ви працюєте з об'єктом, який зберігає стан авторизації або іншу інформацію, що потребує постійного доступу.

Цей принцип може призвести до залежності від глобального стану і зростання складності коду. Тому слід обережно підходити до використання Singleton і ретельно аналізувати його вплив на архітектуру системи.

У цьому випадку використання принципу Singleton для IUserManager і IRoleManager дозволяє забезпечити глобальний доступ до функцій керування користувачами та ролями. Можна одноразово реєструвати ці сервіси в DI контейнері і легко отримувати доступ до них у будь-якій частині програми. Це спрощує управління залежностями і полегшує розробку системи з керуванням доступом.

Головна перевага використання DI контейнера полягає в тому, що він дозволяє розробникам використовувати інверсію керування для управління залежностями між компонентами програми. Реєстрація сервісів у DI контейнері дозволяє визначити, які об'єкти використовуються в яких частинах програми, і автоматично надавати їх при необхідності.

					РП 06.12 001.00 ДП ПЗ	Арк.
						37
Змін.	Арк.	№ докум.	Підпис	Дата		

1.7 Налаштування аутентифікації та авторизації

Для налаштування аутентифікації веб-сайту використовується OpenIddict - це бібліотека для реалізації протоколів аутентифікації та авторизації, таких як OAuth і OpenID Connect приклад життєвого циклу запиту на рисунку 1.18.

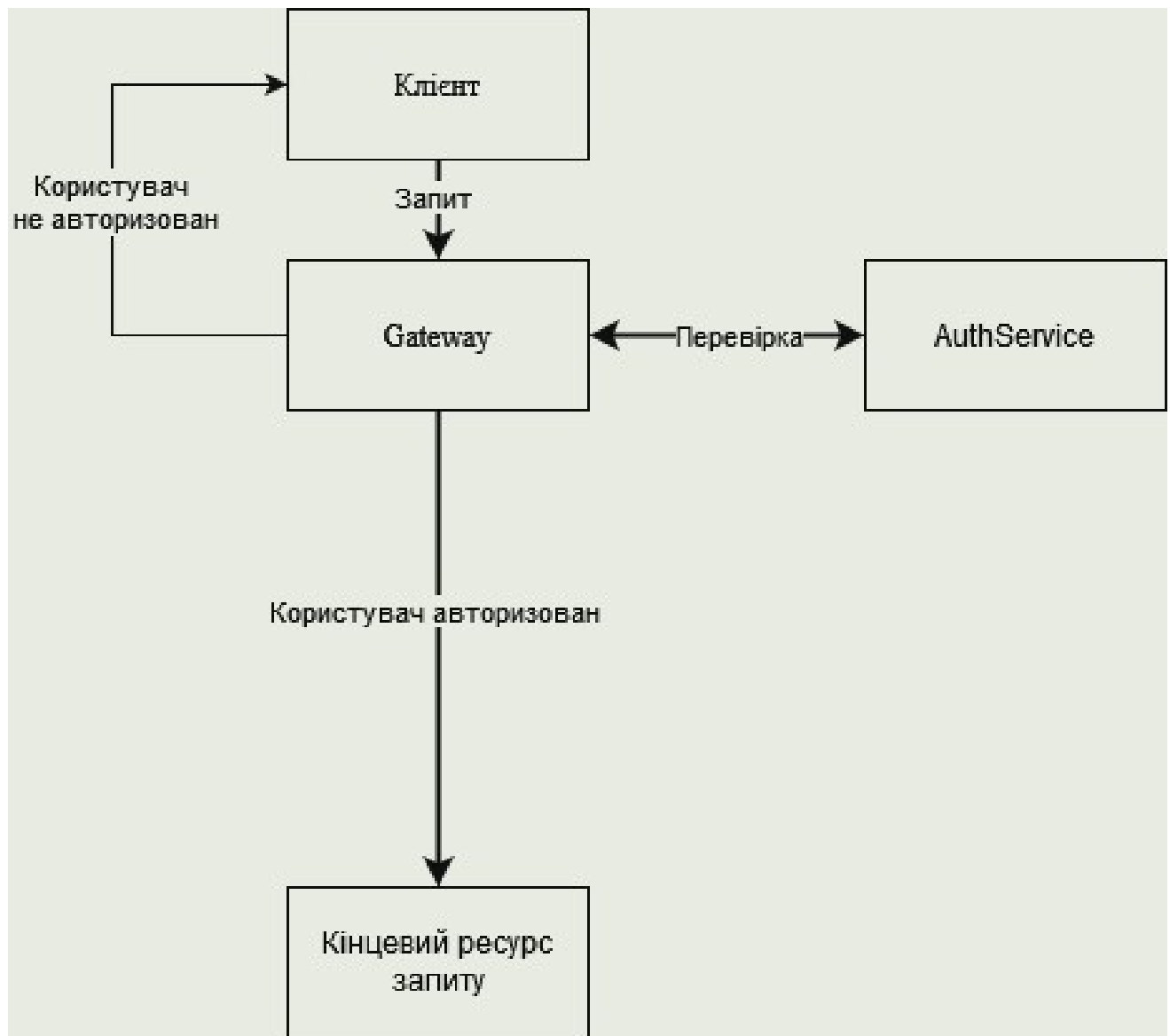


Рисунок 1.18 Схема перевірки авторизації

Використовується метод `AddAuthentication`, який додає сервіс аутентифікації до контейнера залежностей (dependency injection container) в ASP.NET Core. Цей метод налаштовує параметри аутентифікації за допомогою передачі конфігураційних параметрів у вигляді об'єкта `options`.

`DefaultScheme` встановлює схему аутентифікації за замовчуванням для всіх операцій аутентифікації, яка вказує на використання OpenIddict для перевірки та обробки запитів аутентифікації.

DefaultAuthenticateScheme встановлює схему аутентифікації за замовчуванням для процесу автентифікації користувачів.

DefaultChallengeScheme встановлює схему аутентифікації за замовчуванням для виклику автентифікаційного виклику, якщо користувач не має доступу до ресурсу.

Цей код використовується для інтеграції OpenIddict з механізмом аутентифікації ASP.NET Core, щоб забезпечити аутентифікацію користувачів у додатку з використанням стандартів аутентифікації і авторизації, які підтримуються OpenIddict.

Весь процес аутентифікації та авторизації буде оброблятися за допомогою OpenIddict. OpenIddict дозволяє створювати сервери аутентифікації і реалізовувати стандарти, такі як OAuth і OpenID Connect.

Коли користувач намагається автентифікуватися в додатку, ASP.NET Core буде використовувати OpenIddict для перевірки і обробки автентифікаційних запитів. OpenIddict може перевіряти ідентифікаційні дані користувача, надавати токени доступу та оновлювати їх, а також реалізовувати інші операції, пов'язані з аутентифікацією.

Цей код розміщено в методі ConfigureServices файлу Startup.cs ASP.NET Core додатку. Він вказує, що ASP.NET Core має використовувати OpenIddict як основний механізм аутентифікації та авторизації у додатку. Це дозволяє додатку взаємодіяти з іншими сервісами, які підтримують стандарти аутентифікації і авторизації, і забезпечує безпеку та захист інформації користувачів.

– Створено асинхронний метод з назвою "StartAsync", що виконує наступні дії:

– Створюється область ("scope") за допомогою інтерфейсу "_serviceProvider.CreateScope()". Область забезпечує керування інстанціями залежностей, які можуть бути використані в межах обмеженого контексту.

– Отримується інстанція "DataContext" залежності з області. "DataContext" використовується для доступу до бази даних. В цьому контексті викликається асинхронний метод "EnsureCreatedAsync", який створює базу

					РП 06.12 001.00 ДП ПЗ	Арк.
						39
Змін.	Арк.	№ докум.	Підпис	Дата		

даних (якщо вона ще не існує) за допомогою токена скасування "cancellationToken".

- Отримується інстанція "IOpenIddictApplicationManager" залежності з області. Цей менеджер використовується для керування додатками OpenIddict.

- Виконується перевірка наявності додатка з заданим "ClientId" ("egu"). Якщо додаток з таким "ClientId" не знайдено, виконується наступний блок коду.

Викликається метод "CreateAsync" менеджера додатків OpenIddict з параметром "OpenIddictApplicationDescriptor". В цьому методі створюється новий додаток з наступними властивостями:

- 1) "ClientId" - ідентифікатор клієнта, який унікально ідентифікує додаток.
- 2) "ClientSecret" - секретний ключ додатка, що використовується для аутентифікації і авторизації.

- 3) "DisplayName" - відображуване ім'я додатка.

- 4) "RedirectUris" - список URI перенаправлення, де OpenIddict надсилатиме авторизаційні відповіді.

- 5) "Permissions" - дозволи, які надаються додатку для доступу до різних ресурсів і функцій OpenIddict.

- 6) Цей код використовується для налаштування і ініціалізації додатків OpenIddict. OpenIddict - це фреймворк для управління аутентифікацією, авторизацією та видачею tokenів в середині ASP.NET Core додатків. Вказаний метод StartAsync може бути викликаний під час запуску додатка для налаштування необхідних компонентів OpenIddict перед початком обробки запитів користувачів.

У нас є такі налаштування:

- Налаштовується сервіс OpenIddict та реєструються його компоненти для використання в додатку. Компоненти включають ядро OpenIddict та серверні компоненти.

- Використовується Entity Framework Core для роботи з базою даних. Компоненти OpenIddict налаштовані для використання EF Core, зазначаючи контекст бази даних.

					РП 06.12 001.00 ДП ПЗ	Арк.
						40
Змін.	Арк.	№ докум.	Підпис	Дата		

– Виконуються налаштування потоків на серверній стороні для обробки різних типів авторизаційних потоків, наприклад, потік клієнтських облікових даних, потік облікових даних користувача та потік оновлення токенів.

– Конфігуруються тривалості життя токенів доступу та оновлення. Встановлюються URL-шляхи для точок доступу, таких як отримання токенів та отримання інформації про користувача.

