Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка навчального програмного забезпечення для демонстрації використання протоколу HTTP»
за освітньою програмою: «12 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»

Виконав: студент групи «ПЗ1911»

_____ Іван БОЛТЕНКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ доц. Вадим АНДРЮЩЕНКО
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ доц. Світлана ВОЛКОВА
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2023 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note

to Bachelor's Thesis

on the topic: «Development of educational software for demonstrating the use of the HTTP protocol»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1911: / Ivan BOLTENKOV /

Scientific Supervisor: /Vadym ANDRIUSHCHENKO/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____/Вадим ГОРЯЧКІН/
(підпис)
Дата_____

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра
студенту Болтенкову Івану Володимировичу
1. Тема роботи: «Розробка навчального програмного забезпечення для демонстрації використання протоколу HTTP»
Керівник роботи: Андрющенко Вадим Олександрович, доцент
затверджені наказом № 1209 ст від 07.12.2022
2. Строк подання студентом роботи: 18.06.2023 р.
3. Вихідні дані до роботи: _____
_____
4. Пояснювальна записка, зміст (питання, для опрацювання):
4.1 Аналіз предметної галузі
4.2 Формування вимог до програмної системи
4.3 Архітектура та проектування програмного забезпечення
4.4 Опис прийнятих програмних рішень
4.5 Тестування розробленого програмного забезпечення

5. Перелік графічного матеріалу:
5.1 Доповідь до кваліфікаційної роботи бакалавра
5.2 Презентація до кваліфікаційної роботи бакалавра
5.3 Демонстраційне відео програми згідно теми кваліфікаційної роботи бакалавра

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.02.23	10%
2	Розробка специфікацій ПЗ	01.03.23	20%
3	Об'єктний аналіз поставленої задачі	10.03.23	30%
5	Розробка програмного забезпечення	05.05.23	65%
6	Тестування програмного забезпечення	22.05.23	85%
7	Подання кваліфікаційної роботи до кафедри	18.06.23	100%
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.23	

Студент _____ Іван БОЛТЕНКОВ _
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ _доц. Вадим АНДРЮЩЕНКО__
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра:

114 с., 53 рис. , 2 додатки, 10 джерел.

Об’єкт дослідження в бакалаврській роботі – процес розробки та реалізація навчального програмного забезпечення для демонстрації використання HTTP протоколу.

Предмет дослідження – демонстрація роботи HTTP протоколу та її моделювання за допомогою основних методів GET, POST, PUT, DELETE.

Мета дослідження – створення навчального програмного забезпечення для вивчення та моделювання роботи HTTP протоколу з наглядною демонстрацією запитів та відповідей сервера за допомогою бібліотек Axios та jQuery.

Методи дослідження – платформа ASP.NET Core, Azure, HTML, CSS, мова C# та Javascript, Visual Studio.

У результаті роботи було зібрано та проаналізовано предметну галузь та зібрано дані для розробки застосунка, розроблено специфікації програмного засобу, проведено об'єктний аналіз поставленої задачі та розроблено програмне забезпечення, що дозволяє продемонструвати використання HTTP протоколу. Було протестовано програмне забезпечення на модульному рівні та прийнято до застосування в якості навчального програмного забезпечення.

WEB, URL, СЕРВЕР, HTTP ПРОТОКОЛ, МЕРЕЖЕВІ ДОДАТКИ, СТАТУСНІ КОДИ HTTP, МЕТОДИ HTTP, .NET, JAVASCRIPT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP: HyperText Transfer Protocol - протокол передачі гіпертексту, який використовується для обміну даними між веб-клієнтом і веб-сервером.

URL: Uniform Resource Locator - однозначний ідентифікатор ресурсу в Інтернеті, який включає протокол, домен, шлях до ресурсу та іншу інформацію.

HTML: HyperText Markup Language - мова розмітки гіпертексту, що використовується для створення веб-сторінок.

API: Application Programming Interface - інтерфейс програмування додатків, який надає набір функцій та можливостей для взаємодії з програмним забезпеченням.

GET: метод HTTP-запиту, що використовується для отримання ресурсу з сервера.

POST: метод HTTP-запиту, що використовується для надсилання даних на сервер для обробки.

PUT: метод HTTP-запиту, що використовується для оновлення ресурсу на сервері.

DELETE: метод HTTP-запиту, що використовується для видалення ресурсу на сервері.

JSON: JavaScript Object Notation - формат обміну даними, що базується на синтаксисі об'єктів JavaScript, часто використовується для передачі даних між клієнтом і сервером.

TLS: Transport Layer Security - протокол шифрування і безпеки, що використовується для захисту передачі даних через мережу, включаючи HTTP.

IoT: Internet of Things - концепція, за якою фізичні пристрої, обладнані спеціальними сенсорами та здатні підключатися до Інтернету, здатні обмінюватися даними і взаємодіяти один з одним для забезпечення автоматизованої функціональності та сприяння зручності в користуванні.

OSI: Open Systems Interconnection - стандартна модель відкритої системи взаємозв'язку, що складається з сімох рівнів або шарів, які визначають функції і протоколи для забезпечення комунікації між комп'ютерними системами.

XML: eXtensible Markup Language - мова розмітки, що використовується для представлення структурованої інформації у вигляді текстових файлів. XML дозволяє описувати дані та їх структуру, що полегшує обмін інформацією між різними системами.

TCP: Transmission Control Protocol - протокол передачі даних у комп'ютерних мережах, який забезпечує надійну та впорядковану передачу пакетів даних між вузлами мережі.

UML: Unified Modeling Language - стандартизована мова моделювання, яка використовується для візуального опису структури та поведінки програмних систем. UML дозволяє розробникам використовувати спеціальні діаграми для відображення концепцій, класів, відносин та інших аспектів програмного проектування.

ВСТУП

Актуальність роботи. Інтернет став невід'ємною частиною повсякденного життя. В ньому ми здійснюємо покупки, проводимо банківські операції, спілкуємося та розважаємося. З розвитком Інтернету речей (IoT - Internet of Things) все більше пристроїв підключається до мережі, що надає нам можливість отримувати віддалений доступ до них. Реалізація такого доступу стала можливою завдяки розробці кількох технологій, включаючи протокол передачі гіпертексту (Hypertext Transfer Protocol, HTTP)[8].

Значення програмного забезпечення постійно зростає. У цьому контексті протокол HTTP є ключовим елементом веб-технологій, що використовується для передачі даних між клієнтом і сервером. Актуальність розробки навчального програмного забезпечення для демонстрації використання протоколу HTTP полягає в необхідності підготовки кваліфікованих фахівців з програмування, які розуміють і вміють ефективно використовувати цей протокол. Це дозволить їм розробляти веб-додатки, які забезпечують швидку та надійну комунікацію між користувачем та сервером.

Отже, моделювання роботи з HTTP протоколом забезпечить ефективний процес навчання, покращить розуміння протоколу HTTP та його застосування у сучасному програмуванні.

Мета роботи. Основною метою цієї роботи є розробка навчального програмного забезпечення, яке надасть користувачам можливість ознайомитись з протоколом HTTP, його функціональністю та використанням у веб-програмуванні. Програма буде спрямована на забезпечення практичних навичок роботи з протоколом HTTP, а також на розуміння його ролі і впливу на розробку веб-додатків.

Експлуатаційне призначення. Розроблене навчальне програмне забезпечення буде використовуватись в освітніх закладах, де викладаються курси з програмування, веб-розробки та інформаційних технологій. Воно стане цінним інструментом для викладачів та студентів, дозволяючи їм практично

освоїти основи протоколу HTTP та набути навичок розробки веб-додатків, які працюють з цим протоколом.

1 ЗБІР ТА АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Аналіз предметної галузі

1.1.1 Розвиток предметної галузі

HTTP — поширений протокол передачі, який спочатку був призначений для передачі гіпертекстових документів (тобто документів, які можуть містити посилання, що дозволяють організувати перехід до інших документів). Протокол HTTP (абревіатура від англ. HyperText Transfer Protocol, протокол передачі гіпертексту) є протоколом сьомого прикладного рівня моделі OSI. HTTP є основою системи World Wide Web, за допомогою якого надається можливість перегляду, доступу до веб-сторінок у браузері та забезпечується робота Інтернет-мережі. У цьому полягає головна задача протоколу.

Протокол, який початково був розроблений для зв'язку між веб-браузерами і веб-серверами, став також використовуватися і для інших цілей: передачі гіпертекстових документів, передачі зображень та відео або для відправки контенту на сервери. HTTP часто використовується як «транспорт» для передачі інформації для інших протоколів прикладного рівня, таких як SOAP, XML-RPC та WebDAV. API багатьох програмних продуктів також передбачає використання HTTP для передачі даних — самі дані при цьому можуть мати будь-який формат, наприклад, XML або JSON. Як правило, передача даних за протоколом HTTP здійснюється через TCP/IP-з'єднання. Серверне програмне забезпечення при цьому зазвичай використовує TCP-порт 80 (і, якщо порт не вказано явно, то зазвичай клієнтське програмне забезпечення за замовчуванням використовує саме 80-й порт для HTTP-з'єднань, що відкриваються), хоча може використовувати і будь-який інший.

Робота HTTP основана на клієнт-серверній архітектурі передачі даних, де йде “спілкування” між клієнтом та сервером за допомогою запитів та відповідей. Цей протокол використовує класичну модель клієнт-сервер, при якій клієнт (веб-

браузер) робить запит, сервер формує відповідь і відправляє її назад клієнту у вигляді інформації, яку ми бачимо в браузері.

Браузер завжди є ініціатором запитів. Щоб веб-сторінка відобразилася на екрані, він відправляє вихідний запит на отримання HTML-документа. Потім відбувається аналіз файлу, за допомогою додаткових запитів, відповідних до сценаріїв виконання. Далі браузер компонує отримані ресурси і відображає цілісний документ (в даному випадку – веб-сторінку). Сценарії, що виконуються браузером, можуть отримувати більше ресурсів на більш пізніх етапах, тому відбувається оновлення веб-сторінки.

Також необхідно зазначити, що клієнт і сервер не зберігають інформацію про попередні запити. Тому кожен запит містить всю необхідну інформацію.

На протилежному боці каналу зв'язку знаходиться сервер, який обробляє документи за запитом клієнта (в даному випадку – браузера),, формує відповідь і передає його клієнту. Після цього клієнтська програма може продовжити надсилати інші запити, які будуть оброблені аналогічним чином.

Запити (рисунок 1.1) та відповіді формуються в суворій послідовності і відповідають певній структурі. Метод запиту визначає дію, яку потрібно виконати на ресурсі.

Рисунок 1.1 – Складові HTTP запиту

Коли запит надсилається на сервер, сервер відправляє відповідь. Відповідь від сервера дозволяє визначити, чи був запит успішним. Відповіді складаються з наступних компонент:

- код статусу (вказує був запит успішним чи ні);
- HTTP заголовки;
- тіло, що містить вилучений ресурс.

На даний момент дуже багато компаній перейшло на мікросервіси у своїх додатках. Це означає, що додатки складаються з різних секцій, у яких є свої сховища даних, і використовують різні команди для взаємодії з ними.

Більшість мікросервісів використовують API, а API використовує REST-запити і SOAP-запити. Саме їх і треба тестувати.

HTTP є розширюваним протоколом, досить простим у використанні. Структура клієнт-сервер у поєднанні з можливістю простого додавання заголовків дозволяє HTTP розвиватися разом із розширенням можливостей WEB. Також за допомогою тестування API можна протестувати багато сценаріїв, які не можна протестувати через призначений для користувача інтерфейс. Цю стратегію можна використовувати для пошуку недоліків дизайну і безпеки на ранніх стадіях розробки.

Таким чином, протокол HTTP відправляє клієнту дані, що цікавлять його: весь контент, графіку, фотографії, а також банери, і не зберігає інформацію про користувачів. Перевага цього рішення, безумовно, полягає в меншому завантаженні даних на сервер, що призводить до набагато більш високої продуктивності веб-сайту. Однак це також пов'язано з одним особливим недоліком - щоб включити веб-сайт кілька разів, його необхідно знову завантажити з сервера. Є рішення цієї проблеми - це файли cookie. Вони додаються на сайт і постійно збирають дані про користувачів мережі. Зібрана інформація може використовуватися власником веб-сайту для багатьох маркетингових кампаній. За допомогою файлів cookie він отримує інформацію про інтереси та останніх відвіданих сторінках відвідувачів свого веб-сайту. Завдяки цьому він може рекламувати свої нові товари або спонукати їх купувати

конкретний товар, якщо замовлення не було оформлено раніше. Це звичайна тактика інтернет-маркетингу.

Швидкий розвиток технологій і зростаюча необхідність в глибокому розумінні протоколу HTTP створюють потребу у навчальних матеріалах, які забезпечать студентам необхідні знання і навички для розробки програмного забезпечення, що використовує цей протокол. Таким чином, основною метою цієї роботи є проектування навчального програмного забезпечення, яке демонструватиме використання протоколу HTTP і надаватиме користувачам можливість вивчити його роботу та використання на практиці.

Отже, в процесі виконання дипломної роботи необхідно виконати такі завдання:

1. Виконати аналіз аналогів та спроектувати вимоги до навчального програмного забезпечення, що демонструє використання протоколу HTTP.

2. Виконати проектування архітектури програмного засобу з розробкою таких діаграм: Package Diagram, Components Diagram, Use Case Diagram, Deployment Diagram.

3. Реалізувати програмне забезпечення яке відповідає поставленим вимогам та розробленій архітектурі.

4. Провести тестування розробленого програмного забезпечення

1.1.2 Аналіз аналогів

На даний момент у вільному доступі наявно лише декілька програмних забезпечень, що можуть використовуватись для демонстрації використання HTTP протоколу.

Першим прикладом існуючого аналогу є “Postman ” (рисунок 1.2).

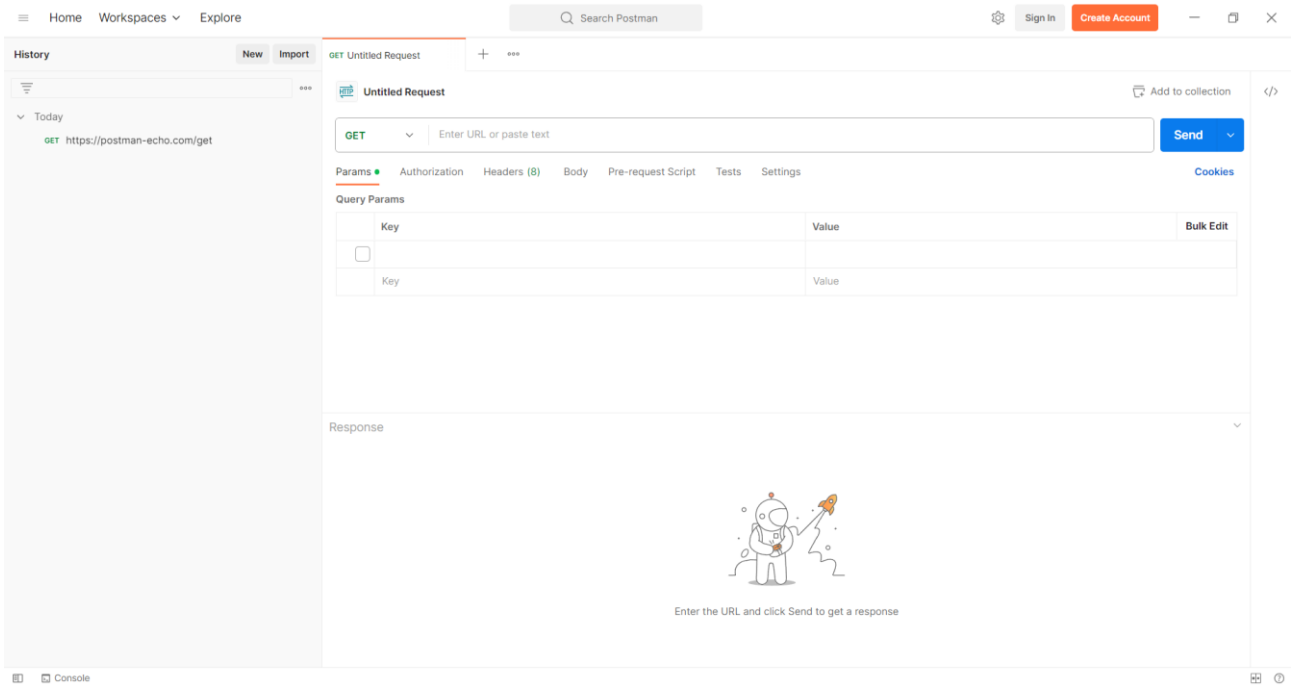

Рисунок 1.2 – Головна сторінка програми Postman

Postman є однією з найпопулярніших програм для тестування та взаємодії з API за допомогою протоколу HTTP. Ця програма надає зручний інтерфейс для створення та відправки HTTP-запитів, включаючи GET, POST, PUT і DELETE запити. Postman також підтримує автоматизацію запитів, налаштування заголовків, параметрів та перевірку відповіді сервера.

Переваги Postman:

1. Інтуїтивний і легкий у використанні: Postman має дружній інтерфейс, що дозволяє швидко орієнтуватися в його функціоналі навіть початківцям. Він пропонує зручний спосіб створення, надсилання та тестування HTTP-запитів.

2. Підтримка різних методів та форматів: Postman дозволяє використовувати різні HTTP-методи (рисунок 1.3), такі як GET, POST, PUT, DELETE та інші, для взаємодії з веб-серверами.

3. Postman також підтримує різні формати даних, включаючи JSON та XML, для передачі та отримання інформації (рисунок 1.4).

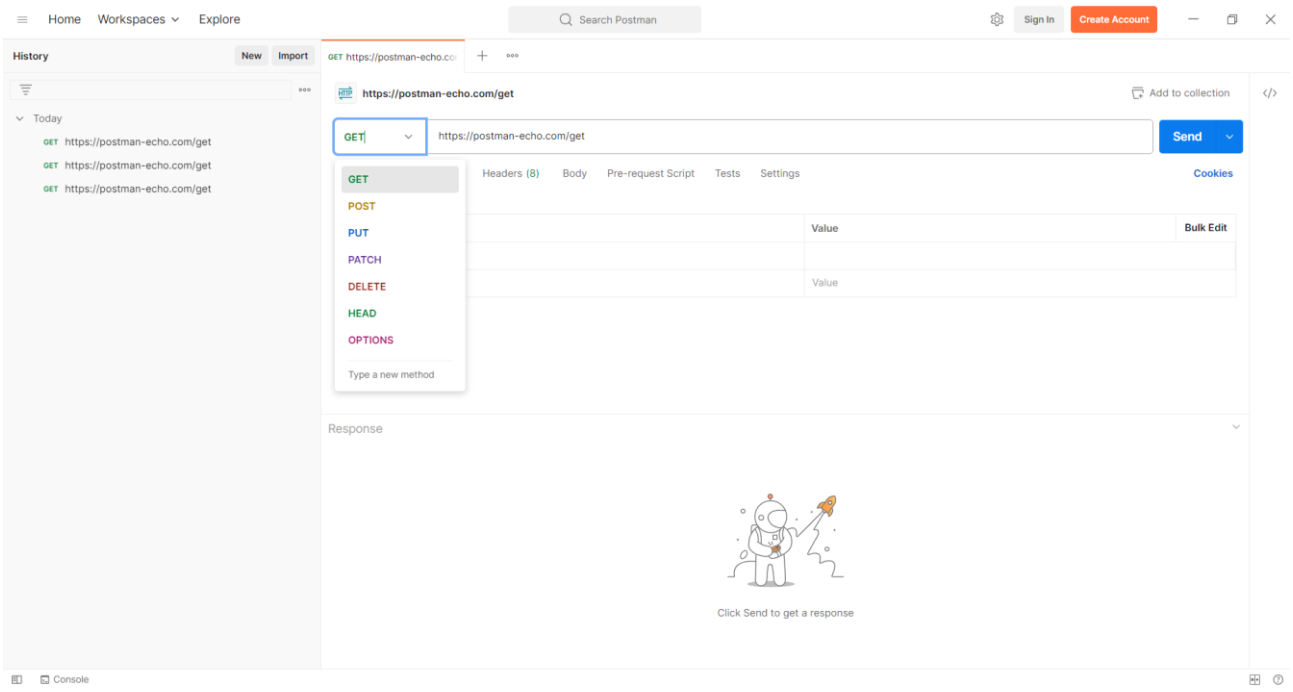

Рисунок 1.3 – Демонстрація доступних методів

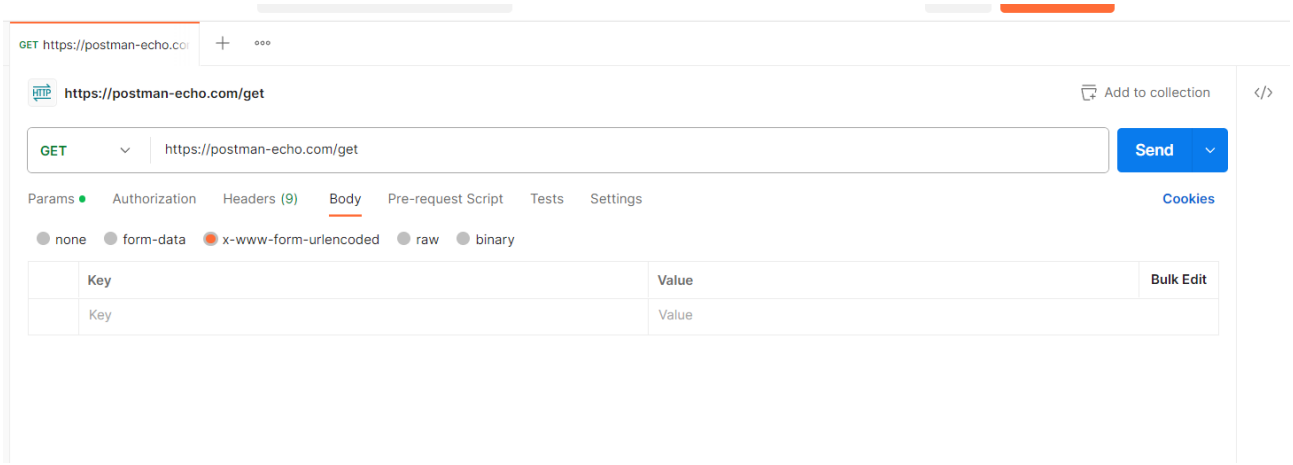

Рисунок 1.4 – Демонстрація можливості введення тіла запиту у зручному форматі

4. Автоматизація та зберігання колекцій: Postman дозволяє створювати колекції запитів, що полегшує організацію та повторне використання ваших HTTP-запитів. Ви можете зберігати, імпортувати та експортувати колекції (рисунок 1.5), а також автоматизувати виконання набору запитів за допомогою зручного інтерфейсу.

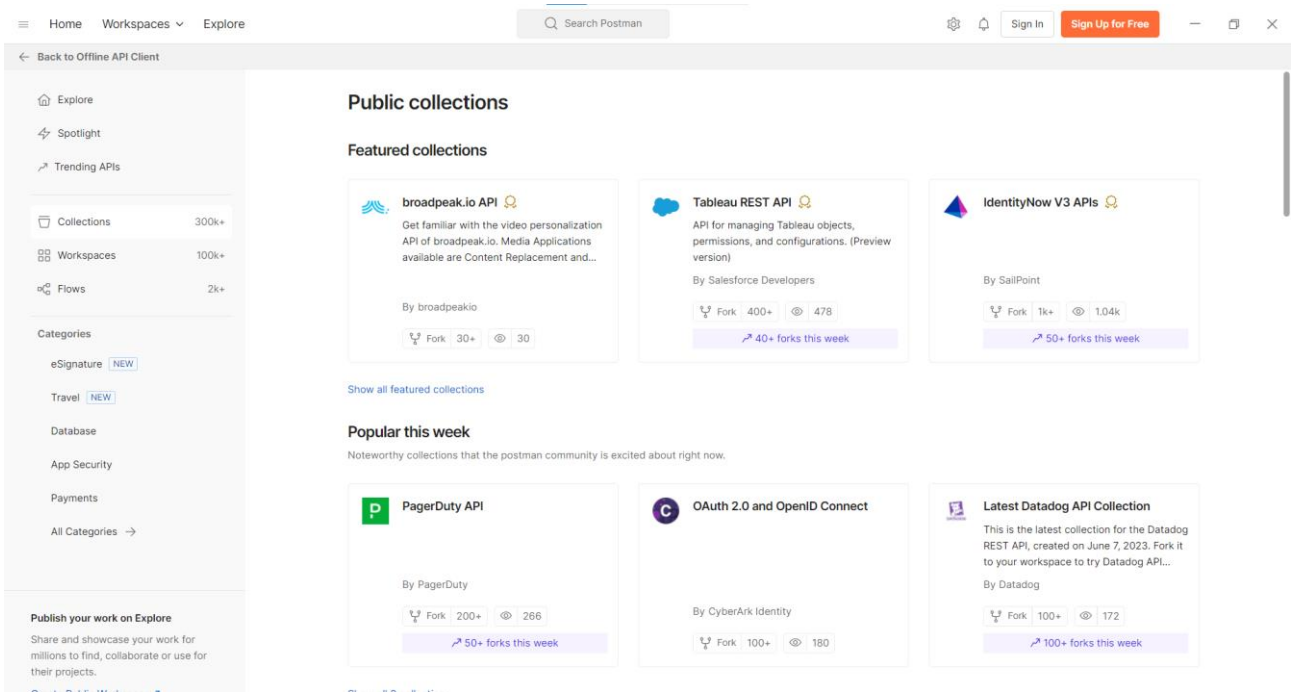

Рисунок 1.5 – Сторінка колекцій

5. Тестування та звіти: Postman надає можливість створювати тести для перевірки правильності відповідей веб-серверів на ваші запити (рисунок 1.6). Ви можете налаштовувати умови тестування, перевіряти статус-коди, заголовки, вміст відповідей та інші параметри. Крім того, Postman генерує детальні звіти, які допомагають вам аналізувати результати тестування.

