КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТІ
МЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп’ютерних систем
Кафедра комп’ютерної інженерії

УДОСКОНАЛЕННЯ ФУНКЦІЙ СТУДЕНТСЬКОГО ІНТЕРФЕЙСУ СИСТЕМИ TRITON ТА МІГРАЦІЯ НА ПЛАТФОРМУ .NET 5

Випускна кваліфікаційна робота бакалавра
студента 4 року навчання
Спеціальність: 123 «Комп’ютерна інженерія»
ОНП «Комп'ютерні системи та мережі»
Олександра БОДНАРА

Науковий керівник:
канд. технічних наук Євген СЛЮСАР,
асистент кафедри комп’ютерної інженерії

Рецензент:
канд. фіз.-мат. наук Іван КОЛОМІЄЦЬ,
асистент кафедри електрофізики ФРЕКС

До захисту допускаю: Завідувач кафедрою
Юрій БОЙКО

Ухвалено на засіданні кафедри “____” _____ 2022 р., протокол № ___

Київ - 2022

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра містить 81 сторінку, 23 рисунка, 3 таблиці, 4 додатки, використано 22 інформаційні джерела.

Мета роботи – удосконалення функцій студентського інтерфейсу системи Triton та міграція на платформу .Net 5.

Об’єкт дослідження – функціонал студентського інтерфейсу системи Triton та платформа .Net.

В першому розділі досліджено платформу .Net, виконано порівняння попередньої версії проекту Triton .Net Core 3.1 та .Net 5; визначено необхідність міграції на платформу .Net 5.

В другому розділі виконано міграцію на платформу .Net 5; виправлені після міграційні помилки; визначені задачі по удосконаленню функцій інтерфейсу системи Triton; удосконалено функції студентського інтерфейсу системи Triton.

МІКРОСЕРВІС, СИСТЕМА ОЦІНЮВАННЯ, .NET, .NET 5, .NET CORE 3.1, СЕРВІС, АРХІТЕКТУРА, LDAP, ENTITY FRAMEWORK CORE, МІГРАЦІЯ НА ПЛАТФОРМУ .NET 5, ФРЕЙМВОРК, MVC.

ЗМІСТ

ОСНОВНІ ПОНЯТТЯ ТА ТЕРМІНИ

LDAP (Lightweight Directory Access Protocol) – це відкритий і кроссплатформний протокол, що використовується для аутентифікації служб каталогів.

.NET — це безкоштовна платформа для розробки з відкритим вихідним кодом для створення різних типів додатків.

.NET 5.0 — універсальна платформа з відкритим вихідним кодом для розробки різних типів програм (хмарних, серверних, веб-додатків, програм для iPhone, Android, Apple Watch), випущена компанією Microsoft у листопаді 2020 року. Платформа об'єднала у собі програмні рішення .Net Core, .Net Framework, Xamarin та Mono.

.NET Core – це горизонтальний розвиток програмної платформи .NET в інші операційні системи, в яких вона забезпечує можливість використання програм, розроблених для Windows.

Нічна зборка – автоматична збірка, яка виконується один раз в день, як правило, після закінчення робочого дня для більшості розробників.

gRPC (Remote Procedure Calls) – це система віддаленого виклику процедур (RPC) з відкритим вихідним кодом, спочатку розроблена в Google у 2015 році. Як транспорт використовується HTTP/2, як мова опису інтерфейсу – Protocol Buffers. gRPC надає такі функції як автентифікація, двонаправлена потокова передача та управління потоком, блокуючі або неблокуючі прив'язки, а також скасування та тайм-аути. Генерує кросплатформні прив'язки клієнта та сервера для багатьох мов. Найчастіше використовується для підключення служб у мікросервісному стилі архітектури та підключення мобільних пристроїв та браузерних клієнтів до серверних служб.

Entity Framework – спеціальна об'єктно-орієнтована технологія на базі фреймворку .NET для роботи з даними. Якщо традиційні засоби ADO.NET

дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework є більш високим рівнем абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо фізично ми оперуємо таблицями, індексами, первинними та зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

MVC – розшифровується як модель-уявлення-контролер (від англ. model-view-controller). Це спосіб організації коду, який передбачає виділення блоків, відповідальних рішення різних завдань. Один блок відповідає за дані програми, інший відповідає за зовнішній вигляд, а третій контролює роботу програми.

Багфіксинг (bug fixing) або фіксування багів – процес усунення несправностей програмного забезпечення. Є одним з етапів створення програмного продукту. На цьому етапі розробники виправляють помилки в функціонуванні програми, виявлені тестерами під час її тестування.

CI (Continuous Integration) – у дослівному перекладі "безперервна інтеграція". Мається на увазі інтеграція окремих шматочків коду програми між собою. CI – це складання, деплой та тестування програми без участі людини.

Пагінація – поділ великого масиву даних, що є на сайті, на окремі сторінки для зручності використання.

Razor – це синтаксис розмітки для впровадження коду на основі .NET у веб-сторінки. Razor синтаксис складається з Razor розмітки, C# та HTML. Файли, які містять Razor зазвичай мають .cshtml розширення. Razor також знаходиться у файлах Razor компонентів (.razor).

Кросплатформність – здатність програмного забезпечення працювати з декількома апаратними платформами або операційними системами. Забезпечується завдяки використанню високорівневих мов

програмування, середовищ розробки та виконання, що підтримують умовну компіляцію, компонування та виконання коду на різних платформ.

IoT (Internet of Things) — це передова система автоматизації та аналітики, яка використовує технологію мереж, зондування, великих даних і штучного інтелекту для забезпечення повних систем для продукту або послуги. Ці системи забезпечують більшу прозорість, контроль і продуктивність у будь-якій галузі чи системі.

Common Language Infrastructure (CLI) – специфікація загальномовної інфраструктури. Найбільш відомими реалізаціями цього стандарту є Microsoft .NET Framework, Mono, DotGNU Portable.NET.

Common Intermediate Language (CIL) — проміжна мова, розроблена корпорацією Microsoft для платформи .NET Framework. JIT-компілятор, який перетворює CIL код в машинний код, є частиною віртуальної машини CLR (Common Language Runtime) — спільного середовища виконання мов для платформи .NET.

Фреймворк (англіцизм) — програмна платформа, що визначає структуру програмної системи, програмне забезпечення, що полегшує розробку та об'єднання різних компонентів великого програмного проекту.

Бенчмарк (апаратне забезпечення) — задача, що служить еталонним тестом продуктивності системи.

Fortunes – тест продуктивності фреймворку, що використовує ORM, підключення до бази даних, колекції динамічного розміру, сортування, шаблони на стороні сервера, контрзаходи XSS та кодування символів.

Garbage collector – складальник сміття .NET керує виділенням та звільненням пам'яті для програми. При кожному створенні об'єкта середовище CLR виділяє пам'ять об'єкта з керованої купи. Поки в керованій купі є доступний адресний простір, середовище виконання виділяє простір для нових

об'єктів. Проте ресурси пам'яті не безмежні. Зрештою збирачу сміття необхідно виконати збір, щоб звільнити пам'ять. Механізм оптимізації збирача сміття визначає найкращий час для виконання збору, ґрунтуючись на виконаних операціях виділення пам'яті. Коли збирач сміття виконує складання, він перевіряє наявність об'єктів у керованій купі, які більше не використовуються програмою, а потім виконує необхідні операції, щоб звільнити пам'ять.

ВСТУП

Актуальність теми дипломної роботи. Оскільки для продуктивнішої роботи будь-який сервіс потребує оновлення, то студентський інтерфейс системи Triton не став виключенням. Попередня версія .Net Core 3.1 потребувала оновлення до уніфікованого об’єднаного стандарту .Net 5.

Окрім міграції на новішу платформу, також для сервісу було необхідним удосконалення функціоналу інтерфейсу системи, новий функціонал був потрібний адміністраторам для швидшого внесення змін в роботу програми та більше навчальної інформації було потрібно надавати користувачам.

Мета роботи – удосконалення функцій студентського інтерфейсу системи Triton та міграція на платформу .Net 5.

Об’єкт дослідження – функціонал студентського інтерфейсу системи Triton та платформа .Net.

РОЗДІЛ 1

«ХАРАКТЕРИСТИКА ПЛАТФОРМИ РОЗРОБКИ .NET»

1.1 Платформа розробки .NET

.NET — це кросплатформне середовище розробки з відкритим кодом для створення багатьох типів додатків. Розроблена Microsoft, платформа підтримує кілька мов програмування та бібліотек для створення веб-додатків, мобільних, настільних, IoT-додатків тощо.

.Net підтримує такі мови програмування:

- C# (C sharp): сучасна об'єктно-орієнтована мова програмування, яка належить до сімейства мов C.
- F# (F sharp): функціонально-орієнтована мова програмування, член сімейства мов ML. Також підтримує парадигму об'єктно-орієнтованого програмування.
- Visual Basic: мова програмування Microsoft. Стала повноцінною об'єктно-орієнтованою мовою програмування в контексті .NET.

.NET підтримує *Common Language Infrastructure* (CLI), тому вихідний код компілюється в *Common Intermediate Language* (CIL), незалежно від мови програмування, яку ви використовуєте. Це гарантує відмінну взаємодію між мовами на платформі [19].

За допомого CLI багато інших мов можуть компілюватись в .NET CIL. Наприклад IronPython, ClojureCLR, Eiffel, PowerBuilder та багато інших [19].

1.1.1 Архітектура та компоненти .NET

Архітектура .NET базується на двох основних компонентах:

- CoreCLR: середовище виконання .NET. Воно відповідає за виконання програм CLI і включає в себе компілятор.

- CoreFX: API платформа, що реалізує стандартні бібліотеки CLI, тобто набір бібліотек, які забезпечують найпоширеніші функціональні можливості, такі як керування файловою системою, обробка винятків, мережеве спілкування, потоки, відображення тощо. Даний компонент також називають *Unified Base Class Library*.

Поверх основних компонентів є різні фреймворки, тобто бібліотеки, які пропонують підтримку для розробки різних типів додатків. Наприклад:

- ASP.NET: фреймворк для створення веб-додатків та веб-API.
- Windows Presentation Foundation (WPF): графічний інтерфейс користувача для настільних програм Windows .
- Xamarin: фреймворк для створення кроссплатформених мобільних, телевізійних і настільних додатків.
- Blazor: фреймворк для створення клієнтських веб-додатків за допомогою C#. Також дозволяє створювати клієнтські веб-програми в коді WebAssembly .
- ML.NET: фреймворк машинного навчання, який спрощує інтеграцію моделей машинного навчання.

На додаток до фреймворків .NET пропонує підтримку для більшості поширених завдань програмування: керування файлами, мережевий зв’язок, безпека доступу до бази даних. На стороні мережі .NET підтримує програмування сокетів, зв’язок HTTP та gRPC. Це дозволяє створювати мікросервіси з протоколом, який краще відповідає вашим потребам [19].

Для будь-яких інших потреб, не вбудованих у фреймворк, можна знайти величезну кількість конкретних бібліотек у загальнодоступному репозиторії NuGet. Фактично NuGet є менеджером

пакетів для .NET. Він дозволяє створювати, ділитися та використовувати багато бібліотек .NET практично для будь-яких цілей [19].

1.1.2 Версії .NET. Від .NET Framework до .NET

.NET Framework – перший тип .NET. Надавався набір API для найпоширеніших потреб програмування та взаємодією з базовою операційною системою. Працює лише на Windows.

Проект Mono був наступною версією .NET для роботи з Linux. Метою Mono є запуск програм Linux, створених для .NET Framework, і навпаки.

.NET Core — це повне переписування .NET Framework з урахуванням міжплатформної мети. Його перероблена архітектура визначає мінімальний набір функцій як загальне ядро для платформ Windows, Linux і Mac. Решта функцій можна завантажити у вигляді бібліотечних пакетів.

Standard .NET – не є іншою реалізацією .NET. Це специфікація API .NET, що допомагає створювати міжплатформні бібліотеки. Якщо платформа підтримує певну версію .NET Standard, то на ній працюватиме бібліотека, яка підтримує ту саму версію, незалежно від типу пристрою та реалізації фреймворка (.NET Framework, Mono, .NET Core).

З самого початку .NET використовувався для позначення універсальної платформи розробки. Однак з тих пір з’явилося багато реалізацій, тому ім’я .NET створило неоднозначність.

.NET 5 має на меті зробити конкретне початкове бачення універсальної платформи розробки. Він замінює поточні існуючі реалізації: .NET Framework, .NET Core і навіть Mono. Отже, починаючи з .NET 5, лише .NET існуватиме як ім’я та як платформа [19].

1.2 Версія .NET Core 3.1

Акцентуючи увагу на тому, що студентський Тритон до оновлення мав саме версію .NET Core 3.1 хотілось б більш детальніше описати специфіку

роботи саме даної версії і вже в наступних розділах детально порівняти всі зміни після оновлення на новий стандарт .NET 5.

Головною особливістю .NET Core 3.1 було те, що це був випуск довгострокової підтримки (LTS) [2].

.NET Core 3.1 підтримується на наступних операційних системах [2]:

- Alpine: 3.10
- Debian: 9
- Ubuntu: 16.04
- Fedora: 29
- RHEL: 6
- openSUSE: 15
- SUSE Enterprise Linux (SLES): 12 SP2
- macOS: 10.13
- Windows Client: 7, 8.1, 10 (1607)
- Windows Server: 2012 R2

З важливого також наступні елементи керування Windows Forms були видалені .NET Core 3.1 [2]:

- DataGrid
- ToolBar
- ContextMenu
- Menu
- MainMenu
- MenuItem

1.2.1 Продуктивність роботи .NET Core 3.1 в базових класах

Порівняння трьох актуальних рантаймів: .NET Framework 4.8, .NET Core 3.1 і .NET Core 2.1. На наступній конфігурації [21]:

BenchmarkDotNet = v0.12, OS = Windows 10

Intel Core i5-7700K CPU 4.20GHz (Kaby Lake), 1 CPU, 8 logical and 4 physical cores

.NET Core SDK=3.1.101

[Host] : X64 RyuJIT, .NET Core 3.1.1 (CoreCLR 4.700.19.60701, CoreFX 4.700.19.60801)

Job-1 : X64 RyuJIT, .NET Framework 4.8

Job-2 : X64 RyuJIT, .NET Core 2.1.15 (CoreCLR 4.6.28325.01, CoreFX 4.6.28327.02)

Job-3 : X64 RyuJIT, .NET Core 3.1.1 (CoreCLR 4.700.19.60701, CoreFX 4.700.19.60801)

Тести проходили на двох додаткових машинах (Haswell та Sky Lake), щоб переконатися, що результати тестів стабільні та відтворюються на іншому залізі.

Клас ValuesGenerator (та й основу для самих бенчмарків) були взяті із репозиторію перфоманс-тестів. Ці тести використовуються мейнтейнерами .NET Core для тестування оптимізації, що пропонуються.

1.2.1.1 List

Код List_IterateFor

Method	Runtime	Size	Mean	Error	StdDev	Ratio
IterateForInt	.NET 4.8	1000	565.09 ns	0.191 ns	0.127 ns	1.00

Method	Runtime	Size	Mean	Error	StdDev	Ratio
IterateForInt	.NET Core 2.1	1000	451.12 ns	0.237 ns	0.157 ns	0.80
IterateForInt	.NET Core 3.1	1000	451.05 ns	0.142 ns	0.086 ns	**0.80**
IterateForString	.NET 4.8	1000	574.70 ns	6.796 ns	4.495 ns	1.00
IterateForString	.NET Core 2.1	1000	460.76 ns	3.772 ns	2.495 ns	0.80
IterateForString	.NET Core 3.1	1000	460.25 ns	0.682 ns	0.406 ns	**0.80**

Таблиця 1.1. Швидкість виконання тесту List

Результат Core 3.1 JIT генерує більш ефективний код, читання елементів з List в циклі **for** стало швидше на **~20%** [21]**.**

1.2.1.2 Цикл foreach

Код List_IterateForEach

Method	Runtime	Size	Mean	Error	StdDev	Ratio
IterateForEachInt	.NET 4.8	1000	1574.5 ns	2.73 ns	1.81 ns	1.00
IterateForEachInt	.NET Core 2.1	1000	1575.8 ns	3.83 ns	2.27 ns	1.00
IterateForEachInt	.NET Core 3.1	1000	1568.1 ns	0.61 ns	0.40 ns	**1.00**
IterateForEachString	.NET 4.8	1000	8046.3 ns	36.51 ns	24.15 ns	1.00
IterateForEachString	.NET Core 2.1	1000	6465.0 ns	15.26 ns	10.09 ns	0.80
IterateForEachString	.NET Core 3.1	1000	5886.3 ns	14.65 ns	9.69 ns	**0.73**

Таблиця 1.2. Швидкість виконання тесту циклу foreach

Ітерації List з типами за посиланням через **foreach** стало швидше на **27%** , але для типів за значенням нічого не змінилося. Можна оцінити, наскільки **foreach** повільніше, ніж **for** . Різниця їх ефективності на Core

становить **3.5x** (value types) і **12x** (reference types), приблизно як і у повному фреймворку [21].