– Виконується шифрування та підпис токенів для забезпечення безпеки. Реєструються області (дозволи) для роботи з різними ресурсами.

– Налаштовується політика файлів cookie для забезпечення безпеки та налаштування ASP.NET Core хосту. Деякі точки доступу можуть бути проксі-переадресовані.

– Виконується налаштування компонентів валідації для перевірки та захисту даних.

Додатково налаштовуються такі компоненти:

– Налаштовується компонент маршрутизації для визначення URL-шляхів, які будуть використовуватися для доступу до авторизаційних точок.

– Встановлюється механізм валідації та перевірки підпису токенів для забезпечення їхньої цілісності та недоступності для модифікації.

– Конфігурується система логування для реєстрації подій, пов'язаних з авторизацією та аутентифікацією, для забезпечення належного моніторингу та налагодження системи.

– Налаштовується інтеграція з існуючою системою аутентифікації користувачів, якщо така існує, для забезпечення єдиної точки входу та авторизації для всіх компонентів системи.

– Здійснюється налаштування обмежень доступу до ресурсів на основі ролей користувачів або інших атрибутів, що дозволяє забезпечити контроль над дозволами та доступом до функціональності системи.

– Встановлюється можливість відновлення паролів користувачів за допомогою електронної пошти або іншими методами підтвердження.

					РП 06.12 001.00 ДП ПЗ	Арк.
						41
Змін.	Арк.	№ докум.	Підпис	Дата		

– Конфігуруються налаштування сесій, такі як тривалість сесії та збереження даних сесії, для забезпечення безпеки та зручності використання системи.

– Налаштовуються компоненти кешування для підвищення продуктивності системи та зменшення навантаження на базу даних.

– Забезпечується можливість локалізації текстових повідомлень, пов'язаних з авторизацією та аутентифікацією, для підтримки різних мов та регіональних налаштувань.

– Налаштовуються заходи безпеки, такі як обмеження спроб авторизації, захист від атак перехоплення сесії та інші заходи для запобігання зловживанням та несанкціонованому доступу до системи.

Ці налаштування дозволяють створити безпечну та функціональну систему авторизації та аутентифікації для ASP.NET Core додатку з використанням протоколу OpenID Connect.

1.8 Розробка мікросервісів.

Розробка мікросервісів є складним процесом, що вимагає докладного планування, проектування та імплементації. Для цього сценарію ми розглянемо розробку таких мікросервісів:

AuthService використовує ASP.NET для створення сервера авторизації, який забезпечує аутентифікацію та авторизацію користувачів системи. Цей сервіс зберігає та перевіряє дані входу користувачів, надаючи доступ до захищених ресурсів системи тільки автентифікованим користувачам.

ScheduleService використовує Python і відповідає за управління розкладом, замінами та коледжними уроками. Цей сервіс забезпечує зберігання та оновлення інформації про розклад занять, дозволяє додавати та змінювати заміни, а також надає доступ до актуального розкладу для викладачів та студентів.

CRM використовує Python і функціонує як система, що зберігає інформацію про користувачів, їх описи та інші відомості. Цей сервіс дозволяє створювати та керувати профілями користувачів, зберігати їх особисті дані та

					РП 06.12 001.00 ДП ПЗ	Арк.
						42
Змін.	Арк.	№ докум.	Підпис	Дата		

іншу інформацію, яка використовується в системі.

TaskService також використовує ASP.NET і схожий на Google Classroom. Цей сервіс дозволяє вчителям призначати завдання учням, ставити їм оцінки та зберігати електронний журнал. Учні можуть переглядати призначені завдання, виконувати їх та переглядати свої оцінки.

Gateway є публічним API, який забезпечує комунікацію між клієнтською частиною та всіма вищезазначеними сервісами. Він використовує ASP.NET і дозволяє клієнтам здійснювати запити до системи, отримувати дані та виконувати різні операції.

Усі ці мікросервіси спілкуються між собою за допомогою Gateway, що забезпечує єдиний доступ до функціоналу системи для клієнтів. Вони працюють разом для забезпечення авторизації, управління розкладом, зберігання інформації про користувачів та надання функцій, пов'язаних з завданнями та оцінками.

					РП 06.12 001.00 ДП ПЗ	Арк.
						43
Змін.	Арк.	№ докум.	Підпис	Дата		

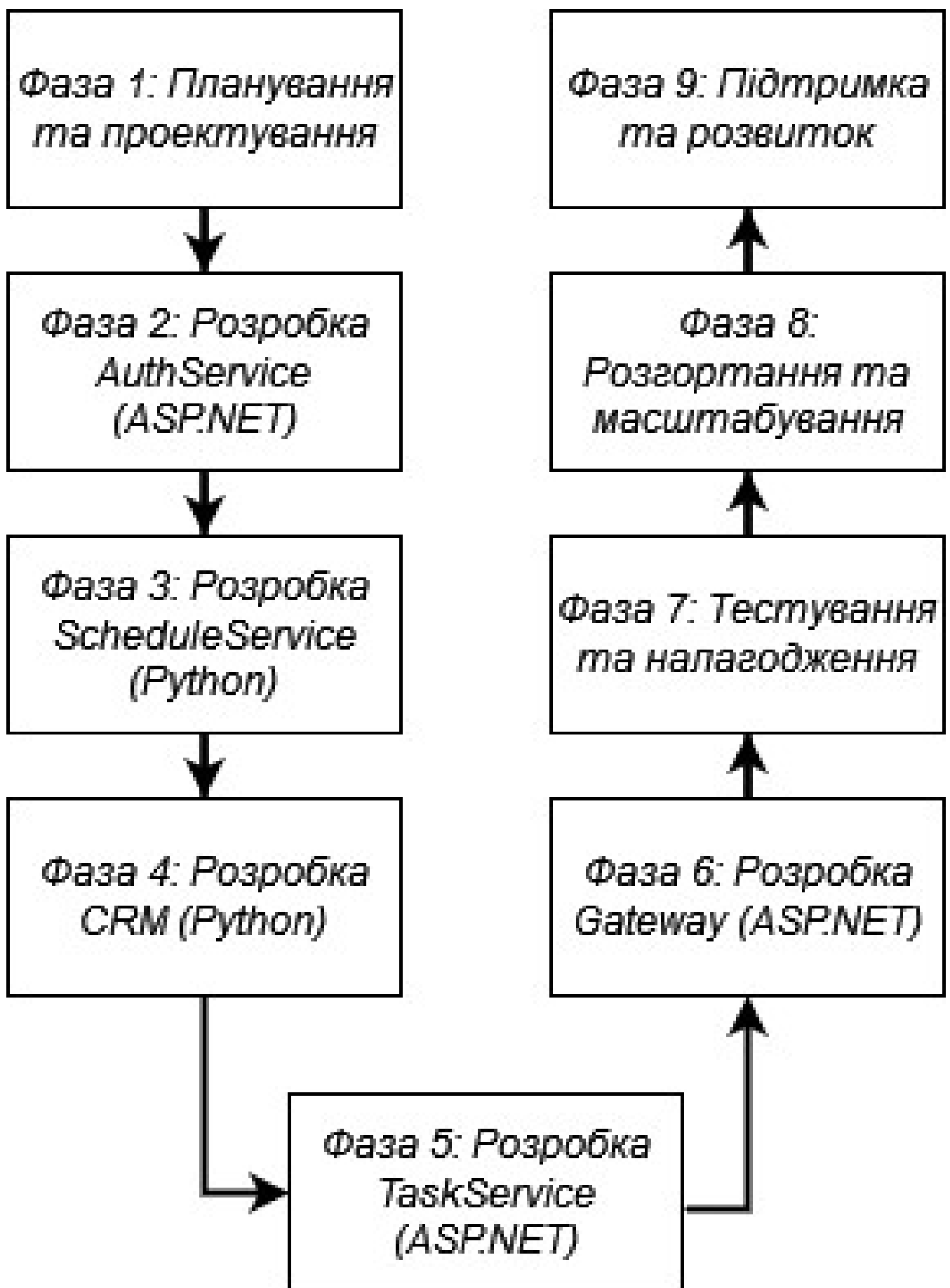


Рисунок 1.19 Планування фаз розробки

Нижче фази з малюнку 1.19 розписано більш детально.

Фаза 1: Планування та проектування

Першим кроком у розробці мікросервісів є планування та проектування архітектури системи. Визначається загальна структура мікросервісної архітектури, включаючи спосіб комунікації між сервісами. У нашому випадку, ми використовуємо REST систему комунікації, де кожен сервіс виступає як окремий вузол і надає API для взаємодії з іншими сервісами

Фаза 2: Розробка AuthService (ASP.NET)

Для розробки AuthService використовується ASP.NET. Створюється новий проект відповідно до типу сервісу та налаштовується середовище розробки. AuthService відповідає за авторизацію та аутентифікацію користувачів. Реалізується система реєстрації та входу користувачів, генерація та перевірка токенів доступу

Фаза 3: Розробка ScheduleService (Python)

Для розробки ScheduleService використовується Python та FastAPI. Створюється новий проект та налаштовується середовище розробки. ScheduleService відповідає за управління розкладом, замінами та коледжними уроками. Реалізується API для отримання та зміни розкладу, додавання замін та керування уроками

Фаза 4: Розробка CRM (Python)

Для розробки CRM використовується Python. Створюється новий проект та налаштовується середовище розробки. CRM відповідає за зберігання інформації про користувачів, їх описи та інші дані. Реалізується API для створення, оновлення та отримання даних користувачів

Фаза 5: Розробка TaskService (ASP.NET)

Для розробки TaskService використовується ASP.NET. Створюється новий проект та налаштовується середовище розробки. TaskService відповідає за завдання, які вчителі можуть призначати учням, оцінки та електронний журнал. Реалізується API для створення, оновлення та оцінювання завдань, а також керування електронним журналом

					РП 06.12 001.00 ДП ПЗ	Арк.
						45
Змін.	Арк.	№ докум.	Підпис	Дата		

Фаза 6: Розробка Gateway (ASP.NET)

Для розробки Gateway використовується ASP.NET. Створюється новий проект та налаштовується середовище розробки. Gateway виступає як публічний API, через який клієнтська частина комунікує з мікросервісами. Реалізується API для маршрутизації запитів від клієнта до відповідного сервісу

Фаза 7: Тестування та налагодження

Після розробки кожного мікросервісу проводиться тестування для перевірки його функціональності та взаємодії з іншими сервісами. Використовуються інструменти, такі як юніт-тести, інтеграційні тести та тести з використанням засобів автоматизації тестування. Після успішного тестування проводиться налагодження, попереднє випробування та оптимізація кожного мікросервісу

Фаза 8: Розгортання та масштабування

Після успішного тестування кожен мікросервіс розгортається на відповідному сервері або в хмарному середовищі. Забезпечується належна конфігурація серверів, безпека та моніторинг. Кожен мікросервіс може бути масштабований горизонтально або вертикально, в залежності від навантаження та потреб користувачів

Фаза 9: Підтримка та розвиток

Після розгортання мікросервісів забезпечується постійна підтримка та вирішення проблем. Процес підтримки включає моніторинг роботи сервісів, виявлення та виправлення помилок, регулярні оновлення та вдосконалення функціоналу. Розвиток системи може включати розширення функціоналу мікросервісів, оптимізацію роботи та впровадження нових технологій

Основні аспекти цієї фази включають моніторинг роботи сервісів, виявлення та виправлення помилок, регулярні оновлення та вдосконалення функціоналу.

