КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«РОЗРОБКА ДОДАТКУ ДЛЯ КООРДИНАЦІЇ РОБОТИ ОСВІТНІХ КУРСІВ З ВИКОРИСТАННЯМ ІНТЕГРАЦІЇ TELEGRAM»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Тарас Гупалик _____

Науковий керівник:
доктор фізико-математичних наук, старший науковий співробітник
Олександр Кукла _____
асистент
Олександр Мурманцев _____

Рецензент:
доктор технічних наук, доцент
Сергій Ольшевський

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра містить: 98 с., 14 джерел, 3 додатки.

У процесі виконання дипломної роботи було розроблено додаток, що сприяє ефективній координації роботи освітніх курсів за допомогою інтеграції з Telegram. За допомогою засобіи платформи .NET була розроблена архітектура програмного рішення, що ґрунтується на N-рівневій структурі, включаючи контролери RESTful API, бізнес-логіку програми та шар доступу до даних. Була реалізована можливість створення звітів у форматі PDF і інтеграція з Telegram за допомогою Telegram-бота.

Результатом роботи є додаток, що має зручну та ефективну архітектуру та функціональність, які сприяють зручній взаємодії між учасниками освітнього курсу та полегшують організацію навчального процесу.

Ключові слова: ASP.NET Core, Entity Framework Core, база даних MS SQL Server, Dependency Injection, Telegram Bot.

ЗМІСТ

ВСТУП

Освітня сфера XXI століття стикається зі значними викликами, спричиненими швидким розвитком технологій, зміною соціально-економічного контексту та зростанням потреб сучасного суспільства. Ці виклики вимагають розуміння та адаптації освітніх систем до нових реалій, забезпечення якісної освіти та підготовки компетентних громадян, здатних впоратися з викликами та проблемами сучасного світу.

Одним із найактуальніших питань сьогодення у сфері освіти є необхідність адаптації до швидкого технологічного розвитку. Розуміння та впровадження нових інформаційних технологій у навчальний процес стає необхідним, оскільки це дозволяє забезпечити студентам доступ до актуальної та розширеної інформації, стимулює їхню творчість та навички критичного мислення. Крім того, впровадження онлайн-навчання та дистанційних платформ стає все більш популярним, забезпечуючи гнучкість та доступність освітніх можливостей.

Тому **актуальність цієї роботи** визначається появою нових технологій та змінами у підходах до навчання, а відтак потребою у створенні цифрових середовищ, які забезпечують доступ до навчальних матеріалів, сприяють інтерактивному навчанню та спілкуванню між учасниками навчального процесу.

Метою роботи є розробка додатку для координації роботи освітніх курсів з використанням інтеграції Telegram, який покликаний:

- спростити та поліпшити організацію навчального процесу;
- забезпечити зручну комунікацію та обмін інформацією між викладачами та студентами;
- сприяти структуруванню навчального матеріалу, забезпечувати доступність ресурсів та допомагати учасникам навчального процесу ефективно використовувати час та ресурси.

Структура роботи. У Розділі 1 роботи детально розглядаються обрані засоби розробки .NET, зокрема мова C#, середовища розробки Visual Studio,

Azure Data Studio та інструменти тестування Postman, бібліотека Entity Framework, СУБД MS SQL Server та RESTful API. .

Розділ 2 присвячений опису програмної реалізації, включаючи загальні принципи роботи додатку, архітектуру програмного рішення, реалізацію автентифікації та використання Telegram bot.

Робота також містить висновки до розділів 1 і 2, загальні висновки, список використаних джерел та додатки.

РОЗДІЛ 1. Засоби розробки

1.1. Платформа .NET

.NET - це платформа для розробників, що складається з інструментів розробки, мов програмування та бібліотек, призначених для створення різноманітних програм. Платформа покликана виконувати такі завдання:

- надавати об’єктно-орієнтоване середовище розробки як для локального збереження та виконання коду, так і для його віддаленого виконання;
- надавати середовище виконання коду, в якому зведена до мінімуму ймовірність конфліктів у процесі розгортання програмного забезпечення та управління його версіями та гарантується безпечне виконання коду, в тому числі коду, створеного третіми особами;
- забезпечувати єдині принципи розробки для різних типів застосунків, таких як програми Windows та веб-застосунки;
- забезпечувати взаємодію на основі промислових стандартів, що гарантує інтеграцію коду платформи .NET з будь-яким іншим кодом.

.NET містить стандартний набір бібліотек базових класів і API, спільних для всіх програм .NET. Кожна модель програми також може передбачати використання додаткових API, специфічних для операційних систем, на яких вона функціонує, або цілей, які вона покликана виконувати. Наприклад, ASP.NET — це кросплатформна веб-платформа, яка передбачає додаткові API для створення веб-програм для Linux або Windows.

За час існування платформи .NET було реалізовано декілька її версій, зокрема:

- .NET Framework, що відкриває доступ до широких функціональних можливостей Windows та Windows Server, а також широко використовується для хмарних обрахунків на основі Windows.
- Mono - кросплатформна реалізація платформи .NET, що використовується для застосунків Android, iOS та Wasm.

- .NET (Core) - кросплатформна та опенсорсна реалізація платформи .NET, що була переосмислена відповідно до запитів епохи хмарних технологій, але водночас залишається значною мірою сумісною з платформою .NET Framework. Використовується для застосунків Linux, macOS та Windows.

1.1.1. Середовище виконання CLR

Середовище виконання CLR (The Common Language Runtime) є основою для усіх програм .NET. Її основними функціями є:

- збирання сміття;
- безпека пам’яті та типів;
- високорівнева підтримка мов програмування;
- кросплатформний дизайн.

Середовище виконання автоматично обробляє макет об’єктів та управляє посиланнями на об’єкти, звільняючи їх, коли вони більше не використовуються. Об’єкти, тривалість життя яких управляється таким чином, називаються керованими даними. Середовище CLR спрощує розробку компонентів та застосунків, об’єкти яких можуть працювати у різних мовах. Наприклад, розробник може визначити клас, а потім на іншій мові створити похідний від нього клас або викликати метод із початкового класу. Така міжмовна інтеграція можлива, оскільки мовні компілятори та засоби, призначені для середовища виконання, використовують систему загальних типів, визначену цим середовищем.

Метадані усіх керованих компонентів містять інформацію про компоненти та ресурси, на основі яких вони побудовані. Середовище виконання використовує ці дані, аби забезпечити наявність усіх ресурсів, необхідних для компоненту або застосунку.

Середовище виконання має такі переваги:

- підвищення швидкодії;
- можливість легко використовувати компоненти, розроблені іншими мовами;

- розширює типи, що надаються бібліотекою класів;
- мовні можливості (наприклад, наслідування, інтерфейси та перевантаження) для об’єктно-орієнтованого програмування;
- підтримка явного вільного потоку, що дозволяє створювати багатопоточні та масштабовані застосунки;
- підтримка структурованої обробки виключень;
- підтримка налаштовуваних атрибутів;
- використання делегатів для підвищення типобезпеки та рівня захисту.

1.1.2. Мови платформи .NET

На платформі .NET Microsoft пропонує декілька мов програмування, до яких належать C#, F# та Visual Basic.

- C# - це сучасна об’єктно-орієнтована та типобезпечна мова програмування, що належить до сімейства мов C. Вона має широку підтримку в екосистемі та усіх робочих навантаженнях .NET. Поруч із об’єктно-орієнтованими принципами, мова C# включає в себе численні функції з інших парадигм, не в останню чергу функціональне програмування. Низькорівневі функції підтримують сценарії з високою ефективністю без написання небезпечного коду. Більша частина середовища виконання та бібліотек .NET написана на C#, а вдосконалення цієї мови приносить користь усім .NET розробникам.
- F# - це мова програмування для написання короткого, надійного та ефективного коду. Програмування на мові F# є орієнтованим на дані, а код передбачає перетворення даних за допомогою функцій. F# повністю розкриває потужності платформи .NET та вдало взаємодіє з мовою C# у програмах, що вимагають використання декількох мов програмування одночасно.
- Visual Basic має більш розлогий синтаксис, що є ближчим до природної людської мови. Розробники Visual Basic користуються перевагами зрілої та стабільної об’єктно-орієнтованої мови у поєднанні з постійно

зростаючою платформою .NET та покращеннями поточних інструментів. Утім, деякі робочі навантаження .NET не підтримуються у Visual Basic, і тому для подібних сценаріїв розробники зазвичай використовують C#.

1.1.3. Бібліотеки класів .NET

Платформа .NET має широкий стандартний набір бібліотек класів, що дають можливість об'єднувати корисний функціонал у модулі і згодом використовувати його у декількох різних програмах. Вони також можуть служити для підключення функцій, що були непотрібними або невідомими під час запуску програми. Бібліотеки класів описуються у форматі файлу .NET Assembly.

Є три типи бібліотек класів, доступних для використання на платформі:

1. Бібліотеки класів, що залежать від платформи. Такі бібліотеки прив'язані до однієї .NET платформи (наприклад, платформа .NET Framework для Windows чи Xamarin для iOS) і відповідно мають доступ до усіх API цієї платформи, але можуть використовуватися лише застосунками та бібліотеками, призначеними для цієї платформи. Бібліотеки, що залежать від платформи, були основним типом бібліотек класів для платформи .NET, і навіть після появи інших реалізацій .NET вони лишаються домінуючим типом бібліотек.
2. Портативні бібліотеки класів мають доступ до кількох наборів API і можуть використовуватися застосунками та бібліотеками, призначеними для декількох платформ. Вони також можуть мати залежності у відомому середовищі виконання, але таке середовище є штучним та являє собою перетин множин конкретних реалізацій .NET. Конфігурація платформи, тобто набір платформ, для яких буде доступна підтримка, обирається при створенні портативної бібліотеки. Багато розробників бібліотек уже перейшли зі створення декількох бібліотек, що залежать від платформи, для одного вихідного коду (з використанням директив умовної компіляції) на портативні бібліотеки. Є декілька підходів щодо

отримання доступу до функцій певної платформи в портативних бібліотеках, найпоширенішим з яких є механізм захоплення.

3. Бібліотеки класів .NET Standard є поєднанням залежних від платформи та портативних бібліотек і поєднують ключові характеристики обох типів. Вони є залежними від платформи в тому сенсі, що вони надають усі функціональні можливості базової платформи (без штучних платформ або їх перетинів) і водночас є портативним в тому сенсі, що вони працюють на усіх підтримуваних платформах. Бібліотеки .NET Standard підтримуються у таких реалізаціях, як .NET Core, .NET Framework, Mono та Універсальна платформа Windows (UWP).

1.1.4. Моделі розгортання

Програми .NET можна публікувати у двох різних режимах:

1) Автономні програми включають в себе середовище виконання .NET та відповідні бібліотеки. Застосунки при цьому можуть бути як одно-, так і багатофайловими, і навіть бути запущеними на комп'ютері, на якому не встановлено середовище виконання .NET. Автономні програми завжди націлені на єдину конфігурацію операційної системи та архітектури.
2) Програми, що залежать від платформи, потребують відповідної сумісної версії середовища виконання .NET, що зазвичай встановлюється глобально. Такі програми можна публікувати для однієї конфігурації операційної системи та архітектури або ж як портативні, призначені для усіх підтримуваних конфігурацій.

Програми .NET за замовчуванням запускаються із власним виконуваним файлом. Виконуваний файл залежить від операційної системи та архітектури. Програми також можна запускати за допомогою команди dotnet.

Сьогодні .NET продовжує розвиватися, і Microsoft регулярно оновлює та вдосконалює платформу. Широко використовувана розробниками в усьому світі, і платформа .NET стала популярною платформою для створення

широкого спектру програм, від десктопних та мобільних застосунків до веб-сервісів і хмарних служб.

1.2. Мова C#

C# є об’єктно-орієнтованою мовою програмування і водночас включає підтримку компонентно-орієнтованого програмування. Сучасний дизайн програмного забезпечення все більше покладається на компоненти програмного забезпечення у формі самодостатніх пакетів функціональних можливостей, що описуються самостійно. Ключовим для таких компонентів є те, що вони представляють модель програмування з властивостями, методами та подіями; вони мають атрибути, які надають декларативну інформацію про компонент; і вони включають власну документацію. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, що робить C# дуже природною мовою для створення та використання програмних компонентів.

1.2.1. Основні характеристики C#

C# дозволяє розробникам створювати численні безпечні та надійні .NET програми завдяки таким ключовим характеристикам мови:

- Збирання сміття автоматично звільняє пам’ять, зайняту недоступними невикористаними об’єктами.
- Типи, що допускають значення Nullable, захищають від змінних, які не посилаються на виділені об’єкти.
- Обробка винятків забезпечує структурований і розширений підхід до виявлення та відновлення помилок.
- Лямбда-вирази підтримують методи функціонального програмування.
- Синтаксис Language Integrated Query (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела.
- Підтримка мови для асинхронних операцій забезпечує синтаксис для побудови розподілених систем.

- Уніфікована система типів. Усі типи C#, включаючи примітивні типи, такі як int і double, успадковуються від одного кореневого типу об’єкта. Усі типи мають спільний набір операцій. Значення будь-якого типу можна зберігати, транспортувати та працювати з ними узгоджено.

Особлива увага у C# приділяється управлінню версіями, що покликане забезпечити сумісність програм та бібліотек з часом. Питання управління версіями суттєво вплинули на такі аспекти розробки C#, як окремі модифікатори virtual та override, правила вирішення перевантаження методів та підтримка явного оголошення членів інтерфейсу.

Все, що стосується декларації класу, у C# міститься в самій декларації. Визначення класів не потребують окремих файлів заголовків або файлів мови визначення інтерфейсу (IDL). C# також забезпечує підтримку структур, що у C# є обмеженим, полегшеним типом, який при створенні пред’являє менше вимог до операційної системи та пам’яті, ніж звичайний клас. Структура не може успадкувати від класу або бути успадкованою, але структура може реалізувати інтерфейс. C# забезпечує повну підтримку делегатів, що є типобезпечними типами за посиланнями та інкапсулюють методи з певними підписами та типами повернення.

C# реалізує компонентно-орієнтовані функції, такі як властивості, події та декларативні конструкції (так звані атрибути). Компонентно-орієнтоване програмування підтримується підтримкою CLR для зберігання метаданих із кодом для класу. Метадані описують клас, включаючи його методи та властивості, а також його потреби в безпеці та інші атрибути, наприклад, чи можна його серіалізувати; код містить логіку, необхідну для виконання своїх функцій. Таким чином, скомпільований клас є самодостатньою одиницею; отже, середовище розміщення, яке знає, як читати метадані та код класу, не потребує іншої інформації, щоб використовувати їх. Використовуючи C# і CLR, можна додавати власні метадані до класу, створюючи власні атрибути.

1.2.2. Система типів в C#

C# є строго типізованою мовою. Кожна змінна та константа має свій тип, як і кожен вираз, результатом обчислення якого є значення. Кожне оголошення методу задає ім’я, тип та вид (значення або посилання) для кожного вхідного параметру та для значення, що повертається. У бібліотеці класів .NET визначені вбудовані числові типи та комплексні типи, що представляють різноманітні конструкції. До них відноситься файлова система, мережеві підключення, колекції та масиви об’єктів, а також дати. Типова програма на C# використовує типи із цієї бібліотеки класів та користувацькі типи, що моделюють унікальні концепції конкретної сфери застосування.

У типах може зберігатися наступна інформація:

- місце, необхідне для зберігання змінної цього типу;
- максимальне та мінімальне значення, що можуть бути представлені;
- члени, що містяться у типі (методи, поля, події і т.д.);
- базовий тип, від якого наслідує цей тип;
- реалізовані ним інтерфейси;
- дозволені види операцій.

Компілятор використовує інформацію про типи, аби перевірити, чи всі операції, що виконуються у коді, є типобезпечними, та додає цю інформацію у виконуваний файл у вигляді метаданих. Середовище CLR використовує ці метадані під час виконання для подальшого забезпечення безпеки типу при виділенні та звільненні пам’яті.

Особливості системи типів у .NET визначаються такими двома ключовими характеристиками:

1) вона підтримує принцип наслідування. Типи можуть бути похідними від інших типів, які називаються базовими типами. Похідний тип наслідує всі (з деякими обмеженнями) методи та властивості базового типу, який в свою чергу може бути похідним від якогось іншого типу, при цьому похідний тип наслідує елементи обох базових типів в ієрархії наслідування. Усі типи, включаючи вбудовані числові типи, в кінцевому рахунку є похідними від одного базового типу System.Object. Ця уніфікована ієрархія типів називається Системою загальних типів CTS.

