Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

Кафедра Інформаційних систем та технологій

Лабораторна робота №4

з дисципліни « Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему: « Імплементація REST API»

Виконав:

студент гр. ІС-11

Воробйов Олексій

Викладач:

Бардін В.

2023 рік

**Завдання:**

1. Використовуючи архітектуру розроблену в попередній роботі

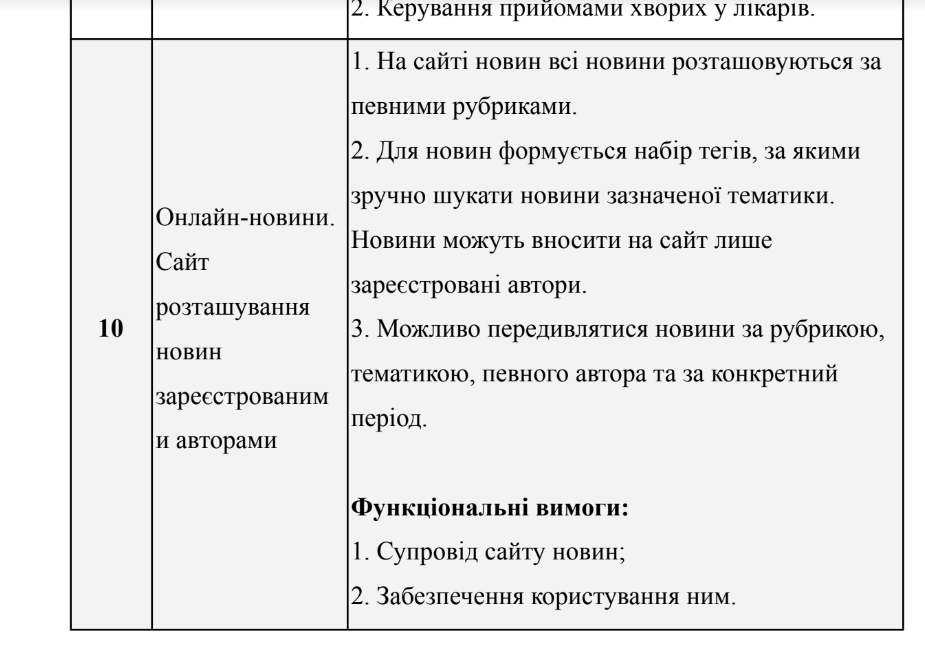
Імплементувати REST веб-API, використавши N Layered архітектуру.

2. Покрити модульними та інтеграційними тестами основні

компоненти рішення.

3. Експортувати розроблене API до Postman.

**Варіант 10:**



**ER-діаграма:**

Зображення, що містить текст, число, знімок екрана, дизайн

Автоматично згенерований опис Посилання доступу: <https://dbdiagram.io/d/Lab3-653d0f68ffbf5169f0a33a94>

**Таблиця:** Authors

**Призначення:** Облікова таблиця зареєстрованих авторів

**Властивості**:

* **Id**: Guid – унікальний ідентифікатор автора
* **Email**: string – поштова адреса автора
* **FullName**: string – повне ім’я автора
* **Sex**: bool? – стать автора
* **PublicInformation**: string? – публічна інформація про автора
* **DateOfRegistration**: DateTime – дата реєстрації автора
* **BirthDate**: DateTime – дата народження автора
* **CreatedAt**: DateTime – дата створення сутності автора
* **UpdatedAt**: DateTime – дата оновлення сутності автора

**Таблиця**: News

**Призначення**: Таблиця новин

**Властивості**:

* **Id**: Guid – унікальний ідентифікатор новини
* **Subject**: string – заголовок новини
* **Content**: string – вміст новини
* **CreatedAt**: DateTime – дата створення сутності новини
* **UpdatedAt**: DateTime – дата оновлення сутності новини

**Таблиця**: Rubrics

**Призначення**: Таблиця рубрик

**Властивості**:

* **Id**: Guid – унікальний ідентифікатор рубрики
* **Name**: string – назва рубрики
* **CreatedAt**: DateTime – дата створення сутності рубрики
* **UpdatedAt**: DateTime – дата оновлення сутності рубрики