1.3 Версія .NET 5

.Net 5 — це уніфікована платформа для створення будь-яких типів додатків .Net за допомогою однієї бібліотеки базових класів. Рішення про створення даної платформи було прийнято для того щоб вирішити проблему з великою кількістю фреймворків, що використовуються в .Net Framework та прийти до одного рішення серед всіх версій. Найпопулярніші фреймворки на момент виходу .Net 5 [2]:

.Net Framework	**Stable version**
.Net Core	5.0.4
.Net Framework	4.8
Mono	6.12.0
.Net Standard	2.1
Xamarin(ios.mac)	5.0

Таблиця 1.3. Найпопулярніші фреймворки на момент виходу .Net 5

Зрозуміло, що кожен фреймворк виконував свою функцію в розробці тих чи інших продуктів.

1.3.1 Можливості та платформи .Net 5

.Net 5 – це єдиний продукт, який має нижченаведені можливості і API, які в основному використовується для:

1. Window Desktop application
2. Cross-Platform Console application
3. Cloud Services
4. Websites
5. Universal Window Platform (UWF)
6. WPF
7. DataScience

8. Machine learning
9. Mobile and IOS devices etc

В тому числі включає такі платформи:

1. Window
2. UNIX
3. Legacy
4. Windows
5. IOS
6. Driod
7. HTML5
8. Macintosh etc

C# 8.0 офіційно підтримується до .Net 3.0, а з .Net 5 і далі буде використовуватись C# 9.0. C# 9.0 має багато нових функцій у мові C#. Як оператор верхнього рівня, відповідність реляційного шаблону, функціональний покажчик тощо [2].

Згідно з даними наданими Microsoft далі нас чекає подальший розвиток уніфікованого фреймворку:

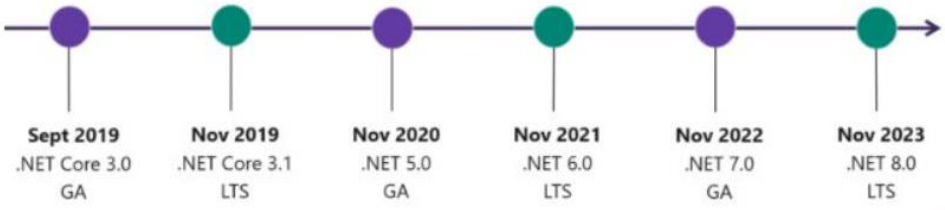

Рис. 1.1. Плани Microsoft щодо подальшого розвитку своєї платформи .Net

1.4 Основні відмінності версій .NET Core 3.1 та .NET 5

.NET 5.0 – це перший реліз на шляху до уніфікації платформи .NET, що дозволяє плавніше мігрувати з .NET Framework.

Насамперед .NET 5.0 це об'єднання різних фреймворків побудованих на .NET – тепер немає Core, Mono, Xamarin - є єдиний .**net5 target**. Фактично, **.net5 target** прийшов на заміну **netcoreapp** і **NetStandard**. Додатково можна вибрати платформу операційної системи, наприклад: net5.0-android, net5.0-ios,

або net5.0-windows. Фреймворк став дійсно кроссплатформеним. У ньому є базовий набір класів та залежні від конкретної операційної системи доповнення. Зокрема, для Windows є підтримка звичного WinForms, який був випущений та перевірений раніше у .NET Core 3.1 [22].

У .NET 5.0 проведено велику роботу з оптимізації як безпосередньо продуктивності додатків, так і швидкості компіляції вихідного коду. Тепер тести та нічні зборки у CI виконуються трохи швидше.

Що стосується продуктивності коду, то в тестах за кількістю gRPC запитів за секунду додаток .NET 5.0 зайняв друге місце після Rust і випередив програмний код, скомпільований Go [20].

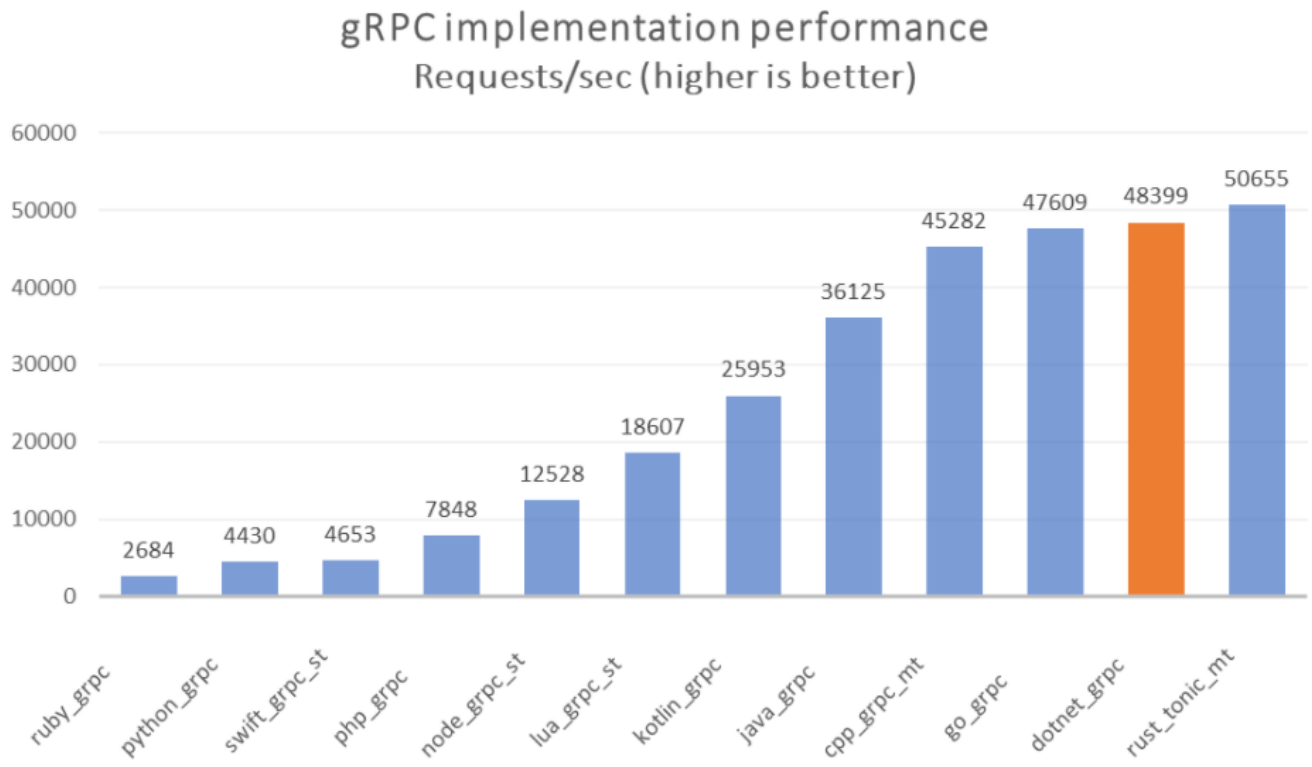

Рис.1.2. Діаграма к-сті gRPC запитів в секунду

Наступним за значимістю нововведенням .NET 5, безумовно, стало включення нової версії C# 9. Найцікавішим нововведенням є поява записів (**records**). Новий синтаксис спрощує опис незмінних моделей даних у

порівнянні з традиційними класами. В результаті вийшов простий синтаксис декларації та ініціалізації.

До складу .NET 5 був включений інсталятор ClickOnce. Тепер є можливість публікації програми у вигляді інсталяційного пакета, який згодом можна запустити на потрібному комп'ютері та розгорнути програму. До складу інсталяційного пакета може бути включений весь фреймворк повністю. Це збільшить обсяг дистрибутива, але спростить встановлення на непідготовлений комп'ютер [21].

Отримав розвиток варіант поширення програми – Single File Application, який з'явився в .NET Core 3.1. Можна створити єдиний файл, що містить у собі всі бібліотеки з можливістю впровадження, у тому числі всіх використовуваних бібліотек фреймворку. Тепер файли стали меншими, і вони швидше запускаються.

1.4.1 Performance .NET 5

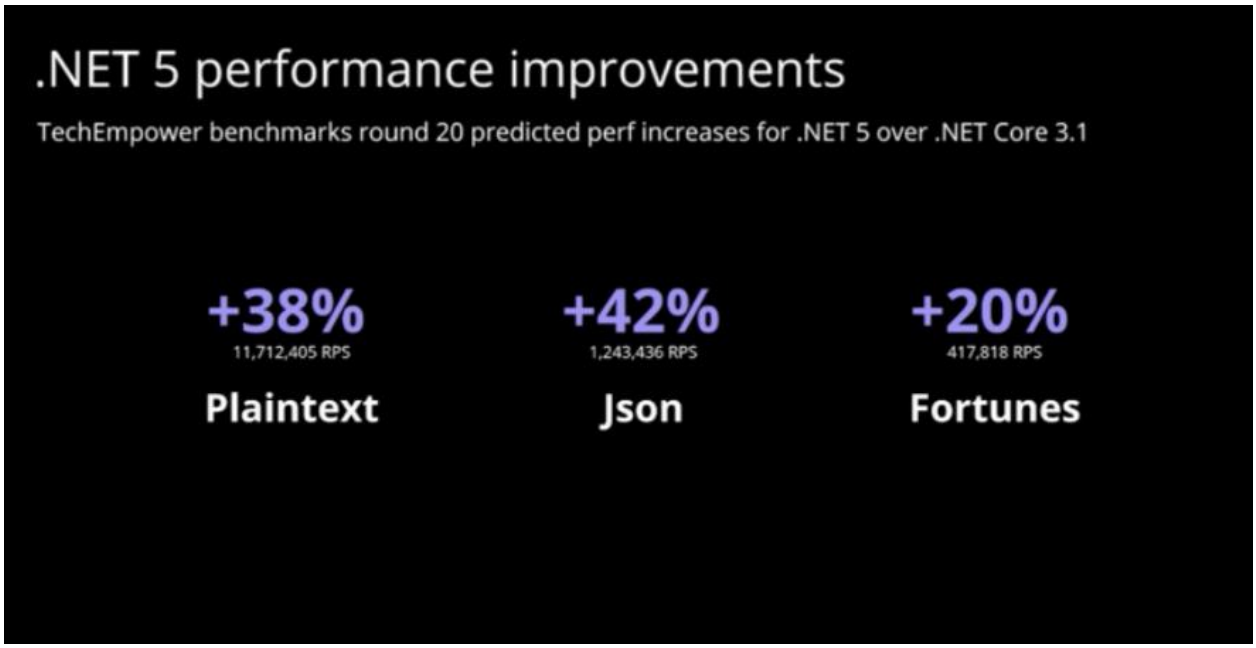

Рис.1.3. .NET 5 Performance покращення порівняно з версією .NET Core 3.1 на основі бенчмарків TechEmpower

Вимірювання продуктивності з даного рисунку засновано на бенчмарках TechEmpower. Тут Microsoft порівнюють результати вимірів продуктивності

.NET 3.1 зі значеннями для .NET 5. Загалом прискорення на 38% для Plaintext, на 42% для Json і на 20% у категорії Fortunes [1].

Крім результатів бенчмарків від TechEmpower, Microsoft надає дані про інші поліпшення. Наприклад, серіалізація JSON стала працювати на 20% швидше, а серіалізація великих колекцій та масивів прискорилася аж утричі [1].

1.4.1.1 Fusion's кеш тест на Ubuntu, Intel, AMD процесор

Тест проводився за допомогою бібліотеки Fusion, секція "Cashing Sample".

Це стрес тести для API, що отримує невеликий об'єкт з бази даних SQL, єдина різниця полягає в тому, як він розкривається (локально або через HTTP) і чи використовується Fusion для забезпечення прозорого кешування [3].

Початковий тест проводився на процесорі Window & AMD, але сьогодні майже всі виробничі служби працюють на Linux, тому було вирішено виправити це і додати до тесту систему на основі Intel [3].

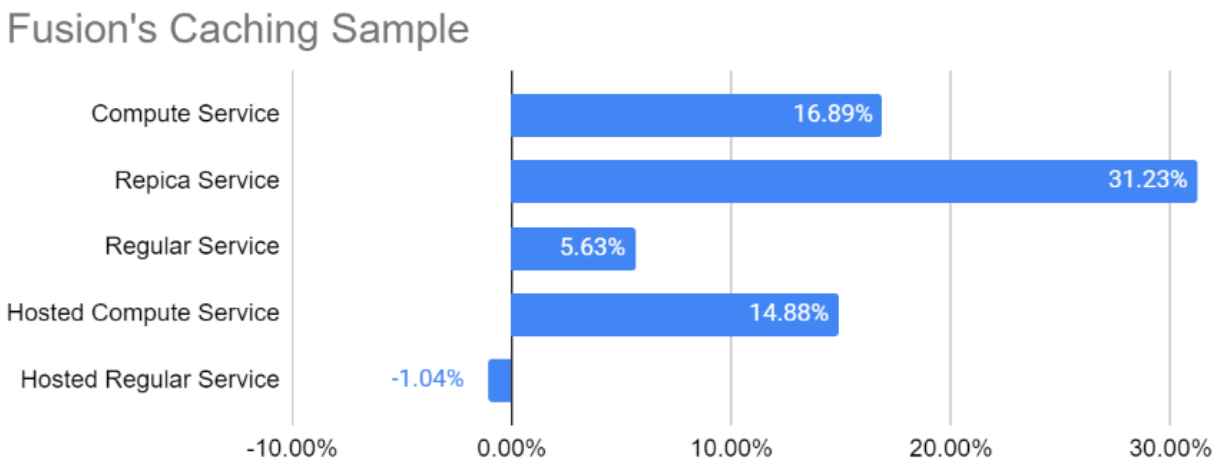

Рис.1.4. .NET 5 vs .NET Core 3.1, Fusion's кеш тест, швидкість, %

Test	Speedup, %	K Operations per second .NET Core 3.1 @ i7-8700K	.NET 5.0 @ i7-8700K	.NET Core 3.1 @ RT3960x	.NET 5.0 @ RT3960x	Test focus
Compute Service	16.89%	5936	7165	22757	25761	JIT compiler efficiency + CPU
Repica Service	31.23%	4908	5938	17517	24935	JIT compiler efficiency + CPU
Regular Service	5.63%	18.306	18.269	34.818	38.926	EF + SQL Server
Hosted Compute Service	14.88%	39.778	38.96	60.275	81.224	ASP.NET Core + JSON.NET
Hosted Regular Service	-1.04%	11.123	11.15	28.027	27.382	ASP.NET Core + EF + SQL Serv
Code: https://github.com/servicetitan/Stl.Fusion.Samples						
OS: Ubuntu 20.04						
System 1: Intel Core i7-8700K, 64 GB RAM						
System 2: Ryzen Threadripper 3960x, 128 GB RAM						

Рис.1.5. .NET 5 vs .NET Core 3.1, 1 Fusion's кеш тест

В даному тесті порівнюється лише час виконання не враховуючи інші бібліотеки.

1.4.1.2 Тест JSON серіалізації

Код [1]:

Method	Runtime	Mean	Ratio	Allocated
LargeArray	.NET FW 4.8	262.06 us	1.00	24256 B
LargeArray	.NET Core 3.1	191.34 us	0.73	24184 B
LargeArray	.NET 5.0	69.40 us	0.26	152 B

Рис.1.6. Результати бенчмарку з серіалізації великих масивів

1.4.1.3 Тест Garbage Collector

Даний тест потрібний для порівняння максимальної продуктивності збору сміття.