2) Кожен тип у CTS визначається як тип за значенням або тип за посиланням. Це справедливо й для усіх користувацьких типів, в тому числі включених у бібліотеку класів .NET або визначених розробником. Якщо у визначенні типу використовується ключове слово struct, він є типом за значенням; якщо ключове слово class або record, він є типом за посиланням. Для типів за посиланням та типів за значенням використовуються різні правила компіляції, і вони демонструють різну поведінку під час виконання.

1.2.3. Мова інтегрованих запитів LINQ

Абревіатура LINQ позначає цілий набір технологій, що створюють та використовують можливості інтеграції запитів безпосередньо в мову C#. Традиційно запити до даних виражаються у вигляді простих рядків без перевірки типів при компіляції або підтримки IntelliSense. До того ж, розробнику доводиться вивчати різноманітні мови запитів для кожного типу джерел даних: баз даних SQL, XML-документів, різноманітних веб-служб тощо. Технології LINQ перетворюють запити на зручну мовну конструкцію, що застосовується за тими ж принципами, що й класи, методи та події.

Для розробника, який створює запити, найбільш очевидною частиною LINQ є інтегровані запити, що використовують декларативний синтаксис. За допомогою синтаксису запитів можна виконувати фільтрацію, впорядкування та групування даних з відповідного джерела, обмежуючись мінімальним об’ємом програмного коду. Одні й ті ж базові запити дозволяють однаково легко отримувати та перетворювати дані з баз даних SQL, наборів даних ADO .NET, XML-документів, XML-потоків та колекцій .NET або будь-яких колекцій об’єктів, що підтримують інтерфейс IEnumerable. До того ж, сторонні розробники забезпечують підтримку LINQ для численних веб-служб та інших реалізацій баз даних.

Безпроблемна інтеграція LINQ з мовою C# вимагала внесення у мову істотних змін:

- Вирази запитів використовують декларативний синтаксис, аналогічний SQL або XQuery, для виконання запитів до колекцій IEnumerable. Під час компіляції синтаксис запиту перетворюється через реалізацію методів розширення стандартних операторів запитів провайдером LINQ. Програми керують стандартними операторами запитів в області, вказуючи відповідний простір імен за допомогою директиви using.
- Неявно типізовані змінні (var). Замість того, щоб явно задавати тип при оголошенні та ініціалізації змінної, можна використовувати модифікатор var, аби повідомити компілятору про необхідність визначити та присвоїти тип відповідній змінній. Змінні, оголошені як var, так само строго типізовані, як і змінні, тип яких задається явно. Використання var уможливлює створення анонімних типів, утім його можна використовувати лише для локальних змінних. Масиви також можуть бути оголошені шляхом неявної типізації.
- Методи розширення. Метод розширення - це статичний метод, що може бути пов’язаним із певним типом і дозволяє викликати його так, наче він є методом екземпляру типу. Ця можливість дозволяє “додавати” нові методи у вже наявні типи, фактично не змінюючи їх. Стандартні оператори запитів - це набір методів розширення, що надає функції запитів LINQ будь-якому типу, що реалізує IEnumerable<T>.
- Лямбда-вирази. Лямбда-вираз - це вбудована функція, що використовує оператор =>, аби відокремлювати вхідні параметри від тіла функції. Під час компіляції вона може бути перетворена на делегат або дерево виразів. У програмуванні LINQ лямбда-вирази можуть фігурувати у процесі виконання прямих викликів методів до стандартних операторів запитів.

1.3. Середовище розробки

1.3.1. Visual Studio

Visual Studio - це інтегроване середовище розробки (IDE), створене Microsoft та вперше випущене у 1997 році. Це один із найпопулярніших інструментів розробки, що дозволяє створювати, тестувати та розгортати програмні застосунки. Цей продукт передбачає все, що може знадобитися у процесі розробки, як, наприклад, редактор коду, відлагоджувач, компілятори та низка інших інструментів та сервісів для оптимізації процесу розробки програмного забезпечення.

До того ж, продукт підтримує широкий спектр мов програмування, включаючи C#, C++ та Python, що дозволяє розробнику обрати мову, що найбільш вдало підходить до його проекту, та надає усі необхідні інструменти для роботи з цією мовою. Як комплексна IDE, Visual Studio включає в себе редактор коду, що підтримує IntelliSense, а також рефакторинг коду, вбудований відлагоджувач, що працює як на рівні джерельного коду, так і на машинному рівні, вбудований термінал, вбудований контроль версій та численні інструменти генерації коду. До того ж, продукт передбачає перевірку коду, автоматизує тестування та розгортання коду, а також має вбудовану підтримку командної роботи та управління проектами.

Visual Studio також передбачає цілу низку доповнень, що розширюють наявний функціонал, як, приміром, розширену мовну підтримку, нові засоби відлагоджування та нові функції для роботи з кодом. Також програму можна легко інтегрувати з різноманітними інструментами та службами, які можна встановити та налаштувати безпосередньо із самої IDE.

1.3.2. Azure Data Studio

Azure Data Studio - це кросплатформений інструмент для роботи з базами даних, націлений на спеціалістів з даних, що працюють із локальними та хмарними платформами даних у Windows, macOS та Linux. Він надає сучасний редактор з такими функціями, як IntelliSense, сніпети коду, інтеграція системи управління версіями та вбудований термінал. Інструмент також передбачає вбудовані діаграми наборів результатів запитів та кастомізовані інформаційні

панелі. Він підтримує редагування коду SQL за допомогою IntelliSense, забезпечуючи ефективність написання коду за допомогою таких функцій, як багатофункціональний редактор SQL, автодоповнення ключових слів, сніпети коду, навігація по колу та інтеграція з Git. Користувачі можуть виконувати необхідні SQL-запити, переглядати та зберігати результати у різноманітних форматах, таких як текст, JSON або Excel, редагувати дані, керувати підключеннями до бази даних та переглядати об’єкти бази даних.

Azure Data Studio також пропонує розумні SQL-сніпети, що генерують правильний синтаксис для створення баз даних, таблиць, процедур, користувачів, логінів, ролей та оновлення існуючих об’єктів. Користувачі також можуть створювати власні SQL-сніпети, кастомізовані панелі сервера та бази даних для моніторингу та усунення проблем з продуктивністю у базах даних. Інструмент підтримує групи серверів для організації інформації про підключення і надає вбудований термінал, де користувачі можуть використовувати обрані інструменти командного рядка, такі як Bash, PowerShell, sqlcmd, bcp, та ssh в інтерфейсі Azure Data Studio. Azure Data Studio підтримує створення та використання доповнень, що дозволяють користувачам розширювати її функціональні можливості та кастомізувати свій досвід.

У порівнянні з SQL Server Management Studio (SSMS), Azure Data Studio використовується насамперед для таких завдань, як редагування та виконання запитів, створення діаграм та візуалізація наборів результатів, виконання адміністративних задач через вбудований термінал і робота на macOS чи Linux. Натомість SSMS більше підходить для складних адміністративних або платформених конфігурацій, управління безпекою, налаштування продуктивності, побудови діаграм баз даних та конструкторів таблиць, доступу до зареєстрованих серверів та використання статистики оперативних запитів або клієнтської статистики.

1.3.3. Postman

Postman - це платформа API, що пропонує повний набір інструментів та функцій для полегшення розробки, тестування, документування та управління API. Клієнт API, основний інструмент Postman, надає зручний інтерфейс для дослідження, відлагоджування та тестування API. Він підтримує різноманітні протоколи, включаючи HTTP, REST, SOAP, GraphQL та WebSockets. Клієнт має функцію автоматичного визначення мови відповіді, посилань та форматування тексту, що робить перевірку відповідей API більш зручною. Він також пропонує вбудовану підтримку популярних протоколів автентифікації, таких як OAuth, AWS Signature та Hawk. Колекції Postman забезпечують ефективну організацію та повторне використання запитів API з додатковою можливістю інтегрування коду JavaScript для координації та автоматизації запитів, а також візуалізації відповідей API у вигляді діаграм та графіків.

Для тестування API Postman пропонує інтегроване середовище тестування, де користувачі можуть створювати та виконувати тести безпосередньо на платформі. Зокрема користувачі можуть писати функціональні, інтеграційні та регресійні тести. До інструментів платформи Postman також належить Newman, інструмент командного рядка, що дозволяє запускати та тестувати колекції Postman як частину конвеєрів неперервної інтеграції / неперервної доставки (CI/CD). Середовище виконання у Postman створене на базі Node.js та передбачає підтримку загальних шаблонів та бібліотек, що полегшує та пришвидшує створення тестів.

1.4. Бібліотека Entity Framework

Entity Framework - це набір технологій в ADO.NET, що підтримують розробку програмних застосунків, орієнтованих на роботу з даними. Це об’єктно-реляційний мапер, що дозволяє розробникам .NET працювати з реляційними даними, використовуючи доменні об’єкти, і за рахунок цього звільняє їх від написання значної частини коду доступу до даних.

Використовуючи Entity Framework, розробники можуть працювати з даними у форматі доменних об’єктів та полів, таких як клієнти та адреси

клієнтів незалежно від таблиць та стовпців бази даних, де вони зберігаються. Завдяки цим технологіям розробники можуть працювати з даними на більш високому рівні абстракції, а також створювати та підтримувати програми, орієнтовані на дані, з меншим об’ємом коду, ніж у традиційних застосунках. Вони можуть вивести концептуальну модель, що базується на інформації про визначені розробником типи об’єктів та додаткові конфігурації. Мапінгові метадані, що створюються у процесі виконання, можуть за потреби стати основою для автоматичного створення бази даних.

Використання технологій Entity Framework має такі переваги:

- Застосунки можуть працювати у межах більш програмоцентричної концептуальної моделі, в тому числі з використанням типів з наслідуванням, складних елементів та відношень;
- Функціонування застосунку стає менш залежним від чітко визначеного механізму обробки даних або схеми зберігання;
- Мапінги між концептуальною моделлю та конкретною схемою зберігання можуть змінюватися без зміни коду самого застосунку;
- Розробники можуть працювати зі сталою об’єктною моделлю застосунку, яку можна співвідносити з різноманітними схемами зберігання, навіть якщо вони реалізовані у різних джерелах даних;
- Декілька концептуальних моделей можуть співвідноситися з однією схемою зберігання;
- Інтегрована підтримка запитів забезпечує перевірку синтаксису запитів на відповідність концептуальної моделі під час компіляції.

1.4.1. Елементи Entity Framework

Entity Framework використовує інформацію в моделі та мапінгових файлах для перетворення об’єктних запитів до типів сутностей, представлених у концептуальній моделі, у запити, що відносяться до джерела даних. Результати запиту матеріалізуються в об’єкти, якими управляє Entity Framework. Entity Framework передбачає такі види запитів до концептуальної моделі:

1) Тип LINQ to Entities забезпечує підтримку запитів, інтегрованих у мову (LINQ) до запитів типів сутностей, визначених у концептуальній моделі. Він дозволяє розробникам писати запити до концептуальної моделі Entity Framework, використовуючи Visual Basic або Visual C#. Запити до Entity Framework представлені запитами дерева команд, які виконуються в контексті об'єкту. LINQ to Entities перетворює запити LINQ у запити дерева команд, виконує запити до Entity Framework та повертає об'єкти, що можуть використовуватися як Entity Framework, так і LINQ.
2) Тип Entity SQL - це незалежний від середовища зберігання діалект SQL, що працює безпосередньо із сутностями в концептуальній моделі та підтримує концепції Entity Data Model. Entity SQL використовується як з об'єктними запитами, так і з запитами, що виконуються за допомогою провайдера EntityClient. Зазвичай Entity SQL використовується у таких випадках:
    - якщо запит доводиться динамічно змінювати під час виконання;
    - якщо запит необхідно визначити як частину визначення моделі;
    - якщо EntityClient використовується для повернення даних сутності у вигляді наборів рядків через EntityDataReader.

1.4.2. Підходи до розробки в Entity Framework

Є три ключові підходи до роботи з моделями та базами даних в Entity Framework:

1) "Спершу база даних" (Database First). При наявності вже готової бази даних конструктор Entity Framework, вбудований у Visual Studio, може автоматично створити модель даних, що складається із класів та властивостей, що відповідають існуючим об'єктам бази даних, таким як таблиці та стовпці. Інформація про структуру бази даних, модель даних та співвідношення між ними зберігається у форматі XML у .edmx файлі.

Конструктор Entity Framework також має графічний користувацький інтерфейс, який можна використовувати для відображення та редагування .edmx файлу.

2) “Спершу модель” (Model First). Якщо готової бази даних ще немає, можна розпочати зі створення моделі у файлі .edmx, використовуючи графічний конструктор Entity Framework у Visual Studio. На основі готової моделі конструктор Entity Framework може генерувати оператори DDL для створення бази даних. Як і у випадку з Database First, інформація про модель та мапінг зберігається у .edmx файлі.
3) “Спершу код” (Code First). Незалежно від наявності готової бази даних, Entity Framework можна використовувати і без використовування конструктора або .edmx файлу. Якщо бази даних немає, розробник може створити свої власні класи та властивості, що відповідають таблицям та стовпцям. Якщо база даних є, інструменти Entity Framework можуть створювати відповідні класи та властивості. Мапінг між схемою зберігання та концептуальною моделлю, представленою написаним кодом, здійснюється за допомогою спеціального мапінгового API. Дозволяючи Code First створити базу даних, розробник може використовувати Code First Migrations для автоматизації процесу розгортання бази даних у робочому середовищі. Migrations також можуть автоматизовувати розгортання змін схеми бази даних у робочому середовищі при зміні моделі даних.

1.5. MS SQL Server

SQL Server - це система управління реляційними базами даних, розроблена та підтримувана Microsoft. Основною функцією SQL Server як сервера баз даних є зберігання та видобуток даних, що використовуються іншими застосунками. Як і інше програмне забезпечення СУБД, SQL Server побудований на основі SQL, стандартної мови для взаємодії з реляційними

базами даних. SQL Server прив'язаний до Transact-SQL, реалізації SQL від Microsoft, що додає набір пропрієтарних програмних конструкцій.

Як і інші СУБД, SQL Server в основному базується на структурі таблиць, що з'єднують між собою пов'язані елементи даних в різноманітних таблицях, такиим чином уникаючи надлишкового зберігання даних у різних місцях бази даних. Реляційна модель також зберігає референтну цілісність задля забезпечення точності даних. Дотримання принципів ACID гарантує надійну обробку транзакцій бази даних.

SQL Server складається із двох основних компонентів:

1) Ядро бази даних, що складається із реляційного механізму, що обробляє запити, та механізму зберігання, який управляє файлами бази даних, сторінками, індексами тощо. Об'єкти бази даних, такі як збережені процедури та тригери також створюються та виконуються ядром бази даних.

2) SQLOS, тобто операційна система SQL Server, що знаходиться під реляційним механізмом та механізмом зберігання. SQLOS відповідає за цілу низку функцій операційної системи, таких як управління пам'яттю та вводом/виводом, обробка винятків і служби синхронізації.

Поруч із SQL Server Microsoft надає численні інструменти та послуги для управління даними та бізнес-аналітики. Для управління даними передбачені такі інструменти, як SQL Server Integration Services (SSIS), SQL Server Data Quality Services та SQL Server Master Data Services. Для розробки баз даних SQL Server має інструменти SQL Server Data, а для їх управління, розгортання та моніторингу - SQL Server Management Studio (SSMS). Для аналізу даних SQL Server пропонує SQL Server Analysis Services (SSAS), а SQL Server Reporting Services (SSRS) надають звіти та візуалізацію даних.

1.6. RESTful API

RESTful API - це інтерфейс, що використовується двома комп'ютерними системами для безпечного обміну інформацією через Інтернет, функціонуючи як шлюз між відповідними клієнтами та ресурсами.