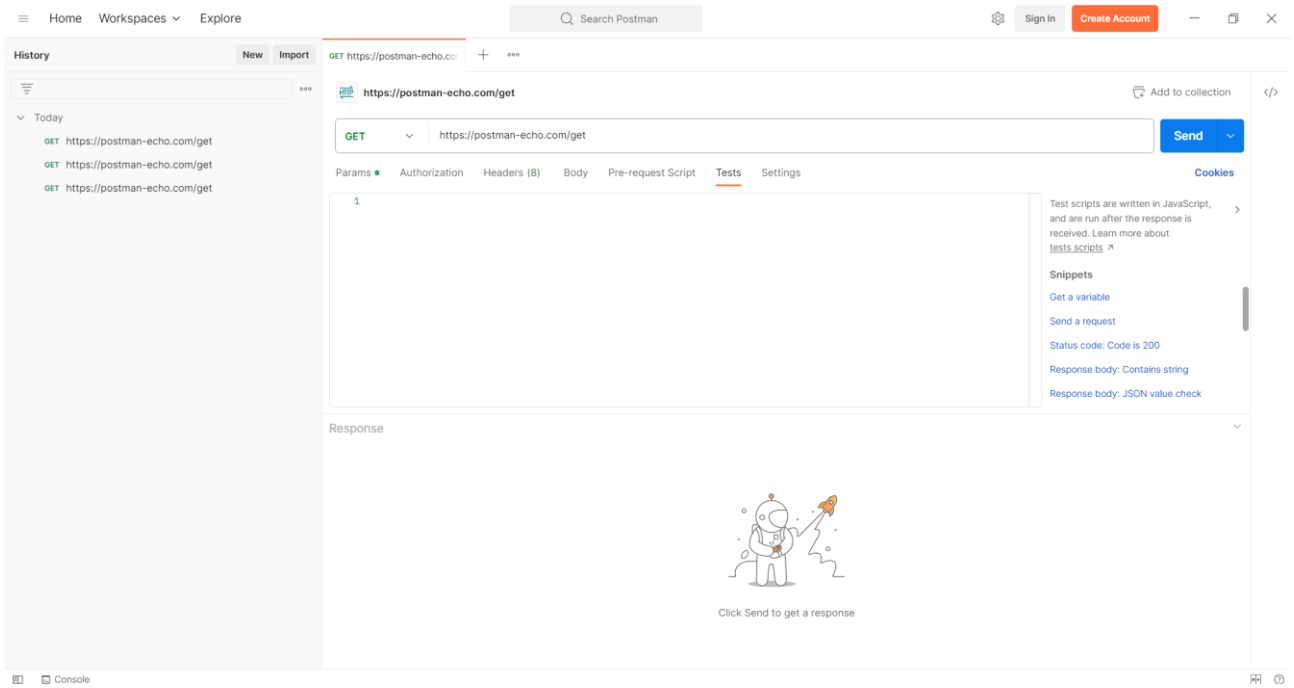

Рисунок 1.6 – Демонстрація наявності вікна для тестування

Недоліки Postman:

1. Обмежена підтримка масштабних проектів: Хоча Postman є потужним інструментом для розробки та тестування API, він може мати обмежену ефективність у великих проектах з багатьма запитами та складними сценаріями.

2. Postman лише демонструє відповідь сервера (рисунок 1.7). На жаль, цього не достатньо для повного розуміння використання протоколу HTTP.

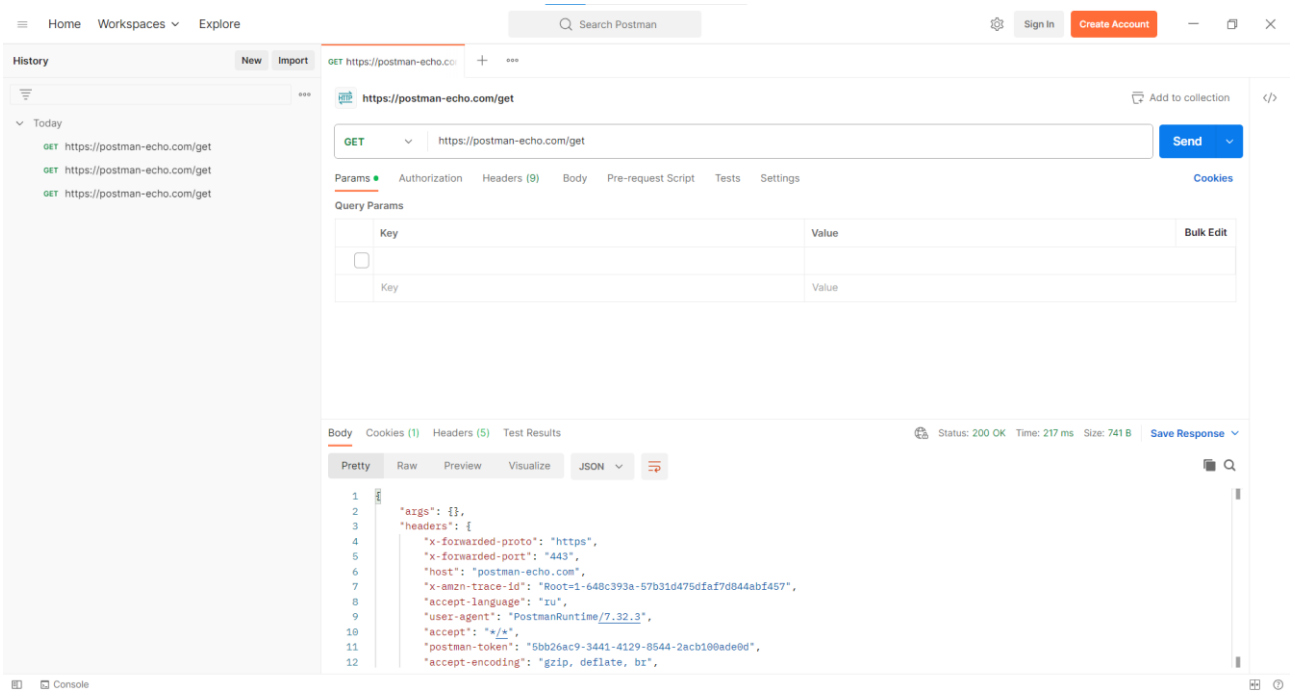

Рисунок 1.7 – Приклад надісланого запиту та отриманої відповіді

3. Відсутні пояснення та підказки щодо взаємодії з HTTP протоколом. Новачкам буде складно зрозуміти, що означають деякі методи та заголовки.

Наступним аналогом є Advanced Rest Client (рисунок 1.8).

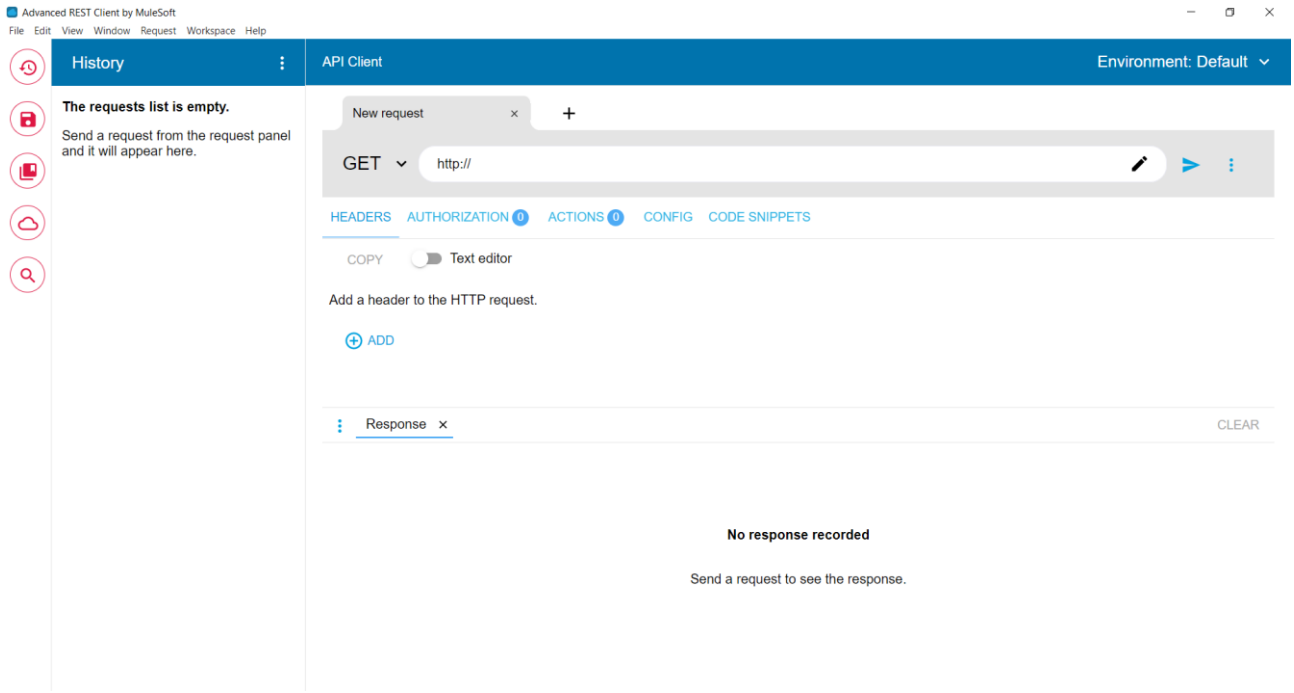

Рисунок 1.8 – Головна сторінка програми Advanced Rest Client

Advanced Rest Client є інструментом для тестування та відлагодження HTTP-запитів і взаємодії з API. ARC надає зручний спосіб створення, відправки та перевірки HTTP-запитів із можливістю роботи з різними форматами даних, такими як JSON, XML, форми даних тощо.

Основні переваги Advanced Rest Client включають:

1. Створення запитів: Ви можете створювати HTTP-запити за допомогою різних методів (GET, POST, PUT, DELETE тощо) (рисунок 1.9), встановлювати заголовки запиту (рисунок 1.10), додавати параметри та тіло запиту в потрібному форматі (рисунок 1.11).

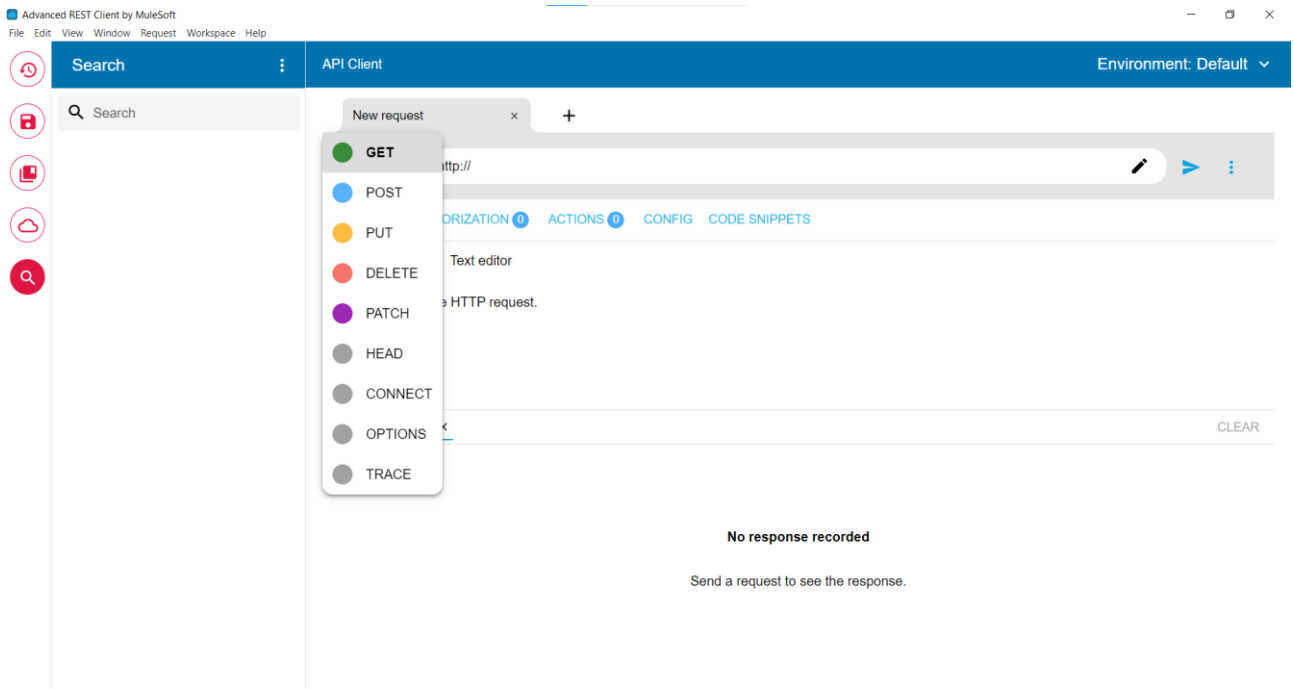

Рисунок 1.9 – Демонстрація вікна для вибору методу

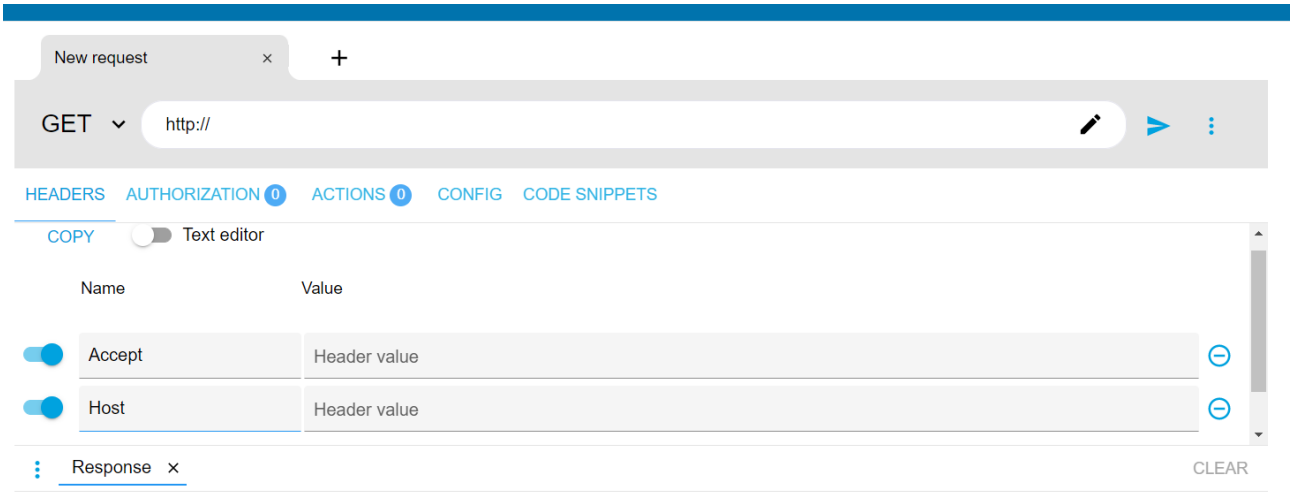

Рисунок 1.10 – Демонстрація можливості додавання заголовків

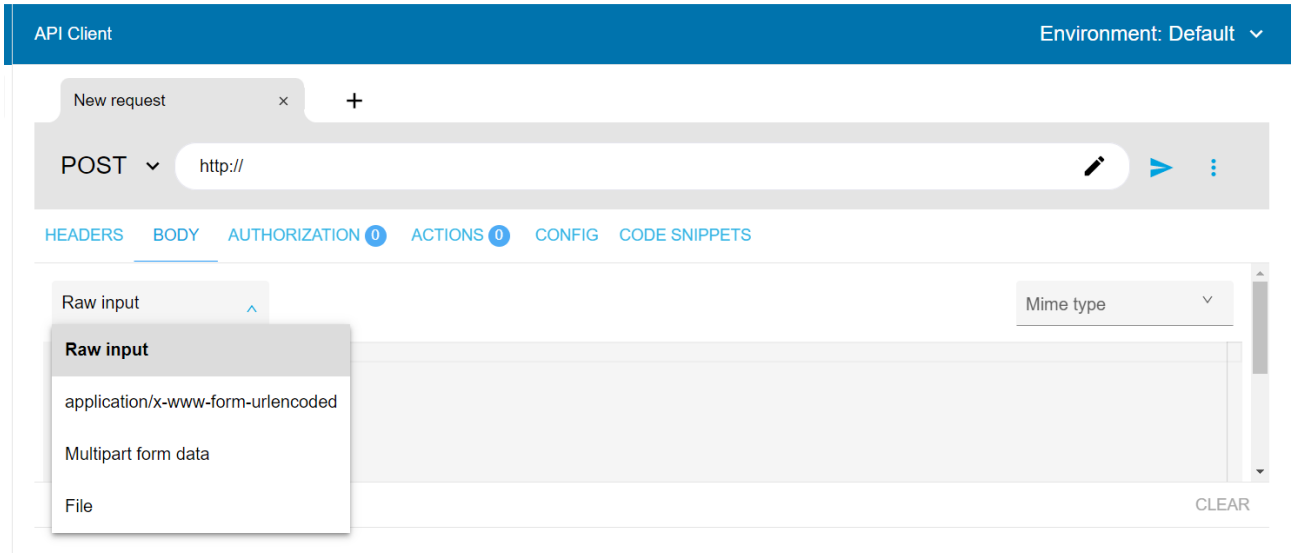

Рисунок 1.11– Демонстрація віконця для вибору типу даних для тіла запиту

2. Відправка запитів: ARC дозволяє відправляти запити на сервер та отримувати відповіді. Ви можете переглядати статус коди та тіло відповіді (рисунок 1.12).

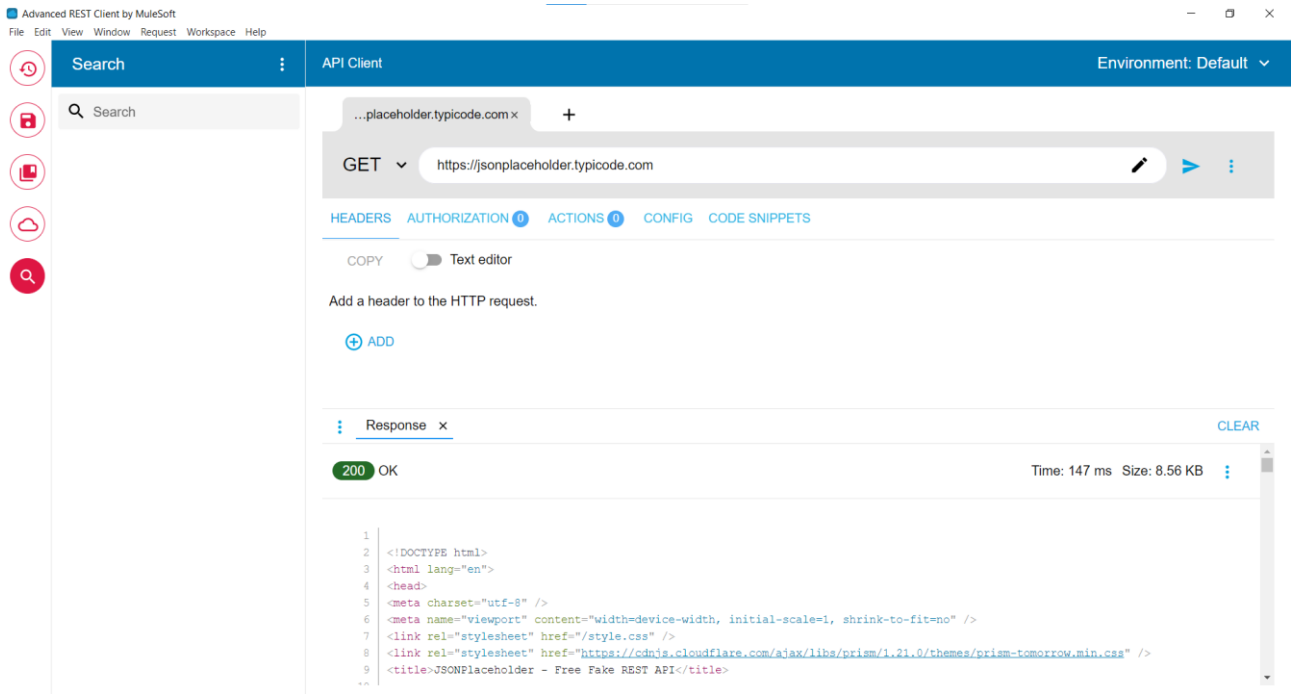

Рисунок 1.12 – Приклад надісланого запиту та отриманої відповіді

3. Робота з форматами даних: ARC підтримує різні формати даних, такі як JSON, XML, форма даних (form data). Ви можете зручно створювати та редагувати дані в цих форматах.

4. Автоматизація запитів: ARC надає можливість створювати скрипти на JavaScript для автоматизації запитів та виконання певних дій. Ви можете використовувати змінні, цикли та умовні вирази для створення скриптів.

5. Тестування та збереження запитів: Ви можете створювати тести для перевірки відповідей сервера та зберігати набір запитів в колекціях для подальшого використання.

Проте є певні недоліки:

1. Не дуже зручний та зрозумілий інтерфейс.

2. Не відображається, у якому вигляді HTTP запит бачить саме сервер (рисунок 1.13).

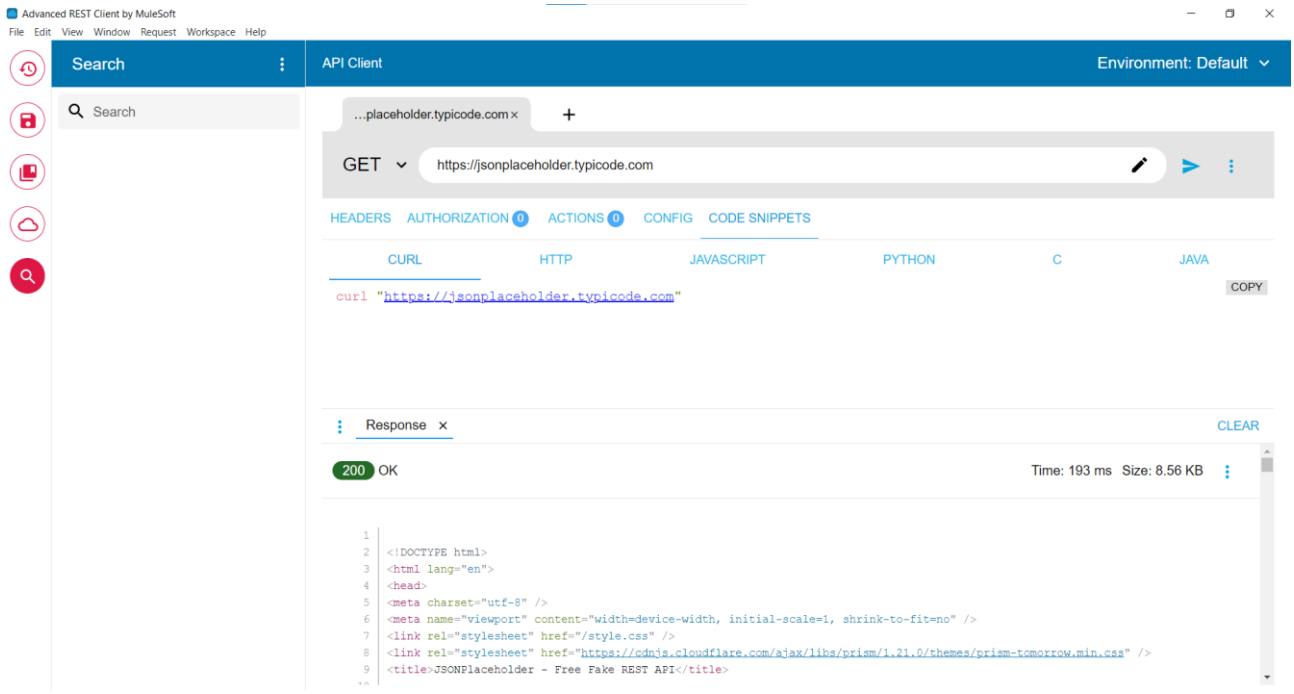

Рисунок 1.13 – Демонстрація отриманої відповіді. Не відображається у якому вигляді запит надійшов до серверу

3. Відсутні пояснення щодо взаємодії з HTTP протоколом. Новачкам буде складно зрозуміти, що означають деякі методи та заголовки.

