МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

Розробка Web API для розподіленого веб сервісу

Виконав:
студент групи ПМІ-44
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Кулиняк В.М
(підпис) (прізвище та ініціали)

Керівник Горлач В.М
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Львів– 2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики _____
Кафедра Інформаційних систем _____
Спеціальність 122 «Комп'ютерні науки»_____ __
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри проф Шинкаренко Г.А

"7 _ "вересня___2023 року

З А В Д А Н Н Я

НА ДИПЛОМНУ У РОБОТУ СТУДЕНТУ

___Кулиняк Василь Миколайович_____
(прізвище, ім'я, по батькові)

1 .Тема роботи
Розробка Web API для розподіленого веб сервісу

керівник роботи Горлач Віталій Михайлович _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **"13" вересня 2022 р. № 15**

2. Строк подання студентом роботи **13.06.2023р.**_____

3. Вихідні дані до роботи

1. Розроблена Web Api

2. Використана архітектура RESTfull

3.Клієнт серверна архітектура

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

Сформулювати вимоги до веб застосунку, використати оптимальний стек технологій для реалізації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
	Огляд існуючих систем та технологій	*вересень 2022*	
	Постановка задач	*жовтень 2022*	
	Розробка схем та алгоритмів	*листопад 2022*	
	Програмна реалізація	*грудень-березень 2023*	
	Апробація	*квітень 2023*	
	Оформлення роботи	*травень 2023*	

Студент _____
(підпис)

Керівник роботи _____
(підпис)

ЗМІСТ

ВСТУП

У сучасному взаємопов’язаному цифровому ландшафті безперервний обмін даними та функціями між програмами та системами став першочерговим. Веб-API (інтерфейси прикладного програмування) стали основоположною технологією, яка полегшує цей зв’язок, дозволяючи програмам взаємодіяти та інтегруватися між платформами, службами та організаціями. Універсальність, гнучкість і масштабованість веб-інтерфейсів API зробили революцію в розробці програмного забезпечення, забезпечивши ефективний пошук даних, маніпуляції та створення сервісів.

Ця дипломна робота має на меті заглибитися у світ веб-API, досліджуючи їхнє значення, базові принципи та практичну реалізацію. Надаючи всебічний посібник із вивчення та впровадження, ця робота озброїть читачів знаннями та навичками, необхідними для ефективного використання можливостей веб-API.

Цілі.Основними завданнями даної дипломної роботи є:

Розуміння концепції веб-інтерфейсів API: робота розпочнеться з надання чіткого та лаконічного визначення веб-інтерфейсів API, пояснення їх призначення та підкреслення їхньої ролі у забезпеченні зв’язку та інтеграції між програмами. Він досліджуватиме ключові принципи та характеристики веб-API, відрізняючи їх від інших форм програмних інтерфейсів.

Вивчення архітектурних стилів API: у роботі буде розглянуто різні архітектурні стилі API, з особливим акцентом на RESTful API та GraphQL. У ньому обговорюватимуться сильні та слабкі сторони та відповідні випадки використання кожного стилю, що дозволить читачам приймати зважені рішення під час вибору найбільш прийнятного підходу для своїх проектів.

Дослідження дизайну веб-API та найкращих практик: у цій дипломній роботі розглядатимуться основні аспекти розробки надійних і зручних веб-інтерфейсів API. Він охоплюватиме такі теми, як моделювання ресурсів, формати запитів/відповідей, обробка помилок, керування версіями та питання безпеки. У роботі також буде розглянуто найкращі практики в галузі та нові стандарти, щоб допомогти читачам у розробці добре структурованих і підтримуваних API.

Впровадження та інтеграція: робота надасть практичну інформацію про впровадження веб-API, охоплюючи ключові технології, фреймворки та інструменти, які зазвичай використовуються в розробці API. Він проведе читачів через покроковий процес створення RESTful API , інтеграції механізмів автентифікації та авторизації та проведення ретельного тестування для забезпечення надійності та ефективності API.
Реальні випадки використання та тематичні дослідження: щоб проілюструвати практичне застосування веб-API, у цій дипломній роботі будуть представлені реальні випадки використання та тематичні дослідження з різних галузей промисловості. Він демонструватиме успішні інтеграції API, підкреслюючи переваги та проблеми, які виникають у кожному сценарії. Досліджуючи ці приклади, читачі отримають цінну інформацію про різноманітні можливості та потенційні підводні камені впровадження веб-API.

На завершення, ця дипломна робота служить вичерпним посібником для розуміння, проектування та впровадження веб-API. Вивчаючи фундаментальні концепції, архітектурні стилі, принципи проектування та методи практичної реалізації, читачі отримають необхідні знання та навички для ефективного використання веб-інтерфейсів API у своїх майбутніх проектах. Дослідження випадків використання в реальному світі та тематичних досліджень ще більше підвищує практичність і актуальність роботи. У зв’язку зі зростаючою значущістю веб-інтерфейсів API у сучасній розробці програмного забезпечення, ця дипломна робота надає читачам необхідний досвід для навігації у мінливому ландшафті взаємопов’язаних систем і програм.