1.6.1. Принципи архітектурного стилю REST

Ключовими принципами архітектурного стилю REST є:

- Єдиний інтерфейс. Сервер передає інформацію у стандартному форматі, навіть якщо цей формат відрізняється від внутрішнього представлення ресурсу у сервісному додатку;
- Відсутність збереження стану. Це конструктивне обмеження передбачає, що сервер виконує кожен клієнтський запит незалежно від усіх попередніх запитів. Клієнти можуть запитувати ресурси у будь-якому порядку, і кожен запит або ізольований від інших запитів, або його стан не зберігається.
- Багаторівнева система. У багаторівневій системній архітектурі клієнт може підключатися до інших авторизованих посередників між клієнтом та сервером і все одно отримувати відповідь від сервера. Сервери також можуть передавати запити іншим серверам.
- Ємність кешу. Веб-служби RESTful підтримують кешування, тобто процес зберігання деяких відповідей на клієнті або на посереднику для скорочення часу відповіді сервера.
- Код за запитом. Сервери можуть тимчасово розширювати або налаштовувати функціональні можливості клієнта, передаючи кож програмного забезпечення.

1.6.2. Переваги RESTful API

Використання RESTful API має такі переваги:

- Можливість масштабування. Системи, що реалізують RESTful API, можуть ефективно масштабуватися завдяки оптимізації взаємодії між сервером та клієнтом. Відсутність збереження станів знімає навантаження із сервера, адже йому не потрібно зберігати інформацію

про попередні запити клієнта, а відлагоджене кешування частково повністю усуває деякі взаємодії між клієнтом та сервером.

- Гнучкість. Веб-служби RESTful підтримують повне розділення клієнта та сервера. Вони спрощують і розділяють різноманітні серверні компоненти, аби кожна частина могла розвиватися незалежно. Зміни платформи або технології у серверній частині не впливають на клієнтську частину. Можливість розділення функцій застосунку на рівні ще більше підвищує гнучкість, адже розробники, наприклад, можуть вносити зміни у рівень бази даних, не переписуючи логіку всього застосунку.
- Незалежність. RESTful API не залежать від використовуваної технології. Як клієнтські, так і серверні застосунки можуть бути створені на різних мовах програмування, не впливаючи при цьому на структуру API. Також можна змінювати базову технологію на будь-якому рівні, не впливаючи на обмін даними.

ВИСНОВКИ ДО РОЗДІЛУ 1

Засоби розробки, описані у першому розділі, є важливими компонентами при розробці програмного забезпечення на платформі .NET.

1) Платформа .NET є потужним набором інструментів для розробки програмного забезпечення, яка надає середовище виконання CLR (Common Language Runtime) і підтримує різні мови програмування, такі як C#, VB.NET, F# та інші. Ця платформа дозволяє розробникам створювати широкий спектр додатків, включаючи веб-додатки, мобільні додатки та додатки для хмарних сервісів, забезпечуючи безпеку, надійність та продуктивність програм.
2) Мова C# є однією з основних мов платформи .NET. Вона має багато характеристик, які роблять її потужним та гнучким інструментом для розробки програм.
3) Entity Framework є важливою бібліотекою для роботи з базами даних в середовищі .NET. Вона надає зручні інструменти для створення, моделювання та взаємодії з базами даних.

4) MS SQLServer є реляційною системою управління базами даних, яка є популярним вибором для розробників .NET-додатків. Вона забезпечує надійне зберігання даних та широкий набір функцій для роботи з ними.
5) RESTful API є архітектурним стилем, який дозволяє побудувати гнучкі та легкі використовувати веб-сервіси. Він базується на принципах ресурсно-орієнтованої архітектури та надає багато переваг, таких як простота використання та розширення.

У цілому, знання та використання засобів розробки, описаних у роботі, дозволять розробникам на платформі .NET створювати потужні та сучасні програмні рішення, ефективно працювати з базами даних, розробляти веб-сервіси та забезпечувати високу якість своїх додатків.

РОЗДІЛ 2. Опис програмної реалізації

2.1. Загальні принципи роботи додатку

У процесі виконання цієї роботи нами було розроблено додаток, функцією якого є координація роботи освітніх курсів з підготовки ЗНО. Основною метою додатку є зробити взаємодію між викладачем та учнями більш ефективною та полегшити організацію навчального процесу.

Розроблений додаток має низку функціональних можливостей, що дозволять викладачу розсилати навчальні матеріали, видавати домашні завдання та перевіряти їх, використовуючи адміністративну панель.

Однією з ключових переваг додатку є його інтеграція з популярною платформою Telegram. Кожен учень отримує доступ до відповідних матеріалів через спеціально розробленого телеграм-бота, що забезпечує миттєвий доступ до навчального контенту та зручну комунікацію між викладачем та учнями.

Основні можливості додатку включають:

1. Розсилання навчальних матеріалів: викладач може легко завантажувати та розсилати навчальні матеріали, такі як презентації, текстові документи, відео або аудіозаписи. Учні отримають сповіщення про новий матеріал та зможуть переглянути його через телеграм-бот.
2. Видавання домашніх завдань: викладач може створювати завдання та надсилати їх учням через адміністративну панель. Вся необхідна інформація буде доступною для кожного учня, що допоможе забезпечити структурованість та організованість навчального процесу.
3. Перевірка та оцінювання домашніх завдань: адміністративна панель дозволяє легко переглядати та оцінювати завдання, надіслані учнями. Це спрощує процес перевірки та надає можливість оперативно додавати коментарі та оцінки.
4. Записи занять: однією із функцій розробленого додатку є доступ до записів проведених занять. Через адміністративну панель викладач додає покликання на відеозапис відповідного заняття, розміщений на платформі

Youtube, і це покликання автоматично відображається у відповідному розділі телеграм-бота.

2.2. Архітектура програмного рішення

У своєму фінальному вигляді наш продукт є сукупністю Інтернет-сервісів, основою якої є RESTful API, що дозволяє здійснювати взаємодії між його компонентами у форматі JSON, використовуючи для цього HTTP-протокол.

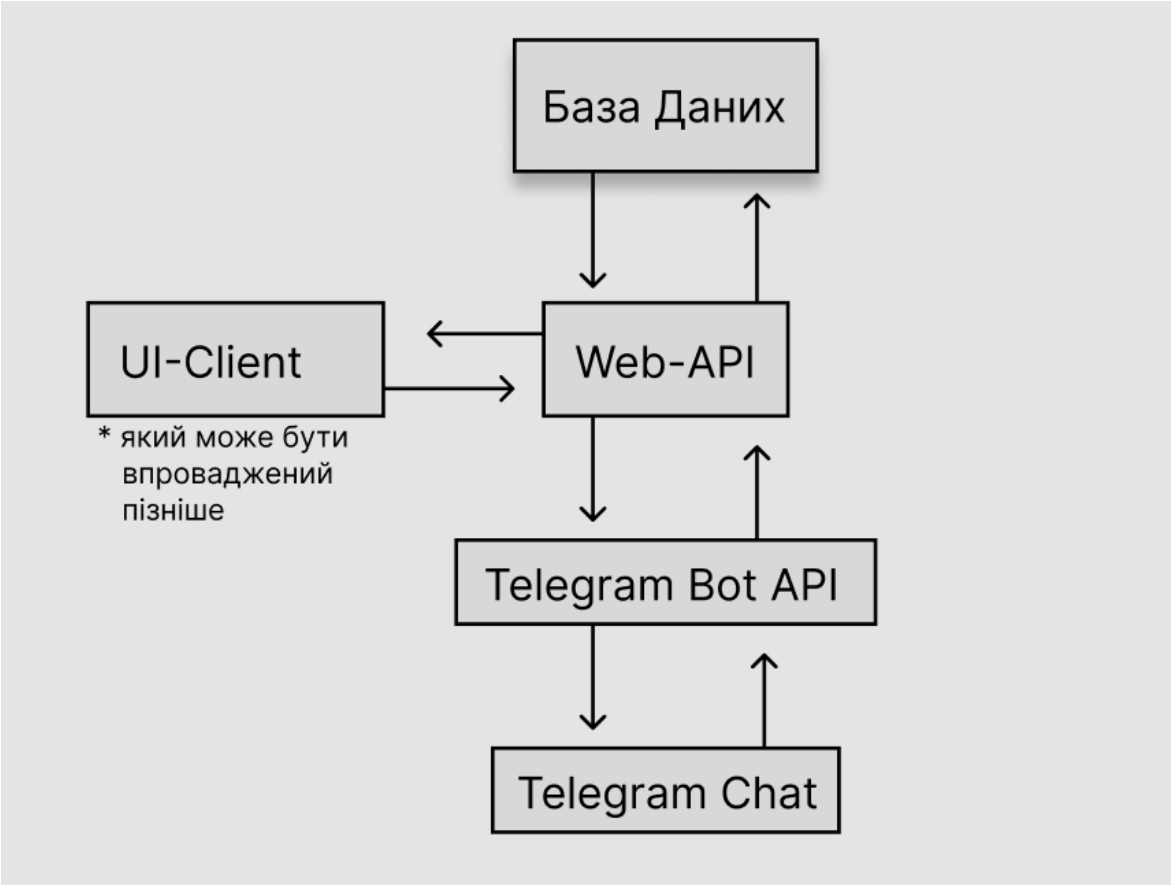
Основними складовими додатку є:

- Web-API (реалізація RESTful платформою .NET)
- База даних MS SQL Server
- Telegram Bot API

Зосередимося на кожному із пунктів детальніше

Web-API - це компонент, що надає користувачу з відповідними правами (наприклад, адміністратору) інтерфейс для всіх можливих дій, дозволених HTTP-методами, а саме: отримання даних (метод GET), додавання даних (метод POST), редагування (метод PUT/PATCH) та видалення (метод DELETE). Дані кінцевого користувача будуть зберігатися в надійній та безпечній базі.

Telegram Bot API є проміжним елементом між чатом з учнем та нашою системою та дозволяє вести обмін інформацією, необхідною для роботи системи.



Таким чином, ланцюг дій, необхідних для отримання запитуваних даних у чаті бота, буде наступним:

Запит у чаті > Telegram Bot API > Обробка запиту нашим сервером(Web-API) > SQL запит до бази даних > Агрегація даних сервером і передача їх до Telegram Bot API > Telegram Bot API > відображення результатів у чаті учня

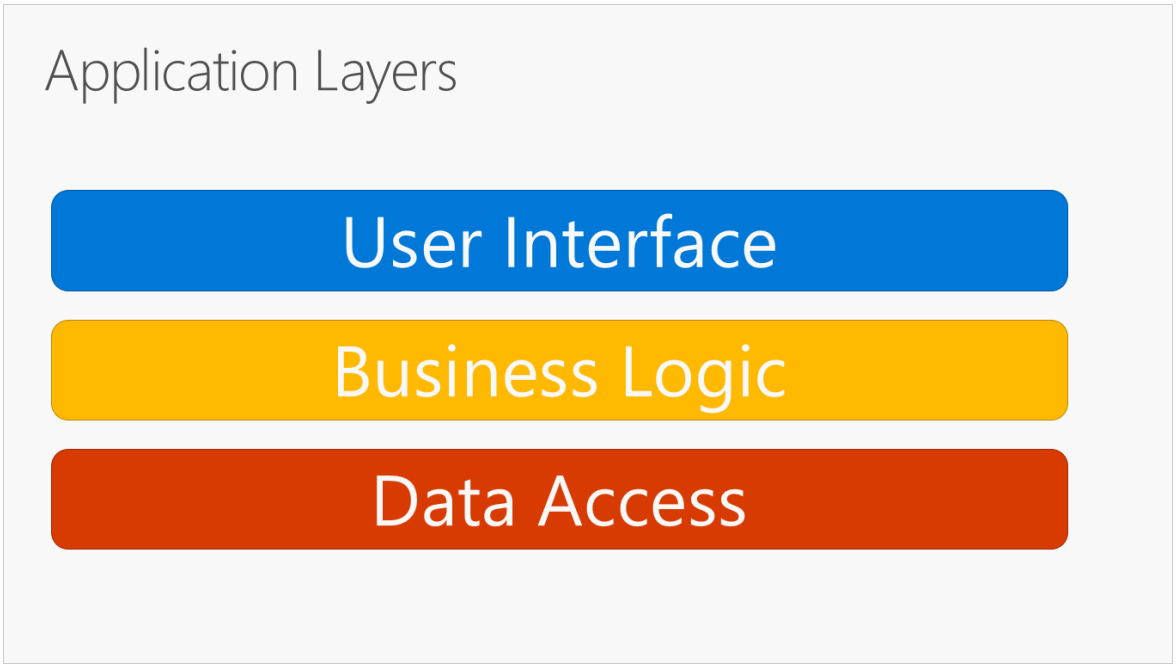
2.3. Дизайн рішення, N-рівнева архітектура

По мірі того, як додатки розростаються при розробці, основним шляхом для правильного управління файлами та залежностями є розділення цього застосунку на рівні відповідно до їх обов'язків.

Ключовим принципом цього підходу є розподіл обов'язків, що дозволяє організувати кодову базу таким чином, щоб розробник міг легко знайти, де реалізована та чи інша функція. Утім, переваги рівневої архітектури виходять далеко за межі організації коду.

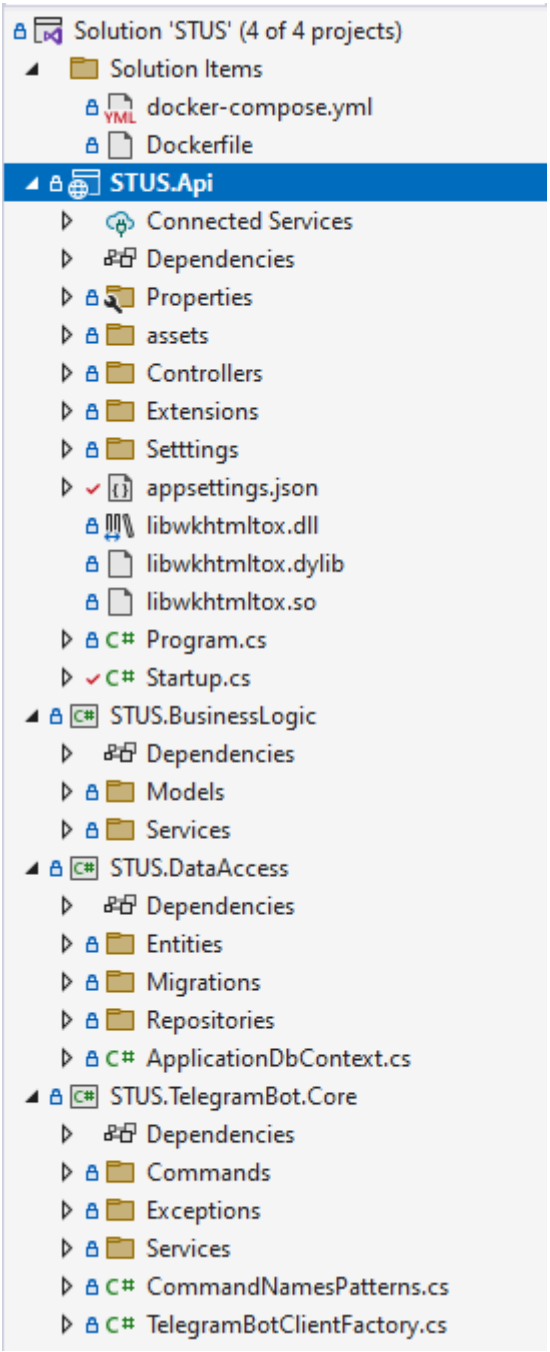
Рівнева організація коду створює можливості для повторного використання спільної функціональності та стандартизації реалізацій застосунку, що робить код більш стислим відповідно до принципу DRY (don't repeat yourself).

Для розміщення файлів класів в солюшені, а також для побудови правильних зв'язків між проектами ми обрали N-рівневу архітектуру ("N-Layer" architecture)



Ці рівні часто скорочено називають UI - рівень презентації (у нашому випадку WebApi), BLL - рівень бізнес логіки (Business Logic Layer), і DAL - рівень доступу до даних (Data Access Layer). Використовуючи цю архітектуру, користувач здійснює запити через UI рівень, який взаємодіє лише з рівнем бізнес логіки. У свою чергу, рівень бізнес логіки може викликати DAL рівень

для доступу до даних. Рівень презентації(UI) при цьому не робить ніяких запитів до DAL напряму. Тобто кожен рівень має свої власні чітко визначені функції.