**Таблиця**: NewsRubrics

**Призначення**: Допоміжна таблиця новин за рубриками

**Властивості**:

* **NewsId**: Guid – унікальний ідентифікатор новини
* **RubricId**: Guid – унікальний ідентифікатор рубрики

**Таблиця**: Tags

**Призначення**: Таблиця тегів

**Властивості**:

* **Id**: Guid – унікальний ідентифікатор тегу
* **Name**: string – назва тегу
* **CreatedAt**: DateTime – дата створення сутності тегу
* **UpdatedAt**: DateTime – дата оновлення сутності тегу

**Таблиця**: NewsTags

**Призначення**: Допоміжна таблиця новин за тегами

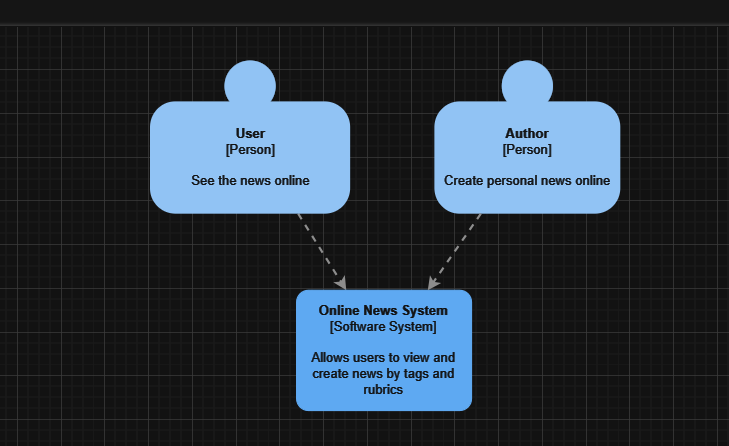
**Властивості**:

* **NewsId**: Guid – унікальний ідентифікатор новини
* **TagId**: Guid – унікальний ідентифікатор тегу

**C4**

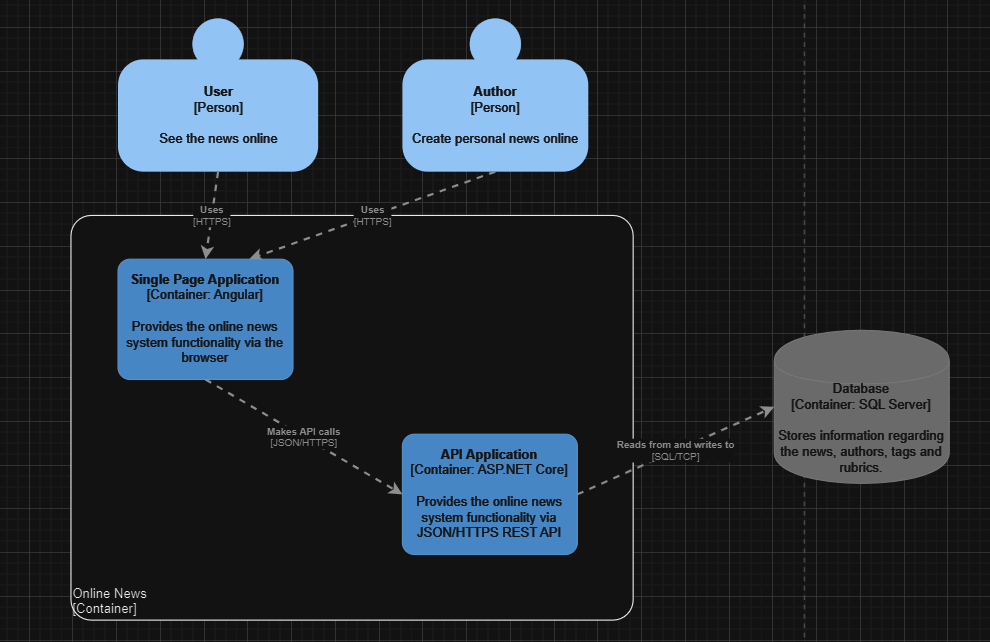
Лінк доступу: <https://drive.google.com/file/d/1KvfUPK10xsk82m1y1-O7sXpXMkrFu8y6/view?usp=sharing>

**Context**



Система онлайн-новин має 2 типи користувачів: User – неавторизований, може проглядати новини онлайн, виділяти за тегами та рубриками. Author – авторизований, може проглядати новини онлайн, виділяти за тегами та рубриками та створювати власні новини.