ТЕХНІЧНЕ ЗАВДАННЯ

Завдання– це створити програмну реалізацію веб-API, розібратись в принципах його роботи на функціювання. Розробити архітектуру саме RESTfull, показати її переваги та чому вона користується такою популярністю.

Виконані завдання для поставленої мети

Визначте призначення та цілі API.

Визначте цільову аудиторію та її потреби.

Визначте функціональні можливості та функції, які має надавати API.

Визначте формати даних і протоколи, які підтримуватиме API (наприклад, JSON, XML, REST, SOAP).Кінцеві точки API дизайну:

Визначте кінцеві точки, які надаватиме API.

Визначте структуру URL-адреси та правила іменування кінцевих точок.

Укажіть методи HTTP (GET, POST, PUT, DELETE тощо) та їхні відповідні дії для кожної кінцевої точки.

Спроектуйте структури даних запиту та відповіді.

Виберіть мову програмування та фреймворк:

Виберіть мову програмування, яка підходить для створення API.

Виберіть структуру або бібліотеку, яка спрощує розробку API і виконує типові завдання (наприклад, маршрутизація, серіалізація, автентифікація).

Впровадити кінцеві точки API:

Напишіть код для обробки кінцевих точок API відповідно до специфікацій дизайну.

Впровадити необхідну бізнес-логіку для обробки вхідних запитів і створення відповідних відповідей.

Перевіряйте та дезінфікуйте введені користувачем дані, щоб забезпечити цілісність і безпеку даних.

Обробляти умови помилок і надавати змістовні повідомлення про помилки.

Реалізація збереження даних:

Визначте, як і де будуть зберігатися дані (наприклад, бази даних, файлові системи).

Налаштуйте підключення до бази даних або будь-якого іншого джерела даних.

Реалізуйте логіку доступу до даних для отримання та зберігання даних відповідно до вимог API.

Реалізуйте логіку авторизації, щоб контролювати доступ до певних кінцевих точок API або ресурсів на основі ролей користувачів або дозволів.

Запровадити перевірку та обробку помилок:

Перевірте вхідні запити, щоб переконатися, що вони відповідають необхідним критеріям.

Обробляти помилки перевірки та повертати відповідні відповіді на помилки.

Застосуйте належну обробку помилок і керування винятками в коді API.

Практична значимість роботи

Розробка веб-API у стилі RESTful пропонує кілька практичних переваг, що робить його популярним вибором для створення API. Ось деякі практичні переваги розробки веб-API у стилі RESTful:

Простота та легкість у використанні: API RESTful дотримуються набору чітко визначених принципів і угод, що робить їх інтуїтивно зрозумілими та легкими для розуміння розробниками. Простота стилю архітектури REST спрощує розробку API, зменшує складність і дозволяє швидше адаптувати та інтегрувати в програми.

Масштабованість: RESTful API розроблені таким чином, щоб бути масштабованими, дозволяючи горизонтальне масштабування для обробки збільшеного трафіку та попиту користувачів. Використовуючи бездіяльність REST, стає легше розподіляти API між кількома серверами та незалежно масштабувати ресурси за потреби.

Сумісність і сумісність: RESTful API використовують стандартні методи HTTP (GET, POST, PUT, DELETE) і дотримуються загальноприйнятих форматів даних, таких як JSON або XML. Ця стандартизація сприяє сумісності та взаємодії, дозволяючи API використовуватися різними клієнтами, фреймворками та мовами програмування.

Гнучкість і розширюваність: RESTful API забезпечують гнучкість щодо представлення даних і формату. Вони можуть підтримувати різні типи медіа, включаючи JSON, XML і навіть спеціальні формати. Крім того, REST дозволяє додавати нові ресурси, кінцеві точки або функції, не впливаючи на існуючі компоненти, що робить його розширюваним і адаптованим до мінливих вимог.

Кешування та продуктивність: RESTful API можуть використовувати переваги механізмів кешування HTTP для підвищення продуктивності та зменшення

навантаження на сервер. Кешування відповідей на клієнті або проксі-сервері-посереднику може значно зменшити кількість запитів і обсяг даних, що передаються через мережу, що призводить до покращення продуктивності та зменшення затримки.

Архітектура без збереження стану: API RESTful дотримуються архітектури без збереження стану, тобто кожен запит від клієнта містить усю необхідну інформацію для того, щоб сервер міг її зрозуміти та обробити. Ця відсутність стану спрощує логіку на стороні сервера, покращує масштабованість і забезпечує кращу відмовостійкість і балансування навантаження.

Розділення клієнт-сервер: RESTful API сприяють чіткому розмежуванню між клієнтським і серверним компонентами. Такий поділ дозволяє незалежну розробку, підтримку та масштабованість кожного компонента. Це дає змогу різним клієнтам (веб-клієнтам, мобільним, комп'ютерним) взаємодіяти з одним API, а також потенціал для паралельної розробки різними командами.

Можливість тестування: RESTful API добре тестуються завдяки своїй простоті та дотриманню стандартних методів HTTP та кодів стану. Інструменти та фреймворки тестування можуть легко імітувати різні типи запитів і відповідей, уможливлюючи комплексне тестування для різних сценаріїв.

