Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

Кафедра Інформаційних систем та технологій

Лабораторна робота №3

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

на тему: « Проектування REST веб-API»

Виконав:

студент гр. ІС-11

Воробйов Олексій

Викладач:

Бардін В.

2023 рік

**Мета**:

1. Ознайомитися з основами створення REST веб-API та методологією

C4 для відображення архітектури системи.

2. Ознайомитися з основами створення ER-діаграм для представлення

структури бази даних.

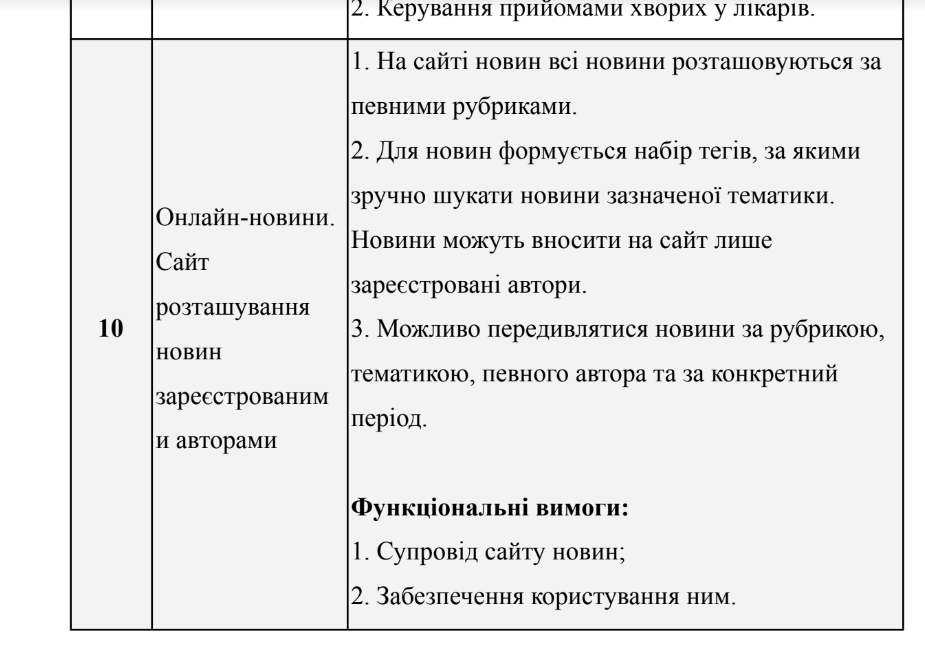
**Завдання:**

1. З дотриманням вимог REST-у спроектувати веб-API для обраної(згідно варіанту) доменної області, використовуючи методологію C4 для створення діаграми архітектури системи.

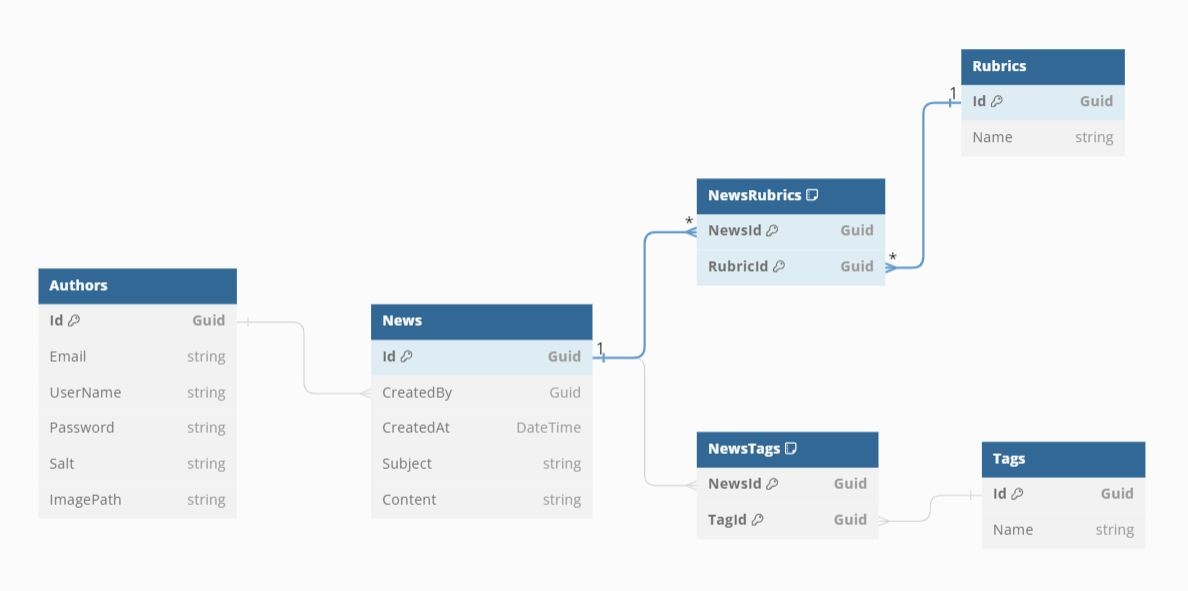
2. Створити ER-діаграму для DAL (Data Access Layer), яка відображатиме структуру бази даних веб-API.

