КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ГЕНЕРАЦІЯ REST API ІЗ ВИКОРИСТАННЯМ PROTO ФАЙЛІВ

Виконав студент 4-го курсу
Романенко Євгеній Миколайович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Кузенко Володимир Федорович

(підпис)

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

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичних основ комп'ютерних наук
«____» _____ 2021 р.,
протокол № ___
Завідувач кафедри
М.С. Нікітченко

(підпис)

Київ - 2021

РЕФЕРАТ

АВТОГЕНЕРАЦІЯ, RESP API.

Обсяг роботи 52 сторінок, 12 ілюстрацій, 5 таблиць, 12 джерела посилань.

Об’єктом розроблення програмного засобу є процес автогенерації різних частин RESP API: серверу та клієнта. Предметом роботи є система, яка складається з 2-х частин: парсера та генераторів.

Метою роботи є створення додатку для генерації різних частин REST API, який легко підтримувати впродовж тривалого часу. Додавання нового функціоналу не повинно викликати труднощів.

Методи розроблення: розробка програмного продукту, архітектурні рішення. Інструменти розроблення: операційна система - Windows 10, Visual Studio 2019, ANTLR4, платформа .NET5.

Результати роботи: Спроектований та реалізований розширюваний, масштабований, додаток. У додатку використані знання про роботу с ANTLR4. Вибрана архітектура дозволяє легко підтримувати, тестувати та продовжувати розробку.

За методами розробки та інструментальними засобами робота виконувалася сумісно з пошуком інформації в всесвітній мережі.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ANTLR – Another Tool for Language Recognition

ATAG – Another Tool for Api Generation

REST – Representational State Transfer

Web Api – Web Application Programming Interface

ВСТУП

Оцінка сучасного стану об’єкта розробки. REST API - один із найпоширеніших стандартів взаємодії Web додатків. Платформа .NET та ASPNET WebApi фреймворк є розповсюдженими інструментами розробки Restful сервісів, тому розроблена система дозволить пришвидшити розробку додатків за домогою генерації коду.

Актуальність роботи та підстави для її виконання. З ростом інтернету все більше представників різного бізнесу прагнуть створити онлайн представництва, тому потреба у Web додатках росте з кожним днем. Додаток дозволить значно пришвидшити розробку Web додатків, економлячи час розробників та гроші клієнтів. Запропоноване рішення дозволить лише з одного proto файлу генерувати код як сервера, так і клієнта. На даний момент існують подібні технології, але вони не дозволяють виконувати повноцінну генерацію як сервера так і клієнта.

Мета й завдання роботи. Метою роботи є розробка додатку для генерації коду сервера та клієнта на основі proto файлу. Код додатку повинен мати масштабовану архітектуру, додавання нового функціоналу не повинно викликати змін в існуючому коді.

Для досягнення цієї мети поставлено такі завдання:

- дослідити існуючі на ринку застосунки;
- спроектувати архітектуру застосунку;
- розробити зрозумілий у використанні клієнт для програмного інтерфейсу;

Об'єкт, методи й засоби розроблення. Об’єктом розроблення програмного засобу «ATAG» є процес створення консольного додатку. Предметом роботи є система, яка складається з парсера proto файлу та генератора файлів коду.
В якості інструментів створення та тестування програмного засобу було обрано наступні інструменти:

- Visual Studio 2019 - інтегроване середовище розробки мовою програмування C#;
- Git – необхідний інструмент для командної розробки, розподілена система контроля версій. SourceTree – клієнт для git з GUI.

Можливі сфери застосування. Застосунок орієнтовний на розробників ПО, які за його допомогою можуть значно скоротити час розробки комплексного Web додатку.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ НА РИНКУ СИСТЕМ

NSwagStudio – додаток, який дозволяє генерувати Api клієнта на основі json, який генерується з вже існуючого серверу (рис 1.1, рис 1.2).

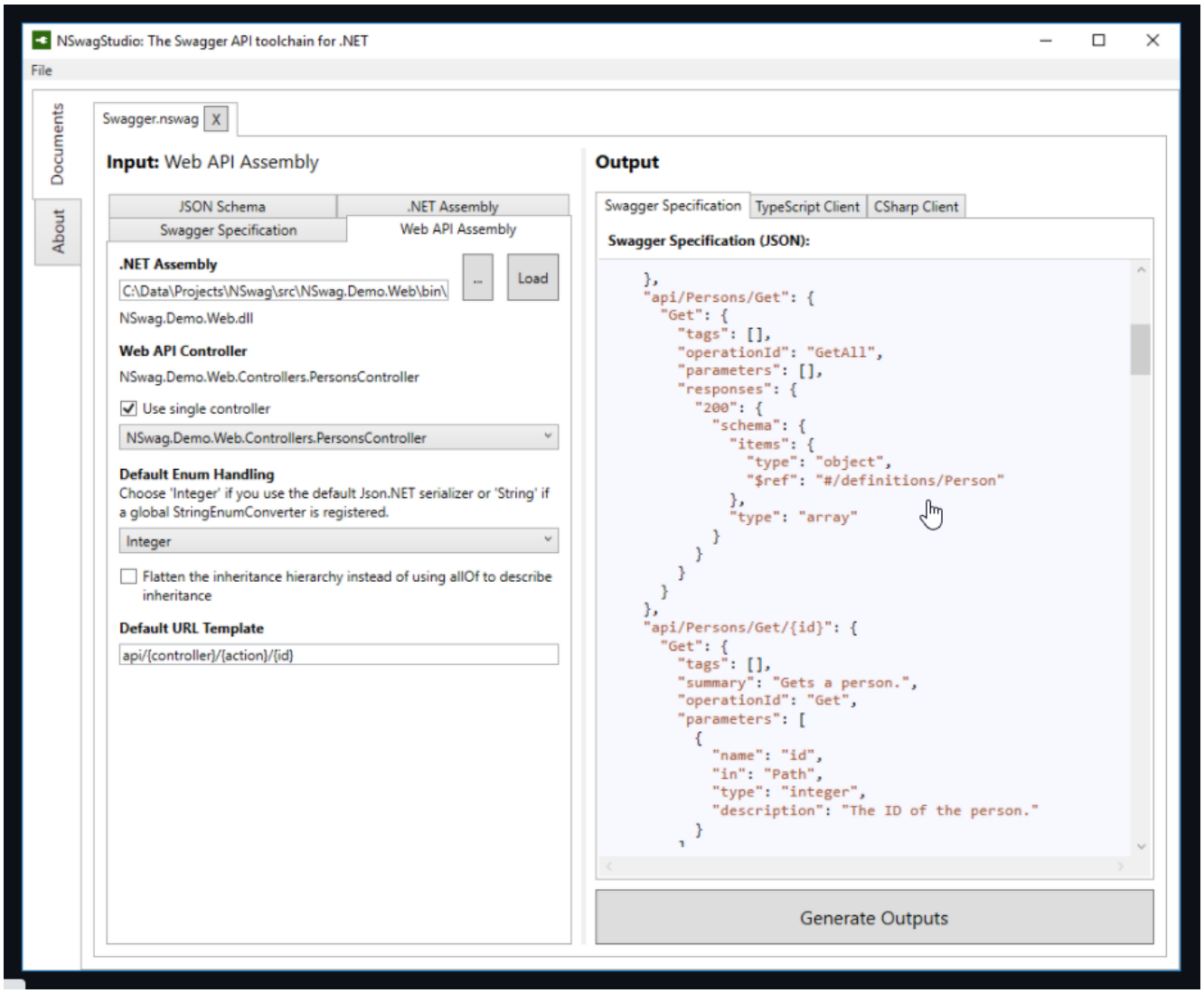


Рисунок 1.1 – приклад інтерфейсу NSwagStudio

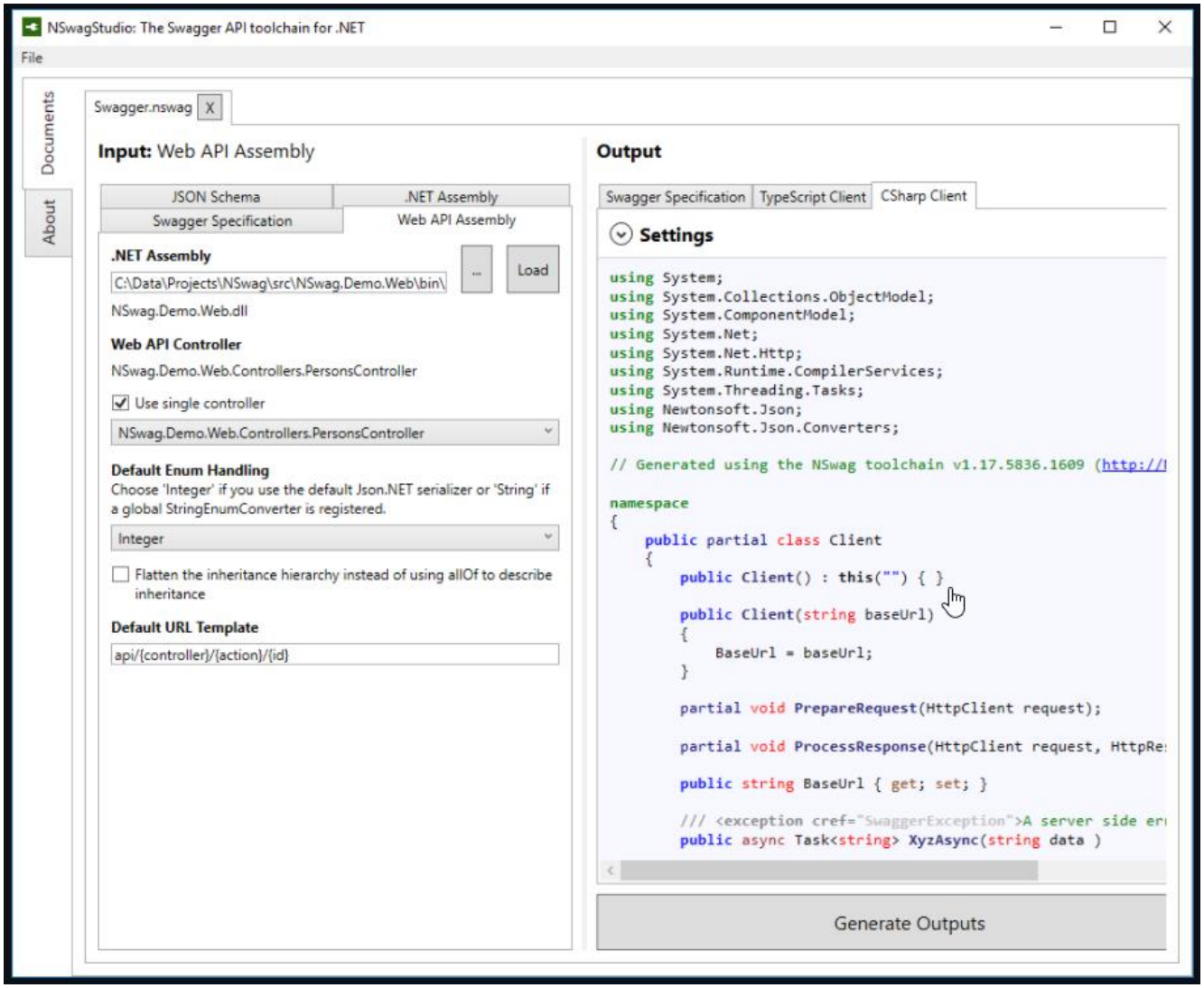


Рисунок 1.2 - приклад інтерфейсу NSwagStudio

Основна перевага розробленого застосунку у тому, що інтерфейс сервера також автогенерується, а також замість json використовується розроблена мова proto файлу, що дозволяє більш щільно нотувати елементи сервера та клієнта.