Екосистема та інструменти: API RESTful мають надійну екосистему з широким набором інструментів, бібліотек і фреймворків, доступних для різних мов програмування та платформ. Ці інструменти забезпечують підтримку розробки API, документування, тестування та моніторингу, полегшуючи створення, підтримку та інтеграцію RESTful API у програми.

Прийняття в промисловості: REST став стандартом де-факто для веб-API, широко прийнятим розробниками, організаціями та платформами. Таке широке впровадження означає, що існує величезна кількість ресурсів, документації та підтримки спільноти, що полегшує навчання, усунення несправностей і співпрацю над розробкою RESTful API. Загалом розробка веб-API у стилі RESTful пропонує такі практичні переваги, як простота, масштабованість, сумісність, гнучкість і продуктивність, які сприяють швидшій розробці, легшій інтеграції та покращенню взаємодії з користувачем під час використання API.

РОЗДІЛ 1

1.1. Визначення цілей API та функціональних вимог

Визначте призначення API для служби ведення блогів:

Метою API для служби блогів є надання стандартизованого інтерфейсу для взаємодії з функціями та даними платформи блогів. API дозволяє розробникам, клієнтам і зовнішнім системам програмно отримувати доступ, створювати, оновлювати та видаляти ресурси, пов'язані з блогами. Ось пояснення мети та цінності, яку API приносить користувачам або системі:

Проблема чи виклик:

API вирішує проблему ручного або повторюваного керування завданнями, пов'язаними з блогами, шляхом автоматизації взаємодії зі службою блогів. Це усуває потребу в ручному управлінні вмістом і забезпечує безперебійну інтеграцію платформи для ведення блогів з іншими системами, програмами або службами сторонніх розробників.

Цінність для користувачів або системи:

API надає кілька цінностей як користувачам, так і системі:

- Підвищена ефективність: API дозволяє користувачам виконувати завдання, пов'язані з блогами, програмно, заощаджуючи час і зусилля на управлінні вмістом блогу, метаданими та взаємодією.
- Повна інтеграція: API забезпечує інтеграцію із зовнішніми системами, такими як мобільні програми, системи керування вмістом або іншими платформами, сприяючи синхронізації даних блогу та покращуючи охоплення та видимість вмісту блогу.
- Налаштування та гнучкість: користувачі можуть використовувати API для розробки спеціальних програм, віджетів або плагінів, які покращують досвід ведення блогів, розширюють функціональність або інтегруються з іншими службами.
- Покращена співпраця: API полегшує співпрацю, дозволяючи кільком користувачам або командам одночасно працювати над завданнями, пов'язаними з блогом, наприклад створенням, редагуванням або модерацією вмісту.

Використання API та користувачі:

API можуть використовувати різні зацікавлені сторони, зокрема:

- Розробники: розробники можуть використовувати API для створення спеціальних програм, плагінів або інтеграцій, які взаємодіють із службою блогів програмним шляхом.
- Менеджери вмісту: менеджери вмісту можуть використовувати API для автоматизації публікації вмісту, керування метаданими або виконання масових операцій над публікаціями блогу.
- Послуги третіх сторін: служби третіх сторін, такі як аналітичні платформи, інструменти соціальних мереж або агрегатори вмісту, можуть інтегруватися з API для доступу та отримання даних блогу для відповідних функцій.

Основні цілі та результати:

Основні цілі API для служби ведення блогів можуть включати:

- Спрощення керування блогами: API має оптимізувати завдання, пов’язані з блогами, дозволяючи користувачам виконувати операції CRUD (створення, читання, оновлення, видалення) над публікаціями блогів, категоріями, тегами, коментарями та іншими відповідними ресурсами.
- Забезпечення інтеграції: API має забезпечувати комплексні кінцеві точки та структури даних, які дозволяють бездоганну інтеграцію із зовнішніми системами, програмами чи службами.
- Забезпечення безпеки та автентифікації: API має реалізовувати безпечні механізми автентифікації (наприклад, OAuth, ключі API), щоб гарантувати, що лише авторизовані користувачі або системи можуть отримувати доступ до ресурсів блогу та маніпулювати ними.
- Підвищення масштабованості та продуктивності: API має бути розроблено та реалізовано для ефективної обробки великого обсягу запитів і забезпечення оптимальної продуктивності, забезпечуючи плавну роботу користувача.

Очікуваними результатами розробки API є підвищення ефективності, підвищена гнучкість, розширена співпраця та кращі можливості інтеграції, що зрештою покращить загальний досвід ведення блогів і дозволить користувачам використовувати

весь потенціал служби блогів.

1.2 Визначення кінцевих точок і ресурсів API

Кінцеві точки та ресурси API є ключовими компонентами RESTful API, які визначають доступні функції та об’єкти даних, до яких можна отримати доступ і маніпулювати ними. Вони відіграють вирішальну роль у визначенні структури та поведінки API. Ось пояснення визначення кінцевих точок і ресурсів API:

Кінцева точка API:

Кінцева точка API представляє певну URL-адресу (уніфікований покажчик ресурсу), яку клієнтська програма може використовувати для взаємодії з певною функціональністю або ресурсом, наданим API. Кінцеві точки ідентифікуються своїми унікальними URL-адресами та зазвичай пов’язані з певними методами HTTP (GET, POST, PUT, DELETE) для виконання відповідних дій на ресурсі. Кожна кінцева точка представляє певну операцію або дію, яку може виконувати API.