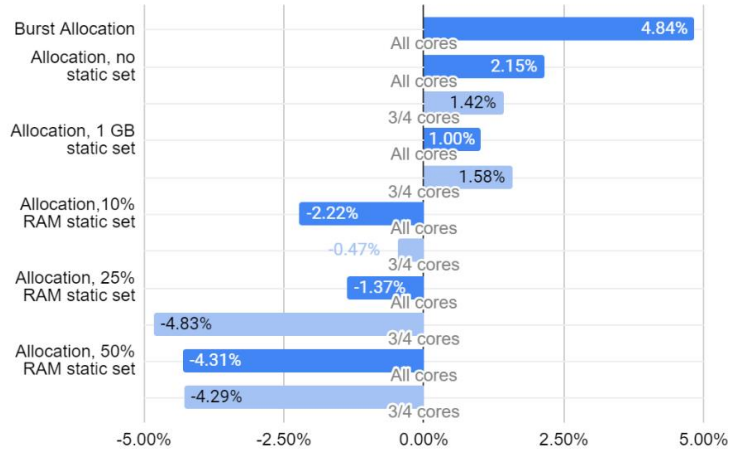

Рис.1.7. .NET 5 vs .NET Core 3.1, тест Garbage Collector, швидкість, %

Test	# of cores allocating	Speedup, %	M Operations per second				Test focus
			.NET Core 3.1 @ i7-8700K	.NET 5.0 @ i7-8700K	.NET Core 3.1 @ RT3960x	.NET 5.0 @ RT3960x	
Burst Allocation	All cores	4.84%	2042.8	2076.85	6188.7	6690.55	Burst Allocator Performance + L1 CPU Cache Bandwidt
Allocation, no static set	All cores	2.15%	79.633	83.079	164.688	164.72	Allocator Performance + RAM Bandwidth
	3/4 cores	1.42%	72.806	74.662	160.894	161.371	
Allocation, 1 GB static set	All cores	1.00%	75.078	76.608	148.266	148.231	Allocator Performance + RAM Bandwidth
	3/4 cores	1.58%	68.598	70.841	151.569	151.435	
Allocation,10% RAM static set	All cores	-2.22%	74.87	71.887	157.845	157.188	Allocator Performance + RAM Bandwidth + GC Efficienc
	3/4 cores	-0.47%	69.623	68.757	157.884	158.367	
Allocation, 25% RAM static set	All cores	-1.37%	70.518	69.134	150.06	148.896	Allocator Performance + RAM Bandwidth + GC Efficienc
	3/4 cores	-4.83%	67.101	64.579	157.791	148.501	
Allocation, 50% RAM static set	All cores	-4.31%	72.033	67.795	158.206	153.925	Allocator Performance + RAM Bandwidth + GC Efficienc
	3/4 cores	-4.29%	66.18	61.757	155.654	152.806	

Рис.1.8. .NET 5 vs .NET Core 3.1, тест Garbage Collector

.NET 5 швидший при розподілі пакетів і менших купах, але повільніший у великих купах, хоча і не сильно [3].

Варто сказати, що GC тест найменш корисний серед всіх тестів: він запускає код, який навряд чи можна побачити у комерційному програмуванні. Його мета — перетворити розподільник пам’яті та GC на вузьке місце для вимірювання їх ефективності, але насправді цього ніколи не відбувається, і навіть якщо обмеження продуктивності розподільника пам’яті будуть досягнуті, у вас є ряд обхідних шляхів для вирішення цієї проблеми [3].

Крім того, GC тест не використовує закріплені об’єкти, тому повністю ігнорує одне з ключових покращень у .NET 5 GC, а саме – спеціальну кучу закріплених об’єктів.

1.5 Висновок

Отже, можна дійти висновків, що висока продуктивність роботи .NET 5, уніфікація платформи, використання C# 9.0 стали головними причинами оновлення студентського інтерфейсу системи Triton до даної версії.

Тестування роботи .NET 5 показало, що висока продуктивність була досягнена завдяки змінам у Garbage Collector, що призвели до зменшення кількості алокацій та прискорення збирання сміття. До речі, всі ці зміни здійснюються одночасно з переписуванням GC на C#, а значить, його код стає куди доступнішим, зрозумілішим і безпечнішим.

Також зміни торкнулися JIT компіляції, асинхронності та багатьох методів базової бібліотеки. Це стосується методів, що працюють з рядками, регулярними виразами та, що особливо важливо, з колекціями та серіалізацією JSON.

Усі ці зміни у прискоренні роботи фреймворка, збільшення можливостей роботи з іншими платформами, зрозуміле та просте оновлення робить необхідним перехід студентського інтерфейсу системи Triton до більш новішої версії в першу чергу для подальшої підтримки та розробки даного продукту іншими розробниками та використанням нових інструментів розробки .NET 5.

РОЗДІЛ 2

«УДОСКОНАЛЕННЯ СТУДЕНТСЬКОГО ІНТЕРФЕЙСУ СИСТЕМИ TRITON»

2.1 Планування розробки, постановка головних задач

Студентський інтерфейс системи Triton – це мікросервіс, що використовує для автентифікації Active Directory/LDAP та зчитує і записує інформацію у 2 бази даних: TritonDatabase та TritonStudentDatabase, сам solution складається з 3 проектів – ADHelpersLib (LDAP бібліотека), Site – фронтенд частина та контролери, TritonStudentModel – моделі бази даних, контролери та процедури для зв’язку з TritonDatabase.

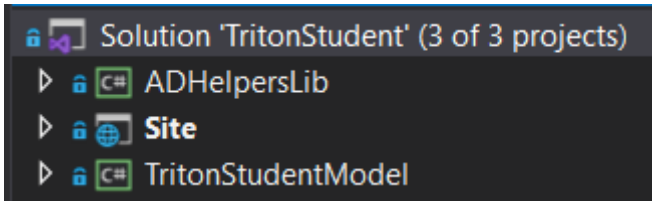

Рис.2.1. Solution TritonStudent

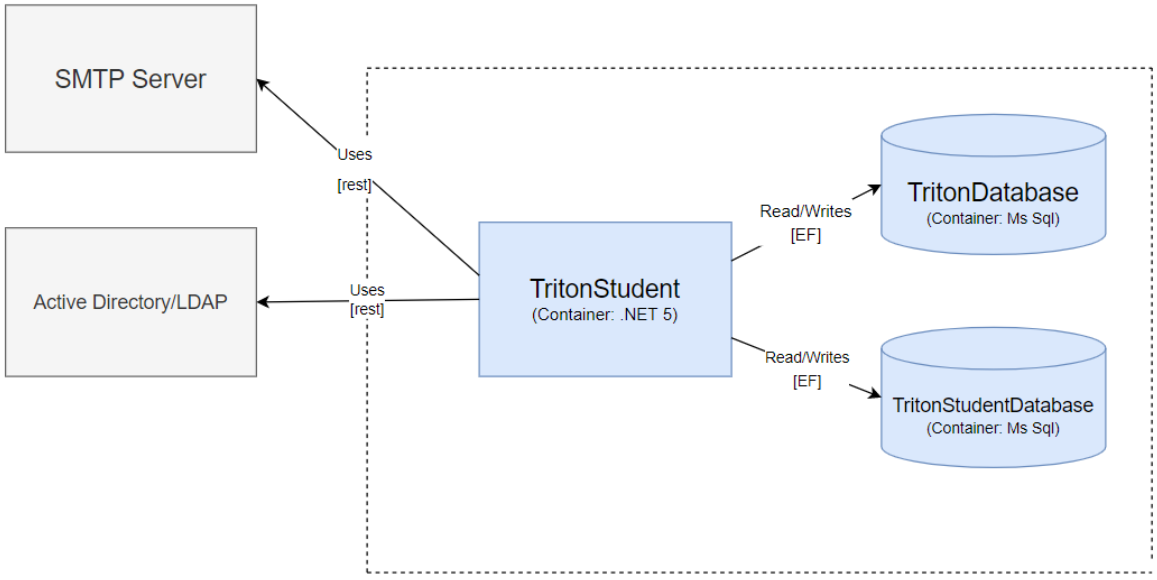

Рис.2.2. Схема роботи мікросервісу студентського інтерфейсу системи Triton

У процесі дослідження роботи сервісу, беручи до уваги головні потреби користувачів та для полегшення розробки інтерфейсу в майбутньому було прийнято рішення зупинитись на такому переліку необхідних змін для подальшого комфортного використання додатку:

1. Оновити сервіс до .Net 5.0. та Entity Framework Core до 5.0 версії.
2. Додати автентифікацію через LDAP сервіс.
3. Для адміністраторів додати відображення
    - панелі запитів з пагінацією;
    - панелі помилок з пагінацією;
    - можливість додавати, видаляти, редагувати новини з логуванням.
4. Для студентів додати відображення:
    - вибору дисциплін;
    - вільного вибору дисциплін;
    - вибору спеціалізацій;
    - навчального плану;
    - практики;
    - державних атестацій;
5. Додати можливість зміни номеру телефона для користувача.

2.2 Архітектура TritonStudent

Для використання автентифікації через LDAP до solution TritonStudent було додано новий проект ADHelpersLib, який далі використовується як бібліотека у проекті Site через який відбувається встановлення зв’язку з сервісом LDAP через контролер при вході користувача.

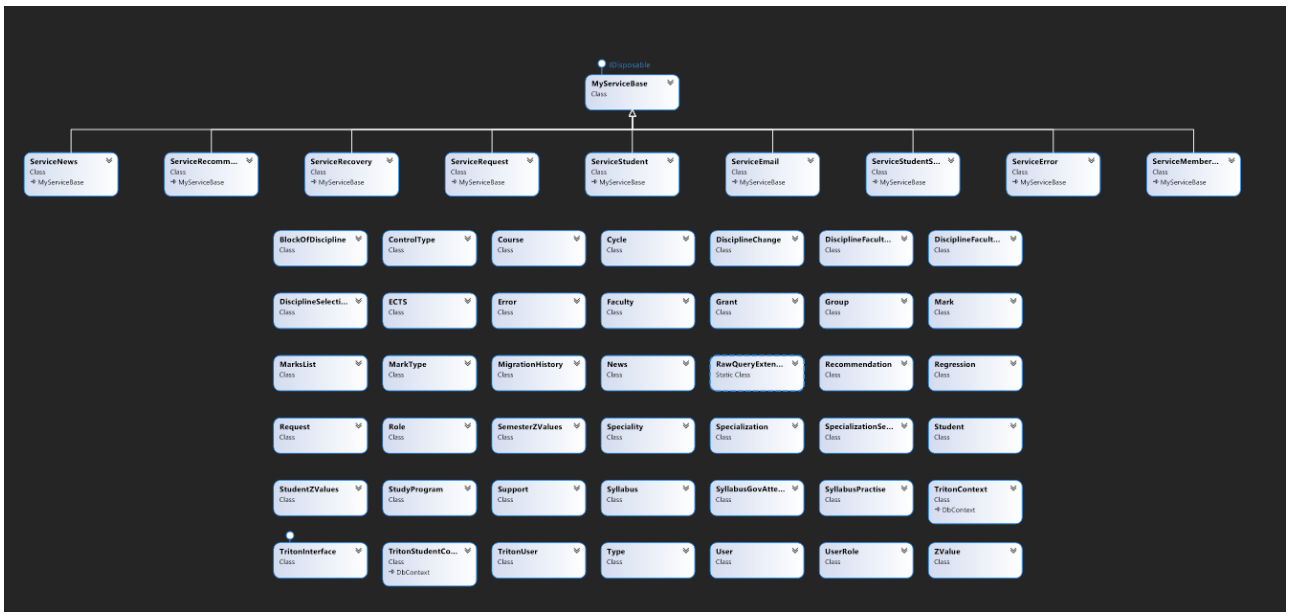

Рис.2.3. Діаграма класів TritonStudentModel

На діаграмі зображено всі Service, де обробляються дані та моделі бази даних.

Рис.2.4. Діаграма класів моделей бази даних Тритона

Власне студентський інтерфейс системи Triton використовує для своєї роботи 2 бази даних TritonContext та більш новішу TritonStudentContext. Для роботи з даними використовується Entity Framework Core 5.0. Проте для більш старішої TritonContext використовуються sql запити класу

TritonInteface, де дані обробляються за допомогою функцій та процедур у базі даних.

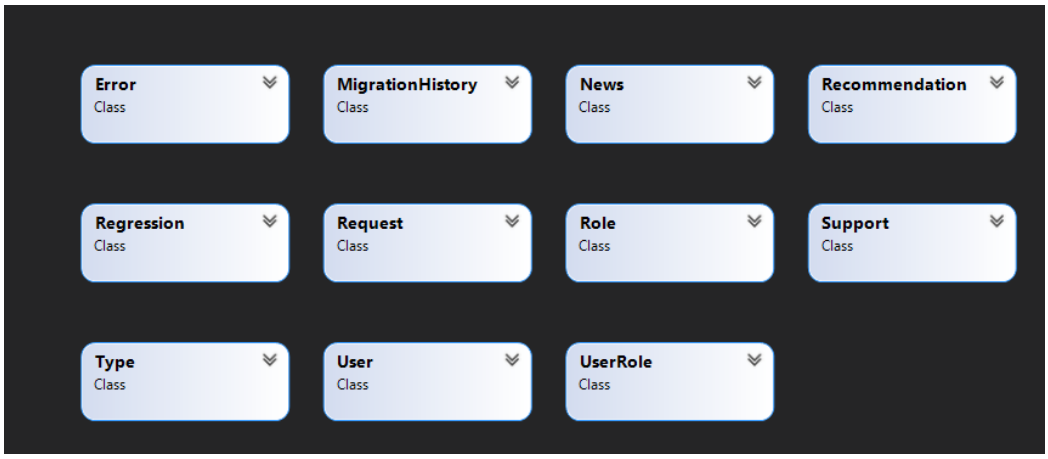

Рис.2.5. Діаграма класів моделей бази даних TritonStudentContext

TritonStudentContext більш новіша база даних, сервіс працює з нею за допомогою Entity Framework Core 5.0. На діаграмі зображені класи даної бази:

- User – клас користувачів.
- UserRole – клас всіх користувачів та їхніх ролей.
- Type – клас довідник, тип запиту: відновлення пошти.
- Support – клас запитів, перелік всіх активних запитів.
- Role – клас довідник, ролі користувачів: адміністратор, староста, студент.
- Request – клас відповідей, перелік всіх відповідей на відновлення пошти.
- Regression – клас дисциплін та регресії.
- Recommendation – клас рекомендацій дисциплін для студентів.
- News – клас новин, перелік всіх доданих новин.
- Error – клас помилок, перелік всіх помилок сервісу.
- MigrationHistory – міграції.

Рис.2.6. Розширена діаграма класів моделей бази даних TritonStudentContext разом з полями, методами, властивостями класів

Для роботи з TritonContext як було сказано раніше використовується клас TritonInterface з sql запитами та допоміжний клас директорії Utils RawQueryExtensions.

Рис.2.7. Діаграма класів TritonContext

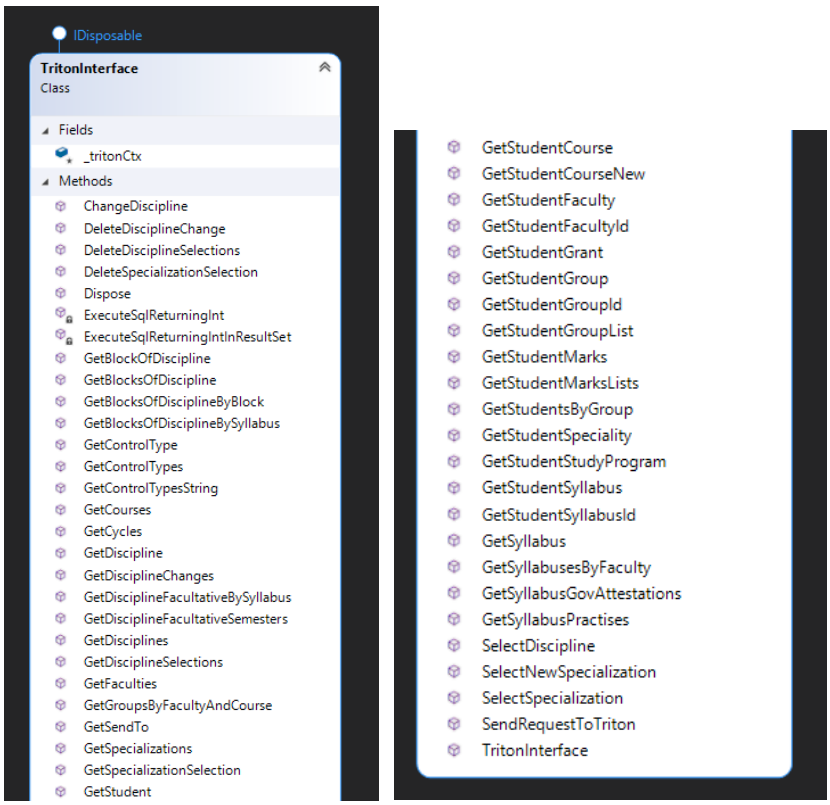

Рис.2.8. Діаграма класу TritonInterface з доступними полями, методами, властивостями

Для роботи з даними на бековій частині в директорії Services використовуються сервіси, це власне сервіс надсилання листів – ServiceEmail, сервіс обробки новин, а саме їх видалення, редагування, оновлення – ServiceNews.

