Приватний заклад вищої освіти

Одеський технологічний університет «ШАГ»

Кафедра інформаційних технологій та фундаментальної підготовки

випускна кваліфікаційна робота

«ІНФОРМАЦІЙНА СИСТЕМА ПІДТРИМКИ РАНЖОВАНОГО УПРАВЛІННЯ БІЗНЕС-ЗАДАЧАМИ (за прикладом сервісу Notion»

на здобуття бакалаврського рівня вищої освіти

зі спеціальності «122 Комп'ютерні науки»

Розробники:

№ з/п	ПІБ	Роль
1	Червона А.О	Розробка UI/UX - дизайну
2	Боярчук Б.М	Розробка (backend)
3	Котліков А.О	Розробка (frontend)
4	Величко В.М	Розробка UI/UX - дизайну

Ментори:

№ з/п	ПІБ, посада	Напрямок
1	Черкезян Крістіне Арутюнівна	Розробка ПЗ
2	Судавная Т.О.	Графічний дизайн

Автор звіту

_____ Боярчук Богдан Михайлович

АНОТАЦІЯ

Боярчук Б.М. Онлайн додаток для створення та збереження нотаток у різних форматах: Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки».- Одеса: ОТУШ, 2024

Програма присвячена створенню онлайн ресурсу для створення та зберігання нотаток клієнтів/користувачів, де дані кожного користувача зберігаються у безпечному форматі та з належним захистом. Основними функціями є: реєстрація, авторизація, автентифікація, створення нотаток, налаштування профілю користувача, пошук, збереження нотаток, видалення та відновлення нотаток, відправлення листів електронною поштою, створення шаблонних нотаток, редагування нотаток, зміна зовнішнього вигляду інтерфейсу.

Актуальність нашого проекту будується на потребах користувачів, їхніх ідеях та бажаннях, можливостях створювати та зберігати контент, який їм потрібен для повсякденного чи робочого життя. Допоможе зібрати свої думки та структурувати свої плани.

У нашій роботі найбільше часу було витрачено на правильне структурування та зберігання нотаток різних форматів, безпеки та цілісності доступу до даних користувачів, системи реєстрації та редагування нотаток.

Багато часу було витрачено на дослідження сучасних алгоритмів та технологій для коректної та ефективної роботи клієнтської та серверної частин програми. Як цілі дослідження було поставлено:

- проаналізувати та вибрати технології для написання клієнтської сторони програми з урахуванням швидкості відображення веб-сторінок.
- вибрати технології для реалізації серверної частини програми
- реалізувати безпечне отримання даних від клієнтської частини з урахуванням можливих помилок, не відповідних даних та шкідливих програм.

- впровадження продуктивних сучасних технологій у додаток для прискорення та спрощення його роботи, таких як Entity Framework Core, LINQ, jwt bearer tokens.

Протягом роботи були використані різні методи для досягнення поставленої мети, зокрема хочеться відзначити використання технік парного програмування в процесі роботи та чіткий розподіл обов'язків у ході розробки. У процесі виникали проблеми з нестачею знань у галузі деяких технологій, які доводилося доучувати та використовувати на ходу. Такими технологіями є ASP WEB API, ASP .NET Identity.

Чому людям потрібні додатки для нотаток і планування?

Усі ми стикаємося з необхідністю зберігати та систематизувати інформацію. Традиційні методи, такі як паперові блокноти або стікери, хоча й мають свою користь, але не завжди здатні задовольнити вимоги сучасного темпу життя. Ось кілька ключових причин, чому цифрові інструменти для нотаток і планування стали настільки популярними:

1. **Управління великим обсягом інформації**:
    - Щодня ми маємо справу з величезною кількістю інформації: від робочих завдань до особистих нагадувань. Веб-застосунок для нотаток дозволяє ефективно організувати цю інформацію, забезпечуючи доступ до неї в будь-який час і з будь-якого пристрою.
2. **Підвищення продуктивності**:
    - Планування завдань і проектів допомагає чітко бачити свої пріоритети, уникати перевантаження та ефективно керувати своїм часом. Завдяки цифровим інструментам можна створювати гнучкі та адаптивні плани, які легко коригуються залежно від обставин.
3. **Покращення співпраці**:
    - У багатьох проектах потрібна спільна робота. Додатки для нотаток і планування дозволяють легко ділитися інформацією з колегами, організовувати спільні зустрічі та координувати дії всієї команди. Це значно спрощує комунікацію і допомагає досягати спільних цілей.
4. **Творче самовираження**:
    - Веб-застосунки для нотаток часто пропонують безліч інструментів для візуалізації ідей, упорядкування інформації через простий і зрозумілий користувацький інтерфейс. Це ідеально підходить для

творчих особистостей, які шукають зручний спосіб організувати свої думки та проекти.

5. **Зручність і доступність**:
    - Використання цифрових нотаток забезпечує доступ до інформації з будь-якого місця. Завдяки хмарним технологіям, усі ваші дані завжди під рукою, незалежно від того, чи ви перебуваєте в офісі, вдома або в дорозі.

Як наш проект задовольняє ці потреби?

Розробляючи наш веб-застосунок, ми прагнули створити інструмент, який би не тільки відповідав вимогам користувачів, але й перевершував їхні очікування. Основна ідея нашого проекту полягає в наданні універсального і гнучкого рішення для організації інформації, яке можна адаптувати до будь-яких потреб.

1. **Різноманітність типів сторінок**:
    - Ми запропонували п'ять основних типів сторінок: порожні сторінки для простих нотаток, дошки для візуального управління завданнями, списки для організації завдань, таблиці для структурованих даних та галереї для зберігання зображень. Це дозволяє користувачам вибрати оптимальний формат для кожного типу інформації.
2. **Інтуїтивно зрозумілий інтерфейс**:
    - Завдяки інтеграції з сучасними UI/UX-інструментами, такими як Figma, і використанню Angular, ми створили зручний і приємний у використанні інтерфейс. Він дозволяє легко створювати, редагувати та організовувати сторінки.
3. **Гнучкість і кастомізація**:
    - Користувачі можуть налаштовувати сторінки за своїм смаком, змінювати зовнішній вигляд елементів і адаптувати інтерфейс під свої потреби. Це забезпечує індивідуальний підхід і підвищує зручність використання.
4. **Ефективне збереження та доступ до даних**:
    - Інформація зберігається автоматично при виході зі сторінки, що дозволяє користувачам зосередитися на своїх завданнях без необхідності постійно думати про збереження даних.

Наша мета – створити веб-застосунок, який допоможе користувачам ефективно керувати своїм часом, завданнями та ідеями. У світі, де кожна хвилина має значення, ми прагнемо надати інструмент, який допоможе зосередитися на важливому і досягати успіху. Наш веб-застосунок для нотаток

і планування − це відповідь на виклики сучасного життя, який забезпечує зручність, гнучкість і продуктивність у повсякденній діяльності.

Технічне завдання до проекту

Інформаційна система підтримки ранжованого управління бізнес-задачами (за прикладом сервісу Notion)

Загальний опис.
Програмне забезпечення (ПЗ) призначене для створення мережевої системи яка дозволяє управляти бізнес процесами з точки зору менеджменту, адміністрування та документування. Функції системи мають забезпечити повний життєвий цикл управління контентом: створення, актуалізація, модифікація та знищення, а також встановлення скорингових параметрів (оцінок) наявного контенту. Головна особливість системи полягає у тому, що вона повинна містити різні засоби подання (візуалізації) даних, ведення дискусій у межах окремої бізнес-задачі.

Призначення:
ПЗ орієнтується на створення онлайн системи на кшталт автоматизованого робочого місця менеджера. Визначальною функцією системи є можливість спільної (колективної) роботи з контентом. ПЗ створюється за прикладом сервісу Notion, у той же час має складатись за принципом універсальності, мати змогу змінити цільове призначення.

Вимоги до функціоналу. Обов'язкова частина
Головна сторінка
Стартовий інтерфейс ПЗ (головна сторінка або активність) має за мету популяризацію системи. На ньому має бути забезпечено відображення популярних показників як-то загальна кількість користувачів та інформаційних одиниць, вибірка з найбільш популярного контенту та нових дописів, зовнішня реклама. Значний акцент має бути зроблений на детальному пошуку контенту за принципом семантичного аналізу. Головна сторінка обмежує функції системи для незареєстрованих користувачів лише переглядом та пошуком, інші функції надаються у разі авторизації користувача.