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Огляд технології ANTLR

ANTLR – це потужний генератор парсерів для читання, обробки, виконання або перекладання структурованого тексту або бінарних файлів. Він використовується для будування мов, інструментів та фреймворків. На основі граматики ANTLR генерує парсер, який може будувати дерево виразів, та обходити його [2].

2.2. Огляд платформи .NET

.NET (читається дот-нет) — програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків. Багато в чому є продовженням ідей та принципів, покладених в технологію Java. Однією з ідей .NET є сумісність служб, написаних різними мовами. Хоча ця можливість рекламується Microsoft як перевага .NET, платформа Java має таку саму можливість.

Середовища розробки .NET-програм: Visual Studio .NET (C++, C#, J#), SharpDevelop, Borland Developer Studio (Delphi, C#) тощо. Середовище Eclipse має додаток для розробки .NET-програм. Застосовні програми також можна розроблювати в текстовому редакторі та використовувати консольний компілятор.

Слід зазначити, що один з перших JIT-компіляторів для Java був також розроблений фірмою Microsoft (тепер в Java використовується досконаліша багаторівнева компіляція — Sun HotSpot). Сучасна технологія динамічної компіляції дозволяє досягнути аналогічного рівня швидкодії з традиційними «статичними» компіляторами (наприклад, C++) і питання швидкодії часто залежить від якості того чи іншого компілятора.[3]

2.2. Огляд платформи .NET Core

.NET Core - це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Сумісна з такими операційними системами як Windows, Linux і macOS. Була випущена компанією Microsoft. У платформи є власне співтовариство на GitHub. Підтримує наступні мови програмування: C #, Visual Basic .NET (частково) і F # .

.NET Core заснована на .NET Framework. Платформа .NET Core відрізняється від неї модульностью, кроссплатформеностю, можливістю застосування хмарних технологій, і тим, що в ній відбувся поділ між бібліотекою CoreFX і середовищем виконання CoreCLR.

.NET Core - модульна платформа. Кожен її компонент оновлюється через менеджер пакетів NuGet, а значить можна оновлювати її модулі окремо, в той час як .NET Framework оновлюється цілком. Кожна програма може працювати з різними модулями і не залежить від єдиного поновлення платформи.

CoreFX - це бібліотека, інтегрована в .NET Core. Серед її компонентів: System.Collections, System.IO, System.Xml.

CoreCLR - це середовище виконання, що включає в себе RyuJIT (JIT-компілятор), вбудований збирач сміття та інші компоненти.[4]

2.3. Огляд фреймворку NUnit

NUnit — відкрите середовище модульного тестування застосунків для .NET. Воно було перенесене з мови Java (бібліотека JUnit). Перші версії NUnit були написані на J#, але потім весь код був переписаний на C# з використанням таких нововведень .NET, як атрибути.

Існують також відомі розширення оригінального пакету NUnit, значна частина з них також з відкритим вихідним кодом. NUnit.Forms доповнює NUnit засобами тестування елементів користувацького інтерфейсу Windows Forms.[7]

2.4. Огляд технології Git

Git – на сьогоднішній день одна з самих популярних розподілених систем контроля версій. Система контроля версій – програмне забезпечення для полегшення роботи зі змінною інформацією. Система контроля версій дозволяє зберігати кілька версій одного документу та при необхідності повертатися до попередніх версій та визначати, хто зробив останні зміни у файлу. Дуже корисний для розробки ПЗ в командах для >=1 чоловік.[16]

SourceTree – десктопний клієнт для git. Використовується для візуалізації і керування репозиторієм git.[17]

GitHub – один з найкрупніших веб-сервісов для хостингу проектів та спільної розробки. Веб-клієнт для git.[18]

2.6 Protocol buffers файли (proto файли) та їх використання

Protocol Buffers файли мають строгу, загальну специфікацію, що зберігає час розробникам. У цих файлах описується контракт взаємодії – імена методів, параметри методів, повертаємий тип та інша додаткова інформація. Це є загальна інструкція, за якою відбувається генерація коду різних частин програми (клієнту та серверу). Тобто розширення функціоналу генератору коду дозволяє на базі proto файлів створювати крос-платформені додатки за допомогою генерації.

gRPC – Один із проектів, який використовує proto файли. Це проект з відкритим доступом, який був розроблений компанією Google у 2015. [12]

2.7 REST, Web API

REST – це архітектурний стиль програмного забезпечення. Використовує протокол HTTP. Цей підхід використовується при створені веб-сервісів. Він дозволяє писати надійні, масштабовані веб-додатки. Це набор принципів, яких слід дотримуватися під час написання програм. Означені наступні основні принципи:

- взаємодія виду клієнти-сервер;
- під час спілкування клієнту та серверу стан не зберігається;
- має бути наявна можливість кешування даних, які не змінюються часто;
- спілкування має проходити за уніфікованим інтерфейсом.

Веб-додаток складається з контролерів та їх методів, кожен з яких має тип(get, post, put…) та унікальний шлях. У контексті .NET розробки контролер – це клас, якій містить у собі список методів. Кожен контролер має свій унікальний шлях, і в свою чергу кожен метод контролера має унікальних шлях, який містить у собі шлях до контролера.[11]

РОЗДІЛ 3. ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ СИСТЕМИ. ВИМОГИ ДО СИСТЕМИ

3.1. Призначення системи

Дана система призначена прискорення процесу розробки REST Web Api сервера та консольного клієнта Web Api шляхом автогенерації коду контролерів, моделей та клієнта.

3.2. Цілі створення системи

Систему було створено з метою:

- забезпечення зручного консольного додатку для автогенерації коду;
- прискорення розробки Web додатків та їх клієнтів;

3.3. Вимоги до системи в цілому

Система повинна мати масштабовану архітектуру. Має бути логічно структурована та однозначна.

В системі повинні бути передбаченій наступні функціональні можливості:

- валідація вхідного proto файлу, на основі якого відбувається автогенерація коду;
- парсинг вхідного файлу;
- створення моделей на основі вхідного файлу, за якими відбувається генерація;
- генерації коду моделей;
- генерація коду контролерів;

- генерація коду консольного клієнту для Api;

Функції, виконувані системою, мають бути простими у редагуванні, для випадків змін або розширення функціоналу.

3.4. Технічні вимоги

До технічних вимог відносяться:

1. Додаток розроблений на .NET5, тому коректна робота забезпечується на усіх популярних OS, таких як Windows, Linux та MacOS.
2. У випадку генерації коду контролерів для коректної роботи згенерованого коду у проєкті, в якому він буде використовуватися, проєкт має мати тип : «Microsoft.NET.Sdk.Web» та платформу .NET Core 3.1+.
3. У випадку генерації коду клієнта для коректної роботи згенерованого коду у проєкті, в якому він буде використовуватися**,** проєкт має використовувати платформу .NET Core 3.1+

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1. Діаграми проекту

Use-Case діаграма зображена на рисунку 4.1:

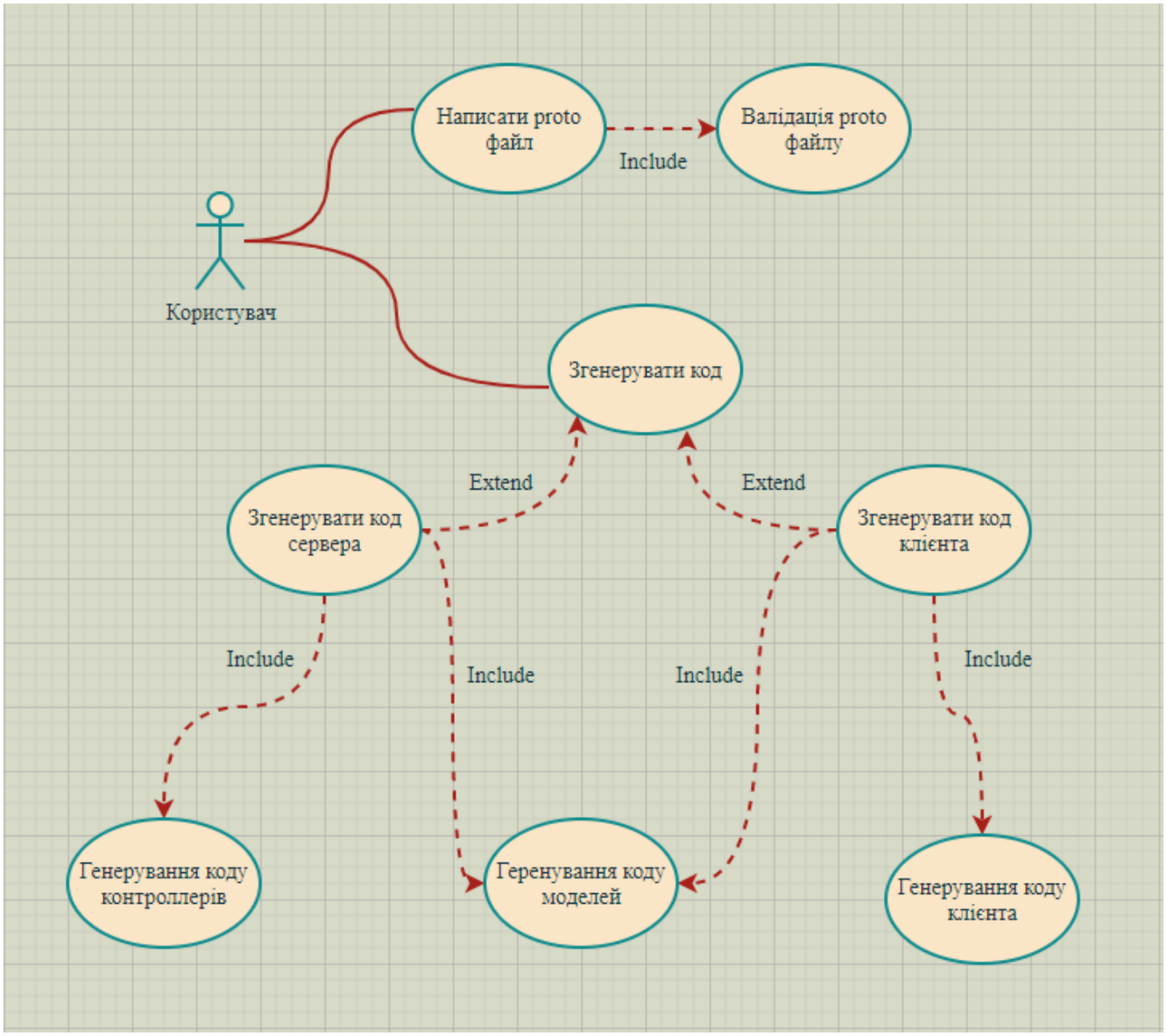


Рисунок 4.1 – діаграма прецендентів

Діаграма класів на рисунку 4.2, елемент Models більш детально описаний у наступній діаграмі на рисунку 4.3:

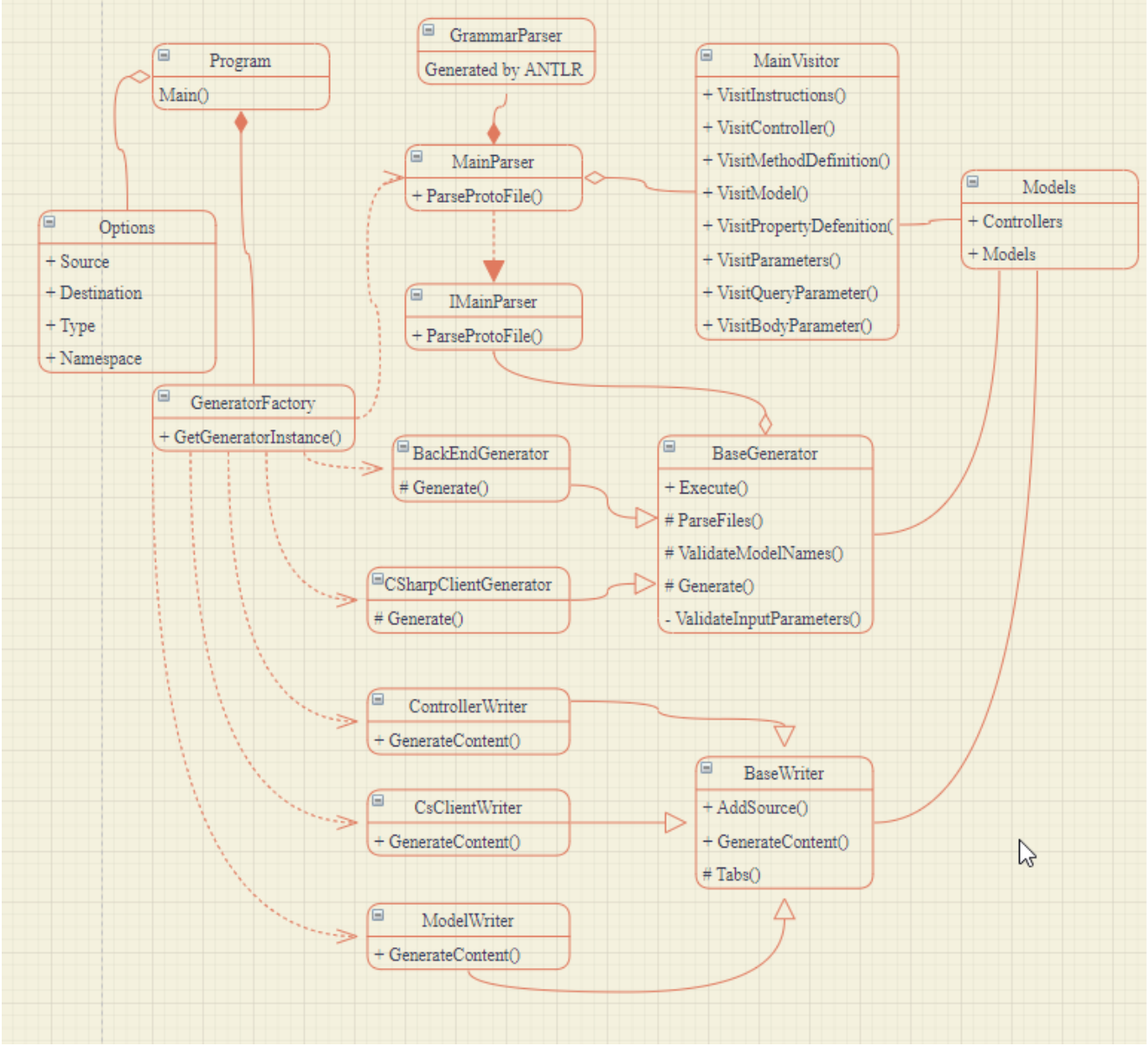


Рисунок 4.2 – діаграма класів

Діаграма елементу Models:

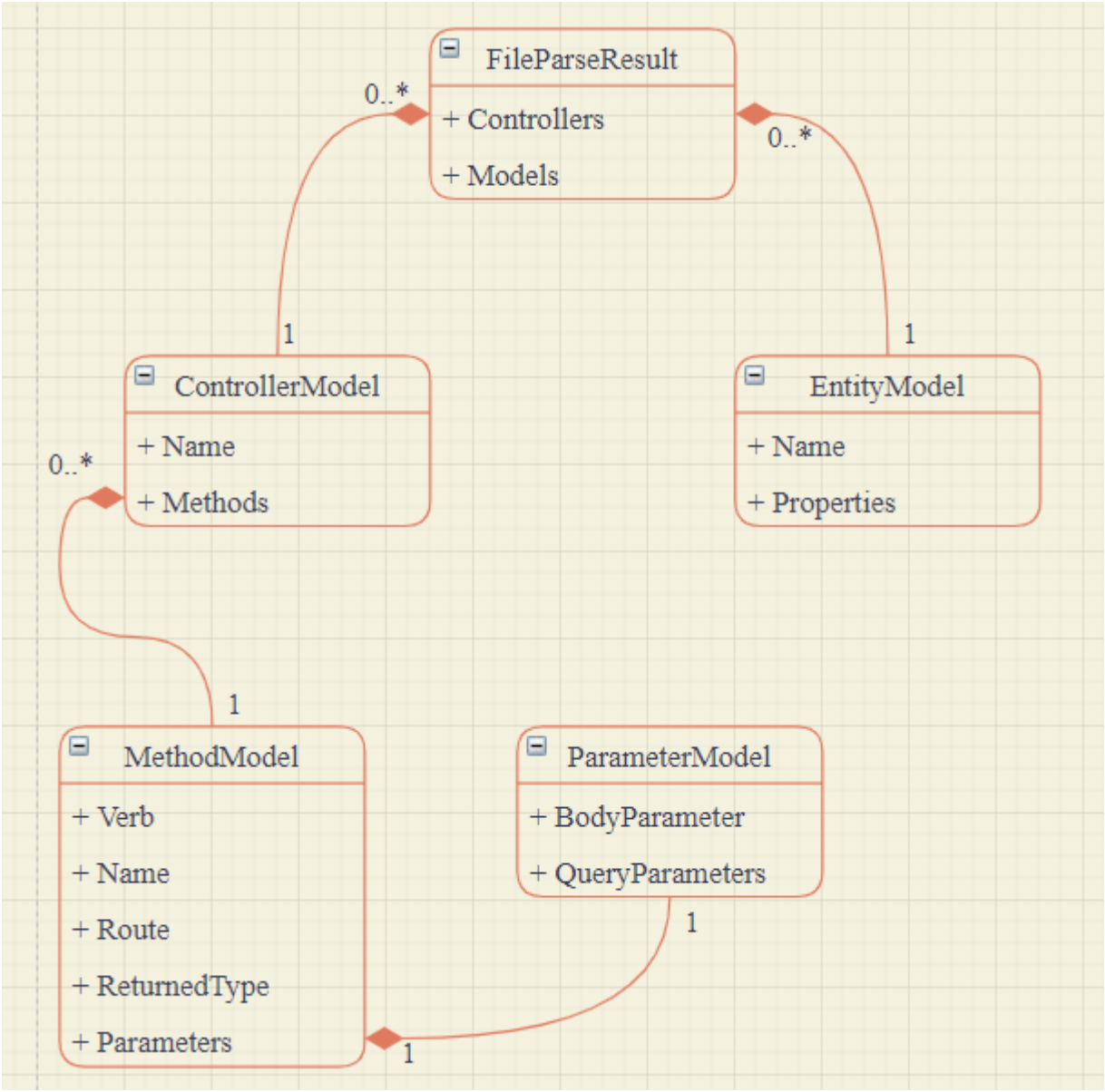


Рисунок 4.3 – діаграма класів моделей

4.2. Реалізація застосунку

4.2.1. Реалізація парсеру proto файлів

Одна з переваг системи полягає у тому, що для синтаксису proto файлів використовується спеціальна мова, граматика якої описана за допомогою ANTLR,

що дозволяє більш щільно описувати усі необхідні компоненти. Парсер proto файлів отримує на вході рядок, у якому описані усі компоненти, та повертає FileParеResult модель. У таблиці 4.1 описані класи, які відповідають за парсинг.

Таблиця 4.1 – опис класів, відповідальних за парсинг

Номер	**Назва**	**Опис**
1	GrammarParser	Клас, згенерований ANLR, який відповідає за будування дерева токенів, за яким будується модель.
2	MainVisitor	Відповідає за будування моделі на основі дерева лекскем.
3	MainParser	Інкапсулює у собі будування дерева токенів та будування моделі. Також це інтерфейс, який використовують інші частини програми.

Граматика. У граматиці наявні наступні основні лексеми:

- інструкція – список моделей та/або контролерів;
- контролер – містить ім'я контролера та список методів;
- модель – містить список полів;
- метод – містить ім'я методу, дієслово (Get, Post …), параметри, повертаємий тип та може містити шлях до методу;
- поле – містить тип та назву.

Також означені ключові слова:

- model – для означення моделі;
- cntrl – для означення контролера;
- return – для вказання повертаємого типу;
- fromBody, fromQuery – для означення різних типів параметрів.

Підсумовуючи, існує дві основні моделі найвищого рівня – контролер та модель. Контролер містить інформацію про методи, описані у контролері. Модель має список полів на їх типів. Ім'я, параметри та повертаємий тип – основні характеристики методу. Існує два типи параметрів – передаваємий у тілі запиту, або у рядку запиту.

Повний опис граматики наявний у додатку А.

MainVisitor . У класі MainVisitor наявні наступні методи:

- VisitInstructions – основний метод, який спрацьовує першим при обході дерева. Повертає FileParseResult модель.
- VisitController – метод який викликається під час обходу контролеру. Повертає ControllerModel.
- VisitMethodDefinition – метод який викликається під час обходу кожного методу з контролеру. Повертає MethodModel.
- VisitModel – викликається під час обходу моделі. Повертає EntityModel.
- VisitPropertyDefinition – викликається під час обходу кожного поля з моделі. Повертає FieldModel.
- VisiParameters – викликається під час обходу параметрів методу. Повертає ParameterModel.
- VisitQueryParameter – викликається під час обходу рядковий параметрів. Повертає List<FieldModel>.
- VisitBodyParameter – викликається під час обходу параметра тіла. Повертає FieldModel.
- ShouldBeUnique – приватний метод, який використовується для перехоплення помилки створення об'єктів з однаковими назвами, коли це недопустимо.

Повний опис класу MainVisitor наявний у додатку Б.

MainParser. У класі MainParser наявний лише один метод ParseProtoFile, який відповідає за ініціалізацію парсингу, побудову дерева та використання класу MainVisitor для обходу усіх лексем.

Повний опис класу MainParser наявний у додатку Б.

4.2.2. Реалізація генераторів

Генератори використовуються для управління, також перевіряють валідність вхідних параметрів. В них інкапсулювана генерація коду за допомогою записувачів. У таблиці 4.2 наведений список класів, до яких відносяться генератори.