Сервіс хешування паролей – ServiceMembership, обробка помилок – ServiceError, батьківський наслідуваний клас – MyServiceBase, сервіс рекомендацій – ServiceRecommendation, сервіс відновлення пошти, паролю, логіна – ServiceRecovery, сервіс запитів – ServiceRequest, ServiceStudent – для обробки даних студентів, ServiceStudentSelection – для навчальної інформації студента.

Рис.2.9. Діаграма класів сервісів TritonStudentModel

2.3 Оновлення студентського інтерфейсу системи Triton до .Net 5.0 та Entity Framework Core до 5.0 версії.

Оновлено версію SDK до версії 5.0.0:

```
{
  "sdk": {
-    "version": "3.1.200"
+    "version": "5.0.100"
  }
}
```

Після цього було виконано оновлення цільової платформи у файлах csproj до версії net5.0 та оновлення інших пакетів для сумісності.

Site.csproj – фронтенд частина проекту Views з моделями та контролерами, його цільовий файл після оновлення:

```
<Project Sdk="Microsoft.NET.Sdk.Web" >
  <PropertyGroup >
    <TargetFramework >net5.0</TargetFramework >
    <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
    <UserSecretsId>024600f1-4c4a-49f2-96ef-
b7586a9a4e62</UserSecretsId>
  </PropertyGroup>
```

```
  <ItemGroup>
    <PackageReference
Include="Google.Apis.Admin.Directory.directory_v1"
Version="1.55.0.2490" />
    <PackageReference
Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore"
Version="5.0.11" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.UI"
Version="5.0.11" />
    <PackageReference
Include="Microsoft.AspNetCore.Mvc.NewtonsoftJson"
Version="5.0.11" />
    <PackageReference
Include="Microsoft.EntityFrameworkCore.Proxies" Version="5.0.11"
/>
    <PackageReference
Include="Microsoft.VisualStudio.Web.CodeGeneration.Design"
Version="5.0.2" />
    <PackageReference Include="X.PagedList.Mvc.Core"
Version="8.1.0" />
  </ItemGroup>
  <ItemGroup>
    <Folder Include="wwwroot\" />
    <Folder Include="Certificates\" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference
Include="..\ADHelpersLib\ADHelpersLib.csproj" />
    <ProjectReference
Include="..\TritonStudentModel\TritonStudentModel.csproj" />
  </ItemGroup>
</Project>
```

TritonStudentModel.csproj – модель відповідальна за роботу з базою даних, включає сервіси, моделі бази даних, його цільовий файл після оновлення:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Accord.Statistics"
Version="3.8.0" />
    <PackageReference
Include="Microsoft.AspNetCore.Http.Abstractions" Version="2.2.0"
/>
    <PackageReference Include="Microsoft.EntityFrameworkCore"
Version="5.0.11" />
    <PackageReference
Include="Microsoft.EntityFrameworkCore.Relational"
Version="5.0.11" />
    <PackageReference
Include="Microsoft.EntityFrameworkCore.SqlServer"
Version="5.0.11" />
    <PackageReference
Include="Microsoft.EntityFrameworkCore.SqlServer.Design"
Version="1.1.6" />
  </ItemGroup>
</Project>
```

У TritonStudentModel було одразу оновлено Entity Framework Core до 5 версії для роботи з базою даних.

Далі було замінено методи HasName() -> HasDatabaseName(), UseDatabaseErrorPage() -> AddDatabaseDeveloperPageExceptionFilter() для сумісністю з новою версією.

2.3.1 Виправлення багів після оновлення

Після оновлення до .NET 5, користуючись офіційною інструкцією Microsoft, все одно виникла низка багів зв’язаних з взаємодією з базою даних, це були як власне відображення інформації для студентів, так і оновлення даних через Entity Framework.

Один з таких був зв’язаний з рекурсією і вирішився викликом методу GetCourses() через об’єкт класу TritonInterface.

2.3.1.1 SqlQuery

Наступний баг був більш об’ємніший у вирішені, частина логіки проекту в вигляді збережених процедур була реалізована в базі даних MS SQL Server. Ці збережені процедури працюють так, що повертають результат не як код повернення (RETURN @result) а через одиночний датасет (SELECT @result).

До міграції з .Net Core 3.1 та Entity Framework Core 3.1 на .Net 5 та Entity Framework Core 5 працювала конструкція:

```
db.Database.SqlQuery<int>("EXECUTE BlahReturningInt p1,
p2").FirstOrDefault()
```

Проте в Entity Framework Core неможливо створити SqlQuery на простий тип результата, обов’язково потрібно визначити клас моделі. Для вирішення баги можна було визначити модель з одного поля, але тоді потрібне точне співпадіння назв колонки, проте таке рішення було недоцільним архітектурно.

В результаті був доданий статичний клас RawQueryExtensions, який повинен був вирішити дану проблему:

```
    public static class RawQueryExtensions
    {
        public static List<T> ExecuteSqlRawWithResultSet<T>(this
DatabaseFacade databaseFacade,
            string sql, IEnumerable<object> parameters,
Func<DbDataReader, T> map)
        {
            var dependencies =
(IRelationalDatabaseFacadeDependencies)((IDatabaseFacadeDependen
ciesAccessor)databaseFacade).Dependencies;

            using
(dependencies.ConcurrencyDetector.EnterCriticalSection())
            {
                var rawSqlCommand =
dependencies.RawSqlCommandBuilder.Build(sql, parameters);
```

```
                using (var result =
rawSqlCommand.RelationalCommand.ExecuteReader(
                    new
RelationalCommandParameterObject(dependencies.RelationalConnecti
on,
                        rawSqlCommand.ParameterValues, null,

((IDatabaseFacadeDependenciesAccessor)databaseFacade).Context,
                        dependencies.CommandLogger)
                    ))
                {
                    var entities = new List<T>();
                    while (result.Read())
                    {
                        entities.Add(map(result.DbDataReader));
                    }
                    return entities;
                }
            }
        }

        public static List<T> SqlQuery<T>(this DatabaseFacade
databaseFacade, string sqlQuery, params object[] parameters)
        {
            return ExecuteSqlRawWithResultSet<T>(databaseFacade,
sqlQuery, parameters, (record) =>
            {
                var obj = Activator.CreateInstance<T>();
                for(int i = 0; i < record.FieldCount; i++)
                {
                    var prop =
obj.GetType().GetProperty(record.GetName(i));
                    if(prop != null)
                    {
                        prop.SetValue(obj,
object.Equals(record[i], DBNull.Value) ? null : record[i]);
                    }
                }
                return obj;
            });
        }

        public static List<TEntity> SqlQuery<TEntity>(this
DbSet<TEntity> dbSet, string sqlQuery, params object[]
parameters) where TEntity: class
        {
            var context =
dbSet.GetService<ICurrentDbContext>().Context;
            return SqlQuery<TEntity>(context.Database, sqlQuery,
parameters);
        }
    }
```

Методи класу RawQueryExtensions викликаються в класі TritonInterface, де викликається partial class TritonContext бази даних TritonStudent з якою встановлюється зв’язок.

RawQueryExtensions складається з трьох статичних методів:

- public static List<T> ExecuteSqlRawWithResultSet<T>(this DatabaseFacade databaseFacade, string sql, IEnumerable<object> parameters, Func<DbDataReader, T> map)
- public static List<T> SqlQuery<T>(this DatabaseFacade databaseFacade, string sqlQuery, params object[] parameters)
- public static List<TEntity> SqlQuery<TEntity>(this DbSet<TEntity> dbSet, string sqlQuery, params object[] parameters) where TEntity: class

Метод, який в результаті вирішив дану багу ExecuteSqlRawWithResultSet(), його можна підключити до проекту та виконуати запроси, що повертають будь-які датасети.

Як приклад метод ExecuteSqlReturningIntInResultSet(), що викликає ExecuteSqlRawWithResultSet() і реалізований в TritonInterface:

```
        private int ExecuteSqlReturningIntInResultSet(string
query, params object[] parameters)
        {
            string qs = string.Format("EXECUTE {0}", query);
            return
_tritonCtx.Database.ExecuteSqlRawWithResultSet(qs, parameters, x
=> (int)x[0]).FirstOrDefault();
             }
```

І як приклад використання даного методу для отримання даних з бази даних через процедури:

```
       public int SelectSpecialization(int selectionId, int
specializationId)
                =>
ExecuteSqlReturningIntInResultSet("[TritonStudent].SelectSpecial
ization {0}, {1}", selectionId, specializationId);
```

2.3.1.2 ServiceEmail

Для виправлення багу з відправкою листів було внесено зміни в HomeController у Post метод EmailReset(), що знаходить в проекті Site та додано актуальні smtp credentials у ServiceEmail() – метод надсилання листів TritonStudentModel.

ServiceEmail:

```
    public class ServiceEmail : MyServiceBase
    {
        public static void SendEmail(string[] To, string
Subject, string Text)
        {
            try
            {
                SmtpClient smtp = new SmtpClient("", 587);
                smtp.Credentials = new NetworkCredential("",
"");

                MailMessage Message = new MailMessage();

                Message.From = new
MailAddress("help+triton@knu.ua");


                foreach (var em in To)
                {
                    Message.To.Add(new MailAddress(em));
                }
                Message.Subject = Subject;
                Message.Body = Text;

                smtp.Send(Message);
            }
            catch (Exception)
            {
            }
        }
        public static void SendEmail(string To, string Subject,
string Text)
        {
            SendEmail(new string[] { To }, Subject, Text);
        }

         }
```

EmailReset:

```
        [HttpPost]
        public IActionResult EmailReset(pmEmailReset pm)
        {
            var emailResetRequest = GetEmailResetRequest();

            if (emailResetRequest == null)
                return RedirectToAction("SessionTimeOut");

            if (ModelState.IsValid)
            {
                if (pm.Password == null)
                    ModelState.AddModelError("Password", "Поле є
обов'язковим до заповненя.");
                else if
(!(servAD.ADAuthenticateUser(servMembership.GetUser().AduserName
, pm.Password)))
                    ModelState.AddModelError("Password",
"Неправильний пароль.");
                if (ModelState.IsValid)
                {
                    emailResetRequest = new EmailResetRequest
                    {
                        AlternativeEmail = pm.AlternativeEmail,
                        EmailVerificationCode =
string.Concat(ServiceMembership.Encode(pm.AlternativeEmail,
2).Select(x => x.ToString("X2")))
                    };
                    SaveEmailResetRequest(emailResetRequest);

                    var altemail =
emailResetRequest.AlternativeEmail;
                    var code =
emailResetRequest.EmailVerificationCode;
                    Task.Factory.StartNew(() => {
ServiceEmail.SendEmail(altemail, "Код перевірки альтернативної
пошти", code); });

                    return RedirectToAction("EmailValidation");
                }
            }
            return View(pm);
             }
```

Отже маємо відповідну MVC модель при надсиланні листів:

Settings(view) -> HomeController(EmailReset()) -> ServiceEmail(smtp запит) -> SMTP сервер.

При змінні пошти у налаштуваннях view модель звертається до контролера Post метода EmailReset()), який перевіряє пароль користувача,

якщо вірний то формується код перевірки та через метод SendEmail() класу ServiceEmail директорії Services надсилається до внутрішнього SMTP серверу, credentials якого вказані у ServiceEmail, далі сервер надсилає лист на попередню пошту і після виконується перенаправлення на перевірку даного коду:

```
RedirectToAction("EmailValidation")
```

Метод SendEmail() перегружений, щоб строку To перетворити в масив строк і додати як адрес відпраки в MailMessage Message:

```
foreach (var em in To)
{
    Message.To.Add(new MailAddress(em));
}
```

2.4 LDAP автентифікація

Для автентифікації використовується бібліотека ADHelpersLib, складається з публічного класу ADHelper, який реалізує інтерфейс IADHelper та фабрики ADHelperFactory для встановлення зв'язку з сервісом LDAP.

Перехід на LDAP був вимушеним у зв'язку з оновленням до .NET 5 та потенційним розгортанням у системі Kubernetes, тому забезпечувати автентифікацію за старою схемою вбудованими засобами Windows, що включена до домену AD, було неможливо.

ADHelperFactory:

```
public class ADHelperFactory
{
    private static Dictionary<string, IADHelper> adHelpers =
new Dictionary<string, IADHelper>();
    public static IADHelper CreateADHelper(string
connectionString)
    {
        lock(adHelpers)
        {
            if(!adHelpers.ContainsKey(connectionString))
            {
                adHelpers.Add(connectionString, new
ADHelper(connectionString));
            }
```

```
                return adHelpers[connectionString];
            }
        }
         }
```

ADHelperFactory складається з методу CreateADHelper() та словника adHelpers. Метод CreateADHelper() виконує блокування з'єднання через оператор lock та перевіряє чи словник включає в собі посилання (connectionString), якщо немає, то додає його словник.

IADHelper:

```
    public interface IADHelper
    {
        bool CreateUserAccount(string username, string password,
string displayName = null, string ouPath = "ou=Users");
        bool IsLoginFree(string user_login);
        bool ResetUserPassword(string username, string
new_password);
        bool ChangeUserPassword(string username, string
oldPassword, string newPassword);
        bool IsUserInGroup(string username, string group_name);
        bool AddUserToGroup(string username, string group_name);
        bool RemoveUserFromGroup(string username, string
group_name);
        bool AuthenticateUser(string username, string password);
        string GetNetbiosLoginName(string username);
        string GetUserAttribute(string username, string
attribute);
        bool SetUserAttribute(string username, string attribute,
string value);
        bool DeleteUserAttribute(string username, string
attribute);
        bool UserHasMailbox(string username);
        string GetUserMailboxAddress(string username);
        KeyValuePair<string, string>?
GetDistributionGroup(string alias);
        bool CreateDistributionGroup(string alias, string name,
string ouPath);
        bool DeleteDistributionGroup(string alias);
    }
     }
```

IADHelper – інтерфейс, що реалізований в класі ADHelper через який відбуваються головні операції по роботі з сервісом LDAP. Це створення аккаунта, перевірка на вільний логін, скидання пароля, зміна пароля, перевірка

чи юзер входить до групи, додавання та видалення юзера з групи, аутентифікація і так далі.

Далі бібліотека ADHelpersLib імпортується до Site та використовується при автентифікації:

```
        <ProjectReference
Include="..\ADHelpersLib\ADHelpersLib.csproj" />
```

2.5 Інтерфейс адміністратора

2.5.1 Перевірка ролі користувача

Перевірка ролі користувача виконується через атрибут авторизації на фронтенд частині для відображення елементу користувацького інтерфейсу наприклад кнопки, та бекенд частині також через атрибути у контролерах:

```
[Authorize(Roles = "5")]

@if (User.IsInRole("5"))
```

Авторизація в ASP.NET контролюється за допомогою AuthorizeAttribute та визначених параметрів, перевірку яких він буде виконувати, в нашому випадку є три ролі для перевірки: староста – 1, адміністратор – 5, студент – 2. При застосуванні атрибута [Authorize] до контролера, дії чи Razor сторінки, якщо користувач не прийшов перевірку, то доступ до даних компонентів обмежується.

2.5.2 Пагінація

Для пагінації додано публічний клас PageViewModel у директорію Models проекту Site, вказати мінімальну к-сть відображуваних елементів потрібно через поле pageSize. Властивість HasNextPage перевіряє наявність наступної сторінки, HasPreviousPage перевіряє наявність попередньої сторінки.