1.2 Виявлення проблем та актуалізація рішень

У результаті проведеного критичного аналізу було виявлено, що існуючі засоби для вивчення протоколу HTTP недостатньо зорієнтовані на теоретичну складову та не забезпечують достатньої інтерактивності. Ці інструменти мають обмежену кількість пояснювального матеріалу, що може ускладнити їх використання у навчальних цілях.

Натомість розроблене програмне забезпечення надасть більше інформації користувачеві, що значно прискорить та покращить процес навчання.

Також розроблене навчальне програмне забезпечення буде відрізнятись від інших рішень, оскільки воно надасть користувачам можливість активно взаємодіяти з протоколом HTTP, емулюючи реальні ситуації взаємодії клієнта і сервера.

1.3 Постановка задач

Метою даної роботи є розробка навчального програмного забезпечення, яке надасть користувачам можливість ознайомитися з основними принципами та використанням протоколу HTTP. Програма повинна бути спрямована на студентів, початківців програмістів та будь-яких осіб, які зацікавлені в навчанні цього протоколу.

Навчальне програмне забезпечення для демонстрації використання HTTP протоколу повинне надавати такий функціонал:

- Забезпечення можливості створення HTTP запитів до веб-серверів та отримання відповідей.
- Підтримка різних видів HTTP методів, таких як GET, POST, PUT, DELETE.
- Відображення статусу запиту та відповіді, включаючи HTTP коди статусу.
- Підтримка передачі параметрів запиту, заголовків та тіла запиту.

- Надання пояснювального матеріалу, включаючи документацію, приклади використання та пояснення щодо використання методів та заголовків.

Очікувані результати: Очікується, що розроблене навчальне програмне забезпечення буде зручним та доступним для користувачів, надаватиме цінний навчальний матеріал і допоможе їм краще розуміти та використовувати протокол HTTP.

Результатом виконання поставленої задачі буде функціональне та ефективне навчальне програмне забезпечення, яке надасть користувачам зручні та інтерактивні інструменти для вивчення та використання протоколу HTTP.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Розроблене навчальне програмне забезпечення для демонстрації використання протоколу HTTP повинно відповідати наступним функціональним вимогам:

- Відображення запитів і відповідей: Програма повинна відображати як запити, так і відповіді, які обмінюються між клієнтом та сервером. Це включає відображення HTTP методу, заголовків, тіла запиту та відповіді.
- Створення запитів: Користувач повинен мати можливість створювати різні типи запитів, включаючи GET, POST, PUT, DELETE. Програма повинна надавати інтерфейс для введення URL, параметрів запиту, тіла запиту та інших відповідних даних.
- Налагодження запитів: Програма повинна забезпечувати можливість налагоджувати запити шляхом додавання, видалення або зміни заголовків, параметрів та тіла запиту. Користувач повинен мати контроль над усіма аспектами запиту.
- Відправлення запитів: Після налаштування запиту, програма повинна надати можливість відправляти запит до визначеного сервера та отримувати відповідь.
- Аналіз відповідей: Програма повинна аналізувати та відображати вміст отриманих відповідей, включаючи статус-код, заголовки та тіло відповіді.
- Підтримка автоматичного заповнення: Програма може надавати функціонал автоматичного заповнення певних полів запиту або вибору заголовків зі списку, щоб полегшити створення запитів.

2.2 Вимоги до програмного забезпечення

Для того, щоб розгорнути серверний застосунок, необхідно мати наступне програмне забезпечення:

−ОС: Windows 10 або вище;

−.Net Core версії 7.0;

−Node.js версії 18.16.0 або вище;

−Npm версії 9.7.1 або вище.

2.3 Вимоги до апаратного забезпечення

Вимоги до апаратного забезпечення для комфортної розробки застосунку:

- 16 гігабайт оперативної пам’яті або більше;
- не менш ніж 25 гігабайт вільного місця на диску;
- чотирьох ядерний процесор, або більше.

2.4 Вхідні дані

Вхідні дані:

•URL: Користувач вводить URL-адресу сервера, з яким вони бажають взаємодіяти. Це може бути URL-адреса реального веб-сайту або локального сервера для тестування.

•HTTP метод: Користувач вибирає HTTP метод, такий як GET, POST, PUT, DELETE, для виконання запиту.

•Параметри запиту: Користувач може вказати параметри запиту, такі як рядки запиту, заголовки, тіло запиту або інші відповідні дані, що відправляються разом з запитом.

2.5 Вихідні дані

Вихідні дані:

- Відповідь сервера: Після виконання запиту до сервера, програма може отримати відповідь сервера, яка містить різні дані, такі як статус-код, заголовки, тіло відповіді та інші відповідні деталі.
- Відповідь на запит: Програма може вивести відповідь на запит користувача, що включає дані, які були отримані від сервера, такі як HTML-код сторінки, JSON-дані або будь-які інші типи вмісту, які можуть бути пов'язані з виконаним запитом.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування ПЗ

На діаграмі прецедентів (рисунок 3.1) можна побачити можливі дії користувача web застосунка.

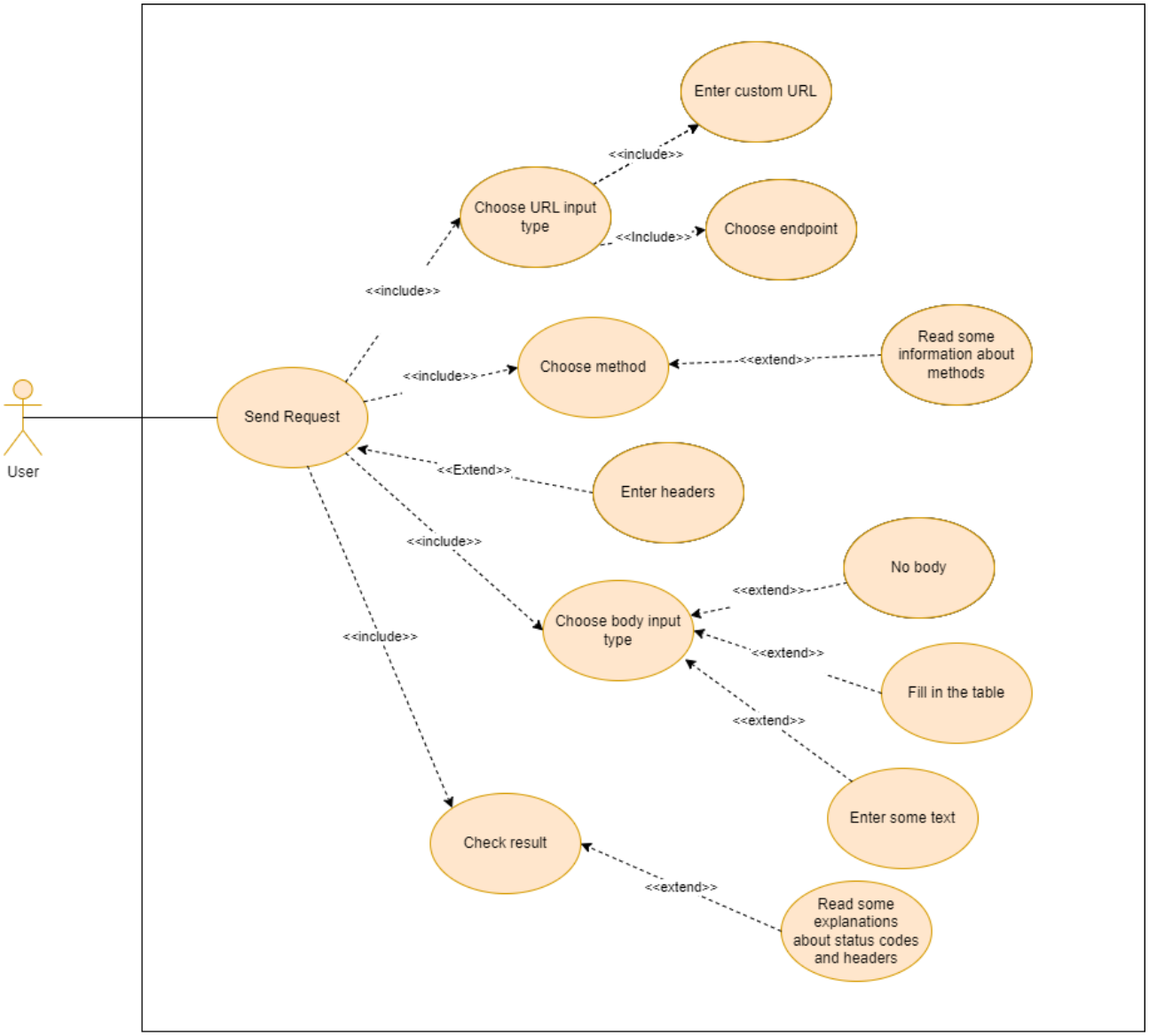

Рисунок 3.1 - Діаграма прецедентів

Користувач може відправити HTTP запит. Для цього обов’язково потрібно виконати наступні кроки:

1. Обрати тип введення URL. У користувача є два варіанта:

- Ввести будь-який URL
- Обрати кінцеву точку з випадаючого списку.

2. Обов’язково треба вказати метод

При виборі методу можна відкрити віконце з теоретичною частиною та ознайомитися з усією необхідною інформацією стосовно обраного методу.

3. Також при необхідності можна додати заголовки до запиту.

4. Необхідно обрати зручний тип введення тіла запиту, якщо є така необхідність.

Вже після цього користувач натискає на кнопку «Відправити запит» та отримує відповідь від сервера.

Можна побачити, у якому виді запит побачив сервер. Та безпосередньо відповідь від сервера.

На діаграмі компонентів (рисунок 3.2) можна побачити внутрішню взаємодію web застосунка.

Робота веб інтерфейсу в більшій частині лягає на контролери, які успадковують базові інтерфейси контролера ASP.NET core. CrudController.cs відповідає за можливість створення, оновлення, отримання та видалення користувачів. RequestDetailsController.cs відповідає за відображення HTTP запиту у тому вигляді, як це бачить свервер.

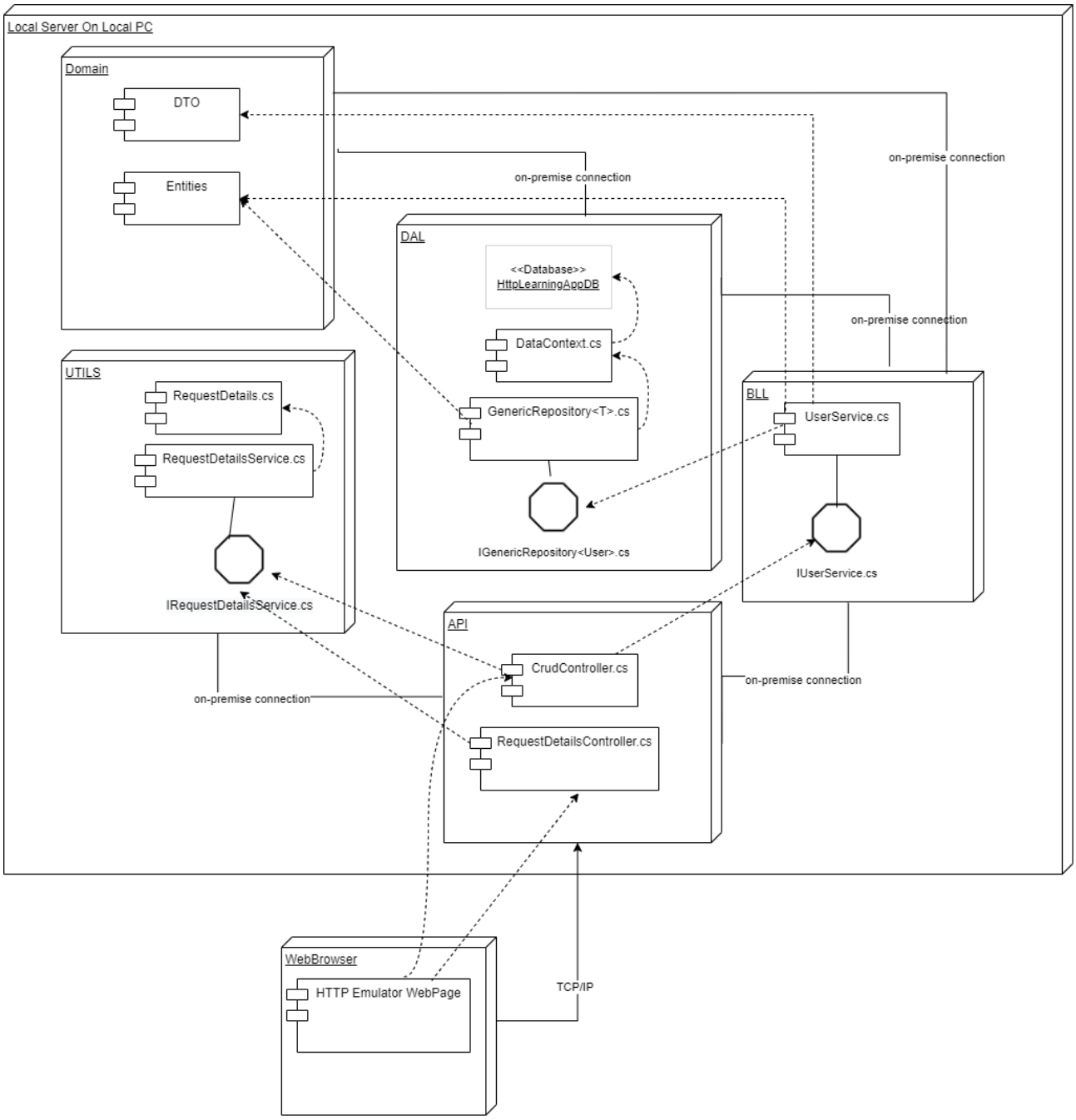

Рисунок 3.2 – Діаграма компонентів

3.2 Проектування архітектури ПЗ

Основні вимоги до архітектури кваліфікаційної роботи - система повинна бути легко масштабованою та повинна справлятися з великими навантаженнями. Для заданих потреб найбільше підходить onion архітектура. Схема якої зображена на рисунку 3.3.

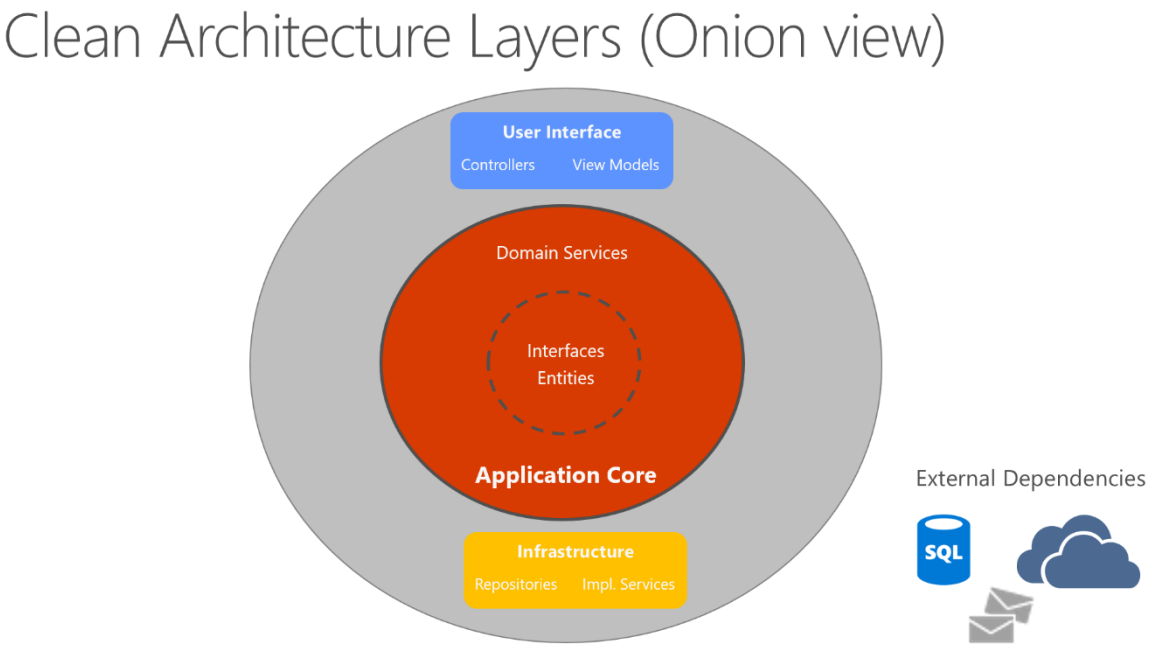

Рисунок 3.3 – Схема структури onion архітектури

Особливістю onion-архітектури є поділ програми на рівні. При чому є один незалежний рівень, який знаходиться в центрі архітектури. Від цього рівня залежить другий рівень, від другого - третій і так далі. Тобто виходить, що навколо першого незалежного рівня нашаровується другий-залежний. Навколо другого нашаровується третій, який також може залежати і від першого. Образно це може бути виражено у вигляді цибулі, в якої також є серцевина, навколо якого нашаровуються всі інші верстви, аж до лушпиння. [6]. На рисунку 3.4 наведена діаграма розгортання системи. На ній видно на які рівні та компоненти поділена система.

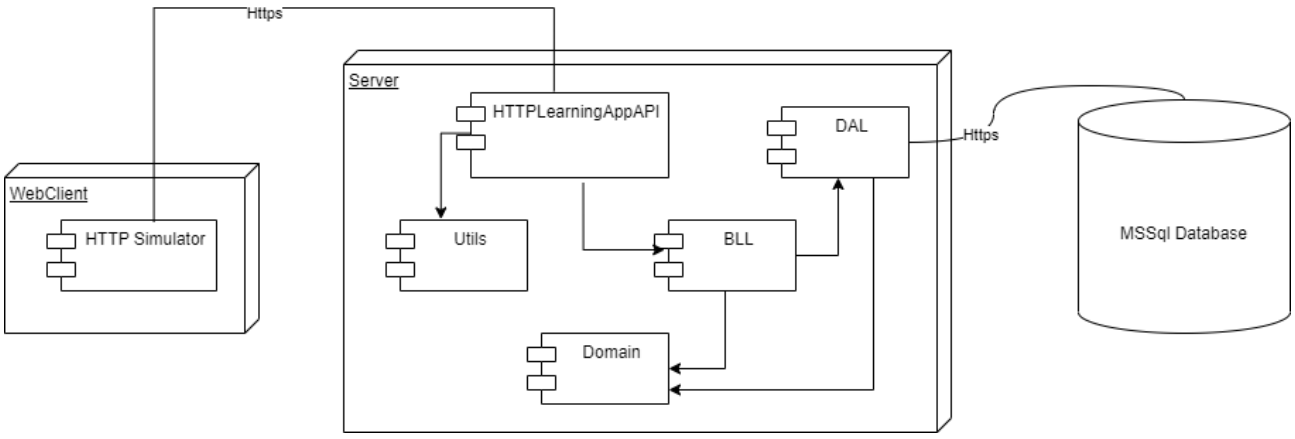

Рисунок 3.4 – Діаграма розгортання

Згідно методології onion архітектури компонент Domain знаходиться у самому центрі системи. Там розташовані сутності, класи DTO та основні інтерфейси. Рівень BLL також входить в ядро системи, в ньому реалізуються сервіси, тобто основна логіка системи. А ось зв’язок між базою даних та системою відбувається лише через DAL і цей рівень знаходиться за яром системи, а саме в рівні Infrastructure. Бо система не повинна бути залежною від конкретної реалізації бази даних та CRUD задачі. Сервіси працюють з даними через інтерфейси які поєднані з конкретною реалізацією через Dependency injection. Саме це дозволяє досягти принципу слабкої зв’язаності, через те що ми можемо в любий момент змінити СУБД і нам не потрібно буде нічого змінювати в самому ядрі програмного забезпечення.

HTTPLearningAppAPI лише приймає запити від користувачів та делегує обробку отриманих даних до рівня сервісів та згодом відправляє відповіді користувачам.

В даній конкретній реалізації в якості СУБД для розробки було обрано Microsoft SQL Server. Microsoft SQL Server — система управління базами даних, яка розробляється корпорацією Microsoft. Але при кінцевому розгортанні програмного зебезпечення повинна буде використовуватися Microsoft Azure SQl Database. На даний момент хмарні технології набирають пікову популярність та допомагають розробникам та власникам бізнесу зберігати дані, хостити

програмне забезпечення, створювати мікро сервісні рішення та об’єднувати їх в одну систему без необхідності покупки та налаштування фізичного обладнання.

Також користувачі хмарних сервісів отримують можливість платити лише за ту кількість ресурсів яку вони використовують і миттєво збільшувати потужність у разі пікового навантаження або за розкладом.

Для зв’язку між хмарною базою даних та між API та веб клієнтом використовується протокол HTTPS. Це безпечний протокол передачі гіпертексту) - це розширення протоколу HTTP, що підтримує шифрування за допомогою криптографічного протоколу TLS.

Отже з вище перерахованого можна зробити висновок що розроблена програмна система дійсно є добре масштабованою та захищеною від атак.

Більш детальна схема компонування програмної системи зображена на рисунку 3.5.

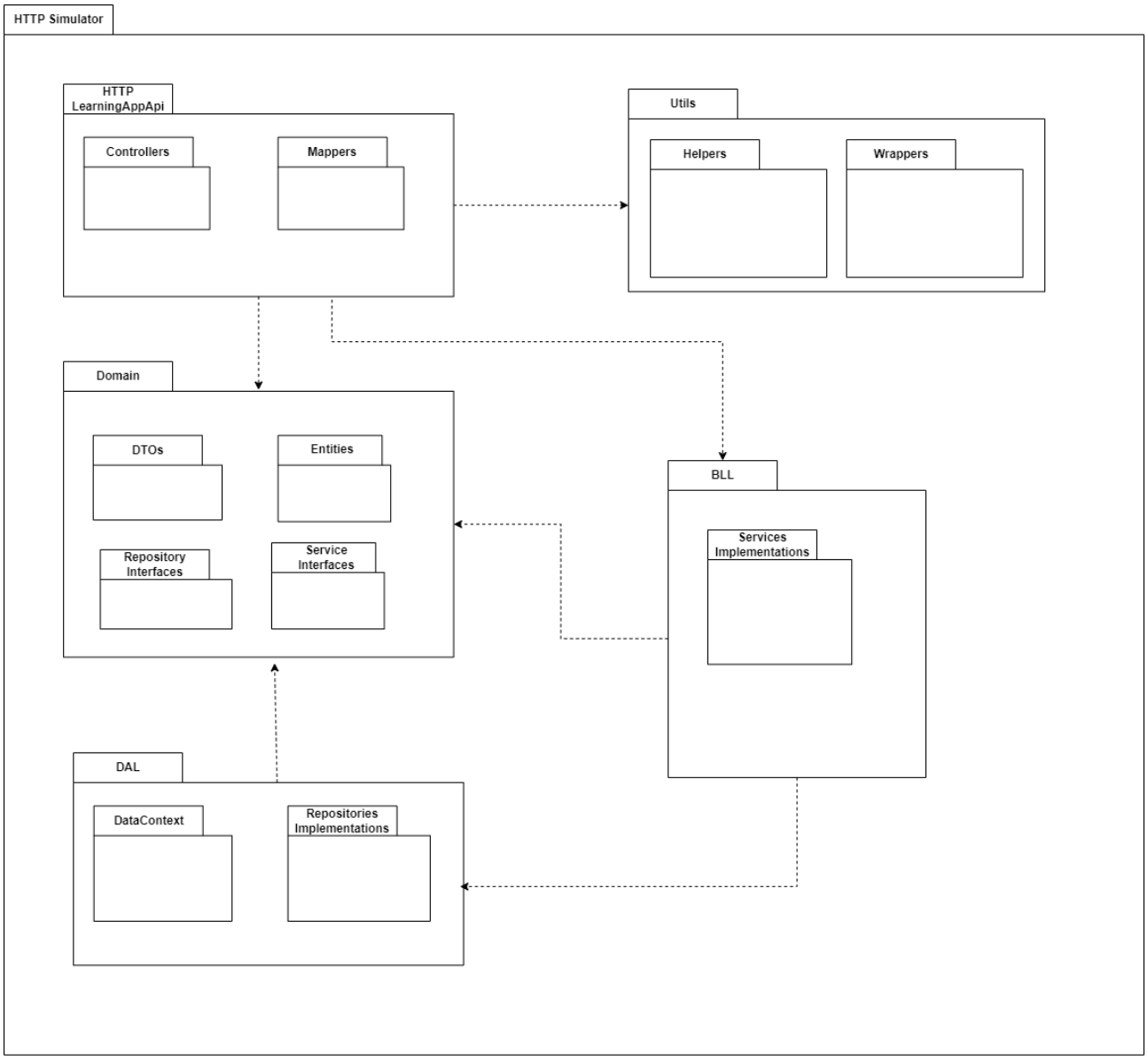

Рисунок 3.5 – Діаграма пакетів

Згідно до наведеної діаграми серверна частина ділиться на 5 рівнів що забезпечують реалізацію шаблонів розподілення обов’язків між класами, а саме низьку зв’язність та високе зачеплення [10].