3. Оформити спроєктоване рішення у вигляді звіту до лабораторної роботи.

**Варіант 10:**



**ER-діаграма:**

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
* **UserName**: string – унікальне ім’я автора
* **Password**: string – захешований пароль автора
* **Salt**: string – унікальне згенероване значення, потрібне для хешування паролю
* **ImagePath**: string – шлях до файлу з фото автора

**Таблиця**: Rubrics

**Призначення**: Таблиця рубрик

**Властивості**:

* **Id**: Guid – унікальний ідентифікатор рубрики
* **Name**: string – назва 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

**Таблиця**: NewsTags

**Призначення**: Допоміжна таблиця новин за тегами

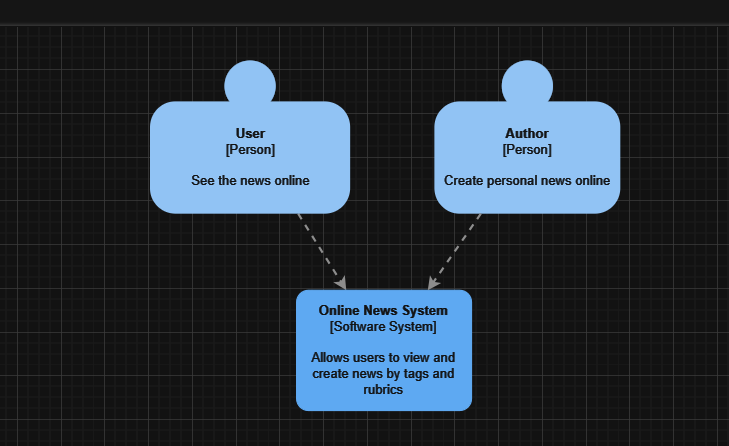
**Властивості**:

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

**C4**

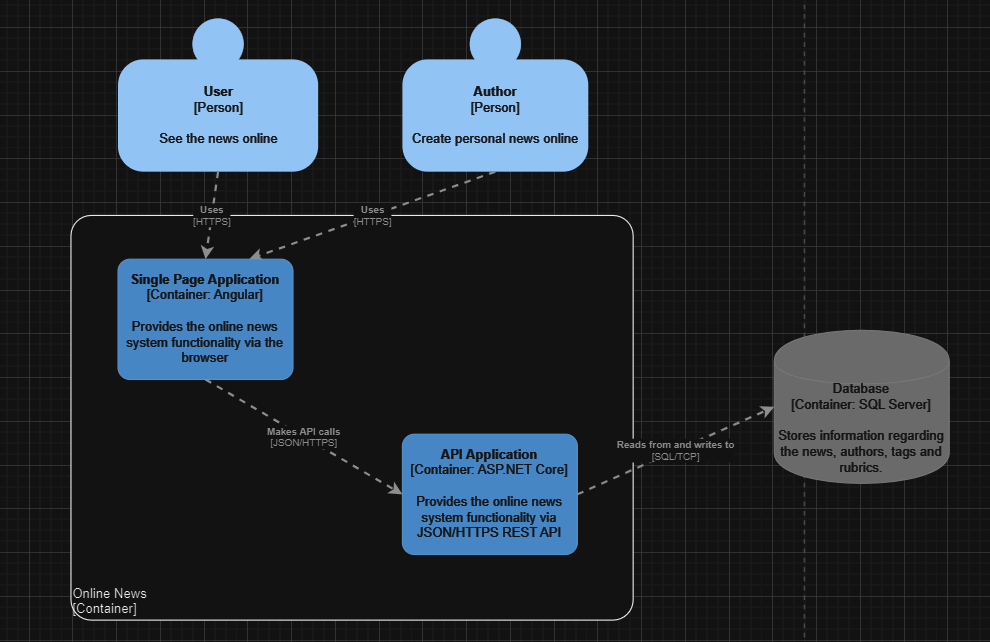
Лінк доступу: <https://drive.google.com/file/d/1KvfUPK10xsk82m1y1-O7sXpXMkrFu8y6/view?usp=sharing>