Моніторинг роботи сервісів: Підтримка системи вимагає постійного моніторингу роботи мікросервісів. Це означає збір та аналіз журналів подій, метрик продуктивності та інших параметрів, що дозволяють оцінити стан

					РП 06.12 001.00 ДП ПЗ	Арк.
						46
Змін.	Арк.	№ докум.	Підпис	Дата		

системи. Моніторинг допомагає виявляти можливі проблеми та перевантаження, а також допомагає забезпечити високу доступність та продуктивність системи.

Виявлення та виправлення помилок: У фазі підтримки важливо швидко виявляти та усувати помилки, що виникають у мікросервісах. Це може включати аналіз журналів, відстеження стеку викликів та використання інструментів моніторингу помилок. Раннє виявлення та оперативне виправлення помилок допомагають забезпечити стабільну та безперебійну роботу системи.

Регулярні оновлення та вдосконалення функціоналу: Для підтримки системи необхідно регулярно оновлювати та вдосконалювати функціонал. Це може включати випуск нових версій програмного забезпечення, впровадження нових функцій та вдосконалень на основі отриманих від користувачів зворотних зв'язків. Такі оновлення допомагають покращити продуктивність, безпеку та зручність використання системи.

Загалом, підтримка системи включає моніторинг, виявлення та виправлення помилок, а також регулярні оновлення та вдосконалення функціоналу. Ці аспекти грають важливу роль у забезпеченні стабільної та ефективної роботи системи.

					РП 06.12 001.00 ДП ПЗ	Арк.
						47
Змін.	Арк.	№ докум.	Підпис	Дата		

2 ЕКОНОМІЧНА ЧАСТИНА

2.1 Резюме

В умовах розвитку сучасного інформаційного суспільства та економічних відносин, які в ньому встановилися, дуже важливою є роль інформаційних ресурсів, до яких відносять веб-сайти

Вдале використання інформаційного забезпечення та інформаційних ресурсів може значно підвищити ефективність функціонування підприємства.

Основна мета створення веб-сайту для коледжу – це забезпечення усіх зацікавлених людей ним зручним користувацьким досвідом

При оцінці ефективності створюваного сайту було виявлено, що сайт має лише функціональну та соціальну ефективність. Відсутність економічної ефективності зумовлена тим, що сайт не має прибутку і несе виключно функціональний та соціальний характер. Людина, що займалася замінами фактично тепер займає посаду адміністратора і має ту ж зарплатню

2.2 Визначення трудомісткості розробки веб-додатку

Загальні витрати (B_z) на створення сайту складаються з декількох параметрів:

$$B_z = B_p + B_v + B_e \quad (2.1)$$

де B_p – витрати на розробку сайту;

B_v – витрати на впровадження сайту;

B_e – витрати на експлуатацію сайту;

Витрати на розробку сайту (B_p) є одноразовими та залежать від етапів розробки сайту

Для визначення витрат на розробку сайту (B_p) розраховуємо оплату праці виконавців, безпосередньо притягнених до її виконання. Для реалізації проекту Web-системи використовуються наступні спеціалісти: дизайнер, архітектор, фронтендер, бекендер, системний адміністратор

					РП 06.12 002.00 ДП ПЗ	Арк.
						48
Змін.	Арк.	№ док.ум.	Підпис	Дата		

Для визначення трудомісткості розробки сайту (B_p) складено план-графік по розробці web-сайту і тривалості виконання робіт. Розподіл робіт по етапах і видах виконавців наведено в таблиці 2.1.

Таблиця 2.1 План-графік по розробці Web-сайту

№	Назва етапу	Час виконання (годин)	Посада виконавця
1	Проектування	10	Архітектор
2	Створення дизайну	25	Дизайнер
3	Реалізація клієнтської частини	100	Фронтендер
4	Реалізація серверної частини	100	Бекендер
5	Розгортка веб-додатку	5	Сис. адмін
ВСЬОГО:		240	

Розрахунок трудомісткості здійснений в наступній послідовності:

1. Складений перелік всіх етапів і видів робіт, які необхідно виконати в ході даної розробки. Після узгодження з керівником проекту допущено виключення, доповнення, об'єднання окремих етапів і видів робіт

2. По кожному виду робіт визначений кваліфікаційний рівень виконавців. В разі виконання однієї роботи виконавцями різної кваліфікації, робота розподілена на ряд паралельних конкретних робіт для кожної категорії виконавця

Тривалість виконання робіт розраховується на основі вірогідних оцінок робіт, що задаються виконавцями

Розмір заробітної плати розраховано виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а також їх середньої заробітної плати за годину.

Витрати на заробітну плату приведені в таблиці 2.2.

В умовах відсутності нормативної бази тривалість виконання окремих робіт розраховуємо на основі вірогідних оцінок робіт, що задаються виконавцями

Таблиця 2.2 Витрати на заробітну плату

№	Персонал	Етапи розробки	Кількість робочих годин	Погодинна ставка, грн.	Заробітна плата, грн.
1	Архітектор	Проектування	10	41	410
2	Дизайнер	Створення дизайну	25	41	1025
3	Фронтендер	Реалізація клієнтської частини	100	41	4100
4	Бекендер	Реалізація серверної частини	100	41	4100
5	Системний адміністратор	Розгортка веб- додатку	5	41	205
ВСЬОГО:					$B_{зп} = 9840$

Розмір єдиного соціального внеску складає 22% від заробітної плати, розраховано за наступною формулою:

$$B_{есв} = B_{зп} \times 0,22 \quad (2.2)$$

$$B_{есв} = 9840 \times 0,22 = 2164.8 \text{ грн.}$$

Загальні витрати (B_p) на розробку веб-сайту розраховано як сума витрат на заробітну плату праці персоналу ($B_{зп}$) та єдиного соціального внеску ($B_{есв}$):

$$B_p = B_{зп} + B_{есв} \quad (2.3)$$

$$B_p = 9840 + 2164.8 = 12\,004.8 \text{ грн.}$$

Витрати на впровадження сайту (B_v) складаються з двох складових:

- витрати на реєстрацію доменного імені на 1 рік (B_{v1});
- витрати на реєстрацію в пошукових системах (B_{v2}),

$$B_v = B_{v1} + B_{v2} \quad (2.4)$$

$$B_v = 432 + 0 = 432 \text{ грн}$$

					РП 06.12 002.00 ДП ПЗ	Арк.
						50
Змін.	Арк.	№ докум.	Підпис	Дата		

Витрати на експлуатацію сайту (B_e) включають вартість робіт з підтримки сайту в робочому стані і вартість послуг по продовженню доменного імені на 1 рік

Роботи по підтримці сайту в робочому стані включають в себе:

1. Оновлення даних на сайті;
2. Створення нових розділів на сайті;
3. Видалення застарілої інформації з сайту;
4. Додавання потрібної інформації на сайт;
5. Налаштування параметрів сервера хостингу;
6. Моніторинг роботи сервера хостингу;
7. Забезпечення щомісячного захисту сайту;
8. Створення резервних копій сайту та ін.

Підтримка сайту в робочому стані буде здійснювати сама організація. Витрати підраховуються виходячи із заробітної плати співробітників організації, призначеного для виконання цього виду робіт. Для певних робіт з цього переліку буде використовуватися обслуговуючий персонал (адміністратор web-сайту). У таблиці 2.3 визначаються постійні витрати як сума витрат на впровадження та експлуатацію сайту протягом року

Таблиця 2.3 Постійні витрати

№	Стаття витрат	Вартість за рік, грн.
1	Оренда серверу	2026
2	Доменне ім'я	432
3	Адміністратор	$6700 \cdot 12 = 80\,400$
Всього:		$B_{\text{пост}} = 82\,858$

Загальні витрати (B_3) на розробку, впровадження та експлуатацію веб-сайту розраховуються за наступною формулою:

$$B_3 = B_p + B_v + B_e \quad (2.5)$$

$$B_3 = 9635 + 205 + 82\,858 = 92\,698 \text{ грн}$$

Функціональна ефективність виявляється у вдосконаленні технологічних процесів, пов'язаних із наданням інформації викладачам та студентам щодо змін у розкладі занять. Це означає, що шляхом використання певного сайту чи системи, коледж забезпечує швидку та точну інформацію про зміни у розкладі, що дозволяє всім зацікавленим сторонам оперативно реагувати та планувати свій час. Така оптимізація технологічних процесів сприяє зменшенню незручностей, пов'язаних зі змінами у розкладі, та підвищує загальну продуктивність навчального процесу.