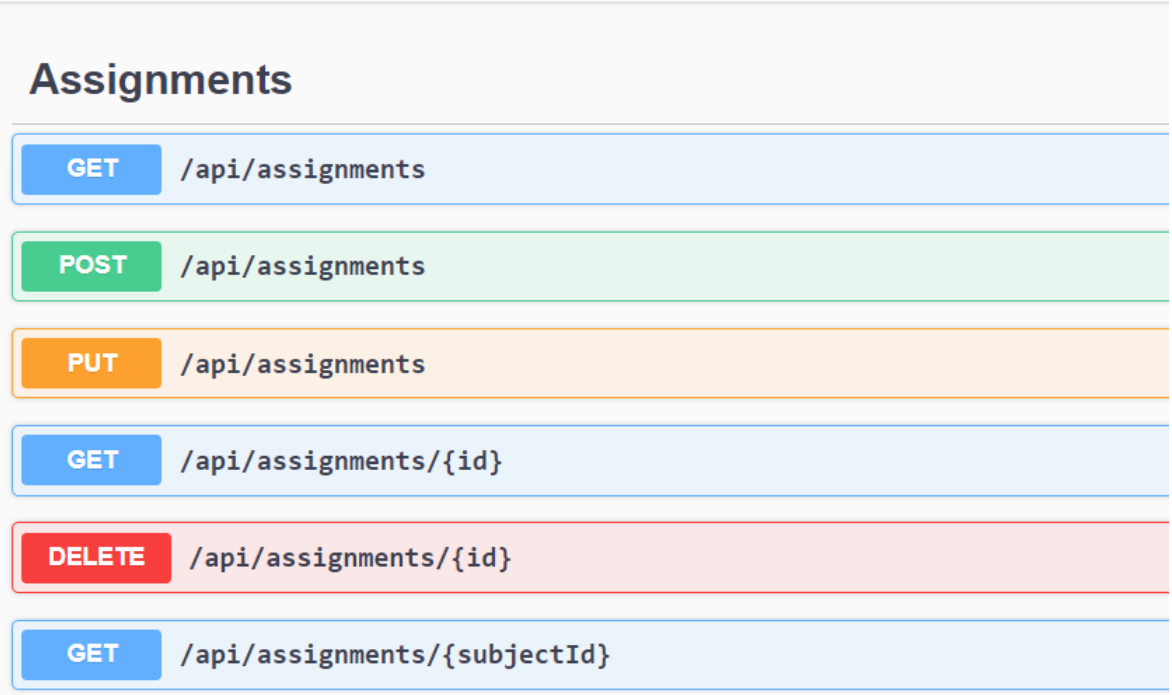


2.3.1. Контролери RESTful API

Основними точками керування даними в RESTful API є ендпоіти (кінцеві точки). Саме до них буде звертатися веб-клієнт для взаємодії з нашим бекенд сервером.

Ендпоінт являє собою URL (Uniform Resource Locator) для виконання потрібних операцій та зазвичай представляє певний ресурс або колекцію в рамках системи. Ці операції включають отримання даних, надсилання даних, оновлення інформації або ж здійснення спеціальних завдань.

На рисунку наведено приклад ендпоінта нашої системи, візуально відображеного через інтерфейс Swagger Open API.

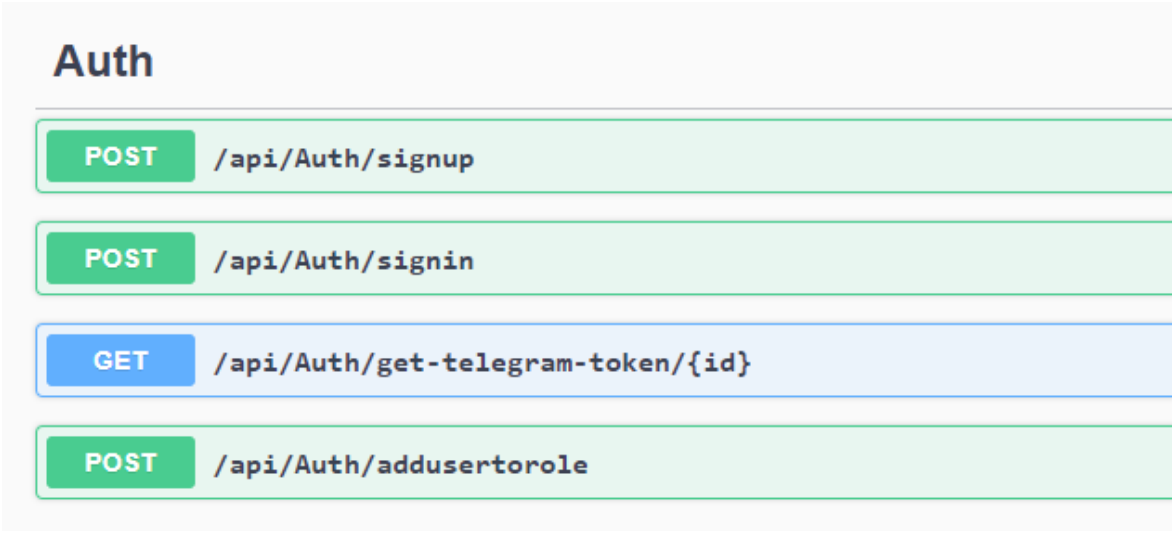


Оскільки на одну кінцеву точку, а саме на один URL припадає декілька можливих операцій, то тип цієї операції буде визначатися через метод HTTP-запиту (GET, POST, PUT, DELETE).

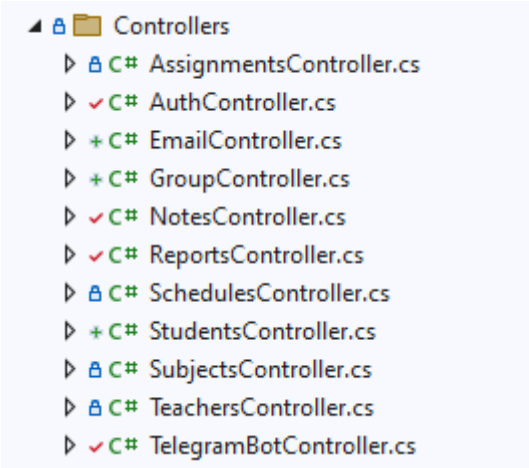
Інструментом, що відповідає за обробку цих запитів у ASP.NET WEB-API, є контролери.

```
        [Authorize(Roles = "Teacher, Administrator")]
         [Route("api/assignments")]
         [ApiController]
         public class AssignmentsController : ControllerBase
         {
             private readonly IAssignmentService _assignmentService;
             private readonly IMapper _mapper;

             public  AssignmentsController(IAssignmentService  assignmentSevice,  IMapper
mapper)
             {
                 _assignmentService = assignmentSevice;
                 _mapper = mapper;
             }


             [HttpGet]
             public async Task<IActionResult> GetAll()
             {
                 var response = await _assignmentService.GetAllAssignmentsAsync();
                 return Ok(response);
             }

             [HttpGet("{id}")]
             public IActionResult Get([FromRoute] Guid id)
             {
                 var response = _assignmentService.GetAssignmentById(id);
                 return Ok(response);
             }
             [HttpPost]
             public    async    Task<IActionResult>    Post([FromBody]    AssignmentDTO
assignmentDTO)
             {
                 var assignment = _mapper.Map<Assignment>(assignmentDTO);
                 var                  response                 =                 await
_assignmentService.AddAssignmentAsync(assignment);

                 return Ok(response);
             }

               [...]
```

Приклад ендпоінтів контролера автентифікації, відображеного через Swagger Open API.



Контролери нашого застосунку.

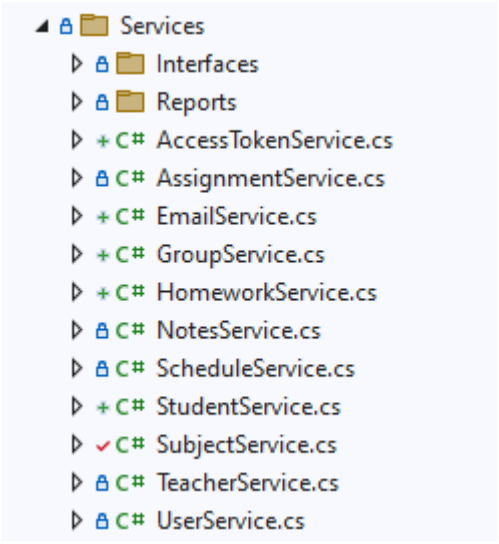


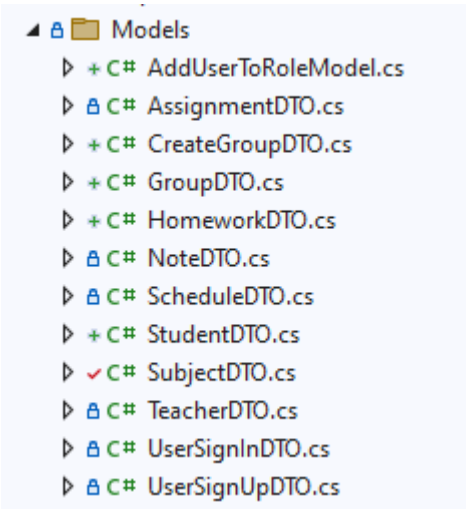
2.3.2. Бізнес логіка програми

Бізнес логіка, або ж Application Core, містить у собі бізнес-модель, що складається із сутностей, сервісів та інтерфейсів. Ці інтерфейси включають абстракції для операцій, що будуть виконані з використанням рівня доступу до даних, таких як файл DataAccess. Іноді сервіси або інтерфейси, визначені на цьому рівні, можуть працювати з типами, не зв’язаними з сутностями бази даних, що не мають залежностей на UI або DAL рівні. Тоді ці типи визначаються як прості DTOs (Data Transfer Objects).

До типів, притаманних цьому рівню, належать:

- Сутності
- Інтерфейси
- Сервіси домену
- Специфікації
- Кастомні виключення
- Доменні івенти та їх обробники (Handlers)





2.3.3. Шар доступу до даних

Шар доступу до даних включає в себе імплементації інструментів доступу до даних. В нашому проекті ці імплементації - це Entity Framework ApplicationDbContex, міграції бази даних тощо.

Типи, притаманні цьому рівню:

- Entity Framework Core типи (ApplicationDbContext, Migration);
- Типи для легкого доступу до даних (Repository);
- Сутності бази через підхід “Код спочатку”(Code first)

Дизайн-патерн Repository було обрано для того, щоб структурувати код на цьому рівні.

Шаблон Репозиторію - це шаблон проектування у межах Domain-Driven Design, призначений для виокремлення питань персистенції зі зв'язаної з доменом моделі системи. В доменній моделі визначаються один або кілька абстракцій персистенції - інтерфейсів, які мають реалізації у вигляді адаптерів, специфічних для персистенції, визначених в інших частинах програми.

Реалізації Репозиторіїв - це класи, які інкапсулюють логіку доступу до джерел даних. Вони централізують загальні функції доступу до даних, забезпечуючи кращу підтримку та зв'язування інфраструктури або технології, використовуваної для доступу до баз даних, від доменної моделі. Якщо використовується об'єктно-реляційний мапер (ORM), такий як Entity Framework, код, який потрібно реалізувати, спрощується завдяки LINQ і сильній типізації. Це дозволяє зосередитись на логіці персистенції даних, а не на роботі з доступом до даних.

Сучасні функції мови C# також дозволяють нам створювати базовий клас для патерну репозиторію у коді C# (наприклад, RepositoryBase), що може бути корисним з кількох причин:

- Забезпечення загального функціоналу: RepositoryBase може містити загальну логіку, яка буде використовуватися всіма конкретними репозиторіями. Наприклад, це може бути код для підключення до бази

даних, управління транзакціями, обробка виключень тощо. Це дозволяє уникнути дублювання коду в кожному окремому репозиторії.

- Визначення загального інтерфейсу: RepositoryBase може визначати загальний інтерфейс для всіх репозиторіїв у системі. Наприклад, це можуть бути методи для додавання, видалення, оновлення та отримання об'єктів з бази даних. Це спрощує використання репозиторіїв і дозволяє використовувати поліморфізм при роботі з ними.
- Розширення функціоналу: RepositoryBase може містити додаткові методи або властивості, які будуть спільними для багатьох репозиторіїв. Наприклад, це може бути метод для пагінації даних або метод для валідації об'єктів перед їх збереженням. Це дозволяє легко розширювати функціонал репозиторіїв за допомогою успадкування від RepositoryBase.

```
public class RepositoryBase<TEntity> : IRepositoryBase<TEntity> where TEntity : class
    {
        private readonly ApplicationDbContext _dbContext;

        public RepositoryBase(ApplicationDbContext dbContext)
        {
            this._dbContext = dbContext;
        }
        public async Task<TEntity> InsertAsync(TEntity entity)
        {
            await _dbContext.Set<TEntity>().AddAsync(entity);
            await _dbContext.SaveChangesAsync();
            return entity;
        }

        public async Task DeleteEntityAsync(TEntity entity)
        {
            _dbContext.Set<TEntity>().Remove(entity);
            await _dbContext.SaveChangesAsync();
        }

        public async Task<List<TEntity>> GetAllAsync()
        {
            return  await _dbContext.Set<TEntity>().AsNoTracking().ToListAsync();
        }
        public async Task<TEntity> UpdateAsync(TEntity entity)
        {
            _dbContext.Set<TEntity>().Update(entity);
            await _dbContext.SaveChangesAsync();
            return entity;
        }
    }
```

Розширення базового репозиторію нащадком через процес наслідування.

```
public class AssignmentRepository : RepositoryBase<Assignment>,IAssignmentRepository
    {
        private readonly ApplicationDbContext _dbContext;

        public AssignmentRepository(ApplicationDbContext dbContext) : base(dbContext)
        {
            _dbContext = dbContext;
        }

        public Assignment GetAssignmentById(Guid id)
        {
            return _dbContext.Assignments.FirstOrDefault(x => x.Id == id);
        }

        public async Task<IEnumerable<Assignment>> GetAssignmentsByPeriod(DateTime
from, DateTime to)
        {
            var result = await _dbContext.Assignments
                .Include(item => item.Subject)
                .ThenInclude(item => item.Teacher)
                .Include(item => item.Group)
                .Where(item => item.Deadline > from && item.Deadline < to)
                .OrderBy(x => x.Deadline)
                .ThenBy(x => x.Subject.Title)
                .ToListAsync();
            return result;
        }
    }
    [...]
```

2.4. DI, впровадження залежностей (Singleton, Scoped, Transient)

Dependency Injection (DI) - це концепція програмування, яка дозволяє введення залежностей об'єктів (компонентів) в інші об'єкти, без прямого створення або залежності від них. Це реалізується за допомогою зовнішнього механізму, який впроваджує залежності в об'єкт під час його створення або пізніше.

У контексті нашого проєкту Dependency Injection використовується для забезпечення легкої заміни, розширення та тестування компонентів системи. Це покращує модульність, читабельність та обслуговування коду.

Слід також підкреслити важливість налаштування lifetimes (тривалості) сервісів. Кожен сервіс (компонент), який використовується в проекті, має свою "тривалість", тобто час життя, протягом якого створюється та зберігається його

інстанс. Налаштування lifetimes важливо для правильної роботи додатку і ефективного використання ресурсів.

Існують кілька типів тривалості сервісів, таких як:

- Singleton: Інстанс сервісу створюється лише один раз під час першого використання та залишається жити протягом усього життя додатку. Це використовується для сервісів, які повинні мати лише один екземпляр та мають загальний стан для всієї системи.

Приклад

```
public void ConfigureServices(IServiceCollection services)
{
    [...]
    services.AddSingleton<TelegramBotClientFactory>();
    [...]
}
```

Також підходить для важких сервісів, новий підйом яких при кожному новому запиті спричиняв би велике навантаження на систему та уповільнював час від відгуку на запит.

Одним з таких сервісів є конвертер html та css до pdf, наведений до прикладу нижче:

```
public void ConfigureServices(IServiceCollection services)
{
    [...]
    services.AddSingleton(typeof(IConverter), new SynchronizedConverter(new PdfTools()));
        [...]
     }
```

Використання Singleton для IConverter дозволяє нам забезпечити оптимальну ефективність роботи додатку, оскільки ми не будемо створювати новий екземпляр сервісу при кожному виклику конвертації. Замість цього, ми матимемо єдиний екземпляр, який буде перевикористовуватися і мінімізуватиме витрати пам'яті та часу на створення нових об'єктів такої складності.

- Transient: Інстанс сервісу створюється при кожному виклику. Кожен запит на сервіс отримує новий екземпляр. Це підходить для сервісів, які не зберігають стану та не вимагають загальної ініціалізації.
- Scoped: Інстанс сервісу створюється один раз на кожний HTTP-запит або інший вид контексту. Це використовується для сервісів, які пов'язані з конкретним контекстом, наприклад, кожен запит клієнта до веб-сервера отримує свій власний екземпляр сервісу.