**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:

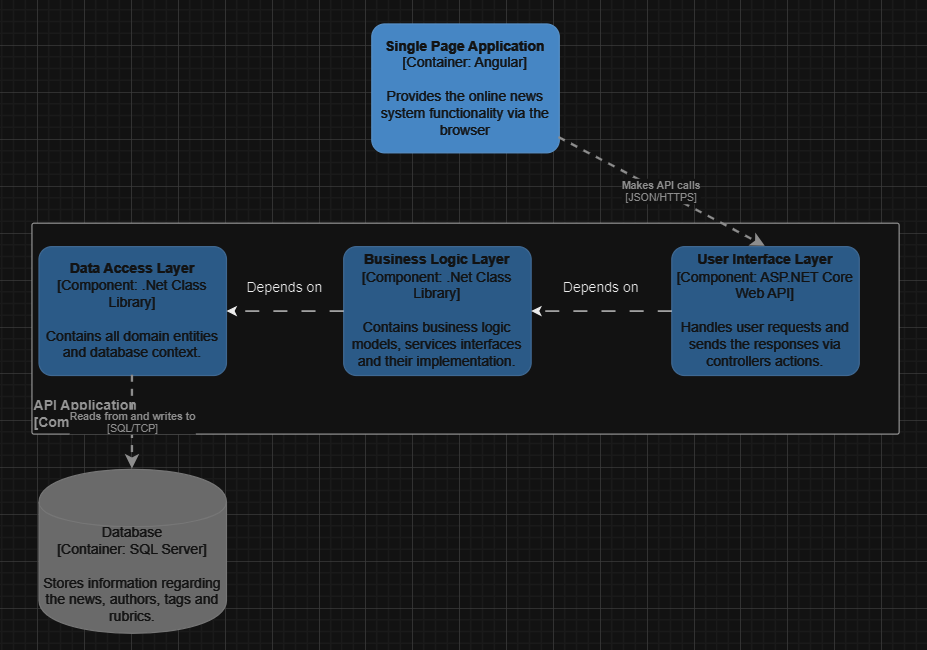
1. Односторінковий веб-застосунок, який надає функціонал онлайн нових

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**



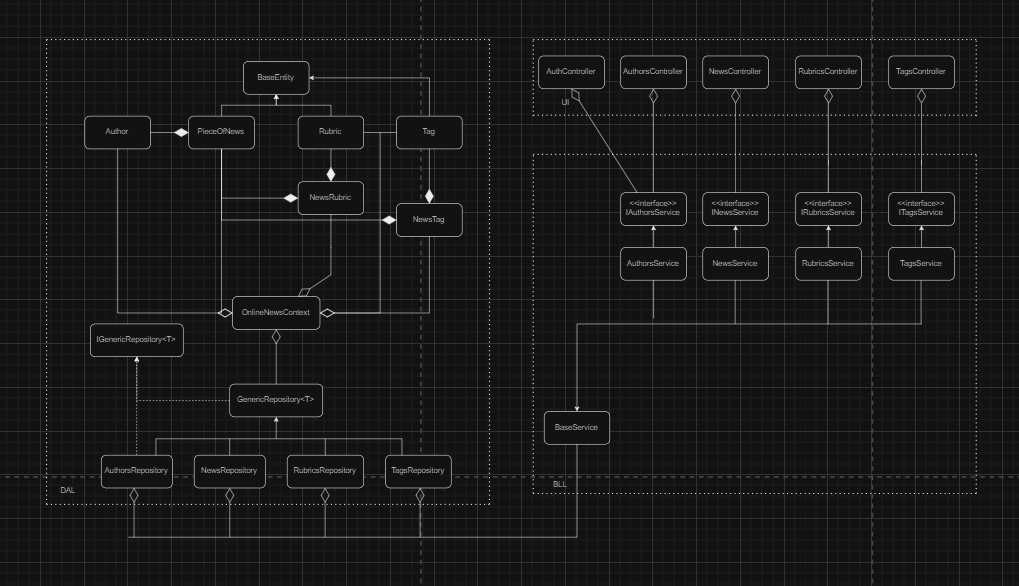
Для розробки .Net застосунку буде застосована “N-Layer” архітектура, яка розділяє застосунок на наступні шари (layers):

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 (окрім Author, який пронаслідує IdentityUser) і міститиме спільні для усіх базових сутностей властивості (наприклад, Id). Схожу функцію виконуватиме базовий абстрактний клас BaseService, який міститиме спільні для усіх сервісів поля (наприклад, контекст бази даних).

