МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Лабораторна робота №3

з дисципліни «Розробка програмного забезпечення на платформі .Net»

на тему:

«Проектування REST веб-API»

Виконала:

Студентка групи ІС-12

Гусєва Тетяна

Викладач:
Бардін В.

Київ – 2023

Завдання:

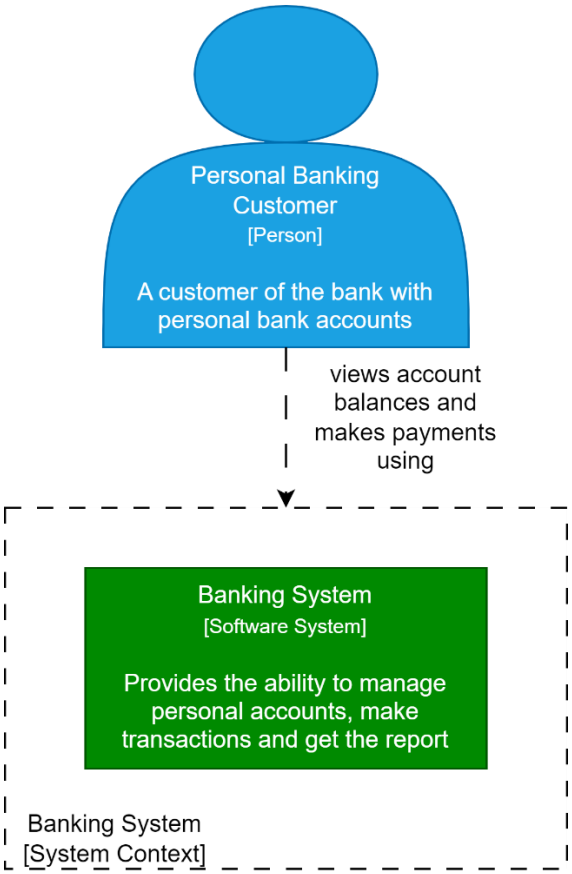
1. З дотриманням вимог REST-у спроектувати веб-API для обраної(згідно варіанту) доменної області, використовуючи методологію C4 для створення діаграми архітектури системи.
2. Створити ER-діаграму для DAL (Data Access Layer), яка відображатиме структуру бази даних веб-API.

Варіант:

4	Гаманець. Керування власним бюджетом та фінансами	1. Власний бюджет складається з декількох рахунків, які поповнюються за заданими статтями прибутку. 2. Гроші цих рахунків можуть бути переведені з одного на інший, можуть витрачатись за заданими статтями витрат. 3. Підсумовуючи витрати та прибутки, можливо отримати інформацію, скільки було витрачено/отримано загалом/за певною статтею по заданому рахунку. **Функціональні вимоги:** 1. Ведення власного бюджету; 2. Отримання звітної інформації по рахунках.

