МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Платформа для дистанційного навчання з використанням Telegram Bot API»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шовкопляс О.А.**

Студента групи ІН – 72 **Кончатний В.В.**

СУМИ 202

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-72 спеціальності "Інформатика" денної форми навчання Кончатного Віталія Володимировича.

Тема: " Платформа для дистанційного навчання з використанням Telegram Bot API "

Затверджена наказом по СумДУ

№_____ від _____2021 г.

Зміст пояснювальної записки: 1) Інформаційний огляд. Огляд проблемної області. Огляд подібних рішень. Актуальність створення чат-бота. Постановка задачі. 2) Вибір методу рішення. Вибір програмної реалізації. Паттерн. Вибір бібліотеки для взаємодії з Telegram. 3) Практична реалізація. Інформаційна модель. Програмна реалізація.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Шовкопляс О.А.

Завдання приняв до виконання _____ Кончатний В.В.

РЕФЕРАТ

Записка: 78 стор., 17 рис., 1 додаток, 16 джерел.

Об’єкт дослідження — проблема організації віддаленого навчального процесу.

Мета роботи — розробка платформи для організації процесу віддаленого навчання з використанням технологій чат-ботів для месенджеру Telegram.

Методи дослідження — методи дослідження аудиторії та порівняння подібних сервісів.

Результати — розроблено програмне забезпечення для розвертання системи віддаленого або змішаного навчання. При цьому, для зручності та спрощення взаємодії з платформою розроблено API та чат-бот який його використовує. Розроблене програмне забезпечення реалізовано за допомогою таких мов програмування та фреймворків: ASP .Net, C#, JavaScript, Python. Серверна частина побудована з використанням паттерну MVC.

ВІДДАЛЕНЕ НАВЧАННЯ, НАВЧАЛЬНИЙ ПРОЦЕС,
НАВЧАЛЬНА ПЛАТФОРМА, ЧАТ-БОТ, API, C#, ASP .NET,
PYTHON, MVC.

ЗМІСТ

ВСТУП

В сучасних умовах карантинних обмежень сфера освіти потребує значної кількості технічних засобів які дозволяють організувати навчальний процес віддалено. Основними аспектами процесу перенесення навчання в віддалений формат є простота рішення, зрозумілість як для викладачів так і для студентів та індивідуалізація по відношенню до студента.

Задача організації віддаленого навчального процесу потребує специфічних програмних засобів які дозволяють організувати спілкування між викладачем та групою студентів та можливість здавати та перевіряти роботи, мати доступ до результативності навчання. Користувач який є студентом повинен вчасно отримувати інформацію від викладача та мати можливість завжди перевірити список завдань та запланованих лекцій, щоб мати можливість підготуватися до заняття. Також у студента має бути можливість зв’язку з викладачем, що дозволяє вирішити питання пов’язані з завданнями або навчальним планом. Для викладача повинні бути реалізовані функції управління навчальною групою, створення завдань, створення оголошень, методичних матеріалів для занять, тощо.

Також повинні використовуватися сучасні можливості в комунікації та інтеграції з іншими сервісами зручними для користувачів, це забезпечить просту авторизацію та зручність використання готового програмного продукту. З масовим переходом на дистанційне навчання більшість спілкування між людьми перейшла в месенджери та засоби для організації онлайн конференцій, тому доцільно використовувати можливості доповнення ними застосунку за допомогою їх API та методів для інтеграції. Також доцільно використовувати технології які дозволяють достатньо швидко та дешево розвернути інформаційну систему, та не витрачати на її підтримку занадто великий бюджет. Для зберігання інформації краще використати сторонні додатки для збереження інформації, це рішення дозволяє значно зменшити розмір розвернутого додатку і відповідно зменшити витрати на хостинг та сховище

1 ІНФОРМАЦІЙНІЙ ОГЛЯД

1.1 Огляд проблемної області

У світі прослідковується глобальна тенденція освітніх закладів до переведення в формат змішаного та віддаленого навчання. Це обумовлює велику кількість проблем пов’язаних з технічними можливостями та інтеграціями між сервісами для створення екосистеми змішаного навчання [1]. Також перехід на технології змішаного навчання привносить в навчальний процес ряд позитивних аспектів таких як: можливість розширити набір курсів та забезпечити можливість вибору студентом індивідуального навчального плану, спрощення навчання для студентів які через роботу або інші причини не можуть повноцінно відвідувати заняття, полегшення роботи для викладача через використання систем для автоматизації повсякденних дій (ведення журналу, зручне відслідковування успіхів студентів тощо).

Особливої актуальності проблема набуває в сфері навчання фахівців в області інформаційних технологій, адже саме тут легше всього впровадити таку систему через специфіку процесу навчання студентів таких спеціальностей, і водночас необхідність змішаного навчання для IT-фахівців підвищена через те, що більшість завдань та тем які вони розглядають в процесі потребують технічних засобів.

Основним завданням тут є розширене використання інтеграцій щоб створити безшовну платформу яка буде відповідати всім потребам як студентів так і викладачів. Інтеграції важливі на усіх рівнях, адже саме це й створює інформаційну систему навчального закладу. Нажаль для більшості навчальних закладів задача створення системи онлайн-навчання є непід’ємною оскільки це потребує великих затрат на серверні потужності або ліцензійне програмне забезпечення. Через що і розкривається актуальність створення спеціалізованого програмного забезпечення для невеликих освітніх закладів яким потрібне просте рішення для контролю над процесом віддаленого або змішаного навчання яке б

дозволяло швидко розвернути життєздатну онлайн систему та не потребувало б багато часу на адаптацію до робочих процесів закладу.

Загалом такі технічні засоби можна використовувати також для змішаного навчання, тобто такого, що передбачає роботу студентів як очно так за допомогою технічних засобів, як онлайн платформи. Змішане навчання в своїй реалізації потребує керівництва викладача під час взаємодії в аудиторії та безпосередньо роботи в онлайн-середовищі. Така практика дозволяє не замінювати аудиторні заняття, а ефективно їх розширювати за рахунок використання технічних засобів, наприклад тестування, або робота з вправами та тренажерами освітньої платформи. Змішане навчання привносить в навчальний процес фактор самоконтролю свого часу студентом, темпу роботи, та обсягу вивчення теми.

Запровадження новітніх технологій в будь-які процеси завжди супроводжується безліччю проблем які потрібно вирішити. Такими проблемами в сфері навчання можуть бути як технічні так і організаційно-методичні. Змішане навчання відкидає традиційну форму та пропонує новий погляд на процес навчання студентів та набування ними досвіду. Окрім викладацького досвіду, матеріально-методичної бази та технічних можливостей для створення курсу який використовує ідею змішаного навчання, також потрібний ряд фахівців в своїх сферах таких як: науковці, методисти, психологи, адміністратори, програмісти.

Навчання змішаним методом надає більше простору в сфері, що дозволяє забезпечити кращі умови для досягнення високих показників навчальних результатів, це обумовлено:

- ширшим використанням інформаційних сервісів навчального закладу;
- доступністю якісних навчальних матеріалів які розміщуються на навчальній платформі;
- використанням інструментів для організації дистанційної роботи [2];

- взаємодією студентів один з одним та з викладачами за допомогою технічних засобів;
- опрацюванням інформаційних ресурсів більш цілеспрямовано через доступність.

Процес для реалізації ідеї змішаного навчання передбачає створення або наявність комфортного та інтегрованого навчального інформаційного середовища. Такою системою і може виступати тема яка розглядається в цій роботі. Створення такої системи дозволяє впроваджувати в навчальний процес нові практики або швидко перевести навчання в віддалений формат. Набутий в результаті досвід в створенні такої системи дозволить в майбутньому розвинути ідею та на основі набутого досвіду про побудову та проблеми таких систем. Також це дозволить розширювати існуючі системи шляхом введення нового функціоналу та створення нових інтеграцій.

1.2 Огляд подібних рішень

На даний момент в інформаційному просторі існує доволі велика кількість сервісів які дозволяють організувати процес віддаленого навчання. Завдяки таким сервісам стає можливим дистанційне навчання для людей які не можуть перебувати на очному навчанні або в випадках коли неможливо проводити традиційне навчання.

Перед розробкою сервісу був проведений огляд вже існуючих ресурсів, таких як:

- classroom.google.com;
- mix.sumdu.edu.ua;
- moodle.org.

Кожен із вище приведених сервісів має свої особливості, які виділяють його від інших.

Наприклад, classroom.google.com має зручну систему авторизації за допомогою сервісів Google та простий, зрозумілий для користувача інтерфейс.

Завдяки якому можна легко виконати потрібну дію та отримати потрібну інформацію. Сервіс окремо показує події які будуть незабаром, наприклад нагадування про дедлайн або нове завдання. Всі події дублюються на пошту що інколи викликає надто велику кількість електронних листів.

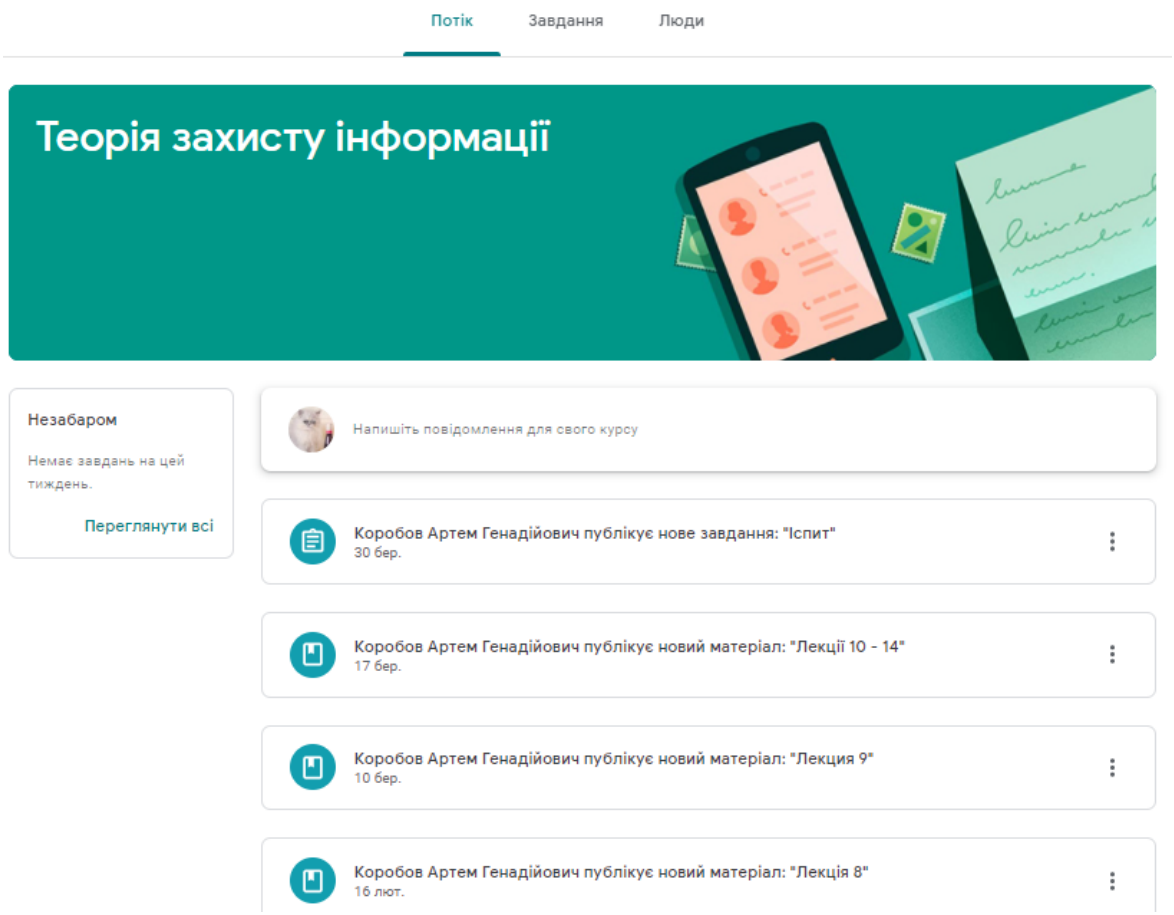

Рисунок 1.1 — Приклад курсу на classroom.google.com

В свою чергу mix.sumdu.edu.ua дещо відрізняється від classroom.google.com тим, що цей сервіс використовується тільки для студентів одного університету і не націлений на широку аудиторію, тому в ньому реалізовані специфічні функції потрібні для організації змішаного навчання саме для СумДУ. Також варто зазначити, що він доповнює екосистему університету і повноцінно працює тільки в зв’язці з іншими сервісами, але це виступає плюсом оскільки цей сервіс дозволяє працювати з іншими сервісами університету за допомогою інтеграцій [3]. В порівнянні з сервісом від Google, Mix має авторизацію через особистий кабінет університету, інколи ця функція працює

нестабільно. Не зважаючи на це сервіс надає можливість організувати навчальний процес і дозволяє не тільки здавати роботи, а також реалізує можливість організації тестування студентів та проведення екзаменів в дистанційному форматі. Нажаль повідомлення про дії на ресурсі надходять тільки на сторінку на самому сайті тому отримувати актуальні новини не так зручно як у попередньому випадку.

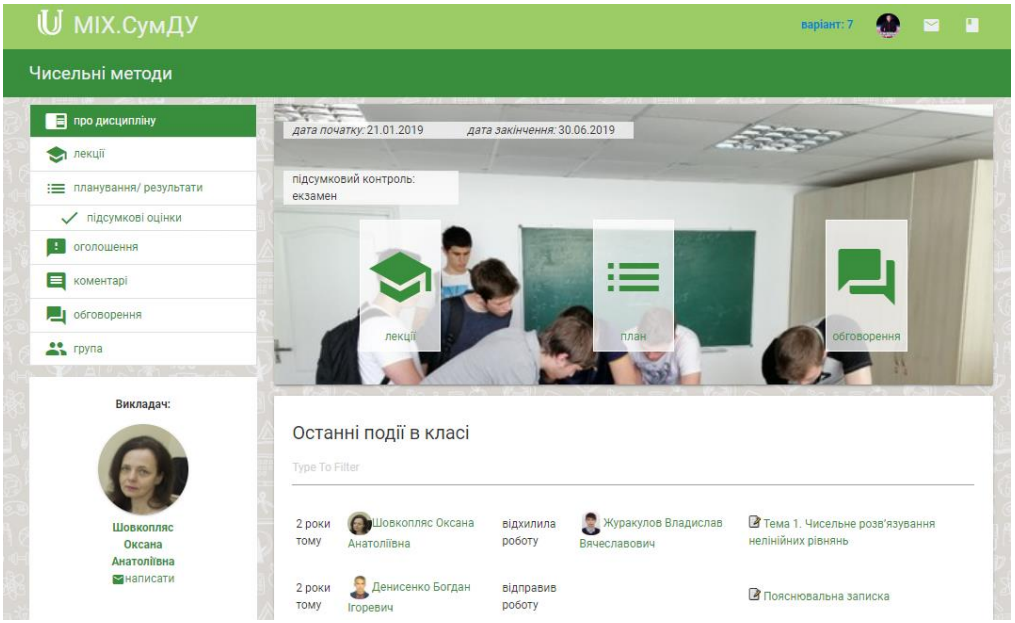

Рисунок 1.2 — Сторінка сервісу mix.sumdu.edu.ua

Що до moodle.org варто відмітити, що сервіс надає програмний продукт для розвертання та налаштування своєї системи віддаленого навчання. Тобто сам Moodle є рушієм для створення систем для віддаленого навчання. Цей сервіс реалізує основні функції для обміну інформації між студентом та викладачем, та дозволяє встановлювати додаткові плагіни для розширення функцій платформи.