```
public class PageViewModel
{
    public int TotalPages { get; private set; }
    public int PageNumber { get; private set; }
```

```
        public PageViewModel(int pageNumber, int count, int
pageSize)
        {
            PageNumber = pageNumber;
            TotalPages = (int)Math.Ceiling(count /
(double)pageSize);
        }

        public bool HasPreviousPage
        {
            get
            {
                return (PageNumber > 1);
            }
        }

        public bool HasNextPage
        {
            get
            {
                return (PageNumber < TotalPages);
            }
        }
         }
```

2.5.3 Панель запитів

Панель запитів складається з LookSupports сторінки доданої в директорію Home та метода IActionResult LookSupports.

LookSupports.cshtml:

```
@using Site.Models

@model IndexViewModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    ViewBag.Title = "Запити";
}

<script src="~/Scripts/jquery-1.10.2.min.js"></script >
<script src="~/Scripts/LookSupports.js"></script >
<h2>Запити</h2>
<div class="well bs-component">
    @using (Html.BeginForm())
    {
        <table class="table">
            <thead>
                <tr>
```

```
                    <th
width="30px">@Html.CheckBox("checkedAll", false, new { title =
"Відмітити усі" })</th>
                    <th class="text-center">Запити</th>
                    <th width="40px">Дата</th>
                </tr>
            </thead>
            <tbody>
                @for (int i = 0; i < Model.Supports.Count; ++i)
                {
                    @Html.HiddenFor(m =>
Model.Supports[i].SupportId)
                    @Html.HiddenFor(m =>
Model.Supports[i].Comment)
                    @Html.HiddenFor(m =>
Model.Supports[i].DateAdd)
                    <tr>
                        <td>@Html.CheckBoxFor(m =>
Model.Supports[i].Checked, new { title = "Відмітити як
зроблене", @class = "checkbox" })</td>
                        <td>@Model.Supports[i].Comment</td>
                        <td>@Model.Supports[i].DateAdd</td>
                    </tr>
                }
            </tbody>
        </table>

        if (Model.Supports.Count != 0)
        {
            <center><input type="submit" value="Зафіксувати
виконані запити" class="btn btn-success" /></center>
        }
        else
        {
            <center><a href="/" class="btn btn-
success">Назад</a></center>
        }
        @if (Model.PageViewModel.HasPreviousPage)
        {
            <a asp-action="LookSupports"
               asp-route-page="@(Model.PageViewModel.PageNumber
- 1)"
               class="btn btn-outline-dark">
                <i class="glyphicon glyphicon-chevron-left"></i>
                Назад
            </a>
        }
        @if (Model.PageViewModel.HasNextPage)
        {
            <a asp-action="LookSupports"
               asp-route-page="@(Model.PageViewModel.PageNumber
+ 1)"
```

```
                class="btn btn-outline-dark">
                 Вперед
                 <i class="glyphicon glyphicon-chevron-
right"></i>
             </a>
         }
     }

</div>
```

IActionResult LookSupports:

```
        [Authorize(Roles = "1, 5")]
        public IActionResult LookSupports(string login, int page
= 1)
        {
            int pageSize = 20;
            int studentId =
serRec.GetStudentIdByLogin(User.Identity.Name);
            int roleId =
serRec.GetUserRoleIdByLogin(User.Identity.Name);
            if (studentId != 0 && roleId == 1)
            {
                List<Support> supports =
serRec.GetSupportsForStudentId(studentId, 1);
                int supportsCount = supports.Count();
                supports = supports.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
                List<pmSupport> pmSups = new List<pmSupport>();
                foreach (Support support in supports)
                    pmSups.Add(new pmSupport { SupportId =
support.Id, Checked = false, Comment = support.Comment, DateAdd
= support.DateAdd.Date.ToString("d") });
                PageViewModel pageViewModel = new
PageViewModel(supportsCount, page, pageSize);
                IndexViewModel viewModel = new IndexViewModel
                {
                    PageViewModel = pageViewModel,
                    Supports = pmSups
                };
                return View(viewModel);
            }
            else if (studentId != 0 && roleId == 5)
            {
                List<Support> supports =
serRec.GetSupportsForAdmin(1);
                int supportsCount = supports.Count();
                supports = supports.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
                List<pmSupport> pmSups = new List<pmSupport>();
                foreach (Support support in supports)
```

```
                    pmSups.Add(new pmSupport { SupportId =
support.Id, Checked = false, Comment = support.Comment, DateAdd
= support.DateAdd.Date.ToString("d") });
                PageViewModel pageViewModel = new
PageViewModel(supportsCount, page, pageSize);
                IndexViewModel viewModel = new IndexViewModel
                {
                    PageViewModel = pageViewModel,
                    Supports = pmSups
                };
                return View(viewModel);
            }
            return RedirectToAction("Index", "Home");
        }

    [HttpPost]
        public IActionResult LookSupports(List<pmSupport>
pmSupports)
        {
            if (ModelState.IsValid)
            {
                foreach (pmSupport support in pmSupports)
                    if (support.Checked)

serRec.MarkTheSupportAsCompleted(support.SupportId);
            }
            return RedirectToAction("LookSupports", "Home");
             }
```

Після авторизації, якщо роль користувача адміністратор йому відображається кнопка запитів. Також є чекбокс для фіксації виконаних запитів.

public IActionResult LookSupports – перегружений екшн, який з пагінацією обробляє запити перед відображенням.

Щоб відобразити кількість запитів у клас MenuHelper додано метод GetCountOfSupports(), що через властивість Count повертає к-сть відповідних моделей. Водночас, якщо запити зв’язані зі старостою, то їй відображається інша логіка та запити, які прийшли лише до неї, обробка виконується за допомогою ролі користувача з моделі Role:

```
        public static int GetCountOfNews(this IHtmlHelper html,
string login)
        {
            ServiceNews serNews = GetService<ServiceNews>(html);
            return serNews.GetNews().ToList().Count;
```

```
            }
```

2.5.4 Панель помилок

Панель помилок складається з LookErrors сторінки доданої в директорію Home та метода IActionResult LookErrors.

LookErrors.cshtml:

```
@using Site.Models
@model IndexViewModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    ViewBag.Title = "Помилки";
}

<script src="~/Scripts/jquery-1.10.2.min.js"></script >
<script src="~/Scripts/LookSupports.js"></script >
<h2>Помилки</h2>
<div class="well bs-component">
    @using (Html.BeginForm())
    {
        <table class="table table-hover">
            <thead>
                <tr>
                    <th
width="30px">@Html.CheckBox("checkedAll", false, new { title =
"Відмітити усі" })</th>
                    <th class="text-center">Exception Type</th>
                    <th class="text-center">Message</th>
                    <th class="text-center">Referrer</th>
                    <th class="text-center">Parameters</th>
                    <th class="text-center">Request Type</th>
                    <th class="text-center">Stack Trace</th>
                    <th class="text-center">UserId</th>
                    <th width="40px">Дата</th>
                </tr>
            </thead>
            <tbody>
                @for (int i = 0; i < Model.Errors.Count; ++i)
                {
                    @Html.HiddenFor(m => Model.Errors[i].Id)
                    @Html.HiddenFor(m => Model.Errors[i].UserId)
                    @Html.HiddenFor(m =>
Model.Errors[i].Message)
                    @Html.HiddenFor(m =>
Model.Errors[i].Referef)
                    @Html.HiddenFor(m =>
Model.Errors[i].LastDate)
```

```
                    @Html.HiddenFor(m =>
Model.Errors[i].ExceptionType)
                    @Html.HiddenFor(m =>
Model.Errors[i].Parameters)
                    @Html.HiddenFor(m =>
Model.Errors[i].RequestType)
                    @Html.HiddenFor(m =>
Model.Errors[i].StackTrace)
                    <tr>
                        <td>@Html.CheckBoxFor(m =>
Model.Errors[i].Checked, new { title = "Відмітити як зроблене",
@class = "checkbox" })</td>
                        <td>
                            <a href="#Exception-@i" data-
toggle="collapse">

@Model.Errors[i].ExceptionType.Substring(0,
Model.Errors[i].ExceptionType.Length > 20 ? 20 :

Model.Errors[i].ExceptionType.Length)
                            </a>
                            <div id="Exception-@i"
class="collapse">
                                @Model.Errors[i].ExceptionType
                            </div>
                        </td>
                        @if (Model.Errors[i].Message != null)
                        {
                            <td>
                                <a href="#Message-@i" data-
toggle="collapse">

@Model.Errors[i].Message.Substring(0,
Model.Errors[i].Message.Length > 30 ? 30 :

Model.Errors[i].Message.Length)
                                </a>
                                <div id="Message-@i"
class="collapse">
                                    @Model.Errors[i].Message
                                </div>
                            </td>
                        }
                        else
                        {
                            <td>Null</td>
                        }
                        <td>@Model.Errors[i].Referef</td>
                        @if (Model.Errors[i].Parameters != null)
                        {
                            <td>
```

```
                                    <a href="#Parameters-@i" data-
toggle="collapse">

@Model.Errors[i].Parameters.Substring(0,
Model.Errors[i].Parameters.Length > 30 ? 30 :

Model.Errors[i].Parameters.Length)
                                    </a>
                                    <div id="Parameters-@i"
class="collapse">
                                        @Model.Errors[i].Parameters
                                    </div>
                                </td>
                            }
                            else
                            {
                                <td>Null</td>
                            }
                            <td>@Model.Errors[i].RequestType</td>
                            <td>
                                <a href="#StackTrace-@i" data-
toggle="collapse">

@Model.Errors[i].StackTrace.Substring(0,
Model.Errors[i].StackTrace.Length > 30 ? 30 :

Model.Errors[i].StackTrace.Length)
                                    </a>
                                    <div id="StackTrace-@i"
class="collapse">
                                        @Model.Errors[i].StackTrace
                                    </div>
                            </td>
                            <td>@Model.Errors[i].UserId</td>
                            <td>@Model.Errors[i].LastDate</td>
                        </tr>
                    }
                </tbody>
            </table>

            if (Model.Errors.Count != 0)
            {
                <center><input type="submit" value="Зафіксувати
виконані помилки" class="btn btn-success" /></center>
            }
            else
            {
                <center><a href="/" class="btn btn-
success">Назад</a></center>
            }
            @if (Model.PageViewModel.HasPreviousPage)
            {
```

```
            <a asp-action="LookErrors"
               asp-route-page="@(Model.PageViewModel.PageNumber
- 1)"
               class="btn btn-outline-dark">
                <i class="glyphicon glyphicon-chevron-left"></i>
                Назад
            </a>
        }
        @if (Model.PageViewModel.HasNextPage)
        {
            <a asp-action="LookErrors"
               asp-route-page="@(Model.PageViewModel.PageNumber
+ 1)"
               class="btn btn-outline-dark">
                Вперед
                <i class="glyphicon glyphicon-chevron-
right"></i>
            </a>
        }
    }

     </div>
```

IActionResult LookErrors:

```
        [Authorize(Roles = "5")]
        public IActionResult LookErrors(int page = 1)
        {
            int pageSize = 20;
            List<pmError> pmErrors = new List<pmError>();
            List<Error> errors = serEr.GetErrors();
            int errorsCount = errors.Count();
            errors = errors.Skip((page - 1) *
pageSize).Take(pageSize).ToList();
            if (errors != null)
            {
                foreach (Error error in errors)
                    pmErrors.Add(new pmError
                    {
                        Id = error.Id,
                        Checked = false,
                        UserId = error.UserId,
                        LastDate =
error.LastDate.Date.ToString("d"),
                        Referef = error.Referef,
                        Message = error.Message.Replace('#', '
'),
                        ExceptionType = error.ExceptionType,
                        Parameters = error.Parameters,
                        RequestType = error.RequestType,
                        StackTrace =
error.StackTrace.Replace('#', ' ')
                    });
```

```
                PageViewModel pageViewModel = new
PageViewModel(errorsCount, page, pageSize);
                IndexViewModel viewModel = new IndexViewModel
                {
                    PageViewModel = pageViewModel,
                    Errors = pmErrors
                };
                return View(viewModel);
            }
            return RedirectToAction("Index", "Home");
        }

        [HttpPost]
        public IActionResult LookErrors(List<pmError> pmErrors)
        {
            if (ModelState.IsValid)
            {
                foreach (pmError error in pmErrors)
                    if (error.Checked)
                        serEr.MarkTheErrorAsCompleted(error.Id);
            }
            return RedirectToAction("LookErrors", "Home");
             }
```

Після перевірки ролі користувача йому відображається кнопка помилок, самій панелі є чекбокс для фіксації вирішених помилок.

public IActionResult LookErrors – перегружений екшн, який з пагінацією обробляє помилки перед відображенням.

Для відображення кількості помило у класі MenuHelper створено методи GetCountOfErrors(), що через властивість Count повертаються к-сть відповідних моделей:

```
        public static int GetCountOfErrors(this IHtmlHelper
html, string login)
        {
            ServiceError serEr = GetService<ServiceError>(html);
            return serEr.GetErrors().Count;
             }
```

2.5.5 Панель новин

Для адміністратора є можливість публікувати новини з відображенням усім користувачам через UI їх кількість. Для цього до контролеру NewsController у проекті Site та класу ServiceNews проекту TritonStudentModel додані відповідні методи на додавання, видалення та редагування новин.

NewsController:

```
    [Authorize]
    public class NewsController : Controller
    {
        ServiceStudent serv;
        private ServiceNews serviceNews;
        private ServiceNewsOperations serviceNewsOperations;

        public NewsController(ServiceNews _serviceNews,
ServiceStudent _serv, ServiceNewsOperations
_serviceNewsOperations)
        {
            serviceNews = _serviceNews;
            serv = _serv;
            serviceNewsOperations = _serviceNewsOperations;
        }

        const int PAGESize = 10;

        private string GetShortBody(News news)
        {
            const int CountWords = 100;

            List<string> words = new
List<string>(news.Body.Split(new char[] { ' ' }));
            if (CountWords < words.Count)
            {
                words.RemoveRange(CountWords, words.Count -
CountWords);
                return String.Join(" ", words);
            }
            return news.Body;
        }

        private pmNews Map(News news)
            => new pmNews
            {
                Id = news.Id,
                Author = news.Author,
                Title = news.Title,
                Date = news.Date,
                Body = news.Body,
                ShortBody = GetShortBody(news)
            };

        public IActionResult Index(int page = 1)
        {
            IEnumerable<pmNews> news =
serviceNews.GetNews().Select(news => Map(news));
            ViewBag.Page = page;
            ViewBag.PageSize = PAGESize;
```

```
            return View(news.ToPagedList(page, PAGESize));
        }

        public IActionResult Details(int? id)
        {
            if (id == null)
            {
                return BadRequest();
            }
            pmNews news = Map(serviceNews.Find(id));
            if (news == null)
            {
                return NotFound();
            }
            return View(news);
        }

        [Authorize(Roles = "5")]
        public IActionResult Create()
        {
            News news = new News();
            var user =
serv.GetUser(serv.GetUserIdByName(User.Identity.Name));
            if (user != null)
            {
                news.Author = user.UserFullName;
                news.Date = DateTime.Now;
            }
            return View(news);
        }

        [HttpPost]
        public IActionResult Create(News news)
        {
            var user =
serv.GetUser(serv.GetUserIdByName(User.Identity.Name));
            if (ModelState.IsValid)
            {
                news.Author = user.UserFullName;
                news.Date = DateTime.Now;
                serviceNews.AddNews(news);
                NewsOperation newsOperations = new NewsOperation
{
                    Author = news.Author,
                    Title = news.Author,
                    Body = news.Body,
                    CreatedOn = news.Date,
                    NewsId = 1
                };

serviceNewsOperations.AddNewsOperation(newsOperations);
            }
```

```
            return RedirectToAction("Index", "News", null);
        }

        [Authorize(Roles = "5")]
        public IActionResult Delete(int? id)
        {
            if (id == null)
            {
                return BadRequest();
            }
            pmNews news = Map(serviceNews.Find(id));
            if (news == null)
            {
                return NotFound();
            }
            return View(news);
        }

        [HttpPost]
        public IActionResult Delete(News news)
        {
            if (ModelState.IsValid)
            {
                serviceNews.DeleteNews(news);
            }
            return RedirectToAction("Index", "News", null);
        }

        [Authorize(Roles = "5")]
        public IActionResult Edit(int? id)
        {
            if (id == null)
            {
                return BadRequest();
            }
            News news = serviceNews.Find(id);
            if (news == null)
            {
                return NotFound();
            }
            return View(news);
        }

        [HttpPost]
        public IActionResult Edit(News news)
        {
            var user =
serv.GetUser(serv.GetUserIdByName(User.Identity.Name));
            if (ModelState.IsValid)
            {
                news.Author = user.UserFullName;
                news.Date = DateTime.Now;
```

```
                serviceNews.UpdateNews(news);
            }
            return RedirectToAction("Index", "News", null);
        }
    }
```

ServiceNews:

```
    public class ServiceNews : MyServiceBase
    {
        public ServiceNews(TritonStudentContext DB) : base(DB) {
}

        public IEnumerable<News> GetNews() => db.News.Where(n =>
n.IsDeleted != true).OrderByDescending(n => n.Date);
        public News Find(int? id) => db.News.Find(id);
        public void AddNews(News news)
        {
            db.News.Add(new News
            {
                Title = news.Title,
                Author = news.Author,
                Date = news.Date,
                Body = news.Body,
            });
            db.SaveChanges();
        }
        public void DeleteNews(News news)
        {
            db.News.FirstOrDefault(n => n.Id ==
news.Id).IsDeleted = true;
            db.SaveChanges();
        }
        public void UpdateNews(News news)
        {
            db.News.FirstOrDefault(n => n.Id == news.Id).Date =
news.Date;
            db.News.FirstOrDefault(n => n.Id == news.Id).Author
= news.Author;
            db.News.FirstOrDefault(n => n.Id == news.Id).Body =
news.Body;
            db.News.FirstOrDefault(n => n.Id == news.Id).Title =
news.Title;
            db.SaveChanges();
        }