Соціальна ефективність полягає в тому, що завдяки впровадженню такої системи або сайту коледж стає більш привабливим для потенційних абітурієнтів. Інформованість студентів та викладачів про заміни у розкладі занять позитивно впливає на загальний імідж навчального закладу. Потенційні студенти бачать, що коледж має ефективну систему комунікації та дбає про своїх студентів, що може спонукати їх обрати цей коледж для навчання. Таким чином, соціальна ефективність сприяє збільшенню привабливості навчального закладу для потенційних абітурієнтів та покращенню його репутації у громадськості.

Загалом, функціональна та соціальна ефективність оптимізації технологічних процесів для інформування викладачів та студентів про заміни у розкладі занять допомагають поліпшити якість навчання та створити сприятливі умови для всіх учасників навчального процесу.

					РП 06.12 002.00 ДП ПЗ	Арк.
						52
Змін.	Арк.	№ докум.	Підпис	Дата		

3 ОХОРОНА ПРАЦІ

Безпека праці є важливою сферою практичної діяльності, що має на меті створення безпечних та нешкідливих умов праці. Забезпечення безпеки та запобігання шкідливим умовам праці вимагає значних фінансових витрат і впровадження науково-дослідних розробок у галузі охорони праці.

Умови праці для програмістів повинні відповідати вимогам безпеки праці, враховуючи нормативно-правові акти, такі як "Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями" (НПАОП 0.00-7.15-18) та "Правила охорони праці під час експлуатації електронно-обчислювальних машин" (НПАОП 0.00-1.28-10), які були затверджені Державним комітетом України з промислової безпеки, охорони праці та гірничого нагляду 26 березня 2010 року наказом № 65.

Тому знання з питань безпеки праці у технічних спеціалістів, зокрема у програмістів, мають вирішальне значення.

3.1 Гігієнічні вимоги до виробничого середовища.

3.1.1 Вимоги до приміщення

– Відповідно до нормативно-правовим актами, площа одного робочого місця повинна бути не меншою за 6.0 квадратних метри, а оюєм 20,0 куб.м.

– Кліматичні умови: Приміщення повинно бути забезпечено системою опалення для збереження комфортної температури, кондиціонуванням повітря для контролю вологості і припливно-витяжною вентиляцією для забезпечення свіжого повітря.

– Поверхня підлоги: Підлога повинна бути рівною, неслизькою і мати антистатичні властивості, щоб запобігти статичному електрику, який може шкодити обладнанню.

Забезпечення вимог безпеки праці включає в себе не лише правильне організування приміщення, але також розгляд конструкції робочого місця та його оснащення для оптимальної робочої пози та комфорту працівника. З цією метою необхідно враховувати наступні вимоги:

					РП 06.12 003.00 ДП ПЗ	Арк.
						53
Змін.	Арк.	№ докум.	Підпис	Дата		

3.1.2 Вимоги до організації робочого місця користувача ПК

Організація робочого місця повинна враховувати не лише вимоги до освітлення, але й інші аспекти ергономіки, такі як правильне розташування обладнання, налаштування стільця та столу для забезпечення оптимальної робочої позиції користувача. Всі ці фактори сприяють збереженню здоров'я, зниженню втоми та підвищенню продуктивності під час роботи з комп'ютером.

– Робочі місця з візуальним екраном (ВДТ) слід розташовувати таким чином, щоб природне світло падало збоку, переважно зліва. Це допоможе уникнути надмірного навантаження на очі працівника.

– Відстань між робочими місцями: Необхідно дотримувати відстань не менше 1,2 метра між бічними поверхнями ВДТ. Також відстань від тильної поверхні одного ВДТ до екрана іншого ВДТ повинна становити 2,5 метра. Це дозволить запобігти зайвим перешкодам та забезпечити комфортну робочу зону.

– Конструкція робочого столу: Робочий стіл має відповідати сучасним вимогам ергономіки. Він повинен забезпечувати оптимальне розташування обладнання (дисплея, клавіатури, принтера) і документів на робочій поверхні. Крім того, робочий стіл повинен мати достатньо простору для ніг, регульовану висоту, кут і нахил сидіння та спинки, а також зручну форму переднього краю сидіння. Регулювання цих параметрів має бути легким, незалежним і надійним.

– Робочий стілець: Робочий стілець повинен бути підйомно-поворотним, регульованим за висотою, кутом і нахилом сидіння та спинки. Поверхня сидіння має бути плоскою, а передній край заокругленим для забезпечення комфорту працівника. Регулювання кожного параметра має бути легким і надійним. Крім того, шаг регулювання елементів стільця повинен бути в межах 15-20 мм для лінійних розмірів і 2-5 градусів для кутових розмірів.

3.2 Освітлення робочого місця

Згідно з ДСанПІН 3.3.2.007-98, у приміщеннях з робочими місцями, де використовуються ВДТ ЕОМ та ПЕОМ, рекомендується використовувати

					РП 06.12 003.00 ДП ПЗ	Арк.
						54
Змін.	Арк.	№ докум.	Підпис	Дата		

систему загального рівномірного освітлення. Це означає, що освітлення повинно бути розподілене рівномірно по всій робочій зоні.

У виробничих та адміністративно-громадських приміщеннях, де переважно проводиться робота з документами, може бути застосована система комбінованого освітлення. Це означає, що крім загального освітлення, встановлюються також світильники місцевого освітлення, які забезпечують додаткове освітлення в окремих зонах, де розміщені документи.

Рекомендована освітленість на поверхні робочого столу в зоні розміщення документів становить 300-500 лк (люкс). Якщо системою загального освітлення досягнути таких значень освітленості не можливо, тоді допускається використовувати місцеве освітлення. Однак, важливо пам'ятати, щоб світильники місцевого освітлення не створювали блисків на поверхні екрана, а освітленість екрана не перевищувала 300 лк.

Зважаючи на ці рекомендації, як джерела світла варто використовувати переважно люмінесцентні лампи типу ЛБ. Однак, у виробничих та адміністративно-громадських приміщеннях також можуть бути застосовані металогалогенні лампи потужністю 250 Вт для влаштування відбитого освітлення. Для світильників місцевого освітлення можуть використовуватись лампи розжарювання.

У разі використання штучного освітлення важливо враховувати, щоб пряме сонячне світло не падало прямо на робоче місце, оскільки це може створювати блиск на екрані і викликати незручності для користувача.

Крім того, лампи типу ЛД (світлодіодні лампи) рекомендується використовувати з розсіювачами та екрануючими сітками. Це допомагає забезпечити рівномірне розподілення світла та уникнути різких тіней чи блисків на робочій поверхні.

					РП 06.12 003.00 ДП ПЗ	Арк.
						55
Змін.	Арк.	№ докум.	Підпис	Дата		

3.3 Мікроклімат

Згідно з нормами ДСН 3.3.6.042.-99 виробничі приміщення з робочими місцями, де використовуються ВДТ, мають мати оптимальні параметри мікроклімату, такі як температура, відносна вологість і рух повітря (згідно з таб. 3.1).

Таблиця 3.1

Пора року	Категорія робіт	Температура повітря, град.С	Відносна вологість повітря	Швидкість руху повітря м/с
		оптимальна	оптимальна	оптимальна
Холодна	Легка-1 а	22-24	40-60	0.1
	Легка-1 б	21-23	40-60	0.1
Тепла	Легка-1 а	23-25	40-60	0.1
	Легка-1 б	22-24	40-60	0.2

Забезпечення цих параметрів важливо для створення комфортних умов на робочих місцях з ВДТ та забезпечення здоров'я та продуктивності працівників

3.4 Іонізація повітря

Згідно з санітарно-гігієнічними нормами N 2152-80, рівні позитивних і негативних іонів у повітрі приміщень з ВДТ повинні відповідати визначеним стандартам. Таблиця 3.2 містить вказівки щодо цих норм.

Таблиця 3.2

Рівні	Кількість іонів в 1 см куб. повітря	
	N+	N-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

3.5 Пожежна безпека

Пожежна безпека визначається як стан об'єкта, в якому з регламентованою ймовірністю виключається можливість виникнення пожежі та її розповсюдження, а також забезпечується захист людей від небезпечних факторів пожежі і збереження матеріальних цінностей.

Можливі причини пожежі в приміщенні включають коротке замикання проводки, неправильне використання побутових електроприладів і порушення протипожежних заходів.

У випадку виникнення пожежі необхідно:

- негайно відключити живлення електромережі.
- негайно викликати пожежну команду за телефоном.
- Евакуювати людей з приміщення відповідно до плану евакуації.
- Приступити до ліквідації пожежі за допомогою вогнегасників.
- Якщо пожежа мала, можна використати доступні засоби для припинення доступу повітря до вогнища.

На робочому місці для гасіння пожежі використовують вуглекислотні та порошкові вогнегасники. Кількість і види вогнегасників, а також їхнє розташування відповідають вимогам стандартів ГОСТ 12.4.009-75 та ISO3941-77.

					РП 06.12 003.00 ДП ПЗ	Арк.
						57
Змін.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У даному дипломному проєкті було проведено розробку серверної частини та проектування баз даних для веб-сайту коледжу. Метою проєкту було створення ефективної та надійної інфраструктури для забезпечення функціональності та доступності веб-сайту для студентів, викладачів та адміністрації коледжу.

У процесі розробки було використано сучасні технології та методики, зокрема мову програмування для серверної частини, систему управління базами даних та фреймворк для розробки веб-додатків. Було враховано основні вимоги до безпеки, масштабованості та продуктивності системи.

Під час проектування баз даних було визначено оптимальну структуру, враховуючи залежності між сутностями та потреби системи. Були створені таблиці для збереження інформації про студентів, викладачів, курси, розклад та інші потрібні дані. База даних була оптимізована для ефективного доступу до інформації та забезпечення цілісності даних.

В результаті роботи було розроблено серверну частину, яка забезпечує обробку запитів користувачів, аутентифікацію та авторизацію, а також взаємодію з базою даних. Було реалізовано функціонал, що дозволяє студентам переглядати розклад занять та отримувати інформацію про викладачів та курси, Викладачам та адміністрації надано можливість управління курсами, розкладом та відомостями студентів.