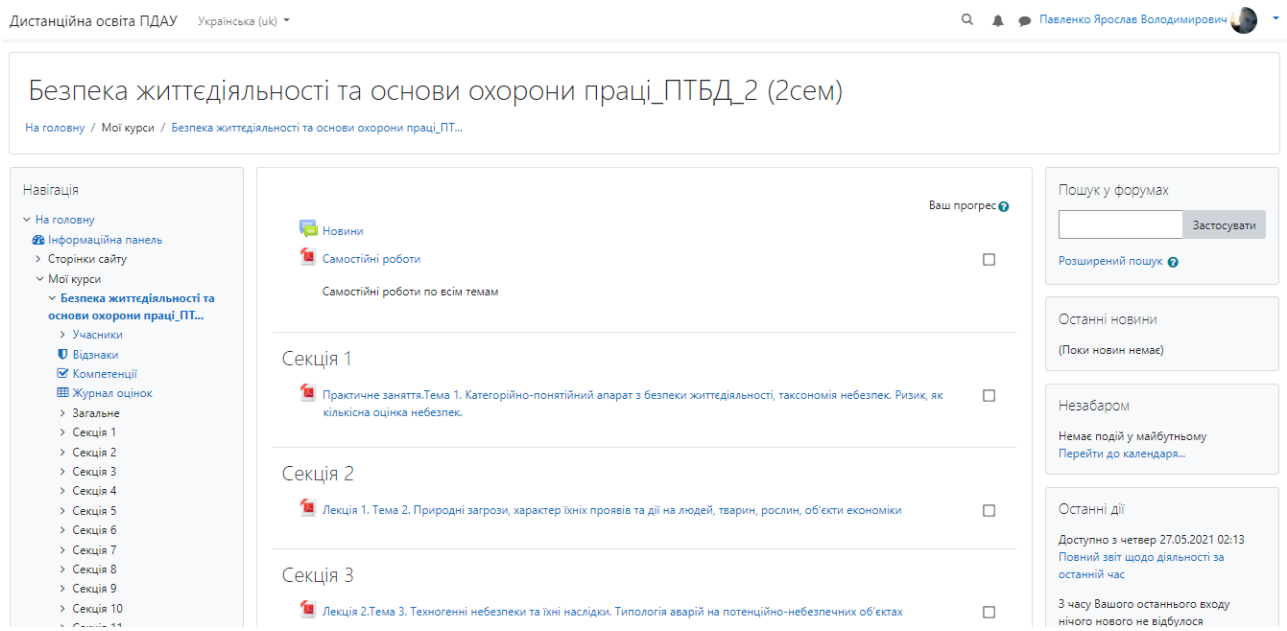

Рисунок 1.3 — Сторінка курсу moodle

Якщо підсумувати то варто виділити, що деякі сервіси реалізують функції простих інструментів для організації навчального процесу або інструментів для змішаного навчання, що дозволяє використовувати їх в малих навчальних закладах або разом з невеликими курсами яким не потрібні функції великої екосистеми для навчання. Інші інструменти ж реалізують повноцінні інформаційні системи для використання в великих навчальних закладах інтегровано з іншими інформаційними сервісами які належать закладу. Обидва підходи дозволяють організувати навчальний процес під потреби користувачів та створити онлайн середовище для навчання студентів більшості дисциплін які потребують підходу пов'язаного з використанням електронних систем для навчання.

Загалом оглянуті вище сервіси мають свої недоліки які майже повністю нівелюються сферою використання додатків, та переваги які дозволяють налагодити більш зручне використання для різних користувачів з своїми потребами. Розглянуті сервіси повністю відповідають своїм цілям та потребам користувачів які обирають роботу з ними.

1.3 Актуальність створення чат-бота

В даній роботі для зручності користувачів та демонстрації роботи API сервісу буде створено чат-бота на платформі Telegram за допомогою Bot API [4]. Таке рішення обумовлено прагненням створити для користувача варіант спрощеного та доступнішого для швидкого використання інтерфейсу взаємодії з платформою та показати приклад використання методів для розширення фунціоналу.

Таке поняття як «чат-бот» походить від двох слів англійського походження: chat – розмова яка є невимушеною і відбувається в цифровому просторі, bot (robot) - скорочено від робот, з цього можна зробити висновок, що це спеціалізовані роботи, призначення яких здійснення комунікації з користувачами в цифровому інформаційному середовищі, такі програми виконують дії або ведуть діалог згідно наміченого розробником сценарію. Такі програми для інтернет-комунікації як чат-боти використовують і базуються на сучасних технологіях в області інформатики та в деякому сенсі психології. В основі таких додатків часто використовують такі технології, як штучний інтелект та різні типи нейронних мереж, що дозволяє створювати чат-ботів які наближаються до людського спілкування і можуть більш органічно спілкуватися з користувачами вирішуючи таким способом комерційні та маркетингові завдання[5] .

В результаті змін культури інформації в суспільстві чат-боти знайшли значну популярність, перейшовши від виключно розваги до більш серйозних задач, такі додатки, в своїй більшості, тепер використовуються у сфері вирішення бізнес-завдань різної складності. В час коли світом правлять інформаційні технології – це можна вважати нормальним явищем. По-перше, чат-боти – це передові «платформи» для пошуку рішень бізнес-завдань. По-друге, чат-бот - це застосунок, який реалізує діалог з користувачем(клієнтом), формуючи відповіді на основі інформації з бази даних, наприклад: ви робите запит, де можна повечеряти і одразу отримуєте відповідь. Окрім цього, чат-боти

реалізують велику кількість функцій які дозволяють виконувати рутинні операції, задачі які пов’язані з об'єднанням даних, роботу з клієнтами на рівні діалогу.

Все частіше в умовах переходу більшості звичних процесів в цифровий простір необхідні нові методи для взаємодії людини з інформаційним середовищем, якщо раніше більшість повсякденних дій пов’язаних з використанням інформаційних сервісів виконувалась за допомогою спеціалізованих додатків або веб ресурсів, для таких задач як: замовлення товарів, пошук місць на карті, формування замовлень, поповнення рахунків, банківські операції, то зараз велику нішу таких задач легко можуть виконувати чат-боти. Такі застосунки зазвичай створюються на базі популярних месенджерів (Telegram, Viber, Discord), що дозволяє залучити велику аудиторію до використання функцій бота якого вигідно створювати для просування або спрощення доступу до вже готових сервісів. Наприклад бот «Приватбанк бот» для Telegram дозволяє здійснювати грошові перекази зручно переславши запит співрозмовнику прямо в чаті з ним, що значно зменшує кількість дій в порівнянні з сайтом банку або мобільним додатком. Таке використання чат ботів покликане спростити взаємодію користувача з сервісом, що позитивно впливає на оцінку сервісу користувачем.

Чат-бот який виступає віртуальним співрозмовником має базу знань, яка виглядає як список можливих запитів користувача та відповідей на них з урахуванням варіативності. Найбільш простими методами формування відповіді на запит може слугувати збіг по ключовим словам або контексту діалогу. Зазвичай існують різні рутинні справи які не займають багато часу але є нудними або просто відсутня мотивація їх виконувати. В таких випадках може допомогти технологія чат-боті. Відтепер ботів можна використовувати для збору інформації та її обробки, нормалізації. Звично, такі дії виконуються з допомогою ведення діалогу з людьми.

Можливості при створенні чат ботів постійно доповнюються новими функціями, оскільки цей напрямок активно розвивається і зараз за допомогою API які є в більшості месенджерів можна не тільки спілкуватися з користувачем або надавати йому інформацію, також можливо приймати платежі або працювати з картами. Такі нововведення від розробників месенджерів сприяють розвитку та покращенню умов для розвитку цієї сфери.

Сучасні чат-боти часто використовують не тільки в контексті месенджерів та для взаємодії з клієнтами, також більш розповсюдженими стають боти помічники, які зазвичай керуються голосом і використовують технології нейронних мереж для формування відповіді. Такі боти використовуються в сфері керування розумним домом або як віртуальні помічники в операційних системах (Microsoft Cortana, Google Assistant, Amazon Alexa, Apple Siri). Створення таких додатків дозволяє спростити використання операційної системи та отримувати інформацію в більш звичному для людини форматі, голосового повідомлення або дії.

Загалом створення чат боту для проекту платформи онлайн навчання дозволить винести найчастіше використовувані функції та нотифікацію в зручного для використання чат-бота, що покращить враження користувача від використання сервісу та дозволить використовувати інформацію про користувача від месенджеру для внутрішніх механізмів аутентифікації та наприклад додання зручних посилань на чат з користувачем в інтерфейс платформи.

Для задачі створення сервісу онлайн навчання така технологія також підходить адже студент та викладач повинен завжди отримувати актуальну інформацію за допомогою нотифікації яку дозволяють організувати месенджери та мати швидкий доступ до основних часто використовуваних дій пов’язаних з начальним процесом. Таким чином студент може без відвідування веб ресурсу дізнатися розклад лекцій і отримати посилання на них або отримати інформацію про свою успішність. В умовах віддаленого навчання це дозволяє організувати

більш зручну взаємодію користувачів з платформою, що дозволяє не перевантажувати сценарій взаємодії з сервісом та звільнювати більше часу під корисні дії [6].

1.4 Постановка задачі

Розробити сервіс, який матиме Frontend частину, серверний додаток для керування функціями додатку та Telegram бота який дозволяє виконувати основні дії та отримувати оголошення. В веб частині необхідно створити такі сторінки:

- авторизація – для авторизації в веб частині за допомогою Telegram;
- головна – для показу списку доступних курсів та переходу до них ;
- курс – для перегляду інформації про курс;
- студенти – для перегляду інформації про студентів приєднаних до курсу, створення оголошень (доступно тільки для викладача);
- матеріали – для створення та перегляду лекцій та завдань які містяться в курсі;
- оцінювання – для виставлення оцінок за здані завдання або перегляду результатів;

Створити API для взаємодії з сервісом через HTTP запити, продемонструвати його роботу через створення Telegram бота який буде взаємодіяти з сервісом та дозволить виконувати базові дії з сервісом без веб частини, реалізує отримання оголошень.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір програмної реалізації

Для створення клієнтської частини програмного продукту було використано HTML, CSS, JavaScript, та фреймворк Bootstrap, такий вибір технологій обумовлено простотою та великою кількістю довідкової інформації, а також досвідом використання та готовими наробками. Окрім цього в створенні сторінок використовувався влаштований в backend фреймворк шаблонізатор Razor Pages [7], використання цієї технології дозволяє сфокусуватися на роботі над сценарієм сторінки та спрощує взаємодію з серверним кодом шляхом влаштування в розмітку алгоритмічних елементів. Таким чином створюються файли розмітки з розширенням «.chtml» які містять в собі код з звичними для HTML тегами, але з додаванням циклів, розгалужень та інших притаманних програмуванню конструкцій.

```
<h2 class="greeting"><span class="time">Доброго дня</span>, @ViewBag.TgUser.TelegramFirstName</h2>
if (@ViewBag.Student.Courses.Count > 0)
{
    <p class="subtext">Ось ваші курси:</p>
}
else
{
    <p class="subtext">Ого, тут порожньо</p>
}
```

Рисунок 2.1 — Приклад розмітки з використанням технології Razor Pages

Фреймворк Bootstrap також дозволив спростити процес створення розмітки, це програмне рішення дозволяє створювати елементи дизайну з вже готовими стилями, створювати сторінки по вивіреним шаблонам розмітки просто вказуючи потрібну кількість колонок та налаштовуючи їхню поведінку. Цей фреймворк дозволяє швидко та просто створювати сторінки які легко привести до адаптивності, таким чином час затрачений на створення розмітки за шаблонами можна скоротити та зменшити ймовірність виникнення великої кількості помилок які часто виникають при верстанні сторінок. Також це дозволяє використовувати деякі вже готові конструкції, таким чином можна

організувати використання на сторінках модальних вікон, що також зменшує затрати часу через складність реалізації такого механізму на сторінці власноруч. Таким чином Bootstrap дозволить витрачати менше часу на створення типових для сторінки об’єктів таких як: модальні вікна, кнопки, таблиці, колонки, картки.

Для реалізації сценаріїв на створених сторінках використовується мова програмування JavaScript, цей інструмент дозволяє виконувати дії з вже побудованою сторінкою шляхом маніпуляцій з DOM деревом сторінки яке являє собою структуру яка представляє всі елементи на сторінці та їх зв’язки між собою. Такий підхід дозволяє створити сторінку яка буде динамічною та буде взаємодіяти з користувачем, змінюватися залежно від ситуації. Також цей механізм дозволяє асинхронно виконувати запити для поповнення сторінки інформацією або фіксації на сервері дій які виконує користувач. Таким чином використання мови JavaScript при створенні веб частини додатку дозволить зробити сторінки динамічним та реалізувати функціонал сторінки, нормалізувати данні підставленні шаблонізатором, відповідати на дії користувача та оброблювати зміни які він вносить.

Якщо підсумувати то така зв’язка технологій для створення клієнтської частини додатку дозволяє створювати сторінки які будуть мати естетичний вигляд та матимуть адаптивність, це в свою чергу дозволить використовувати сервіс з мобільних пристроїв, адже зараз це актуально через те що користувачі все частіше використовують для роботи з сервісами мобільні пристрої. Окрім цього використання саме цих технологій дозволяє створювати розмітку з вже влаштованими в неї даними з бази даних за допомогою шаблонізатора, що зменшує кількість запитів при завантаженні сторінки і зменшує час на цей процес. Окрім цього буде використана незначна кількість бібліотек для JavaScript, це дозволить працювати з календарями та створювати деякі складні елементи сторінок.

Інша частина додатку представлятиме собою серверний код створений за допомогою одної з найбільш використовуваних та затребуваних на ринку мов

програмування яка має назву C# та створена в 1998-2001 роках невеликою групою інженерів Microsoft [8]. Ця мова створена як інструмент для розробки додатків під Microsoft .Net Framework. Синтаксис перейняв в себе більшість кращих практик з таких мов як Java, C, C++. Це дозволило вирішити більшість синтаксичних проблем цих мов, також при створенні розробники відмовились від деяких проблемних концепцій, тому в C# немає множинного наслідування класів, але допускається множинна реалізація інтерфейсів. Таким чином ця мова дозволяє створювати високоефективний код, при цьому рівень абстракції знаходиться на достатньому рівні. Отже, рівень входження в технологію достатньо низький, також мова має обширну базу документації та велику кількість статей, що дозволяє швидко знаходити рішення виникаючих проблем.

Для створення веб додатків C# може запропонувати платформу для розробки веб-додатків ASP .NET [9], ця платформа включає в себе декілька програмних моделей таких як:

- ASP.NET Web Forms — фреймворк який дозволяє створювати модульні сторінки з компонентів;
- ASP.NET MVC — фреймворк для створення веб сторінок з використанням паттерну MVC;
- ASP .NET Web Pages — являє собою спрощений синтаксис призначений для створення динамічного коду та доступу до даних в всередині розмітки HTML;
- ASP.NET Web API — фреймворк для створення Web API поверх .Net.