Основна обробка логіки здійснюється за допомогою 3 компонентів, а саме Domain, BLL – ядро розробленої системи та DAL (з реалізацією роботи з конкретною СУБД) що знаходиться на рівні Infrastructures. Детальніше взаємодія між цими компонентами показана на рисунку 3.6.

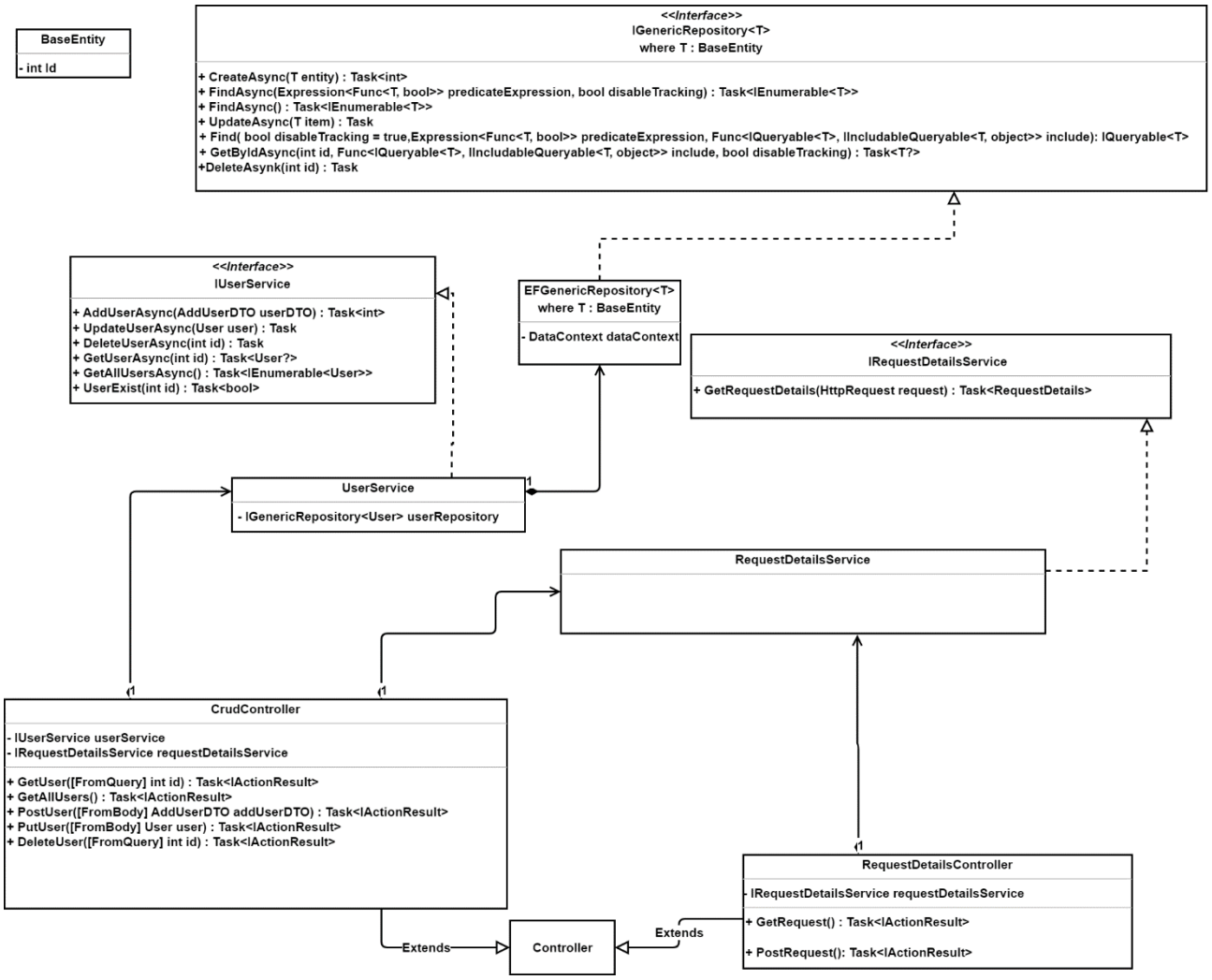

Рисунок 3.6 – Діаграма класів

До першого компоненту належить HTTPLearningAppApi. Це загальний рівень взаємодії з системою. Веб-клієнт взаємодіє з системою посилаючи запити до HTTPLearningAppApi та отримує відповідь оброблену сервісами. На діаграмі класів цей компонент зображений у вигляді контролерів що наслідують базовий контролер. В контролерах не виконується жодної логіки, вони лише приймають параметри та делегують виконання рівню сервісів. Одним з полів класу контролера є як раз екземпляр необхідного класу-сервісу.

До другого компоненту належить Business layer. Це рівень бізнес-логіки всіх сервісів. Він містить набір сервісів з якими взаємодіє API layer. Сервіси виконують основну логіку системи, в них реалізовані найбільш важливі і

основоположні алгоритми. В класі сервісів зберігається екземпляр необхідного репозиторію для роботи з даними.

Останнім компонентом є Data Access layer. Це рівень доступу до даних. Він містить реалізацію CRUD задачі. Єдиний рівень який працює з БД напряму через Entity Framework Core. DAL містить в собі лише DataContext та реалізацію інтерфейсу репозиторія, який визначений в Domain. Архітектура спроектована таким чином що усі сутності до яких необхідний доступ в базі даних наслідують клас BaseEntity який гарантує що у всіх сутностей є цілочисловий Id. Тому ми маємо лише один узагальнений інтерфейс IGenericRepository<T> та його реалізацію EFGenericRepository<T>, де T – BaseEntity. Це значно зменшує повторення коду. Для передачі підключення до БД та передачі залежностей між інтерфейсами та реалізаціями використовується Dependency injection. Данна передача залежностей забезпечує слабку зв’язаність бізнес-логіки з DAL (Inversion of control), адже щоб змінити технологію зберігання даних розробникам потрібно лише наново реалізувати інтерфейси для нової технології та змінити відношення інтерфейс-реалізація [9].

Для реалізації цієї частини проекту було обрано сучасну технологію ASP.NET Core Web Api. Цей фреймворк повністю відповідає потребам методології REST системи [3].

Додатково повинно бути використано Entity Framework Core для взаємодії з базою даних.

3.3 Проектування структури зберігання даних

У даному розділі буде розглянуто функціонал CRUD контролера, який дозволяє працювати з сутністю User в базі даних. Контролер надає базові операції створення, оновлення, отримання та видалення записів користувачів.

Створення користувача (Create): Контролер має метод, який дозволяє створювати нового користувача в базі даних. Цей метод приймає вхідні дані, такі як ім'я та електронна пошта користувача, та зберігає їх у базі даних. Наприклад,

при отриманні POST-запиту з даними нового користувача, контролер перевіряє правильність та повноту отриманих даних, створює новий об'єкт User з відповідними значеннями та зберігає його у базі даних.

Оновлення користувача (Update): Контролер надає можливість оновити існуючого користувача в базі даних. Цей метод отримує ідентифікатор користувача та нові дані про нього, такі як змінене ім'я чи електронна пошта, і змінює відповідний запис у базі даних. Наприклад, при отриманні PUT-запиту з ідентифікатором користувача та новими даними, контролер знаходить відповідний запис у базі даних та оновлює його значення згідно отриманих даних.

Отримання користувача (Read): Контролер надає можливість отримати деталі про конкретного користувача. Цей метод приймає ідентифікатор користувача та повертає інформацію про нього, таку як ім'я, електронна пошта. Наприклад, при отриманні GET-запиту з ідентифікатором користувача, контролер знаходить відповідний запис у базі даних та повертає його значення у відповідь на запит.

Видалення користувача (Delete): Контролер надає можливість видалити існуючого користувача з бази даних. Цей метод отримує ідентифікатор користувача та видаляє відповідний запис з бази даних. Наприклад, при отриманні DELETE-запиту з ідентифікатором користувача, контролер знаходить відповідний запис у базі даних та видаляє його.

Функціонал CRUD контролера дає змогу ефективно управляти користувачами у базі даних. Використання цього контролера спрощує взаємодію з базою даних та забезпечує зручний доступ до операцій створення, оновлення, отримання та видалення даних користувачів.

Окрім цього, використання CRUD контролера є імітацією реального контролера з функціоналом для роботи з даними користувачів. Це дозволяє нашому серверу емулювати роботу контролера та продемонструвати деталі отриманого запиту. Такий підхід є корисним для навчальних цілей, тестування або створення прототипів програмного забезпечення.

3.4 Приклади найцікавіших алгоритмів та методів

RequestDetailsService є класом, який виконує роль сервісу для отримання деталей запиту веб-застосунка. Його основна відповідність полягає в тому, щоб зробити аналіз запиту та витягнути необхідну інформацію з нього, таку як заголовки, тип вмісту та тіло запиту.

Перший крок полягає в отриманні заголовків запиту. За допомогою методу ToDictionary з параметрами header => header.Key та header => header.Value.ToString(), заголовки запиту перетворюються на словник, де ключами є назви заголовків, а значеннями - їх вміст.

Далі, за допомогою властивості ContentType, отримується тип вмісту запиту. Це дозволяє визначити, який формат використовується для передачі даних у тілі запиту.

Тоді, якщо тіло запиту не є порожнім, воно конвертується у рядок. Для цього перевіряється, чи не є request.Body порожнім, і якщо це не так, використовується StreamReader для зчитування тіла запиту із кодуванням UTF-8. Отриманий рядок зберігається в змінній body, яка пізніше використовується для подальшої обробки.

Для різних типів вмісту запиту (наприклад, JSON, текст, XML, форма x-www-form-urlencoded), виконується відповідна обробка тіла запиту. Залежно від типу вмісту, використовуються відповідні бібліотеки та методи. Наприклад, для application/json тіло запиту десеріалізується в об'єкт, а потім знову серіалізується назад у JSON-рядок.

Якщо виникає помилка під час обробки тіла запиту, змінна body отримує значення "Failed to read the request body."

На основі отриманих даних про запит формується об'єкт RequestDetails, який містить інформацію про метод, схему, хост, шлях, рядок запиту, заголовки, тип вмісту та тіло запиту. Цей об'єкт повертається як результат роботи методу GetRequestDetails.

Отже, RequestDetailsService виконує важливу функцію аналізу запиту та отримання деталей про нього, що допомагає розуміти, як клієнти взаємодіють з веб-застосунком та які дані вони передають.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Перші кроки користувача

Користувач одразу потрапляє до головної веб сторінки, яка має такий загальний вигляд (Рисунок 4.1)

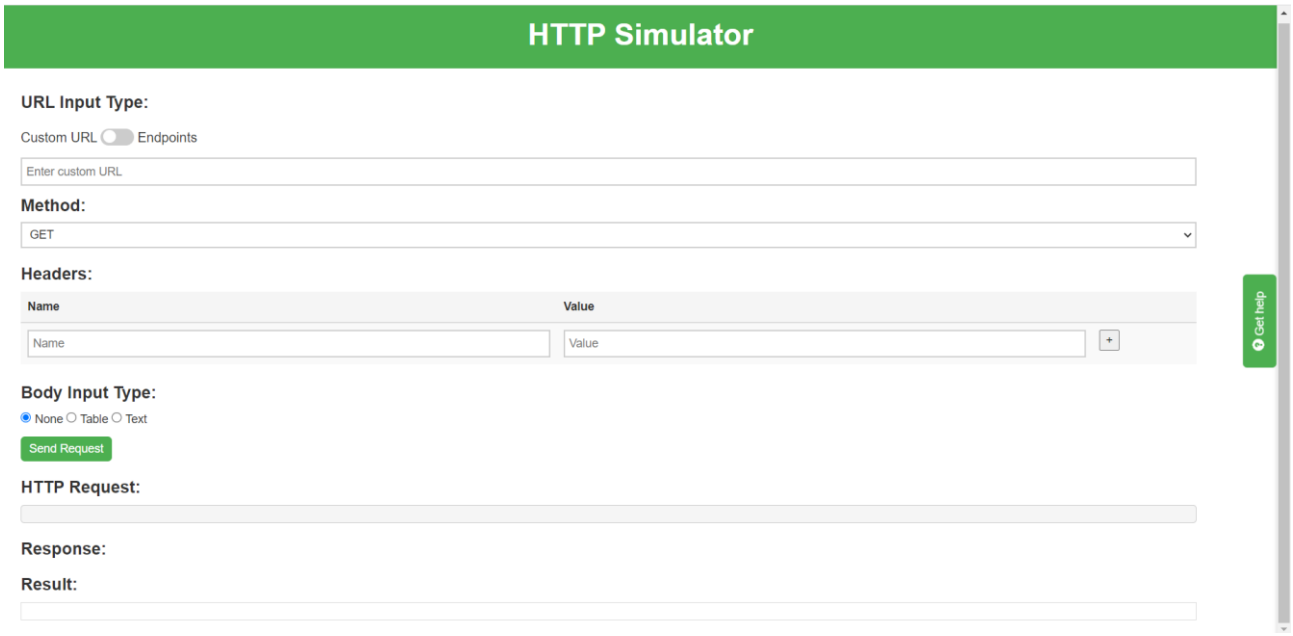

Рисунок 4.1 – Початкова сторінка

Далі користувач має змогу обрати тип введення URL. В даному випадку треба використовувати кнопку-перемикач, що змінює тип введення в залежності від вибору користувача.

Є два варіанта:

1) Ввести URL самостійно (рисунок 4.2). В такому випадку немає жодних обмежень.

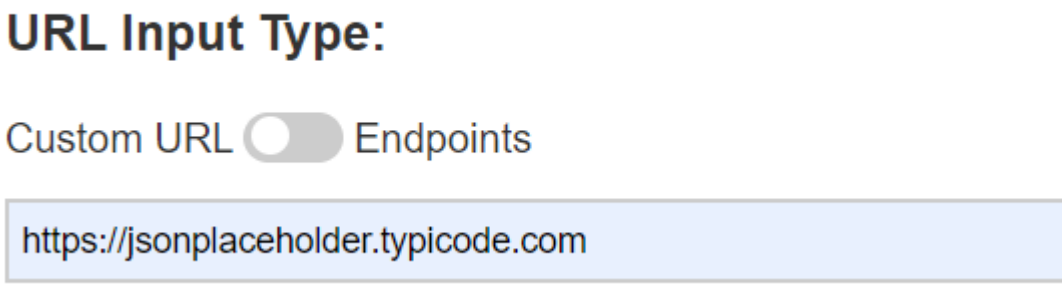

Рисунок 4.2 – Приклад введеного URL

2) Обрати кінцевий шлях серед запропонованих (рисунок 4.3). Користувач обирає з випадаючого списку.

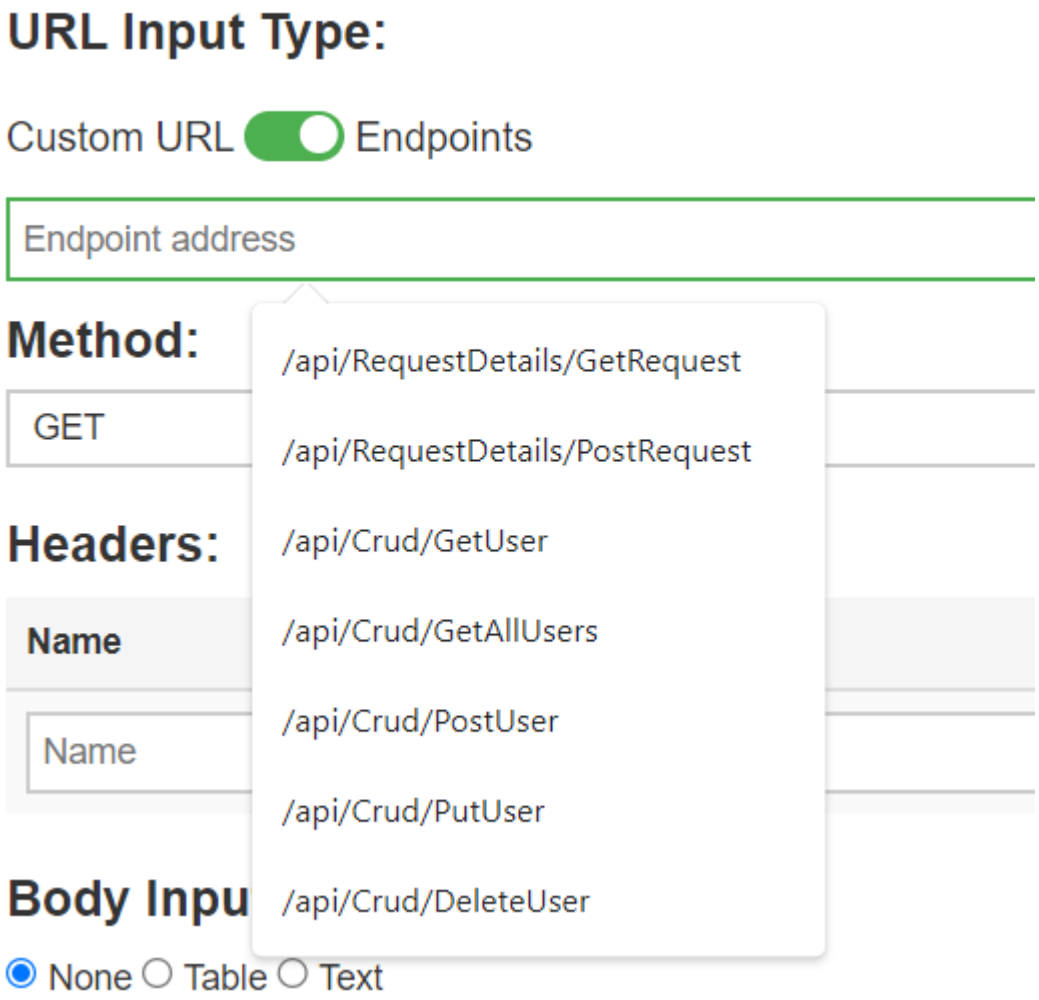

Рисунок 4.3 – Випадаючий список з кінцевими шляхами

Після цього користувач має обрати метод з випадаючого списку (рисунок 4.4).

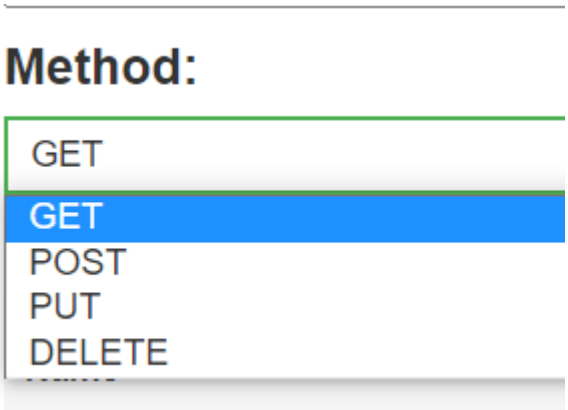

Рисунок 4.4 – Випадаючий список з можливими методами

У разі, якщо користувач не знає, для чого використовується якийсь з перелічених методів, він може натиснути на кнопку “Get help”. В результаті з’явиться віконце з теоретичною частиною про обраний метод (рисунок 4.5).

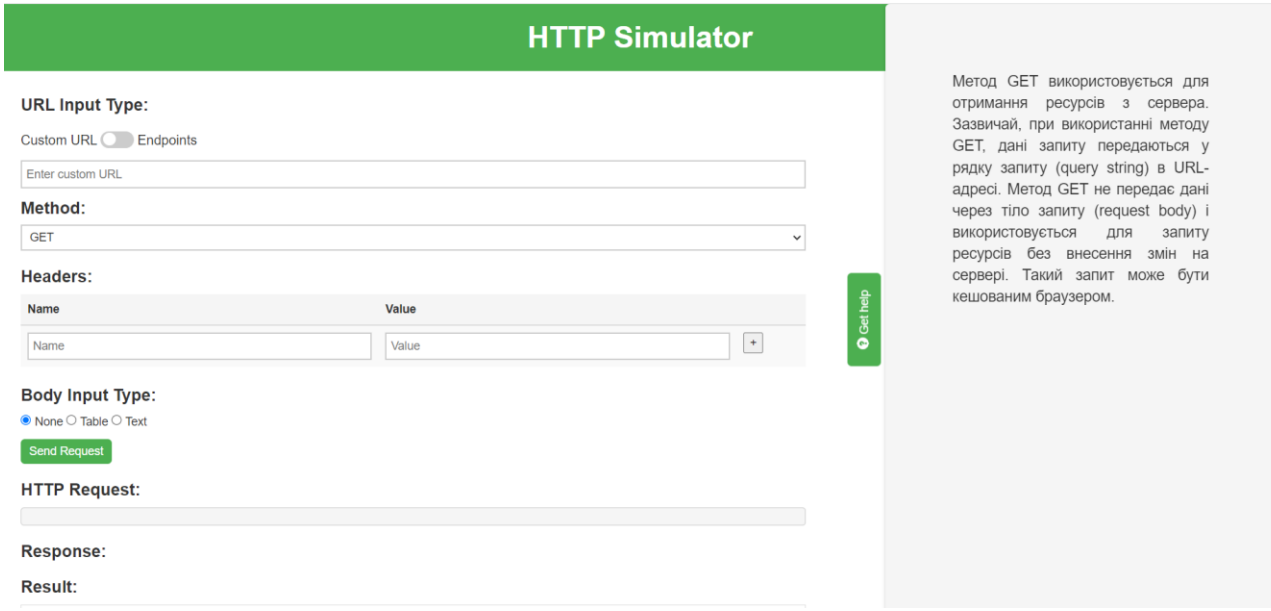

Рисунок 4.5 – Демонстрація віконця з теоретичною частиною для методу GET

При зміні метода також автоматично оновлюється віконце з теоретичною частиною про обраний метод (рисунок 4.6). Ось приклад інформації при обраному методі Delete.

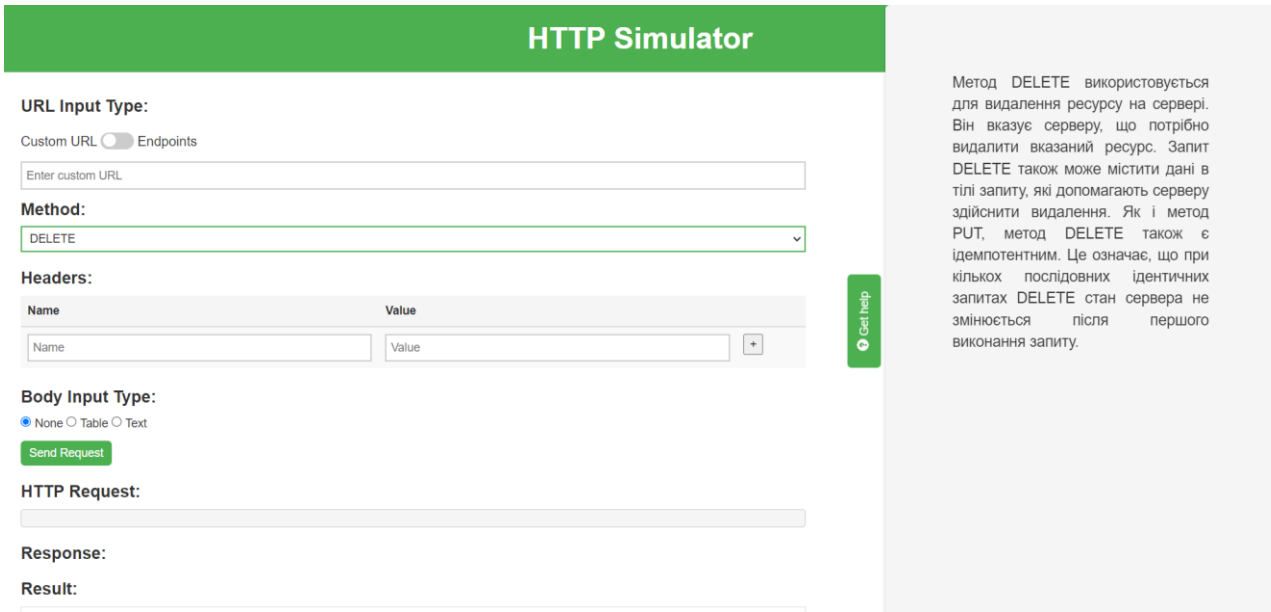

Рисунок 4.6 - Демонстрація оновленого віконця з теоретичною частиною для методу DELETE

Потім при необхідності можна додати заголовки, які теж з’являються у випадаючому списку. Випадаючий список з заголовками автоматично оновлюється при зміні методу.