Отримані результати свідчать про успішну реалізацію поставлених завдань проєкту. Розроблена серверна частина та бази даних забезпечують надійну та швидку роботу веб-сайту коледжу, що дозволяє користувачам отримувати необхідну інформацію та виконувати потрібні дії без зайвих затримок або проблем.

					РП 06.12 003.00 ДП ПЗ	Арк.
						58
Змін.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pro ASP.NET Core MVC [Книга] / Adam Freeman 2016.
2. ASP.NET Core в дії [Книга] / Andrew Locke 2018.
3. С# в деталях [Книга] / John Skeet 2008.
4. CLR через С# [Книга] / Jeffrey Richter
5. Занурення в ПАТЕРНИ ПРОЕКТУВАННЯ [Книга] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/book>
6. PostgreSQL: Up and Running [Книга] / Reggie Breitenbach 2012.
7. С# 7.0 in a Nutshell [Книга] / Joseph Albahari, Ben Albahari 2017.
8. OpenIdDict [Електроний ресурс] / Calvin Chalet 2023. Посилання: github.com/openiddict/openiddict-core
9. Офіційна документація PostgreSQL [Електроний ресурс] / Michael Stonebraker 2023. Посилання: www.postgresql.org/docs/
10. Офіційний веб-сайт JetBrains [Електроний ресурс] / Максим Шафіров 2023. Посилання: www.jetbrains.com/
11. Офіційна документація Entity Framework [Електроний ресурс] / Microsoft 2023. Посилання: docs.microsoft.com/uk-ua/ef/
12. Книга Entity Framework Core in Action [Книга] / John Smith 2018.
13. Офіційна документація ASP.NET [Електроний ресурс] / Microsoft 2023. Посилання: docs.microsoft.com/uk-ua/aspnet/
14. Патерн Одинак [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/singleton>
15. Патерн Фабричний метод [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/factory-method>
16. Патерн Спостерігач [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/observer>
17. Патерн Шаблонний метод [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/template-method>
18. Патерн Ітератор [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/iterator>

					РП 06.12 000.00 ДП ПЗ	Арк.
Змін.	Арк.	№ докум.	Підпис	Дата		59

19. Патерн Будівельник [Електроний ресурс] / Олександр Швець 2023. Посилання: <https://refactoring.guru/uk/design-patterns/builder>
20. Setting up an Authorization Server with OpenIddict [Електроний ресурс] / Robin van der Кнаар 2020. Посилання: <https://dev.to/robinvanderknaap/setting-up-an-authorization-server-with-openiddict-part-i-introduction-4jid>
21. Intro to Entity Framework Core 5 - New features [Електроний ресурс] / Mohammad Lowend 2021. Посилання: <https://dev.to/moe23/intro-to-entity-framework-core-5-new-features-1kle>

					РП 06.12 000.00 ДП ПЗ	Арк.
						60
Змін.	Арк.	№ докум.	Підпис	Дата		

Фрагмент коду контролеру AuthenticationController.cs

```
namespace AuthService
{
    [ApiExplorerSettings(IgnoreApi = true)]
    public class AuthenticationController : ControllerBase
    {
        private readonly IUserManager _userManager;
        private readonly IRoleManager _roleManager;
        private readonly IOpenIddictTokenManager _tokenManager;

        public AuthenticationController(IUserManager userManager,
            IOpenIddictTokenManager tokenManager,
            IRoleManager roleManager)
        {
            _userManager = userManager;
            _tokenManager = tokenManager;
            _roleManager = roleManager;
        }

        [HttpPost("~/oidc/token")]
        [Consumes("application/x-www-form-urlencoded")]
        [Produces("application/json")]
        public async Task<IActionResult> Exchange()
        {
            var oidcRequest = HttpContext.GetOpenIddictServerRequest();
            if (oidcRequest.IsPasswordGrantType())
                return await TokensForPasswordGrantType(oidcRequest);
        }
    }
}
```

```

if (oidcRequest.IsRefreshTokenGrantType())
{
    return await TokensForRefreshTokenGrantType(oidcRequest);
}

```

```

if (oidcRequest.GrantType == "custom_flow_name")
{
    // return tokens for custom flow
}

```

```

return BadRequest(new OpenIdDictResponse
{
    Error = OpenIdDictConstants.Errors.UnsupportedGrantType
});
}

```

```

/// <summary>

```

```

/// Retrieves user information for the authenticated user.

```

```

/// </summary>

```

```

/// <returns>An <see cref="IActionResult"/> that represents the response to the user
information request.</returns>

```

```

[HttpGet("~/oidc/userinfo")]

```

```

[HttpPost("~/oidc/userinfo")]

```

```

[Authorize]

```

```

public async Task<IActionResult> UserInfo()

```

```

{

```

```

    var subject = User.Claims.First(c => c.Type == "sub").Value;

```

```

    var user = await _userManager.GetShorUserModel(Guid.Parse(subject));

```

```

    if (user == null)

```

```
return NotFound();
```

```
return Ok(user);  
}
```

```
private async Task<IActionResult>  
TokensForPasswordGrantType(OpenIddictRequest request)  
{  
    if (request.Username is null || request.Password is null)  
        return BadRequest(new OpenIddictResponse  
        {  
            Error = OpenIddictConstants.Errors.InvalidRequest  
        });  
  
    var signInResult = await _userManager.SignInAsync(request.Username,  
request.Password);  
  
    var user = await _userManager.GetUserByUsername(request.Username);  
  
    var claimsPrincipal = await CreateClaims(user);  
  
    return SignIn(claimsPrincipal,  
OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);  
}
```

```
private async Task<IActionResult>  
TokensForRefreshTokenGrantType(OpenIddictRequest request)  
{
```

```

        var info = await
HttpContext.AuthenticateAsync(OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);
        if (info.Principal != null)
        {
            var id = Guid.Parse(info.Principal.Claims.First(c => c.Type == "sub").Value);
            var claimsPrincipal = await CreateClaims(id);
            return SignIn(claimsPrincipal,
OpenIddictServerAspNetCoreDefaults.AuthenticationScheme);
        }

        return
            Unauthorized();
    }

```

```

private async Task<ClaimsPrincipal> CreateClaims(User user)
{
    var identity = new ClaimsIdentity(
        TokenValidationParameters.DefaultAuthenticationType,
        OpenIddictConstants.Claims.Name,
        OpenIddictConstants.Claims.Role);

    identity.AddClaim(OpenIddictConstants.Claims.Subject, user.Id.ToString(),
        OpenIddictConstants.Destinations.AccessToken);
    identity.AddClaim(OpenIddictConstants.Claims.Name, user.Username,
        OpenIddictConstants.Destinations.AccessToken);
    // Add more claims if necessary

    var roles = await _roleManager.GetRoleByUserId(user.Id);
    foreach (var role in roles)

```

```
{  
    identity.AddClaim(OpenIddictConstants.Claims.Role, role.NormalizedName,  
        OpenIddictConstants.Destinations.AccessToken);  
}
```