Таблиця 4.2 - опис класів-генераторів

Номер	**Назва**	**Опис**
1	BaseGenerator	Базовий клас для усіх генераторів. Відповідає за перевірку вхідних параметрів, та побудову моделі для генерації.
2	BackEndGenerator	Нащадок базового класу, відповідає за генерацію моделей та контролерів, використовуючи відповідні записувачі.
3	CSharpClientGenerator	Нащадок базового класу, відповідає за генерацію моделей та клієнту, використовуючи відповідні записувачі.

BaseGenerator. Базовий клас для усіх генераторів. У класі BaseGenerator наявні наступні методи:

- ValidateInputParameters – приватний метод, перевіряє валідність вхідних параметрів, зокрема існування proto файлу за вказаним шляхом та перевірку розширення файлу. Також створює вихідну директорію якщо її немає.
- Execute – метод, який використовується для початку генерування файлів, від парсингу та валідації до безпосередньо генерації.
- ParseFiles – використовуючи MainParser парсить вхідний файл для створює FileParseResult об'єкт.
- ValidateModelNames – метод, який використовується задля перевірки на існування моделей, вказаних у proto файлі.
- Generate – абстрактний метод, який реалізують похідні класи. Викликається у методі Execute.

Повний опис класу BaseGenerator наявний у додатку В.

BackEndGenerator. Похідний клас від BaseGenerator. У класі BackEndGenerator реалізований метод Execute, у якому відбувається генерація моделей та контролерів. Для генерації використовує об'єкти класів ControllerWriter та ModelWriter.

Повний опис класу BackEndGenerator наявний у додатку В.

CSharpClientGenerator. Похідний клас від BaseGenerator. У класі реалізований метод Execute, у якому відбувається генерація моделей та клієнту. Для генерації використовує об'єкти класів CsClientWriter та ModelWriter.

Повний опис класу CSharpClientGenerator наявний у додатку В.

4.2.3. Реалізація записувачів

Записувачі використовуються безпосередньо для генерації коду. В них інкапсульовані шаблони, за якими генерується код. Записувачі генерують код на

основі вхідних моделей за приготованими шаблонами. У таблиці 4.3 наведений список класів-записувачів.

Таблиця 4.3 – Опис класів-записувачів

Номер	Назва	Опис
1	BaseWriter	Базовий клас для усіх записувачів. Також відповідає за створення файлів у файловій системі.
2	ControllerWriter	Записувач, який відповідає за генерацію коду контролера.
3	ModelWriter	Записувач, який відповідає за генерацію коду моделі.
4	CsClientWriter	Записувач, який відповідає за генерацію коду клієнта.

BaseWriter. Базовий клас для усіх записувачів. Наявні наступні методи:

- AddSource – статичний метод для створення файлів.
- GenerateContent – абстрактний метод, який реалізуються похідні класи.
- Tabs – статичний метод, спільний для усіх записувачів. Потрібен для створення відступну довільної довжини.

Повний опис класу BaseWriter наявний у додатку Г.

ControllerWriter. Похідний клас від BaseWriter, який реалізує метод GenerateContent, який відповідає за генерацію коду контролеру, використовуючи об'єкт StringBuilder.

Повний опис класу ControllerWriter наявний у додатку Г.

ModelWriter. Похідний клас від BaseWriter, який реалізує метод GenerateContent, який відповідає за генерацію коду моделей, використовуючи об'єкт StringBuilder.

Повний опис класу ModelWriter наявний у додатку Г.

CsClientWriter. Похідний клас від BaseWriter, який реалізує метод GenerateContent, який відповідає за генерацію коду клієнта, використовуючи об'єкт StringBuilder.

Повний опис класу CsClientWriter наявний у додатку Г.

4.2.4 Реалізація моделей

Моделі використовуються для транспортування даних всередині додатку. Вхідний текст парситься у ці моделі, також генерація коду відбувається на основі цих моделей. Опис моделей наведений у таблиці 4.4.

Таблиця 4.4 – опис моделей

Номер	Назва	Опис
1	FileParseResult	Основна модель, яка містить у собі список моделей контролерів, та список моделей сутностей. Результатом парсингу є саме ця модель
2	EntityModel	Модель, яка описує сутність. Має такі поля, як ім'я, та список пар: тип та назва.
3	ControllerModel	Модель, яка описує контролер. Має ім'я та список методів.
4	MethodModel	Модель, описує метод. Має наступні поля: тип – http verb (get, post, put …); ім'я; шлях – url методу; повертаємий тип; параметри – параметри методу

5	ParameterModel	Модель описує параметри методу. Має поле для body параметру, та поле для параметрів запиту

4.3 Взаємодія компонент

У proto файлі наявні дві базові інструкції – моделі та контролери. На базі proto файлу відбувається створення моделі FileParseResult, у якої два основні поля – також контролери та моделі.

Розглянемо контролери. У proto файлі контролер позначається наступним чином:

```
cntrl <ім'я контролера>{
     [“<http шлях методу>”] <тип методу> <ім'я методу>(<параметри методу >) return <повертаємий тип>;
}
```

, де cntrl – зарезервоване слово, після нього йде ім'я контролера. Потім у тілі контролера означений список методів. Для кожного метода є шлях до нього, тип методу (get, post, put, delete), ім'я, параметри, та повертаємий тип. У результаті парсингу отримуємо модель контролеру ControllerModel, у якій є ім'я та список методів. У моделі методу є всі ті поля, які описані у proto файлі. Після оброки proto файлу та отримання результатів у вигляді моделей та їх валідації відбувається генерація коду. Під час генерації коду у першу чергу записуються усі необхідні using оператори. Після цього генерується тіло простіру, атрибути ApiController та Route для контролеру, і після цього генерується тіло контролеру. Тобто генерується абстрактний клас, який унаслідується від ControllerBase та має список абстрактних методів. В залежності від типу методу генерується відповідний атрибут (HttpGet, HttpPost…), ім'я, повертаємий тип є Task<ActionResult<тип> >, типізований типом з моделі та вхідні body або query параметри.

Також но основі ControllerModel генеруться клієнт серверу. В першу чергу також додається усі необхідні using операції, після цього генерується простіру і в ньому генерується клас клієнта. Через конструктор ініціалізується об'єкт HttpClient та url серверу. Для кожного методу контролера генерується окремий метод клієнта, з таким самим повертаємим типом та вхідними параметрами. В середині кожного методу в першу чергу генерується рядок запиту, а потім викликається один з чотирьох приватних методів, кожен з яких відповідає за окремий вид запиту (get, post, put, delete). Після виклику приватного методу повертається результат.

Модель у proto файлі позначається так:

```
model <ім'я моделі>{

      <Тип поля> <ім'я поля>;

}
```

, де model – зарезервоване слово, далі ім'я моделі. У тілі моделі записані поля моделі, одне поле це тип та ім'я. У результаті парсингу отримуємо модель EntityModel, у якій є ім'я та список полів. Після парсингу починається процес генерації коду. Під час генерації коду у першу чергу записуються усі необхідні using оператори. Після цього генерується тіло простіру, та клас моделі, з полями, зазначеними у proto файлі.

4.4 Життєвий цикл програми

Життєвий цикл програми складається з декількох частин:

1. У першу чергу відбувається парсинг proto файлу за допомогою засобів ANTLR, будується дерево лексем.
2. Наступним кроком у класі MainVisitor відвідується кожна лексема дерева та на їх основі будується модель FileParseResult, у якій міститься інформація про моделі та їх поля, і про контролери та їх методи.

3. Потім, в залежності від вхідних параметрів програми, відбувається генерація контенту файлу, тобто генерується код або серверу та моделей, або клієнту та моделей.
4. Після цього вже генеруються файли коду у файловій системі. Завершення роботи програми.

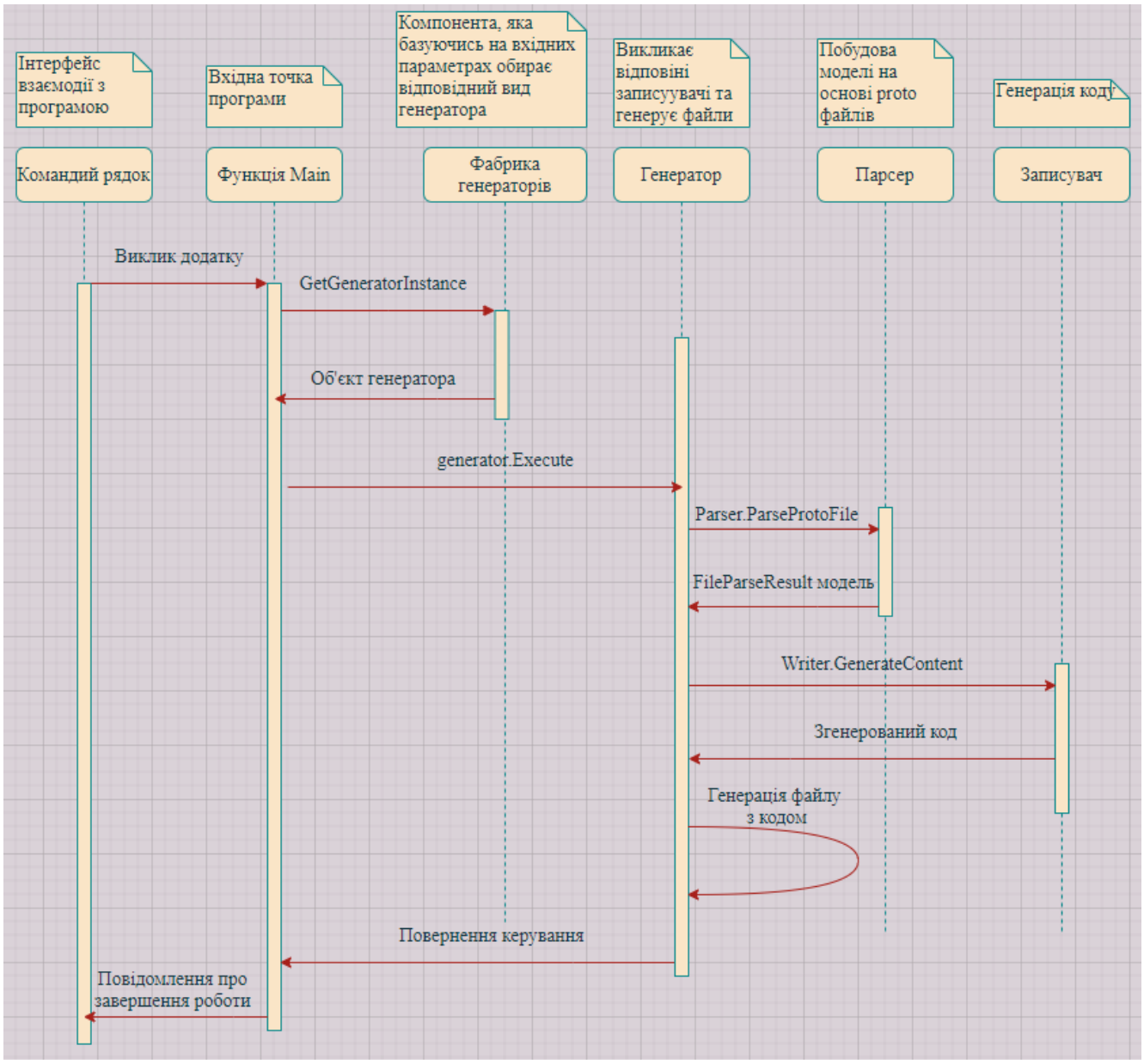


Рисунок 4.4 – діаграма життєвого циклу додатку

РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА

5.1. Синтаксис Proto файлів

Два основні елемента, які використовуються при написанні Proto файлів – це контролери та моделі. Весь Proto файл складається з цих блоків: моделей або контролерів.