Наведемо декілька прикладів:

1) Обираємо метод Get. При введені заголовків з’являється випадаючий список з наступними заголовками (Рисунок 4.7)

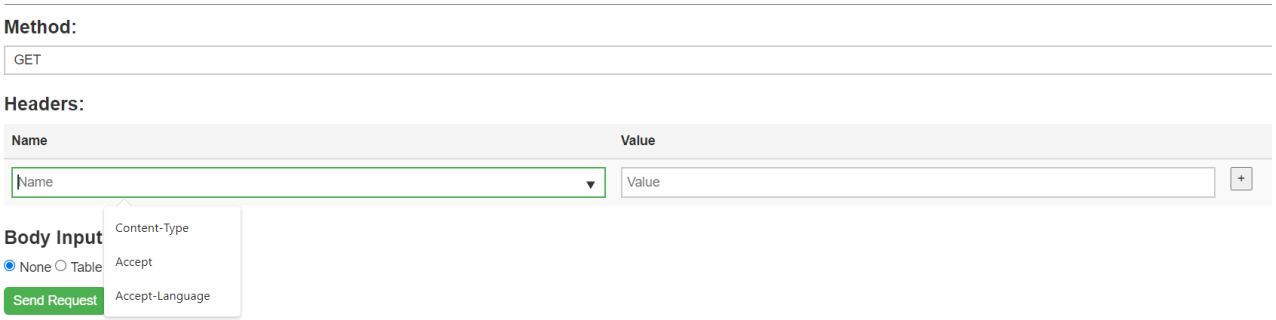

Рисунок 4.7 - Випадаючий список з заголовками для методу Get

2) Обираємо метод Post. При введені заголовків з’являється випадаючий список з наступними заголовками (Рисунок 4.6)

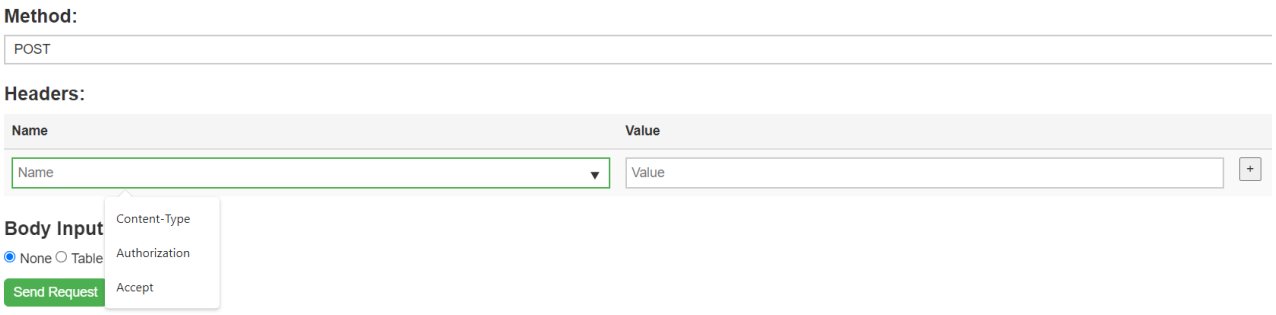

Рисунок 4.6 - Випадаючий список з заголовками для методу Post

Також можна додавати нові заголовки, натискаючи на кнопку “+” (Рисунок 4.7)

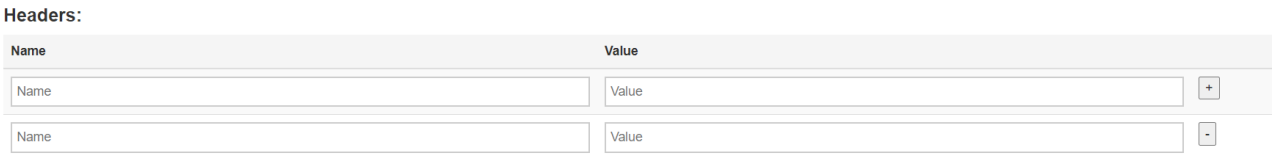

Рисунок 4.7 – Демонстрація додавання нового заголовку

Наступним кроком буде вибір типу введення тіла запиту.

Можна обрати три різних варіанта:

1) Без тіла запиту (Рисунок 4.8)

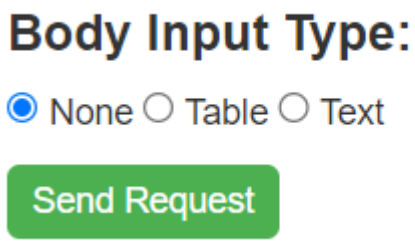

Рисунок 4.8 - Без тіла запиту

2) Тіло запиту у вигляді таблиці (Рисунок 4.9)

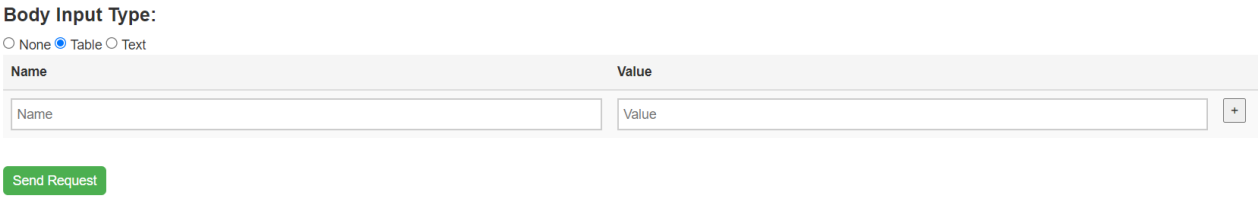

Рисунок 4.9 – Тіло запиту у вигляді таблиці

3) Тіло запиту у вигляді тексту (Рисунок 4.10)

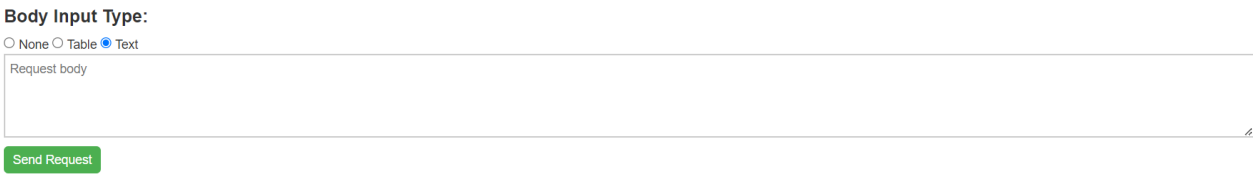

Рисунок 4.10 - Тіло запиту у вигляді тексту

Після цього треба натиснути на кнопку “Send Request”.

В результаті, користувач отримує відповідь від сервера (рисунок 4.11)

Рисунок 4.11 – Відповідь від сервера

4.2 Використання HTTP протоколу для роботи з Базою Даних

В даному розділі будуть розглянуті різні аспекти використання HTTP протоколу для роботи з базою даних, зокрема методи HTTP запитів, такі як GET, POST, PUT та DELETE, які використовуються для створення, зчитування, оновлення та видалення даних у базі даних.

1. Розберемо використання методу Post для створення нового запису про користувача в базі даних.

Цей метод використовується для створення нового запису в базі даних. У цьому випадку ідентифікатор користувача вказувати не треба.

1) Для надсилання такого HTTP запиту ми обираємо необхідний кінцевий шлях з випадаючого списку (рисунок 4.12).

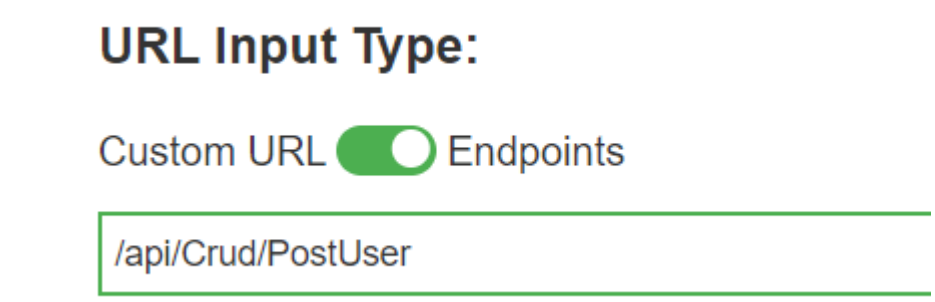

Рисунок 4.12 – Обраний кінцевий шлях для методу POST

2) Далі обираємо метод Post (рисунок 4.13).

Рисунок 4.13 – Обраний метод POST

3) Додаємо заголовок з відповідними параметрами (Рисунок 4.14).

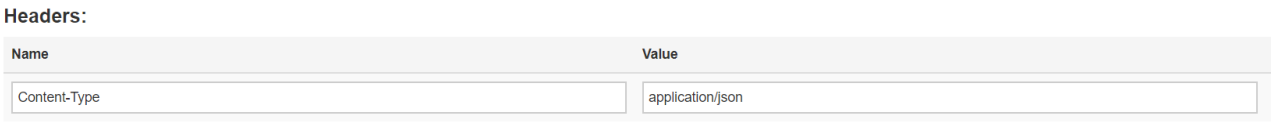

Рисунок 4.14 – Демонстрація необхідних заголовків

4) Обираємо тип введення даних тіла запиту text (Рисунок 4.15).

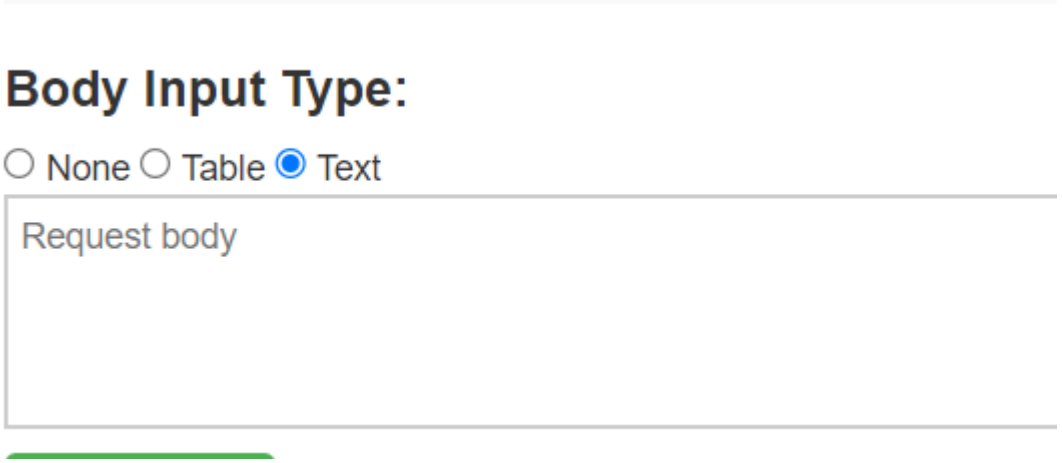

Рисунок 4.15 – Обрано тип введення даних тіла запиту text

5) Та вводимо дані про нового користувача у форматі application/json (Рисунок 4.16).

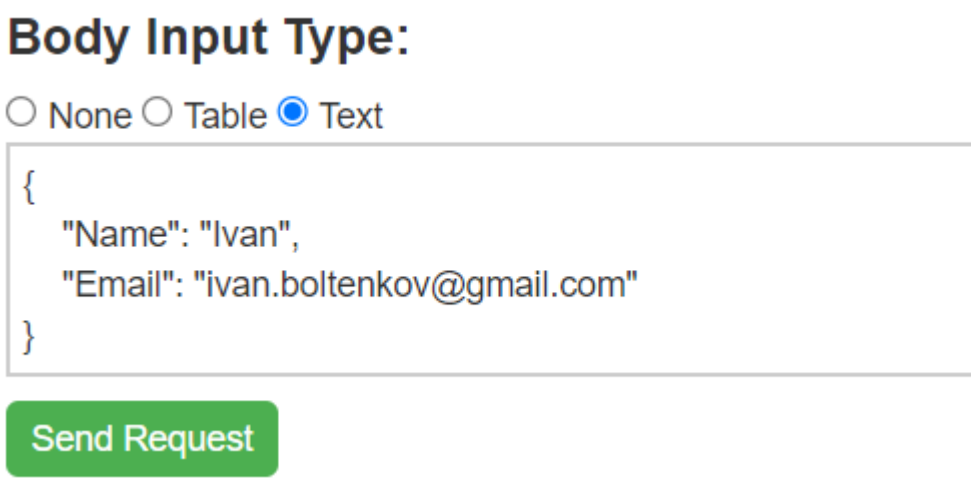

Рисунок 4.16 – Введені дані про нового користувача у форматі application/json

6) Натискаємо на кнопку “Send Request” та отримуємо відповідь від сервера. Тут ми побачимо, як сервер бачить наш запит (Рисунок 4.17). Та відповідь від сервера (Рисунок 4.18).

The request view on the server:

method: "POST"
scheme: "https"
host: "localhost:44319"
pathBase: ""
path: "/api/Crud/PostUser"
queryString: ""

Accept	application/json, text/plain, */*
Accept-Encoding	gzip, deflate, br
Accept-Language	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection	close
Content-Length	63
Content-Type	application/json
Host	localhost:44319
Referer	https://localhost:44319/
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36
sec-ch-ua	"Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"
sec-ch-ua-mobile	?0
sec-ch-ua-platform	"Windows"
origin	https://localhost:44319
sec-fetch-site	same-origin
sec-fetch-mode	cors
sec-fetch-dest	empty

contentType: "application/json"

```
Body:
{
  "name": "Ivan",
  "email": "ivan.boltenkov@gmail.com"
}
```

Рисунок 4.17 – Вікно, що відображає, як сервер бачить наш запит

Response:

Status: 201
Headers:

content-type	application/json; charset=utf-8
date	Sat, 17 Jun 2023 13:05:07 GMT
location	/Crud/GetUser?id=11
server	Microsoft-IIS/10.0
x-powered-by	ASP.NET

```
Body:
11
```

Рисунок 4.18 – Вікно, що відображає відповідь від сервера

В результаті ми отримали код 201, що свідчить про успішно виконаний запит та створення нового ресурса. Також в тілі відповіді ми отримали номер, що є ідентифікатором нового користувача.

Тепер при необхідності отримати, оновити або видалити дані про цього користувача, ми будемо вказувати цей ідентифікатор.

2. Розберемо використання методу Put для оновлення інформації про вже існуючого користувача. Цей метод використовується для оновлення інформації про конкретного користувача у базі даних. В такому разі треба зазначити ідентифікатор користувача, що був наданий користувачу під час створення нового запису в базі даних. Але також цей метод можна застосовувати для створення нового запису в базі даних. У цьому випадку ідентифікатор користувача вказувати не треба.

1) Для надсилання такого HTTP запиту ми обираємо необхідний кінцевий шлях з випадаючого списку (рисунок 4.19).

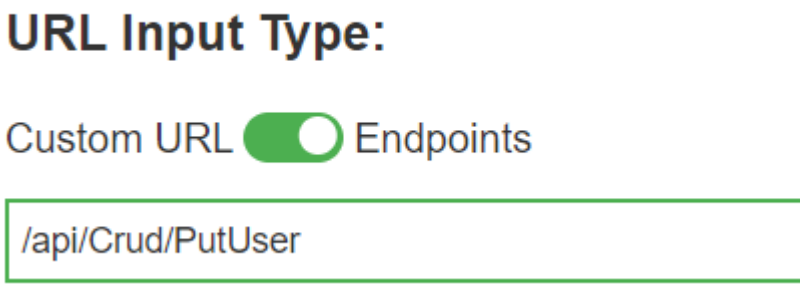

Рисунок 4.19 – Демонстрація обраного кінцевого шляху для методу PUT

2) Далі обираємо метод Put (рисунок 4.20).

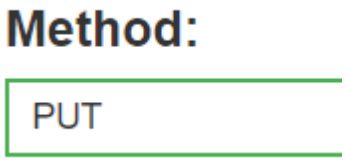

Рисунок 4.20 – Демонстрація обраного методу PUT

3) Додаємо заголовок з відповідними параметрами (рисунок 4.21)

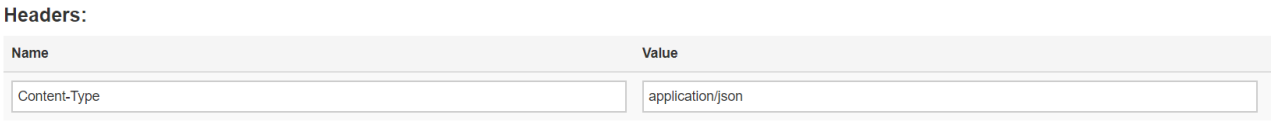

Рисунок 4.21 – Демонстрація необхідних заголовків

4) Обираємо тип введення даних тіла запиту text (рисунок 4.22)

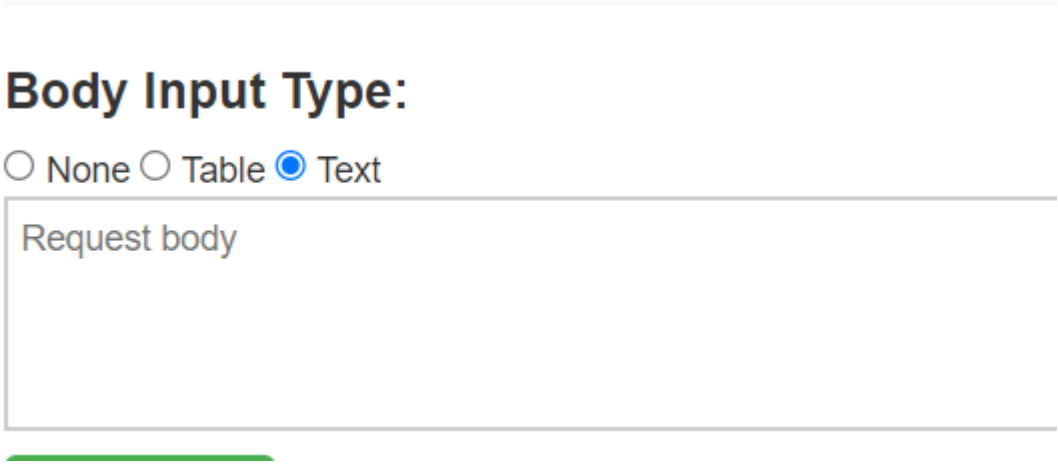

Рисунок 4.22 – Демонстрація обраного типу введення даних тіла запиту text

5) Та вводимо оновлені дані про користувача у форматі application/json. Наприклад можемо замінити email. Але також треба обов'язково вказати ідентифікатор користувача, щодо якого ми вносимо зміни (рисунок 4.23).

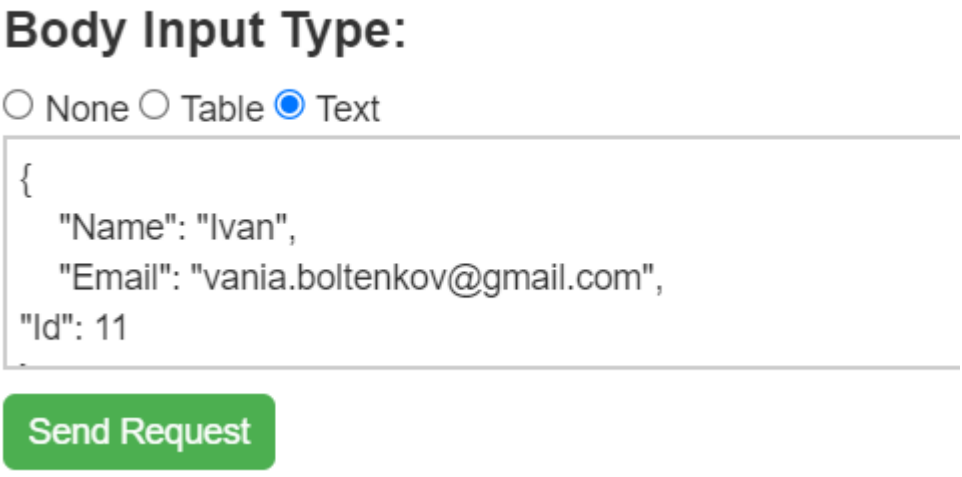

Рисунок 4.23 – Демонстрація введених оновлених даних про користувача

6) Натискаємо на кнопку “Send Request” та отримуємо відповідь від сервера (рисунок 4.24).

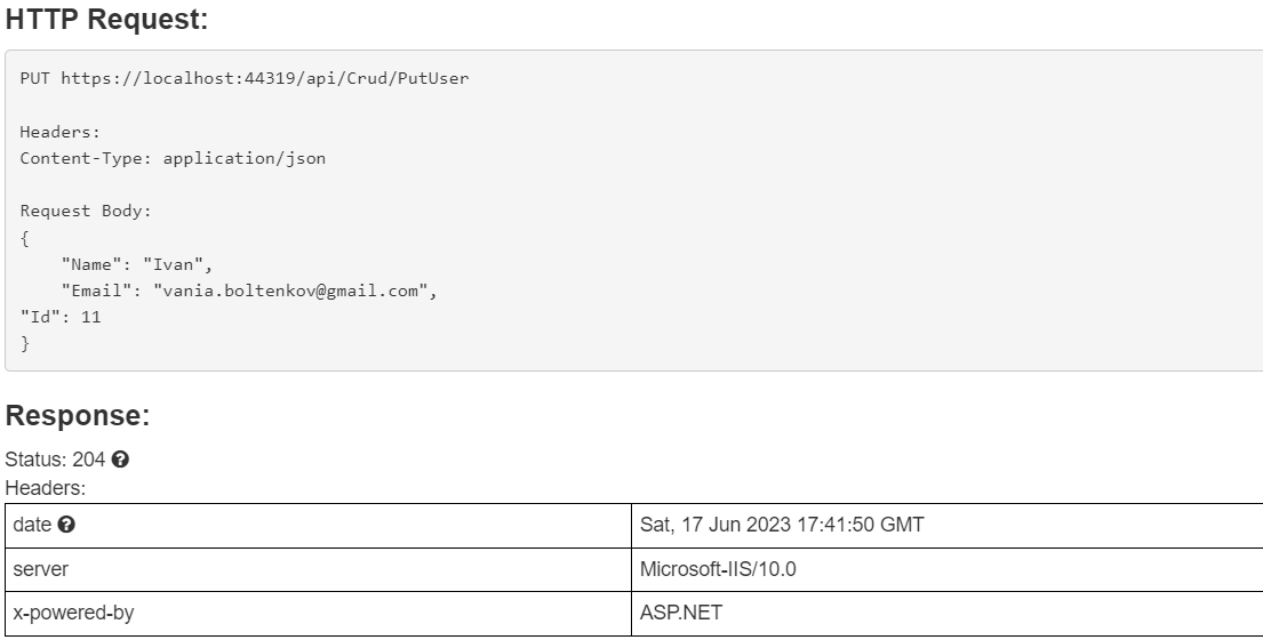

Рисунок 4.24 – Демонстрація відповіді від сервера

В результаті ми отримали код 204.

Цей код, також відомий як "No Content" (немає вмісту), є одним з стандартних кодів статусу HTTP, що використовується для відповіді сервера на клієнтський запит. Основне значення коду 204 полягає в тому, що сервер успішно обробив запит клієнта, але відповідь не містить додаткового вмісту або тіло.

3.Розберемо використання методу Get для отримання інформації про вже існуючого користувача. Цей метод використовується для отримання інформації про конкретного користувача у базі даних. В такому разі треба зазначити ідентифікатор користувача, що був наданий користувачу під час створення нового запису в базі даних

1) Для надсилання такого HTTP запиту ми обираємо необхідний кінцевий шлях з випадаючого списку. Але потім додаємо “?Id=” та номер ідентифікатор користувача (рисунок 4.25).

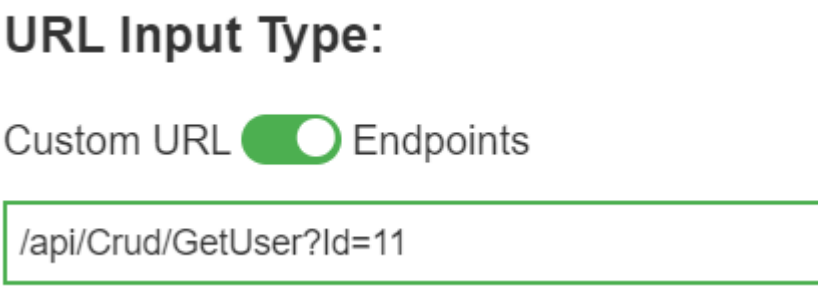

Рисунок 4.25 – Демонстрація обраного кінцевого шляху з додаванням ідентифікатора користувача

2) Далі обираємо метод Get (рисунок 4.26).

Рисунок 4.26 – Демонстрація обраного методу GET

3) Обираємо тип введення даних тіла запиту none (рисунок 4.27).

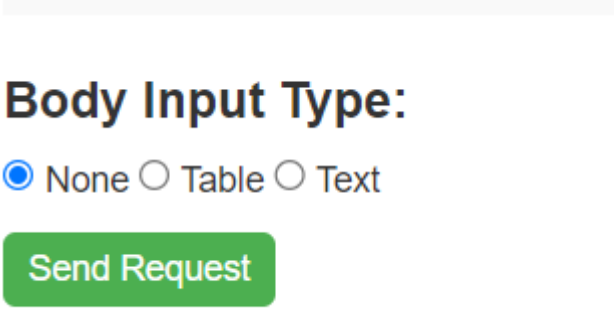

Рисунок 4.27 – Демонстрація обраного типу введення даних тіла запиту none

4) Натискаємо на кнопку “Send Request” та отримуємо відповідь від сервера. Тут ми побачимо, як сервер бачить наш запит (Рисунок 4.28). Та відповідь від сервера (Рисунок 4.29).