Крім того, DAL міститиме сутності як Author, PieceOfNews, 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 IAuthorsService, INewsService, IRubricsService, ITagsService та їх імплементацію відповідно: 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.

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

*UpdatedNewPieceOfNewsRequest –* 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.

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

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

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

**AuthController:**

* *api/auth/login* – POST (NewUserResponse) **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 REST веб-API та які його основні принципи?*

REST (Representational State Transfer) є архітектурним стилем для розробки веб-сервісів, який базується на принципах взаємодії між клієнтом і сервером. Основні принципи REST допомагають створювати ефективні, масштабовані та добре організовані веб-сервіси. Серед основних принципів:

1. Client-Server Architecture (Клієнт-Сервер)

2. Statelessness (Відсутність стану)

3. Cacheability (Кешування)

4. Uniform Interface (Однорідний інтерфейс)

5. Layered System (Шари абстракції)

6. Code on demand (Запитування коду)

*2)* *Які основні компоненти включаються в архітектуру REST API?*

В архітектуру REST API включаються три основні компоненти: URL Endpoint, HTTP Verb, Body.

**URL Endpoint**: URL-посилання, яке представляє ресурси, до яких ми хочемо отримати доступ. Ресурсами можуть бути текст, зображення, документи або будь-який запис даних. Наприклад, example.com.

**HTTP Verb**: повідомляє серверу, що ми хочемо зробити з ресурсом кінцевої точки URL-адреси. Наприклад, запит POST означає, що ми хочемо створити новий шаблон опитування, а запит GET означає, що ми хочемо переглянути існуючий шаблон опитування.

**Body**: необов’язкове настроюване корисне навантаження, яке містить повідомлення з властивостями та значеннями, які ми хочемо використовувати для створення або оновлення певного ресурсу.

*3)* *Які HTTP методи зазвичай використовуються в REST веб-API і для*

*чого кожен з них?*

1. Get – отримати дані з ресурсу або колекції ресурсів.

2. Post – створити новий ресурс. Для цього клієнту потрібно надіслати дані, які будуть використовуватись для створення нового об’єкту.

3. Put – оновити існуючий ресурс або створити його, якщо такого не існує.

4. Patch – частково оновити існуючий ресурс. Надсилаються тільки дані, які клієнт хоче оновити.

5. Delete – видалити ресурс або колекцію ресурсів.

6. Options – отримати інформацію про можливість використання методів або характеристик ресурсу без їх виклику.

7. Head – повертає заголовки без тіла відповіді (body). Використовується для отримання метаданих без фактичного завантаження ресурсу.

*4)* *Що таке C4 model і які чотири рівні архітектури вона включає?*

C4 model (або "Context, Containers, Components, Code") - це архітектурна модель, яка служить інструментом для визначення та опису архітектури програмного забезпечення. Включає в себе чотири рівні:

1. **Context** - показується високорівнева картина системи та її взаємодії з зовнішнім середовищем. Він визначає головні зацікавлені сторони (стейкхолдери) та інші системи, які взаємодіють з розглядуваною системою.

2. **Containers** - концентрується на визначенні великих контейнерів, або додатків, які складають систему. Наприклад, це може бути веб-додаток, мобільний додаток, база даних тощо. Контейнери взаємодіють між собою та із зовнішніми системами.

3. **Components -** описується внутрішня структура кожного контейнера. Компоненти представляють собою класи або модулі програмного коду та показують, як вони взаємодіють між собою.

4. **Code -** відображається рівень програмного коду, включаючи класи, модулі, функції тощо.

*5)* *Як методологія C4 може сприяти проектуванню та документуванню*

*архітектури системи?*

1. **Зрозуміла візуалізація**: C4 надає чітку візуалізацію архітектурних елементів на різних рівнях. Це дозволяє команді легше розуміти взаємозв'язки та внутрішню структуру системи.

2. **Є керованою концепціями**: Кожен рівень C4 відображає певний аспект архітектури, що допомагає в ієрархічному поданні концепцій. Це сприяє систематизації та структуруванню інформації про архітектуру.