Приклад реєстрації сервісу з тривалістю Scoped:

```
public void ConfigureServices(IServiceCollection services)
{
    [...]
    services.AddScoped<IReportsService, ReportsService>();
    [...]
}
```

2.5. Автентифікація, авторизація (JWT bearer, ролі автентифікації AspNet)

У додатку реалізована система автентифікації та авторизації, що забезпечує захист доступу до функціональних можливостей та даних. Для досягнення цих цілей використовуються ролі та JWT (JSON Web Token) токени.

Ролі

У системі існують різні ролі, які надають доступ до відповідних функціональних можливостей:

1) Адміністратор: Ця роль надає повний доступ до адміністративної панелі та управління системою. Адміністратор може керувати вчителями, додавати нових вчителів та видавати їм ролі.
2) Вчитель: має доступ до всіх потрібних йому для навчального процесу інструментів, проте не може назначати ролі користувачам, як це робить адміністратор.

3) Учень: в цій системі учень обмежений функціями, доступними у чаті телеграм-боту.

JWT токени

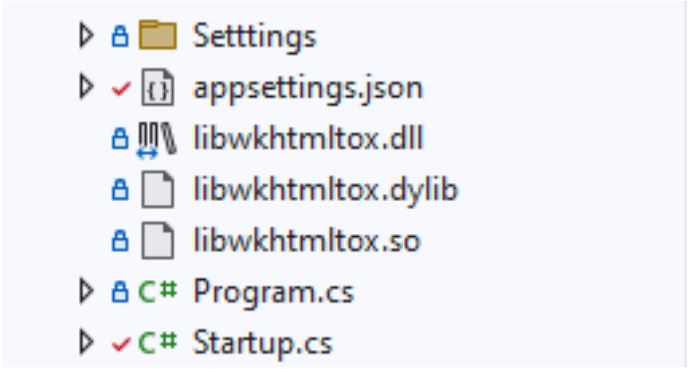
JWT токен використовується для аутентифікації та авторизації користувачів у системі. Після успішної аутентифікації користувач отримує JWT токен, який потім включається в заголовок кожного запиту, що потребує авторизації.

Система перевіряє та верифікує JWT токен при кожному запиті, що має обмежений доступ. Перевірка включає перевірку підпису токена, перевірку строку дії токена та перевірку ролі користувача.

2.6. Генерація звітів у pdf форматі

Для можливості формування звітів було обрано такий інструмент, як DinkToPDF - популярну опенсорс бібліотеку, що використовується для конвертування HTML, CSS та іншого веб контенту у PDF формат. Цей підхід є простим способом генерувати PDF-документи, використовуючи мову програмування C#.

DinkToPDF побудований на рендеринговому рушії WebKit, котрий дозволяє доволі точно рендерити HTML та CSS елементи, включаючи складні макети, зображення, шрифти і стилі, і конвертує їх у PDF документи високої якості.



libwkhtmltox.dll

libwkhtmltox.dylib

libwkhtml.so

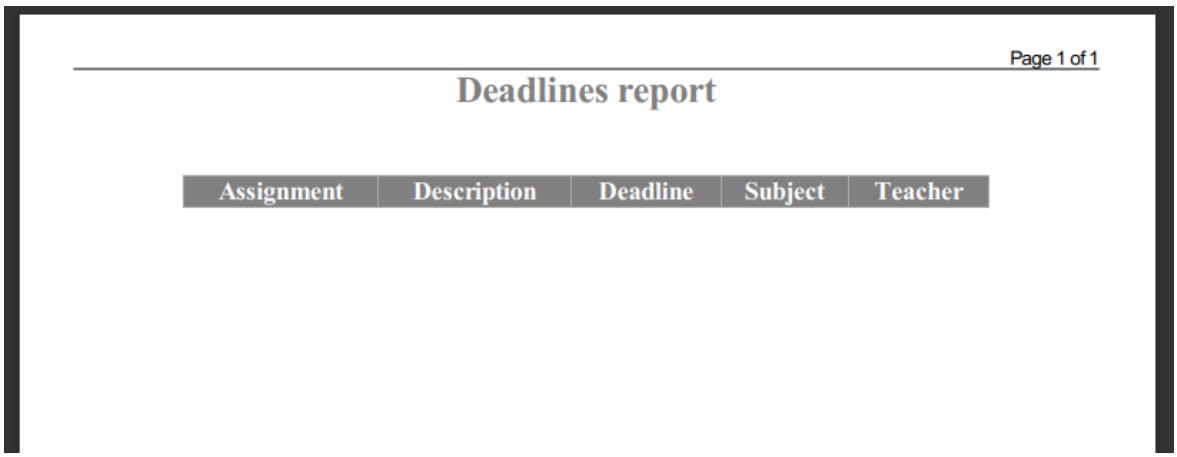
Основні файли роботи з цим інструментом.

```
private string GetHTMLreport(IEnumerable<Assignment> assigments)
{var sb = new StringBuilder();
    sb.Append(@"
        <html>
            <head></head>
            <body>
                <div class='header'><h1>Deadlines report</h1></div>
                <table align='center'>
                    <tr>
                        <th>Assignment</th>
                        <th>Description</th>
                        <th>Deadline</th>
                        <th>Subject</th>
                        <th>Teacher</th>
                    </tr>");
    foreach (var item in assigments)
    {sb.Append(
        @$"
            <tr>
                <td>{item.Title}</td>
                <td>{item.Description}</td>
                <td>{item.Deadline}</td>
                <td>{item.Subject.Title}</td>
                <td>{item?.Subject?.Teacher?.LastName ?? "----"}</td>
            </tr>");
    }
    sb.Append(@"
            </table>
         </body>
        </html>");
    return sb.ToString();
```

Та стилі, що застосовуються до розмітки, продемонстрованої вище.

```
.header{
    text-align: center;
    color: green;
    padding-bottom: 35px;
}

table{
    width:80%;
    border-collapse: collapse;
}

td, th{
    border : 1px solid gray;
    padding:15px;
    font-size:22px;
    text-align: center;

}

table th{
    background-color: green;
    color: white;
}
```

Що в результаті дає таку структуру, яка в подальшому легко заповнюється відповідними даними.



2.7. Telegram bot

Телеграм бот є автоматизованим інтерфейсом для взаємодії з користувачами у месенджері Telegram. Він надає можливість користувачам отримувати інформацію, виконувати дії та отримувати відповіді на запитання за допомогою текстових повідомлень. Додатково бот включає адмін панель, яка дозволяє адміністраторам керувати функціональністю бота через REST API.

Архітектура

Телеграм бот з адмін панеллю має наступну архітектуру:

- Telegram Bot: Ця складова відповідає за обробку вхідних повідомлень від користувачів та відправку відповідей. Вона взаємодіє з Telegram Bot API, який надає можливості зчитування та надсилання повідомлень через HTTP-запити.
- Адміністративна панель: Ця складова забезпечує можливість адміністраторам керувати функціональністю телеграм бота через REST API. Адмін панель включає набір ендпоінтів, які можуть бути викликані зовнішніми додатками для зміни налаштувань, управління користувачами, додавання нового контенту тощо.

- Бізнес логіка: Ця складова відповідає за обробку логіки додатку, включаючи обробку запитів користувачів, валідацію даних, збереження інформації тощо. Бізнес логіка включає набір методів, які можуть бути використані як з телеграм ботом, так і з адмін панеллю.

Основною структурою даних, яка використовується при спілкуванні з Telegram Bot API, є update. Update - це об'єкт, який представляє вхідну подію або повідомлення, яке бот отримує від користувача чи від Telegram-сервера.

Коли користувач надсилає повідомлення боту, Telegram надсилає це повідомлення у вигляді update до бота. Update містить інформацію про тип події (повідомлення, запит на клавіатуру, запит на зображення тощо) та відповідні дані, наприклад, текст повідомлення, ідентифікатор користувача тощо.

Для правильної роботи бота важливо правильно обробляти вищезгадане Update повідомлення. У нашій системі обробка відбувається в у класі TelegramBotController.cs у HTTP post методі Update. Там визначається тип команди для виконання або ж прогрес минулої команди.

Командою в нашому контексті є клас, що наслідується від базового абстрактного класу BaseCommand. Використання команд, що успадковуються від базового класу BaseCommand, має декілька переваг:

- Єдинообразність: Кожна команда використовує спільний інтерфейс та шаблон класу, що спрощує розробку та управління командами в боті. Всі команди можуть бути оброблені через одну зручну точку входу.
- Розширюваність: Додавання нових команд стає простішим завдяки успадкуванню від базового класу. Розробники можуть легко створювати нові класи-команди та визначати їхню унікальну функціональність, не змінюючи загальну структуру бота.
- Модульність: Кожна команда може мати свою власну логіку в методі ExecuteAsync, що дозволяє зосередитись на конкретній функціональності команди без впливу на інші частини бота.
- Швидкість розробки: Використання базового класу дозволяє використовувати вже наявні методи бізнес-логіки, які були написані

раніше для інших частин бота. Це дозволяє прискорити розробку нових команд та спрощує підтримку та модифікацію системи.

Сам базовий клас

```
public abstract class BaseCommand
{
    public abstract string Name { get; }
    public abstract Task ExecuteAsync(Update update);
}
```

Приклад класу, наслідуваного від BaseCommand

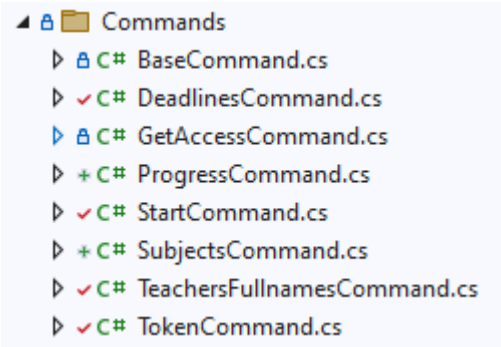
```
public class DeadlinesCommand : BaseCommand
    {
        private readonly ITelegramUserService _userService;
        private readonly IReportsService _reportsService;
        private readonly TelegramBotClient _botClient;

        public DeadlinesCommand(ITelegramUserService userService,IReportsService reportsService, STUS.Core.TelegramBotClientFactory telegramBot)
        {
            _userService = userService;
            _reportsService = reportsService;
            _botClient = telegramBot.GetBot().Result;
        }
        public override string Name => "Deadlines";

        public override async Task ExecuteAsync(Update update)
        {
            var user = await _userService.GetOrCreate(update);
            if (user.UserId == null)
            {
                throw new TelegramException(_botClient, user.ChatId, "Не авторизований юзер." +" Для авторизації введіть токен отриманий від адміністатора у форматі" +
                    " \"/token [email] [token]");
            }
            var from = new DateTime();
            var to = DateTime.UtcNow + new TimeSpan(14, 0, 0, 0);
```

```
                var deadlinesBytes = await
_reportsService.GetDeadlinesReportAsync(from, to);

                Stream deadlinesFile = new MemoryStream(deadlinesBytes);
                InputOnlineFile file = new InputOnlineFile(deadlinesFile,
"deadlines.pdf");

                await _botClient.SendDocumentAsync(user.ChatId, file);
            }
        }
```

Його виклик у контролері.

```
    [HttpPost("update")]
            public async Task<IActionResult> Update([FromBody] object upd)
            {
                try
                {
                [...]
                case "Мої дедлайни":
                      await _deadlines.ExecuteAsync(upd);
                      break;
                [...]
```

Та список інших команд, що забезпечують роботу цього бота.



ВИСНОВКИ ДО РОЗДІЛУ 2

В результаті роботи було розроблено додаток, який допомагає забезпечити ефективну координацію роботи освітніх курсів з використанням інтеграції з Telegram. Впроваджені принципи роботи, архітектура та функціональність

дозволяють забезпечити зручну та ефективну взаємодію учасників курсу та полегшують організацію навчального процесу.

1) Розроблено архітектуру програмного рішення, яка базується на N-рівневій архітектурі, що дозволяє логічно розділити компоненти системи і спрощує розробку, тестування та підтримку додатку. Ця архітектура включає контролери RESTful API, бізнес-логіку програми та шар доступу до даних. Такий підхід дозволяє легко розширювати функціональність системи та забезпечує її гнучкість та швидкість реагування.
2) Впровадження залежностей (Dependency Injection) із використанням шаблонів Singleton, Scoped та Transient дозволяє зменшити залежність між компонентами системи, полегшує тестування та роботу з кодом.
3) Для забезпечення безпеки та контролю доступу до системи була використана автентифікація та авторизація за допомогою JWT bearer і ролей автентифікації AspNet. Це дозволяє забезпечити доступ до функцій та ресурсів системи лише авторизованим користувачам.
4) Реалізовано можливість генерації звітів у форматі PDF. Це дозволяє студентам та викладачам зручно отримувати звіти про прогрес, успішність та іншу інформацію, пов'язану з освітніми курсами.
5) Реалізовано інтеграцію з Telegram за допомогою Telegram bot. Це дозволяє забезпечити зручну комунікацію між учасниками курсу та можливість отримувати сповіщення про нові матеріали, запитання та інші повідомлення.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено додаток, який допомагає забезпечити ефективну координацію роботи освітніх курсів з використанням інтеграції з Telegram. Цей додаток відкриває перед учасниками навчального процесу широкі можливості для зручного і швидкого обміну інформацією, покращує організацію навчання та сприяє активному спілкуванню між викладачами та студентами.

Завдяки функціоналу додатка викладачі можуть зручно створювати розклад занять, надсилати завдання та матеріали для вивчення, а також вести моніторинг прогресу студентів. Студенти у свою чергу можуть отримувати актуальну інформацію про курс, виконувати завдання та спілкуватися з викладачами через зручний інтерфейс Telegram.

Застосування Telegram в якості каналу комунікації має багато переваг, таких як широке поширення месенджера серед користувачів, зручність використання та можливість доступу з різних платформ. Це робить додаток привабливим для використання в освітніх установах, дистанційних навчальних програмах або навіть в корпоративному навчанні.

Використання засобів розробки .NET виявилося надзвичайно корисним при створенні додатку. Широкий набір інструментів, бібліотек і фреймворків, які полегшують розробку різних компонентів, дозволив зосередитися на функціональності додатку, збільшивши швидкість розробки, а потужний синтаксис мови програмування C# і вбудована підтримка асинхронного програмування значно підвищила його продуктивність.

У перспективі додаток може бути розширений новими функціональними можливостями або ж навіть стати основою для реалізації повноцінної освітньої платформи. Перспективи розширення додатку включають можливість створення інтерактивних навчальних курсів з використанням різних медіа-ресурсів, функціонал для відстеження академічного прогресу студентів, системи оцінювання та звітування, спільну роботу над проектами та багато іншого.