Синтаксис блоку контролера:

```
cntrl <ім'я контролера>{
    [“<http шлях методу>”] <тип методу> <ім'я методу>(<параметри методу >) return <повертаємий тип>;
}
```

Блок контролерів містить список методів. Підтримуються наступні типи методів: get, post, put, delete. Також параметри методу мають мати існуючі типи, тобто або примітивні типи (int, string та інш.), або типи, які описані у моделях.

Синтаксис блоку моделі:

```
model <ім'я моделі>{
    <Тип поля> <ім'я поля>;
}
```

Блок моделі містить список полів. Типами полів мають бути або примітивні типи, або типи, описані у моделях.

5.2 Генерація файлів коду на основі Proto файлу

Після написання Proto файлу можна приступити до генерацій файлів коду.

Для початку вам потрібно завантажити цей додаток. Потім встановити у будь-яку зручну для вас директорію. Після цього потрібну запустити командну стрічку у цій директорії

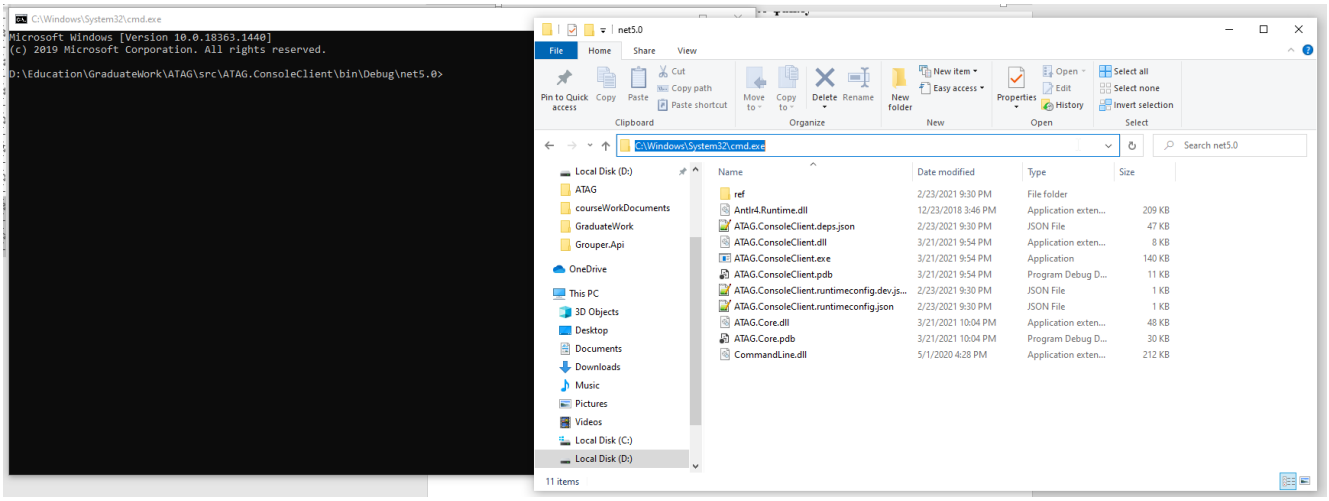


Рисунок 5.1 – запуск командної стрічки

Тепер ви можете запустити додаток. Додаток має наступні параметри:

Таблиця 5.1 – список параметрів

Номер	Коротка форма	Довга форма	Опис
1	-s	--source	Шлях до Proto файлу
2	-d	--dest	Шлях до директорії, у якій згенеруються файли
3	-t	--type	Тип файлів коду, які будуть згенеровані. Це може бути BackEnd – код серверу, та CSharpClient – код клієнту
4	-n	--namespace	Необов'язковий параметр. Вказує простір, у якому будуть згенеровані класи. Значення по замовчуванню : AtagGenerated

Також присутній параметр –help, який покаже всі доступні параметри та інформацію про них.

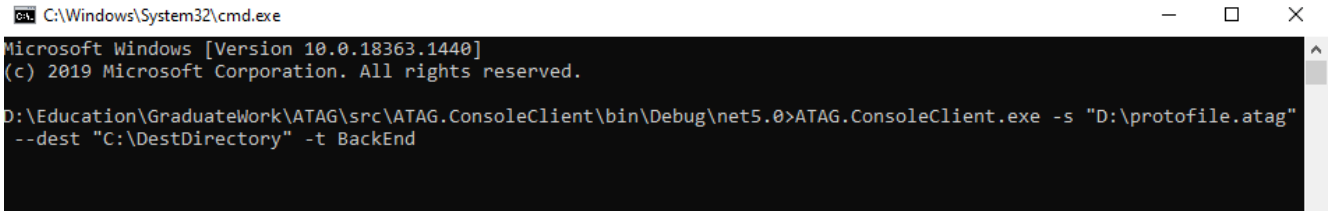


Рисунок 5.2 – приклад запуску додатку

У цьому прикладі ми отримуємо Proto файл за шляхом "D:\protofile.atag" та генеруємо файли BackEnd серверу у директорію "C:\DestDirectory".

5.3 Приклад роботи

Створимо Proto файл з наступним контентом:

```
cntrl ControllerName1{

      ["test_fdfted_sf"] get MethodName1() return string;
      post MethodName2(fromBody: string StrParameter; fromQuery: int qpInt,
string qpStr) return Model1;
      ["test-route{1}/wer"] put MethodName3(fromBody: Model1 m1) return int;
}

model Model1{
    string Str1;
    int Int1;
}

model Test_Model{
    int m2int2;
    int m2int3;
}
```

Та збережемо його директорії зі шляхом «C:\Users\Evgentus\Desktop\Test» та назвою «ProtoTest.atag». Після цього у директорії з додатком відкриємо командний рядок та запустимо наступну команду:

```
ATAG.ConsoleClient.exe -s "C:\Users\Evgentus\Desktop\Test\ProtoTest.atag" -d "C:\Users\Evgentus\Desktop\Test\backend" -t BackEnd
```

Після виконання команди у директорії за шляхом `"C:\Users\Evgentus\Desktop\Test\backend"` будуть згенеровані наступні файли:

Test_Model.cs (рис 5.3):

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Runtime;
5   namespace AtagGenerated
6   {
7       public class Test_Model
8       {
9           public int m2int2 { get; set; }
10          public int m2int3 { get; set; }
11      }
12  }
```

Рисунок 5.3 – згенерований файл

ControllerName1Controller.cs (5.4):

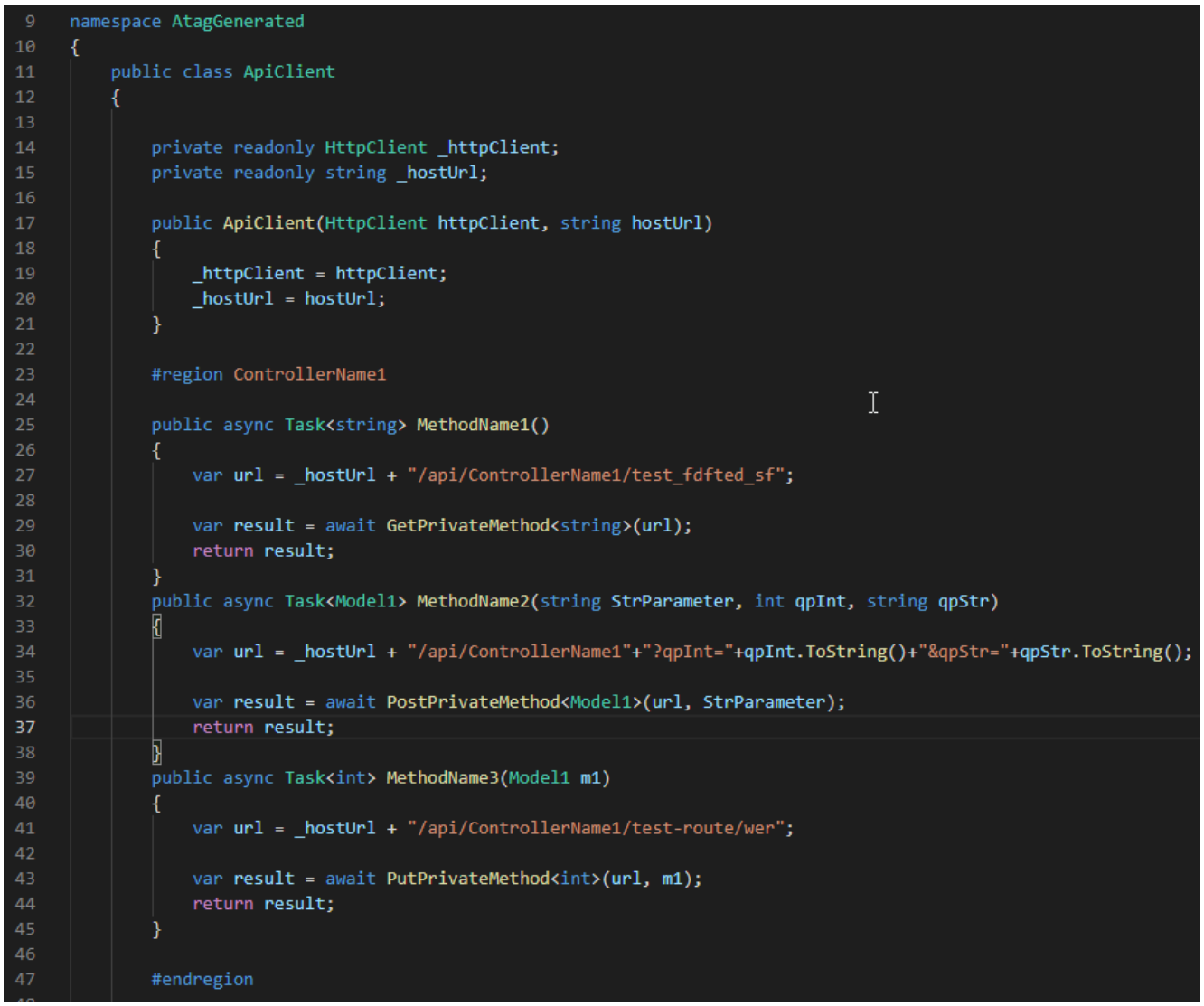
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using System.Runtime;
namespace AtagGenerated
{
    [Route("api/[controller]")]
    [ApiController]
    public abstract class ControllerName1Controller : ControllerBase
    {
        [HttpGet]
        [Route("test_fdfted_sf")]
        public abstract Task<ActionResult<string>> MethodName1();

        [HttpPost]
        public abstract Task<ActionResult<Model1>> MethodName2([FromBody]string StrParameter, [FromQuery]int qpInt, [FromQuery]string qpStr);

        [HttpPut]
        [Route("test-route/wer")]
        public abstract Task<ActionResult<int>> MethodName3([FromBody]Model1 m1);

    }
}
```

Рисунок 5.4 – згенерований файл

Також згенеруємо код для клієнта. Для цього у командному рядку виконаємо команду: `ATAG.ConsoleClient.exe -s "C:\Users\Evgentus\Desktop\Test\ProtoTest.atag" -d "C:\Users\Evgentus\Desktop\Test\client" -t CSharpClient`