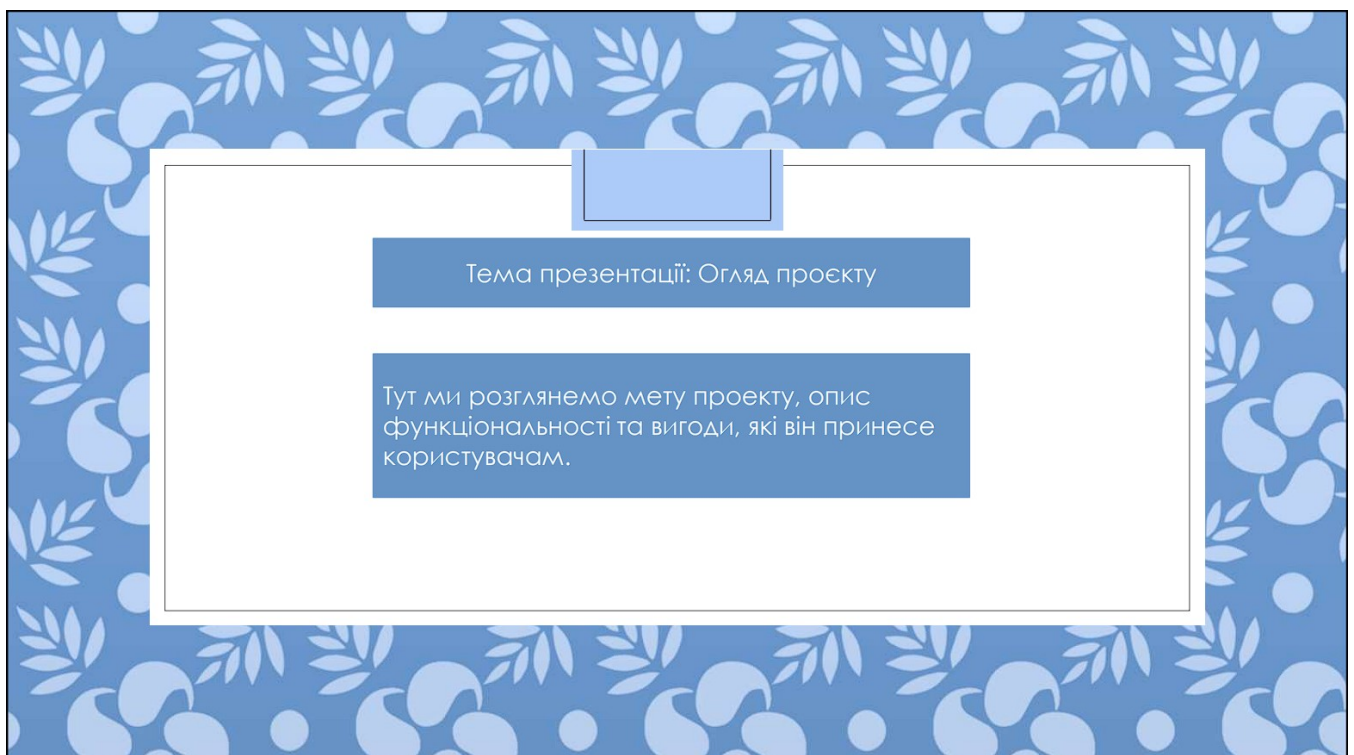
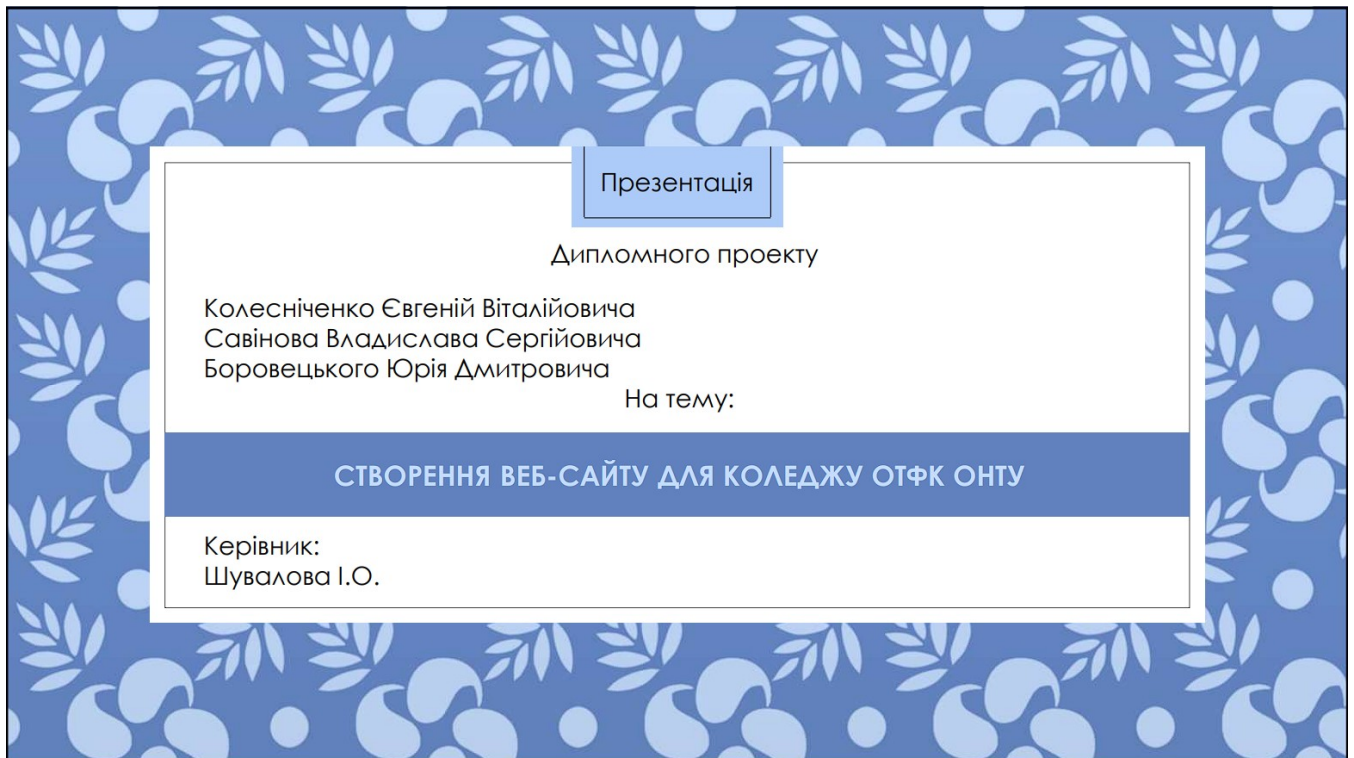
```
var claimsPrincipal = new ClaimsPrincipal(identity);  
claimsPrincipal.SetScopes(new string[]  
{  
    OpenIddictConstants.Scopes.Roles,  
    OpenIddictConstants.Scopes.OfflineAccess,  
    OpenIddictConstants.Scopes.Email,  
    OpenIddictConstants.Scopes.Profile,  
});
```

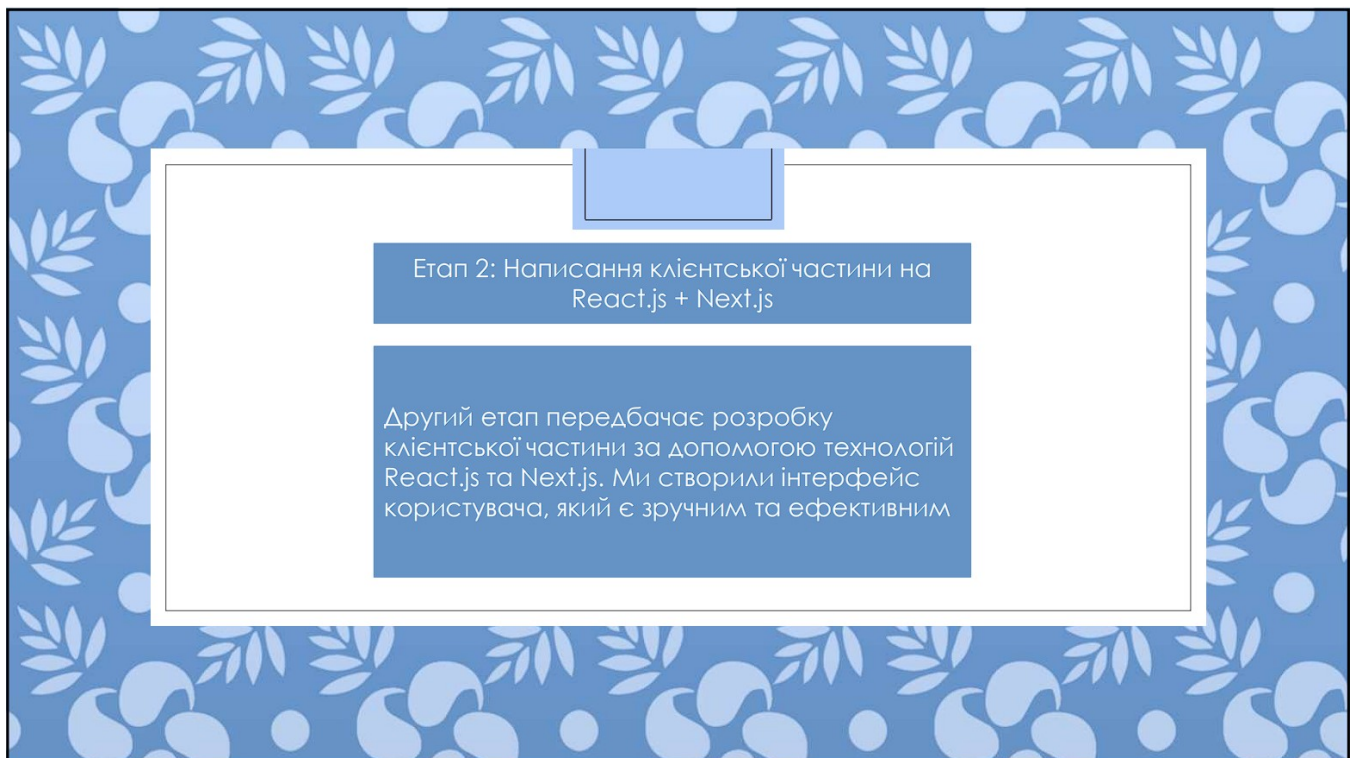
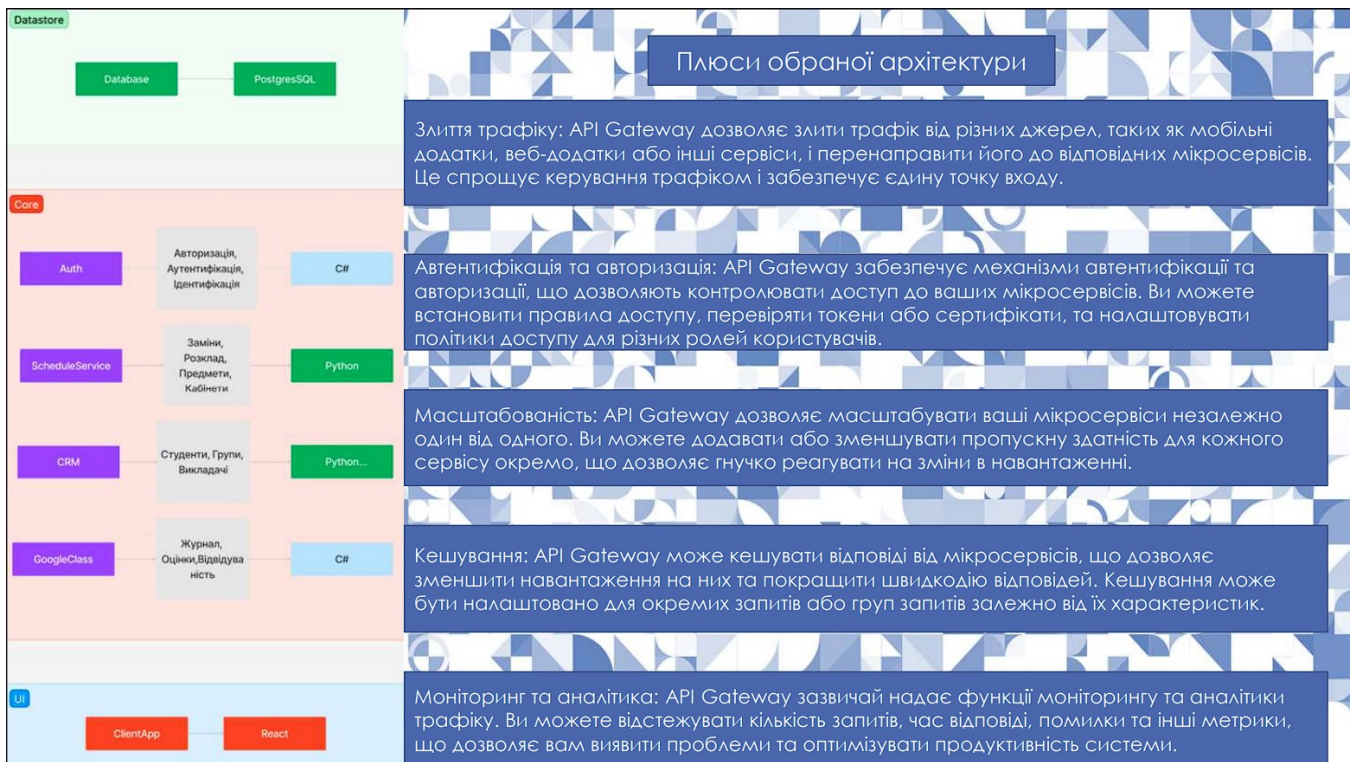
```
return claimsPrincipal;  
}
```