**Container**



Система онлайн нових розділяється на три основні компоненти:

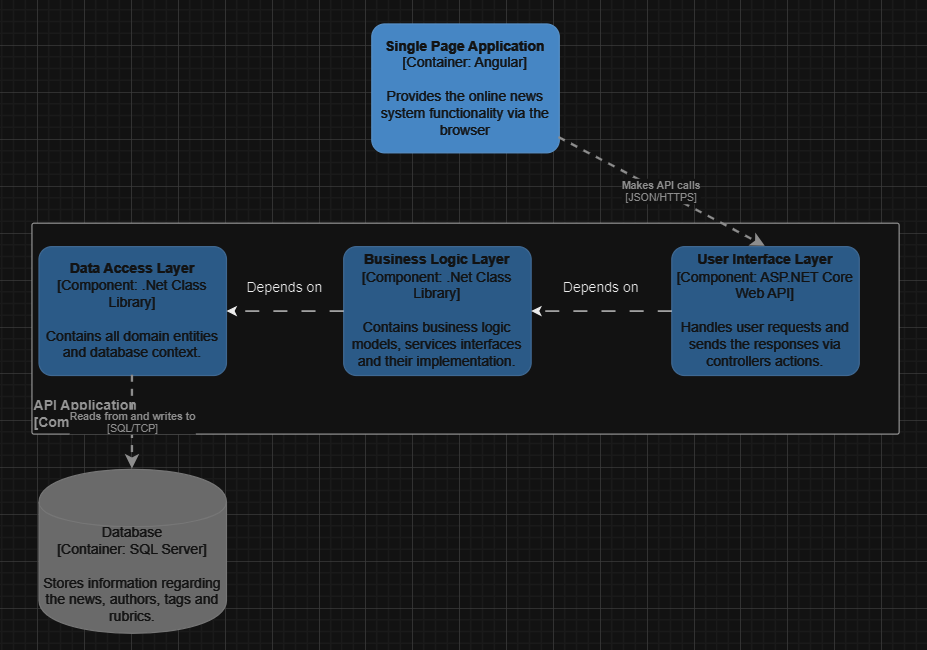
1. Односторінковий веб-застосунок, який надає функціонал онлайн нових

через веб-браузер.

2. Серверний застосунок, який реалізує функціонал через REST API

3. База даних, яка зберігає усю інформацію про новини, авторів, теги та рубрики.

**Component**



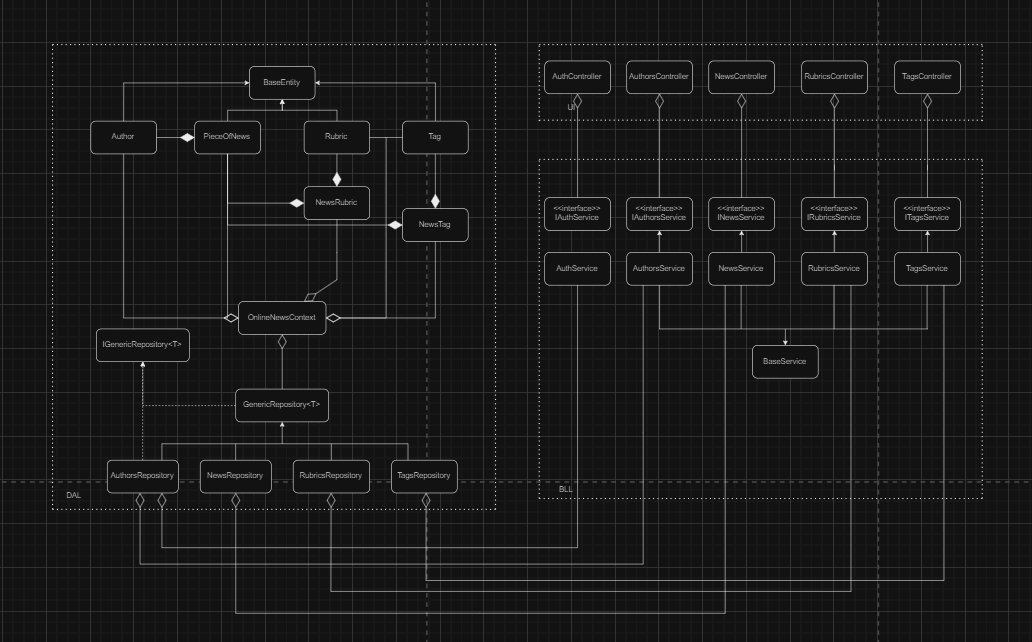
Для розробки .Net застосунку буде застосована “N-Layer” архітектура, яка розділяє застосунок на наступні шари (layers):