Наприклад, розглянемо API блогів із такими кінцевими точками:

- `GET /posts`: отримує список усіх публікацій блогу.
- `POST /posts`: створює нову публікацію в блозі.
- `GET /posts/{id}`: отримує конкретну публікацію блогу за її ідентифікатором.
- `PUT /posts/{id}`: оновлює наявну публікацію блогу за її ідентифікатором.
- `DELETE /posts/{id}`: видаляє публікацію блогу за її ідентифікатором.

У цьому прикладі `/posts` є кінцевою точкою для отримання списку публікацій блогу, а `/posts/{id}` є кінцевою точкою для отримання, оновлення чи видалення певної публікації блогу, визначеної її ідентифікатором.

Ресурс API:

Ресурс API представляє логічну сутність або об’єкт даних, який надається через API. Це може бути один об’єкт або набір об’єктів, якими може керувати API. Ресурси зазвичай представлені іменниками в структурі URL-адрес API та можуть мати кілька кінцевих точок, пов’язаних з ними для виконання різних операцій.

У прикладі API для ведення блогів ресурсом є «пости», що представляє дописи блогу. Кінцеві точки, пов’язані з ресурсом «дописи», визначають, як клієнти можуть взаємодіяти з дописами блогу та маніпулювати ними. Інші приклади ресурсів API можуть включати «користувачів», «коментарі» або «категорії», залежно від функціональних можливостей і об’єктів даних, наданих API.

Кінцеві точки та ресурси API працюють разом, щоб визначити функціональність і структуру API. Кінцеві точки забезпечують конкретні операції або дії, які можна виконувати з ресурсами, тоді як ресурси представляють сутності даних або об’єкти, якими маніпулюють. Розробляючи чіткі та інтуїтивно зрозумілі кінцеві точки та ресурси, розробники API можуть надати клієнтам стандартизований і узгоджений інтерфейс для взаємодії з API та доступу до базових об’єктів даних

1.3Структура URL-адреси та правила іменування для API:

Структура URL-адреси та правила іменування для API відіграють важливу роль у створенні добре організованого, інтуїтивно зрозумілого та зручного для розробників API. Вони допомагають визначити структуру та ієрархію кінцевих точок і ресурсів API, полегшуючи клієнтам розуміння та взаємодію з API. Ось пояснення структури URL-адреси та правил іменування для API:

- Використовуйте описові URL-адреси та URL-адреси на основі іменників:

URL-адреси API мають використовувати описові терміни та терміни на основі іменників, які точно представляють ресурси, до яких здійснюється доступ або маніпуляції. Це допомагає зробити URL-адреси зрозумілими та легшими для розуміння. Наприклад, замість використання загальних термінів, як-от `/data` або `/object`, використовуйте конкретні та значущі терміни, як-от `/users` або `/products` для представлення відповідних ресурсів.

- Підтримуйте послідовність у структурі URL:

Важливо підтримувати узгодженість у структурі URL-адрес у всьому API. Ця узгодженість допомагає розробникам ознайомитися з шаблонами API та легко переміщатися між різними кінцевими точками. Послідовна структура URL також покращує зручність використання API та робить його більш інтуїтивно зрозумілим для клієнтів.

- Використовуйте іменники у множині для назв ресурсів:

При іменуванні ресурсів в API рекомендується використовувати іменники у множині замість іменників в однині. Наприклад, використовуйте `/users` замість `/user`, щоб представити колекцію користувачів. Іменники у множині узгоджуються з концепцією представлення колекції або групи сутностей і ширше застосовуються в дизайні API.

- Використовуйте ієрархію для вкладених ресурсів:

Якщо між ресурсами існують зв’язки або асоціації, їх зазвичай представляють за допомогою ієрархічної структури URL-адрес. Наприклад, якщо публікація в блозі містить коментарі, структура URL-адреси може бути `/posts/{postId}/comments`, щоб представляти коментарі, пов’язані з певною публікацією. Ця ієрархія допомагає

встановити зв’язки між ресурсами та забезпечує логічну структуру для доступу до вкладених ресурсів.

- Використовуйте дієслова HTTP для різних дій:

Дієслова HTTP (GET, POST, PUT, DELETE тощо) відіграють важливу роль у визначенні дій, що виконуються над ресурсами. Найкраще використовувати відповідне дієслово HTTP, щоб вказати заплановану дію на ресурсі. Наприклад, використовуйте `GET /posts`, щоб отримати список публікацій блогу, і `POST /posts`, щоб створити нову публікацію блогу.

- Включіть ідентифікатори для конкретних примірників ресурсу:

Під час доступу до конкретного екземпляра ресурсу або маніпулювання ним в URL-адресу зазвичай додається унікальний ідентифікатор (ID). Наприклад, використовуйте `/users/{userId}`, щоб представити конкретного користувача, ідентифікованого за його ідентифікатором. Це дозволяє клієнтам виконувати дії з певним екземпляром ресурсу.