HTTP Request:

```
GET https://localhost:44319/api/Crud/GetUser?Id=11

Headers:

Request Body:
```

The request view on the server:

method: "GET"
scheme: "https"
host: "localhost:44319"
pathBase: ""
path: "/api/Crud/GetUser"
queryString: "?Id=11"

Accept	application/json, text/plain, */*
Accept-Encoding	gzip, deflate, br
Accept-Language	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection	close
Host	localhost:44319
Referer	https://localhost:44319/
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36
sec-ch-ua	"Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"
sec-ch-ua-mobile	?0
sec-ch-ua-platform	"Windows"
sec-fetch-site	same-origin
sec-fetch-mode	cors
sec-fetch-dest	empty

contentType: null

Рисунок 4.28 - Вікно, що відображає, як сервер бачить наш запит

Response:

Status: 200
Headers:

content-type	application/json; charset=utf-8
date	Sat, 17 Jun 2023 17:50:48 GMT
server	Microsoft-IIS/10.0
x-powered-by	ASP.NET

```
Body:
{
  "name": "Ivan",
  "email": "vania.boltenkov@gmail.com",
  "id": 11
}
```

Рисунок 4.29 - Вікно, що відображає відповідь від сервера

В результаті ми отримали код 200.

Код 200 HTTP, також відомий як "OK", є одним з найпоширеніших кодів статусу HTTP і використовується для позначення успішної відповіді сервера на клієнтський запит. Код 200 підтверджує, що запит був успішно оброблений і сервер повертає відповідь з очікуваними даними. В результаті, ми отримали інформацію про користувача. Та ми можемо бачити, що інформація була дійсно оновлена.

Отже, ми розглянули приклади використання методів, таких як Post, Put та Get. Використання HTTP протоколу для взаємодії з базою даних дозволяє зручно та ефективно керувати даними, використовуючи стандартні HTTP методи та формати даних. Цей підхід забезпечує простоту, переносимість та широкі можливості для взаємодії з базою даних у розподілених системах.

4.3 Реалізація рівня DAL та використання Dependency Injection

DAL – це рівень програмної системи в якому відбувається взаємодія з конкретною СУБД. На цьому рівні створюється data context, який є джерелом усіх об’єктів відображених через зв’язок з базою даних.

Як було вказано у розділі розробки архітектури, розроблена програмна система використовує за основу модель onion архітектури. Отже спочатку ми створюємо інтерфейс репозиторію на рівні Domain – тобто в самому ядрі системи. Наведемо приклад створеного інтерфейсу:

```
public interface IGenericRepository<T> where T : BaseEntity
    {
        Task<int> CreateAsync(T entity);

        Task<IEnumerable<T>> FindAsync(
            Expression<Func<T, bool>> predicateExpression = null,
            bool disableTracking = true);
```

```
        IQueryable<T> Find(
            bool disableTracking = true,
            Expression<Func<T, bool>> predicateExpression = null,
            Func<IQueryable<T>, IIncludableQueryable<T, object>> include =
null);

        Task DeleteAsync(int id);

        Task UpdateAsync(T entity);

        Task UpdateRangeAsync(IEnumerable<T> entities);

        Task<T> GetByIdAsync(
            int id,
            Func<IQueryable<T>,
            IIncludableQueryable<T, object>> include = null,
            bool disableTracking = true);

        Task DeleteRangeAsync(IEnumerable<T> entities);
    }
```

Є декілька цікавих особливостей в цьому інтерфейсі. По перше завдяки тому, що всі сутності в базі даних наслідують клас BasicEntity, що в свою чергу гарантує що у всіх сутностей є цілочисельний id був створений узагальнений інтерфейс. Це дає значні перевагу з точки зору зменшення повторення коду.

По друге методи є асинхронними – це надає змогу не блокувати потік під час оброки запиту користувача.

Наведемо приклад реалізації одного з методів інтерфейсу узагальненого доступу до даних:

```
public IQueryable<T> Find(bool disableTracking = true,
   Expression<Func<T, bool>> predicateExpression = null,
   Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null)
{
   var query = this.dataContext.Set<T>().AsQueryable();

   if (disableTracking)
   {
      query.AsNoTracking();
   }

   if (include != null)
   {
      query = include(query);
   }

   if (predicateExpression != null)
   {
      query = query.Where(predicateExpression);
   }

   return query;
}
```

І одна з найбільш цікавих особливостей це повернення колекції IQueryable<T>. Цей об’єкт надає змогу виконати відкладений запит до бази даних з найкращою оптимізацією.

Наприклад, якщо ми виконаємо метод Find, вказавши якийсь фільтруючий параметр, цей метод поверне нам насправді лише сам запит і мі можемо

продовжити додавати параметри до нього на рівні логіки. Лише після виконання асинхронної операції над цією колекцією, такої як FirstAsync() або ToListAsync(), база даних виконає запит зразу включивши всі параметри та повернувши бажаний результат.

Натомість якщо ми використовували б колекцію IEnumerable<T> після виконання методу з DAL сервіс отримує безпосередньо список об’єктів. Тому якщо ми збираємось продовжити фільтрацію без потреби до всіх інших об’єктів у цій колеції це виглядає ірраціонально. Особливо при великій кількості записів в БД це можу суттєво вплинути на продуктивність роботи системи.

Для передачі підключення до БД та передачі залежностей між інтерфейсами та реалізаціями використовується Dependency injection.

Це шаблон проектування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію управління» для отримання залежностей.

Впровадження — це передача залежності (тобто, сервісу) залежному об'єкту (тобто, клієнту). Передавати залежності клієнту замість дозволити клієнту створити сервіс є фундаментальною вимогою до цього шаблону проектування.

В .Net Core Dependency injection вбудований в середу виконання. Налаштування залежностей відбувається в класі Program.cs:

```
builder.Services.AddDbContext<DataContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString(DEFAULT_CONNECTION)));

builder.Services.AddScoped<IGenericRepository<User>, GenericRepository<User>>();
builder.Services.AddTransient<IUserService, UserService>();
```

builder.Services.AddTransient<IRequestDetailsService, RequestDetailsService>();

Отже в об'єкті WebApplicationBuilder відбувається налаштування залежностей. Важливо звернути увагу на методі завдяки яким додаються залежності. Вони визначають їх час життя.

Для метода AddTransient при кожному зверненні до сервісу створюється новий об'єкт сервісу. Протягом одного запиту може бути кілька звернень до сервісу, відповідно при кожному зверненні буде створюватися новий об'єкт. Подібна модель життєвого циклу найбільш підходить для легких сервісів, які не зберігають даних про стан

Для метода AddScoped для кожного запиту створюється свій об'єкт сервісу. Тобто якщо протягом одного запиту є кілька звернень до одного сервісу, то при всіх цих зверненнях буде використовуватися один і той же об'єкт сервісу.

Для метода AddSingleton об'єкт сервісу створюється при першому зверненні до нього, всі наступні запити використовують один і той же раніше створений об'єкт сервісу.

Дуже важливо правильно оцінити необхідний час життя. Деякі сервіси мають не стабільну роботу якщо обрана не правильно модель залежностей. Наприклад як ми можемо бачити при створенні DataContext не вказаний час життя. Але він вказаний за замовченням – Transient. Це надзвичайно важливо бо кожна дія з базою даних є атомарною і повинна виконатися або не виконатися, і це не повинно впливати на роботу інших запитів.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення - це процес аналізу програмного засобу і супутньої документації з метою виявлення дефектів і підвищення якості продукту [7].

Тестування відрізняється по видам: функціональне та нефункціональне. Якщо розглядати функціональне тестування то його можна розглянути на усіх рівнях:

- компонентному або модульному (Component / Unit testing);
- інтеграційному (Integration testing);
- системному (System testing);
- приймальному (Acceptance testing).

Так як розробка даної програмної системи проводилась одним розробником то було обране найбільш зрозуміле і швидке тестування на модульному рівні.

Модульне (компонентне) тестування (unit testing, module testing, component testing) - направлено на перевірку окремих невеликих частин програми, які (як правило) можна досліджувати ізольовано від інших подібних частин. При виконанні даного тестування можуть перевірятися окремі функції або методи класів, самі класи, взаємодія класів, невеликі бібліотеки, окремі частини програми. часто даний вид тестування реалізується з використанням спеціальних технологій та інструментальних засобів автоматизації тестування, що значно спрощують і прискорюють розробку відповідних тест-кейсів [7].

Саме модульне тестування найчастіше проводиться самими розробниками під час циклу розробки програмного забезпечення. Основною парадигмою є те що тестується лише і тільки лише один модуль системи (наприклад один метод) без залежності з другими методами. Також під час модульного тестування насамперед не використовуються реальні данні з бази даних. Якщо такі данні все таки необхідні то вони підміняються так званими мок-апами.

Отже для тестування розробленого ПЗ був обраний фрймворк xUnit, що в свою чергу на даний момент э найбільш популярним фреймворком для модульного тестування в середі .Net. Для підміни реалізації рівня репозиторія, який використовується для взаємодії з базою даних, був обраний фреймворк Moq.

Приклад написаного unit-case можна переглянути на рисунку 5.1.

```
[Fact]
| Ссылок: 0
public async Task GetRequest_WhenCalled_ReturnsOkResult()
{
    // Arrange
    var httpRequestMock = new Mock<HttpRequest>();
    httpRequestMock.SetupGet(r => r.Headers)
        .Returns(new HeaderDictionary(new Dictionary<string, StringValues>()));
    httpRequestMock.SetupGet(r => r.ContentType)
        .Returns("application/json");
    httpRequestMock.SetupGet(r => r.Body)
        .Returns(new MemoryStream());

    requestDetailsServiceMock.Setup(service => service.GetRequestDetails(httpRequestMock.Object))
        .ReturnsAsync(new RequestDetails());

    var httpContextMock = new Mock<HttpContext>();
    httpContextMock.SetupGet(c => c.Request)
        .Returns(httpRequestMock.Object);

    var controllerContext = new ControllerContext()
    {
        HttpContext = httpContextMock.Object
    };
    controller.ControllerContext = controllerContext;

    // Act
    var result = await controller.GetRequest();

    // Assert
    Assert.IsType<OkObjectResult>(result);
}
```

Рисунок 5.1 – Приклад модульного тесту для методу GetRequest

Важливим моментом є використання партерну написання модульних тестів – AAA (Arrange-Act-Assert) . Ці слова що записують у виді коментарів створюють зрозумілу структуру тест-кейсу. Спочатку ми створюємо тестові данні, оголошуємо зміні та створюємо сам клас що підлягає тестуванню. У частині “Act” ми викликаємо метод та передаємо в нього тестові параметри (за необхідністю). І в заключній частині ми порівнюємо отриманий результат з очікуваним. Важливо що очікуваний результат не обов’язково має бути якимось значенням. Досить часто очікуваним результатом може стати викид правильної

помилки методом, що підлягає тестування. Натомість заборонено викидати помилки в методі самого юніт-кейсу.

Результаті проходження інших модульних тестів можливо переглянути на рисунку 5.2.

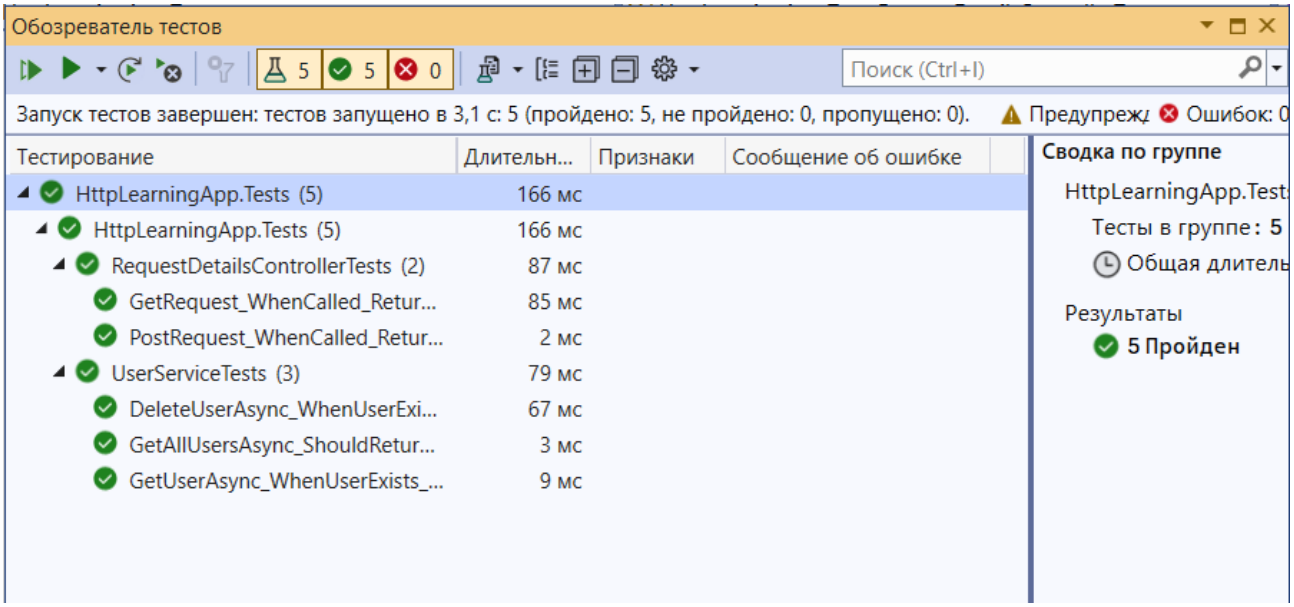

Рисунок 5.2 – Список проходження всіх модульних тестів

Отже ми можемо бачити що усі написані модульні тести були успішно пройдені, що в свою чергу вказує на правильність роботи програмного забезпечення.

Можна зробити висновок що модульне тестування дуже корисна річ. Що дозволяє розробнику дуже швидко перевірити написаний. Найбільш правильним з цієї точки зору підходом є TDD. Якщо слідувати за цією парадигмою створення програмного застосунку починається як раз з написання модульних тестів. А вже після цього реалізуються інтерфейси для яких були розроблені тести.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

У результаті виконання кваліфікаційної роботи було сформовано вимоги до навчального програмного забезпечення, яке надає можливість зрозуміти та навчитись взаємодії з HTTP протоколом. Проведено аналіз предметної галузі, описано функціональні вимоги та визначено існуючі аналоги. Висновки роботи підтвердили, що існуючі засоби для вивчення протоколу HTTP не забезпечують достатньої інтерактивності та орієнтованість на навчальну складову.

Отже, актуальність роботи полягала у розробці навчального програмного забезпечення, виклад матеріалу якого буде більш структурований та зрозумілий, дає чітке уявлення про внутрішні процеси під час взаємодії з HTTP протоколом у контексті навчальної програми для початківців.

Виходячи з цього, було розроблене навчальне програмне забезпечення, яке, на відміну від існуючих аналогів, надає користувачам наочну, більш поглиблену інформацію про формування запитів до серверу, що сприяє прискоренню та покращенню процесу навчання. Детально спроектована архітектура веб-застосунку включає опис бази даних, серверного та клієнтського рівнів.

Реалізоване програмне забезпечення має набір ключових функцій, таких як відображення запитів і відповідей, створення запитів різних типів, налагодження запитів, відправлення запитів, аналіз відповідей та підтримку автоматичного заповнення. Ці функції забезпечують користувачам можливість активно взаємодіяти з протоколом HTTP, емулюючи реальні ситуації взаємодії клієнта і сервера.

Зроблені висновки підкреслюють, що розроблене навчальне програмне забезпечення є цінним інструментом для засвоєння концепцій HTTP протоколу та набуття навичок у веб-розробці. Воно вирішує проблеми обмеженої кількості пояснювального матеріалу і недостатньої інтерактивності, що характерні для існуючих рішень. Розробка програмного забезпечення була здійснена з

урахуванням поточних вимог та тенденцій у сфері HTTP протоколу та веб-розробки.

У майбутньому рекомендується активно поширювати навчальне програмне забезпечення через презентації, семінари та вебінари для підвищення зацікавленості студентів та викладачів у використанні HTTP протоколу. Через це розроблене програмне забезпечення може стати чудовим помічником для тих, хто хоче краще зрозуміти, як саме працює HTTP протокол та набути навичок розробки веб-додатків, які працюють з цим протоколом. Також важливим кроком у подальшій роботі є проведення оцінки ефективності програмного забезпечення та врахування отриманих відгуків для подальшого вдосконалення.

ПЕРЕЛІК ПОСИЛАНЬ

1. A Guide to JavaScript HTTP Requests – URL: https://kinsta.com/knowledgebase/javascript-http-request/ (дата звернення: 21.04.2023)

2. Common web application architectures – URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures#clean-architecture (дата звернення: 05.04.2023).

3. Microsoft Developer Network. Библиотека MSDN. Разработка на .NET / MSDN – сеть разработчиков Microsoft. - URL: https://msdn.microsoft.com/ru-ru/ (дата звернення: 10.05.2023).

4. Tutorial: Create a web API with ASP.NET Core – URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio (дата звернення: 01.04.2023).

5. Дейт К.Дж. Введение в системы баз данных / К.Дж. Дейт. - 10-е изд. – М.:Издательский дом „Вильямс”, 2012.- 1328 с..

6. Еванс Е. Предметно орієнтоване проектування (DDD) Е. Еванс – М.: «Вильямс», 2011. — С. 448.

7. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – Минск: Четыре четверти, 2017. – 312 с.

8. Поллард Б. П26 HTTP/2 в действии / пер. с анг. П. М. Бомбаковой. – М.: ДМК Пресс, 2021. –424 с.: ил.

9. Руководство по ASP.NET Core 3.1 – URL: https://metanit.com/sharp/aspnet5 (дата звернення: 10.05.2023).

10. Фаулер М. Шаблоны корпоративных приложений / М. Фаулер – М.: «Вильямс», 2012. — 544 с.

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор

Українського державного університету науки і технології

Анатолій РАДКЕВИЧ

РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01286-01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Вадим АНДРЮЩЕНКО

Виконавець

_____Іван БОЛТЕНКОВ

Нормоконтролер

_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01286-01-ЛЗ

РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP

Технічне завдання
1116130.01286-01
Листів 14

2023

ЗМІСТ

1 **ВВЕДЕННЯ**

У сучасному світі, де веб-технології використовуються широкою аудиторією, розуміння і володіння HTTP протоколом стає ключовим для розробників веб-додатків та інших IT-спеціалістів. HTTP (Hypertext Transfer Protocol) є основою комунікації в мережі Інтернет і використовується для передачі даних між веб-клієнтами та серверами. Він є важливим елементом розуміння того, як працюють веб-додатки та як взаємодіяти з ними.

З метою полегшити процес вивчення та розуміння HTTP протоколу, виникає потреба у створенні навчального програмного забезпечення, яке надаватиме користувачам можливість взаємодіяти з протоколом у практичному середовищі. Це програмне забезпечення має на меті надати студентам, розробникам та всім зацікавленим особам зручні та ефективні інструменти для вивчення та експериментування з HTTP протоколом.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.2022 №1209 ст ректора Українського державного університету науки і технологій “Про призначення керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи - “РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP”. Керівник - доц. Андрющенко В.О.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – навчальне програмне забезпечення має забезпечувати можливість створювати HTTP запити до веб-серверів та отримувати відповіді.

Експлуатаційне призначення – розроблене навчальне програмне забезпечення буде використовуватись в освітніх закладах, де викладаються курси з програмування, веб-розробки та інформаційних технологій. Воно стане цінним інструментом для викладачів та студентів, дозволяючи їм практично освоїти основи протоколу HTTP та набути навичок розробки веб-додатків, які працюють з цим протоколом.

4 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Розроблене навчальне програмне забезпечення для демонстрації використання протоколу HTTP повинно відповідати наступним функціональним вимогам:

- Відображення запитів і відповідей: Програма повинна відображати як запити, так і відповіді, які обмінюються між клієнтом та сервером. Це включає відображення HTTP методу, заголовків, тіла запиту та відповіді.
- Створення запитів: Користувач повинен мати можливість створювати різні типи запитів, включаючи GET, POST, PUT, DELETE. Програма повинна надавати інтерфейс для введення URL, параметрів запиту, тіла запиту та інших відповідних даних.
- Налагодження запитів: Програма повинна забезпечувати можливість налагоджувати запити шляхом додавання, видалення або зміни заголовків, параметрів та тіла запиту. Користувач повинен мати контроль над усіма аспектами запиту.
- Відправлення запитів: Після налаштування запиту, програма повинна надати можливість відправляти запит до визначеного сервера та отримувати відповідь.
- Аналіз відповідей: Програма повинна аналізувати та відображати вміст отриманих відповідей, включаючи статус-код, заголовки та тіло відповіді.
- Підтримка автоматичного заповнення: Програма може надавати функціонал автоматичного заповнення певних полів запиту або вибору заголовків зі списку, щоб полегшити створення запитів.

Вхідні дані:

- URL: Користувач вводить URL-адресу сервера, з яким вони бажають взаємодіяти. Це може бути URL-адреса реального веб-сайту або локального сервера для тестування.
- HTTP метод: Користувач вибирає HTTP метод, такий як GET, POST, PUT, DELETE, для виконання запиту.
- Параметри запиту: Користувач може вказати параметри запиту, такі як рядки запиту, заголовки, тіло запиту або інші відповідні дані, що відправляються разом з запитом.

Вихідні дані:

- Відповідь сервера: Після виконання запиту до сервера, програма може отримати відповідь сервера, яка містить різні дані, такі як статус-код, заголовки, тіло відповіді та інші відповідні деталі.
- Відповідь на запит: Програма може вивести відповідь на запит користувача, що включає дані, які були отримані від сервера, такі як HTML-код сторінки, JSON-дані або будь-які інші типи вмісту, які можуть бути пов'язані з виконаним запитом.

4.1 **Вимоги до надійності**

Вимоги до надійності наступні:

– при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;

– наявність архівної копії тексту програми на зовнішньому носії;

– наявність резервної копії бази даних на зовнішньому носії.

4.2 **Умови експлуатації**

Програмний продукт повинен використовуватись у приміщеннях які відповідають умовам роботи ЕОМ. Для стійкого експлуатування програми потрібно витримувати температурний режим, де знаходяться робочі ПК 21-25°С, відносна вологість 40-60% (ДНАОП 0.00-1.31-99).

З програмою може працювати будь-яка людина, що має досвід роботи з ЕОМ або мобільним телефоном та знайома з керівництвом користувача.

4.3 Вимоги до складу та параметрів технічних засобів

Продукт, що розробляється повинен використовуватись на мобільних пристроях, що мають наступні характеристики:

Вимоги до складу і параметрів технічних засобів наступні:

- 16 гігабайт оперативної пам’яті або більше;
- не менш ніж 25 гігабайт вільного місця на диску;
- чотирьох ядерний процесор, або більше;
- клавіатура;
- миша

4.4 Вимоги до інформаційної та програмної сумісності

Для того, щоб розгорнути серверний застосунок, необхідно мати наступне програмне забезпечення:

- ОС: Windows 10 або вище;
- .Net Core версії 7.0;
- Node.js версії 18.16.0 або вище;
- Npm версії 9.7.1 або вище.

4.5 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На упаковці повинно бути вказано: назва програми, розробник, реквізити розробника.

Приклад маркування приведений на рисунку 1.

РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP

Розробник: Болтенков І.В. , студент ПЗ1911

УДУНТ, кафедра КІТ

2023

Рис. 1. Приклад маркування

4.6 **Вимоги до транспортування та зберігання**

Транспортування можна виконувати будь-яким способом, що виключає механічні та електромагнітні впливи на носії інформації. Транспортування програмного продукту передбачається через microUSB порт.

Місце зберігання повинно бути сухим, з відсутністю доступу прямих сонячних променів, без пилу.

Строк зберігання обумовлений строком зберігання інформації на носії. Рекомендується проводити профілактичні роботи перевірки якості носіїв кожні 6 місяців.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

– специфікація;

– текст програми;

– опис програми;

– керівництво користувача.

Вся документація програмного додатку повинна задовольняти вимоги до програмної документації.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програмного продукту приведені в табл. 1.

Таблиця 1. – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	01.03.23 -18.03.23
Робочий проект	Програмування та відлагодження програми.	19.03.23 - 20.05.23
	Тестування програми	20.05.23 - 27.05.23
	Розробка, узгодження і затвердження програмної документації.	27.05.23 - 18.06.23

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Андрющенко В.О.

Прийом здійснюється комісією у складі:

- Горячкін В. М. (керівник підрозділу);
- Андрющенко В.О. (керівник розробки).