3. **Легкість розуміння для стейкхолдерів**: Картини, створені за допомогою C4, зазвичай прості та доступні для розуміння не лише розробникам, а й іншим зацікавленим сторонам (стейкхолдерам). Це дозволяє краще взаєморозумітись між всіма учасниками проекту.

4. **Документація, яка росте з проектом**: Починаючи з високорівневого контексту та переходячи до деталей коду, C4 дозволяє створювати документацію, яка може рости разом із розвитком проекту. При внесенні змін у систему легко оновлювати візуалізації та документацію на різних рівнях.

5. **Підтримка різних аудиторій**: C4 розрізняє різні рівні абстракції, що дозволяє представляти інформацію у вигляді, зрозумілому різним аудиторіям, таким як розробники, архітектори, менеджери та інші стейкхолдери.

*6)* *Яке призначення ER-діаграми і як вона може допомогти в проектуванні структури бази даних?*

ER-діаграма є інструментом для моделювання структури бази даних. Серед переваг:

1. **Моделювання відносин між сутностями**: ER-діаграма дозволяє визначити сутності, які важливі для системи, і відобразити взаємозв'язки між ними. Це включає відносини "один до одного", "один до багатьох" і "багато до багатьох". Моделювання цих відносин допомагає зрозуміти, як дані взаємодіють одне з одним.

2. **Визначення атрибутів сутностей**: ER-діаграма дозволяє вказати атрибути, що визначають кожну сутність. Атрибути - це властивості або характеристики сутності, які зберігають інформацію. Моделювання атрибутів допомагає визначити, які дані мають бути збережені для кожної сутності.

3. **Визначення первинних та зовнішніх ключів**: ER-діаграма допомагає визначити первинні ключі (уникальні ідентифікатори сутностей) та зовнішні ключі (зв'язки між таблицями). Це важливо для забезпечення цілісності даних та визначення, як вони пов'язані між собою.

4. **Оптимізація структури бази даних**: Розробка ER-діаграми може допомогти виявити можливість оптимізації структури бази даних. Наприклад, можна виявити повторюваність даних, яка може бути усунута за допомогою нормалізації.

5. **Основа для фізичної реалізації**: ER-діаграма служить основою для розробки фізичної схеми бази даних. Вона надає основні структури даних та їх зв'язки, що використовуються при створенні конкретних таблиць та визначенні обмежень.

*7)* *Яким чином ви б підійшли до створення діаграми C4 для вашого*

*конкретного проекту?*

Перед створенням проєкту, я б створив контекстну діаграму для стейкхолдерів і узгодив її з усіма людьми, пов’язаними до проєкту. Потім я створив би діаграму контейнерів, яку узгодив би з усіма технічними розробниками на проєкті. І лише після цього зайнявся б створенням діаграми компонентів і коду.

*8)* *Як ви можете організувати версіонування в вашому REST веб-API?*

Версію можна включити в URL, заголовок запиту, параметр запиту або піддомен.

*9)* *Яке значення "stateless" у контексті REST та як це впливає на дизайн*

*API?*

У контексті REST (Representational State Transfer), "stateless" означає відсутність збереження стану на сервері між запитами від клієнта. Кожен запит включає всю необхідну інформацію для його обробки, і сервер не зберігає жодної інформації про стан клієнта між запитами. Це спрощує архітектуру системи та полегшує масштабованість.

*10) Чи можете ви пояснити принципи REST, такі як Uniform Interface та*

*Layered System?*

Принцип **Uniform Interface** передбачає створення уніфікованого інтерфейсу для взаємодії між компонентами системи. Цей принцип складається з чотирьох підпринципів:

1. **Identification of Resources** (Ідентифікація Ресурсів):

Кожен ресурс в системі повинен бути однозначно ідентифікований за допомогою URI (Uniform Resource Identifier).

2. **Manipulation of Resources through Representations** (Опрацювання Ресурсів через Представлення):

Клієнт взаємодіє з ресурсами через представлення ресурсу. Наприклад, можливість використовувати різні медіа-типи для представлення ресурсу (наприклад, JSON або XML).

3. **Self-descriptive Messages** (Самоописувані Повідомлення):

Кожен запит повинен містити достатньо інформації для сервера для обробки запиту. Самоописуваність повідомлень сприяє незалежності клієнта та сервера.