Даний програмний засіб спрощує більшість процесів при розробці серверного коду, наприклад створення системи авторизації та моделей для користувачів можна зробити автоматично за допомогою кросплатформеного рішення Entity Framework яке дозволяє створювати типові сутності та використовувати принцип «Code First». Цей підхід дозволяє організувати спрощену в порівнянні з традиційною роботу з проектування бази даних, таблиці створюються по описаних класах які називають моделями. Зв’язки між таблицями також

описуються за допомогою зав’язків між класами. Це дозволяє спочатку писати код, а не думати над структурою бази даних. Такий принцип дозволяє створювати базу даних на основі програмного коду автоматично, це також спрощує доступ до даних оскільки не потрібно робити явні запити, звертання до даних відбувається за допомогою об’єктів в програмному коді та обробників (контролерів). В результаті дані з бази даних представляють собою колекцію об’єктів з якою зручно працювати за допомогою внутрішніх інструментів C#, наприклад з використанням технології LINQ [10]. Використання цієї технології дозволяє виконувати запити до даних в зручній формі мовних конструкцій, таким чином можна виконувати запити, фільтрацію упорядкування, групування даних з джерела, обходячись мінімальним об’ємом програмного коду.

Також буде розроблено API для реалізації керування сервісом без використання веб частини. Всі отримання даних та дії з ними можна виконувати не тільки за допомогою веб інтерфейсу але й за допомогою HTTP запитів. Щоб продемонструвати це буде створено програмний продукт який буде використовувати API, також реалізує механізми авторизації, реєстрації та нотифікації користувачів. Цей засіб буде використовувати платформу Telegram та являтиме собою бота розробленого за допомогою мови Python. Таким чином бот буде представляти користувачу основні дії з сервісом, реалізуватиме основні функції та зможе використовуватись як спосіб для швидкого отримання інформації. Такі програми простіше та швидше всього розробляти за допомогою мови Python через її простоту та велику кількість засобів для роботи з предметною областю. Використання Python обумовлене зручними механізмами для роботи з API та загальною швидкістю роботи. Окрім цього ця мова програмування дозволяє використовувати готові бібліотеки для створення подібних рішень, що значно зменшує час на розробку. Розвертати готовий код можна на будь якому типі серверів адже мова Python є кроссплатформенною.

2.2 Паттерн

В наш час розробка веб додатків повільно відходить від методики написання окремої Frontend частини яка тримає зв'язок з серверною частиною лише за допомогою запитів. Такий підхід збільшує час розробки додатку і для ефективної та швидкої розробки доводиться додавати в команду більше людей, що збільшує ціну розробки таким методом.

Веб технології стрімко розвиваються та потребують нових рішень через постійно зростаючу складність задач та додатків які їх вирішують. Існує велика кількість фреймворків, які допомагають розробникам спрощувати роботу та витрачати менше часу на вирішення типових задач, наприклад Angular, React, Django, Ruby on Rails. Додаток, зазвичай, створюється цілою командою спеціалістів у своїх сферах, кожен з яких працює зі своєю опанованою ним технологією, такими як HTML та CSS, JavaScript для клієнтської частини додатку. ASP, Python, Java, Node.js для написання серверної частини або інтерфейсів для взаємодії з сервісом ззовні та MySQL (або OracleDatabase, Microsoft SQL Server тощо) для збереження інформації користувачів або метрик та управління нею.

Кожному з цих членів команди необхідно вносити покращення і зміни в код таким чином щоб їхні фрагменти коду вписувались у загальний дизайн програми що потребує додаткової документації для узгодження командної роботи. Наприклад, розробнику на клієнтській стороні (Frontend) необхідно змінити код HTML, CSS і JavaScript таким чином, щоб він не порушив процеси та алгоритм в коді розробника який відповідає за серверну частину додатку (Backend) і не викликав конфліктів при використанні. Відповідно, коли розробник бази даних вносить зміни в діаграму таблиць або процеси всередині БД, на серверній стороні може бути необхідно внести багато змін в код який працює з цією структурою, аби програма працювала коректно і ще раз провести тестування, щоб впевнитись, що всі модулі додатку працюють з внесеними змінами.

Тож важливо зазначити, що існує гостра потреба в тому щоб відокремити презентацію даних від логіки та збереження цих даних в базі даних додатку. Це дозволяє розмежувати роботу спеціалістів та створювати продукт який легко підтримувати та розширювати.

Існують парадигми та схеми дизайну додатків (паттерни), які дозволяють вирішити цю проблему і будувати програмний продукт по деяких схемах і принципах, але в цій роботі увага приділяється шаблону проектування MVC.

Ідея розділення додатку за шаблоном MVC вперше була запропонована Тригве Реенскаугом у 1970-х. Його думка була такою, «головна мета MVC - це усунути розрив між ментальною моделлю користувача та цифровою моделлю, що існує в комп'ютері»[11].

Він виділяє, що є велика користь в веб дизайні та мережевому маркетингу, якщо створювати додатки які враховують принцип модульності та розділяються на окремі частини які працюють разом але не залежать один від одного. "Ізоляція функціональних блоків один від одного максимально дозволяє дизайнеру додатків зрозуміти та змінити кожен конкретний блок, не знаючи про все, що стосується інших."

Структура архітектури MVC розділяє створюваний додаток на три основні групи компонентів: моделі, презентації і контролери. Такий підхід дозволяє реалізовувати раціональне розділення задач. Згідно з цією структурою запити користувачів потрапляють в контролер який відповідає за роботу з моделлю для виконання дій користувачів або отримання результатів їх запитів. Також контролер вибирає вид щоб відобразити його з всіма необхідними даними моделі яку запитав користувач.

Паттерн MVC розбиває відповідальність на три базові ролі, що забезпечує ефективну співпрацю в порівнянні з традиційною розробкою без використання цих принципів. Ці ключові ролі - це розробка, дизайн та інтеграція, вони уособлюють собою всю концепцію паттерну.

Задачу розробки беруть на себе професійні розробники, які беруть на себе відповідальність за написання логіки серверної частини програми та структури того як інформація зберігається та використовується в інформаційній системі. Ці члени команди описують роботу запитів, перевірку даних, безпеку та доступи користувачів, обробку даних та їх збереження тощо.

Задачу розробки дизайну отримують розробники та дизайнери, які відповідають за те якою програму побачить її користувач. Вони займаються створенням логіки того як відображати дані які отримують від розробників відповідальних за задачу розробки серверної логіки.

Задача інтеграції полягає в тому щоб об’єднати два вище описаних модулі і змусити їх правильно працювати обмінюючись інформацією і відображаючи її користувачу додатку, це завдання зводить два модулі в одну повноцінну інформаційну систему.

Модель дизайну додатків MVC так добре підходить для розробки веб-орієнтованих сервісів, оскільки такі додатки поєднують декілька технологій, які зазвичай, розділені на набір частин працюючих окремо. Також, типовою поведінкою MVC є можливість надсилання різних представлень даних для різних типів користувачів.

Взаємодія користувача з розробленим за паттерном MVC додатком відбувається за цілком закономірним і природним циклом: користувач виконує «дію», а у відповідь програма вносить зміни в свою модель даних та надає її в оновленому вигляді користувачеві. Користувач знову виконує «дію» і цикл повторюється. Тобто взаємодія користувача з системою будується на природних принципі «Виконав дію - отримав результат», Це неймовірно зручно для веб-орієнтованих додатків, що виконують свою роботу за допомогою серії HTTP-запитів та відповідей, адже за таким принципом працює більшість популярних сервісів та ресурсів.

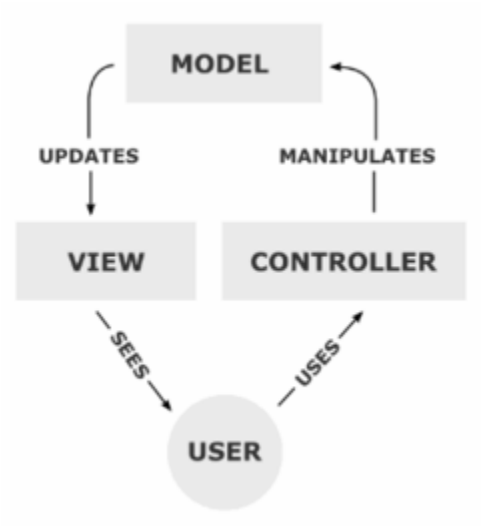

Рисунок 2.2 — Схема роботи MVC

Model - це структурна частина системи, яка керує всіма задачами які пов’язані з управлінням даними якими оперує додаток: перевірки, контроль сесій користувачів, зв’язки між таблицями, полями та сутностями.

На рівні моделі програмний код відповідає за бізнес логіку програми. Код написаний для моделі інкапсулює методи доступу до даних які знаходяться в БД або файлах та робить бібліотеку класів доступною для використання на інших рівнях, що значно спрощує роботу з даними в контролерах та представленнях. Звично Model створюється враховуючи валідацію та цілісність даних, що дозволяє не турбуватися про невідповідність даних в базі даних або появу некоректних записів.

Окрім цього, Model описує класи, які визначають ту область яка представляє дані. Ці об'єкти, за часту не тільки інкапсулюють дані, що знаходяться в базах даних, але також реалізують код, який виконує функції для маніпулювання даними та виконання бізнес-логіки.

Як висновок можна виділити, що Model виконує функцію обробки та абстракції даних та перевірку доступу до них. Model реалізує методи та способи взаємодії з джерелами даних в додатку.

Код View відповідає за графічне представлення та керування інтерфейсом користувача. Це означає що ця частина додатку відповідає за всі графічні елементи такі як: кнопки, форми, модальні вікна, таблиці, картки сутностей, які

користувач бачить під час використання програми. View також використовують для того щоб виводити помилки та системну інформацію для розробників в процесі пошуку багів. Можливість відокремити дизайн програми від логіки її роботи, значно зменшує ризик виникнення помилок, коли в процесі роботи з’явиться потреба змінити частину елементу інтерфейсу наприклад таблицю чи кнопку. Окрім того робота розробників серверної частини значно спрощується та скорочується, оскільки їм тепер не потрібно бачити звірятися і перевіряти елементи коду клієнтської частини, елементи структури сторінок, графічні елементи.

View- це та частина додатку яку зазвичай можна назвати веб-дизайном або шаблоном сторінки. Цей рівень контролює те як будуть відображенні дані та те як користувач буде з ними взаємодіяти. Він також реалізує способи збору даних які надають користувачі наприклад через форми. Зазвичай в View, використовуються технології для створення клієнтської частини такі як HTML, CSS та JavaScript.

Як правило, View не містять елементів які належать до логіки інформаційної системи. Це робиться для того щоб полегшити дизайнеру роботу, тому логічні блоки роблять максимально простими і маленькими. Такі блоки використовуються для не складних алгоритмічних операцій, наприклад для створення деякої кількості елементів за допомогою циклу.

Більша частина фреймворків веб-ресурсів використовує деякі механізми для створення шаблонів, які дозволяють генерувати контент на сторінці за допомогою алгоритмічних блоків, щоб зменшити кількість HTML-коду і зменшити ризик помилок при створенні контенту вручну, що збільшує ризик помилок при заповненні та друкарських помилок. Такі генератори використовуються для створення веб-сторінок які складно заповнювати вручну, таких як таблиці, форми, списки тощо. Такі механізми використовують спеціальні частини HTML-коду. Розробник додає алгоритмічні блоки в прямо в код розмітки, потім ця розмітка потрапляє в генератор де алгоритмічні блоки

виконуються та розмітка доповнюється новими елементами. Наприклад для користувача з роллю «Студент» потрібно показати таблицю з результатами його навчання, і в той же момент користувачу «Викладач» на цьому ж місці потрібно показати список завдань на перевірку. Така задача легко вирішується за допомогою умовного блоку в шаблоні. Такий підхід робить весь процес прозорим для розробника, бо він може бачити всю розмітку сторінки, перш ніж використовувати її.

Контролер (Controller) відповідає за обробку подій. Події може викликати як користувач так і системні процеси. Контролер отримує запити та формує дані для відповіді. Він так само несе визначає формат відповіді. Контролер взаємодіє з моделлю, щоб отримати дані та створити на їх основі View. Коли запит потрапляє на сервер, паттерн MVC віддає його методу в контролері відповідно до URL-адреси. Controller керує зв'язком між View і Model. Він формує відповіді на запити користувачів, передає данні з них в Model та вирішує, який View слід генерувати та відобразити користувачу.

2.3 Вибір бібліотеки для взаємодії з Telegram

Як засіб для роботи з чат ботом на платформі Telegram потрібна бібліотека яка дозволить виконувати запити до Telegram Bot API простими синтаксичними конструкціями, це дозволить сконцентруватися на розробці логіки роботи бота, а не на методах для взаємодії з месенджером. Таким чином для рішення цього завдання була вибрана бібліотека для мови Python яка має назву pyTelegramBotAPI [12]. Це рішення базується на відкритому програмному коді та дозволяє організувати зручну роботу з Telegram Bot API використовуючи Python. Цей засіб надає велику кількість методів для зручної роботи з платформою, а також спрощує доступ до даних пов’язаних з взаємодією. Окрім цього надається колекція типів для створення типових для Telegram об’єктів для взаємодії з користувачем таких як: клавіатури, кнопки, опитування та інші типи повідомлень. Використання цієї бібліотеки дозволяє скоротити час на розробку бота та зробити код більш чистим та простим.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Для розуміння принципу роботи платформи та її складових побудована схема її роботи у вигляді абстрактної моделі (рис. 3.1). ЇЇ компонентами є складові частини додатку такі як:

- Серверна частина — частина додатку яка відповідає за керування даними та API сервісу, формує сторінки та керує доступом до ресурсів;
- Клієнтська частина — частина яку бачить користувач, сторінки, форми;
- Telegram бот — являє собою додаток який реалізує частину функцій за допомогою API платформи;
- Telegram Bot API — інтерфейс для доступу до бота на платформі Telegram.

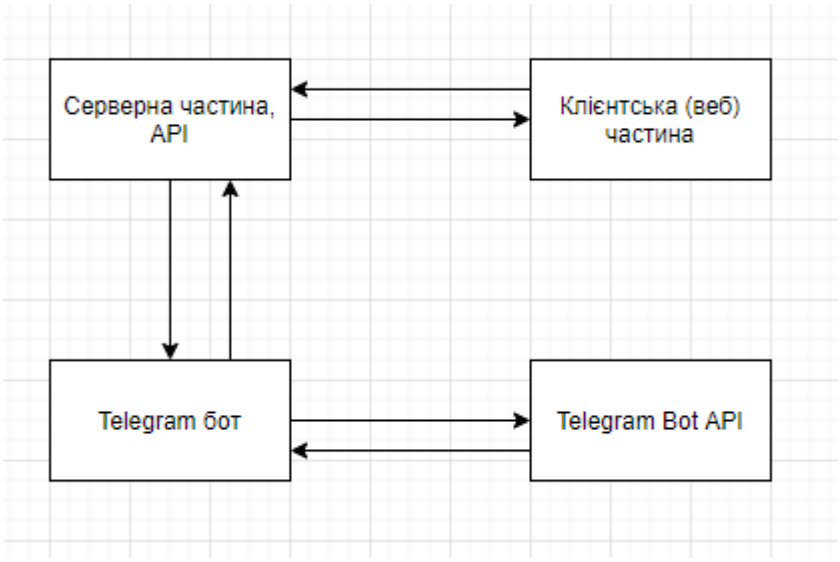

Рисунок 3.1 — Структура додатку

Звідси видно, що всі модулі взаємодіють між собою, клієнтська частина взаємодіє з серверною та API для отримання даних та виконання дій пов’язаних з ними, в той же час чат-бот взаємодіє з сервісом для виконання дій та отримання даних, а також з Telegram. Таким чином він може взаємодіяти з користувачем через відправку повідомлень та calback.