    }
```

- public IActionResult Create(News news) – метод приймає клас новина з полями новини це Id, заголовок (Title), автор (Author), час створення

новини (Date) та текст (Body). Id, автор та дата визначається автоматично, де автор визначається через авторизацію без можливості зміни на сайті, а дата через метод DateTime.Now(), який повертає час на даний момент. Далі у методі викликається метод AddNews(News news), що знаходиться у класі ServiceNews, який за допомогою Entity Framework Core працює з моделлю таблиці новин db.News і відповідно додає запис у базу даних та db.NewsOperations, куди додаються всі дії виконані з новиною для логування даних.

- public IActionResult Delete(int? id) – метод приймає id – ідентифікаційний номер запису новини у базі даних, яку потрібно видалити, далі перевіряє чи існує така новина через метод Find(int? id) класу ServiceNews, якщо існує, то за допомогою методу DeleteNews(News news) видаляє запис. Також у базу даних TritonStudentContext до таблиці News та відповідно моделі бази, класу – News додано поле IsDeleted типу bool для позначення видалених новин, якщо новина видалена, то значення IsDeleted відповідно true, дане поле враховується при відображені новин, власне таким чином жодна новина не видаляється.
- public IActionResult Edit(int? Id) – метод для виправлення новини, приймає id новини, яку потрібно виправити, аналогічно методу Delete() перевіряє чи існує новини, після звертається до ServiceNews, де перезаписує дані, також змінюючи дату та автора, який виконував виправлення.
- public IActionResult Index(int page = 1) – також внесено зміни у метод відображення всіх новин, оскільки було додано поле IsDeleted для видалення новин, тому у метод GetNews(), який викликався в Index, було додано перевірку значення IsDeleted.

2.6 Інтерфейс студентів

Якщо під час авторизації роль користувача відповідає ролі студента, то він отримає відповідний інтерфейс і функціонал, тому функціонал адміністратора для нього буде закритий.

2.6.1 Вільний вибір дисциплін

Вільний вибір дисциплін працює через DisciplineChangeController, є можливість обрати блок дисциплін користувачем, якщо такий вибір є у нього.

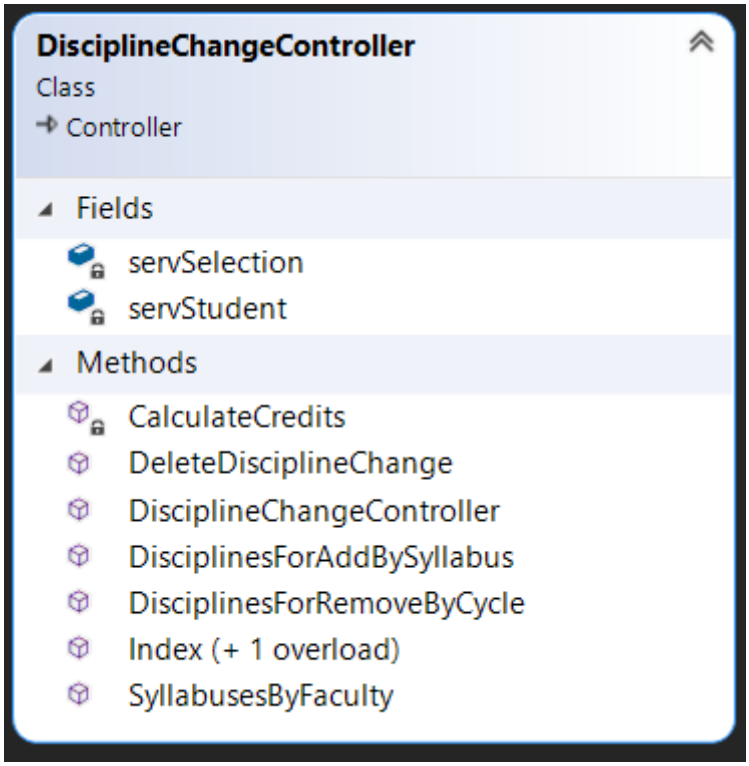

Рис.2.10. Діаграма класу DisciplineChangeController

CalculateCredits – приймає List<DisciplineChange>(ліст дисиплін) та повертає кількість кредитів ECTS відносно блоку дисциплін.

DeleteDisciplineChange – приймає id дисипліни та видаляє обраний вибір.

DisciplinesForAddBySyllabus – приймає id навчального плану і повертає блок дисциплін, який можна обрати.

DisciplinesForRemoveByCycle – приймає номер курсу та id користувача та видаляє обраний блок дисциплін.

SyllabusesByFaculty – приймає id факультета та повертає навчальний план.

2.6.2 Вибір дисциплін

Вибір дисциплін це вибір з переліку доступних блоків дисциплін через чекбокс. Якщо користувач вже вибрав блок йому відображаються усі можливі в його році блоки.

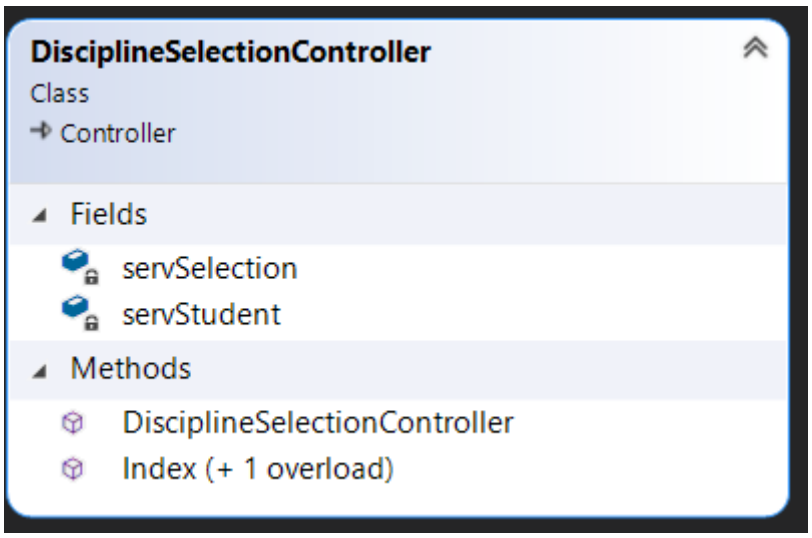

Рис.2.11. Діаграма класу DisciplineSelectionController

Index – перегружений екшн, що повертає інформацію про доступні блоки дисциплін відносно id користувача.

2.6.3 Вибір спеціалізацій

Вибір спеціалізації передбачає, що одночасно обираються всі дисципліни, які до неї включено.

DisciplinesBySpecialization – приймає id користувача та повертає доступні йому спеціалізації.

```
[Authorize]
public class SpecializationSelectionController : Controller
{
    ServiceStudentSelections servSelection;
    ServiceStudent servStudent;
```

```
        public SpecializationSelectionController(ServiceStudent
_servStudent, ServiceStudentSelections _servSelection)
        {
            servSelection = _servSelection;
            servStudent = _servStudent;
        }

        [Menu("Вибір", NodeId = "Selection", NodeName =
"Вибір")]
        [Menu("Вибір спеціалізації", ParentNodeId =
"Selection")]
        public IActionResult Index()
        {
            var pm = new
pmSpecialization(servStudent.GetUserIdByName(User.Identity.Name)
, servSelection, servStudent);
            return View(pm);

        }

        [HttpPost]
        public IActionResult Index(pmSpecialization pm)
        {
            if (ModelState.IsValid
                && ((pm.SpecializationId == -1 &&
servSelection.DeleteSpecializationSelection(pm.StudentId) == 1)
                    || (pm.SpecializationSelections.Where(s =>
(s.StatusTypeId == 0 || s.StatusTypeId == -1)).Count() != 0
                        &&
servSelection.SelectSpecialization(pm.StudentId,
pm.Specializations.Where(s => s.Id ==
pm.SpecializationId).FirstOrDefault(),
pm.SpecializationSelections.Where(s => (s.StatusTypeId == 0 ||
s.StatusTypeId == -1)).FirstOrDefault().Id) == 1)
                    ||
(servSelection.SelectSpecialization(pm.StudentId,
pm.Specializations.Where(s => s.Id ==
pm.SpecializationId).FirstOrDefault()) == 1)))
                return RedirectToAction("Success", "Home",
null);

            if (pm.SpecializationSelections == null)
pm.SpecializationSelections = new
List<SpecializationSelection>();
            return View(pm);
        }

        [HttpGet]
        public JsonResult DisciplinesBySpecialization(int id)
        {
```

```
            var user =
servSelection.GetUser(servStudent.GetUserIdByName(User.Identity.
Name));
            if (user.StudentId.HasValue)
                return
Json(servSelection.GetSpecializationDisciplines(id,
user.StudentId.Value).Select(t => new SelectListItem { Text =
t.Name }), new Newtonsoft.Json.JsonSerializerSettings());

            return null;
        }


    }
```

2.6.4 Навчальний план

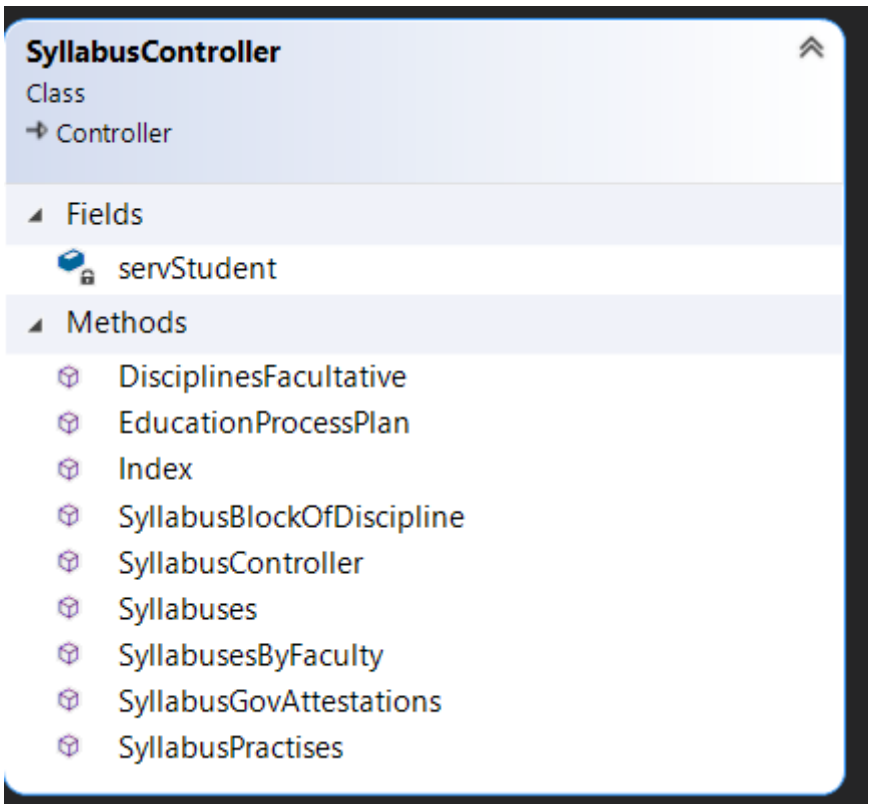

Рис.2.12. Діаграма класу SyllabusController

Навчальний план повертає метод EducationProcessPlan() класу SyllabusController, він приймає id навчального плану:

```
        public IActionResult EducationProcessPlan(int
syllabusId)
        {
            pmPlan pm = new pmPlan(syllabusId, servStudent);
            return View(pm);
        }
```

EducationProcessPlan() викликає конструктор класу pmPlan, що відображає усі дисципліни, блоки дисциплін, години та кредити ETCS кожної дисципліни студента.

2.6.5 Практики

Усі доступні практики отримуються через метод SyllabusPractises(), який приймає id навчального плану і викликає конструктор класу pmSyllabusPractises, який викликає метод GetSyllabusPractises(), що викликає процедуру на стороні TritonDatabase та повертає назву, курс, семестр та тривалість доступних практик.

SyllabusPractises:

```
        public IActionResult SyllabusPractises(int syllabusId)
        {
            pmSyllabusPractises pm = new
pmSyllabusPractises(syllabusId, servStudent);
            return View(pm);
        }
```

pmSyllabusPractises:

```
        public pmSyllabusPractises(int syllabusId,
ServiceStudent servStudent)
        {
            Disciplines =
servStudent.GetSyllabusPractises(syllabusId);
            var syllabus = servStudent.GetSyllabus(syllabusId);
            SyllabusName = syllabus.Name;

        }
```

GetSyllabusPractises:

```
        public List<SyllabusPractise> GetSyllabusPractises(int
syllabusId)
            => _triton.GetSyllabusPractises(syllabusId);
```

2.6.6 Державні атестації

Державні атестації відображаються через метод SyllabusGovAttestations(), який приймає id навчального плану і викликає конструктор класу pmSyllabusGovAttestations, який викликає метод

GetSyllabusGovAttestations(), що викликає процедуру на стороні TritonDatabase та повертає назву, курс, та семестр.

SyllabusGovAttestations:

```
        public IActionResult SyllabusGovAttestations(int
syllabusId)
        {
            pmSyllabusGovAttestations pm = new
pmSyllabusGovAttestations(syllabusId, servStudent);
            return View(pm);
        }
```

pmSyllabusGovAttestations:

```
        public pmSyllabusGovAttestations(int syllabusId,
ServiceStudent servStudent)
        {
            Disciplines =
servStudent.GetSyllabusGovAttestations(syllabusId);
            var syllabus = servStudent.GetSyllabus(syllabusId);
            SyllabusName = syllabus.Name;

        }
```

GetSyllabusGovAttestations:

```
        public List<SyllabusGovAttestation>
GetSyllabusGovAttestations(int syllabusId)
            => _triton.GetSyllabusGovAttestations(syllabusId);
```

2.6.7 Додавання номеру телефона

Додавання номеру телефону відбувається через екшн AddPhoneNumber, який приймає клас pmPhoneNumber, отримує id користувача через метод GetUserIdByName() та через метод AddPhoneNumber(), який приймає id користувача та номер телефону додає його до бази.