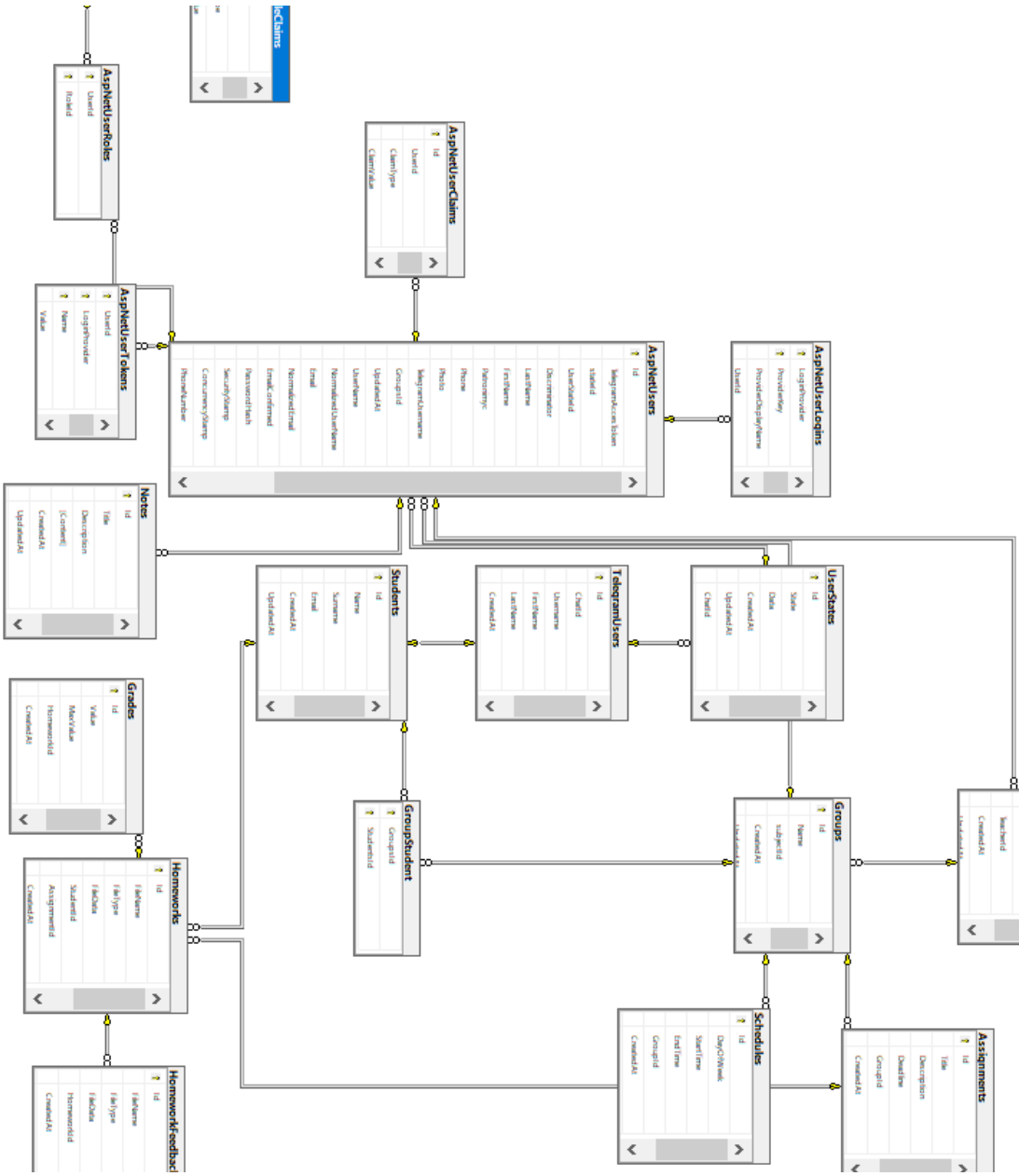
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Authentication. API Documentation & Design Tools for Teams | Swagger. URL: https://swagger.io/docs/specification/authentication/ (date of access: 08.06.2023).
2. Castro P., Melnik S., Adya A. ADO.NET entity framework. the 2007 ACM SIGMOD international conference, Beijing, China, 11–14 June 2007. New York, New York, USA, 2007. URL: https://doi.org/10.1145/1247480.1247609 (date of access: 08.06.2023)
3. Code First vs Database First vs Model First - EntityFramework Approaches Explained - .NET Core Tutorials. .NET Core Tutorials. URL: https://dotnetcoretutorials.com/code-first-vs-database-first-vs- model-first-entityframework-approaches-explained/ (date of access: 08.06.2023)
4. Common Language Runtime (CLR) in C# - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/common- language-runtime-clr-in-c-sharp/ (date of access: 08.06.2023)
5. Hejlsberg, A., Torgersen, M., Wiltamuth, S., & Golde, P. The C# programming language / A. Hejlsberg et al. Pearson Education, 2008.
6. Dependency injection in ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-7.0 (date of access: 08.06.2023)
7. Lerman J. Programming Entity Framework. O'Reilly Media, Incorporated, 2009.
8. .NET (and .NET Core) - introduction and overview. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/core/introduction (date of access: 08.06.2023).
9. Pautasso C., Wilde E. RESTful web services. the 19th international conference, Raleigh, North Carolina, USA, 26–30 April 2010. New York, New York, USA, 2010. URL: https://doi.org/10.1145/1772690.1772929 (date of access: 08.06.2023).

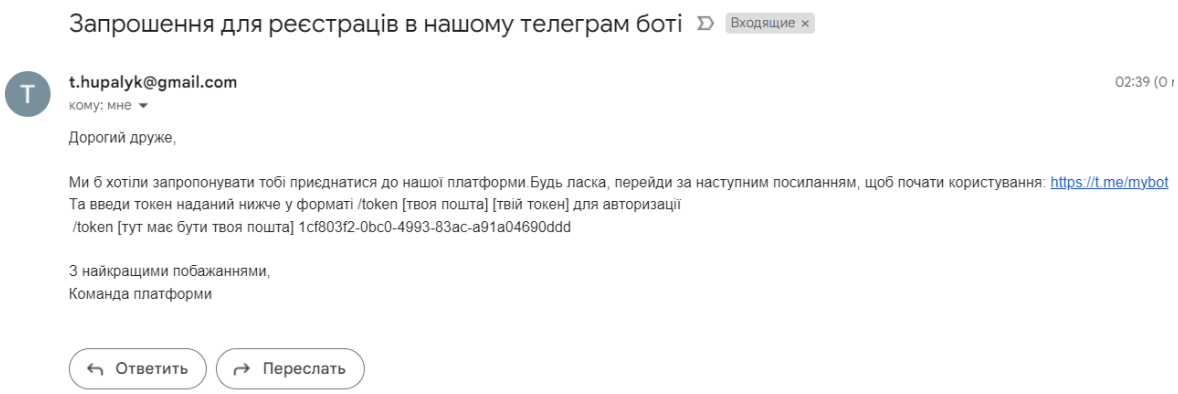
10. Subramanian H., Raj P. Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs. Packt Publishing, 2019. 378 p.
11. SQL Server technical documentation - SQL Server. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (date of access: 08.06.2023)
12. Three Tier Architecture In ASP.NET Core 6 Web API. C# Corner - Community of Software and Data Developers. URL: https://www.c-sharpcorner.com/article/three-tier-architecture-in-asp-net-core-6-web-api/ (date of access: 08.06.2023).
13. What Is Azure Data Studio? Overview, Installation & Advantages. Intellipaat Blog. URL: https://intellipaat.com/blog/azure-data-studio/?US (date of access: 08.06.2023)
14. What is Postman? Postman API Platform. Postman API Platform. URL: https://www.postman.com/product/what-is-postman/ (date of access: 08.06.2023).

ДОДАТКИ

Додаток А: Схема сутностей бази даних



Додаток Б: Робота користувача з додатком



Запрошення для реєстрацій в нашому телеграм боті Входящие ×

t.hupalyk@gmail.com 02:39 (0

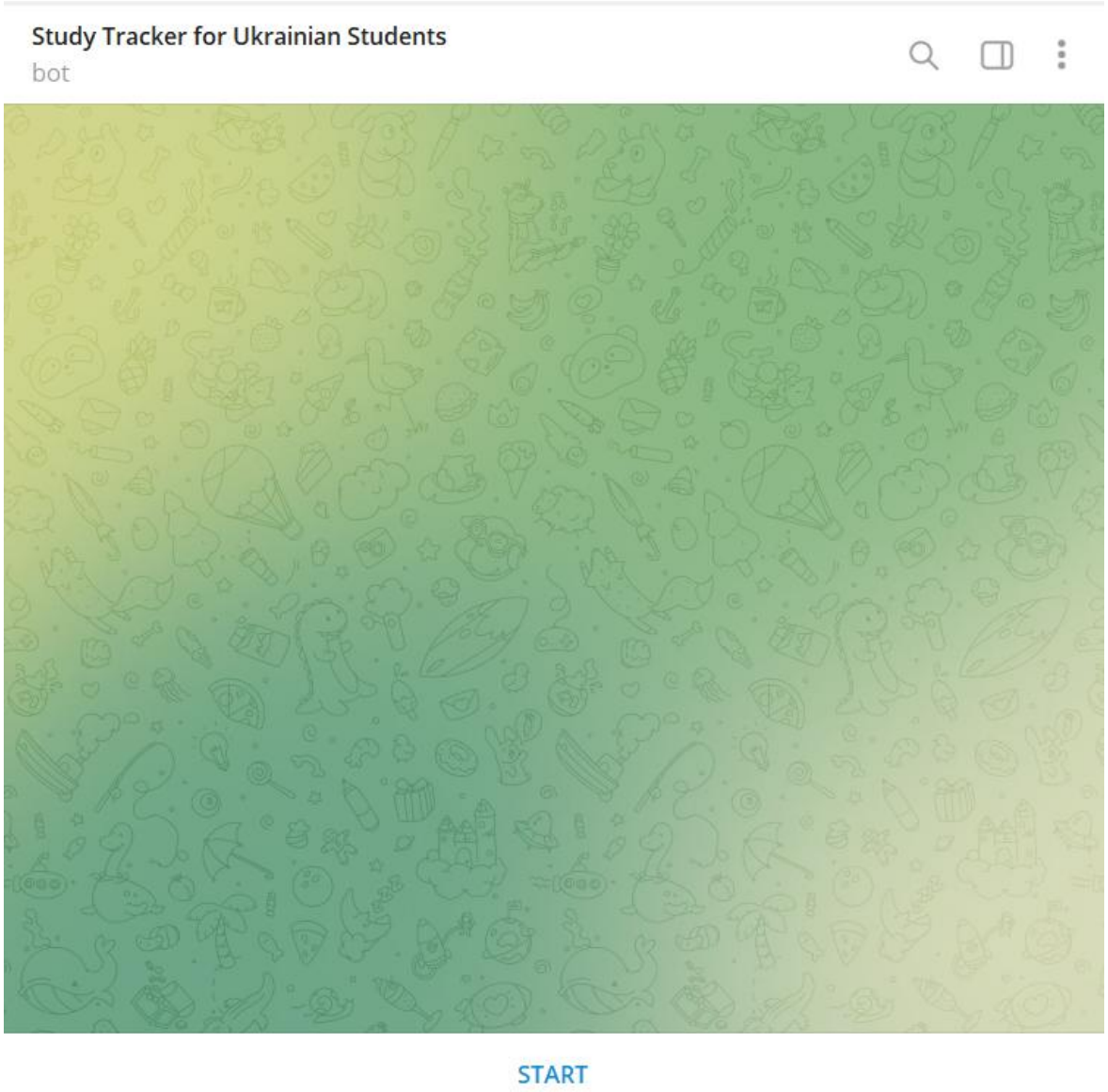
кому: мне

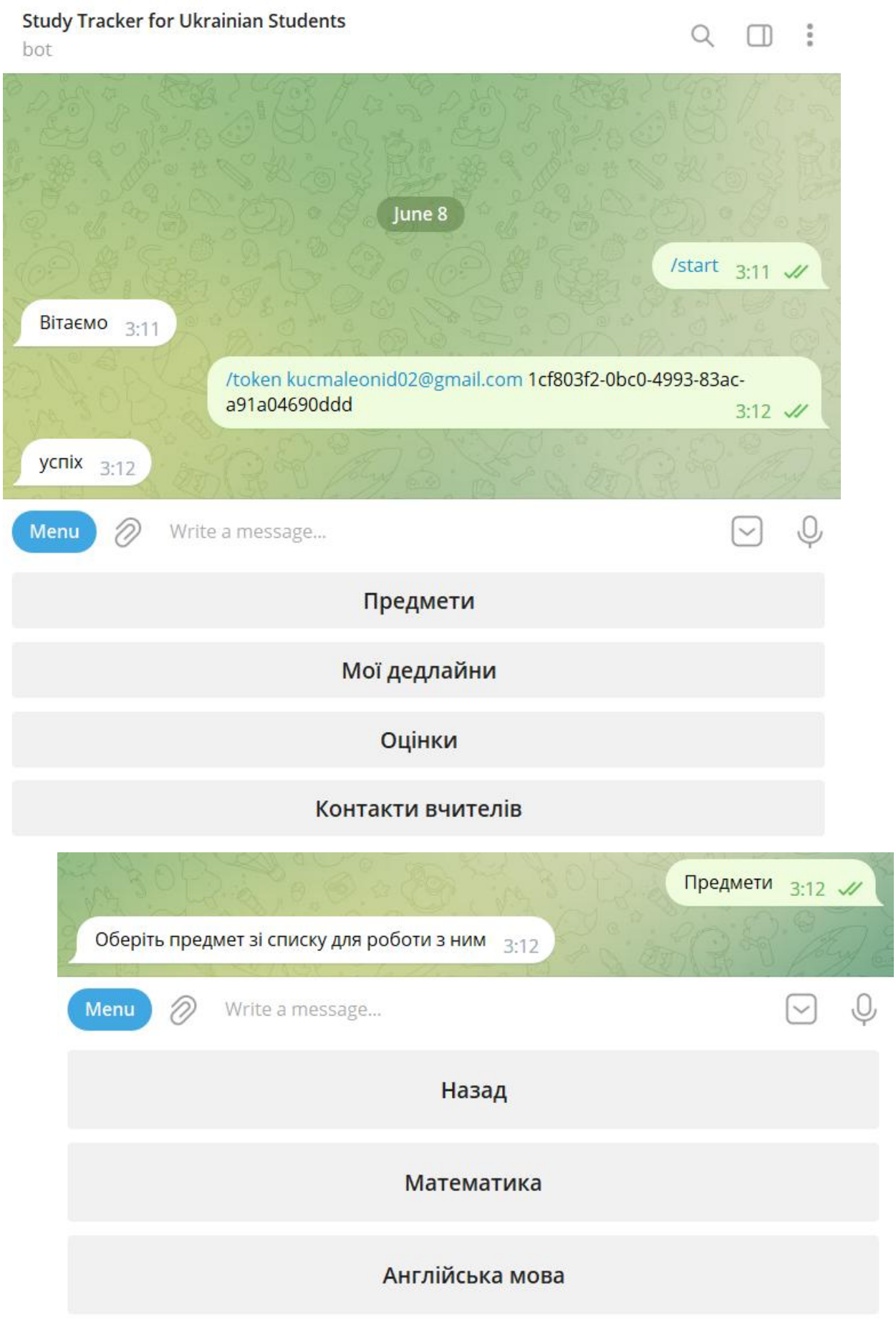
Дорогий друже,

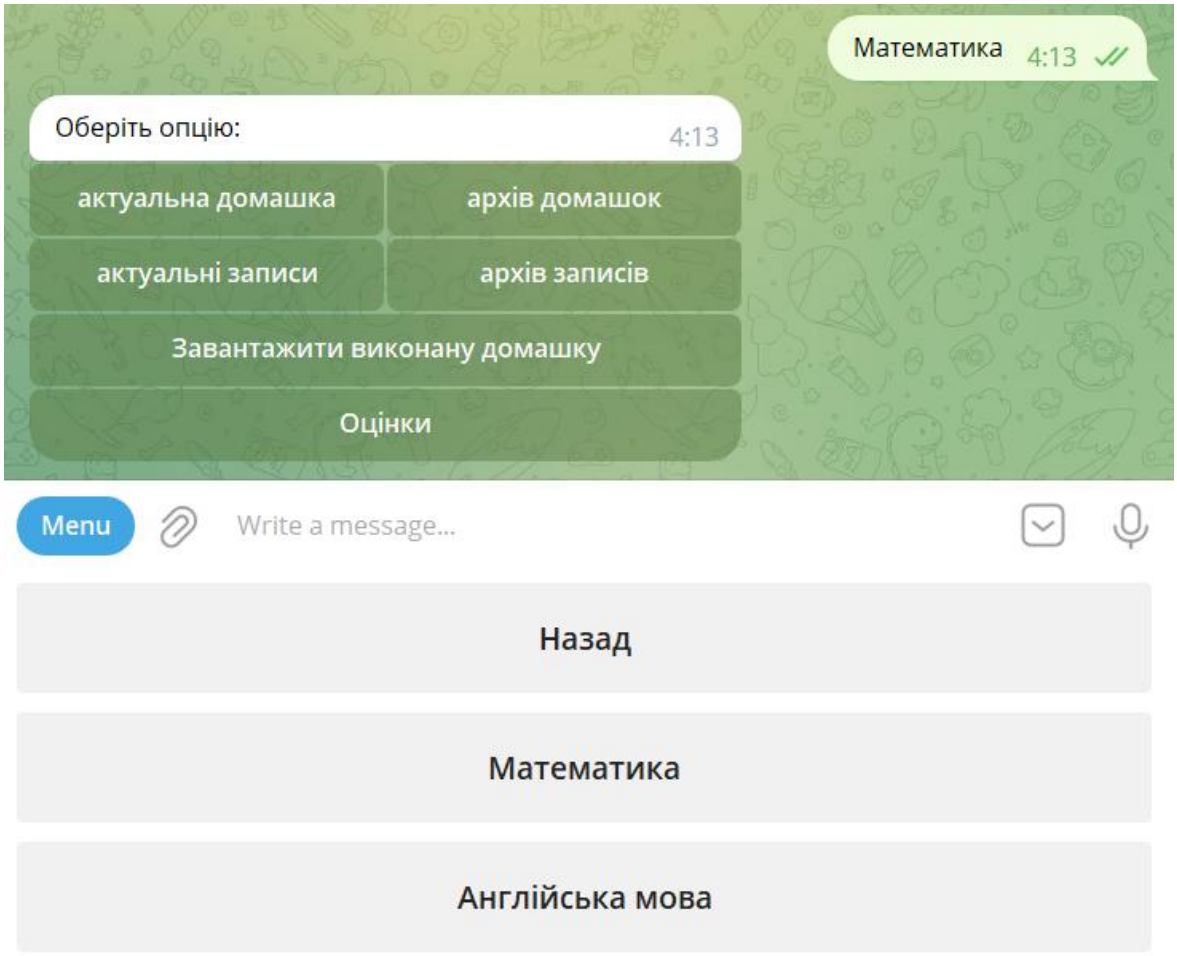
Ми б хотіли запропонувати тобі приєднатися до нашої платформи.Будь ласка, перейди за наступним посиланням, щоб почати користування: https://t.me/mybot
Та введи токен наданий нижче у форматі /token [твоя пошта] [твій токен] для авторизації
/token [тут має бути твоя пошта] 1cf803f2-0bc0-4993-83ac-a91a04690ddd