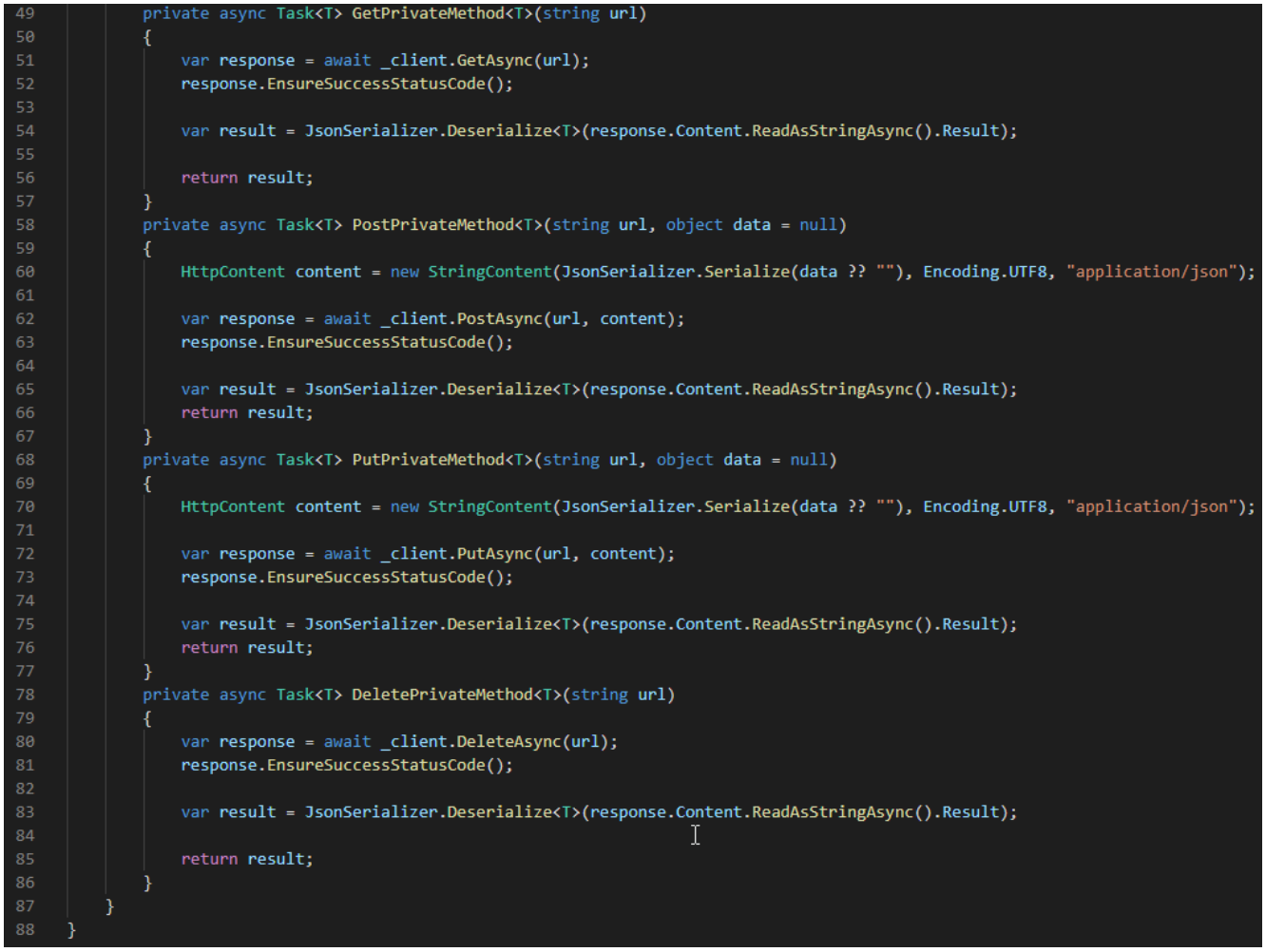
Виконання команди згенерує код для таких саме моделей, як і при генерації контролера, а також код клієнту:

ApiClient.cs (рис 5.5, рис 5.6):



```
namespace AtagGenerated
{
    public class ApiClient
    {

        private readonly HttpClient _httpClient;
        private readonly string _hostUrl;

        public ApiClient(HttpClient httpClient, string hostUrl)
        {
            _httpClient = httpClient;
            _hostUrl = hostUrl;
        }

        #region ControllerName1

        public async Task<string> MethodName1()
        {
            var url = _hostUrl + "/api/ControllerName1/test_fdfted_sf";

            var result = await GetPrivateMethod<string>(url);
            return result;
        }
        public async Task<Model1> MethodName2(string StrParameter, int qpInt, string qpStr)
        {
            var url = _hostUrl + "/api/ControllerName1"+"?qpInt="+qpInt.ToString()+"&qpStr="+qpStr.ToString();

            var result = await PostPrivateMethod<Model1>(url, StrParameter);
            return result;
        }
        public async Task<int> MethodName3(Model1 m1)
        {
            var url = _hostUrl + "/api/ControllerName1/test-route/wer";

            var result = await PutPrivateMethod<int>(url, m1);
            return result;
        }

        #endregion
```

Рисунок 5.5 – згенерований файл



```
49          private async Task<T> GetPrivateMethod<T>(string url)
50          {
51              var response = await _client.GetAsync(url);
52              response.EnsureSuccessStatusCode();
53
54              var result = JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
55
56              return result;
57          }
58          private async Task<T> PostPrivateMethod<T>(string url, object data = null)
59          {
60              HttpContent content = new StringContent(JsonSerializer.Serialize(data ?? ""), Encoding.UTF8, "application/json");
61
62              var response = await _client.PostAsync(url, content);
63              response.EnsureSuccessStatusCode();
64
65              var result = JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
66              return result;
67          }
68          private async Task<T> PutPrivateMethod<T>(string url, object data = null)
69          {
70              HttpContent content = new StringContent(JsonSerializer.Serialize(data ?? ""), Encoding.UTF8, "application/json");
71
72              var response = await _client.PutAsync(url, content);
73              response.EnsureSuccessStatusCode();
74
75              var result = JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
76              return result;
77          }
78          private async Task<T> DeletePrivateMethod<T>(string url)
79          {
80              var response = await _client.DeleteAsync(url);
81              response.EnsureSuccessStatusCode();
82
83              var result = JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
84
85              return result;
86          }
87      }
88  }
```

Рисунок 5.6 – згенерований файл

Після виконання роботи програми маємо готовий код для клієнту Api, код моделей та абстрактний клас контролеру.

ВИСНОВКИ

Під час написання додатку були досліджені методи парсингу файлів, генерація файлів. Також був досліджений ринок подібних технологій. Були проаналізовані потреби розробників при написанні додатків виду «сервер-клієнт». Використання даного додатку зможе значно прискорити розробку серверних та клієнтських застосунків. Також завдяки слабо зв'язній архітектурі додаток може розширюватися без змін існуючого коду.

Кінцевий продукт сповна відповідає вимогами які були висунуті системі.

При подальшій розробці проекту може бути реалізована наступна функціональність: створення клієнтів Api для різних мов або фреймворків (Java, JavaScript, Angular), розширення та кастомізація генерації коду, додавання додаткових налаштувань при генерації, обробка схеми бази даних та створення відповідних запитів за допомогою Entity Framework, тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ОС Windows 10 [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Windows_10.
2. ANTLR [Електронний ресурс] – Режим доступу до ресурсу: https://www.antlr.org/
3. .Net Framework [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/.NET_Framework
4. .Net Core [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/.NET_Core
5. Консольні додатки [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Console_application
6. Юніт тестування NUnit [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/NUnit
7. Git [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Git
8. SourceTree [Електронний ресурс] – Режим доступу до ресурсу: https://www.sourcetreeapp.com/
9. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/GitHub
10. GitFlow [Електронний ресурс] – Режим доступу до ресурсу: https://bitworks.software/2019-03-12-gitflow-workflow.html
11. REST [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/REST
12. gRPC [Електроний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/GRPC

ДОДАТОК

Код додатку можна знайти за посиланням: https://github.com/Evgentus0/ATAG

Для піднімання проекту локально потрібна Visual Studio 2019 та .NET5 Sdk, яку можна встановити за посиланням: https://visualstudio.microsoft.com/downloads/

ДОДАТОК А Опис граматики

```
grammar Grammar;

/*
 * Parser Rules
 */

instructions : (controller | model)+;

controller : CONTROLLER name=CAPTION LBRACE (methodDefinition';')+ RBRACE;

model : MODEL name=CAPTION LBRACE (propertyDefenition';')+ RBRACE;

methodDefinition : (LSQUAREPAREN route=VALUE RSQUAREPAREN)? verb=VERB
name=CAPTION LPAREN (parameters)? RPAREN RETURN returnedType=CAPTION;

parameters : (bodyParameter | queryParameter | bodyParameter ';'
queryParameter);
bodyParameter : FROMBODY ':' propertyDefenition;
queryParameter : FROMQUERY ':' propertyDefenition (',' propertyDefenition);

propertyDefenition: type=CAPTION name=CAPTION;

/*
 * Lexer Rules
 */

MODEL : 'model';

CONTROLLER : 'cntrl';

RETURN : 'return';

FROMBODY : 'fromBody';
FROMQUERY : 'fromQuery';

VERB : 'get' | 'post' | 'put' | 'delete';
```

```
LBRACE : '{';
RBRACE : '}';

LPAREN : '(';
RPAREN : ')';

LSQUAREPAREN : '[';
RSQUAREPAREN : ']';

EQUAL : '=';

CAPTION : [A-Za-z]+([A-Za-z] | [0-9] | '_')+;

VALUE : '"' ([A-Za-z] | [0-9] | '-' | '/' | '_' | LBRACE | RBRACE)+ '"';

WS : [ \t\r\n] -> channel(HIDDEN);
```