```
        [HttpPost]
        public IActionResult AddPhoneNumber(pmPhoneNumber pm)
        {
            if (ModelState.IsValid)
            {
```

```
                if
(servMembership.GetUser(servStudent.GetUserIdByName(User.Identit
y.Name)).PhoneNumber != null)
                {
                    if (pm.Password == null)
                        ModelState.AddModelError("Password",
"Поле є обов'язковим для заповненя.");
                    else if
(!servAD.ADAuthenticateUser(User.Identity.Name, pm.Password))
                        ModelState.AddModelError("Password",
"Неправильний пароль.");
                }
                if (ModelState.IsValid)
                {

servMembership.AddPhoneNumber(pm.PhoneNumber,
servStudent.GetUserIdByName(User.Identity.Name));
                    return RedirectToAction("Settings");
                }
            }
            return View(pm);
        }
```

2.6.8 Дата іспиту

Тепер в Тритоні також відображається час іспитів, якщо такий внесений в базу:

```
@markslist.ConsultationDate.Value.ToString("g",
CultureInfo.CreateSpecificCulture("es-ES"))
```

Висновки

Аналіз платформи .Net та порівняння продуктивності роботи версій .Net Core 3.1 і .Net 5 показав необхідність міграції на новішу платформу .Net 5, у результаті була виконана міграція на платформу .Net 5.

Після міграції були виправлені помилки роботи функціоналу сервісу та у зв’язку з неможливістю забезпечувати автентифікацію за старою схемою виконано перехід на LDAP автентифікацію.

Аналіз функціоналу інтерфейсу студентів та адміністраторів показав необхідність додавання нових функцій роботи сервісу та дозволив сформулювати вимоги для удосконаленого інтерфейсу, які були вирішені під час удосконалення функцій студентського інтерфейсу системи Triton.

Перелік використаних джерел

1) Performance improvements in .NET 5 [Електроний ресурс]
   URL: https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5/#gc
2) Introduction to .NET 5 [Електроний ресурс]
   URL: https://www.c-sharpcorner.com/article/introduction-of-net-5/
3) Astonishing performance of .NET 5 [Електроний ресурс]
   URL: https://medium.com/swlh/astonishing-performance-of-net-5-more-data-5cdc8d821e8c
4) What is new in .NET Core 3.1 [Електроний ресурс]
   URL: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-1
5) Breaking changes in .NET Core 3.1 [Електроний ресурс]
   URL: https://docs.microsoft.com/en-us/dotnet/core/compatibility/3.1?toc=/dotnet/fundamentals/toc.json&bc=/dotnet/breadcrumb/toc.json
6) Прайс М. C# 9 і .NET 5. Розробка і оптимізація, 2022.-832с.
7) Migrate from ASP.NET Core 3.1 to 5.0 [Електроний ресурс]
   URL: https://docs.microsoft.com/en-us/aspnet/core/migration/31-to-50?view=aspnetcore-6.0&tabs=visual-studio
8) Garbage collection [Електроний ресурс]
   URL: https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/
9) Fusion samples [Електроний ресурс]
   URL: https://github.com/servicetitan/Stl.Fusion.Samples
10) Garbage collection / allocation performance tests for C# / .NET and Go [Електроний ресурс]
   URL: https://github.com/alexyakunin/GCBurn
11) Breaking changes in .NET 5 [Електроний ресурс]

URL: https://docs.microsoft.com/en-us/dotnet/core/compatibility/5.0

12) Троелсен Е., Джепікс Ф. Мова програмування C# 9 та платформа .NET 5. Основні принципи та практики програмування. Том 1 (10-те видання), 2022.-770с.

13) Албахарі Б., Аблахарі Д., C# 9.0. Кишеньковий довідник., 2021.-256с.

14) Фрімен А. ASP.NET Core 3 з прикладами на C# для професіоналів, том 1, 8-е видання., 2021.-480с.

15) Ріхтер Д. CLR via C# програмування на платформі Microsoft .Net Framework 4.5 на мові C#, 4-те видання., 2013.-896с.

16) Скіт Д. C# для професіоналів. Тонкості програмування, 3-тє видання., 2014.-608с.

17) Migration From .Net Core 3.1 to 5.0— Real Project [Електроний ресурс]
URL: https://medium.com/swlh/migration-from-asp-net-core-3-1-to-5-0-real-project-69b1ff68328

18) Мартін Р. Чистий код. Створення, аналіз і рефакторинг. 2008.-368с.

19) What is .NET? An Overview of the Platform [Електроний ресурс]
URL: https://auth0.com/blog/what-is-dotnet-platform-overview/

20) gRPC performance improvements in .NET 5 [Електроний ресурс]
URL: https://devblogs.microsoft.com/dotnet/grpc-performance-improvements-in-net-5/

21) .NET Core vs Framework [Електроний ресурс]
URL: https://habr.com/ru/post/481558/

22) Announcing .NET 5.0 [Електроний ресурс]
URL: https://devblogs.microsoft.com/dotnet/announcing-net-5-0/

Додатки

Додаток А

Посилання на Gitlab репозиторій TritonStudent:

https://gl.icc.knu.ua/triton/triton-student-core

Додаток Б

Project name: ADHelpersLib

ADHelper:

```
    public class ADHelper : IADHelper
    {
        private LdapDirectoryIdentifier ldapDirectory;
        private NetworkCredential ldapCredentials;
        private LdapConnection ldapConnection;
        private string ldapServer;
        private string baseDN;
        private string netbiosDomainName;
        private string dnsRootDomainName;

        public ADHelper(string connectionString)
        {
            var csb = new DbConnectionStringBuilder();
            csb.ConnectionString = connectionString;
            ldapServer = csb["Server"].ToString();
            ldapDirectory = new LdapDirectoryIdentifier(ldapServer,
636);

            string username = csb.ContainsKey("User Id") ? csb["User
Id"].ToString() : null;
            string password = csb.ContainsKey("Password") ?
csb["Password"].ToString() : null;
            ldapCredentials = new NetworkCredential(username,
password);

            ldapConnection = OpenLdapConnection(ldapCredentials);

            var rootDSE = GetRootDSE();
            baseDN = csb.ContainsKey("Base DN") ? csb["Base
DN"].ToString() :
rootDSE.Attributes["defaultNamingContext"][0].ToString();

            netbiosDomainName = csb.ContainsKey("Netbios Domain Name")
? csb["Netbios Domain Name"].ToString() : null;
            dnsRootDomainName = csb.ContainsKey("DNS Root Domain") ?
csb["DNS Root Domain"].ToString() : null;

            if(netbiosDomainName == null || dnsRootDomainName == null)
            {
                string configurationNamingContext =
rootDSE.Attributes["configurationNamingContext"][0].ToString();
                string partitionsDN =
string.Format("cn=Partitions,{0}", configurationNamingContext);
                string filter =
string.Format("(&(objectcategory=Crossref)(nCName={0})(nETBIOSName=*))
", baseDN);
                var result = LdapSearchOne(partitionsDN, filter,
SearchScope.OneLevel, "nETBIOSName", "dnsRoot");
                if (result != null)
                {
```

```
                    if (netbiosDomainName == null)
                        netbiosDomainName =
result.Attributes["nETBIOSName"][0].ToString();
                    if (dnsRootDomainName == null)
                        dnsRootDomainName =
result.Attributes["dnsRoot"][0].ToString();
                }
            }
        }

        private LdapConnection OpenLdapConnection(NetworkCredential
cred = null)
        {
            if (cred == null)
                cred = ldapCredentials;
            LdapConnection ldapConnection;
            ldapConnection = new LdapConnection(ldapDirectory, cred,
AuthType.Basic);
            ldapConnection.SessionOptions.AutoReconnect = true;
            ldapConnection.SessionOptions.PingKeepAliveTimeout = new
TimeSpan(0, 0, 60);
            ldapConnection.SessionOptions.SecureSocketLayer = true;
            ldapConnection.SessionOptions.VerifyServerCertificate =
new VerifyServerCertificateCallback((ldapcon, cer) => { return true;
});
            ldapConnection.SessionOptions.ProtocolVersion = 3;
            ldapConnection.Bind();
            return ldapConnection; //Authentication succeeded
        }

        private SearchResultEntryCollection LdapSearch(string baseDN,
string filter, SearchScope scope, params string[] attributes)
        {
            var searchRequest = new SearchRequest(baseDN, filter,
scope, attributes);
            var response = ldapConnection.SendRequest(searchRequest)
as SearchResponse;
            return response.Entries;
        }

        private SearchResultEntry LdapSearchOne(string baseDN, string
filter, SearchScope scope, params string[] attributes)
        {
            var result = LdapSearch(baseDN, filter, scope,
attributes);
            return (result.Count == 0) ? null : result[0];
        }

        private SearchResultEntry GetRootDSE()
            => LdapSearchOne(null, "(&(objectClass=*))",
SearchScope.Base);

        private bool LdapDeleteEntry(string dn)
        {
            var deleteRequest = new DeleteRequest(dn);
            var response = ldapConnection.SendRequest(deleteRequest);
            return (response.ResultCode == ResultCode.Success);
```

```
        }

        private static readonly ReadOnlyDictionary<string, string>
LdapFilterEscapes
            = new ReadOnlyDictionary<string, string>(
                new Dictionary<string, string>()
                {
                    { "*",  "\\2a" },
                    { "(",  "\\28" },
                    { ")",  "\\29" },
                    { "/",  "\\2f" },
                    { "\\", "\\5c" },
                    { "\0", "\\00" }
                }
            );
        private static string EscapeLDAPFilterValue(string value)
        {
            foreach (var rl in LdapFilterEscapes)
            {
                value = value.Replace(rl.Key, rl.Value);
            }
            return value;
        }

        private SearchResultEntry FindEntryByDN(string
distinguishedName, params string[] extraAttributes)
            => LdapSearchOne(distinguishedName, "(&(objectClass=*))",
SearchScope.Base, extraAttributes);

        private SearchResultEntry FindByUniqueAttribute(string
objectClass, string attrName, string attrValue, params string[]
extraAttributes)
        {
            List<string> attributes = new List<string>
            {
                attrName,
                "distinguishedName",
            };
            if (extraAttributes != null)
            {
                attributes.AddRange(extraAttributes);
            }
            string filter =
string.Format("(&(objectClass={0})({1}={2}))", objectClass,
EscapeLDAPFilterValue(attrName), EscapeLDAPFilterValue(attrValue));
            return LdapSearchOne(baseDN, filter, SearchScope.Subtree,
attributes.ToArray());
        }

        private SearchResultEntry FindUserBySAMAccountName(String
username, params string[] extraAttributes)
            => FindByUniqueAttribute("user", "sAMAccountName",
username, extraAttributes);

        private SearchResultEntry FindGroupByName(String group)
            => FindByUniqueAttribute("group", "name", group,
"member");
```

```
        public bool AuthenticateUser(string username, string password)
        {
            var de = FindUserBySAMAccountName(username);
            if (de == null) // no such user
                return false;
            try
            {
                var userDN = de.DistinguishedName;
                var ldapConnection = OpenLdapConnection(new
NetworkCredential(userDN, password));
            }
            catch (LdapException)
            {
                //Authentication failed, exception will dictate why
                return false;
            }
            return true; //Authentication succeeded
        }

        public string GetNetbiosLoginName(string username)
        {
            var de = FindUserBySAMAccountName(username);
            if (de == null) // no such user
                return null;
            return string.Format("{0}\\{1}", netbiosDomainName,
de.Attributes["sAMAccountName"][0].ToString());
        }

        public bool IsLoginFree(string username)
        {
            var de = FindUserBySAMAccountName(username);
            return (de == null);
        }

        public string GetUserAttribute(string username, string
attribute)
        {
            var de = FindUserBySAMAccountName(username, new string[] {
attribute });
            if (de == null)
                return null;
            if(de.Attributes.Contains(attribute))
                return de.Attributes[attribute][0].ToString();
            return null;
        }

        private bool AddAttributeValue(string dn, string attribute,
string value)
        {
            var request = new ModifyRequest(dn,
DirectoryAttributeOperation.Add, attribute, value);
            var response = ldapConnection.SendRequest(request);
            return (response.ResultCode == ResultCode.Success);
        }
```

```
        private bool DeleteAttributeValue(string dn, string attribute,
string value)
        {
            var request = new ModifyRequest(dn,
DirectoryAttributeOperation.Delete, attribute, value);
            var response = ldapConnection.SendRequest(request);
            return (response.ResultCode == ResultCode.Success);
        }

        public bool SetUserAttribute(string username, string
attribute, string value)
        {
            var de = FindUserBySAMAccountName(username, new string[] {
attribute });
            try
            {
                bool hadValue = de.Attributes.Contains(attribute);
                bool addValue = (value != null);
                ModifyRequest request = null;
                if (hadValue && addValue)
                {
                    request = new ModifyRequest(de.DistinguishedName,
DirectoryAttributeOperation.Replace, attribute, value);
                }
                else if(hadValue && !addValue)
                {
                    request = new ModifyRequest(de.DistinguishedName,
DirectoryAttributeOperation.Delete, attribute);
                }
                else if(!hadValue && addValue)
                {
                    request = new ModifyRequest(de.DistinguishedName,
DirectoryAttributeOperation.Add, attribute, value);
                }

                if(request != null)
                {
                    var response =
ldapConnection.SendRequest(request);
                    return (response.ResultCode ==
ResultCode.Success);
                }
                return true;
            }
            catch
            {
                return false;
            }
        }

       public bool DeleteUserAttribute(string username, string
attribute)
            => SetUserAttribute(username, attribute, null);

        private static byte[] GetPasswordData(string password)
        {
            string formattedPassword;
```

```
            formattedPassword = String.Format("\"{0}\"", password);
            return (Encoding.Unicode.GetBytes(formattedPassword));
        }

        private bool SetUserPassword(string userDN, string password,
LdapConnection theLdapConnection = null)
        {
            if (theLdapConnection == null)
                theLdapConnection = ldapConnection;
            var request = new ModifyRequest(userDN,
DirectoryAttributeOperation.Replace, "unicodePwd",
GetPasswordData(password));
            var response = theLdapConnection.SendRequest(request);
            return (response.ResultCode == ResultCode.Success);
        }

        private bool ChangePassword(string userDN, string oldPassword,
string newPassword, LdapConnection theLdapConnection = null)
        {
            if (theLdapConnection == null)
                theLdapConnection = ldapConnection;

            var deleteMod = new DirectoryAttributeModification();
            deleteMod.Name = "unicodePwd";
            deleteMod.Add(GetPasswordData(oldPassword));
            deleteMod.Operation = DirectoryAttributeOperation.Delete;

            var addMod = new DirectoryAttributeModification();
            addMod.Name = "unicodePwd";
            addMod.Add(GetPasswordData(newPassword));
            addMod.Operation = DirectoryAttributeOperation.Add;

            var request = new ModifyRequest(userDN, deleteMod,
addMod);
            var response = theLdapConnection.SendRequest(request);
            return (response.ResultCode == ResultCode.Success);
        }

        public bool ResetUserPassword(string username, string
newPassword)
        {
            var de = FindUserBySAMAccountName(username);
            if (de == null) // no such user
                return false;
            try
            {
                return SetUserPassword(de.DistinguishedName,
newPassword);
            }
            catch
            {
                return false;
            }
        }

        public bool ChangeUserPassword(string username, string
oldPassword, string newPassword)
```

```
        {
            var de = FindUserBySAMAccountName(username);
            if (de == null) // no such user
                return false;
            try
            {
                var userDN = de.DistinguishedName;
                var ldapConnection = OpenLdapConnection(new
NetworkCredential(userDN, oldPassword));
                return ChangePassword(de.DistinguishedName,
oldPassword, newPassword, ldapConnection);
            }
            catch
            {
                return false;
            }
        }

        public List<string> GetUserMemberGroups(string username)
        {
            var de = FindUserBySAMAccountName(username, "memberOf");
            if (de == null) // no such user
                return null;
            var result = new List<string>();
            if (!de.Attributes.Contains("memberOf"))
                return result;

            for(int i = 0; i < de.Attributes["memberOf"].Count; i++)
            {
                var gr =
FindEntryByDN(de.Attributes["memberOf"][i].ToString());

                if(gr != null &&
!result.Contains(gr.Attributes["name"][0].ToString()))
                    result.Add(gr.Attributes["name"][0].ToString());
            }
            return result;
        }
        public bool IsUserInGroup(string username, string group_name)
        {
            var groups = GetUserMemberGroups(username);
            return
                groups.Contains(group_name,
StringComparer.InvariantCultureIgnoreCase);
        }

        public bool AddUserToGroup(string username, string group_name)
        {
            if (IsUserInGroup(username, group_name))
                return true;
            try
            {
                var user = FindUserBySAMAccountName(username);
                var group = FindGroupByName(group_name);
                return AddAttributeValue(group.DistinguishedName,
"member", user.DistinguishedName);
            }
```

```
            catch
            {
                return false;
            }
        }
        public bool RemoveUserFromGroup(string username, string
group_name)
        {
            if (!IsUserInGroup(username, group_name))
                return true;
            try
            {
                var user = FindUserBySAMAccountName(username);
                var group = FindGroupByName(group_name);
                return DeleteAttributeValue(group.DistinguishedName,
"member", user.DistinguishedName);
            }
            catch
            {
                return false;
            }
        }

       public bool UserHasMailbox(string username)
        {
            var user = FindUserBySAMAccountName(username,
"msExchRecipientTypeDetails");
            if (user == null)
                return false;
            if
(!user.Attributes.Contains("msExchRecipientTypeDetails"))
                return false;
            return
(int)(user.Attributes["msExchRecipientTypeDetails"][0]) == 1;
        }

        public string GetUserMailboxAddress(string username)
        {
            var user = FindUserBySAMAccountName(username, "mail");
            if (user == null)
                return null;
            if (!user.Attributes.Contains("mail"))
                return null;
            return user.Attributes["mail"][0].ToString();
        }

        public KeyValuePair<string, string>?
GetDistributionGroup(string alias)
        {
            if (!alias.Contains('@'))
                alias += "@" + dnsRootDomainName;
            try {
                var de = FindByUniqueAttribute("group",
"proxyAddresses", "SMTP:" + alias, "name", "mail");
                return new KeyValuePair<string,
string>(de.Attributes["name"][0].ToString(),
de.Attributes["mail"][0].ToString());
```