8 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор

Українського державного університету науки і технології

Анатолій РАДКЕВИЧ

РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01286-01 12 01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Вадим АНДРЮЩЕНКО

Виконавець

_____Іван БОЛТЕНКОВ

Нормоконтролер

_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01286-01 12 01-ЛЗ

РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP

Текст програми
1116130.01286-01 12 01
Листів 35

2023

АНОТАЦІЯ

Документ 44165850.94101-01 12 01 «РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕМОНСТРАЦІЇ ВИКОРИСТАННЯ ПРОТОКОЛУ HTTP» входить до складу програмної документації на додаток, що реалізовують взаємодію бази даних з інтерфейсом.

У даному документі представлений текст програм. Програми написані на мові C# та JavaScript. Об'єм пам'яті, що займають програми комплексу, складає 70 Мб. Конфігурація комп'ютера стандартна. Комплекс функціонує в середовищі MS WINDOWS 10.

ЗМІСТ

1 ТЕКСТ ПРОГРАМИ

1.1 UserService.cs

```
using AutoMapper;
using HttpLearningApp.Domain.DTOs;
using HttpLearningApp.Domain.Entities;
using HttpLearningApp.Domain.Interfaces.Repositories;
using HttpLearningApp.Domain.Interfaces.Services;
using Microsoft.EntityFrameworkCore;

namespace HttpLearningApp.BLL.Implementation
{
    public class UserService : IUserService
    {
        private readonly IGenericRepository<User> userRepository;
        private readonly IMapper mapper;

        public UserService(IGenericRepository<User> userRepository, IMapper mapper)
        {
            this.userRepository = userRepository;
            this.mapper = mapper;
        }

        public async Task<int> AddUserAsync(AddUserDTO userDTO)
        {
            if (await this.userRepository.Find().AnyAsync(user => user.Email ==
userDTO.Email))
            {
                throw new InvalidOperationException("User with this email already
exist");
            }

            var user = this.mapper.Map<User>(userDTO);

            return await this.userRepository.CreateAsync(user);
        }

        public async Task<int> AddUserAsync(User user)
        {
            if (await this.userRepository.Find().AnyAsync(us => us.Email ==
user.Email))
            {
                throw new InvalidOperationException("User with this email already
exist");
            }

            return await this.userRepository.CreateAsync(user);
        }

        public async Task DeleteUserAsync(int id)
        {
            await this.userRepository.DeleteAsync(id);
        }

        public async Task<IEnumerable<User>> GetAllUsersAsync()
        {
            return await this.userRepository.FindAsync();
        }

        public async Task<bool> UserExist(int id)
        {
```

```
            return await this.userRepository.Find().AnyAsync(user => user.Id == id);
        }

        public async Task<User?> GetUserAsync(int id)
        {
            return await this.userRepository.GetByIdAsync(id);
        }

        public async Task UpdateUserAsync(User user)
        {
            await this.userRepository.UpdateAsync(user);
        }
    }
}
```

1.2 IGenericRepository.cs

```
using HttpLearningApp.Domain.Entities;
using Microsoft.EntityFrameworkCore.Query;
using System.Linq.Expressions;

namespace HttpLearningApp.Domain.Interfaces.Repositories
{
    public interface IGenericRepository<T> where T : BaseEntity
    {
        Task<int> CreateAsync(T entity);

        Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> predicateExpression
= null,
            bool disableTracking = true);

        Task<IEnumerable<T>> FindAsync();

        IQueryable<T> Find(
            bool disableTracking = true,
            Expression<Func<T, bool>> predicateExpression = null,
            Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null);

        Task DeleteAsync(int id);

        Task UpdateAsync(T entity);

        Task UpdateRangeAsync(IEnumerable<T> entities);

        Task<T?> GetByIdAsync(int id, Func<IQueryable<T>, IIncludableQueryable<T,
object>> include = null, bool disableTracking = true);

        Task DeleteRangeAsync(IEnumerable<T> entities);
    }
}
```

1.3 IUserRepository.cs

```
using HttpLearningApp.Domain.DTOs;
using HttpLearningApp.Domain.Entities;

namespace HttpLearningApp.Domain.Interfaces.Services
{
    public interface IUserService
    {
        Task<int> AddUserAsync(AddUserDTO userDTO);
```

```
        Task<int> AddUserAsync(User user);

        Task UpdateUserAsync(User user);

        Task DeleteUserAsync(int id);

        Task<User?> GetUserAsync(int id);

        Task<IEnumerable<User>> GetAllUsersAsync();

        Task<bool> UserExist(int id);
    }
}
```

1.4 DataContext.cs

```
using HttpLearningApp.Domain.Entities;
using Microsoft.EntityFrameworkCore;

namespace HttpLearningApp.DAL.Data
{
    public class DataContext : DbContext
    {
        public DbSet<User> Users { get; set; }

        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<User>().Property(x =>
x.Name).IsRequired().HasMaxLength(1000);
            modelBuilder.Entity<User>().HasIndex(x => x.Email).IsUnique();
        }
    }
}
```

1.5 GenericRepository.cs

```
using HttpLearningApp.DAL.Data;
using HttpLearningApp.Domain.Entities;
using HttpLearningApp.Domain.Interfaces.Repositories;
using Microsoft.EntityFrameworkCore.Query;
using System.Linq.Expressions;
using Microsoft.EntityFrameworkCore;

namespace HttpLearningApp.DAL.RepositoryImplementation
{
    public class GenericRepository<T> : IGenericRepository<T>
        where T : BaseEntity
    {
        private readonly DataContext dataContext;

        public GenericRepository(DataContext dataContext)
        {
            this.dataContext = dataContext;
        }
```

```
        public async Task<int> CreateAsync(T entity)
        {
            var entityEntry = await this.dataContext.Set<T>().AddAsync(entity);
            await this.dataContext.SaveChangesAsync();

            return entityEntry.Entity.Id;
        }

        public async Task DeleteAsync(int id)
        {
            var entity = await this.GetByIdAsync(id);

            this.dataContext.Set<T>().Remove(entity);
            await this.dataContext.SaveChangesAsync();
        }

        public async Task DeleteRangeAsync(IEnumerable<T> entities)
        {
            this.dataContext.Set<T>().RemoveRange(entities);

            await this.dataContext.SaveChangesAsync();
        }

        public IQueryable<T> Find(bool disableTracking = true,
            Expression<Func<T, bool>> predicateExpression = null,
            Func<IQueryable<T>, IIncludableQueryable<T, object>> include = null)
        {
            var query = this.dataContext.Set<T>().AsQueryable();

            if (disableTracking)
            {
                query.AsNoTracking();
            }

            if (include != null)
            {
                query = include(query);
            }

            if (predicateExpression != null)
            {
                query = query.Where(predicateExpression);
            }

            return query;
        }

        public async Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>>
predicateExpression = null,
            bool disableTracking = true)
        {
            var query = this.dataContext.Set<T>().AsQueryable();

            if (disableTracking)
            {
                query.AsNoTracking();
            }

            if (predicateExpression != null)
            {
                query = query.Where(predicateExpression);
            }
```

```
            var users = await query.ToListAsync();
            return users;
        }

        public async Task<IEnumerable<T>> FindAsync() => await
this.dataContext.Set<T>().ToListAsync();

        public async Task<T?> GetByIdAsync(int id, Func<IQueryable<T>,
IIncludableQueryable<T, object>> include = null, bool disableTracking = true)
        {
            IQueryable<T?> query = this.dataContext.Set<T>().AsQueryable();

            if (disableTracking)
            {
                query.AsNoTracking();
            }

            if (include != null)
            {
                query = include(query);
            }

            return await query.FirstOrDefaultAsync(entity => entity.Id == id);
        }

        public async Task UpdateAsync(T entity)
        {
            this.dataContext.Set<T>().Update(entity);
            await this.dataContext.SaveChangesAsync();
        }

        public async Task UpdateRangeAsync(IEnumerable<T> entities)
        {
            this.dataContext.Set<T>().UpdateRange(entities);
            await this.dataContext.SaveChangesAsync();
        }
    }
}
```

1.6 IRequestDetailsService.cs

```
using Microsoft.AspNetCore.Http;

namespace HttpLearningApp.Utils.RequestDetailsHelper
{
    public interface IRequestDetailsService
    {
        Task<RequestDetails> GetRequestDetails(HttpRequest request);
    }
}
```

1.7 RequestDetails.cs

```
namespace HttpLearningApp.Utils.RequestDetailsHelper
{
    public class RequestDetails
    {
        public string Method { get; set; }
```

```
        public string Scheme { get; set; }

        public string Host { get; set; }

        public string PathBase { get; set; }

        public string Path { get; set; }

        public string QueryString { get; set; }

        public Dictionary<string, string> Headers { get; set; }

        public string ContentType { get; set; }

        public object Body { get; set; }

    }
}
```

1.8 RequestDetailsService.cs

```
using System.Text;
using System.Xml;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.WebUtilities;
using Newtonsoft.Json;

namespace HttpLearningApp.Utils.RequestDetailsHelper
{
    public class RequestDetailsService : IRequestDetailsService
    {
        public async Task<RequestDetails> GetRequestDetails(HttpRequest request)
        {
            var headers = request.Headers.ToDictionary(
                header => header.Key,
                header => header.Value.ToString());

            // Get the Content-Type of the request
            var contentType = request.ContentType;

            // Convert the request body to a string if it's a type we can handle
            var body = string.Empty;

            if (request.Body == null)
            {
                body = "The request body is empty";

                return new RequestDetails
                {
                    Method = request.Method,
                    Scheme = request.Scheme,
                    Host = request.Host.Value,
                    PathBase = request.PathBase.Value,
                    Path = request.Path.Value,
                    QueryString = request.QueryString.Value,
                    Headers = headers,
                    ContentType = contentType,
                    Body = body
                };
            }
```

```
            if (request.Body != null && contentType is "application/json" or
"text/plain")
            {
                using (var reader = new StreamReader(request.Body, Encoding.UTF8,
true, 1024, true))
                {
                    body = await reader.ReadToEndAsync();
                }

                // Rewind the request body stream position so it can be read again
if needed
                request.Body.Position = 0;
            }

            object deserializedBody = null;
            try
            {
                if (contentType == "application/json")
                {
                    deserializedBody = JsonConvert.DeserializeObject(body);
                    deserializedBody =
JsonConvert.SerializeObject(deserializedBody);
                }
                else if (contentType == "text/plain")
                {
                    deserializedBody = body; // No deserialization needed for plain
text
                }
                else if (contentType == "application/xml")
                {
                    var xmlDocument = new XmlDocument();
                    xmlDocument.LoadXml(body);
                    deserializedBody = xmlDocument;
                }
                else if (contentType == "application/x-www-form-urlencoded")
                {
                    var formCollection = await new
FormReader(request.Body).ReadFormAsync();
                    deserializedBody = formCollection.Keys.ToDictionary(k => k, k =>
formCollection[k].ToString());
                }
            }
            catch
            {
                body = "Failed to read the request body.";
            }

            return new RequestDetails
            {
                Method = request.Method,
                Scheme = request.Scheme,
                Host = request.Host.Value,
                PathBase = request.PathBase.Value,
                Path = request.Path.Value,
                QueryString = request.QueryString.Value,
                Headers = headers,
                ContentType = contentType,
                Body = deserializedBody ?? body
            };
        }
    }
}
```

1.9 ApiError.cs

```
namespace HttpLearningApp.Utils.Wrappers.Response
{
    public class ApiError
    {
        public string Message { get; set; }

        public string Key { get; set; }

        public string Detail { get; set; }

        public ApiError(string key, string message)
        {
            this.Message = message;
            this.Key = key;
        }

        public ApiError(string message)
        {
            this.Message = message;
        }
    }
}
```

1.10 ResponseWrapper.cs

```
using HttpLearningApp.Utils.RequestDetailsHelper;

namespace HttpLearningApp.Utils.Wrappers.Response
{
    public class ResponseWrapper<T>
    {
        public ResponseWrapper(T data, RequestDetails requestDetails, bool succeeded
= true)
        {
            this.RequestDetails = requestDetails;
            this.Data = data;
            this.Succeeded = succeeded;
        }

        public ResponseWrapper(RequestDetails requestDetails, bool succeeded = true)
        {
            this.RequestDetails = requestDetails;
            this.Succeeded = succeeded;
        }

        public T? Data { get; set; }

        public RequestDetails RequestDetails { get; set; }

        public bool Succeeded { get; set; }

    }
}
```

1.11 CrudController.cs

```
using HttpLearningApp.Domain.DTOs;
using HttpLearningApp.Domain.Entities;
using HttpLearningApp.Domain.Interfaces.Services;
using HttpLearningApp.Utils.RequestDetailsHelper;
using HttpLearningApp.Utils.Wrappers.Response;
using Microsoft.AspNetCore.Mvc;

namespace HttpLearningApp.API.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class CrudController : Controller
    {
        private readonly IUserService userService;
        private readonly IRequestDetailsService requestDetailsService;

        public CrudController(IUserService userService, IRequestDetailsService
requestDetailsService)
        {
            this.userService = userService;
            this.requestDetailsService = requestDetailsService;
        }

        [HttpGet]
        public async Task<IActionResult> GetUser([FromQuery] int id)
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);

            var user = await this.userService.GetUserAsync(id);

            if (user == null)
            {
                return NotFound(new ResponseWrapper<object>(requestDetails, false));
            }

            return Ok(new ResponseWrapper<User>(user, requestDetails));
        }


        [HttpGet]
        public async Task<IActionResult> GetAllUsers()
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);

            var users = await this.userService.GetAllUsersAsync();
```

```
            return Ok(new ResponseWrapper<IEnumerable<User>>(users,
requestDetails));
        }

        [HttpPost]
        public async Task<IActionResult> PostUser([FromBody] AddUserDTO addUserDTO)
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);
            requestDetails.Body = addUserDTO;

            try
            {
                var createdUserId = await this.userService.AddUserAsync(addUserDTO);
                var uri = $"/Crud/GetUser?id={createdUserId}";

                return Created(uri, new ResponseWrapper<int>(createdUserId,
requestDetails));
            }
            catch (InvalidOperationException ex)
            {
                return BadRequest(new ResponseWrapper<string>(ex.Message,
requestDetails, false));
            }
        }

        [HttpPut]
        public async Task<IActionResult> PutUser([FromBody] User user)
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);
            requestDetails.Body = user;
            try
            {
                if (!await this.userService.UserExist(user.Id))
                {
                    var createdUserId = await this.userService.AddUserAsync(user);
                    var uri = $"/Crud/GetUser?id={createdUserId}";
                    return Created(uri, new ResponseWrapper<int>(createdUserId,
requestDetails));
                }

                await this.userService.UpdateUserAsync(user);
                return NoContent();
            }
            catch (Exception ex)
            {
                return BadRequest(new ResponseWrapper<string>(ex.Message,
requestDetails, false));
            }
        }

        [HttpDelete]
        public async Task<IActionResult> DeleteUser([FromQuery] int id)
        {
            var request = HttpContext.Request;
```

```
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);

            if (!await this.userService.UserExist(id))
            {
                return NotFound(new ResponseWrapper<object>(requestDetails, false));
            }

            await this.userService.DeleteUserAsync(id);

            return NoContent();
        }
    }
}
```

1.12 RequestDetailsController.cs

```
using HttpLearningApp.Utils.RequestDetailsHelper;
using HttpLearningApp.Utils.Wrappers.Response;
using Microsoft.AspNetCore.Mvc;


namespace HttpLearningApp.API.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class RequestDetailsController : ControllerBase
    {
        private readonly IRequestDetailsService requestDetailsService;

        public RequestDetailsController(IRequestDetailsService
requestDetailsService)
        {
            this.requestDetailsService = requestDetailsService;
        }

        [HttpGet]
        public async Task<IActionResult> GetRequest()
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);

            return Ok(new ResponseWrapper<object>(requestDetails));
        }

        [HttpPost]
        public async Task<IActionResult> PostRequest()
        {
            var request = HttpContext.Request;
            request.EnableBuffering(); // Enables request body buffering

            var requestDetails = await
this.requestDetailsService.GetRequestDetails(request);

            return Ok(new ResponseWrapper<object>(requestDetails));
        }
```

```
    }
}
```

1.13 Program.cs

```
using HttpLearningApp.API.Mappers;
using HttpLearningApp.BLL.Implementation;
using HttpLearningApp.DAL.Data;
using HttpLearningApp.DAL.RepositoryImplementation;
using HttpLearningApp.Domain.Entities;
using HttpLearningApp.Domain.Interfaces.Repositories;
using HttpLearningApp.Domain.Interfaces.Services;
using HttpLearningApp.Utils.RequestDetailsHelper;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

const string DEFAULT_CONNECTION = "DefaultConnection";

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<DataContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString(DEFAULT_CONNECTION)))
;

builder.Services.AddScoped<IGenericRepository<User>, GenericRepository<User>>();
builder.Services.AddTransient<IUserService, UserService>();
builder.Services.AddTransient<IRequestDetailsService, RequestDetailsService>();

builder.Services.AddAutoMapper(typeof(MapperProfile));

var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseDefaultFiles();
app.UseStaticFiles();

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();
```

```
app.Run();
```