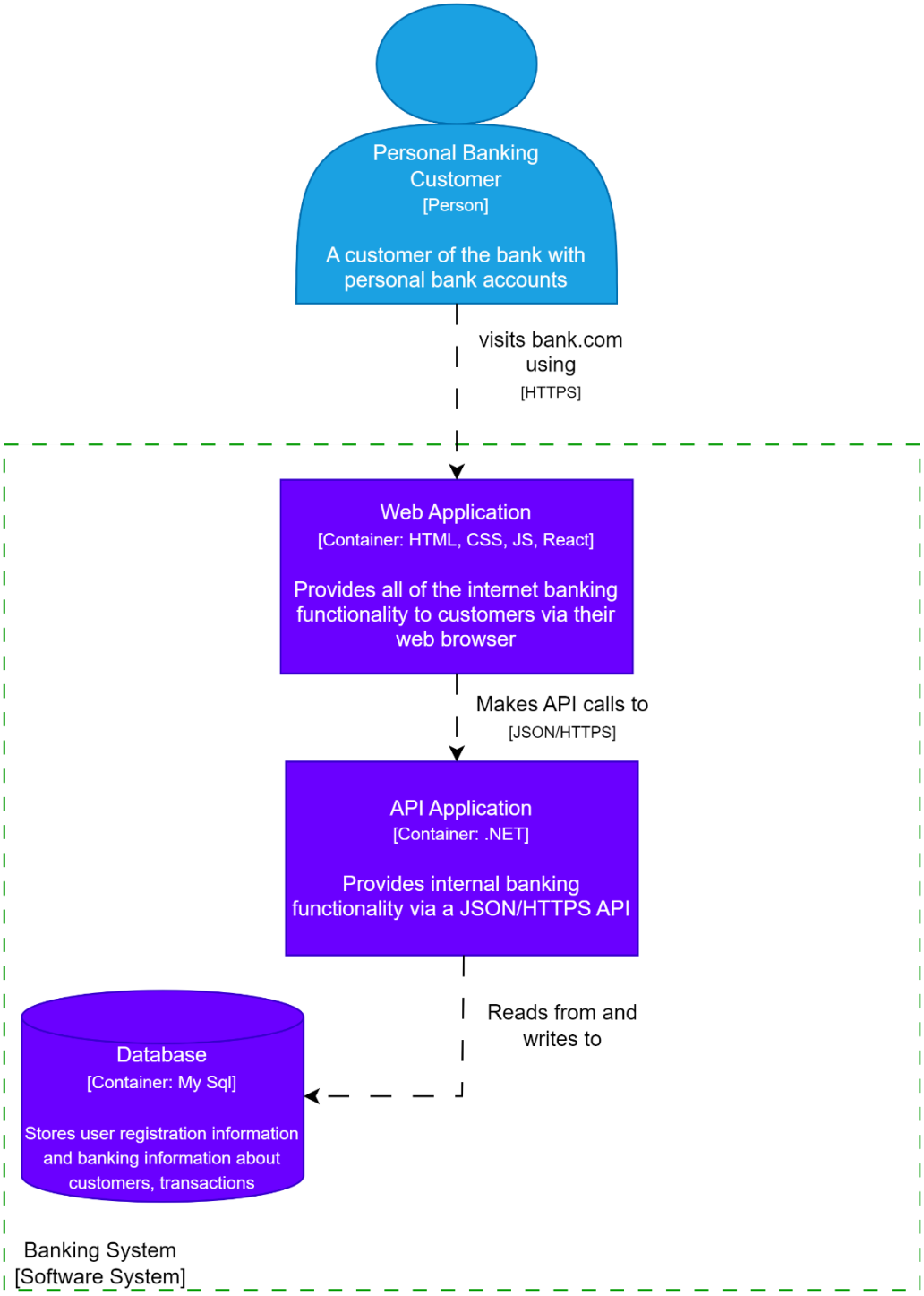
1. З дотриманням вимог REST-у спроектувати веб-API для обраної(згідно варіанту) доменної області, використовуючи методологію C4 для створення діаграми архітектури системи.

Level 1. System Context.



Визначення акторів: User Користувач, який має змогу керувати власним бюджетом та фінансами в системі гаманець.

Level 2. Containers.



Система керування бюджету складається з трьох основних контейнерів:

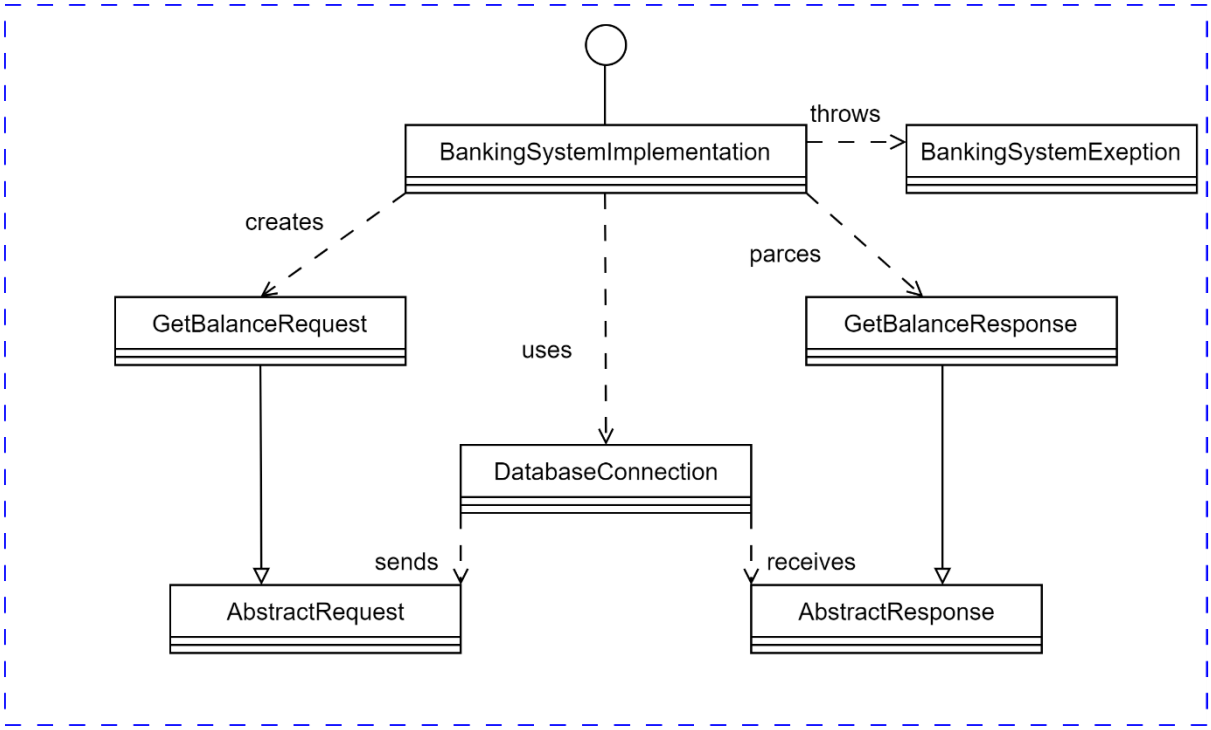
- Веб-застосунок, який надає користувачу функціонал для керування свого бюджету через веб-браузер.
- Серверна програма, яка реалізує функціонал REST API та приймає запити від браузера.
- База даних, яка зберігає моделі основних сутностей, які задіяні у програмі.

Level 3. Components.

Для розробки застосунку використовується N-layer архітектура, яка поділяє застосунок на 3 шари:

- User Interface Layer (Шар представлення): Цей рівень відповідає за взаємодію з користувачем та представлення інформації. Він містить графічний інтерфейс та обробку введення. Тут знаходяться контролери.
- Application Layer (Шар бізнес-логіки): Цей рівень містить бізнес-логіку програми, яка визначає логіку операцій та обробку даних. Містить інтерфейси сервісів та їхню імплементацію.
- DAL (Шар доступу до даних): Цей рівень відповідає за доступ до даних та взаємодію з базою даних.