Реєстрація користувачів:

Користувач ПЗ має змогу зареєструватись як за допомогою авторизаційних даних (логіну/телефону/E-mail/паролю), так і за допомогою облікових записів популярних соціальних мереж (Google, Facebook, тощо).
При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - телефон та електронна пошта. На тестовому етапі, за умови ускладнення замовлення послуг телефонії чи СМС, можливе використання лише електронної пошти. Засіб оперативного контактування при реєстрації підлягає верифікації через відповідь на тестовий виклик чи надісланий лист, або введення коду, пересланого через повідомлення довільного типу.
Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.
За умови що користувач забув пароль, ЗОК має бути використаний для його відновлення. Схема відновлення паролю жорстко не регламентується, рекомендовано вживання тимчасових паролів (one time passwords, OTP), але допускаються й інші схеми.
Також передбачається наявність у системі окремих користувачів з додатковими правами: модераторів та адміністраторів. Реєстрація кореневого адміністратора системи має бути реалізоване при інсталяції системи, наступним користувачам змінюють права наявні адміністратори.
Звичайні користувачі системи поділяються на менеджерів та виконавців. Менеджери є представниками компанії, мають, зокрема, право змінювати офіційну документацію (Company wiki). При реєстрації менеджеру вимагається додаткова перевірка його прав на представництво компанії. На тестовому етапі достатньо лише факту надсилання користувачеві запиту на відповідні документи, надісланого через ЗОК.
Для користувачів усіх категорій вимагається формальна валідація усіх реєстраційних даних (відповідність загальновживаним або власним шаблонам).

Особисті кабінети користувачів:
Особисті кабінети різних груп користувачів мають забезпечувати однакову базову функціональність, додаючи менеджерам окремі можливості щодо редагування контенту. Дозволяється на початковому етапі (до впровадження механізмів перевірки ліцензій надавачів послуг) функції надавача послуг делегувати користувачам з підвищеним статусом керування (адміністраторам) через що кабінет надавача послуги може виконуватись як панель адміністрування контента.
Особистий кабінет має забезпечити необхідні засоби для

- Управління даними та метаданими (нотаток, ресурсів та їх описів, задач тощо)
- Створення, редагування, видалення (блокування), перегляд розміщеного контента.
- Перегляд поточної активності щодо власного контенту (коментарів, оцінок, історії перегляду/завантажень)

- Контактування з іншими користувачами, що зареєстровані в системі, за допомогою ЗОК або через внутрішні засоби системи (чат або мессенджер)

Розширені засоби особистого кабінету менеджера має забезпечити необхідні засоби для

- Постановки задач іншим користувачам тієї ж організації
- Пошуку контенту інших користувачів організації за метаданими, групування та фільтрування результатів за різними ознаками, наявними у знайденому контенті, а також за цільовою групою. Для об'ємних результатів має бути забезпечена пагінація.
- Редагування контенту організації (Company wiki), підтвердження іншим користувачам їх належність до організації

Також засобами особистих кабінетів має бути передбачено можливість

- Зміни персональних даних, зокрема, контактних засобів. У разі зміни важливих даних, що раніше проходили верифікацію, вимагається їх повторна верифікація.
- Зворотного зв'язку - написання відгуку (коментаря) або рекомендації, що буде передано іншому користувачеві або адміністратору системи.

Вимоги до функціоналу. Опціональна частина

Модерація

Для забезпечення дотримання загальноприйнятих норм спілкування, а також з метою недопущення публікування контенту, який містить заборонені компоненти (ненормативні висловлювання, заклики до протиправних дії, приниження чи дискримінація, тощо), усі інформаційні одиниці, а також створені користувачами метадані (маркери) мають в обов'язковому порядку пройти погодження (модерацію). Право надати погодження належить кореневому адміністратору системи, а також модераторам. При погодженні інформаційна одиниця має зберігати ідентифікатор користувача, який надав погодження.