Також представлена Use-case діаграма [13] (рис. 3.2) щоб показати ймовірні сценарії використання, та моделювання процесів додатку. Це дозволяє правильно розуміти як повинна працювати система та які ролі користувачів відповідають за дії які вони мать виконувати. Основним призначенням цієї діаграми є показати функціональність і поведінку, це дозволяє замовнику, кінцевому користувачу та розробнику спільно обговорювати систему яка проектується.

Рисунок 3.2 — Use-case діаграма

Окрім цього в проекті використовується база даних, отже потрібно скласти схему таблиць, що допоможе розуміти принцип зберігання та структуру даних в додатку. В проекті використовується одразу дві бази даних, одна з яких повністю відповідає за збереження даних про користувачів та авторизації, а друга зберігає

та структурує дані додатку які пов’язані з даними курсів, оголошеннями та ролями користувачів. Таке розділення дозволяє більш чітко структурувати дані та оброблювати їх. За допомогою концепції «Code first» яку використовує ASP .Net ми не створюємо цю структуру явно, структура бази даних формується на основі моделей додатку, такі моделі є класами які представляють сутності, кожна сутність це таблиця, а кожне поле це колонка таблиці. Таким чином зміни в структуру бази даних вносяться коли змінюються моделі додатку. Середовище розробки веб-додатків на платформі ASP .Net не передбачає створення діаграми бази даних, через те, що концепція цього не потребує але можливо створити діаграми цього типу підключившись до створеної бази за допомогою Microsoft SQL Server Management Studio. За допомогою цього інструменту можна впевнитись в правильності створеної автоматично структури бази даних. Це дозволяє контролювати створення інформаційної моделі, виду таблиць та зв’язки між ними.

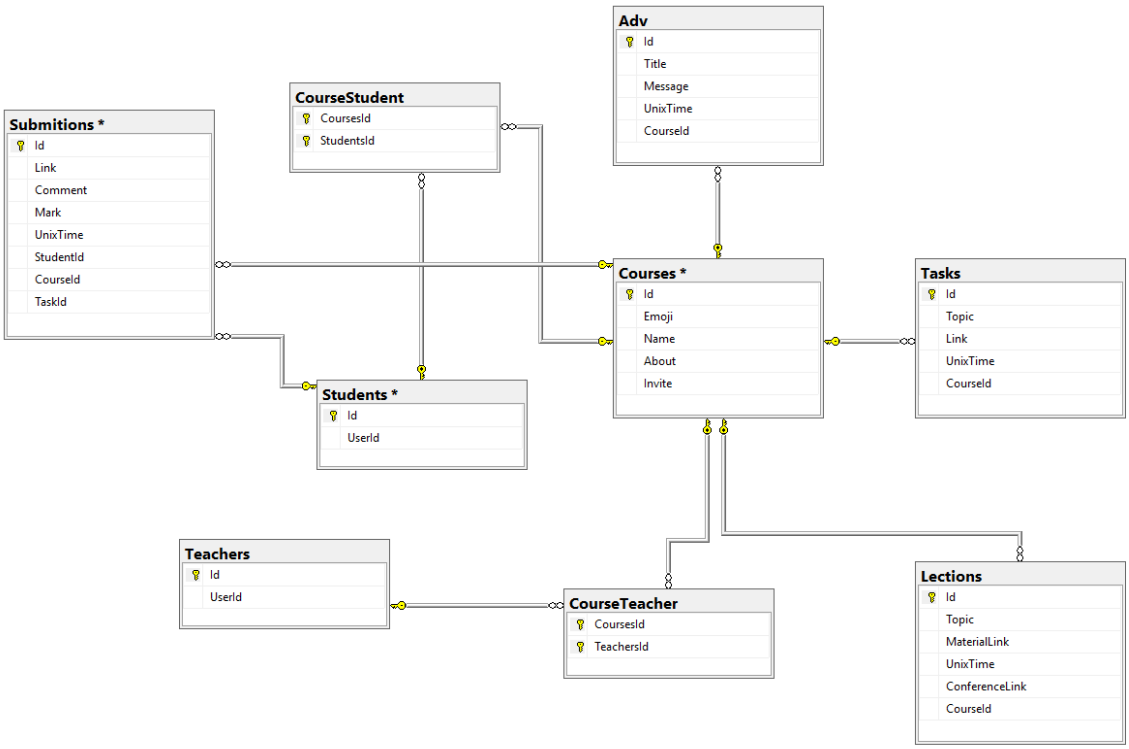

Рисунок 3.3 — Діаграма бази даних додатку

З цієї діаграми ми бачимо зв’язки між сутностями в основній базі даних яка відповідає за дані додатку. Ця база даних використовується для збереження інформації про курси, оголошення, завдання, лекції тощо. Зв’язки між такими таблицями як: Courses і Students, Courses і Teachers влаштовані за принципом «Many to many». Дані про час зберігаються в базі даних за допомогою Unix мітки часу, що спрощує використання цієї інформації через стандартизацію.

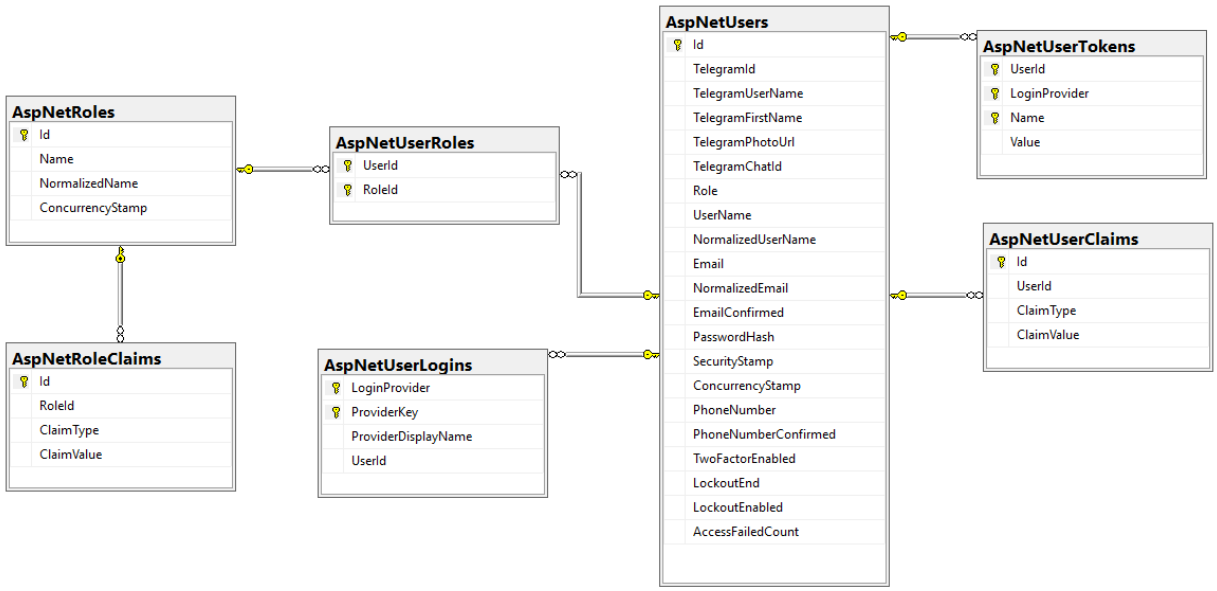

Рисунок 3.4 — Діаграма бази даних користувачів

Тут ми можемо бачити як працює збереження даних про користувачів та їхню авторизацію, також за допомогою ролей працює система доступу до інформації, наприклад студент не зможе отримати доступ до створення курсу бо відбувається перевірка його ролі.

Отже можна підсумувати, що інформаційна модель додатку містить дві бази даних кожна з яких відповідає за свої специфічні задачі але вони з’єднуються за допомогою спільних полів, це дещо ускладнює отримання даних, але дозволяє розмежовувати роботу з даними користувача та даними додатку. Бази даних створюються автоматично з класів моделей, та зазвичай не потребують змін або доробок. Такий функціонал забезпечується за допомогою

Entity Framework [14], ця технологія дозволяє зменшити затрати часу на розробку інформаційної моделі додатку, та спрощує її опис.

3.2 Програмна реалізація

Першим кроком створимо проект ASP .Net з паттерном MVC, середовище розробки Microsoft Visual Studio створить всю необхідну структуру папок та файлів, а також ініціалізує всі файли конфігурацій та класи необхідні для запуску. Після створення проекту необхідно підключити систему контролю версій Git. Введення в проект системи контролю версій дозволить контролювати розробку проекту та зберігати код, на випадок його втрати або пошкодження.

Після виконання цих кроків ми отримаємо порожній проект і можна починати розробку додатку, почнемо з введення системи авторизації. Для реалізації цього механізму будемо використовувати Telegram Login Widget [15], та розширимо функціональність влаштованої в початковий проект системи Identity.

Клас який наслідує стандартний клас користувача та доповнює його необхідними полями для реалізації авторизації за допомогою Telegram:

```
public class User : IdentityUser
{
    public int TelegramId { get; set; }
    public string TelegramUserName { get; set; }
    public string TelegramFirstName { get; set; }
    public string TelegramPhotoUrl { get; set; }
    public string TelegramChatId { get; set; }
    public string Role { get; set; }
}
```

Таким чином модель стандартного користувача доповниться полями які призначенні для зберігання даних від Telegram.

Далі необхідно створити запит для реєстрації користувача за його даними, так як за задумкою реєстрацію користувача буде здійснювати чат-бот, то потрібно створити для цієї дії, отже код для додавання нового користувача в систему виглядає так:

```
var result = await _userManager.CreateAsync(user);

if (!result.Succeeded)
{
```

```
            return BadRequest();
        }

        if (status == "Teacher")
        {
            Teacher teacher = new Teacher { UserId = user.Id };
            db.Teachers.Add(teacher);
        }

        if (status == "Student")
        {
            Student student = new Student { UserId = user.Id };
            db.Students.Add(student);
        }
        db.SaveChanges();
        await _userManager.AddToRoleAsync(user, status);
```

Далі необхідно виконати написання коду для перенаправлення користувача на сторінку авторизації та створити саму сторінку авторизації з запитами. Код який відповідає за показ сторінки авторизації:

```
[HttpGet]
public IActionResult Login()
{
    return View();
}
```

Тут виконується маршрутизація користувача і йому генерується і повертається відповідний код сторінки. Код який відповідає за авторизацію користувача та перенаправлення на головну сторінку має такий вигляд:

```
            if (loginWidget.CheckAuthorization(fields) == Authorization.Valid)
            {
                var user = await _userManager.FindByEmailAsync(id.ToString());

                if(user != null)
                {
                    //update user
                    user.TelegramUserName = username;
                    user.TelegramPhotoUrl = photo_url;
                    user.TelegramFirstName = first_name;
                    await _userManager.UpdateAsync(user);

                    //sign the user and go to home
                    await _signInManager.SignInAsync(user, isPersistent: true);
                    return Ok(user.TelegramUserName + " sign in");
                }
                else
                {
                    return BadRequest("Not found");
                }
            }
```

Окрім цього в системі авторизації не вистачає механізму для того щоб користувач міг вийти з свого облікового запису, код для реалізації цього функціоналу виглядає так:

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync();
            return RedirectToAction("Login", "Account");
        }
```

Отже таким чином було створено механізми для авторизації та виходу з облікового запису, кінцевий результат має такий вигляд (рис. 3.5).

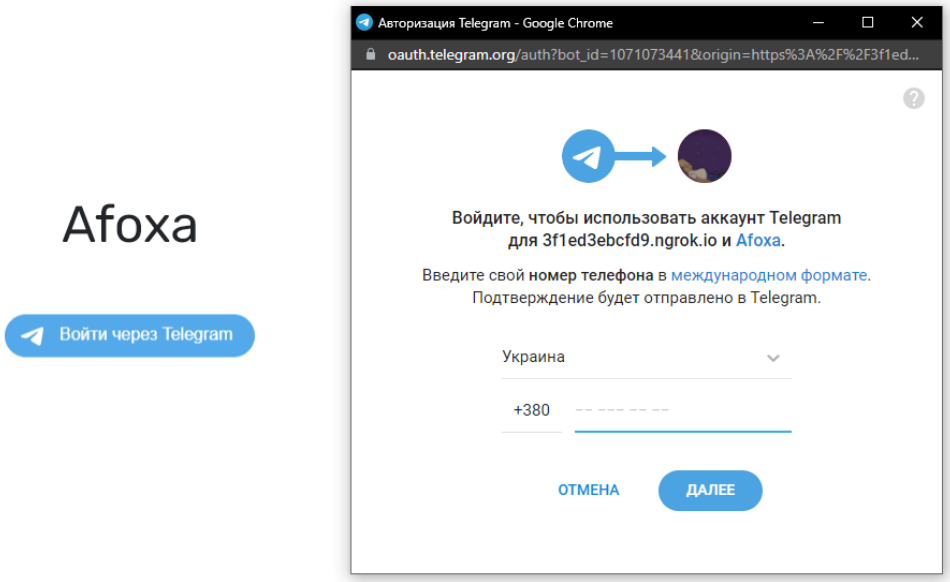

Рисунок 3.5 — Сторінка авторизації

Механізм авторизації готовий, отже можна переходити до створення початкової сторінки яка буде показувати користувачам список їх курсів. Почнемо з написання моделі курсу. Вона виглядає так:

```
    public class Course
    {
        public int Id { get; set; }

        [Required]
        public string Emoji { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string About { get; set; }
        public string Invite { get; set; }
        public virtual List<Student> Students { get; set; } = new List<Student>();

        public virtual ICollection<Teacher> Teachers { get; set; } = new
List<Teacher>();
    }
```

Модель курсу містить список студентів та список викладачів які прив’язані до цього курсу.

Далі необхідно написати код сторінки яка буде показувати користувачу список його курсів, код який відповідає за створення картки курсу на сторінці виглядає так:

```
<div class="course-list">
    @foreach (var course in ViewBag.Student.Courses)
    {
        <button class="course-card redirect-button" id="@course.Id">
            <div class="course-icon">
                <span>@course.Emoji</span>
            </div>
            <p class="course-name">@course.Name</p>
        </button>
        <br />
    }
</div>
```

Також потрібно для викладача додати функціонал створення нового курсу, для цього створимо нове модальне вікно та запит для створення нового курсу який буде доступний тільки для ролі «Викладач», фрагмент коду який за це відповідає виглядає так:

```
            if (course.Id == 0)
            {
                // add teacher to course
                course.Teachers.Add(teacher);
                db.Courses.Add(course);
                db.SaveChanges();
                course.Invite = generateInvite(course.Id);
                db.SaveChanges();
                return Ok(course.Id);
            }
```

Окрім цього для створення курсу викладачеві потрібно вибрати іконку яка буде прикріплена до курсу, для цього необхідно підключити на сторінку елемент вибору іконки за допомогою JavaScript:

```
if (document.getElementById('content').dataset.role == 'Teacher') {
    document.querySelector('emoji-picker')
        .addEventListener('emoji-click', event => replaceEmoji(event.detail));
}
```

Код який відповідає за відправку запиту на створення нового курсу має такий вигляд:

```
$('.create-btn').click(function () {
    let emojiIcon = $('.emoji').text();
    let name = $('#name').val();
    let about = $('#about').val();
```

```
    let course = {
        Name: name,
        About: about,
        Emoji: emojiIcon
    };