- Використовуйте параметри запиту для фільтрації або сортування:

Параметри запиту можна використовувати для надання додаткових параметрів фільтрації, сортування або розбиття на сторінки для пошуку ресурсу. Наприклад, `/posts?category=technology` можна використовувати для отримання лише публікацій блогу в категорії "технології". Параметри запиту забезпечують гнучкість у отриманні певних підмножин ресурсів на основі вимог клієнта.

- Уникайте дієслів в URL-шляхах:

Зазвичай рекомендується уникати використання дієслів у шляхах URL. Натомість зарезервуйте дієслова для методів HTTP, які використовуються у взаємодії API. Використання іменників у URL-шляхах забезпечує більш зосереджену на ресурсі та послідовну угоду про найменування.

Дотримуючись цієї структури URL-адреси та правил іменування, розробники API в тому числі і я можу створювати інтуїтивно зрозумілі, узгоджені та зрозумілі API. Це сприяє простоті використання для розробників, покращує видимість кінцевих точок API і покращує загальний досвід роботи розробників під час роботи з API.

1.4Методи HTTP та дії для кожної кінцевої точки

Методи HTTP, також відомі як дієслова HTTP, відіграють вирішальну роль у дизайні RESTful API, оскільки вони визначають дії, які можна виконувати на кінцевих точках API. Кожна кінцева точка зазвичай відповідає певному методу HTTP, і комбінація методу та кінцевої точки визначає дію, яка буде виконана на пов'язаному ресурсі. Ось пояснення типових методів HTTP та відповідних дій для кожної кінцевої точки:

1. GET:

Метод GET використовується для отримання представлення ресурсу або колекції ресурсів. Це безпечний і ідемпотентний метод, тобто він не повинен змінювати стан сервера або ресурсів. Поширені дії, пов'язані з методом GET, включають:

- Отримати ресурс: `GET /posts/{postId}` отримує конкретну публікацію блогу за її ідентифікатором.
- Отримати колекцію ресурсів: `GET /posts` отримує список усіх публікацій блогу.

2. POST:

Метод POST використовується для створення нового ресурсу. Він надсилає дані на сервер, який потім обробляє та створює ресурс. Поширені дії, пов'язані з методом POST, включають:

- Створення ресурсу: `POST /posts` створює нову публікацію в блозі, використовуючи дані, надані в тілі запиту.
- Надішліть форму або дані: `POST /comments` надсилає коментар до публікації в блозі.

3. PUT:

Метод PUT використовується для оновлення існуючого ресурсу. Він замінює все представлення ресурсу наданими даними. Поширені дії, пов'язані з методом PUT, включають:

- Оновіть ресурс: `PUT /posts/{postId}` оновлює вміст певної публікації блогу за допомогою даних, наданих у тілі запиту.

4. PATCH:

Метод PATCH використовується для часткового оновлення існуючого ресурсу.

Він застосовує зміни до певних полів або атрибутів ресурсу, не замінюючи все представлення. Поширені дії, пов’язані з методом PATCH, включають:
- Часткове оновлення ресурсу: `PATCH /posts/{postId}` оновлює певні поля, такі як заголовок або вміст, допису блогу даними, наданими в тілі запиту.
5. DELETE:

Метод DELETE використовується для видалення певного ресурсу. Він видаляє ресурс із сервера. Поширені дії, пов’язані з методом DELETE, включають:
- Видалити ресурс: `DELETE /posts/{postId}` видаляє певну публікацію блогу за її ідентифікатором.

Важливо зазначити, що дії, описані вище, є загальними умовностями, але конкретні дії, пов’язані з кожною кінцевою точкою, можуть відрізнятися залежно від конструкції та вимог API. Розробники API можуть гнучко визначати дії та поведінку кожної кінцевої точки відповідно до потреб своєї програми.
Правильно відображаючи відповідний метод HTTP для кожної кінцевої точки API, розробники можуть гарантувати, що клієнти взаємодіють з API узгодженим і стандартизованим способом, дотримуючись принципів архітектури RESTful.

РОЗДІЛ2

2.1 Вибір мови програмування

Писати веб Апі можна на різних мовах Python, JavaScript , тощо, але мій вибір впав саме на мову програмування PHP (Hypertext Preprocessor) , зв’язано це з тим що вона сама по собі досить проста та має високий рівень абстракції, простий синтаксис та зручні інструменти для дебагу.

Рисунок 1 php

Також у своєму проекті я не можу обійтись і без бази даних. Я вибрав phpMyAdmin

Рисунок 2 phpMyAdmin

Так як мене цікавило максимально розібратись в темі то я вирішив обійтись без фреймворків які спростять мені роботу і писати на чистому php як люблять говорити американці from scratch. Використовував програмне середовище Visual Studio Code.