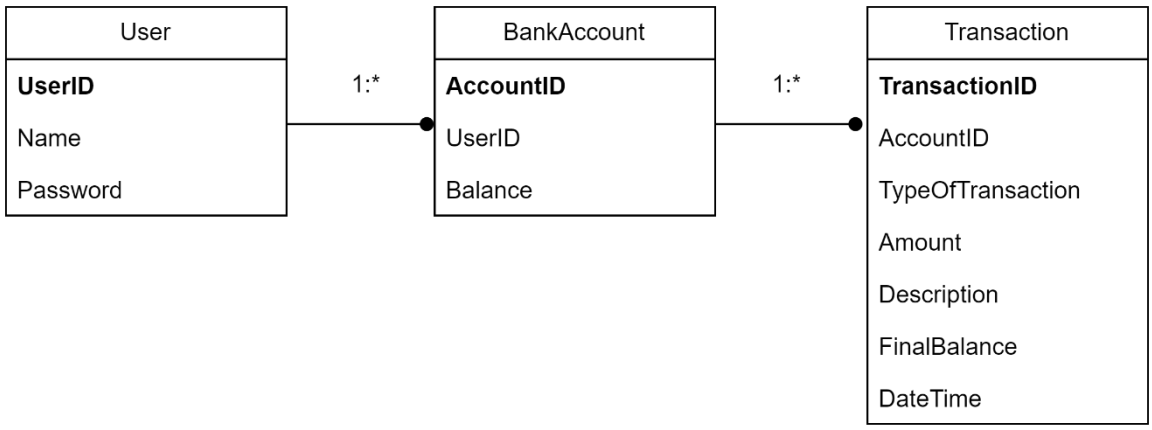
Level 4. Code.



В залежності від запиту користувача, будуть використані певні методи:

- POST: створення нового рахунку, переказ коштів.
- GET: отримання звітності по певному рахунку (доходи, витрати), отримання всіх рахунків.
- PUT: оновлення рахунку.
- DELETE: видалення рахунку.

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



Дана ER-діаграма представляє тип зв’язку між сутностями:

- 1 : many – User : BankAccount, BankAccount : Transaction.

Таблиця User:

- UserPhoneNumber – Number unique key.
- Name – VarChar.
- Password – VarChar.

Таблиця BankAccount:

- AccountID – Guid.
- UserPhoneNumber – Number unique key.
- Balance – Decimal.

Таблиця Transaction**:**

- TransactionID – Guid.
- AccountID – Guid.
- TypeOfTransaction – VarChar.
- FinalBalance – Decimal.
- DateTime – DateTime.
- Amount – Decimal.
- Description – VarChar.

Перелік ендпоінтів

1. **Додавання рахунку:**
    - Метод: POST
    - URL: /api/account
    - Параметри запиту: Новий об'єкт рахунку (назва рахунку, баланс тощо)
    - Опис: Створення нового рахунку для користувача.
2. **Видалення рахунку:**
    - Метод: DELETE
    - URL: /api/account/{id}
    - Параметри запиту: Ідентифікатор рахунку
    - Опис: Видалення рахунку користувача.
3. **Зміна даних рахунку:**
    - Метод: PUT
    - URL: /api/account/{id}
    - Параметри запиту: дані рахунку
    - Опис: Оновлення даних рахунку(назва, баланс).
4. **Отримати рахунок за Id:**
    - Метод: GET
    - URL: /api/account
    - Параметри запиту: Ідентифікатор рахунку
    - Опис: Отримання рахунку за Id.
5. **Отримати всі рахунки юзера:**
    - Метод: GET
    - URL: /api/account/getAll
    - Параметри запиту: Ідентифікатор юзера
    - Опис: Отримання всі рахунків юзера.

6. **Додавання транзакції:**
    - Метод: POST
    - URL: /api/transaction
    - Параметри запиту: Новий об'єкт транзакції (тип транзакції, сума, опис, рахунок тощо)
    - Опис: Додавання нової транзакції до обраного рахунку.
7. **Видалення транзакції:**
    - Метод: DELETE
    - URL: /api/transaction
    - Параметри запиту: Ідентифікатор транзакції
    - Опис: Видалення транзакції
8. **Зміна даних транзакції:**

- Метод: PUT
- URL: /api/transaction
- Параметри запиту: об'єкт
- Опис: Оновлення даних транзакції

9. **Переказ коштів між рахунками:**
   - Метод: POST
   - URL: /api/transaction/transfer
   - Параметри запиту: рахунок відправник, рахунок отримувач та сума
   - Опис: Переказ коштів з одного рахунку на інший
10. **Отримання інформації про всі транзакції за рахунком:**
    - Метод: GET
    - URL: /api/transaction?{id}
    - Параметри запиту: Ідентифікатор рахунку
    - Опис: Отримання інформації про транзакції.
11. **Отримання інформації про транзакції за рахунком враховуючи тип та тег:**
    - Метод: GET
    - URL: /api/transaction?{id}&{type}&{tag}
    - Параметри запиту: Ідентифікатор рахунку
    - Опис: Отримання інформації про транзакції.