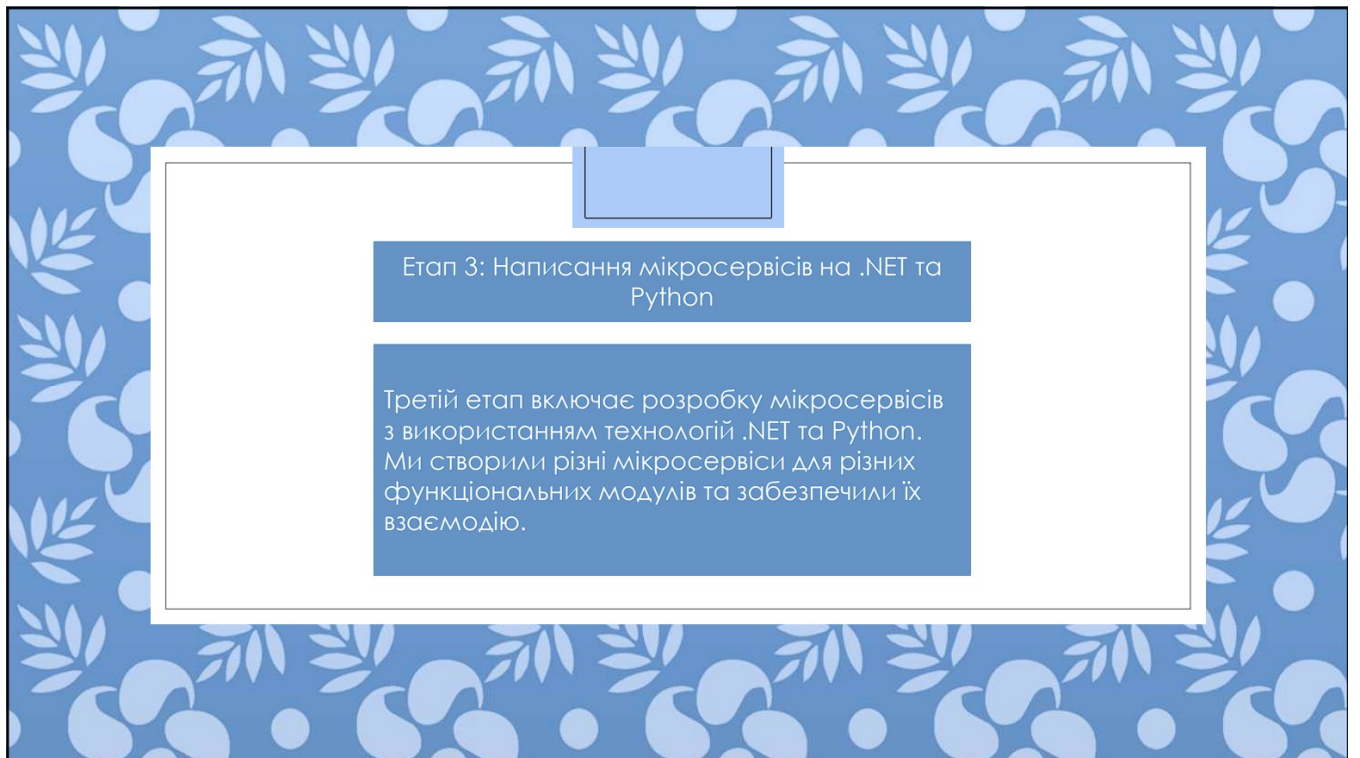
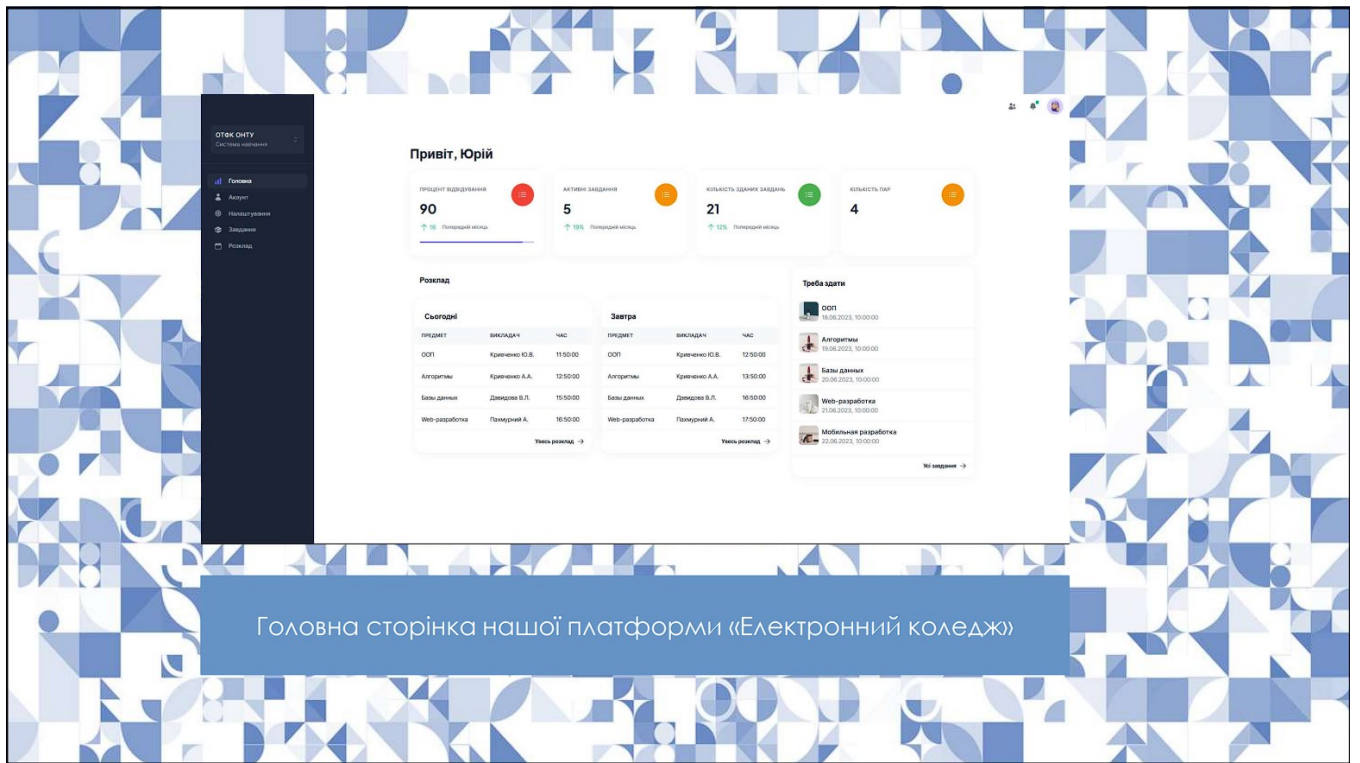
```
private async Task<ClaimsPrincipal> CreateClaims(Guid id)  
{  
    var user = await _userManager.GetUserById(id);  
    var claimsPrincipal = await CreateClaims(user);  
    return claimsPrincipal;  
}
```