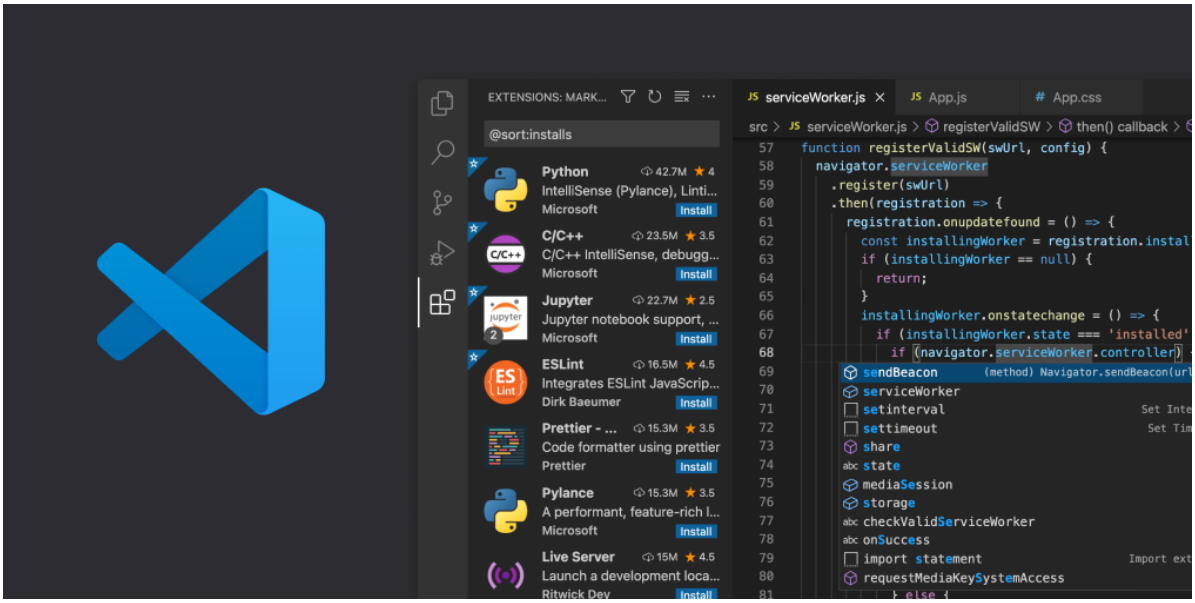

Рисунок 3 Visual Studio Code

2.2Структура проекту

У проекті присутні 3 головні директорії api, config, models.

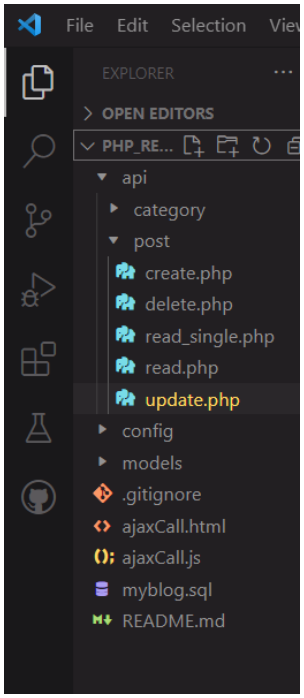

Рисунок 4 Структура проекту

У роботі в основному використовував об’єктно-орієнтоване програмування разом із PDO для підключення та запиту бази даних MySQL.

Директорія api містить в собі category та post в якій розписані методи http.

Директорія config містить в собі підключення до бази даних.

Директорія models містить моделі

Для оформлення даних на сторінці браузера було використано html.

Структура бази даних наступна

Рисунок 5 Структура бази даних

Для тестування http запитів використовувався Postman

Рисунок 6 Postman

Клієнт -Сервер

Модель клієнт-сервер:

У моделі клієнт-сервер система поділяється на два основних компоненти: клієнт і сервер.

- Клієнт: Клієнт - це програма або система, яка надсилає запити на сервер для доступу або маніпулювання ресурсами. Клієнтами можуть бути різні пристрої, наприклад веб-браузери, мобільні програми або інші сервери, які використовують API.

- Сервер: сервер відповідає за обробку запитів клієнтів, їх обробку та надання відповідей. Він керує та відкриває ресурси, з якими клієнти можуть взаємодіяти. Сервер містить бізнес-логіку, сховище даних і будь-яку необхідну обробку для

виконання запитів клієнта.

Клієнт і сервер взаємодіють через мережу за допомогою стандартизованого протоколу, як правило, HTTP (Hypertext Transfer Protocol), для обміну інформацією.

RESTful API:

API RESTful дотримуються принципів REST, який є архітектурним стилем для проектування мережевих програм. REST використовує HTTP як протокол зв’язку та використовує його методи, коди стану та уніфіковані ідентифікатори ресурсів (URI) для взаємодії з ресурсами.

- Ресурси: ресурси представляють сутності або об’єкти даних, надані сервером. Кожен ресурс ідентифікується унікальним URI (уніфікованим ідентифікатором ресурсу), і до нього можна отримати доступ, маніпулювати ним або діяти за допомогою стандартних методів HTTP.

- Зв'язок без стану: RESTful API не має стану, тобто кожен запит клієнта повинен містити всю необхідну інформацію для того, щоб сервер міг її зрозуміти та обробити. Сервер не підтримує будь-який стан клієнта між запитами. Замість цього клієнт може включити маркери автентифікації або ідентифікатори сеансу для підтримки безпеки та керування сеансом.

- Представлення ресурсів: ресурси зазвичай представлені за допомогою стандартних форматів, таких як JSON (нотація об’єктів JavaScript) або XML (розширена мова розмітки). Ці формати забезпечують загальну структуру для обміну даними між клієнтом і сервером.