1.14 Script.js

```
const form = document.getElementById('requestForm');
const urlInput = document.getElementById('urlInput');
const methodSelect = document.getElementById('methodSelect');
const headersTable = document.getElementById('headersTable');

const bodyInputType = document.getElementById('bodyInputType');
const bodyTable = document.getElementById('bodyTable');
const bodyText = document.getElementById('bodyText');
const requestContainer = document.getElementById('requestContainer');
const responseContainer = document.getElementById('responseContainer');
const responsePreContainer = document.getElementById('responsePreContainer');
const requestViewContainer = document.getElementById('requestViewContainer');
const requestViewPreContainer = document.getElementById('requestViewPreContainer');
const requestViewH = document.getElementById('requestViewH');
const resultH = document.getElementById('resultH');
const resultContainer = document.getElementById('resultContainer');
const toggleButton = document.getElementById('toggle');
const urlSelect = document.getElementById('urlSelect');
const endpointInput = document.getElementById('endpointInput');
// При нажатии на кнопку окно выдвигается или скрывается, кнопка перемещается
var helpButton = document.querySelector('.help-button');
var helpWindow = document.querySelector('.help-window');
var leftSection = document.querySelector('.left-section');
var isOpen = false; // Флаг для отслеживания состояния окна

const hostUrl = window.location.protocol + "//" + window.location.host;

const httpHeadersDictionary = {
    "accept": "Media type(s) that is/are acceptable for the response.",
    "accept-charset": "Character sets that are acceptable.",
    "accept-encoding": "List of acceptable encodings.",
    "accept-language": "List of acceptable human languages for response.",
    "accept-datetime": "Acceptable version in time.",
    "access-control-request-method": "Used when issuing a preflight request to let
the server know what HTTP method will be used.",
    "access-control-request-headers": "Used when issuing a preflight request to let
the server know what HTTP headers will be used.",
    "authorization": "Authentication credentials for HTTP authentication.",
    "cache-control": "Directives for caching mechanisms in both requests and
responses.",
    "connection": "Control options for the current connection and list of hop-by-hop
response fields.",
    "content-length": "The size of the entity-body, in decimal number of octets,
sent to the recipient.",
    "content-md5": "An MD5 sum of the entity-body for the purpose of providing an
end-to-end message integrity check.",
    "content-type": "The Media type of the body of the request (used with POST and
PUT requests).",
    "cookie": "An HTTP cookie previously sent by the server with Set-Cookie
(below).",
    "date": "The date and time that the message was originated.",
    "expect": "Indicates that particular server behaviors are required by the
client.",
    "forwarded": "Disclose original information of a client connecting to a web
server through an HTTP proxy.",
    "from": "The email address of the user making the request.",
```

```
    "host": "The domain name of the server (for virtual hosting), and the TCP port
number on which the server is listening.",
    "if-match": "Only perform the action if the client supplied entity matches the
same entity on the server.",
    "if-modified-since": "Allows a 304 Not Modified to be returned if content is
unchanged.",
    "if-none-match": "Allows a 304 Not Modified to be returned if content is
unchanged.",
    "if-range": "If the entity is unchanged, send me the part(s) that I am missing;
otherwise, send me the entire new entity.",
    "if-unmodified-since": "Only send the response if the entity has not been
modified since a specific time.",
    "max-forwards": "Limit the number of times the message can be forwarded through
proxies or gateways.",
    "origin": "Initiates a request for cross-origin resource sharing with Origin.",
    "pragma": "Implementation-specific fields that may have various effects anywhere
along the request-response chain.",
    "proxy-authorization": "Authorization credentials for connecting to a proxy.",
    "range": "Request only part of an entity.",
    "referer": "This is the address of the previous web page from which a link to
the currently requested page was followed.",
    "te": "The transfer encodings the user agent is willing to accept.",
    "user-agent": "The user agent string of the user agent.",
    "upgrade": "Ask the server to upgrade to another protocol.",
    "via": "Informs the server of proxies through which the request was sent.",
    "warning": "A general warning about possible problems with the entity body.",
    "location": "The Location header is an HTTP header used in server responses to
instruct the client to redirect to a different URL. It specifies the new URL where
the requested resource can be found."
    // Add more headers as needed
};


var httpStatusCodes = {
    100: "Continue - The server has received the initial part of the request and is
waiting for the client to send the remaining parts.",
    101: "Switching Protocols - The server is changing protocols according to the
client's request.",
    200: "OK - The request has succeeded. The response depends on the request
method.",
    201: "Created - The request has been fulfilled, and a new resource is created as
a result.",
    202: "Accepted - The request has been accepted for processing, but the
processing has not been completed yet.",
    203: "Non-Authoritative Information - The server successfully processed the
request, but is returning information from a different source.",
    204: "No Content - The server successfully processed the request, but there is
no content to send back.",
    205: "Reset Content - The server successfully processed the request, and the
user agent should reset the document view.",
    206: "Partial Content - The server is delivering only part of the resource due
to a range header sent by the client.",
    300: "Multiple Choices - The requested resource has multiple choices, each with
different locations.",
    301: "Moved Permanently - The requested resource has been permanently moved to a
new location.",
    302: "Found - The requested resource has been temporarily moved to a different
location.",
    303: "See Other - The response to the request can be found under a different
URI.",
    304: "Not Modified - The client can use the cached version of the requested
resource.",
```

```
    307: "Temporary Redirect - The requested resource has been temporarily moved to
a different location.",
    308: "Permanent Redirect - The requested resource has been permanently moved to
a different location.",
    400: "Bad Request - The server could not understand the request due to malformed
syntax or invalid data.",
    401: "Unauthorized - The client must authenticate itself to get the requested
response.",
    402: "Payment Required - Reserved for future use. The original intention was
that this code might be used as part of some form of digital cash or micropayment
scheme.",
    403: "Forbidden - The client does not have permission to access the requested
resource.",
    404: "Not Found - The server could not find the requested resource.",
    405: "Method Not Allowed - The method specified in the request is not allowed
for the resource.",
    406: "Not Acceptable - The server cannot generate a response that matches the
list of acceptable values defined in the request's headers.",
    407: "Proxy Authentication Required - The client must authenticate itself with
the proxy.",
    408: "Request Timeout - The server timed out waiting for the request.",
    409: "Conflict - The request could not be completed due to a conflict with the
current state of the resource.",
    410: "Gone - The requested resource is no longer available and has been
permanently removed.",
    411: "Length Required - The server requires a valid 'Content-Length' header to
be specified in the request.",
    412: "Precondition Failed - The precondition given in the request evaluated to
false by the server.",
    413: "Payload Too Large - The request entity is larger than the server is
willing or able to process.",
    414: "URI Too Long - The request URI exceeds the maximum length allowed by the
server.",
    415: "Unsupported Media Type - The server does not support the media type used
in the request.",
    416: "Range Not Satisfiable - The requested range cannot be fulfilled by the
server.",
    417: "Expectation Failed - The server cannot meet the requirements specified in
the Expect request header.",
    422: "Unprocessable Entity - The request was well-formed but unable to be
followed due to semantic errors.",
    429: "Too Many Requests - The user has sent too many requests in a given amount
of time.",
    500: "Internal Server Error - The server encountered an unexpected condition
that prevented it from fulfilling the request.",
    501: "Not Implemented - The server does not support the functionality required
to fulfill the request.",
    502: "Bad Gateway - The server, while acting as a gateway or proxy, received an
invalid response from an upstream server.",
    503: "Service Unavailable - The server is currently unable to handle the request
due to a temporary overload or maintenance of the server.",
    504: "Gateway Timeout - The server, while acting as a gateway or proxy, did not
receive a timely response from an upstream server.",
    505: "HTTP Version Not Supported - The server does not support the HTTP protocol
version used in the request.",
};

console.log(hostUrl);

helpButton.addEventListener('click', function () {
    isOpen = !isOpen; // Инвертируем состояние окна

    if (isOpen) {
```

```
        helpWindow.style.transform = 'translateX(0)';
        helpButton.style.right = 'calc(33.33% - 70px)'; /* При открытии окна кнопка
перемещается вместе с ним */
        leftSection.style.width = 'calc(67% - 40px)'; /* При открытии окна левая
часть сдвигается и занимает 2/3 экрана */
    } else {
        helpWindow.style.transform = 'translateX(100%)';
        helpButton.style.right = '-35px'; /* При закрытии окна кнопка возвращается к
правому краю без отступа */
        leftSection.style.width = '95%'; /* При закрытии окна левая часть
возвращается к 100% ширины экрана */
    }
});

// Функция для обновления информации о методе
function updateMethodInfo(method) {
    let theory = '';

    switch (method) {
        case 'GET':
            theory = 'Метод GET використовується для отримання ресурсів з сервера.
Зазвичай, при використанні методу GET, дані запиту передаються у рядку запиту (query
string) в URL-адресі. Метод GET не передає дані через тіло запиту (request body) і
використовується для запиту ресурсів без внесення змін на сервері. Такий запит може
бути кешованим браузером.';
            break;
        case 'POST':
            theory = 'Метод POST використовується для надсилання даних на сервер для
обробки. У відмінність від методу GET, дані запиту в методі POST передаються через
тіло запиту (request body) і не відображаються у URL-адресі. Цей метод зазвичай
використовується для створення нових ресурсів на сервері або виконання дій, що
змінюють стан існуючих ресурсів.';
            break;
        case 'PUT':
            theory = 'Метод PUT використовується для оновлення існуючого ресурсу на
сервері. Він також передає дані через тіло запиту, подібно до методу POST. Основна
відмінність між методами PUT і POST полягає в тому, що метод PUT є ідемпотентним. Це
означає, що при кількох послідовних ідентичних запитах PUT стан ресурсу на сервері
не змінюється після першого виконання запиту.';
            break;
        case 'DELETE':
            theory = 'Метод DELETE використовується для видалення ресурсу на
сервері. Він вказує серверу, що потрібно видалити вказаний ресурс. Запит DELETE
також може містити дані в тілі запиту, які допомагають серверу здійснити видалення.
Як і метод PUT, метод DELETE також є ідемпотентним. Це означає, що при кількох
послідовних ідентичних запитах DELETE стан сервера не змінюється після першого
виконання запиту.';
            break;
        default:
            theory = 'Інформація про обраний метод';
            break;
    }
    helpWindow.textContent = theory;
    helpWindow.style.backgroundColor = "WhiteSmoke";
}

// Устанавливаем информацию о методе GET по умолчанию
updateMethodInfo('GET');

// Обработчик события для обновления информации при изменении выбранного метода
methodSelect.addEventListener('change', function () {
    const selectedMethod = this.value;
    updateMethodInfo(selectedMethod);
```

```
    // Clear the headers table and add a default row on page load
    clearHeadersTable();
    updateHeaderAutocomplete();
});

// Обработчик события изменения типа ввода для URL
toggleButton.addEventListener('change', function () {
    if (toggleButton.checked) {
        endpointInput.style.display = 'block';
        urlInput.style.display = 'none';
        resultH.style.display = 'none';
        resultContainer.style.display = 'none';
    } else {
        endpointInput.style.display = 'none';
        urlInput.style.display = 'block';
        resultH.style.display = 'block';
        resultContainer.style.display = 'block';
    }
});

// Valid headers list
const validHeaders = {
    GET: ['Content-Type', 'Accept', 'Accept-Language'],
    POST: ['Content-Type', 'Authorization', 'Accept'],
    PUT: ['Content-Type', 'Authorization', 'Accept'],
    DELETE: ['Authorization']
};

function isValidHeader(method, header) {
    const headers = validHeaders[method];
    return headers && headers.includes(header);
}

// Обработчик события изменения типа ввода для body
bodyInputType.addEventListener('change', function () {
    const selectedInputType =
document.querySelector('input[name="bodyType"]:checked').value;
    if (selectedInputType === 'none') {
        bodyTable.style.display = 'none';
        bodyText.style.display = 'none';
    } else if (selectedInputType === 'table') {
        bodyTable.style.display = 'table';
        bodyText.style.display = 'none';
    } else if (selectedInputType === 'text') {
        bodyTable.style.display = 'none';
        bodyText.style.display = 'block';
    }
});

function addBodyRow() {
    const tbody = bodyTable.querySelector('tbody');
    const newRow = document.createElement('tr');

    const nameCell = document.createElement('td');
    const nameInput = document.createElement('input');
    nameInput.type = 'text';
    nameInput.className = 'body-name';
    nameInput.placeholder = 'Name';
    nameCell.appendChild(nameInput);
    newRow.appendChild(nameCell);

    const valueCell = document.createElement('td');
```

```
    const valueInput = document.createElement('input');
    valueInput.type = 'text';
    valueInput.className = 'body-value';
    valueInput.placeholder = 'Value';
    valueCell.appendChild(valueInput);
    newRow.appendChild(valueCell);

    const controlsCell = document.createElement('td');
    const removeButton = document.createElement('button');
    removeButton.type = 'button';
    removeButton.textContent = '-';
    removeButton.onclick = () => removeBodyRow(newRow);
    controlsCell.appendChild(removeButton);
    newRow.appendChild(controlsCell);

    tbody.appendChild(newRow);
}

function removeBodyRow(row) {
    const tbody = bodyTable.querySelector('tbody');
    tbody.removeChild(row);
}

// Update header autocomplete options based on selected method
function updateHeaderAutocomplete() {
    const method = methodSelect.value;
    const headerNames = validHeaders[method] || [];
    const headerNamesDatalist = document.getElementById('headerNames');

    // Clear existing options
    headerNamesDatalist.innerHTML = '';

    // Create new options
    headerNames.forEach(header => {
        const option = document.createElement('option');
        option.value = header;
        headerNamesDatalist.appendChild(option);
    });

    // Update autocomplete for new header fields
    const headerNameInputs = Array.from(document.querySelectorAll('.header-name'));
    headerNameInputs.forEach(input => {
        input.setAttribute('list', 'headerNames');
    });
}

function clearHeadersTable() {
    const tbody = headersTable.querySelector('tbody');
    const rows = tbody.querySelectorAll('tr');

    // Удаляем все строки, начиная со второй
    for (let i = 1; i < rows.length; i++) {
        rows[i].remove();
    }

    // Очищаем значения первой строки
    const defaultRow = rows[0];
    const nameInput = defaultRow.querySelector('.header-name');
    const valueInput = defaultRow.querySelector('.header-value');
    nameInput.value = '';
    valueInput.value = '';
    updateHeaderAutocomplete();
}
```

```
function removeHeaderRow(row) {
    const tbody = headersTable.querySelector('tbody');
    tbody.removeChild(row);
}

function addHeaderRow() {
    const tbody = headersTable.querySelector('tbody');
    const newRow = document.createElement('tr');

    const nameCell = document.createElement('td');
    const nameInput = document.createElement('input');
    nameInput.type = 'text';
    nameInput.className = 'header-name';
    nameInput.placeholder = 'Name';
    nameInput.setAttribute('onchange', 'updateHeaderValueAutocomplete(this)');
    nameCell.appendChild(nameInput);
    newRow.appendChild(nameCell);

    const valueCell = document.createElement('td');
    const valueInput = document.createElement('input');
    valueInput.type = 'text';
    valueInput.className = 'header-value';
    valueInput.placeholder = 'Value';
    valueCell.appendChild(valueInput);
    newRow.appendChild(valueCell);

    const controlsCell = document.createElement('td');
    const removeButton = document.createElement('button');
    removeButton.type = 'button';
    removeButton.textContent = '-';
    removeButton.onclick = () => removeHeaderRow(newRow);
    controlsCell.appendChild(removeButton);
    newRow.appendChild(controlsCell);

    tbody.appendChild(newRow);

    updateHeaderAutocomplete(); // Обновляем автозаполнение после добавления нового
поля заголовка
}

function updateHeaderValueAutocomplete(input) {
    const selectedHeader = input.value;
    const row = input.closest('tr');
    const valueInput = row.querySelector('.header-value');
    const headerValues = document.getElementById('headerValues');

    // Clear existing options
    headerValues.innerHTML = '';

    if (selectedHeader === 'Content-Type') {
        // Define suggested values for "Content-Type"
        const contentTypes = ['application/json', 'application/xml', 'application/x-
www-form-urlencoded', 'text/plain'];

        // Create new options
        contentTypes.forEach(contentType => {
            const option = document.createElement('option');
            option.value = contentType;
            headerValues.appendChild(option);
        });
    }
```

```
    // Update autocomplete for value input field
    valueInput.setAttribute('list', 'headerValues');
}

function parseBodyTable() {
    const bodyData = {};
    const tbody = bodyTable ? bodyTable.querySelector('tbody') : null;

    if (tbody) {
        const rows = Array.from(tbody.querySelectorAll('tr'));
        rows.forEach((row) => {
            const nameInput = row.querySelector('.body-name');
            const valueInput = row.querySelector('.body-value');
            const name = nameInput ? nameInput.value.trim() : '';
            const value = valueInput ? valueInput.value.trim() : '';
            if (name !== '' && value !== '') {
                bodyData[name] = value;
            }
        });
    }

    const params = new URLSearchParams(bodyData);
    return params.toString();
}

// Add event listener to the form
form.addEventListener('submit', async (e) => {
    e.preventDefault();



    var url;
    if (toggleButton.checked) {
        url = hostUrl + endpointInput.value;
    } else {
        url = urlInput.value;
    }

    const method = methodSelect.value;

    const headers = {};
    const headerRows = Array.from(headersTable.querySelectorAll('tbody tr'));
    headerRows.forEach((row) => {
        const nameInput = row.querySelector('.header-name');
        const valueInput = row.querySelector('.header-value');
        const name = nameInput.value.trim();
        const value = valueInput.value.trim();
        if (name !== '' && value !== '' && isValidHeader(method, name)) {
            headers[name] = value;

        }

    });

    let body = '';
    const selectedInputType =
document.querySelector('input[name="bodyType"]:checked').value;
    if (selectedInputType === 'table') {
        body = parseBodyTable();
    } else if (selectedInputType === 'text') {
        body = bodyText.value;
    }
```

```
    removeAllChildNodes(requestViewContainer);
    requestViewPreContainer.textContent = "";
    resultContainer.innerHTML = '';

    const config = {
        method,
        url,
        headers,
        data: body
    };

    try {
        const response = await axios(config);

        displayRequest(config);
        if (response.status !== 204) {
            displayRequestView(response);
        } else {
            preContainerCleanAndInvisibility(requestViewPreContainer);
            requestViewH.style.display = 'none';
        }
        displayResponse(response);
        displayResult(response.data);
    } catch (error) {
        if (error.response !== null && error.response !== undefined) {
            displayRequest(config);
            displayError(error);
        }
    }
});

function displayRequest(request) {
    const { method, url, headers, data } = request;

    let requestText = `${method} ${url}\n\n`;

    requestText += 'Headers:\n';
    for (const [name, value] of Object.entries(headers)) {
        requestText += `${name}: ${value}\n`;
    }

    requestText += '\nRequest Body:\n';

    // Check if the data is JSON and format it accordingly
    if (typeof data === 'object') {
        requestText += decodeURIComponent(parseBodyTable());

    } else {
        requestText += data;
    }

    requestContainer.textContent = requestText;
}

function displayRequestView(response) {
    if (!toggleButton.checked) {
        requestViewH.style.display = 'none';
        preContainerCleanAndInvisibility(requestViewPreContainer);
        return;
    }
    removeAllChildNodes(requestViewContainer);

    const data = response.data.requestDetails;
```

```
    requestViewH.style.display = 'block';

    for (const [name, value] of Object.entries(data)) {
        if (name.toLowerCase() == 'headers') {
            addHeadersTable(requestViewContainer, value);
            continue;
        }
        if (name.toLowerCase() == 'body') {
            displayBody(value, requestViewPreContainer);
            continue;
        }

        let textNode = document.createTextNode(`${name}: ${JSON.stringify(value,
null, 2)}`);
        requestViewContainer.appendChild(textNode);
        addNewLine(requestViewContainer);
    }
}

function displayBody(value, preContainer) {
    if (value !== null && value !== "" && value !== undefined) {
        preContainer.style.display = 'block';
        let bodyText = 'Body:\n'
        bodyText += JSON.stringify(value, null, 2);
        preContainer.textContent = bodyText;
    } else {
        preContainerCleanAndInvisibility(preContainer);
    }
}



function displayResponse(response) {
    removeAllChildNodes(responseContainer);

    const headers = response.headers;
    const data = response.data.data;


    let statusTextNode = document.createTextNode(`Status: ${response.status} `);
    const infoIcon = document.createElement('i');
    infoIcon.className = 'fa fa-question-circle';
    infoIcon.title = httpStatusCodes[response.status];
    responseContainer.appendChild(statusTextNode);
    responseContainer.appendChild(infoIcon);
    addNewLine(responseContainer);


    let headersTextNode = document.createTextNode('Headers:');
    responseContainer.appendChild(headersTextNode);
    addNewLine(responseContainer);

    addHeadersTable(responseContainer, headers);

    if (toggleButton.checked) {
        displayBody(data, responsePreContainer);
    } else {
        displayBody(response.data, responsePreContainer);
    }


}

function displayResult(data) {
```

```
    if (toggleButton.checked) {
        return;
    }

    resultContainer.innerHTML = '';

    const iframe = document.createElement('iframe');
    iframe.srcdoc = data;
    iframe.style.width = '100%';
    iframe.style.height = '500px';

    resultContainer.appendChild(iframe);
}


function displayError(error) {
    removeAllChildNodes(responseContainer);

    let response = error.response;
    let statusTextNode = document.createTextNode(`Status: ${response.status} `);
    const infoIcon = document.createElement('i');
    infoIcon.className = 'fa fa-question-circle';
    infoIcon.title = httpStatusCodes[response.status];
    responseContainer.appendChild(statusTextNode);
    responseContainer.appendChild(infoIcon);
    addNewLine(responseContainer);

    let headersTextNode = document.createTextNode('Headers:');
    responseContainer.appendChild(headersTextNode);
    addNewLine(responseContainer);

    addHeadersTable(responseContainer, response.headers);

    if (response.data !== null && response.data !== "") {
        if (response.data.succeeded !== null && response.data.succeeded !==
undefined && !response.data.succeeded) {

            displayRequestView(response);
            if (response.data.data !== null && response.data.data !== undefined &&
response.data.data !== "") {
                let errorText = 'Data:\n'
                errorText += JSON.stringify(response.data.data, null, 2);
                responsePreContainer.style.display = 'block';
                responsePreContainer.textContent = errorText;
            } else {
                preContainerCleanAndInvisibility(responsePreContainer);
            }

        } else {
            let errorText = 'Data:\n'
            errorText += JSON.stringify(response.data, null, 2);
            responsePreContainer.style.display = 'block';
            responsePreContainer.textContent = errorText;
            preContainerCleanAndInvisibility(requestViewPreContainer);
        }
    } else {
        preContainerCleanAndInvisibility(responsePreContainer);
        preContainerCleanAndInvisibility(requestViewPreContainer);
        requestViewH.style.display = 'none';
    }
}

function removeAllChildNodes(parent) {
```

```
    while (parent.firstChild) {
        parent.removeChild(parent.firstChild);
    }
}

function addNewLine(container) {
    var br = document.createElement("span");
    br.innerHTML = "<br/>";
    container.appendChild(br);
}

function preContainerCleanAndInvisibility(container) {
    container.textContent = "";
    container.style.display = 'none';
}

function addHeadersTable(container, headers) {
    // Create a table and add styles
    const headersTable = document.createElement('table');
    headersTable.style.width = '100%';
    headersTable.style.borderCollapse = 'collapse';

    // For each header, create a new row in the table
    for (const [name, value] of Object.entries(headers)) {
        const row = headersTable.insertRow();

        // Insert a cell for the header name and add styles
        const nameCell = row.insertCell();
        nameCell.style.border = '1px solid black';
        nameCell.style.padding = '5px';

        // If the header is in the dictionary, add an info icon with the description
as the title
        if (name.toLowerCase() in httpHeadersDictionary) {
            const infoIcon = document.createElement('i');
            infoIcon.className = 'fa fa-question-circle';
            infoIcon.title = httpHeadersDictionary[name.toLowerCase()];
            nameCell.appendChild(document.createTextNode(name + ' '));
            nameCell.appendChild(infoIcon);
        }
        else {
            nameCell.textContent = name;
        }

        // Insert a cell for the header value and add styles
        const valueCell = row.insertCell();
        valueCell.textContent = value;
        valueCell.style.border = '1px solid black';
        valueCell.style.padding = '5px';
    }

    // Add the created table to your container
    container.appendChild(headersTable);
}

// Initialize header autocomplete based on the default method
updateHeaderAutocomplete();
```

1.15 Index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>HTTP Simulator</title>
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <link rel="stylesheet" href="css/font-awesome/css/font-awesome.min.css">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin-bottom: 20px;
            max-width: 100%;
        }

        h1 {
            background-color: #4CAF50;
            text-align: center;
            margin-top: 0px;
            padding-top: 20px;
            padding-bottom: 20px;
            font-weight: 700;
            color: white;
        }

        h2 {
            font-size: 20px;
            font-weight: 700;
        }

        form {
            margin-bottom: 20px;
        }

        label {
            display: block;
            font-weight: 700;
            margin-top: 10px;
            font-size: 20px;
        }

        input[type="text"],
        select,
        textarea {
            width: 100%;
            padding: 5px;
        }

        button[type="submit"] {
            margin-top: 10px;
            padding: 5px 10px;
            background-color: #4CAF50;
            color: white;
            border: none;
            cursor: pointer;
            border-radius: 5px;
        }
```

```
    button[type="submit"]:hover {
        background-color: #3e8e41;
        color: #fff;
    }

pre {
    background-color: #f5f5f5;
    padding: 10px;
    overflow-x: auto;
}

#resultContainer {
    border: 1px solid #ddd;
    padding: 10px;
}

iframe {
    width: 100%;
    height: 500px;
    border: none;
}

table {
    width: 100%;
    margin-bottom: 10px;
}

th {
    background-color: #f5f5f5;
    padding: 5px;
    text-align: left;
}

td {
    padding: 5px;
}

.span-url {
    font-size: medium;
}

.toggle-button {
    position: relative;
    display: inline-block;
    width: 40px;
    height: 35px;
    padding-top: 12px;
    margin-bottom: 10px;
}

    .toggle-button input {
        opacity: 0;
        width: 0;
        height: 0;
    }

.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
```

```
    background-color: #ccc;
    -webkit-transition: .4s;
    transition: .4s;
}

    .slider:before {
        position: absolute;
        content: "";
        height: 15px;
        width: 15px;
        left: 3px;
        bottom: 3px;
        background-color: white;
        -webkit-transition: .4s;
        transition: .4s;
    }

input:checked + .slider {
    background-color: #4CAF50;
}

input:focus + .slider {
    box-shadow: 0 0 1px #4CAF50;
}

input:checked + .slider:before {
    -webkit-transform: translateX(19px);
    -ms-transform: translateX(19px);
    transform: translateX(19px);
}

/* Rounded sliders */
.slider.round {
    border-radius: 34px;
}

    .slider.round:before {
        border-radius: 50%;
    }

/* Стили для кнопки */
.help-button {
    position: fixed;
    top: 50%;
    right: -35px;
    /* Кнопка прикреплена к правому краю с отступом 20px */
    transform: translateY(-50%) rotate(-90deg);
    /* Поворот на 90 градусов */
    padding: 10px 20px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: right 0.3s ease-out;
    /* Плавный переход при анимации */
}

    .help-button:hover {
        background-color: #3e8e41;
        color: #fff;
    }
```

```
/* Стили для окна */
.help-window {
    position: fixed;
    top: 0;
    right: 0;
    width: calc(33.33% - 40px);
    /* Ширина окна (1/3 экрана) */
    height: 100%;
    padding: 80px;
    background-color: #f2f2f2;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
    transform: translateX(100%);
    /* Изначально окно сдвинуто за правую границу экрана */
    transition: transform 0.3s ease-out;
    /* Плавный переход при анимации */
    font-size: large;
    text-align: justify;
}



.left-section {
    position: absolute;
    left: 0;
    width: 95%;
    /* Изначально левая часть занимает 90% экрана */

    padding: 20px;
    padding-top: 0px;
    background-color: #fff;
    transition: width 0.3s ease-out;
    /* Плавный переход при анимации */
    overflow-y: auto;
    /* Разрешаем вертикальную прокрутку */
}

.method-select {
    border: 2px solid #ccc;
    padding: 5px;
    transition: border-color 0.3s ease;
}

    .method-select:focus {
        outline: none;
        border-color: #4CAF50;
    }



.header-name {
    border: 2px solid #ccc;
    padding: 5px;
    transition: border-color 0.3s ease;
}

    .header-name:focus {
        outline: none;
        border-color: #4CAF50;
    }
```

```
        .header-value {
            border: 2px solid #ccc;
            padding: 5px;
            transition: border-color 0.3s ease;
        }

            .header-value:focus {
                outline: none;
                border-color: #4CAF50;
            }

        .body-name {
            border: 2px solid #ccc;
            padding: 5px;
            transition: border-color 0.3s ease;
        }

            .body-name:focus {
                outline: none;
                border-color: #4CAF50;
            }

        .body-value {
            border: 2px solid #ccc;
            padding: 5px;
            transition: border-color 0.3s ease;
        }

            .body-value:focus {
                outline: none;
                border-color: #4CAF50;
            }
    </style>
</head>



<body>
    <h1>HTTP Simulator</h1>

    <!-- Окно с информацией -->
    <div class="help-window"></div>

    <!-- Кнопка "Help Me" -->

    <button class="help-button">
        <span class="glyphicon glyphicon-question-sign"></span> Get help
    </button>

    <div class="left-section">
        <form id="requestForm">
            <h2>URL Input Type:</h2>
            <span class="span-url">Custom URL</span>
            <div class="toggle-button">
                <input type="checkbox" id="toggle" name="urlType" value="custom">
                <label class="slider round" for="toggle"></label>
            </div>
            <span class="span-url">Endpoints</span>

            <div id="urlInputs">
                <input type="text" placeholder="Endpoint address" style="display:
none;" list="endpointsValues" id="endpointInput" class="method-select">
                <datalist id="endpointsValues">
```

```
                    <option value="/api/RequestDetails/GetRequest"></option>
                    <option value="/api/RequestDetails/PostRequest"></option>
                    <option value="/api/Crud/GetUser"></option>
                    <option value="/api/Crud/GetAllUsers"></option>
                    <option value="/api/Crud/PostUser"></option>
                    <option value="/api/Crud/PutUser"></option>
                    <option value="/api/Crud/DeleteUser"></option>
                </datalist>
                <!-- Custom URL Input -->
                <input type="text" id="urlInput" placeholder="Enter custom URL"
style="display: block;" class="method-select">
            </div>

            <label for="methodSelect">Method:</label>
            <select id="methodSelect" onchange="updateHeaderAutocomplete()"
class="method-select">
                <option value="GET">GET</option>
                <option value="POST">POST</option>
                <option value="PUT">PUT</option>
                <option value="DELETE">DELETE</option>
            </select>

            <h2>Headers:</h2>
            <table id="headersTable" class="table table-striped">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Value</th>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>
                            <input type="text" class="header-name"
placeholder="Name" list="headerNames"
                                   onchange="updateHeaderValueAutocomplete(this)">

                            <datalist id="headerNames"></datalist>
                        </td>

                        <td>
                            <input type="text" class="header-value"
placeholder="Value" list="headerValues">
                            <datalist id="headerValues"></datalist>
                        </td>
                        <td class="header-controls">
                            <button type="button"
onclick="addHeaderRow()">+</button>
                        </td>
                    </tr>
                </tbody>
            </table>

            <label for="bodyInputType">Body Input Type:</label>
            <div id="bodyInputType">
                <input type="radio" name="bodyType" value="none" checked> None
                <input type="radio" name="bodyType" value="table"> Table
                <input type="radio" name="bodyType" value="text"> Text
            </div>

            <div id="bodyInputs">
                <!-- Таблица для ввода данных -->
```

```
                <table id="bodyTable" style="display: none;" class="table table-
striped">
                    <thead>
                        <tr>
                            <th>Name</th>
                            <th>Value</th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>
                                <input type="text" class="body-name"
placeholder="Name">
                            </td>
                            <td>
                                <input type="text" class="body-value"
placeholder="Value">
                            </td>
                            <td class="body-controls">
                                <button type="button"
onclick="addBodyRow()">+</button>
                            </td>
                        </tr>
                    </tbody>
                </table>

                <!-- Поле для ввода произвольного текста -->
                <textarea id="bodyText" rows="4" placeholder="Request body"
style="display: none;"
                          class="method-select"></textarea>
            </div>

            <button type="submit">Send Request</button>
        </form>

        <h2>HTTP Request:</h2>
        <pre id="requestContainer"></pre>

        <h2 style="display: none;" id="requestViewH">The request view on the
server:</h2>
        <div id="requestViewContainer"></div>
        <pre id="requestViewPreContainer" style="display: none;"></pre>

        <h2>Response:</h2>
        <div id="responseContainer"></div>
        <pre id="responsePreContainer" style="display: none;"></pre>

        <h2 style="display: block;" id="resultH">Result:</h2>
        <div id="resultContainer" style="display: block;"></div>
    </div>

    <script src="js/script.js"></script>
</body>

</html>
```