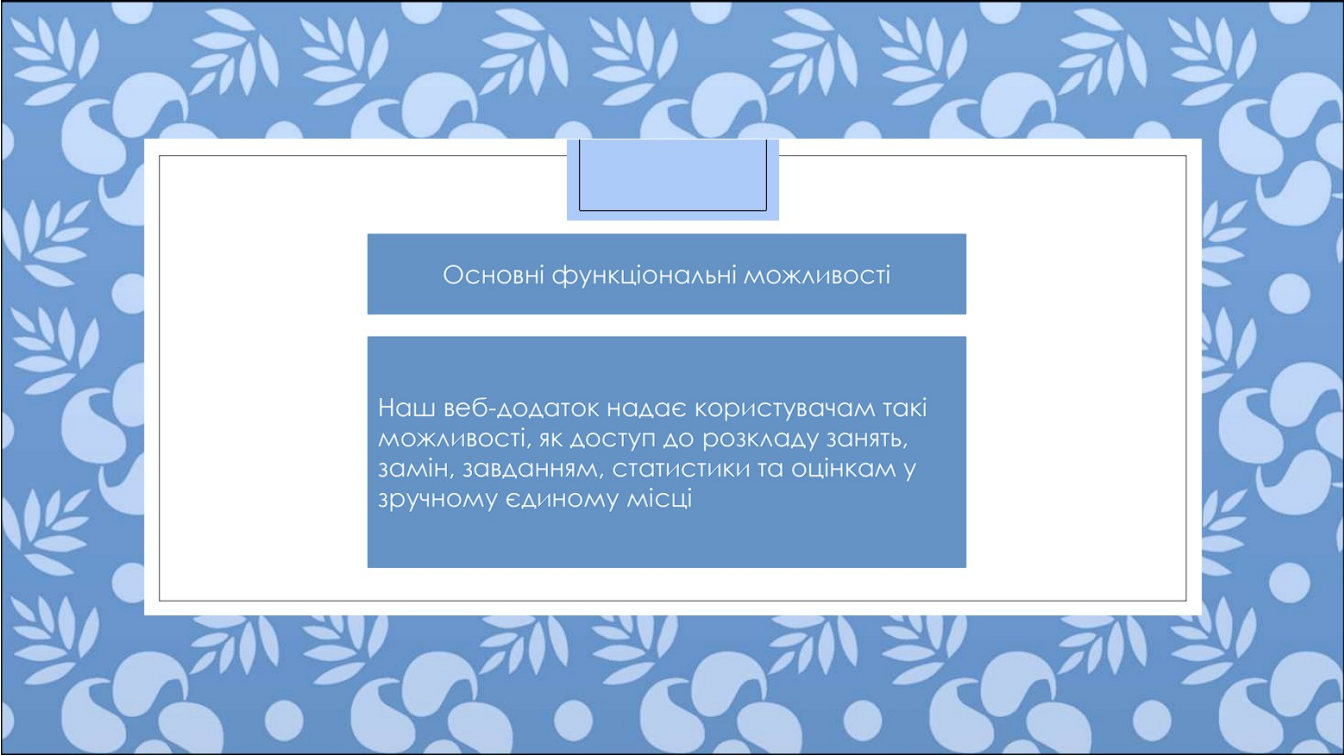
```
}  
  
}
```

Слайди мультимедійної презентації



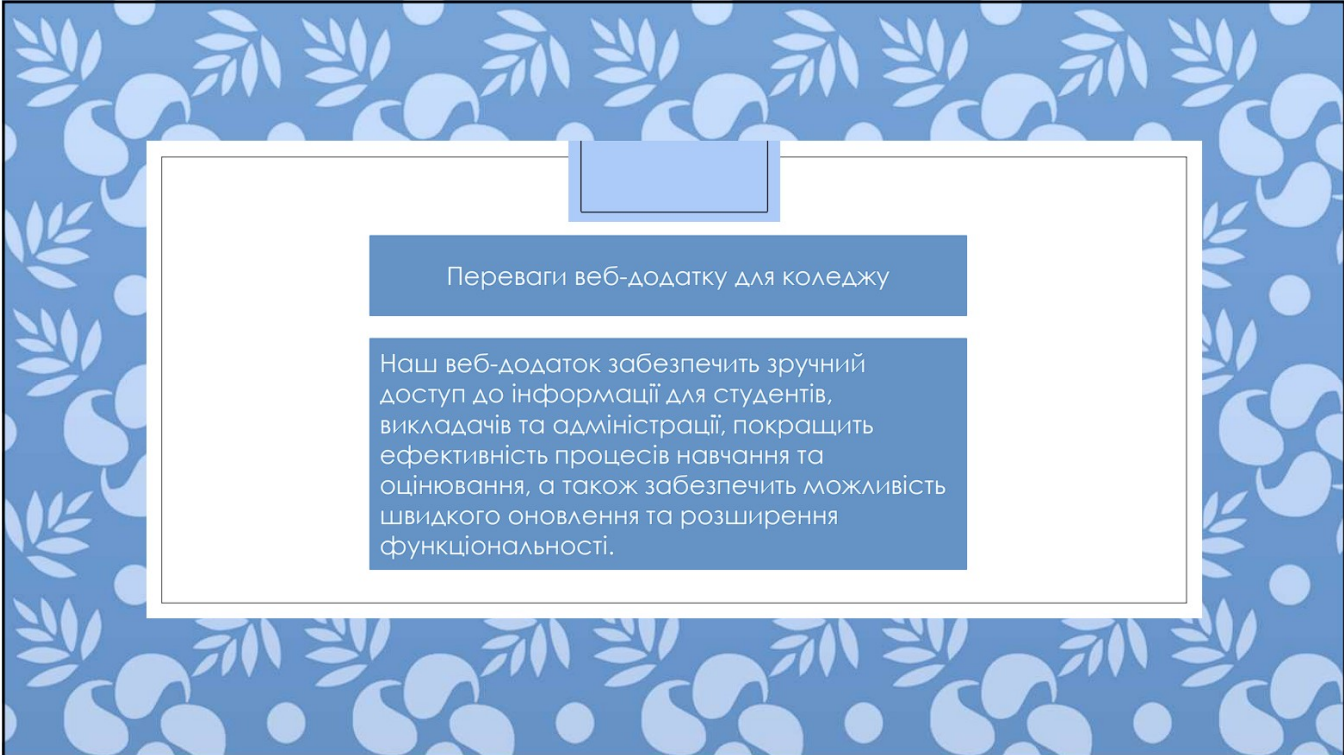






Основні функціональні можливості

Наш веб-додаток надає користувачам такі можливості, як доступ до розкладу занять, замін, завданням, статистики та оцінкам у зручному єдиному місці



Переваги веб-додатку для коледжу

Наш веб-додаток забезпечить зручний доступ до інформації для студентів, викладачів та адміністрації, покращить ефективність процесів навчання та оцінювання, а також забезпечить можливість швидкого оновлення та розширення функціональності.

Ризики та виклики

У процесі реалізації проекту ми зіткнулись з технічними проблемами, терміновими завданнями, а також потребою інтеграції з наявною інфраструктурою. Також ми забезпечили надійне шифрування даних та безпеку їх зберігання

Підсумок

У цій презентації ми оглянули проєкт створення веб-додатку для коледжу, звернули увагу на етапи проєктування мікросервісної архітектури, написання клієнтської частини на React.js + Next.js, а також написання мікросервісів на .NET та Python. Розглянули переваги, а також ризики та виклики, з якими ми стикнулись під час реалізації проєкту.

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти
відділення комп'ютерних систем

Колесніченко Євгеній Віталійович

(прізвище, ім'я та по батькові)

Спеціальність 121 "Інженерія програмного забезпечення"

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Шувалова Ірина Олегівна

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) Розробка серверної частини та проектування
баз даних для сайту ВСП «ОТФК ОНТУ»

Обсяг розрахунково-пояснювальної записки 71 сторінок

Обсяг графічної (презентаційної) частини 10 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню
Дипломний проект відзначається високою актуальністю, так як зосереджується на розробці серверної частини і проектуванні баз даних, що є важливою складовою сучасних веб-сервісів.

б) характеристика виконання кожного розділу дипломного проекту (роботи) _____
Проект складається з різних розділів, які докладно розглядають аспекти створення серверних додатків, проектування та оптимізації баз даних. Підхід до розробки базується на сучасних практиках і стандартах.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Розрахунково-пояснювальна записка містить глибокий аналіз необхідних технологій і методик. Графічна частина яскраво ілюструє архітектурні рішення та схеми баз даних.

г) перелік позитивних якостей дипломного проекту (роботи) _____

До позитивних якостей проекту можна віднести ретельне проектування баз даних, гнучкість архітектури серверної частини, а також увагу до аспектів безпеки.

д) основні недоліки дипломного проекту (роботи) _____

Щодо недоліків, проект міг би мати більш детальний опис підходів до оптимізації продуктивності сервера та можливостей масштабування.

Оцінка розрахункової частини _____ добре

Оцінка графічної частини _____ добре

Загальна оцінка _____ добре

Прізвище, ім'я, по батькові рецензента _____

Васіліу Євген Вікторович

Місце роботи і посада рецензента _____

Державний університет інтелектуальних технологій

і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ, декан факультету інформаційних технологій та кібербезпеки

Підпис: _____

« 10 » 06 2023 р.



ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти
відділення комп'ютерних систем

Колесніченко Євгеній Віталійович

(прізвище, ім'я та по батькові)

Спеціальність: 121 - Інженерія програмного забезпечення

Освітня програма: Розробка програмного забезпечення

Тема дипломного проекту: Розробка серверної частини та проектування
баз даних для сайту ВСП «ОТФК ОНТУ»

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки)

Пояснювальна записка виконана згідно вимогам ДСТУ. Обсяг і зміст роботи повністю відповідають завданню на дипломне проектування. Розрахунково-пояснювальна записка проекту виконано грамотно.

б) самостійність роботи над проектом:

Колесніченко Є. В. оброблено науковий матеріал, проаналізовані методи і підходи до розробки веб-орієнтованих систем, уміння аналізувати літературні джерела та застосовувати сучасні інформаційні технології при моделюванні та експериментальних дослідженнях.

в) теоретична підготовка випускника (випускниці):

Теоретична і практична підготовка студента Колесніченко Є.В. на високому рівні. Продемонстрував зібраність, старанність, відповідальність.

г) вміння розв'язувати виробничі та конструкторські питання _____

Студент Колесніченко Є.В. за період роботи показав Високий рівень теоретичної та практичної підготовки

Оцінка розрахункової частини «Відмінно»

Оцінка графічної частини «Відмінно»

Загальна оцінка «Відмінно»

Прізвище, ім'я, по батькові керівника дипломного проекту _____

Шувалова Ірина Олегівна

Місце роботи і посада керівника дипломного проекту викладач

Підпис _____

« 09 » 06 2023 р.

Ім'я користувача:
Наталія Вікторівна Копусь

Дата перевірки:
12.06.2023 09:31:44 EEST

Дата звіту:
12.06.2023 09:40:43 EEST

ID перевірки:
1015553880

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100011688

Назва документа: 4РП-06 Колесніченко Є.В

Кількість сторінок: 53 Кількість слів: 9565 Кількість символів: 74449 Розмір файлу: 2.43 MB ID файлу: 1015206114

3.05% Схожість

Найбільша схожість: 0.93% з Інтернет-джерелом (https://gov.liga.com.ua/b_text.php?base=1&id=1016192&type=3)

3.05% Джерела з Інтернету

428

Сторінка 55

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0% Вилучень

Немає вилучених джерел

**ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»**

Ми, що нижче підписалися,

Колесніченко Євгеній Віталійович,
здобувач освіти гр. 4РП-06, та

Шувалова Ірина Олегівна,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи молодшого спеціаліста на тему:

«Розробка серверної частини та проектування баз даних для сайту ВСП «ОТФК ОНТУ»» (автор роботи – Колесніченко Є.В., керівник роботи – Шувалова І.О.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець  / Колесніченко Є.В. /

Керівник  / Шувалова І.О. /

« 09 » 06 20 23 р.