3. Потік взаємодії:

Потік взаємодії між клієнтом і сервером за допомогою RESTful API зазвичай відбувається за такими кроками:

- Клієнт ініціює запит, надсилаючи запит HTTP до певного URI (ресурсу) на сервері.

- Запит містить метод HTTP, додаткові заголовки запиту та будь-які необхідні дані чи параметри.

- Сервер отримує запит, обробляє його і виконує необхідні дії над запитуваним ресурсом.

- Сервер генерує відповідну відповідь HTTP, яка містить код стану, що вказує на

результат запиту (наприклад, успіх, помилка), і може містити дані відповіді у вказаному форматі.

- Сервер надсилає відповідь назад клієнту.
- Клієнт отримує відповідь і обробляє її відповідно до коду стану та вмісту, що повертається.
- Клієнт може вибрати виконання додаткових запитів за потреби, взаємодіючи з різними ресурсами на сервері.

ВИСНОВОК

На завершення, ця дипломна робота служить вичерпним посібником для розуміння, проектування та впровадження веб-API. Вивчаючи фундаментальні концепції, архітектурні стилі, принципи проектування та методи практичної реалізації, читачі отримають необхідні знання та навички для ефективного використання веб-інтерфейсів API у своїх майбутніх проектах. Дослідження випадків використання в реальному світі та тематичних досліджень ще більше підвищує практичність і актуальність роботи. У зв’язку зі зростаючою значущістю веб-інтерфейсів API у сучасній розробці програмного забезпечення, ця дипломна робота надає читачам необхідний досвід для навігації у мінливому ландшафті взаємопов’язаних систем і програм

СПИСОК ЛІТЕРАТУРИ

1. PHP Web Services: APIs for the Modern Web 2nd Edition
2. https://www.youtube.com/watch?v=-nq4UbD0NT8
3. https://www.youtube.com/results?search_query=web+api+php
4. Building RESTful Web Services with PHP 7by Haafiz Waheed-ud-din Ahmad
5. Mastering ASP.NET Web API: Build powerful HTTP services and make the most of the ASP.NET Core Web API platform 1st Edition, Kindle Edition

Додатки

```sql
CREATE TABLE `categories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
);

INSERT INTO `categories` (`id`, `name`) VALUES
(1, 'Technology'),
(2, 'Gaming'),
(3, 'Auto'),
(4, 'Entertainment'),
(5, 'Books');

CREATE TABLE `posts` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `category_id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `body` text NOT NULL,
  `author` varchar(255) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
);

INSERT INTO `posts` (`id`, `category_id`, `title`, `body`, `author`) VALUES
(1, 1, 'Technology Post One', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut interdum est nec lorem mattis interdum
(2, 2, 'Gaming Post One', 'Adipiscing elit. Ut interdum est nec lorem mattis interdum. Cras augue est, interdum eu consectetur et
(3, 1, 'Technology Post Two', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut interdum est nec lorem mattis interdum
(4, 4, 'Entertainment Post One', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut interdum est nec lorem mattis inter
(5, 4, 'Entertainment Post Two', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut interdum est nec lorem mattis inter
(6, 1, 'Technology Post Three', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut interdum est nec lorem mattis interd
```

Рисунок 6 Запис у базу даних

```php
<?php
  // Headers
  header('Access-Control-Allow-Origin: *');
  header('Content-Type: application/json');
  header('Access-Control-Allow-Methods: POST');
  header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization,X

  include_once '../../config/Database.php';
  include_once '../../models/Category.php';
  // Instantiate DB & connect
  $database = new Database();
  $db = $database->connect();

  // Instantiate blog post object
  $category = new Category($db);

  // Get raw posted data
  $data = json_decode(file_get_contents("php://input"));

  $category->name = $data->name;

  // Create Category
  if($category->create()) {
    echo json_encode(
      array('message' => 'Category Created')
    );
  } else {
    echo json_encode(
      array('message' => 'Category Not Created')
    );
  }
```

Рисунок 7 Метод Create

```php
<?php
  // Headers
  header('Access-Control-Allow-Origin: *');
  header('Content-Type: application/json');
  header('Access-Control-Allow-Methods: DELETE');
  header('Access-Control-Allow-Headers: Access-Control-Allow-Headers, Content-Type, Access-Control-Allow-Methods, Authorization,X

  include_once '../../config/Database.php';
  include_once '../../models/Category.php';
  // Instantiate DB & connect
  $database = new Database();
  $db = $database->connect();

  // Instantiate blog post object
  $category = new Category($db);

  // Get raw posted data
  $data = json_decode(file_get_contents("php://input"));

  // Set ID to UPDATE
  $category->id = $data->id;

  // Delete post
  if($category->delete()) {
    echo json_encode(
      array('message' => 'Category deleted')
    );
  } else {
    echo json_encode(
      array('message' => 'Category not deleted')
    );
```