    $.post('/Course/CreateOrUpdate', course)
        .done(function (id) {
            addCourse(course, id);
            $('#Modal').modal('toggle');
            $('#name').val('');
            $('#about').val('');
            $('.emoji').text('');
        })
        .fail(function () {
            alert('error');
        });
});
```

Таким чином створено початкову сторінку яка відповідає за відкриття та показ курсів які пов’язані з користувачем. Вона має такий вигляд (рис. 3.6 та рис. 3.7).

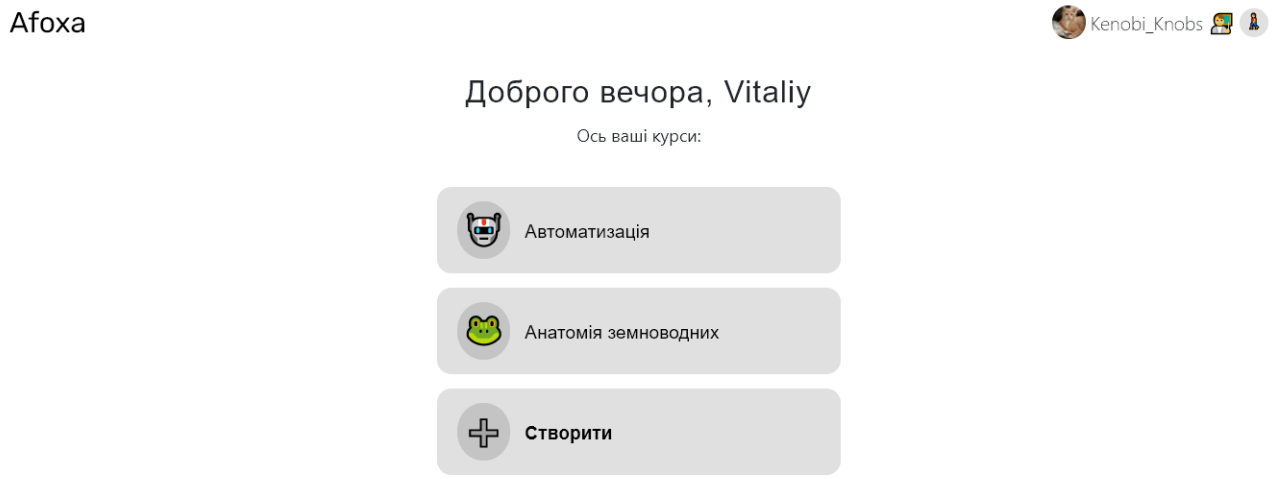

Рисунок 3.6 — Сторінка вибору курсу

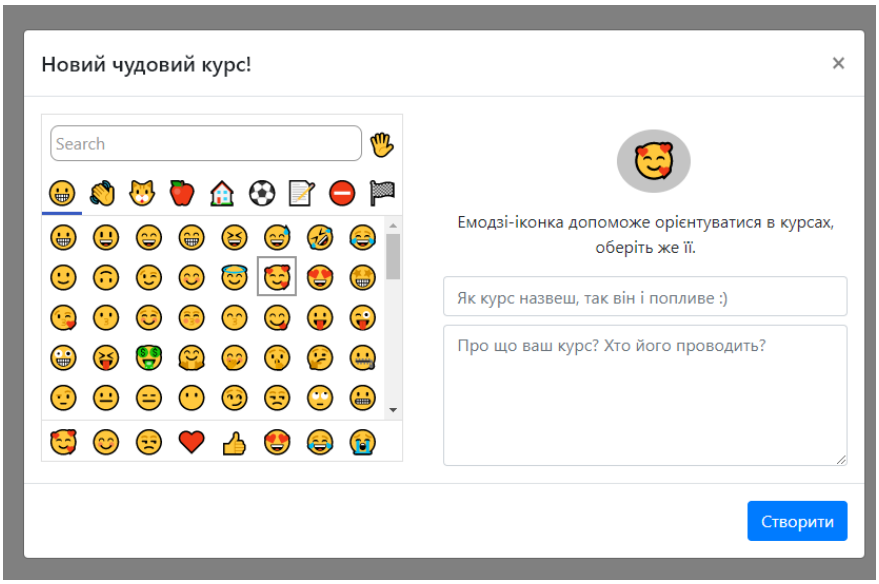

Рисунок 3.7 — Вікно створення нового курсу

Далі створимо сторінку з інформацією про курс, на ній користувач зможе ознайомитися з інформацією про обраний курс, а викладач має можливість редагувати цю інформацію. Для цього створимо метод для маршрутизації та нове представлення. Створимо розмітку сторінки а також напишемо скрипт який дозволить викладачу редагувати курс. Будь-який користувач у якого є доступ до курсу за допомогою цієї сторінки може отримати список викладачів які закріплені за курсом. Код для реалізації цього представлений так:

```
        <div class="col">
            <h3 class="header">Викладачі:</h3>
            @foreach (var user in ViewBag.Teachers)
            {
                <div class="teacher">
                    <img class="avatar avatar-list" src="@user.TelegramPhotoUrl" />
                    <a class="username teacher-name"
href="https://t.me/@user.TelegramUserName">@user.TelegramUserName</a>
                </div>
            }
          </div>
```

Таким чином фінальний результат розробки цієї сторінки має вигляд (рис. 3.8)

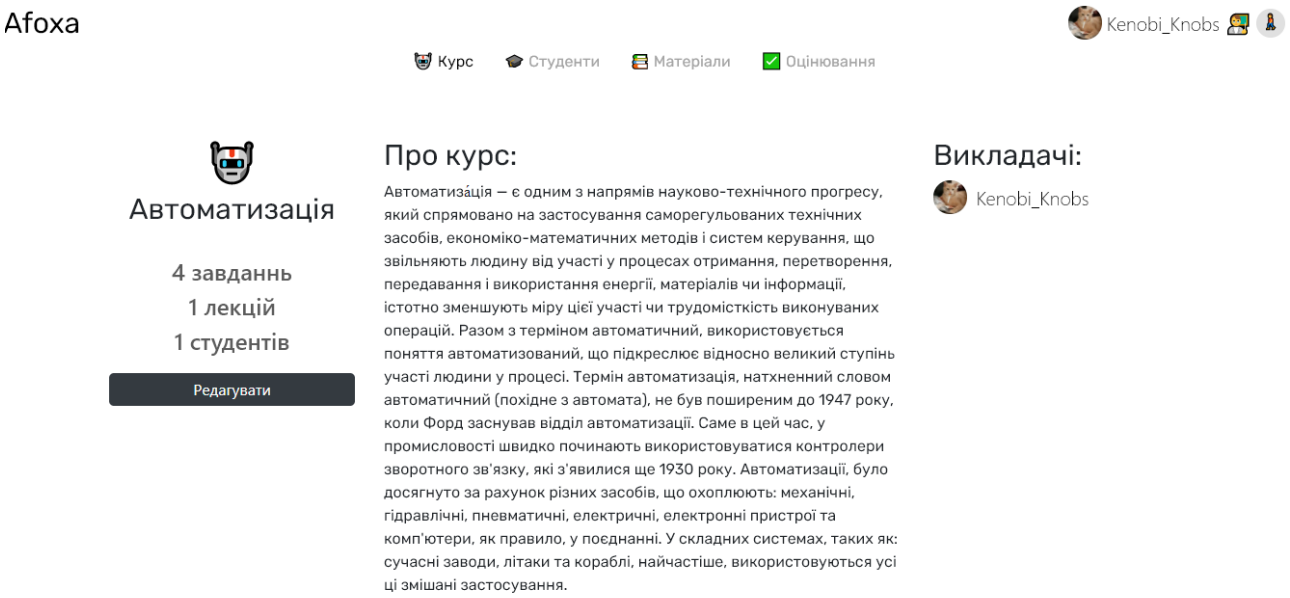

Рисунок 3.8 — Сторінка інформації про курс

Перейдемо до розробки сторінки яка відповідає за роботу з студентами. Цей функціонал доступний тільки користувачу з роллю «Викладач» який є власником курсу, інші користувачі не зможуть потрапити на цю сторінку. На цій сторінці викладач зможе переглядати список студентів, має можливість виключити студента з курсу, створює оголошення, може отримати або оновити посилання для запрошення студентів до курсу. Код який відповідає за додавання таблиці з студентами на сторінку виглядає так:

```
                    @{int i = 1; }
                     @foreach (var user in ViewBag.Students)
                     {
                     <tr class="@user.Id">
                         <td>@i</td>
                         @{i++; }
                         <td>
                             <div class="table-user">
                                 <img class="table-avatar" src="@user.TelegramPhotoUrl"
/>
                                 <a class="table-username"
href="https://t.me/@user.TelegramUserName">@user.TelegramUserName</a>
                             </div>
                         </td>
                         <td>
                             <div class="dropdown dropleft">
                                 <i class="fas fa-ellipsis-h" data-toggle="dropdown"
style="font-family: 'FontAwesome'; color:black;"></i>
                                 <div class="dropdown-menu">
                                     <a class="dropdown-item kick-modal" data-
toggle="modal" data-target="#kickModal" name="@user.TelegramUserName" id="@user.Id">❌
Вигнати</a>
                                 </div>
```

```
                    </div>
                </td>
            </tr>
            }
```

Для створення оголошень потрібно створити додаткову модель яка міститиме дані про оголошення. Код моделі має такий вигляд:

```
public class Ad
    {
        public int Id { get; set; }

        [Required]
        public string Title { get; set; }

        [Required]
        public string Message { get; set; }

        public long UnixTime { get; set; }

        public int CourseId { get; set; }

        public Course Course { get; set; }
    }
```

Ця модель зберігає нові запити на оголошення від викладача, та містить інформацію про час відправки що дозволяє виконувати відкладену відправку оголошень. Далі створимо методи для створення та оновлення коду доступу до курсу. Код доступу до курсу можна поширювати серед студентів завдяки ньому вони можуть приєднатися до курсу, фрагмент коду для приєднання студента до курсу виглядає так:

```
var course = db.Courses.FirstOrDefault(c => c.Invite == token);
var user = idb.Users.FirstOrDefault(u => u.TelegramId == userId);
var student = db.Students.FirstOrDefault(s => s.UserId == user.Id);

if (course == null || student == null)
{
    return NotFound();
}

course.Students.Add(student);
try
{
    db.SaveChanges();
    return Ok(200);
}
catch
{
    return BadRequest();
}
```

В результаті ми отримаємо сторінку яка виглядає таким чином (рис. 3.9)

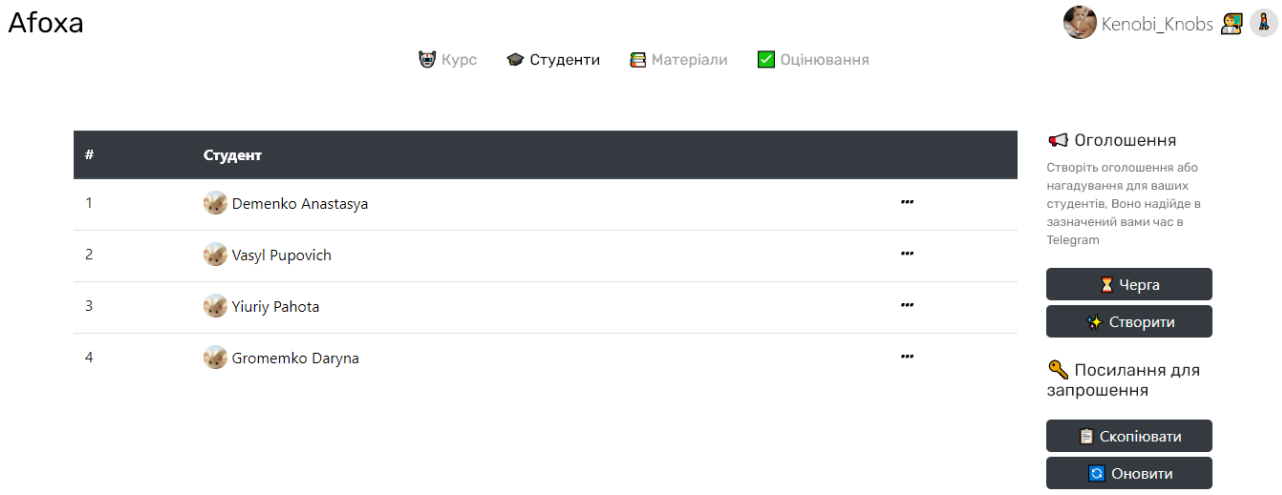

Рисунок 3.9 — Сторінка управління студентами

Перейдемо до створення сторінки матеріалів, вона дозволяє викладачу створювати лекції та завдання а студент отримує можливість ознайомитися з ними та відправити готові завдання. Код який відповідає за відправку завдання студентом виглядає так:

```
        if (student.Courses.Contains(loadCourse) && student.Id ==
submition.StudentId && taskContain)
            {
                if (submition.Id == 0)
                {
                    submition.Mark = -1;
                    db.Submitions.Add(submition);
                    db.SaveChanges();
                    return Ok(submition.Id);
                }
                else
                {
                    return BadRequest();
                }
            }
```

Розглянемо приклад створення лекції, ця типова для викладача дія реалізується таким кодом:

```
            if (teacher.Courses.Contains(loadCourse))
               {

                   if(lection.Id == 0) {
                       db.Lections.Add(lection);
                       db.SaveChanges();
                       return Ok(lection.Id);
                   }
                   else
                   {
                       db.Lections.Update(lection);
```

```
            db.SaveChanges();
            return Ok("Update lection id=" + lection.Id);
        }
    }
```

В результаті створена сторінка реалізує всі закладені в неї функції і має наступний вигляд (рис. 3.10).

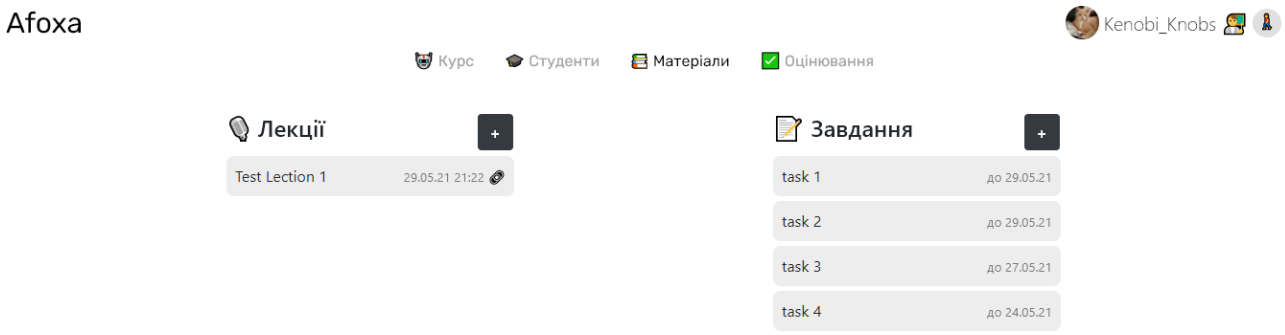

Рисунок 3.10 — Сторінка матеріалів

Сторінка оцінювання дозволяє оцінювати роботи які відправили студенти для викладача, або переглядати перевірені роботи та свою оцінку для студента. Для цього потрібно створити запит який реалізуватиме функцію оцінювання для викладача. Код такої функції виглядає так:

```
if (teacher.Courses.Contains(loadCourse))
            {
                Submition submition = db.Submitions.FirstOrDefault(s => s.Id ==
submitionId);
                submition.Mark = mark;
                db.Submitions.Update(submition);
                db.SaveChanges();

                return Ok(200);
            }
            else
            {
                return Forbid();
            }
```

Сторінка реалізує фінальну частину життєвого циклу такої сутності як «Submition». В результаті маємо останню веб сторінку проекту (рис. 3.11).

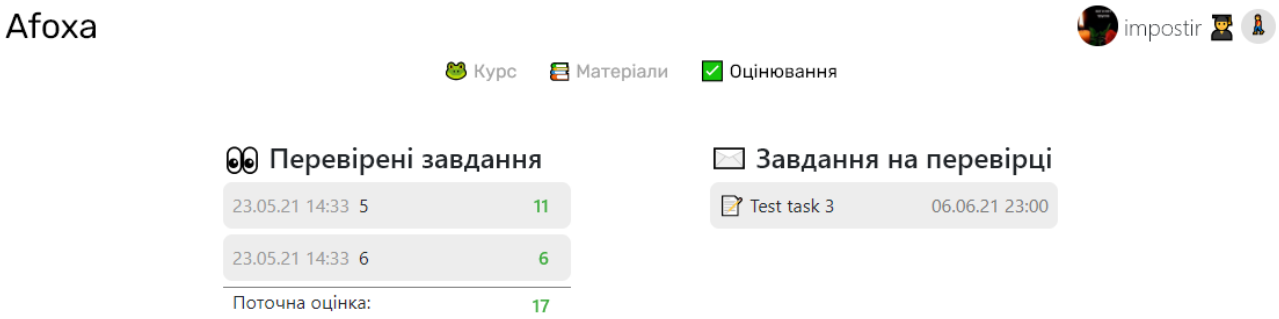

Рисунок 3.11 — Сторінка оцінювання для студента

Окрім веб інтерфейсу потрібно розробити інтерфейс для доступу до системи за допомогою API, для вирішення цієї задачі потрібно створити новий контролер та наповнити його методами які реалізують запити API, для прикладу розглянемо метод отримання інформації про курс:

```
[HttpGet]
        public ActionResult GetCourseInfo(string BotToken, int CourseId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                Dictionary<string, int> counters = new Dictionary<string, int>();
                Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);
                if (course != null)
                {
                    counters.Add("tasks", db.Tasks.Where(t => t.CourseId ==
CourseId).Count());
                    counters.Add("lects", db.Lections.Where(t => t.CourseId ==
CourseId).Count());
                    counters.Add("students", db.Students.Where(s =>
s.Courses.Contains(course)).Count());
                    return Json(counters);
                }
                else
                {
                    return NotFound();
                }
            }
            else
            {
                return Forbid();
            }
              }
```

Створене API дозволяє розроблювати сторонні засоби та розширення для платформи, згідно з завданням створимо чат бота який буде реалізувати основні функції та братиме на себе роль реалізації нотифікації користувачів. Створимо новий файл з розширенням «.py» та додамо до нього такий код:

```
bot = telebot.TeleBot(token);
```

Цей рядок створює обєкт бота на основі токена. Далі створимо функції які реалізують функції бота, наприклад функція для отримання даних про користувача виглядає так:

```
def get_user(user, chat_id):
    params = {'BotToken': token, 'TelegramId': user.id}
    r = requests.get(domen + "/API/GetUser", params=params)
    if r.status_code == requests.codes.ok:
        return r.json()
    else:
        bot.send_message(chat_id, "Реєструю, секундочку...")
        if register_user(user, chat_id):
            bot.send_message(chat_idd, "Тепер ви зареєстровані як студент та
можете ввійти в кабінет")

            return get_user(user, chat_id)
        else:
            bot.send_message(chat_id, "Хмм не можу зареєструвати, зверніться в
підтримку")
```

Далі необхідно розробити функцію яка буде приймати запити від користувачів та частину яка обробляє натискання кнопок, наприклад таким чином обробляється натискання на кнопку «Курси»:

```
if c.data == 'courses':
        courses = get_courses(c)
        keyboard = create_course_keyboard(courses)
        bot.send_message(c.message.chat.id, "Ваші курси:", reply_markup =
keyboard)
    bot.answer_callback_query(c.id)
```

Окрім цього необхідно створити механізм який буде надсилати оголошення які залишив викладач згідно розкладу, ця можливість реалізується за допомогою модулю який за розкладом запускає функцію запиту повідомлень які потрібно відправити і виконує відправку, код цієї функції виглядає так:

```
def send_notifications():
    params = {'BotToken': token}
    r = requests.get(domen + "/API/GetNotifications", params=params)
    if r.status_code == requests.codes.ok:
        data = r.json()
        for ad in data:
            course = get_course(ad['courseId'])
            text = '📢 Оголошення ( ' + course['emoji'] + " " + course['name'] +
' )\n\n'
            text += ad['title'] + "\n\n"
            text += ad['message']
            params = {'BotToken': token, 'CourseId': ad['courseId']}
            s = requests.get(domen + "/API/GetStudents", params=params)
            for student in s.json():
```

```
                bot.send_message(student['telegramChatId'], text)
            accept_sending(ad['id'])
            print("ad sending")
    else:
        print("[Warning!] Notification module error")
```

В результаті отримуємо чат-бота в Telegram який реалізує основні функції сервісу робота з ним представлена нижче (рис. 3.12)

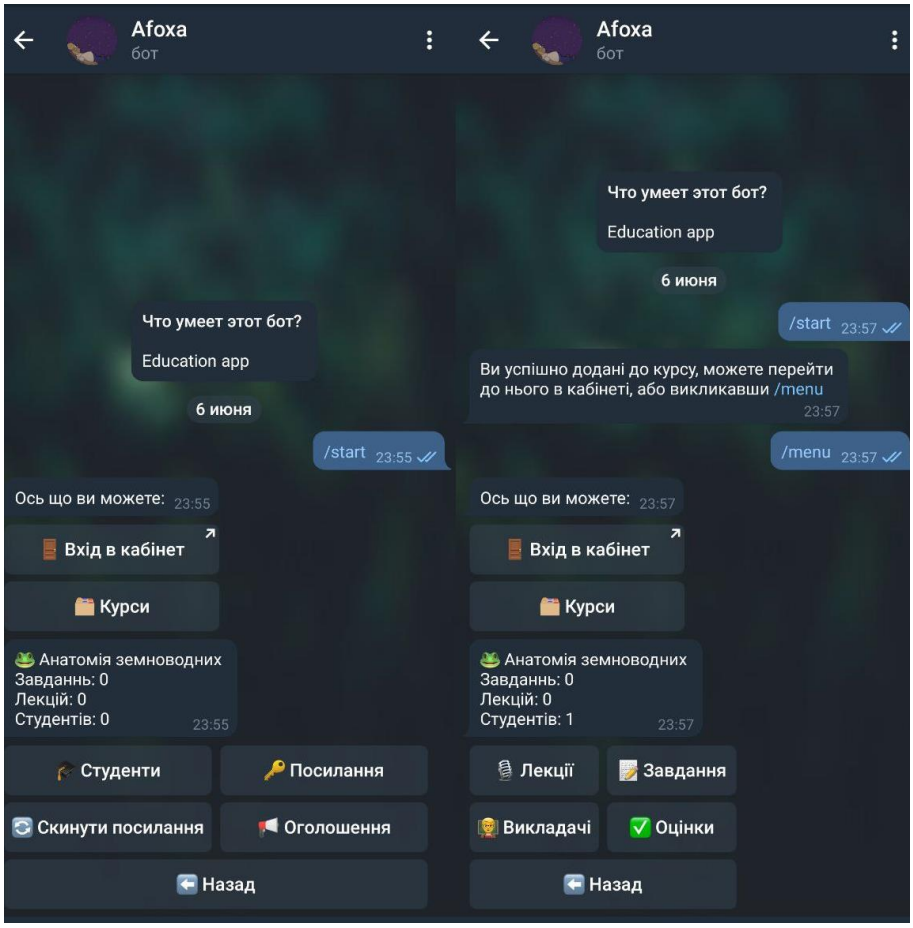

Рисунок 3.12 — Діалог з чат-ботом

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі проаналізована проблематика створення платформи для організації віддаленого або змішаного навчання. Було розглянуто рішення які нагадують тему проекту та виконують подібні функції. Опісля було проведено дослідження методів рішення задачі та аналіз інструментів. В результаті було обрано фреймворк ASP .Net для створення веб додатку і API та мову Python для реалізації чат-боту. Після цього кроку були сформовані чіткі вимоги до розроблюваного продукту, а для більш чіткого розуміння заданої вимогами задачі було створено схему структури додатку, Use-case діаграму, діаграми бази даних. Наступним кроком стала розробка платформи супутнього API та чат бота для платформи Telegram. В попередньому розділі був описаний процес створення додатку та бота, розглянуті ключові фрагменти програмного коду. В результаті роботи виконані всі вимоги які ставилися на етапі проектування, усунуті всі помилки та неточності які вдалося знайти особисто та за допомогою невеликої фокус групи знайомих які погодилися випробувати готовий програмний продукт. Більша частина результатів роботи була представлена на міжнародній науково-технічній-конференції студентів та молодих науковців. «Інформатика, математика, автоматика» (IMA – 2021) (Суми, 2021 р.) [16]. Рецензенти відзначили зовнішній вигляд та інтерфейс проекту, а також зручність використання і актуальність проблематики. В майбутньому проект можна розширити, чому сприяє використання API, або інтегрувати деякі з наробок в інші системи в тому числі в середовище змішаного навчання Сумського Державного Університету.

СПИСОК ЛІТЕРАТУРИ

1. Модель організації змішаного навчання у вищому навчальному закладі : звіт про НДР (остаточний) / кер. О. А. Шовкопляс. – Суми : СумДУ, 2020. – 91 с.
2. Tech against Coronavirus [Electronic resourse]. – Access mode : https://techagainstcoronavirus.com/.
3. Концепція розбудови єдиного освітнього середовища e-learning в СумДУ [Електронний ресурс]: Протокол вченої ради // Реєстр нормативної бази Сумського державного університету. – Режим доступу : https://normative.sumdu.edu.ua/?task=getfile&tmpl=component&id=800c72f4-f364-e411-afcd-001a4be6d04a&kind=1.
4. Telegram Bot API [Електронний ресурс] – Режим доступу: https://core.telegram.org/bots/api.
5. Чат-бот для бизнеса [Електронний ресурс] – Режим доступу: https://www.integrity.com.ua/activities/chatbots-for-business/.
6. AI and Chatbots in Education [Електронний ресурс] – Режим доступу: https://chatbotsmagazine.com/ai-and-chatbots-in-education-what-doesthe-futurehold-9772f5c13960.
7. Введение в Razor Pages [Електронний ресурс] – Режим доступу: https://metanit.com/sharp/aspnet5/29.1.php.
8. Троелсен Э. Язык программирования C# 2005 и платформа .NET 2.0 – 3-е изд. / Э. Троелсен. – М. : "Вильямс", 2008. – 1168 с
9. Freeman A. Pro ASP.NET MVC 3 Framework / A. Freeman, S. Sanderson. – New York: Apress, 2011. – 852 с.
10. Создание запросов LINQ на языке C# [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/linq/write-linq-queries.
11. T. Reenska. Model-View-Controller (MVC) / T. Reenska // JAOO, Орхус, 2003
12. pyTelegramBotAPI [Електронний ресурс] – Режим доступу: https://github.com/eternnoir/pyTelegramBotAPI.

13. Use Case Diagram Tutorial [Електронний ресурс] – Режим доступу: https://online.visual-paradigm.com/diagrams/tutorials/use-case-diagram-tutorial/.
14. Programming Entity Framework 1st Edition / J. Lerman. – O'Reilly Media, 2009. – 832 с.
15. Telegram Login Widget [Електронний ресурс] – Режим доступу: https://core.telegram.org/widgets/login.
16. Кончатний В. В. Платформа для дистанційного навчання з використанням Telegram Bot API //Матеріали та програма міжнародної науково-технічної конференції студентів та молодих вчених «Інформатика, математика, автоматика» (IMA – 2021) (Суми, 2021 р.), м. Суми, Україна. - Суми, 2021. - С. 61 - https://elitconference.sumdu.edu.ua.

ДОДАТОК А

Лістинг основних частин коду

Файл AccountController.cs:

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Afoxa.Models;
using Microsoft.AspNetCore.Identity;
using System.Collections.Generic;
using Telegram.Bot.Extensions.LoginWidget;
using System;
using Microsoft.Extensions.Configuration;

namespace Afoxa.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;
        private readonly Models.AppContext db;
        public IConfiguration AppConfiguration { get; set; }

        public AccountController(UserManager<User> userManager,
SignInManager<User> signInManager, Models.AppContext context)
        {
            var builder = new ConfigurationBuilder().AddJsonFile("conf.json");
            AppConfiguration = builder.Build();
            _userManager = userManager;
            _signInManager = signInManager;
            db = context;
        }

        [HttpPost]
        public async Task<IActionResult> Register(int id, string userName, string
firstName, string status, string chatId, string BotToken)
        {
            if (BotToken != AppConfiguration["BotToken"])
            {
                return Forbid();
            }
            User user = new User { Email = id.ToString(), UserName =
id.ToString(), TelegramFirstName = firstName, TelegramId = id, TelegramUserName =
userName, Role = status};
            // добавляем пользователя
            if (status == "Student")
            {
                user.TelegramChatId = chatId;
            }
            var result = await _userManager.CreateAsync(user);

            if (!result.Succeeded)
            {
                return BadRequest();
            }

            if (status == "Teacher")
```

```
            {
                Teacher teacher = new Teacher { UserId = user.Id };
                db.Teachers.Add(teacher);
            }

            if (status == "Student")
            {
                Student student = new Student { UserId = user.Id };
                db.Students.Add(student);
            }
            db.SaveChanges();
            await _userManager.AddToRoleAsync(user, status);

            return Ok();
        }

        [HttpPost]
        public async Task<IActionResult> Login(int id, string first_name, string
last_name, string username, string photo_url, string auth_date, string hash)
        {

            Dictionary<string, string> fields = new Dictionary<string, string>()
            {
                { "id", id.ToString() },
                { "first_name", first_name },
                { "auth_date", auth_date },
                { "hash", hash }
            };

            if (photo_url != null)
            {
                fields.Add("photo_url", photo_url);
            }

            if (username != null)
            {
                fields.Add("username", username);
            }

            if (last_name != null)
            {
                fields.Add("last_name", last_name);
            }

            var loginWidget = new LoginWidget(AppConfiguration["BotToken"]);

            try
            {
                if (loginWidget.CheckAuthorization(fields) ==
Authorization.Valid)
                {
                    var user = await
_userManager.FindByEmailAsync(id.ToString());

                    if(user != null)
                    {
                        //update user
                        user.TelegramUserName = username;
                        user.TelegramPhotoUrl = photo_url;
                        user.TelegramFirstName = first_name;
```

```
                        await _userManager.UpdateAsync(user);

                        //sign the user and go to home
                        await _signInManager.SignInAsync(user, isPersistent:
true);
                        return Ok(user.TelegramUserName + " sign in");
                    }
                    else
                    {
                        return BadRequest("Not found");
                    }
                }
                else
                {
                    return BadRequest(loginWidget.CheckAuthorization(fields));
                }
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync();
            return RedirectToAction("Login", "Account");
        }
    }
}
```