12. **Реєстрація користувача**
    - Метод: POST
    - URL: /api/auth/register
    - Параметри запиту: дані користувача
    - Опис: Реєстрація нового юзера
13. **Логін користувача**
    - Метод: POST
    - URL: /api/auth/login
    - Параметри запиту: дані користувача
    - Опис: Вхід до свого акаунту

Відповіді на питання

1. Що таке REST веб-API та які його основні принципи?

REST (Representational State Transfer) є стиль архітектури для розробки веб-сервісів, що базується на стандартах протоколу HTTP.

Основні принципи REST включають:

Ресурси (Resources): У REST архітектурі, всі дані або служби ідентифікуються як ресурси. Ресурси можуть бути будь-чим - об'єктами, послугами, чи будь-чим іншим.

Представлення (Representation): Ресурси можуть мати різні представлення, такі як HTML, JSON, XML тощо. Клієнт вибирає, яке представлення найбільше підходить для нього.

Стан (Stateless): Кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для зрозуміння та обробки запиту. Сервер не повинен зберігати жодного стану між запитами від одного клієнта до іншого.

Операції з ресурсами через стандартні методи HTTP:

GET: Отримання ресурсу або його представлення.

POST: Створення нового ресурсу.

PUT: Оновлення існуючого ресурсу або створення нового, якщо він не існує.

DELETE: Видалення ресурсу.

Також можуть використовуватися інші HTTP-методи, такі як PATCH, OPTIONS, HEAD, тощо.

URI (Uniform Resource Identifier): Кожен ресурс ідентифікується унікальним URI, що використовується для доступу до ресурсу. URI повинні бути незмінними та стабільними.

Повідомлення (Stateless Communication): Комунікація між клієнтом і сервером повинна бути безстанційною. Кожен запит клієнта містить всю інформацію, необхідну для зрозуміння та обробки запиту, а сервер не повинен зберігати стан між запитами від одного клієнта до іншого.

Гіпермедіа як дієве засіб (HATEOAS): Сервер повинен надсилати клієнту всю необхідну інформацію про доступні дії, щоб клієнт міг переходити між ресурсами, не залежно знанням конкретних URL.

2. Які основні компоненти включаються в архітектуру REST API?

Ресурси (Resources):

Опис: Ресурси представляють собою концепції або об'єкти, з якими взаємодіє ваш API. Наприклад, користувачі, замовлення, продукти, тощо.

Приклад URI: /users, /orders, /products.

URI (Uniform Resource Identifier):

Опис: Це адреса, яка ідентифікує ресурс в мережі. У випадку REST API, URI використовується для доступу до ресурсів.

Приклад URI: /api/v1/users/123.

HTTP Методи (HTTP Methods):

Опис: Використовуються для вказівки, яку операцію потрібно виконати з ресурсом. Основні методи включають GET, POST, PUT, DELETE, PATCH та інші.

Приклади:

GET - для отримання ресурсу чи його списку.

POST - для створення нового ресурсу.

PUT - для оновлення існуючого ресурсу.

DELETE - для видалення ресурсу.

Представлення (Representation):

Опис: Визначає формат, в якому представлення ресурсу надсилається або отримується з боку клієнта. Зазвичай використовуються формати JSON або XML.

Статус коди HTTP (HTTP Status Codes):

Опис: Використовуються для повідомлення про стан виконання запиту. Наприклад, 200 OK, 201 Created, 404 Not Found, тощо.