4. **Hypermedia as the Engine of Application State** (Гіпермедіа як Механізм Стану Додатка):

Клієнт отримує всю інформацію про стан додатка через гіпермедіа, яке включено до представлення ресурсу. Такий підхід забезпечує гнучкість та незалежність клієнта від сервера.

**Layered System**:

Принцип Layered System передбачає розподілення системи на підсистеми (шари), де кожен шар (компонент) взаємодіє лише з безпосередніми сусідами, не знаючи нічого про те, що відбувається в інших шарах. Це сприяє модульності та розширюваності системи.

Кожен шар може бути розглянутий як окремий компонент, який виконує певні функції. Це полегшує зміну або заміну одного шару без впливу на інші, що підвищує гнучкість системи.

*11) Яке значення HATEOAS в архітектурі REST?*

HATEOAS (Hypermedia As The Engine Of Application State) - це принцип архітектури REST, який визначає, що гіпермедіа (гіпертекст та гіперпосилання) повинні бути використані для визначення стану додатка та керування ним. Принцип HATEOAS робить систему більш самоописаною та становить один із ключових елементів утримання стану між клієнтом і сервером.

*12) Як можна застосувати паттерн Specification в контексті проектування*

*REST веб-API?*

Паттерн "Specification" — це структурний паттерн проектування, який визначає набір правил для відфільтрування об'єктів. У контексті проектування REST веб-API використання паттерну Specification може бути спрямоване на визначення критеріїв пошуку та фільтрації ресурсів.

*13) Які засади RESTful дизайну можуть бути порушені без значних*

*наслідків, та які є критичними?*

**Можна порушити без значних наслідків:**

*1. Деталі Реалізації:* Деякі деталі реалізації можуть бути порушені, наприклад, формат даних у відповіді (JSON або XML). Однак це може бути технічно обґрунтованим у випадках, коли додаток використовує конкретний формат.

*2. Засіб Автентифікації:* Можливість вибору засобу автентифікації може бути порушено у випадках, коли система має специфічні вимоги щодо безпеки.

**Критичні для RESTful дизайну:**

*1. Ідентифікація Ресурсів через URI:* Унікальна ідентифікація ресурсів через URI є критичною для REST. Зміни в ідентифікації ресурсів можуть порушити принцип "Uniform Interface".

*2. Використання HTTP Методів*: Використання правильних HTTP методів (GET, POST, PUT, DELETE) для взаємодії з ресурсами є важливим. Порушення цього принципу може призвести до неоднозначності та неочікуваних побічних ефектів.

*3. Stateless (Відсутність Стану):* Відсутність збереження стану на сервері між запитами важлива для прозорості та простоти взаємодії клієнта і сервера.

*4. Повернення Представлення Ресурсу:* Клієнт повинен отримувати представлення ресурсу, яке може включати гіпермедіа-елементи для подальшої навігації. Порушення цього принципу може призвести до неефективного взаємодії.

*14) Які інструменти ви можете використовувати для створення та*

*управління ER-діаграмами?*

Draw.io, dbdiagram, Lucidchart.

*15) Чому важливо враховувати потреби різних груп зацікавлених сторін*

*(stakeholders) під час створення діаграм архітектури методом C4?*

**1. Забезпечення зрозумілості**: Різні групи зацікавлених сторін можуть мати різний рівень технічної експертизи. Врахування їхніх потреб у діаграмах допомагає забезпечити зрозумілість для всіх рівнів зацікавлених сторін і сприяє ефективному спілкуванню між різними групами.

**2. Визначення вимог**: Різні групи мають різні вимоги і очікування від системи. Врахування цих вимог на ранніх етапах моделювання дозволяє точніше визначити функціональні та нефункціональні вимоги системи.

**3. Сприяння прийняттю рішень**: Врахування потреб різних груп допомагає приймати обгрунтовані рішення з точки зору всіх зацікавлених сторін. Це допомагає уникнути конфліктів та забезпечити те, що архітектура системи відповідає вимогам всіх стейкхолдерів.

**4. Ефективна комунікація**: Моделі архітектури є засобом комунікації між розробниками, менеджерами, клієнтами та іншими стейкхолдерами. Якщо діаграми архітектури враховують потреби різних груп, це полегшує ефективну комунікацію та взаєморозуміння.

**5. Мінімізація ризиків**: Врахування потреб всіх зацікавлених сторін допомагає виявляти можливі ризики та протиріччя на ранніх етапах проекту. Це дозволяє приймати запобіжні заходи та зменшує ризик непорозумінь або неправильних вирішень.