З найкращими побажаннями,
Команда платформи

Ответить Переслать


Study Tracker for Ukrainian Students
bot
START


Study Tracker for Ukrainian Students
bot
June 8
/start 3:11
Вітаємо 3:11
/token kucmaleonid02@gmail.com 1cf803f2-0bc0-4993-83ac-a91a04690ddd 3:12
успіх 3:12
Menu
Write a message...
Предмети
Мої дедлайни
Оцінки
Контакти вчителів
Предмети 3:12
Оберіть предмет зі списку для роботи з ним 3:12
Menu
Write a message...
Назад
Математика
Англійська мова


Математика 4:13
Оберіть опцію: 4:13
актуальна домашка
архів домашок
актуальні записи
архів записів
Завантажити виконану домашку
Оцінки
Menu
Write a message...
Назад
Математика
Англійська мова

Додаток В: Код застосунку

Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src

COPY ["STUS.Api/STUS.Api.csproj", "STUS.Api/"]
COPY ["STUS.BusinessLogic/STUS.BusinessLogic.csproj", "STUS.BusinessLogic/"]
COPY ["STUS.DataAccess/STUS.DataAccess.csproj", "STUS.DataAccess/"]

RUN dotnet restore "STUS.Api/STUS.Api.csproj"

COPY . .
WORKDIR /src/STUS.Api
RUN dotnet build "STUS.Api.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "STUS.Api.csproj" -c Release -o /app/publish

 COPY ["STUS.Api/libwkhtmltox.dll", "app"]
 COPY ["STUS.Api/libwkhtmltox.dylib", "app"]
 COPY ["STUS.Api/libwkhtmltox.so", "app"]

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS final
WORKDIR /app
COPY ["STUS.Api/assets", "/app/assets"]
COPY --from=publish /app/publish .
ENTRYPOINT [ "dotnet", "STUS.Api.dll" ]
```

docker-compose.yml

```
version: '3.4'
```

```
services:

  db:
    image: mcr.microsoft.com/mssql/server
    environment:
      SA_PASSWORD: "MyPassword123"
      ACCEPT_EULA: "Y"
    ports:
      - "1433:1433"

  stusapi:
    image: stusapi
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 443:80
    depends_on:
      - db
```

Startup.cs

```
using DinkToPdf;
using DinkToPdf.Contracts;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Cors.Infrastructure;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.OpenApi.Models;
using STUS.Api.Extensions;
using STUS.Api.Setttings;
using STUS.BusinessLogic.Interfaces;
using STUS.BusinessLogic.Models;
using STUS.BusinessLogic.Services;
using STUS.BusinessLogic.Services.Interfaces;
using STUS.BusinessLogic.Services.Reports;
using STUS.Core;
using STUS.DataAccess;
using STUS.DataAccess.Entities;
using STUS.DataAccess.Interfaces;
using STUS.DataAccess.Repositories;
using STUS.DataAccess.Repositories.Interfaces;
using STUS.TelegramBot.Core.Commands;
using STUS.TelegramBot.Core.Services;
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Net;
using System.Net.Mail;
using System.Threading.Tasks;

namespace STUS.Api
{
    public class Startup
```

```
{
    private const string CorsPolicy = "CorsPolicy";

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {

        services.AddDbContext<ApplicationDbContext>(builder =>
        {

builder.UseSqlServer(Configuration.GetConnectionString("StudyTrackerDb"));
        });

        services.AddIdentity<User, Role>(option => option.Lockout.MaxFailedAccessAttempts = 5)
            .AddEntityFrameworkStores<ApplicationDbContext>()
            .AddDefaultTokenProviders();

        services.Configure<JwtSettings>(Configuration.GetSection("Jwt"));
        var jwtSettings = Configuration.GetSection("jwt").Get<JwtSettings>();

        services.AddCors(options =>
        {
```

```
            options.AddPolicy(CorsPolicy,
                builder =>
builder.WithOrigins(Configuration.GetSection("AllowedOrigins").Get<List<string>
>().ToArray())
                    .AllowAnyMethod()
                    .AllowAnyHeader());
        });

        services.AddAutoMapper(
            cfg =>
            {
                cfg.CreateMap<Subject, SubjectDTO>().ReverseMap();
                cfg.CreateMap<Note, NoteDTO>().ReverseMap();
                cfg.CreateMap<Teacher, TeacherDTO>().ReverseMap();
                cfg.CreateMap<Assignment, AssignmentDTO>().ReverseMap();
                cfg.CreateMap<Schedule, ScheduleDTO>().ReverseMap();
            });

        services.AddSingleton(typeof(IConverter), new
SynchronizedConverter(new PdfTools()));

        var emailSettings = Configuration.GetSection("EmailSettings");

        services.AddSingleton<SmtpClient>(s =>
        {
            var smtpClient = new SmtpClient(emailSettings["SmtpServer"],
int.Parse(emailSettings["SmtpPort"]))
            {
                UseDefaultCredentials = false,
                Credentials = new NetworkCredential(emailSettings["Username"],
emailSettings["Password"]),
```

```
                EnableSsl = bool.Parse(emailSettings["UseSsl"])
            };

            return smtpClient;
        });

        services.AddScoped<IUserRepository, UserRepository>();
        services.AddScoped<ITeacherRepository, TeacherRepository>();
        services.AddScoped<ISubjectsRepository, SubjectsRepository>();
        services.AddScoped<INoteRepository, NoteRepository>();
        services.AddScoped<IAssignmentRepository, AssignmentRepository>();
        services.AddScoped<IScheduleRepository, ScheduleRepository>();
        services.AddScoped<ITelegramUserRepository,
TelegramUserRepository>();
        services.AddScoped<IStudentRepository, StudentRepository>();
        services.AddScoped<IGroupRepository, GroupRepository>();

        services.AddScoped<IBotAccessTokenService,
BotAccessTokenService>();
        services.AddScoped<ITelegramUserService, TelegramUserService>();
        services.AddScoped<IUserService, UserService>();
        services.AddScoped<ITeacherService, TeacherService>();
        services.AddScoped<IReportsService, ReportsService>();
        services.AddScoped<ISubjectService, SubjectService>();
        services.AddScoped<INotesService, NotesService>();
        services.AddScoped<IAssignmentService, AssignmentService>();
        services.AddScoped<IScheduleService, ScheduleService>();
        services.AddScoped<IEmailService, EmailService>();
        services.AddScoped<IAccessTokenService, AccessTokenService>();
        services.AddScoped<IStudentService, StudentService>();
        services.AddScoped<IGroupService, GroupService>();
```

```
            services.AddSingleton<TelegramBotClientFactory>();
            services.AddScoped<StartCommand>();
            services.AddScoped<DeadlinesCommand>();
            services.AddScoped<TeachersFullnamesCommand>();
            services.AddScoped<SubjectsCommand>();
            services.AddScoped<TokenCommand>();


            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
               c.SwaggerDoc("v1", new OpenApiInfo { Title = "STUS.API",
Version = "v1" });
               c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme()
               {
                  Description = "Jwt containing userId claim",
                  Name = "Authorization",
                  In = ParameterLocation.Header,
                  Type = SecuritySchemeType.ApiKey
               });
               var security =
               new OpenApiSecurityRequirement
               {
                  {
                     new OpenApiSecurityScheme()
                     {
                        Reference = new OpenApiReference
                        {
                           Id ="Bearer",
                           Type =ReferenceType.SecurityScheme
```

```
                    },
                    UnresolvedReference = true
                },
                new List<string>()
            }
        };
        c.AddSecurityRequirement(security);
    });

    services.AddAuth(jwtSettings);
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, IServiceProvider serviceProvider)
{
    // if (env.IsDevelopment())
    // {
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "STUS.Api v1"));
    //}

serviceProvider.GetRequiredService<TelegramBotClientFactory>().GetBot().Wait();


    app.UseCors(CorsPolicy);
    app.UseHttpsRedirection();
```

```
            app.UseRouting();

            app.UseAuth();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
```

AuthExtension.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using STUS.Api.Setttings;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace STUS.Api.Extensions
{
    public static class AuthExtension
    {
        public static IServiceCollection AddAuth(
            this IServiceCollection services,
            JwtSettings jwtSettings)
        {
            services
                .AddAuthorization()
```

```
            .AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            })
            .AddJwtBearer(options =>
            {
                options.RequireHttpsMetadata = false;
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidIssuer = jwtSettings.Issuer,
                    ValidAudience = jwtSettings.Issuer,
                    IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Secret)),
                    ClockSkew = TimeSpan.Zero
                };
            });

        return services;
    }

    public static IApplicationBuilder UseAuth(this IApplicationBuilder app)
    {
        app.UseAuthentication();
        app.UseAuthorization();
```

```
            return app;
        }
    }
}
```

AuthController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using STUS.Api.Setttings;
using STUS.BusinessLogic.Models;
using STUS.DataAccess.Entities;
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Data;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace STUS.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
```

```
{
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<Role> _roleManager;
    private readonly IBotAccessTokenService _accessTokenService;
    private readonly JwtSettings _jwtSettings;

    public AuthController(
        UserManager<User> userManager,
        RoleManager<Role> roleManager,
        IOptionsSnapshot<JwtSettings> jwtSettings,
        IBotAccessTokenService accessTokenService)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _accessTokenService = accessTokenService;
        _jwtSettings = jwtSettings.Value;
    }


    [HttpPost("signup")]
    public async Task<IActionResult> SignUp([FromBody] UserSignUpDTO
userDTO)
    {
        var user = new User
        {
            UserName = userDTO.Username,
            Email = userDTO.Email
        };
        var userCreatedResult = await _userManager.CreateAsync(user,
userDTO.Password);
        if (userCreatedResult.Succeeded)
```

```
            {
                return Ok(200);
            }
            return Problem(userCreatedResult.Errors.First().Description, null, 500);
        }


        [HttpPost("signin")]
        public async Task<IActionResult> SignIn([FromBody] UserSignInDTO userDTO)
        {
            var user = _userManager.Users.SingleOrDefault(x => x.Email == userDTO.Email);
            if (user == null)
            {
                return NotFound("user not found");
            }

            var userSingInResult = await _userManager.CheckPasswordAsync(user, userDTO.Password);
            if (userSingInResult == true)
            {
                var roles = await _userManager.GetRolesAsync(user);
                return Ok(GenerateJwt(user, roles));
            }

            return BadRequest("password or password incorrect");
        }


        [HttpGet("get-telegram-token/{id}")]
```

```
public async Task<IActionResult> GetTgToken([FromRoute] Guid id)
{
    try
    {
        var result = await _accessTokenService.GenerateTokenAsync(id);
        return Ok(result);
    }
    catch (Exception)
    {

        return BadRequest();
    }
}



private string GenerateJwt(User user, IList<string> roles)
{
    var claims = new List<Claim>
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
        new Claim(ClaimTypes.Name, user.UserName),
        new                                    Claim(JwtRegisteredClaimNames.Jti,
Guid.NewGuid().ToString()),
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
    };

    var roleClaims = roles.Select(r => new Claim(ClaimTypes.Role, r));
    claims.AddRange(roleClaims);
```

```
            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.Secret));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var expires = DateTime.Now.AddDays(Convert.ToDouble(_jwtSettings.ExpirationInDays));

            var token = new JwtSecurityToken(
                issuer: _jwtSettings.Issuer,
                audience: _jwtSettings.Issuer,
                claims,
                expires: expires,
                signingCredentials: creds
                );

            return new JwtSecurityTokenHandler().WriteToken(token);
        }

        [HttpPost("addusertorole")]
        [Authorize(Roles = "Administrator")]
        public async Task<IActionResult> AddUserToRole(AddUserToRoleModel model)
        {
            var user = await _userManager.FindByEmailAsync(model.Email);
            if (user == null)
            {
                return NotFound("User not found");
            }

            var roleExists = await _roleManager.RoleExistsAsync(model.RoleName);
```

```
            if (!roleExists)
            {
                return NotFound("Role not found");
            }

            var result = await _userManager.AddToRoleAsync(user,
model.RoleName);

            if (result.Succeeded)
            {
                return Ok();
            }

            return BadRequest(result.Errors);
        }
    }
}
```

EmailController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;
using System;
using STUS.BusinessLogic.Services;

namespace STUS.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class EmailController : ControllerBase
```

```
    {
        private readonly IEmailService _emailService;

        public EmailController(IEmailService emailService)
        {
            _emailService = emailService;
        }


        [HttpPost("send-invitation")]
        public async Task<IActionResult> SendInvitationAsync(List<string> userIds)
        {
            if (userIds == null || userIds.Count == 0)
            {
                return BadRequest("Не надано список id користувачів.");
            }

            try
            {
                await _emailService.SendInvitationAsync(userIds);
                return Ok(200);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"An error occurred while sending the invitations: {ex.Message}");
            }
        }
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.EntityFrameworkCore;
```

```
using STUS.BusinessLogic.Interfaces;
using STUS.BusinessLogic.Models;
using STUS.DataAccess;
using System;
using System.Collections.Generic;
using System.Linq;

namespace STUS.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class GroupController : ControllerBase
    {
        private readonly IGroupService _groupService;
        private readonly ApplicationDbContext _context;

        public GroupController(IGroupService groupService,
ApplicationDbContext context)
        {
            _groupService = groupService;
            _context = context;
        }

        [HttpGet("{id}")]
        public ActionResult<GroupDTO> GetGroupById(Guid id)
        {
            GroupDTO groupDto = _groupService.GetGroupById(id);

            if (groupDto == null)
            {
                return NotFound();
```

```
    }

    return Ok(groupDto);
}

[HttpGet]
public ActionResult<IEnumerable<GroupDTO>> GetAllGroups()
{
    IEnumerable<GroupDTO> groupDtos = _groupService.GetAllGroups();

    return Ok(groupDtos);
}

[HttpPost]
public ActionResult CreateGroup(CreateGroupDTO groupDto)
{
    _groupService.CreateGroup(groupDto);

    return Ok();
}

[HttpPut("{id}")]
public ActionResult UpdateGroup(Guid id, GroupDTO groupDto)
{
    if (id != groupDto.Id)
    {
        return BadRequest();
    }

    _groupService.UpdateGroup(groupDto);
```

```
            return Ok();
        }

        [HttpDelete("{id}")]
        public ActionResult DeleteGroup(Guid id)
        {
            _groupService.DeleteGroup(id);

            return Ok();
        }

        // POST api/groups/{groupId}/students/{studentId}
        [HttpPost("{groupId}/students/{studentId}")]
        public IActionResult AddStudentToGroup(Guid groupId,  Guid studentId)
        {
            var group = _context.Groups.Include(g => g.Students).FirstOrDefault(g
=> g.Id == groupId);
            if (group == null)
                return NotFound();

            var student = _context.Students.FirstOrDefault(s => s.Id == studentId);
            if (student == null)
                return NotFound();

            group.Students.Add(student);
            _context.SaveChanges();
            return NoContent();
        }

        // DELETE api/groups/{groupId}/students/{studentId}
        [HttpDelete("{groupId}/students/{studentId}")]
```

```
        public IActionResult RemoveStudentFromGroup(Guid groupId, Guid studentId)
        {
            var group = _context.Groups.Include(g => g.Students).FirstOrDefault(g => g.Id == groupId);
            if (group == null)
                return NotFound();

            var student = group.Students.FirstOrDefault(s => s.Id == studentId);
            if (student == null)
                return NotFound();

            group.Students.Remove(student);
            _context.SaveChanges();
            return NoContent();
        }
    }
}
```