Гіпермедіа як дієве засіб (HATEOAS):

Опис: Включає в себе відправлення клієнту всієї необхідної інформації про доступні дії для навігації між ресурсами без предварительного знання конкретних URL.

Запити та Відповіді (Requests and Responses):

Опис: Взаємодія між клієнтом і сервером здійснюється за допомогою структурованих запитів та відповідей, які використовують HTTP протокол.

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

чого кожен з них?

GET:

Призначення: Використовується для отримання інформації про ресурс чи його представлення.

Приклад: Отримання інформації про користувача, список продуктів.

POST:

Призначення: Використовується для створення нового ресурсу або виконання певної дії.

Приклад: Створення нового користувача, відправлення даних форми.

PUT:

Призначення: Використовується для оновлення існуючого ресурсу або створення нового, якщо він не існує.

Приклад: Оновлення інформації про користувача, внесення змін у вже існуючий ресурс.

DELETE:

Призначення: Використовується для видалення ресурсу.

Приклад: Видалення користувача, видалення запису з бази даних.

PATCH:

Призначення: Використовується для часткового оновлення ресурсу, коли клієнт відправляє тільки ті дані, які потрібно змінити.

Приклад: Оновлення тільки певних полів інформації про користувача, а не всієї інформації.

OPTIONS:

Призначення: Використовується для отримання інформації про можливі методи та параметри для ресурсу (CORS - Cross-Origin Resource Sharing).

Приклад: Перевірка, які методи із заголовком Access-Control-Allow-Methods підтримуються сервером для конкретного ресурсу.

HEAD:

Призначення: Аналогічний методу GET, але сервер повертає лише заголовки відповіді без самої інформації про ресурс.

Приклад: Використовується для перевірки наявності ресурсу та отримання інформації про його характеристики без отримання вмісту.

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

C4 model (C4 stands for "Context, Containers, Components, Code") — це легкий фреймворк для моделювання архітектури програмного забезпечення. Він надає зручний спосіб відображення та спілкування архітектурних концепцій у великих та складних проектах. Context, Containers, Components, Code.

5. Як методологія C4 може сприяти проектуванню та документуванню архітектури системи?

Просто та зрозуміло, спрощує документацію, фокус на ключових елементах, підтримує принципи REST-у.

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

ER-діаграма (сутнісно-реляційна діаграма) є графічним інструментом для моделювання структури бази даних, визначаючи сутності (об'єкти чи концепції) та їх взаємозв'язки. Вона допомагає в проектуванні баз даних, надаючи візуальне представлення структури даних та зв'язків між ними.

7. Яким чином ви б підійшли до створення діаграми C4 для вашого конкретного проекту?

Визначити основних акторів, описати основні ф-кції, деталізація гаманця.

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

Включення версії в URI:

https://api.wallet.example.com/v1/budget

https://api.wallet.example.com/v2/budget

Використання Заголовка "Accept":

Accept: application/vnd.wallet.v1+json

Використання Заголовка "X-API-Version":

X-API-Version: 1

9. Яке значення "stateless" у контексті REST та як це впливає на дизайн API?

"Stateless" (безстанційний) у контексті REST означає відсутність збереження стану на сервері між послідовними запитаннями від клієнта.

Незалежність запитів, зменшення навантаження на сервер, спрощення кешування.

10. Чи можете ви пояснити принципи REST, такі як Uniform Interface та Layered System?

Принцип Уніфікованого Інтерфейсу передбачає стандартний та уніфікований спосіб взаємодії між клієнтом та сервером у REST-системі. Цей принцип розбиває систему на чотири основні складові:

Identification of Resources (Ідентифікація Ресурсів): Кожен ресурс (наприклад, URL) повинен бути унікально ідентифікований. Клієнт ідентифікує ресурс за його URI.

Manipulation of Resources Through Representations (Опрацювання Ресурсів через Представлення): Клієнт взаємодіє з ресурсами за допомогою їхніх представлень, які можуть бути HTML, XML, JSON тощо.