ДОДАТОК Б Парсер proto файлів

MainVisitor

```
using Antlr4.Runtime.Misc;
using Antlr4.Runtime.Tree;
using ATAG.Core.Exceptions;
using ATAG.Core.Models;
using ATAG.Core.Models.Enums;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATAG.Core.Visitors
{
    public class MainVisitor : GrammarBaseVisitor<object>
    {
        public override object VisitInstructions([NotNull] GrammarParser.InstructionsContext
context)
        {
            var result = new FileParseResult();

            if (context == null)
                return result;

            if (context.exception is not null)
                throw context.exception;

            try
            {
                foreach (var child in context.children)
                {
                    var instructionType = child.GetChild(0).GetText();

                    switch (instructionType)
                    {
                        case "cntrl":
                            var controller = (ControllerModel)Visit(child);

                            ShouldBeUnique("Controller", controller.Name,
                                result.Controllers.Select(x => x.Name), child);
```

```
                            result.Controllers.Add(controller);
                            break;

                        case "model":
                            var model = (EntityModel)Visit(child);

                            ShouldBeUnique("Model", model.Name,
                                result.Models.Select(x => x.Name), child);

                            result.Models.Add(model);
                            break;

                        default:
                            throw new GrammarException($"Can not parse isntruction
{instructionType}")
                            {
                                FromLine = child.SourceInterval.a,
                                ToLine = child.SourceInterval.b
                            };
                    }
                }

                return result;
            }
            catch
            {
                throw;
            }
        }

        public override object VisitController([NotNull] GrammarParser.ControllerContext
context)
        {
            var controller = new ControllerModel();

            if (context == null)
                return controller;

            if (context.exception is not null)
                throw context.exception;

            controller.Name = context.name.Text;

            try
            {
                foreach (var method in context.methodDefinition())
                {
                    var methodModel = (MethodModel)Visit(method);

                    ShouldBeUnique("Method", methodModel, controller.Methods, method);

                    controller.Methods.Add(methodModel);
                }

                return controller;
            }
            catch
            {
                throw;
            }
        }

        public override object VisitMethodDefinition([NotNull]
GrammarParser.MethodDefinitionContext context)
        {
            var methodModel = new MethodModel();

            if (context == null)
                return methodModel;
```

```
            if (context.exception is not null)
                throw context.exception;

            methodModel.Verb = (HttpVerb)Enum.Parse(typeof(HttpVerb), context.verb.Text);
            methodModel.Name = context.name.Text;
            methodModel.ReturnedType = context.returnedType.Text;
            methodModel.Route = context.route?.Text ?? string.Empty;

            if (!string.IsNullOrEmpty(methodModel.Route))
            {
                methodModel.Route = methodModel.Route.Trim(new char[]{' ', '"'});
            }

            methodModel.Parameters = context.parameters() != null ?
                (ParameterModel)Visit(context.parameters()) : new ParameterModel();


            return methodModel;
        }

        public override object VisitModel([NotNull] GrammarParser.ModelContext context)
        {
            var model = new EntityModel();

            if (context == null)
                return model;

            if (context.exception is not null)
                throw context.exception;

            model.Name = context.name.Text;

            foreach(var prop in context.propertyDefenition())
            {
                var property = (FieldModel)Visit(prop);

                ShouldBeUnique("Property", property,
                    model.Properties, prop);

                model.Properties.Add(new FieldModel { Type = property.Type, Name =
property.Name });
            }


            return model;
        }

        public override object VisitPropertyDefenition([NotNull]
GrammarParser.PropertyDefenitionContext context)
        {
            if (context == null)
                return new FieldModel();

            if (context.exception is not null)
                throw context.exception;

            return new FieldModel { Type = context.type.Text, Name = context.name.Text };
        }

        public override object VisitParameters([NotNull] GrammarParser.ParametersContext
context)
        {
            var parameters = new ParameterModel();
            if (context == null)
                return parameters;

            if (context.exception is not null)
                throw context.exception;
```

```
            parameters.BodyParameter = context.bodyParameter() != null ?
                (FieldModel)Visit(context.bodyParameter()) : new FieldModel();
            parameters.QueryParameters = context.queryParameter() != null ?
                (List<FieldModel>)Visit(context.queryParameter()) : new List<FieldModel>();

            return parameters;
        }

        public override object VisitQueryParameter([NotNull]
GrammarParser.QueryParameterContext context)
        {
            var parameters = new List<FieldModel>();
            if (context == null)
                return parameters;

            if (context.exception is not null)
                throw context.exception;

            foreach (var prop in context.propertyDefenition())
            {
                var property = (FieldModel)Visit(prop);

                ShouldBeUnique("QueryParameter", property.Name,
                    parameters.Select(x => x.Name), prop);

                parameters.Add(new FieldModel{ Type = property.Type, Name = property.Name});
            }

            return parameters;
        }

        public override object VisitBodyParameter([NotNull]
GrammarParser.BodyParameterContext context)
        {
            if (context == null)
                return new FieldModel();

            if (context.exception is not null)
                throw context.exception;

            var keyValue = (FieldModel)Visit(context.propertyDefenition());
            return keyValue;
        }

        private void ShouldBeUnique<T>(string memberName, T suspect,
            IEnumerable<T> collection, IParseTree currentTree)
        {
            if(collection.Contains(suspect))
                throw new GrammarException($"{memberName} with name \"{suspect}\" is already
exist!")
                {
                    FromLine = currentTree.SourceInterval.a,
                    ToLine = currentTree.SourceInterval.b
                };
        }
    }
}
```

MainParser

```
using Antlr4.Runtime;
using ATAG.Core.Interfaces;
using ATAG.Core.Models;
using ATAG.Core.Visitors;

namespace ATAG.Core
{
```

```
    public class MainParser : IMainParser
    {
        private MainVisitor _mainVisitor;

        public MainParser(MainVisitor mainVisitor)
        {
            _mainVisitor = mainVisitor;
        }

        public FileParseResult ParseProtoFile(string content)
        {
            var lexer = new GrammarLexer(new AntlrInputStream(content));
            lexer.RemoveErrorListeners();
            var tokens = new CommonTokenStream(lexer);
            var parser = new GrammarParser(tokens);

            var tree = parser.instructions();
            var result = _mainVisitor.Visit(tree);

            return (FileParseResult)result;
        }
    }
}
```

ДОДАТОК В Реалізація генераторів

BaseGenerator

```
using ATAG.Core;
using ATAG.Core.Extentions;
using ATAG.Core.Interfaces;
using ATAG.Core.Models;
using ATAG.Core.Models.Inbound;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATAG.Core.Generators
{
    public abstract class BaseGenerator
    {
        private const string _extention = ".atag";
        private readonly string[] _supportedTypes = { "int", "long", "string", "double",
"decimal", "float" };
        private readonly IMainParser _mainParser;

        protected readonly InputParametersModel _parameters;

        public BaseGenerator(IMainParser mainParser, InputParametersModel parameters)
        {
            _mainParser = mainParser;
            _parameters = parameters;

            ValidateInputParameters();
        }

        private void ValidateInputParameters()
        {
            if(!File.Exists(_parameters.SourceFilePath))
                throw new ArgumentException($"File does not exists:
{_parameters.SourceFilePath}");
```

```
            var sourceFile = _parameters.SourceFilePath;
            var extention = Path.GetExtension(sourceFile);
            if (extention != _extention)
                throw new ArgumentException($"Incorrect extention: {extention}");

            if (!Directory.Exists(_parameters.DestinationPath))
                Directory.CreateDirectory(_parameters.DestinationPath);
        }

        public void Execute()
        {
            FileParseResult parseResult = ParseFiles();

            var validationResult = ValidateModelNames(parseResult);

            if (validationResult.isSuccess)
            {
                Generate(parseResult);
            }
            else
            {
                throw new Exception(validationResult.message);
            }
        }

        protected virtual FileParseResult ParseFiles()
        {
            try
            {
                var                                    result                                    =
_mainParser.ParseProtoFile(File.ReadAllText(_parameters.SourceFilePath));

                return result;
            }
            catch(Exception ex)
            {
                throw new Exception("Error during parsing proto file", ex);
            }
        }

        protected virtual (bool isSuccess, string message) ValidateModelNames(FileParseResult
parseResult)
        {
            var supportedTypes = new List<string>(_supportedTypes);
            var customTypes = parseResult.Models.Select(x => x.Name);
            supportedTypes.AddRange(customTypes);

            var usedTypes = new List<string>();

            foreach (var model in parseResult.Models)
            {
                var properties = model.Properties.Select(x => x.Type);

                usedTypes.AddRange(properties);
            }

            foreach(var controller in parseResult.Controllers)
            {
                foreach(var method in controller.Methods)
                {
                    if(method.ReturnedType != null)
                        usedTypes.Add(method.ReturnedType);

                    if(method.Parameters.BodyParameter.Type != null)
                        usedTypes.Add(method.Parameters.BodyParameter.Type);

                    if(!method.Parameters.QueryParameters.Select(x                          =>
x.Type).IsNullOrEmpty())
```

```
                        usedTypes.AddRange(method.Parameters.QueryParameters.Select(x     =>
x.Type));
                }
            }

            foreach(var type in usedTypes)
            {
                if (!supportedTypes.Contains(type))
                {
                    return (false, $"Incorrect type: {type}!");
                }
            }

            return (true, string.Empty);
        }

        protected abstract void Generate(FileParseResult parseResult);

    }
}
```

BackEndGenerator

```
using ATAG.Core.Generators.Writers;
using ATAG.Core.Interfaces;
using ATAG.Core.Models;
using ATAG.Core.Models.Inbound;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace ATAG.Core.Generators
{
    public class BackEndGenerator : BaseGenerator
    {
        private readonly BaseWriter _controllerWriter;
        private readonly BaseWriter _modelWriter;

        public   BackEndGenerator(IMainParser   mainParser,   InputParametersModel   input,
BaseWriter controllerWriter, BaseWriter modelWriter)
            :base(mainParser, input)
        {
            _controllerWriter = controllerWriter;
            _modelWriter = modelWriter;
        }

        protected override void Generate(FileParseResult parseResult)
        {
            foreach (var controller in parseResult.Controllers)
            {
                if (!controller.Name.EndsWith("Controller"))
                    controller.Name += "Controller";

                string content = _controllerWriter.GenerateContent(controller);

                string      fullPath      =      Path.Combine(_parameters.DestinationPath,
$"{controller.Name}.cs");

                BaseWriter.AddSource(fullPath, content);
            }

            foreach (var model in parseResult.Models)
            {
                if (!model.Name.EndsWith("Model"))
                    model.Name += "Model";
```

```
                string content = _modelWriter.GenerateContent(model);

                string      fullPath      =      Path.Combine(_parameters.DestinationPath,
$"{model.Name}.cs");

                BaseWriter.AddSource(fullPath, content);
            }
        }
    }
}
```

CSharpClientGenerator

```
using ATAG.Core.Generators.Writers;
using ATAG.Core.Interfaces;
using ATAG.Core.Models;
using ATAG.Core.Models.Inbound;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATAG.Core.Generators
{
    public class CSharpClientGenerator : BaseGenerator
    {
        private readonly BaseWriter _csClientWriter;
        private readonly BaseWriter _modelWriter;

        public  CSharpClientGenerator(IMainParser  mainParser,  InputParametersModel  input,
BaseWriter csClientWriter, BaseWriter modelWriter)
            :base(mainParser, input)
        {
            _csClientWriter = csClientWriter;
            _modelWriter = modelWriter;
        }
        protected override void Generate(FileParseResult parseResult)
        {
            {
                string content = _csClientWriter.GenerateContent(parseResult.Controllers);
                string fullPath = Path.Combine(_parameters.DestinationPath, "ApiClient.cs");
                BaseWriter.AddSource(fullPath, content);
            }

            foreach (var model in parseResult.Models)
            {
                if (!model.Name.EndsWith("Model"))
                    model.Name += "Model";

                string content = _modelWriter.GenerateContent(model);

                string      fullPath      =      Path.Combine(_parameters.DestinationPath,
$"{model.Name}.cs");

                BaseWriter.AddSource(fullPath, content);
            }
        }
    }
}
```

ДОДАТОК Г Реалізація записувачів

BaseWriter

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace ATAG.Core.Generators.Writers
{
    public abstract class BaseWriter
    {
        protected readonly string _namespace;

        public BaseWriter(string @namespace)
        {
            _namespace = @namespace;
        }

        public static void AddSource(string fullPath, string content)
        {
            using var writer = File.CreateText(fullPath);
            writer.Write(content);
        }

        public abstract string GenerateContent(object entity);

        protected static string Tabs(int n)
        {
            return new string('\t', n);
        }
    }
}
```

ControllerWriter