1. Data Access Layer (DAL) – надає спрощений доступ до даних, що зберігаються в певному постійному сховищі – SQL Server Database. Містить в собі усі доменні сутності (Domain entities) і контекст бази даних.

2. Business Logic Layer (BLL) – містить у собі усю бізнес-логіку застосунку, інтерфейси та імплементацію усіх сервесів. Залежить від DAL.

3. User Interface Layer (UI) – шар, який відповідає за обробку запитів користувача. Містить контролери. Залежить від BLL.

**Code**



Серед особливостей: клас BaseEntity буде використовуватись як базовий для усіх базових сутностей і міститиме спільні для усіх базових сутностей властивості (наприклад, Id, CreatedAt, UpdatedAt). Схожу функцію виконуватиме базовий абстрактний клас BaseService, який міститиме спільні для усіх сервісів поля (наприклад, UserManager).

Крім того, DAL міститиме сутності як Author, News, Rubric, Tag, NewsRubric, NewsTag та контекст бази даних – OnlineNewsContext. Також буде реалізовано патерн Repository з загальним інтерфейсом IGenericRepository<T> та загальним базовим класом для всіх репозиторіїв GenericRepository<T>.

BLL міститиме базовий абстрактний клас BaseService та інтерфейси сервісів IauthService, IAuthorsService, INewsService, IRubricsService, ITagsService та їх імплементацію відповідно: AuthService, AuthorsService, NewsService, RubricsService, TagsService.

UI міститиме контролери: AuthorsController, RubricsController, TagsController NewsController та AuthController.

**Допоміжні класи для Endpoints:**

**Класи для запиту:**

*PageSettings* – відповідає за фільтрацію, сортування та пагінацію даних.

*PageList<T>* - відповідатиме за відфільтроване, відсортоване та пагінований набір даних типу T, що повертається в якості результату запиту.

*UpdatedAuthorRequest* – клас, який має усі властивості, які може редагувати автор про себе.

*NewNewsRequest –* клас, який має усі властивості для створення нової новини.

*UpdatedNewNewsRequest –* клас, який має усі властивості для оновлення інформації про новину.

*UserLoginRequest* – клас, який має усі властивості для того, щоб залогінити користувача в систему.

*UserRegisterRequest* – клас, який має усі властивості для того, щоб зареєструвати користувача в систему.

*NewTagRequest* – клас, який має усі властивості для створення нового тегу.

*UpdateTagRequest* – клас, який має усі властивості для оновлення інформації про тег.

*NewRubricRequest* – клас, який має усі властивості для створення нової рубрики.

*UpdateRubricRequest* – клас, який має усі властивості для оновлення інформації про рубрику.

**Класи для відповіді:**

*AuthorResponse –* клас, який міститиме усі властивості про автора з відповіді.

*NewsResponse* – клас, який міститиме усі властивості про новину з відповіді.

*NewUserResponse* – клас, який міститиме усі властивості про зареєстрованого користувача.