```
            }
            catch
            {
                return null;
            }
        }

        public bool DeleteDistributionGroup(string alias)
        {
            if (!alias.Contains('@'))
                alias += "@" + dnsRootDomainName;
            try
            {
                var de = FindByUniqueAttribute("group",
"proxyAddresses", "SMTP:" + alias);
                return LdapDeleteEntry(de.DistinguishedName);
            }
            catch
            {
                return false;
            }
        }

        public bool CreateDistributionGroup(string alias, string name,
string ouPath)
        {
            if (!alias.Contains('@'))
                alias += "@" + dnsRootDomainName;
            var alias_nick = alias.Split('@')[0];
            try
            {
                var de = FindByUniqueAttribute("group",
"proxyAddresses", "SMTP:" + alias);
                if(de != null)
                    return true;

                var addReq = new
AddRequest(string.Format("cn={0},{1},{2}", name, ouPath, baseDN),
"group");
                addReq.Attributes.Add(new DirectoryAttribute("cn",
name));
                addReq.Attributes.Add(new DirectoryAttribute("name",
name));
                addReq.Attributes.Add(new DirectoryAttribute("mail",
alias));
                addReq.Attributes.Add(new
DirectoryAttribute("proxyAddresses", "SMTP:" + alias));
                addReq.Attributes.Add(new
DirectoryAttribute("mailNickname", alias_nick));
                addReq.Attributes.Add(new
DirectoryAttribute("groupType", /* GROUP_TYPE_UNIVERSAL_GROUP */
0x8));
                addReq.Attributes.Add(new
DirectoryAttribute("sAMAccountName", alias_nick));
                addReq.Attributes.Add(new
DirectoryAttribute("sAMAccountType", /* SAM_NON_SECURITY_GROUP_OBJECT
*/ 0x10000001));
```

```
                addReq.Attributes.Add(new
DirectoryAttribute("msExchRequireAuthToSendTo", true));
                addReq.Attributes.Add(new
DirectoryAttribute("reportToOriginator", true));

                var response = ldapConnection.SendRequest(addReq);
                return (response.ResultCode == ResultCode.Success);
            }
            catch
            {
                return false;
            }
        }

        public bool CreateUserAccount(string username, string
password, string displayName = null, string ouPath = "ou=Users")
        {
            try
            {
                var userDN = string.Format("cn={0},{1},{2}", username,
ouPath, baseDN);
                var addReq = new AddRequest(userDN, "user");
                //addReq.Attributes.Add(new
DirectoryAttribute("objectClass", "user"));
                addReq.Attributes.Add(new DirectoryAttribute("cn",
username));
                addReq.Attributes.Add(new DirectoryAttribute("name",
username));
                addReq.Attributes.Add(new
DirectoryAttribute("userPrincipalName", username + "@" +
dnsRootDomainName));
                addReq.Attributes.Add(new
DirectoryAttribute("displayName", displayName ?? username));
                addReq.Attributes.Add(new
DirectoryAttribute("sAMAccountName", username));

                var response = ldapConnection.SendRequest(addReq);
                if (response.ResultCode != ResultCode.Success)
                    return false;

                if (!SetUserPassword(userDN, password))
                {
                    LdapDeleteEntry(userDN);
                    return false;
                }

                var de = FindEntryByDN(userDN, "userAccountControl");
                int uac =
int.Parse(de.Attributes["userAccountControl"][0].ToString());
                uac &= ~0x0002 /* ACCOUNTDISABLE */;
                uac |= /* NORMAL_ACCOUNT */ 0x0200 | /*
DONT_EXPIRE_PASSWORD */ 0x10000;

                var modReq = new ModifyRequest(userDN);
                var dam1 = new DirectoryAttributeModification();
                dam1.Name = "lockOutTime";
                dam1.Operation = DirectoryAttributeOperation.Replace;
```

```
                dam1.Add("0");
                modReq.Modifications.Add(dam1);

                var dam2 = new DirectoryAttributeModification();
                dam2.Name = "userAccountControl";
                dam2.Operation = DirectoryAttributeOperation.Replace;
                dam2.Add(uac.ToString());
                modReq.Modifications.Add(dam2);

                response = ldapConnection.SendRequest(modReq);
                if (response.ResultCode != ResultCode.Success)
                {
                    LdapDeleteEntry(userDN);
                    return false;
                }
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
```

ADHelperFactory:

```
    public class ADHelperFactory
    {
        private static Dictionary<string, IADHelper> adHelpers = new
Dictionary<string, IADHelper>();
        public static IADHelper CreateADHelper(string
connectionString)
        {
            lock(adHelpers)
            {
                if(!adHelpers.ContainsKey(connectionString))
                {
                    adHelpers.Add(connectionString, new
ADHelper(connectionString));
                }
                return adHelpers[connectionString];
            }
        }
    }
```

IADHelper:

```
   public interface IADHelper
    {
        bool CreateUserAccount(string username, string password,
string displayName = null, string ouPath = "ou=Users");
```

```
        bool IsLoginFree(string user_login);
        bool ResetUserPassword(string username, string new_password);
        bool ChangeUserPassword(string username, string oldPassword,
string newPassword);
        bool IsUserInGroup(string username, string group_name);
        bool AddUserToGroup(string username, string group_name);
        bool RemoveUserFromGroup(string username, string group_name);
        bool AuthenticateUser(string username, string password);
        string GetNetbiosLoginName(string username);
        string GetUserAttribute(string username, string attribute);
        bool SetUserAttribute(string username, string attribute,
string value);
        bool DeleteUserAttribute(string username, string attribute);
        bool UserHasMailbox(string username);
        string GetUserMailboxAddress(string username);
        KeyValuePair<string, string>? GetDistributionGroup(string
alias);
        bool CreateDistributionGroup(string alias, string name, string
ouPath);
        bool DeleteDistributionGroup(string alias);
    }
```

Додаток В

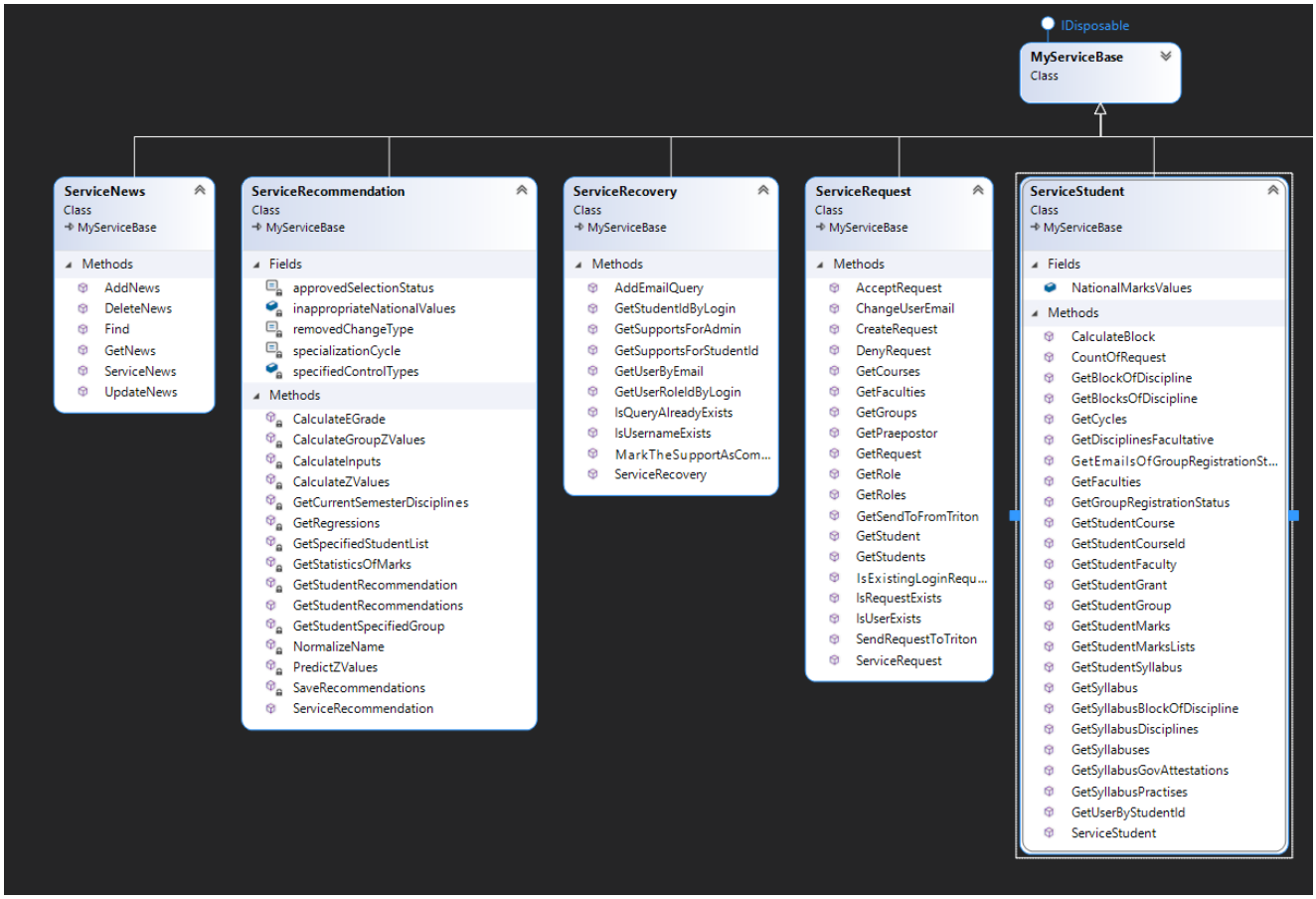

Рис. Діаграма класів сервісів TritonStudentModel проекту

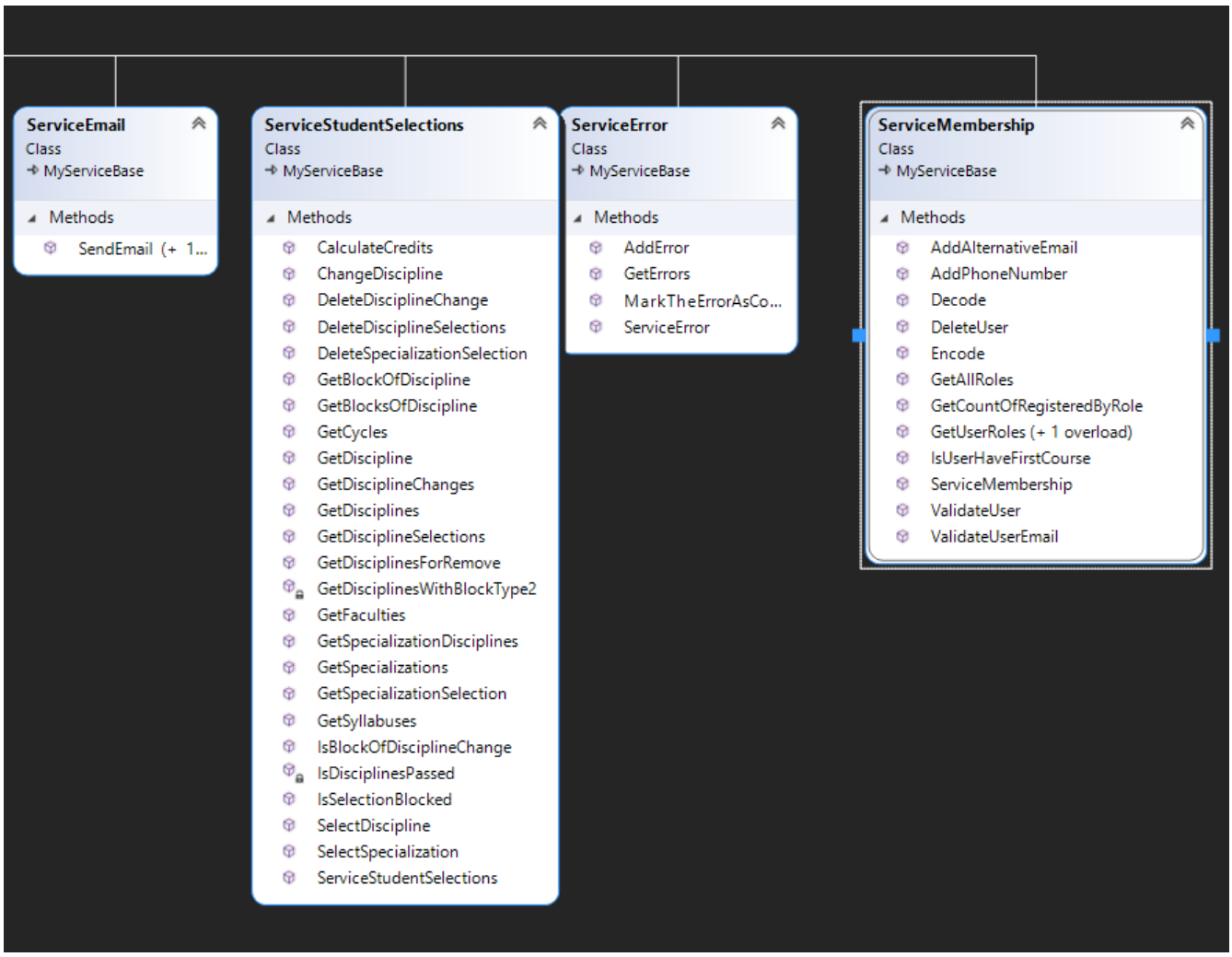

Рис. Діаграма класів сервісів TritonStudentModel проекту

Додаток Г

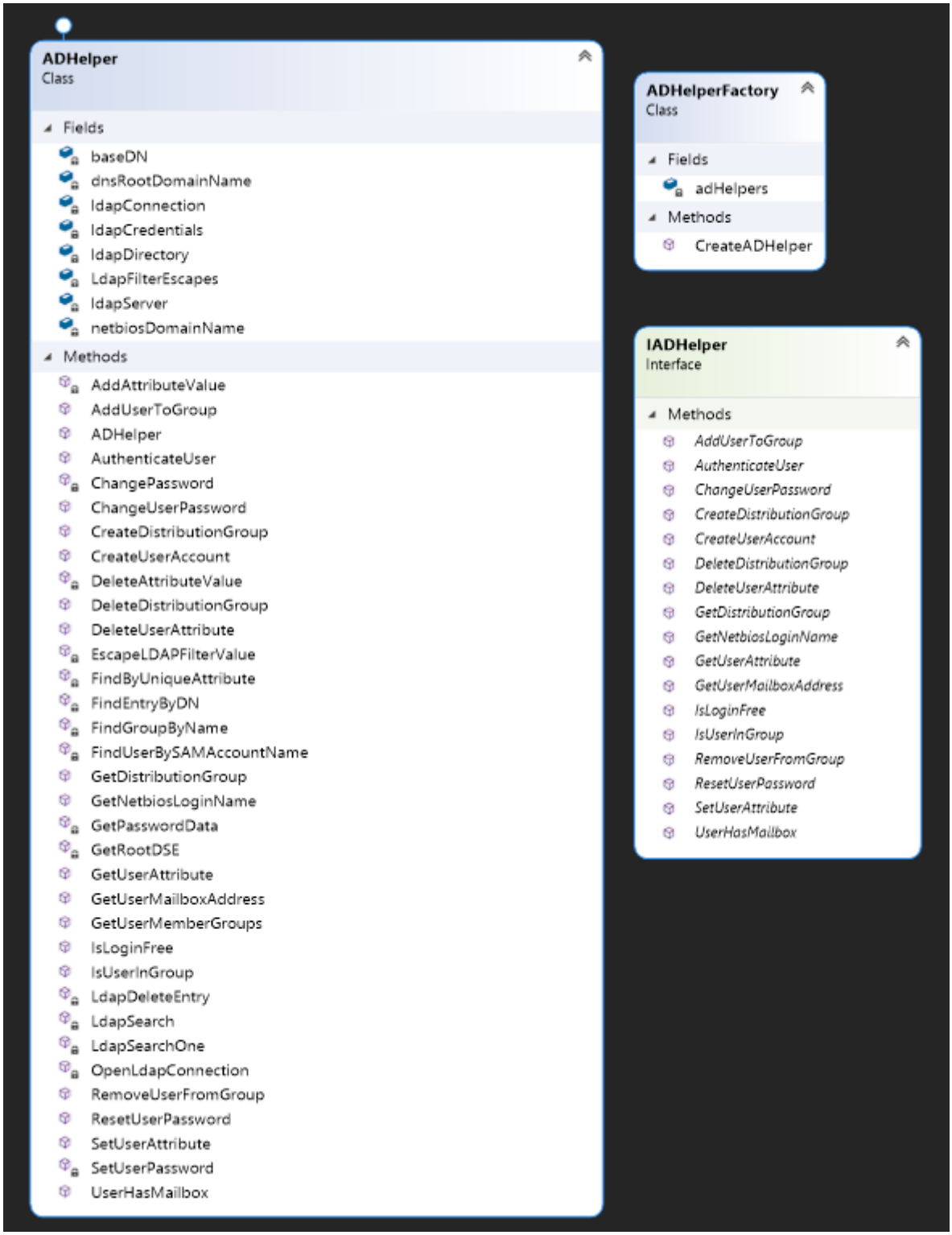

Рис. Діаграма ADHelpersLib – бібліотека, що використовується для LDAP аутентифікації