Self-Descriptive Messages (Самоописовні Повідомлення): Кожне повідомлення містить всю необхідну інформацію для обробки запиту. Відповіді сервера повинні містити достатньо інформації, щоб клієнт міг зрозуміти відповідь та подальші можливі дії.

Hypermedia as the Engine of Application State (Гіпермедіа як Рушій Стану Додатку): Клієнти отримують всю необхідну інформацію для переходу між станами додатку через вбудовані гіперпосилання в представленнях ресурсів.

Принцип Шарової Системи передбачає, що архітектура може бути побудована у вигляді низки шарів, де кожен шар виконує конкретні функції та надає абстракції для шарів, що знаходяться нижче. Це полегшує розподілений характер системи та дозволяє розділити функціональність на логічні рівні.

11. Яке значення HATEOAS в архітектурі REST?

HATEOAS (Hypermedia As The Engine Of Application State) - це принцип архітектури REST, який вказує, що гіпермедіа (гіпертекстове представлення додатку) повинно бути основним засобом навігації в додатку та визначенням його стану. Принцип HATEOAS виникає з ідеї, що клієнт повинен бути повністю залежним від представлення ресурсів та вміти взаємодіяти з системою, використовуючи лише гіпермедіа, яке він отримав від сервера.

12. Як можна застосувати паттерн Specification в контексті проектування REST веб-API?

Паттерн Specification - це поведінковий паттерн проектування, який дозволяє визначити набір правил або умов, які можуть бути скомбіновані для визначення більш складних умов в системі. В контексті проектування REST веб-API паттерн Specification може бути використаний для створення складних логічних умов для фільтрації, сортування та вибірки ресурсів.

13. Які засади RESTful дизайну можуть бути порушені без значних наслідків, та які є критичними?

Можна порушити без значних наслідків:

Сервіси відсутність стану (Statelessness):

В окремих випадках, де збереження стану може виявитися корисним (наприклад, в системах, де необхідно виконати багато послідовних запитів, і взаємодія становить суттєву частину ділової логіки), можна вирішити, що невелика кількість стану допустима для збереження на клієнті або сервері.

Ідентифікація Ресурсів (Resource Identification):

У деяких випадках, наприклад, при проектуванні внутрішніх служб, ідентифікатор ресурсу може бути частково вкладеним в URL, а не повністю відокремленим від URL.

Критичні для збереження принципів RESTful:

Сервіси відсутність стану (Statelessness):

Без цього принципу важко досягти масштабованості, надійності та ефективності. Сервіси, які залежать від стану, можуть призвести до складних сценаріїв керування станом та збільшити навантаження на сервер.

Ідентифікація Ресурсів (Resource Identification):

Важливо, щоб ідентифікатори ресурсів були сталі та відокремлені від URL. Це забезпечує гнучкість та стабільність API, оскільки URL може змінюватися, але ідентифікатор ресурсу залишається сталим.

Уніфікований Інтерфейс (Uniform Interface):

Збереження цього принципу дозволяє розділити архітектуру, зробити систему більш гнучкою та спростити взаємодію між клієнтом та сервером через уніфікований інтерфейс.

14. Які інструменти ви можете використовувати для створення та управління ER-діаграмами?

Draw.io, Lucidchart, MIcrosoftVisio.

15. Чому важливо враховувати потреби різних груп зацікавлених сторін (stakeholders) під час створення діаграм архітектури методом C4?

Розуміння Вимог та Очікувань:

Різні групи зацікавлених сторін можуть мати різні вимоги та очікування від системи. Врахування цих різних потреб дозволяє точніше визначити вимоги до системи та забезпечити, що проект архітектури відповідає цим потребам.

Забезпечення Гнучкості та Підтримки:

Різні зацікавлені сторони можуть вносити важливі внески в процес проектування та використовувати систему для різних цілей.

Забезпечення Повноцінного Розгортання:

Різні групи зацікавлених сторін можуть взаємодіяти з системою на різних етапах її життєвого циклу.