ReportsController.cs

```
using Microsoft.AspNetCore.Mvc;
using STUS.BusinessLogic.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace STUS.Api.Controllers
{
    [Route("api/reports")]
    [ApiController]
    public class ReportsController : ControllerBase
    {
```

```
        private readonly IReportsService _reportsService;

        public ReportsController(IReportsService reportsService)
        {
            _reportsService = reportsService;
        }
        [HttpGet("/upcoming-deadlines")]
        public async Task<FileContentResult> GetDeadlines( )
        {
            var from = new DateTime();
            var to = DateTime.UtcNow+ new TimeSpan(14,0,0,0);
            var reportInBytes = await
_reportsService.GetDeadlinesReportAsync(from, to);

            var response = File(reportInBytes, "application/pdf", "deadlines.pdf");

            return response;


        }
```

AccessTokenService.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using STUS.DataAccess.Entities;
using STUS.DataAccess.Repositories;
using STUS.DataAccess.Repositories.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace STUS.TelegramBot.Core.Services
{
    public interface IAccessTokenService
    {
        Task<Guid> GenerateTokenAsync(Guid apiUserID);

        Task<bool> AuthenticateByTokenAsync(string userName, Guid token, long chatId);
    }
    public class AccessTokenService : IAccessTokenService
    {
        private readonly UserManager<User> _userManager;
        private readonly ITelegramUserRepository _telegramUserRepository;

        public AccessTokenService(UserManager<User> userManager, ITelegramUserRepository telegramUserRepository )
        {
            _userManager = userManager;
            _telegramUserRepository = telegramUserRepository;
        }

        public async Task<Guid> GenerateTokenAsync(Guid apiUserID)
        {
            var user = await _userManager.Users.FirstOrDefaultAsync(x => x.Id == apiUserID);


            var token = Guid.NewGuid();

            user.TelegramAccesToken = token;
```

```
            await _userManager.UpdateAsync(user);


            return token;
        }


        public async Task<bool> AuthenticateByTokenAsync(string userName, Guid token, long chatId)
        {
            var user = await _userManager.Users.FirstOrDefaultAsync(x => x.TelegramAccesToken == token && x.UserName == userName);
            if (user == null)
            {
                return false;
            }

            var telegramUser = await _telegramUserRepository.GetUserByChatId(chatId);
            if (telegramUser == null) { }

            telegramUser.Student.Id = user.Id;
            await _telegramUserRepository.UpdateAsync(telegramUser);

            return true;
        }

    }
}
```

EmailService.cs

```
using Microsoft.AspNetCore.Identity;
using STUS.DataAccess.Entities;
using STUS.TelegramBot.Core.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mail;
using System.Text;
using System.Threading.Tasks;

namespace STUS.BusinessLogic.Services
{
    public interface IEmailService
    {
        Task SendInvitationAsync(List<string> emailAddresses);

    }

    public class EmailService : IEmailService
    {
        private readonly SmtpClient _smtpClient;
        private readonly UserManager<User> _userManager;
        private readonly IAccessTokenService _accessTokenService;

        public EmailService(SmtpClient smtpClient,
            UserManager<User> userManager,
            IAccessTokenService accessTokenService )
        {
            _smtpClient = smtpClient;
            _userManager = userManager;
            _accessTokenService = accessTokenService;
```

```
}

public async Task SendInvitationAsync(List<string> userIds)
{
    if (userIds == null)
    {
        throw new ArgumentNullException(nameof(userIds));
    }

    var users = new List<User>();
    foreach (var id in userIds)
    {
        var user = await _userManager.FindByIdAsync(id);
        users.Add(user);
    }

    foreach (var user in users)
    {
        var mailMessage = new MailMessage
        {
            From = new MailAddress("stus@gmail.com", "Школа підготовки до ЗНО"),
            Subject = "Запрошення для реєстрацiв в нашому телеграм боті",
            Body = "Дорогий друже,\n\nМи б хотіли запропонувати тобі приєднатися до нашої платформи. Будь ласка, перейди за наступним посиланням, щоб почати користування: https://t.me/mybot \n Та введи токен у форматі /token [твій токен] для авторизації\n\nЗ найкращими побажаннями,\nКоманда платформи"
        };
```

```
                mailMessage.Body += $"\n\n{user.UserName} - {await _accessTokenService.GenerateTokenAsync(user.Id)}";
                mailMessage.To.Add(user.Email);
                await _smtpClient.SendMailAsync(mailMessage);
            }

        }

    }
}
using DinkToPdf;
using DinkToPdf.Contracts;
using STUS.BusinessLogic.Services.Interfaces;
using STUS.DataAccess.Entities;
using STUS.DataAccess.Repositories.Interfaces;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace STUS.BusinessLogic.Services.Reports
{
    public class ReportsService : IReportsService
    {
        private readonly IAssignmentRepository _assignmentRepository;
        private readonly IConverter _converter;

        public ReportsService(IAssignmentRepository assignmentRepository, IConverter converter)
```

```
        {
            _assignmentRepository = assignmentRepository;
            _converter = converter;
        }

        public async Task<byte[]> GetDeadlinesReportAsync(DateTime from, DateTime to)
        {
            var assigments = await _assignmentRepository.GetAssignmentsByPeriod(from, to);
            var htmlReport = GetHTMLreport(assigments);

            var globalSettings = new GlobalSettings
            {
                ColorMode = ColorMode.Color,
                Orientation = Orientation.Portrait,
                PaperSize = PaperKind.A4,
                Margins = new MarginSettings { Top = 10 },
                DocumentTitle = "PDF Report"
            };

            var objSettings = new ObjectSettings
            {
                PagesCount = true,
                HtmlContent = htmlReport,
                WebSettings = { DefaultEncoding = "utf-8", UserStyleSheet = Path.Combine(Directory.GetCurrentDirectory(), "assets", "deadline-report-style.css") },
                HeaderSettings = { FontName = "Arial", FontSize = 9, Right = "Page [page] of [toPage]", Line = true },
```

```
            FooterSettings = { FontName = "Arial", FontSize = 9, Line = true,
Center = "Report Footer" }
        };

        var pdf = new HtmlToPdfDocument()
        {
            GlobalSettings = globalSettings,
            Objects = { objSettings }
        };

        var file = _converter.Convert(pdf);

        return file;


    }

    private string GetHTMLreport(IEnumerable<Assignment> assigments)
    {var sb = new StringBuilder();
        sb.Append(@"
            <html>
                <head></head>
                <body>
                    <div class='header'><h1>Deadlines report</h1></div>
                    <table align='center'>
                        <tr>
                            <th>Assignment</th>
                            <th>Description</th>
                            <th>Deadline</th>
                            <th>Subject</th>
                            <th>Teacher</th>
```

```
                    </tr>");
            foreach (var item in assigments)
            {sb.Append(
               @$"
                  <tr>
                     <td>{item.Title}</td>
                     <td>{item.Description}</td>
                     <td>{item.Deadline}</td>
                     <td>{item.Group.Subject.Title}</td>
                     <td>{item?.Group.Subject?.Teacher?.LastName ?? "----"}</td>
                  </tr>");
            }
            sb.Append(@"
                  </table>
               </body>
               </html>");
            return sb.ToString();
        }
    }
}
```

BaseCommand.cs

```
using System.Threading.Tasks;
using Telegram.Bot.Types;

namespace STUS.TelegramBot.Core.Commands
{
    public abstract class BaseCommand
    {
        public abstract string Name { get; }
        public abstract Task ExecuteAsync(Update update);
    }
```

```
}
```

DeadlinesCommand.cs

```
using STUS.BusinessLogic.Services.Interfaces;
using STUS.TelegramBot.Core.Exceptions;
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace STUS.TelegramBot.Core.Commands
{
    public class DeadlinesCommand : BaseCommand
    {
        private readonly ITelegramUserService _userService;
        private readonly IReportsService _reportsService;
        private readonly TelegramBotClient _botClient;

        public DeadlinesCommand(ITelegramUserService userService,IReportsService reportsService, STUS.Core.TelegramBotClientFactory telegramBot)
        {
            _userService = userService;
            _reportsService = reportsService;
            _botClient = telegramBot.GetBot().Result;
        }
```

```
        public override string Name => throw new NotImplementedException();

        public override async Task ExecuteAsync(Update update)
        {
            var user = await _userService.GetOrCreate(update);

            if (user?.Student.Id == null)
            {
                throw new TelegramException(_botClient, user.ChatId, "Не авторизований юзер." +
                    " Для авторизації введіть токен отриманий від адміністатора у форматі" +
                    " \"/token [email] [token]");
            }
            var from = new DateTime();
            var to = DateTime.UtcNow + new TimeSpan(14, 0, 0, 0);
            var deadlinesBytes = await _reportsService.GetDeadlinesReportAsync(from, to);

            Stream deadlinesFile = new MemoryStream(deadlinesBytes);
            var document = InputFile.FromStream(stream: deadlinesFile, fileName: "deadlines.pdf");
            var message = await _botClient.SendDocumentAsync(user.ChatId, document);
        }
    }
}
```

StartCommand.cs

```
using STUS.TelegramBot.Core.Services.Interfaces;
using STUS.TelegramBot.Core;
using System;
```

```
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;
using STUS.Core;

namespace STUS.TelegramBot.Core.Commands
{
    public class StartCommand : BaseCommand
    {
        private readonly ITelegramUserService _userService;
        private readonly TelegramBotClient _botClient;

        public StartCommand(ITelegramUserService userService,
TelegramBotClientFactory telegramBot)
        {
            _userService = userService;
            _botClient = telegramBot.GetBot().Result;
        }
        public override string Name => throw new NotImplementedException();

        public override async Task ExecuteAsync(Update update)
        {
            var user = await _userService.GetOrCreate(update);
            var replyKeyboard = new ReplyKeyboardMarkup(new[]
            {
                new []
                {
                    new KeyboardButton("Предмети")
                },
```

```
                new[]
                {
                    new KeyboardButton("Мої дедлайни")
                },
                new[]
                {
                    new KeyboardButton("Оцінки")
                },
                new[]
                {
                    new KeyboardButton("Контакти вчителів")
                },

            });

            await _botClient.SendTextMessageAsync(user.ChatId, "Hi there ",
                replyMarkup: replyKeyboard);
        }
    }
}
```

SubjectsCommand.cs

```
using STUS.BusinessLogic.Services.Interfaces;
using STUS.TelegramBot.Core.Exceptions;
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;
```

```
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;

namespace STUS.TelegramBot.Core.Commands
{
    public class SubjectsCommand : BaseCommand
    {

        private readonly ITelegramUserService _userService;
        private readonly ISubjectService _subjectService;
        private readonly TelegramBotClient _botClient;

        public SubjectsCommand(ITelegramUserService userService, ISubjectService subjectsService, STUS.Core.TelegramBotClientFactory telegramBot)
        {
            _userService = userService;
            _subjectService = subjectsService;
            _botClient = telegramBot.GetBot().Result;
        }

        public override string Name => throw new NotImplementedException();

        public override async Task ExecuteAsync(Update update)
        {
            var user = await _userService.GetOrCreate(update);

            if (user?.Student.Id == null)
            {
                throw new TelegramException(_botClient, user.ChatId, "Не авторизований юзер." +
```

```
                " Для авторизації введіть токен отриманий від адміністатора у форматі" +
                " \"/token [email] [token]");
            }

            var subjects = await _subjectService.GetSubjectsByTgUserIdAsync(user.Id);

            var keyboardButtons = new List<KeyboardButton[]>
            {
               new KeyboardButton[] { new KeyboardButton("Назад") }
            };
            foreach (var subject in subjects)
            {
               keyboardButtons.Add(new KeyboardButton[] { new KeyboardButton(subject.Title) });
            }

            var replyKeyboard = new ReplyKeyboardMarkup(keyboardButtons.ToArray());


            await _botClient.SendTextMessageAsync(user.ChatId, "Оберіть предмет зі списку для роботи з ним",
                replyMarkup: replyKeyboard);
        }
    }
}
```

TeachersFullnamesCommand.cs

```
using STUS.BusinessLogic.Services.Interfaces;
using STUS.TelegramBot.Core.Exceptions;
```

```
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace STUS.TelegramBot.Core.Commands
{
    public class TeachersFullnamesCommand : BaseCommand
    {
        private readonly ITelegramUserService _userService;
        private readonly ITeacherService _teacherService;
        private readonly TelegramBotClient _botClient;

        public TeachersFullnamesCommand(ITelegramUserService userService,
ITeacherService teacherService, STUS.Core.TelegramBotClientFactory telegramBot)
        {
            _userService = userService;
            _teacherService = teacherService;
            _botClient = telegramBot.GetBot().Result;
        }

        public override string Name => throw new NotImplementedException();

        public override async Task ExecuteAsync(Update update)
        {
            var user = await _userService.GetOrCreate(update);
```

```
            if (user?.Student.Id == null)
            {
                throw new TelegramException(_botClient, user.ChatId, "Не авторизований юзер." +
                    " Для авторизації введіть токен отриманий від адміністатора у форматі" +
                    " \"/token [email] [token]");
            }

            var teachers = await _teacherService.GetAllTeachersAsync();

            var result = "";
            foreach (var teacher in teachers)
            {
                result += teacher.ToString();

            }
            await _botClient.SendTextMessageAsync(update.Message.Chat.Id, result);
        }
    }
}
```

TokenCommand.cs

```
using STUS.Core;
using STUS.TelegramBot.Core.Services.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using Telegram.Bot;
```

```
using Telegram.Bot.Types;

namespace STUS.TelegramBot.Core.Commands
{
    public class TokenCommand : BaseCommand
    {
        private readonly IBotAccessTokenService _tokenService;
        private readonly TelegramBotClient _botClient;

        public TokenCommand( IBotAccessTokenService tokenService, TelegramBotClientFactory telegramBot)
        {
            _tokenService = tokenService;
            _botClient = telegramBot.GetBot().Result;
        }
        public override string Name => throw new NotImplementedException();

        public async override Task ExecuteAsync(Update update)
        {
            var regex = new Regex(CommandNamesPatterns.Token);
            var message = Regex.Replace(update.Message.Text, CommandNamesPatterns.Token, " ").Trim().Split(" ");
            if (message.Length != 2)
            {

                throw new Exception();
            }
           var result = await _tokenService.AuthenticateByTokenAsync(message[0], new Guid(message[1]), update.Message.Chat.Id );
```

```
            if(result)
            {
                await _botClient.SendTextMessageAsync(update.Message.Chat.Id,
"success");
            }
        }
    }
}
```

TelegramUserService.cs

```
using STUS.DataAccess.Entities;
using STUS.DataAccess.Repositories.Interfaces;
using STUS.TelegramBot.Core.Services.Interfaces;
using System.Threading.Tasks;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;

namespace STUS.TelegramBot.Core.Services
{
    public class TelegramUserService : ITelegramUserService
    {
        private readonly ITelegramUserRepository _telegramUserRepository;

        public TelegramUserService(ITelegramUserRepository
telegramUserRepository)
        {
            _telegramUserRepository = telegramUserRepository;
        }

        public async Task<TelegramUser> GetOrCreate(Update update)
        {
            var newUser = update.Type switch
```

```
            {
                UpdateType.CallbackQuery => new TelegramUser
                {
                    Username = update.CallbackQuery.From.Username,
                    ChatId = update.CallbackQuery.Message.Chat.Id,
                    FirstName = update.CallbackQuery.Message.From.FirstName,
                    LastName = update.CallbackQuery.Message.From.LastName
                },
                UpdateType.Message => new TelegramUser
                {
                    Username = update.Message.Chat.Username,
                    ChatId = update.Message.Chat.Id,
                    FirstName = update.Message.Chat.FirstName,
                    LastName = update.Message.Chat.LastName
                }
            };

            var user = await
_telegramUserRepository.GetUserByChatId(newUser.ChatId);

            if (user != null) return user;

            var result = await _telegramUserRepository.InsertAsync(newUser);
            return result;
        }
    }
}
```