*LoginUserResponse* - клас, який міститиме усі властивості про залогіненого користувача та його JWT токен.

*TagResponse –* клас, який міститиме усі властивості про тег.

*RubricResponse –* клас, який міститиме усі властивості про рубрику.

**Endpoints**:

**AuthorsController:**

* *api/authors/{pageSettings} –* GET (PageList<AuthorResponse>) **GetAllAuthorsAsync** (PageSettings pageSettings) - Отримати всіх авторів новин.
* *api/authors/{id} –* GET (AuthorResponse) **GetAuthorAsync** (Guid id) – Отримати автора за його id.
* *api/authors/* - PUT (AuthorResponse) **UpdateAuthorAsync** ([Body] UpdatedAuthorRequest newAuthor) – Оновити інформацію автора про себе.
* *api/authors/{id}* – DELETE () **DeleteAuthorAsync** (Guid id) – Видалити автора і всі пов’язані з ним новини.

**NewsController:**

* *api/news/{pageSettings}* – GET (PageList<NewsResponse*>*) **GetAllNewsAsync** (PageSettings pageSettings) – Отримати всі новини.
* *api/news/by-rubric/{rubricId}&{pageSettings}* - GET (PageList<NewsResponse*>*) **GetNewsByRubricAsync** (Guid RubricId, PageSettings pageSettings) – Отримати новини за рубрикою.
* *api/news/by-tags/{tagsIds}&{pageSettings}* - GET (PageList<NewsResponse*>*) **GetNewsByTagsAsync** (List<Guid> TagsIds, PageSettings pageSettings) – Отримати новини за тегами.
* *api/news/by-author/{authorId}&{pageSettings} –* GET (PageList<NewsResponse*>*) **GetNewsByAuthorAsync** (Guid AuthorId, PageSettings pageSettings) – Отримати новини конкретного автора.
* *api/news/{startDate}&{endDate}&{pageSettings}* - GET (PageList<NewsResponse*>*) **GetNewsByPeriodOfTimeAsync** (DateTime startDate, DateTime endDate, PageSettings pageSettings) – Отримати новини за проміжок часу від startDate до endDate.
* *api/news*/ - POST (NewsResponse) **CreateNewNewsAsync** ([Body] NewNewsRequest newNews) – Створити нову новину.
* *api/news* – PUT (NewsResponse) **UpdateNewsAsync** ([Body] UpdatedNewNewsRequest updatedNews) - Оновити вміст новини.
* *api/news/{id}* – DELETE () **DeleteNewsAsync** (Guid id) – Видалити новину.

**AuthController:**

* *api/auth/login* – POST (LoginUserResponse) **LoginAsync** ([Body] UserLoginRequest userLogin) – Залогінитись в систему.
* *api/auth/register* - POST (NewUserResponse) **RegisterAsync** ([Body] UserRegisterRequest userRegister) – Зареєструватись в систему.

**TagsController:**

* *api/tags* – GET (PageList<TagResponse>) **GetAllTagsAsync** (PageSettings pageSettings) – Отримати всі теги.
* *api/tags/{id}* – GET (TagResponse) **GetTagByIdAsync** (Guid id) – Отримати тег за його id.
* *api/tags* – POST (TagResponse) **CreateNewTagAsync** ([Body] NewTagRequest newTag) – Створити новий тег.
* *api/tags* – PUT (TagResponse) **UpdateTagAsync** ([Body] UpdateTagRequest updateTag) – Оновити інформацію про тег.
* *api/tags/{id}* – DELETE () **DeleteTagAsync** (Guid id) – Видалити тег.

**RubricsController:**

* *api/rubrics* – GET (PageList<RubricResponse>) **GetAllRubricsAsync** (PageSettings pageSettings) – Отримати всі рубрики.
* *api/rubrics/{id}* – GET (RubricResponse) **GetRubricByIdAsync** (Guid id) – Отримати рубрику за його id.
* *api/rubrics* – POST (RubricResponse) **CreateNewRubricAsync** ([Body] NewRubricRequest newRubric) – Створити нову рубрику.
* *api/rubrics* – PUT (RubricResponse) **UpdateRubricAsync** ([Body] UpdateRubricRequest updateRubric) – Оновити інформацію про рубрику.
* *api/rubrics/{id}* – DELETE () **DeleteRubricAsync** (Guid id) – Видалити рубрику.