Рисунок 8 Метод Delete

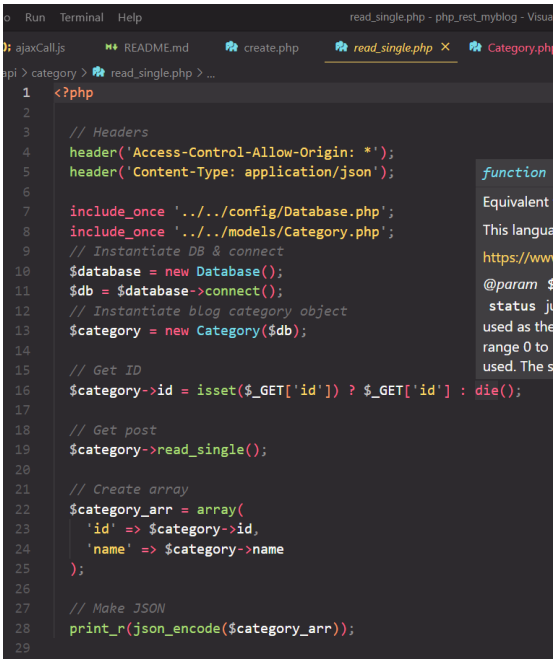

Рисунок 9 Читання одного параметру

```php
class Database {
  // DB Params
  1 reference
  private $host = 'localhost';
  1 reference
  private $db_name = 'myblog';
  1 reference
  private $username = 'root';
  1 reference
  private $password = '';
  4 references
  private $conn;

  // DB Connect
  1 reference | 0 overrides
  public function connect() {
    $this->conn = null;

    try {
      $this->conn = new PDO('mysql:host=' . $this->host . ';dbname=' . $this->db_name, $this->username, $this->password);
      $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch(PDOException $e) {
      echo 'Connection Error: ' . $e->getMessage();
    }

    return $this->conn;
  }
}
```

Рисунок 10 Підключення до Бази Даних

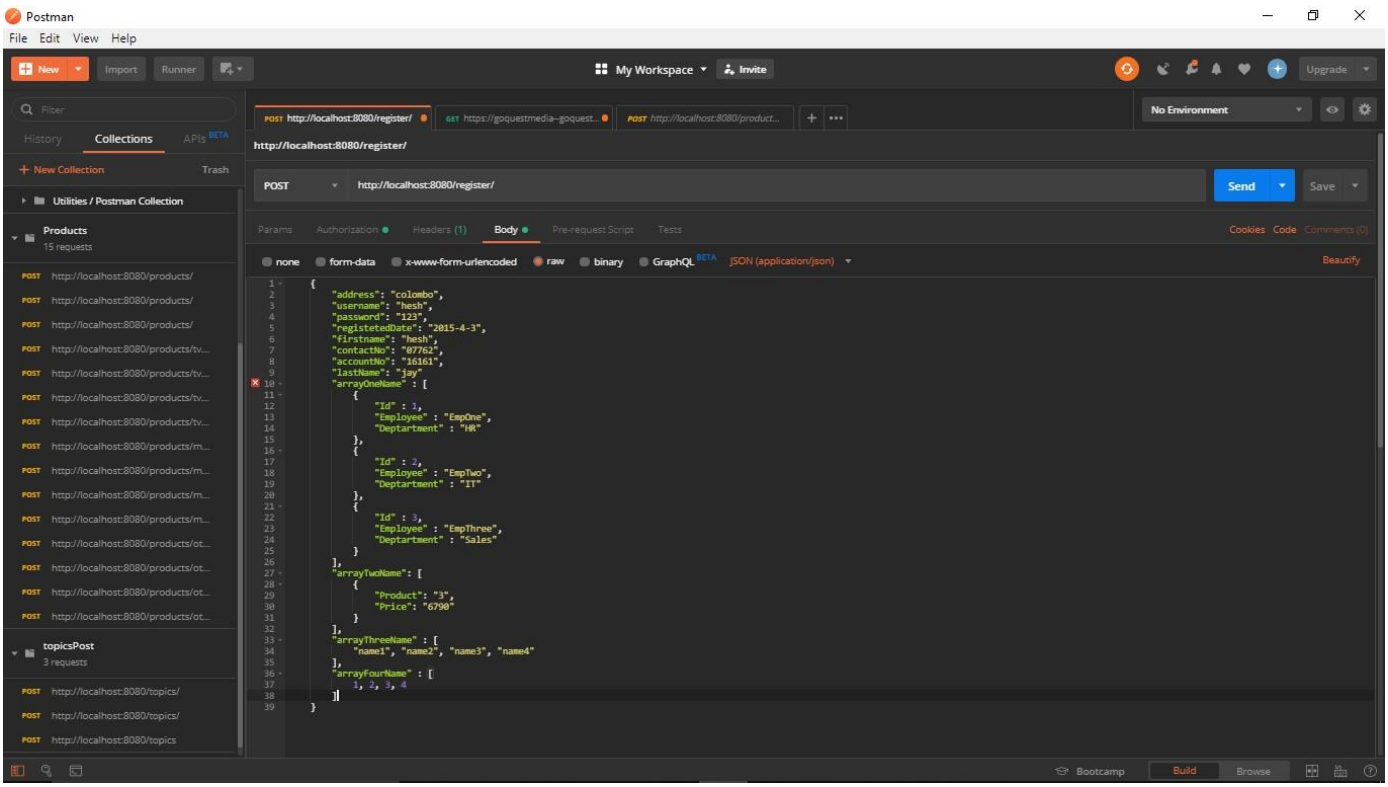

Рисунок 11 Список блогів отриманий від API