Файл CourseController.cs:

```
using Afoxa.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using AppContext = Afoxa.Models.AppContext;
using IdentityContext = Afoxa.Models.IdentityContext;


namespace Afoxa.Controllers
```

```
    {
        public class CourseController : Controller
        {
            private readonly AppContext db;
            private readonly IdentityContext idb;
            private readonly UserManager<User> _userManager;
            public IConfiguration AppConfiguration { get; set; }

            public CourseController(AppContext context, UserManager<User> userManager,
IdentityContext iContext)
            {
                var builder = new ConfigurationBuilder().AddJsonFile("conf.json");
                AppConfiguration = builder.Build();
                idb = iContext;
                db = context;
                _userManager = userManager;
            }

            private string setUserData()
            {
                string userName = User.Identity.Name;
                var user = _userManager.FindByNameAsync(userName).Result;
                var                              roleName                              =
_userManager.GetRolesAsync(user).Result.FirstOrDefault();

                ViewBag.Role = roleName;
                ViewBag.TgUser = user;

                return user.Id;
            }

            public bool InitializeAsync(int Id)
            {
                try
                {
                    Course course = db.Courses.FirstOrDefault(course => course.Id ==
Id);

                    db.Entry(course).Collection(c => c.Students).Load();
                    db.Entry(course).Collection(c => c.Teachers).Load();
                    List<User> Teachers = new List<User>();
```

```
                    List<User> Students = new List<User>();
                    List<Ad> Ads = new List<Ad>();

                    foreach (var ad in db.Adv.ToList())//check
                    {
                        if (ad.CourseId == course.Id)
                        {
                            Ads.Add(ad);
                        }
                    }

                    foreach (var teacher in course.Teachers)
                    {
                        var user = idb.Users.FirstOrDefault(user => user.Id == teacher.UserId);
                        Teachers.Add(user);
                    }
                    foreach (var student in course.Students)
                    {
                        var user = idb.Users.FirstOrDefault(user => user.Id == student.UserId);
                        Students.Add(user);
                    }

                    ViewBag.Ads = Ads;
                    ViewBag.Teachers = Teachers;
                    ViewBag.Students = Students;
                    ViewBag.Course = course;
                    return true;
                }catch
                {
                    return false;
                }
            }

            [Authorize(Roles = "Teacher")]
            public ActionResult Students(int Id)
            {
                if (Id == 0)
                {
```

```
                    return NotFound();
                }
                setUserData();
                if(InitializeAsync(Id))
                {
                    if (ViewBag.Teachers.Contains(ViewBag.TgUser))
                    {
                        ViewBag.ViewHeader = true;
                        ViewBag.Id = Id;
                        return View();
                    }
                    else
                    {
                        return Forbid();
                    }
                }else
                {
                    return Forbid();
                }
            }

            [Authorize]
            public ActionResult Details(int Id)
            {
                if(Id == 0)
                {
                    return NotFound();
                }
                setUserData();
                if (InitializeAsync(Id))
                {
                    if          (ViewBag.Teachers.Contains(ViewBag.TgUser)          ||
ViewBag.Students.Contains(ViewBag.TgUser))
                    {
                        ViewBag.ViewHeader = true;
                        ViewBag.Id = Id;
                        ViewBag.LectionsCount = db.Lections.Where(c => c.CourseId ==
Id).Count();
                        ViewBag.TasksCount   =   db.Tasks.Where(c   =>   c.CourseId   ==
Id).Count();
```

```
                        return View();
                    }
                    else
                    {
                        return Forbid();
                    }
                }
                else
                {
                    return Forbid();
                }
            }

            [Authorize]
            public ActionResult Materials(int Id)
            {
                if (Id == 0)
                {
                    return NotFound();
                }
                var UserId = setUserData();
                if (InitializeAsync(Id))
                {
                    if          (ViewBag.Teachers.Contains(ViewBag.TgUser)          ||
ViewBag.Students.Contains(ViewBag.TgUser))
                    {
                        ViewBag.ViewHeader = true;
                        ViewBag.Id = Id;
                        ViewBag.Lections = db.Lections.Where(c => c.CourseId == Id);
                        ViewBag.Tasks = db.Tasks.Where(c => c.CourseId == Id);
                        ViewBag.TasksCount   =   db.Tasks.Where(c   =>   c.CourseId   ==
Id).Count();
                        ViewBag.LectionsCount = db.Lections.Where(c => c.CourseId ==
Id).Count();
                        if (ViewBag.Role == "Student")
                        {
                            var StudentId = db.Students.FirstOrDefault(c => c.UserId
== UserId).Id;
                            ViewBag.StudentId = StudentId;
                            List<int> SubmitionsId = new List<int>();
```

```
                            foreach(var submition in db.Submitions.Where(c =>
c.StudentId == StudentId && c.CourseId == Id).ToList())
                            {
                                SubmitionsId.Add(submition.TaskId);
                            }
                            ViewBag.SubmitionsId = SubmitionsId;
                        }
                        return View();
                    }
                    else
                    {
                        return Forbid();
                    }
                }
                else
                {
                    return Forbid();
                }
            }

            [Authorize]
            public ActionResult Marks(int Id)
            {
                if (Id == 0)
                {
                    return NotFound();
                }
                setUserData();
                if (InitializeAsync(Id))
                {
                    if          (ViewBag.Teachers.Contains(ViewBag.TgUser)          ||
ViewBag.Students.Contains(ViewBag.TgUser))
                    {
                        if (ViewBag.Role == "Teacher")
                        {
                            List<Submition> UnmarkedSubmitions = db.Submitions.Where(c
=> c.CourseId == Id && c.Mark == -1).ToList();
                            List<User> SubmitionStudents = new List<User>();
                            List<Task> SubmitionTasks = new List<Task>();
                            foreach (Submition submition in UnmarkedSubmitions)
```

```
                {
                    string UserId = db.Students.FirstOrDefault(s => s.Id
== submition.StudentId).UserId;
                    SubmitionStudents.Add(idb.Users.FirstOrDefault(u   =>
u.Id == UserId));
                    SubmitionTasks.Add(db.Tasks.FirstOrDefault(t  =>  t.Id
== submition.TaskId));
                }
                ViewBag.UnmarkedSubmitions = UnmarkedSubmitions;
                ViewBag.SubmitionStudents = SubmitionStudents;
                ViewBag.SubmitionTasks = SubmitionTasks;
            }
            else
            {
                string userId = ViewBag.TgUser.Id;
                var student = db.Students.FirstOrDefault(s => s.UserId ==
userId);


                List<Submition> UnmarkedSubmitions = db.Submitions.Where(c
=> c.CourseId == Id && c.Mark == -1 && c.StudentId == student.Id).ToList();
                List<Task> SubmitionTasks = new List<Task>();
                foreach (Submition submition in UnmarkedSubmitions)
                {
                    SubmitionTasks.Add(db.Tasks.FirstOrDefault(t  =>  t.Id
== submition.TaskId));
                }
                ViewBag.UnmarkedSubmitions = UnmarkedSubmitions;
                ViewBag.SubmitionTasks = SubmitionTasks;
            }


            ViewBag.ViewHeader = true;
            ViewBag.Id = Id;
            return View();
        }
        else
        {
            return Forbid();
        }
```

```
                }
                else
                {
                    return Forbid();
                }
            }

            // POST: Course/CreateOrUpdate
            [HttpPost]
            [Authorize(Roles = "Teacher")]
            public ActionResult CreateOrUpdate(Course course)
            {
                if (!ModelState.IsValid)
                {
                    return BadRequest("invalid");
                }

               try
               {
                    string userName = User.Identity.Name;
                    var user = _userManager.FindByNameAsync(userName);
                    var teacher = db.Teachers.Include(c => c.Courses).Where(t =>
t.UserId == user.Result.Id).First();

                    db.Entry(teacher).Collection(c => c.Courses).Load();

                    if (course.Id == 0)
                    {
                        // add teacher to course
                        course.Teachers.Add(teacher);
                        db.Courses.Add(course);
                        db.SaveChanges();
                        course.Invite = generateInvite(course.Id);
                        db.SaveChanges();
                        return Ok(course.Id);
                    }
                    else
                    {
                        var loadCourse = db.Courses.FirstOrDefault(item => item.Id ==
course.Id);
```

```
                    if (loadCourse == null)
                    {
                        return NotFound();
                    }

                    // teacher is owner this course?
                    if (teacher.Courses.Contains(loadCourse))
                    {
                        loadCourse.Emoji = course.Emoji;
                        loadCourse.Name = course.Name;
                        loadCourse.About = course.About;
                        db.SaveChanges();
                        return Ok("Update");
                    }
                    else
                    {
                        return Forbid();
                    }
                }
            }
            catch (Exception)
            {
                return BadRequest();
            }
        }

        private string generateInvite(int id)
        {
            var                                chars                                =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
            var stringChars = new char[8];
            var random = new Random();

            for (int i = 0; i < stringChars.Length; i++)
            {
                stringChars[i] = chars[random.Next(chars.Length)];
            }

            var finalString = new String(stringChars);
```

```
                string invite = id + "_" + finalString;
                return invite;
            }

            // POST: Course/Delete
            [HttpPost]
            [Authorize(Roles = "Teacher")]
            public ActionResult Delete(int? id)
            {
                if (id == null)
                {
                    return BadRequest();
                }
                else
                {
                    var course = db.Courses.Where(i => i.Id == id).FirstOrDefault();
                    string userName = User.Identity.Name;
                    var user = _userManager.FindByNameAsync(userName);
                    var teacher = db.Teachers.Include(c => c.Courses).Where(t =>
t.UserId == user.Result.Id).First();
                    db.Entry(teacher).Collection(c => c.Courses).Load();

                    if(course == null)
                    {
                        return NotFound();
                    }

                    // teacher is owner this course?
                    if (teacher.Courses.Contains(course))
                    {
                        db.Courses.Remove(course);
                        db.SaveChanges();
                        return Ok("Deleted");
                    }
                    else
                    {
                        return BadRequest();
                    }
```

```
            }

        }

        // POST: Course/Revoke
        [HttpPost]
        [Authorize(Roles = "Teacher")]
        public ActionResult Revoke(int? id)
        {
            if (id == null)
            {
                return BadRequest();
            }
            else
            {
                var course = db.Courses.Where(i => i.Id == id).FirstOrDefault();
                string userName = User.Identity.Name;
                var user = _userManager.FindByNameAsync(userName);
                var teacher = db.Teachers.Include(c => c.Courses).Where(t =>
t.UserId == user.Result.Id).First();
                db.Entry(teacher).Collection(c => c.Courses).Load();

                if (course == null)
                {
                    return NotFound();
                }

                // teacher is owner this course?
                if (teacher.Courses.Contains(course))
                {
                    course.Invite = generateInvite(course.Id);
                    db.SaveChanges();
                    return Ok(course.Invite);
                }
                else
                {
                    return BadRequest();
                }

            }
```

```
}

// POST: Course/AddStudent
[HttpPost]
public ActionResult AddStudent(int? userId, string token, string BotToken)
{
    if (BotToken != AppConfiguration["BotToken"])
    {
        return Forbid();
    }
    if (!ModelState.IsValid || token == null || userId == null)
    {
        return BadRequest("invalid");
    }
    var course = db.Courses.FirstOrDefault(c => c.Invite == token);
    var user = idb.Users.FirstOrDefault(u => u.TelegramId == userId);
    var student = db.Students.FirstOrDefault(s => s.UserId == user.Id);

    if (course == null || student == null)
    {
        return NotFound();
    }

    course.Students.Add(student);
    try
    {
        db.SaveChanges();
        return Ok(200);
    }
    catch
    {
        return BadRequest();
    }
}

// POST: Course/Kick
[HttpPost]
[Authorize(Roles = "Teacher")]
public ActionResult Kick(int? courseId, string userId)
```

```
            {
                if (courseId == null || userId == null)
                {
                    return BadRequest();
                }
                else
                {
                    var course = db.Courses.Where(i => i.Id == courseId).FirstOrDefault();
                    string userName = User.Identity.Name;
                    var user = _userManager.FindByNameAsync(userName);
                    var teacher = db.Teachers.Include(c => c.Courses).Where(t => t.UserId == user.Result.Id).First();
                    db.Entry(teacher).Collection(c => c.Courses).Load();
                    var kickStudent = db.Students.FirstOrDefault(s => s.UserId == userId);

                    if (course == null)
                    {
                        return NotFound();
                    }

                    // teacher is owner this course?
                    if (teacher.Courses.Contains(course))
                    {
                        db.Entry(course).Collection("Students").Load();
                        course.Students.Remove(kickStudent);
                        db.SaveChanges();
                        return Ok("kick");
                    }
                    else
                    {
                        return BadRequest();
                    }

                }

            }
        }
    }
```

Файл APIController.cs:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using Afoxa.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Newtonsoft.Json;
using AppContext = Afoxa.Models.AppContext;

namespace Afoxa.Controllers
{
    public class APIController : Controller
    {
        private readonly AppContext db;
        private readonly IdentityContext idb;
        private readonly UserManager<User> _userManager;
        public IConfiguration AppConfiguration { get; set; }

        public APIController(AppContext context, UserManager<User> userManager,
IdentityContext iContext)
        {
            var builder = new ConfigurationBuilder().AddJsonFile("conf.json");
            AppConfiguration = builder.Build();
            idb = iContext;
            db = context;
            _userManager = userManager;
        }

        [HttpGet]
        public ActionResult GetUserSubmitions(int UserId, int CourseId, string
BotToken)
        {

            if (BotToken == AppConfiguration["BotToken"])
            {
                User user = idb.Users.FirstOrDefault(u => u.TelegramId == UserId);
```

```
                    Student student = db.Students.FirstOrDefault(s => s.UserId ==
user.Id);
                    var userSubmitions = db.Submitions.Where(s => s.StudentId ==
student.Id && s.CourseId == CourseId).ToList();
                    Dictionary<string, Submition> result = new Dictionary<string,
Submition>();
                    foreach (var submition in userSubmitions)
                    {
                        result.Add(db.Tasks.FirstOrDefault(t      =>      t.Id      ==
submition.TaskId).Id.ToString(), submition);
                    }

                    return Json(result);
                }
                else
                {
                    return Forbid();
                }
            }

            [HttpGet]
            public ActionResult GetTeachers(string BotToken, int CourseId)
            {
                if (BotToken == AppConfiguration["BotToken"])
                {
                    Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);
                    var teachers = db.Teachers.Where(t => t.Courses.Contains(course));
                    List<User> users = new List<User>();
                    foreach (var teacher in teachers)
                    {
                        users.Add(idb.Users.FirstOrDefault(u      =>      u.Id      ==
teacher.UserId));
                    }

                    return Json(users);
                }
                else
                {
                    return Forbid();
                }
```

```
        }

        [HttpGet]
        public ActionResult GetStudents(string BotToken, int CourseId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);
                var students = db.Students.Where(t => t.Courses.Contains(course));
                List<User> users = new List<User>();
                foreach (var student in students)
                {
                    users.Add(idb.Users.FirstOrDefault(u      =>      u.Id      ==
student.UserId));
                }

                return Json(users);
            }
            else
            {
                return Forbid();
            }
        }

        [HttpGet]
        public ActionResult GetTasks(string BotToken, int CourseId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                return Json(db.Tasks.Where(t => t.CourseId == CourseId));
            }
            else
            {
                return Forbid();
            }
        }

        [HttpGet]
        public ActionResult GetLections(string BotToken, int CourseId)
        {
```

```
            if (BotToken == AppConfiguration["BotToken"])
            {
                return Json(db.Lections.Where(l => l.CourseId == CourseId));
            }
            else
            {
                return Forbid();
            }
        }

        [HttpGet]
        public ActionResult GetCourseInfo(string BotToken, int CourseId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                Dictionary<string, int> counters = new Dictionary<string, int>();

                Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);

                if (course != null)
                {
                    counters.Add("tasks",  db.Tasks.Where(t  =>  t.CourseId  ==
CourseId).Count());
                    counters.Add("lects",  db.Lections.Where(t  =>  t.CourseId  ==
CourseId).Count());
                    counters.Add("students",        db.Students.Where(s        =>
s.Courses.Contains(course)).Count());
                    return Json(counters);
                }
                else
                {
                    return NotFound();
                }
            }
            else
            {
                return Forbid();
            }
        }
```

```
        [HttpGet]
        public ActionResult GetCourse(string BotToken, int CourseId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);
                if (course != null)
                {
                    return Json(course);
                }
                else
                {
                    return NotFound();
                }
            }
            else
            {
                return Forbid();
            }
        }

        [HttpGet]
        public ActionResult GetCourses(string BotToken, int TelegramId)
        {
            if (BotToken == AppConfiguration["BotToken"])
            {
                User user = idb.Users.FirstOrDefault(u => u.TelegramId ==
TelegramId);
                if (user == null)
                {
                    return NotFound();
                }
                else
                {
                    List<Course> courses = new List<Course>();
                    if (user.Role == "Teacher")
                    {
                        var teacher = db.Teachers.Where(u => u.UserId ==
user.Id).FirstOrDefault();
                        db.Entry(teacher).Collection(c => c.Courses).Load();
```

```
                courses = teacher.Courses.ToList();
            }
            else if (user.Role == "Student")
            {
                var student = db.Students.Where(u => u.UserId == user.Id).FirstOrDefault();
                db.Entry(student).Collection(c => c.Courses).Load();
                courses = student.Courses.ToList();
            }

            Dictionary<string, int> result = new Dictionary<string, int>();

            foreach (var course in courses)
            {
                result.Add(course.Emoji + " " + course.Name, course.Id);
            }

            return Json(result);
        }
    }
    else
    {
        return Forbid();
    }
}

[HttpGet]
public ActionResult GetUser(string BotToken, int TelegramId)
{
    if (BotToken == AppConfiguration["BotToken"])
    {
        User user = idb.Users.FirstOrDefault(u => u.TelegramId == TelegramId);
        if (user == null)
        {
            return NotFound();
        }
        else
        {
```

```
            return Json(user);
        }
    }
    else
    {
        return Forbid();
    }
}

[HttpGet]
public ActionResult RevokeInvite(string BotToken, int CourseId)
{
    if (BotToken == AppConfiguration["BotToken"])
    {
        Course course = db.Courses.FirstOrDefault(c => c.Id == CourseId);
        if (course != null)
        {
            course.Invite = generateInvite(CourseId);
            db.SaveChanges();
            return Ok();
        }
        else
        {
            return NotFound();
        }
    }
    else
    {
        return Forbid();
    }
}

[HttpGet]
public ActionResult GetAds(string BotToken, int CourseId)
{
    if (BotToken == AppConfiguration["BotToken"])
    {
        return Json(db.Adv.Where(l => l.CourseId == CourseId));
    }
    else
```

```
        {
            return Forbid();
        }
    }

    [HttpGet]
    public ActionResult GetNotifications(string BotToken)
    {
        if (BotToken == AppConfiguration["BotToken"])
        {
            long currentTime = DateTimeOffset.Now.ToUnixTimeSeconds();
            return Json(db.Adv.Where(l => l.UnixTime <= currentTime));
        }
        else
        {
            return Forbid();
        }
    }

    [HttpGet]
    public ActionResult AcceptSending(string BotToken, int AdId)
    {
        if (BotToken == AppConfiguration["BotToken"])
        {
            var ad = db.Adv.FirstOrDefault(a => a.Id == AdId);
            db.Adv.Remove(ad);
            db.SaveChanges();
            return Ok("Deleted");
        }
        else
        {
            return Forbid();
        }
    }

    private string generateInvite(int id)
    {
        var                                chars                                =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        var stringChars = new char[8];
```

```
                var random = new Random();

                for (int i = 0; i < stringChars.Length; i++)
                {
                    stringChars[i] = chars[random.Next(chars.Length)];
                }

                var finalString = new String(stringChars);

                string invite = id + "_" + finalString;
                return invite;
            }
        }
    }
```

Файл bot.py:

```
import telebot
import requests
import json
import configparser
from telebot import types
from keyboards import *
import datetime
import time
import schedule
import threading

config = configparser.ConfigParser()
config.read("settings.ini")

token = config["bot"]["token"]
domen = config["bot"]["domen"]
bot_name = config["bot"]["bot_name"]

bot = telebot.TeleBot(token);

def register_user(user, chat_id):
    params    =    {'id':    user.id,'userName':    user.first_name,'firstName':
user.first_name, 'status': 'Student', 'chatId': chat_id, 'BotToken': token }
    r = requests.post(domen + "/Account/Register", data=params)
    return r.status_code == requests.codes.ok
```

```
    def get_user(user, chat_id):
        params = {'BotToken': token, 'TelegramId': user.id}
        r = requests.get(domen + "/API/GetUser", params=params)
        if r.status_code == requests.codes.ok:
            return r.json()
        else:
            bot.send_message(chat_id, "Реєструю, секундочку...")
            if register_user(user, chat_id):
                bot.send_message(chat_id, "Тепер ви зареєстровані як студент та можете
ввійти в кабінет")

                return get_user(user, chat_id)
            else:
                bot.send_message(chat_id, "Хмм не можу зареєструвати, зверніться в
підтримку")

    def add_to_course(data, key):
        params = {'BotToken': token, 'UserId': data['telegramId'], 'Token': key}
        r = requests.post(domen + "/Course/AddStudent", data=params)
        return r.status_code == requests.codes.ok

    def get_courses(c):
        params = {'BotToken': token, 'TelegramId': c.from_user.id}
        r = requests.get(domen + "/API/GetCourses", params=params)
        if r.status_code == requests.codes.ok:
            return r.json()

    def get_course(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetCourse", params=params)
        if r.status_code == requests.codes.ok:
            return r.json()

    def get_course_info(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetCourseInfo", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
```

```
            result = "Завдання: " + str(data['tasks']) + "\nЛекцій: " +
str(data['lects']) + "\nСтудентів: " + str(data['students'])
            return result

    def show_course(user, message, course):
        course_user = get_user(user, message.chat.id)
        info = get_course_info(course['id'])
        text = course['emoji'] + " " + course['name'] + "\n" + info
        keyboard = get_course_keyboard(course_user['role'], course['id'])
        bot.edit_message_text(chat_id=message.chat.id, message_id=message.message_id,
text=text, reply_markup=keyboard)

    def get_lections(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetLections", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            if len(data) == 0:
                return 0
            else:
                text = "Лекції курсу: \n\n"
                for lect in data:
                    timestamp = datetime.datetime.fromtimestamp(lect['unixTime'])
                    text += lect['topic'] + "        "+ timestamp.strftime('%H:%M
%d.%m.%Y') + "\n"
                    text += "[Посилання на матеріали](" + lect['materialLink'] + ")"
                    if lect['conferenceLink'] is None:
                        text += " 👀\n\n"
                    else:
                        text += "\n[Посилання на конференцію](" +
lect['conferenceLink'] + ") 🔗\n\n"
                return text

    def get_tasks(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetTasks", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            if len(data) == 0:
                return 0
```

```
            else:
                text = "Завдання : \n\n"
                for task in data:
                    timestamp = datetime.datetime.fromtimestamp(task['unixTime'])
                    text += task['topic'] + "    "+ timestamp.strftime('до %d.%m.%Y')
+ "\n"
                    text += "[Посилання на завдання](" +  task['link'] + ")\n\n"
                return text

    def get_teachers(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetTeachers", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            text = "Викладачі: \n\n"
            for teacher in data:
                text     +=     teacher['telegramFirstName']     +     "\n@"     +
teacher['telegramUserName'] + "\n\n"
            return text

    def get_students(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetStudents", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            if len(data) == 0:
                return 0
            else:
                text = "Студенти: \n\n"
                counter = 1
                for student in data:
                    text += str(counter) + ") " + student['telegramFirstName'] + " (
@" + student['telegramUserName'] + " )\n\n"
                return text

    def get_marked_submition(course_id, user_id):
        params = {'BotToken': token, 'CourseId': course_id, 'UserId': user_id}
        r = requests.get(domen + "/API/GetUserSubmitions", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
```

```
            if len(data.keys()) == 0:
                return 0
            else:
                text = "Оцінені завдання:\n"
                finalmark = 0
                for key in data.keys():
                    t = requests.get(domen + "/Task/Get", params={"Id": int(key)})
                    task = t.json();
                    if data[key]['mark'] >= 0:
                        timestamp                                                    =
datetime.datetime.fromtimestamp(data[key]['unixTime'])
                        text += ["+ task['topic'] +"](" +  task['link'] + ")     " +
timestamp.strftime('%H:%M %d.%m.%Y') + "\n"
                        text += "[Відповідь](" +  data[key]['link'] + ")     Оцінка: "
+ str(data[key]['mark'])  + "\n\n"
                        finalmark += data[key]['mark']
                if finalmark >= 0:
                    text += "Поточна оцінка: " + str(finalmark)
                return text

    def get_ads(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/GetAds", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            if len(data) == 0:
                return 0
            else:
                text = "Заплановані оголошення:\n\n"
                for ads in data:
                    timestamp = datetime.datetime.fromtimestamp(ads['unixTime'])
                    text += ads['title'] + "          " + timestamp.strftime('%H:%M
%d.%m.%Y') + "\n\n"
                return text

    def revoke_invite(course_id):
        params = {'BotToken': token, 'CourseId': course_id}
        r = requests.get(domen + "/API/RevokeInvite", params=params)
        return r.status_code == requests.codes.ok
```

```
    def accept_sending(ad_id):
        params = {'BotToken': token, 'AdId': ad_id}
        requests.get(domen + "/API/AcceptSending", params=params)

    def send_notifications():
        params = {'BotToken': token}
        r = requests.get(domen + "/API/GetNotifications", params=params)
        if r.status_code == requests.codes.ok:
            data = r.json()
            for ad in data:
                course = get_course(ad['courseId'])
                text = '📢 Оголошення ( ' + course['emoji'] + " " + course['name'] +
' )\n\n'
                text += ad['title'] + "\n\n"
                text += ad['message']
                params = {'BotToken': token, 'CourseId': ad['courseId']}
                s = requests.get(domen + "/API/GetStudents", params=params)
                for student in s.json():
                    bot.send_message(student['telegramChatId'], text)
                accept_sending(ad['id'])
                print("ad sending")
        else:
            print("[Warning!] Notification module error")

    @bot.message_handler(commands=['start'])
    def start(message):
        user_data = get_user(message.from_user, message.chat.id)
        com = message.text.split(" ")
        if len(com) > 1:
            if add_to_course(user_data, com[1]):
                bot.send_message(message.chat.id, "Ви успішно додані до курсу, можете
перейти до нього в кабінеті, або викликавши /menu")
            else:
                bot.send_message(message.chat.id,  "Нажаль немає доступу до цього
курсу, перевірте посилання")
        else:
            menu(message)

    @bot.message_handler(commands=['menu'])
    def menu(message):
```

```
            get_user(message.from_user, message.chat.id)
            bot.send_message(message.chat.id, "Ось що ви можете:", reply_markup =
menu_keyboard)

        @bot.callback_query_handler(func=lambda call: True)
        def callback_inline(c):
            if c.data == 'courses':
                courses = get_courses(c)
                keyboard = create_course_keyboard(courses)
                bot.send_message(c.message.chat.id, "Ваші курси:", reply_markup =
keyboard)
                bot.answer_callback_query(c.id)

            if c.data == 'none':
                bot.answer_callback_query(c.id)

            if c.data == 'return_to_course_list':
                courses = get_courses(c)
                keyboard = create_course_keyboard(courses)
                bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id, text="Ваші курси", reply_markup=keyboard)
                bot.answer_callback_query(c.id)

            com = c.data.split("_")

            if len(com) == 2:
                if com[0] == "course":
                    course = get_course(com[1])
                    if course is None:
                        bot.send_message(c.message.chat.id, "Щось пішло не так(")
                        bot.answer_callback_query(c.id)
                    else:
                        show_course(c.from_user, c.message, course)
                        bot.answer_callback_query(c.id)

            if len(com) == 3:
                keyboard = types.InlineKeyboardMarkup(row_width=1)
                back_btn = types.InlineKeyboardButton(text="⬅ Назад",
callback_data="course_" + com[1])
                if com[2] == "lectlist":
```

```
                text = get_lections(com[1])
                if text == 0:
                    bot.answer_callback_query(c.id,   show_alert=True,   text="Лекцій
немає 🤷‍♀️")
                else:
                    keyboard.add(back_btn)
                    bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,           parse_mode="Markdown",           text=text,
reply_markup=keyboard, disable_web_page_preview=True)
                    bot.answer_callback_query(c.id)

            if com[2] == "tasklist":
                text = get_tasks(com[1])
                if text == 0:
                    bot.answer_callback_query(c.id,  show_alert=True,  text="Завданнь
немає 🤷‍♀️")
                else:
                    keyboard.add(back_btn)
                    bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,           parse_mode="Markdown",           text=text,
reply_markup=keyboard, disable_web_page_preview=True)
                    bot.answer_callback_query(c.id)

            if com[2] == "teachers":
                text = get_teachers(com[1])
                keyboard.add(back_btn)
                bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,           text=text,           reply_markup=keyboard,
disable_web_page_preview=True)
                bot.answer_callback_query(c.id)

            if com[2] == "rate":
                text = get_marked_submition(com[1], c.from_user.id)
                if text == 0:
                    bot.answer_callback_query(c.id,   show_alert=True,   text="Немає
результатів 🤷‍♀️")
                else:
                    keyboard.add(back_btn)
```

```
                    bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,          text=text,          parse_mode="Markdown",
reply_markup=keyboard, disable_web_page_preview=True)
                    bot.answer_callback_query(c.id)

            if com[2] == "students":
                text = get_students(com[1])
                if text == 0:
                    bot.answer_callback_query(c.id,    show_alert=True,    text="Немає
студентів, запросіть їх за допомогою посилання 👩")
                else:
                    keyboard.add(back_btn)
                    bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,          text=text,          parse_mode="Markdown",
reply_markup=keyboard, disable_web_page_preview=True)
                    bot.answer_callback_query(c.id)

            if com[2] == "link":
                course = get_course(com[1])
                keyboard.add(back_btn)
                text  =  "Посилання  для  запрошення:\nhttps://t.me/"  +  bot_name  +
"?start=" + course['invite']
                bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,          text=text,          reply_markup=keyboard,
disable_web_page_preview=True)
                bot.answer_callback_query(c.id)

            if com[2] == "revoke":
                if revoke_invite(com[1]):
                    bot.answer_callback_query(c.id,  show_alert=True,  text="Посилання
скинуто, старе запрошення більше не дійсне")

            if com[2] == "ads":
                text = get_ads(com[1])
                if text == 0:
                    bot.answer_callback_query(c.id,    show_alert=True,    text="Немає
запланованих оголошенн 👩")
                else:
                    keyboard.add(back_btn)
```

```
                    bot.edit_message_text(chat_id=c.message.chat.id,
message_id=c.message.message_id,          text=text,          parse_mode="Markdown",
reply_markup=keyboard, disable_web_page_preview=True)
                    bot.answer_callback_query(c.id)

    def shed():
        schedule.every(60).seconds.do(send_notifications)
        while True:
            schedule.run_pending()
            time.sleep(1)

    shed_thread = threading.Thread(target=shed)
    shed_thread.start()

    try:
        print("> Bot runing succesful!")
        bot.polling(none_stop=True, interval=0)
    except:
        print("> connecting...")
        bot.polling(none_stop=True, interval=0)
```