**Питання до роботи:**

*1)* *Що таке N Layered та K Tiered архітектура, і які їх основні*

*компоненти?*

**N Layered Architecture**:

Основна ідея: В N Layered архітектурі програмне забезпечення розбивається на різні "шари" або рівні, кожен з яких відповідає за певні аспекти функціоналу. Основні компоненти:

* Presentation Layer (Шар презентації): Відповідає за взаємодію з користувачем та відображення даних.
* Business Logic Layer (Шар бізнес-логіки): Містить логіку програми та обробку бізнес-правил.
* Data Access Layer (Шар доступу до даних): Забезпечує доступ до бази даних та взаємодію з нею.

**K Tiered Architecture**:

Основна ідея: В K Tiered архітектурі програмне забезпечення розбивається на "рівні" або "сервери", де кожен сервер відповідає за конкретний аспект додатка.Основні компоненти:

* Client Tier (Клієнтський рівень): Містить інтерфейс користувача та відповідає за взаємодію з користувачем.
* Application Tier (Рівень застосування): Містить бізнес-логіку та обробку запитів від клієнта.
* Data Tier (Рівень даних): Забезпечує доступ до бази даних та управління даними.

*2)* *Як в ASP.NET Core реалізується шар Data Access Layer (DAL) з*

*використанням EF Core?*

ASP.NET Core зазвичай використовує Entity Framework Core (EF Core) для доступу до бази даних. Шар Data Access Layer (DAL) включає в себе всі операції з базою даних, такі як запити, оновлення та видалення даних.

*3)* *Опишіть основні принципи роботи BLL (Business Logic Layer) в*

*контексті архітектури N Layered.*

Основні принципи роботи BLL в архітектурі N Layered включають:

* **Розділення відповідальностей**: BLL відповідає за логіку бізнес-процесів і не повинен безпосередньо взаємодіяти з базою даних або з даними, представленими користувачу. Його завдання - забезпечити правильність і цілісність даних, обробляти бізнес-правила та здійснювати потрібні перевірки.
* **Валідація даних**: BLL виконує перевірки введених даних для переконання в їхній правильності та відповідності бізнес-правилам. Це включає в себе перевірки на відповідність формату, обов'язкові поля, унікальність тощо.
* **Обробка бізнес-логіки**: Логіка, яка визначає, як система повинна взаємодіяти з даними та виконувати бізнес-правила, розміщена в BLL. Це може включати розрахунки, агрегації, перетворення даних і інші операції, необхідні для досягнення бізнес-цілей.
* **Незалежність від інфраструктури**: BLL повинен бути незалежним від конкретної реалізації Data Access Layer та Presentation Layer. Він не повинен напряму залежати від технологій бази даних або механізмів відображення даних на інтерфейс користувача.
* **Взаємодія з DAL**: Хоча BLL самостійно не взаємодіє з базою даних, він використовує сервіси, які надаються Data Access Layer. Це може бути шар репозиторіїв або сервісів, які надають доступ до даних.
* **Тестування**: Логіка в BLL повинна бути покрита тестами для забезпечення її коректності та стабільності. Це сприяє підтримці високої якості коду та можливості швидко виявляти та виправляти помилки.
* **Використання інтерфейсів**: Використання інтерфейсів сприяє вирішенню проблеми розділення відповідальностей і дозволяє вам легко замінювати частини бізнес-логіки без впливу на інші частини системи.

*4)* *Що таке REST API і як його можна імплементувати у ASP.NET Core?*

REST API (Representational State Transfer Application Programming Interface) - це архітектурний стиль для розробки веб-служб, який використовує стандартні HTTP-протоколи для отримання, внесення змін, видалення або перегляду ресурсів. REST часто використовується для побудови веб-сервісів, які є легкими, масштабованими та можуть взаємодіяти з різними клієнтами.