```
using ATAG.Core.Extentions;
using ATAG.Core.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ATAG.Core.Generators.Writers
{
    public class ControllerWriter : BaseWriter
    {
        public ControllerWriter(string @namespace): base(@namespace)
        { }

        public override string GenerateContent(object entity)
        {
            var controller = (ControllerModel)entity;

            StringBuilder sb = new StringBuilder();
            int tabLevel = 0;

            sb.Append("using System;");
            sb.AppendLine();
            sb.Append("using System.Collections.Generic;");
            sb.AppendLine();
            sb.Append("using System.Linq;");
```

```
            sb.AppendLine();
            sb.Append("using System.Threading.Tasks;");
            sb.AppendLine();
            sb.Append("using Microsoft.AspNetCore.Mvc;");
            sb.AppendLine();
            sb.Append("using Microsoft.AspNetCore.Http;");
            sb.AppendLine();
            sb.Append("using System.Runtime;");
            sb.AppendLine();

            sb.Append($"namespace {_namespace}");
            sb.AppendLine();
            sb.Append("{");
            sb.AppendLine();

            tabLevel++;

            sb.Append($"{Tabs(tabLevel)}[Route(\"api/[controller]\")]");
            sb.AppendLine();
            sb.Append($"{Tabs(tabLevel)}[ApiController]");
            sb.AppendLine();
            sb.Append($"{Tabs(tabLevel)}public    abstract    class    {controller.Name}    :
ControllerBase");
            sb.AppendLine();
            sb.Append($"{Tabs(tabLevel)}{{");
            sb.AppendLine();

            foreach (var method in controller.Methods)
            {
                tabLevel++;

                sb.Append($"{Tabs(tabLevel)}[{method.Verb.GetDescription()}]");
                sb.AppendLine();
                if (!string.IsNullOrEmpty(method.Route))
                {
                    sb.Append($"{Tabs(tabLevel)}[Route(\"{method.Route}\")]");
                    sb.AppendLine();
                }

                sb.Append($"{Tabs(tabLevel)}public                                    abstract
Task<ActionResult<{method.ReturnedType}>> " +
                    $"{method.Name}(");

                bool                            hasBodyParameter                                =
!string.IsNullOrEmpty(method.Parameters.BodyParameter.Type);
                if (hasBodyParameter)
                {
                    var bp = method.Parameters.BodyParameter;
                    sb.Append($"[FromBody]{bp.Type} {bp.Name}");
                }

                if (method.Parameters.QueryParameters.Count > 0)
                {
                    if (hasBodyParameter)
                        sb.Append(", ");

                    foreach (var qp in method.Parameters.QueryParameters)
                    {
                        sb.Append($"[FromQuery]{qp.Type} {qp.Name}, ");
                    }
                    sb.Length -= 2;
                }
                sb.Append(");");

                sb.AppendLine();
                sb.AppendLine();

                tabLevel--;
            }
```

```
            sb.Append($"{Tabs(tabLevel)}}}");
            sb.AppendLine();
            sb.Append("}");

            return sb.ToString();
        }
    }
}
```

ModelWriter

```
using ATAG.Core.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace ATAG.Core.Generators.Writers
{
    class ModelWriter : BaseWriter
    {
        public ModelWriter(string @namespace) : base(@namespace)
        { }
        public override string GenerateContent(object entity)
        {
            var model = (EntityModel)entity;

            var sb = new StringBuilder();
            int tabLevel = 0;

            sb.Append("using System;");
            sb.AppendLine();
            sb.Append("using System.Collections.Generic;");
            sb.AppendLine();
            sb.Append("using System.Linq;");
            sb.AppendLine();
            sb.Append("using System.Runtime;");
            sb.AppendLine();

            sb.Append($"namespace {_namespace}");
            sb.AppendLine();
            sb.Append("{");
            sb.AppendLine();

            tabLevel++;

            sb.Append($"{Tabs(tabLevel)}public class {model.Name}");
            sb.AppendLine();
            sb.Append($"{Tabs(tabLevel)}{{");
            sb.AppendLine();

            foreach (var prop in model.Properties)
            {
                tabLevel++;

                sb.Append($"{Tabs(tabLevel)}public {prop.Type} {prop.Name} {{ get; set; }}");
                sb.AppendLine();

                tabLevel--;
            }
            sb.Append($"{Tabs(tabLevel)}}}");
            sb.AppendLine();
            sb.Append("}");

            return sb.ToString();
        }
    }
}
```

CsClientWriter

```
using ATAG.Core.Models;
using ATAG.Core.Models.Enums;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATAG.Core.Generators.Writers
{
	public class CsClientWriter : BaseWriter
	{

		public CsClientWriter(string @namespace) : base(@namespace)
		{ }

		public override string GenerateContent(object entity)
		{
			var controllers = (List<ControllerModel>)entity;

			StringBuilder sb = new StringBuilder();
			int tabLevel = 0;

			sb.AppendLine("using System;");
			sb.AppendLine("using System.Collections.Generic;");
			sb.AppendLine("using System.Text;");
			sb.AppendLine("using System.Net.Http;");
			sb.AppendLine("using System.Text.Json;");
			sb.AppendLine("using System.Linq;");
			sb.AppendLine("using System.Threading.Tasks;");
			sb.AppendLine();

			sb.AppendLine($"namespace {_namespace}");
			sb.AppendLine("{");

			tabLevel++;

			sb.AppendLine($"{Tabs(tabLevel)}public class ApiClient");
			sb.AppendLine($"{Tabs(tabLevel)}{{");
			sb.AppendLine();

			tabLevel++;

			sb.AppendLine($"{Tabs(tabLevel)}private readonly HttpClient
_httpClient;");
			sb.AppendLine($"{Tabs(tabLevel)}private readonly string _hostUrl;");

			sb.AppendLine();

			sb.AppendLine($"{Tabs(tabLevel)}public ApiClient(HttpClient httpClient,
string hostUrl)");
			sb.AppendLine($"{Tabs(tabLevel)}{{");
			tabLevel++;
			sb.AppendLine($"{Tabs(tabLevel)}_httpClient = httpClient;");
			sb.AppendLine($"{Tabs(tabLevel)}_hostUrl = hostUrl;");
			tabLevel--;
			sb.AppendLine($"{Tabs(tabLevel)}}}");
			sb.AppendLine();

			foreach(var controller in controllers)
			{
				sb.AppendLine($"{Tabs(tabLevel)}#region {controller.Name}");
				sb.AppendLine();
```

```
                        foreach(var method in controller.Methods)
                        {
                                sb.Append($"{Tabs(tabLevel)}public                      async
Task<{method.ReturnedType}> {method.Name}(");

                                bool                  hasBodyParameter                    =
!string.IsNullOrEmpty(method.Parameters.BodyParameter.Type);

                                if (hasBodyParameter)
                                {
                                        var bp = method.Parameters.BodyParameter;
                                        sb.Append($"{bp.Type} {bp.Name}");
                                }

                                if (method.Parameters.QueryParameters.Count > 0)
                                {
                                        if (hasBodyParameter)
                                                sb.Append(", ");

                                        foreach            (var            qp            in
method.Parameters.QueryParameters)
                                        {
                                                sb.Append($"{qp.Type} {qp.Name}, ");
                                        }
                                        sb.Length -= 2;
                                }
                                sb.Append(")");
                                sb.AppendLine();

                                sb.AppendLine($"{Tabs(tabLevel)}{{");

                                tabLevel++;

                                var controllerName = controller.Name.EndsWith("Controller")
?
                                        controller.Name.Substring(0,  controller.Name.Length
- 1 - "Controller".Length)
                                        : controller.Name;

                                StringBuilder   url   =   new   StringBuilder($"_hostUrl   +
\"/api/{controllerName}");
                                if (!string.IsNullOrEmpty(method.Route))
                                {
                                        url.Append("/" + method.Route);
                                }
                                url.Append("\"");
                                if (method.Parameters.QueryParameters.Count > 0)
                                {
                                        url.Append("+\"?");
                                        foreach            (var            qp            in
method.Parameters.QueryParameters)
                                        {

        url.Append($"{qp.Name}=\"+{qp.Name}.ToString()+\"&");
                                        }
                                        url.Length -= 3;
                                }

                                sb.AppendLine($"{Tabs(tabLevel)}var url = {url};");
                                sb.AppendLine();

                                sb.Append($"{Tabs(tabLevel)}var     result     =     await
{_verbsMethods[method.Verb]}<{method.ReturnedType}>(url");

                                if (hasBodyParameter)
                                {
                                        sb.Append($",
{method.Parameters.BodyParameter.Name}");
                                }
                                sb.Append(");");
```

```
                sb.AppendLine();
                        sb.AppendLine($"{Tabs(tabLevel)}return result;");
                        tabLevel--;
                        sb.AppendLine($"{Tabs(tabLevel)}}}}");
                }
                sb.AppendLine();
                sb.AppendLine($"{Tabs(tabLevel)}#endregion");
        }

        GeneratePrivateGeneralMethods(sb, tabLevel);
    sb.AppendLine();

        tabLevel--;
        sb.AppendLine($"{Tabs(tabLevel)}}}}");
        tabLevel--;
        sb.AppendLine($"{Tabs(tabLevel)}}}}");

        return sb.ToString();
    }

    private void GeneratePrivateGeneralMethods(StringBuilder sb, int tabLevel)
    {
        var getMethod = @"
    private async Task<T> GetPrivateMethod<T>(string url)
    {
        var response = await _client.GetAsync(url);
        response.EnsureSuccessStatusCode();

        var                                 result                                 =
JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);

        return result;
    }";
        sb.Append(getMethod);

        var postMethod = @"
    private async Task<T> PostPrivateMethod<T>(string url, object data = null)
    {
        HttpContent content = new StringContent(JsonSerializer.Serialize(data ??
""""), Encoding.UTF8, ""application/json"");

        var response = await _client.PostAsync(url, content);
        response.EnsureSuccessStatusCode();

        var                                 result                                 =
JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
        return result;
    }";
        sb.Append(postMethod);

        var putMethod = @"
    private async Task<T> PutPrivateMethod<T>(string url, object data = null)
    {
        HttpContent content = new StringContent(JsonSerializer.Serialize(data ??
""""), Encoding.UTF8, ""application/json"");

        var response = await _client.PutAsync(url, content);
        response.EnsureSuccessStatusCode();

        var                                 result                                 =
JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);
        return result;
    }";
        sb.Append(putMethod);

        var deleteMethod = @"
    private async Task<T> DeletePrivateMethod<T>(string url)
    {
        var response = await _client.DeleteAsync(url);
        response.EnsureSuccessStatusCode();
```

```
                var                                     result                                  =
JsonSerializer.Deserialize<T>(response.Content.ReadAsStringAsync().Result);

                return result;
            }";
                sb.Append(deleteMethod);
            }

            private Dictionary<HttpVerb, string> _verbsMethods =>
                new Dictionary<HttpVerb, string>
                {
                    [HttpVerb.get] = "GetPrivateMethod",
                    [HttpVerb.post] = "PostPrivateMethod",
                    [HttpVerb.put] = "PutPrivateMethod",
                    [HttpVerb.delete] = "DeletePrivateMethod",
                };
        }
}
```