*5)* *Як Specification Pattern інтегрується в архітектуру рішення і які*

*проблеми він допомагає вирішити?*

Specification Pattern - це патерн проектування, який дозволяє визначати бізнес-правила у вигляді об'єктів, які можуть бути складені для створення складних запитів або умов для валідації об'єктів. Цей патерн інтегрується в архітектуру рішення, дозволяючи зручно виражати та використовувати бізнес-правила в коді.

*6)* *Що таке NSubstitute і як його використовувати для модульного*

*тестування в .NET?*

NSubstitute - це бібліотека, яка надає можливості для створення підроблених (mock) об'єктів під час модульного тестування. Вона дозволяє замінювати реальні об'єкти у ваших тестах на контрольовані підроблені об'єкти, щоб перевірити, чи правильно вони використовуються та взаємодіють з іншими частинами коду.

*7)* *Що таке AutoBogus і як його можна використати для автоматичної*

*генерації тестових даних?*

AutoBogus - це бібліотека для мови програмування C#, яка дозволяє автоматично генерувати тестові дані для об'єктів на основі їхніх типів та правил. Це дозволяє розробникам швидко та зручно створювати тестові дані для модульних тестів або інших сценаріїв, де потрібно заповнити об'єкти фіктивними, але реалістичними значеннями.

*8)* *Як Microsoft.DependencyInjection спрощує роботу з залежностями в*

*ASP.NET Core?*

Microsoft.DependencyInjection є контейнером вбудованих служб та системою управління залежностями для ASP.NET Core. Вона спрощує роботу з залежностями в ASP.NET Core та дозволяє легко використовувати інверсію управління (Inversion of Control, IoC) та інжекшн залежностей (Dependency Injection, DI) у проекті.

*9)* *Як використовувати Specification Pattern для створення гнучких і*

*повторно використовуваних запитів до бази даних?*

Specification Pattern - це патерн проектування, який дозволяє виокремлювати бізнес-логіку, пов'язану з фільтрацією та умовами, від логіки доступу до бази даних. Використання Specification Pattern для створення гнучких та повторно використовуваних запитів до бази даних може значно полегшити розробку та підтримку коду.

*10) Що представляє собою Dependency Injection в ASP.NET Core і як він*

*реалізований з допомогою Microsoft.DependencyInjection?*

Dependency Injection (DI) - це принцип програмування, за яким клас отримує всі свої залежності зовні, замість того, щоб створювати їх самостійно. Це робить код більш гнучким, тестируємим та легше супроводжуваним. В ASP.NET Core, DI вбудований у фреймворк, і Microsoft надає бібліотеку Microsoft.DependencyInjection для реалізації цього принципу.

*11) Як створити та використовувати фейкові об'єкти з допомогою*

*NSubstitute для модульного тестування в ASP.NET Core?*

Пункт 6

*12) Як експортувати розроблене API до Postman, і які переваги це дає*

*при тестуванні та документуванні API?*

Експорт API до Postman - це важливий етап розробки, оскільки Postman надає зручні інструменти для тестування, документування та взаємодії з API. Вотнеденні завдання, такі як надсилання запитів, перевірка відповідей, автоматизація тестування та створення колекцій для спільної роботи, можна легко виконати в Postman.

*13) Що таке міграції в EF Core і як їх використовувати для управління*

*схемою бази даних?*

Міграції (Migrations) в Entity Framework Core (EF Core) - це механізм, який дозволяє автоматично оновлювати схему бази даних при зміні моделі даних або при вперше створенні бази даних. Міграції допомагають управляти версіями бази даних і забезпечують автоматизований спосіб внесення змін до бази даних без втрати даних.

*14) Як реалізувати аутентифікацію та авторизацію в ASP.NET Core REST*

*API?*

Аутентифікація та авторизація - це ключові аспекти забезпечення безпеки веб-додатків, включаючи REST API в ASP.NET Core. Реалізувати їх можна за домогою Microsoft Identity.