Привілейовані функції

З метою покращення економічної ефективності проекту передбачається наявність додаткових функцій системи, що доступні лише за умови платної підписки (або внутрішніх досягнень) користувачів. До таких функцій можна віднести окремі групи контенту, що є доступними тільки для привілейованих користувачів, відображення ЗОК інших користувачів (для прямого контактування), розширені засоби фільтрації та групування. Також можливе встановлення обмеження щодо кількості опублікованих одиниць контенту або замовлень, варіантів перегляду звітного матеріалу (канбан дошок тощо).

Архітектура

ПЗ має бути спроектоване за клієнт-сервеною розподіленою архітектурою з чітко відокремленими структурними вузлами системи:

Mockup - візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.
Backend - серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.
Web App - сайт або застосунок, доступний з веб-простору, що забезпечує візуальний інтерфейс користувачів усіх категорій
Mobile App (опціонально) - застосунок, що встановлюється на мобільних пристроях (смартфон, планшет). Дана частина виконується за наявності у команді проекту трьох і більше програмістів.

Усі модулі ПЗ (окрім візуальної моделі Mockup) повинні обов'язково включати до свого складу засоби

- тестування функціональності та модульні тести,
- розгортання на новому пристрої (інсталяції) із встановленням початкових таблиць у БД та кореневого користувача-адміністратора

Технології
Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту).
Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на кшталт:

- root
    - Backend
    - Web
    - Mobile
    - Mockup (вихідні файли або посилання на зовнішній проект)

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

Backend - .NET або Java, або NodeJS технології з хмарним розміщенням (Azure, AWS, Google Cloud),
Web - серверні технології (ASP, Razor Pages, JSP) або React чи Angular
Mobile - Java, або Kotlin, або Flutter, або React Native
Mockup - визначається ментором з дизайну

Команда проекту
Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проекту.
Мінімальна команда проекту - два учасники, у такому складі один учасник виконує модуль Backend та програмну частину Web, другий - модель Mockup та розміткову частину Web.
Рекомендована команда проекту - три учасники, у такій команді кожен з учасників відповідає за одну з частин: Mockup, Backend, Web.

Дозволяється включення до команди більшої кількості учасників, у такому разі модуль Mobile App набуває обов'язкової реалізації. Також модуль Mockup може бути поділений на моделі Web та Mobile частин.

Звітність
За результатами виконання завдань із створення ПЗ кожен з учасників команди проекту складає власний (індивідуальний) звіт, що складається з практичної частини та автореферату. Практична частина є прямим результатом роботи і подається у вигляді посилання на репозиторій (чи інший URL) або у вигляді архіву на довільному носієві даних. Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.
Автореферат містить описову частину, у якій розкриваються основні концепти проекту, деталізуються завдання до виконавця, встановлюється актуальність роботи, її об'єкт та предмет, обґрунтовується вибір технологій та відзначається шлях виконання та ступінь досягнення поставлених задач, посилання на джерела інформації.

Архітектура проекту

Проект реалізований з використанням клієнт-серверної архітектури. Фронтенд розроблений на основі Angular (TS, HTML), що забезпечує динамічний та інтерактивний користувацький інтерфейс. Бекенд написаний на мові програмування C#, що забезпечує надійну та масштабовану серверну частину.

Клієнт-серверна архітектура дозволяє ефективно розділити обробку даних між клієнтською та серверною частинами, забезпечуючи швидкий відгук користувацького інтерфейсу та надійне збереження даних на сервері.

Команда проекту

Над проектом працювала команда з чотирьох осіб. Програмуванням займалися два розробники: один (Котліков Альберт) спеціалізувався на фронтенді, використовуючи Angular, а інший (Боярчук Богдан) працював над бекендом на мові C#. Дизайном сайту займалися два дизайнери (Червона Анна та Величко Владислав), які розробили зручний та привабливий

користувацький інтерфейс. Завдяки їх спільним зусиллям, проект був реалізований на високому рівні якості.

РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

для реалізації серверної частини були використані такі технології:

Як програмне забезпечення: Microsoft Visual Studio, Radmin VPN, SQL Server Management Studio

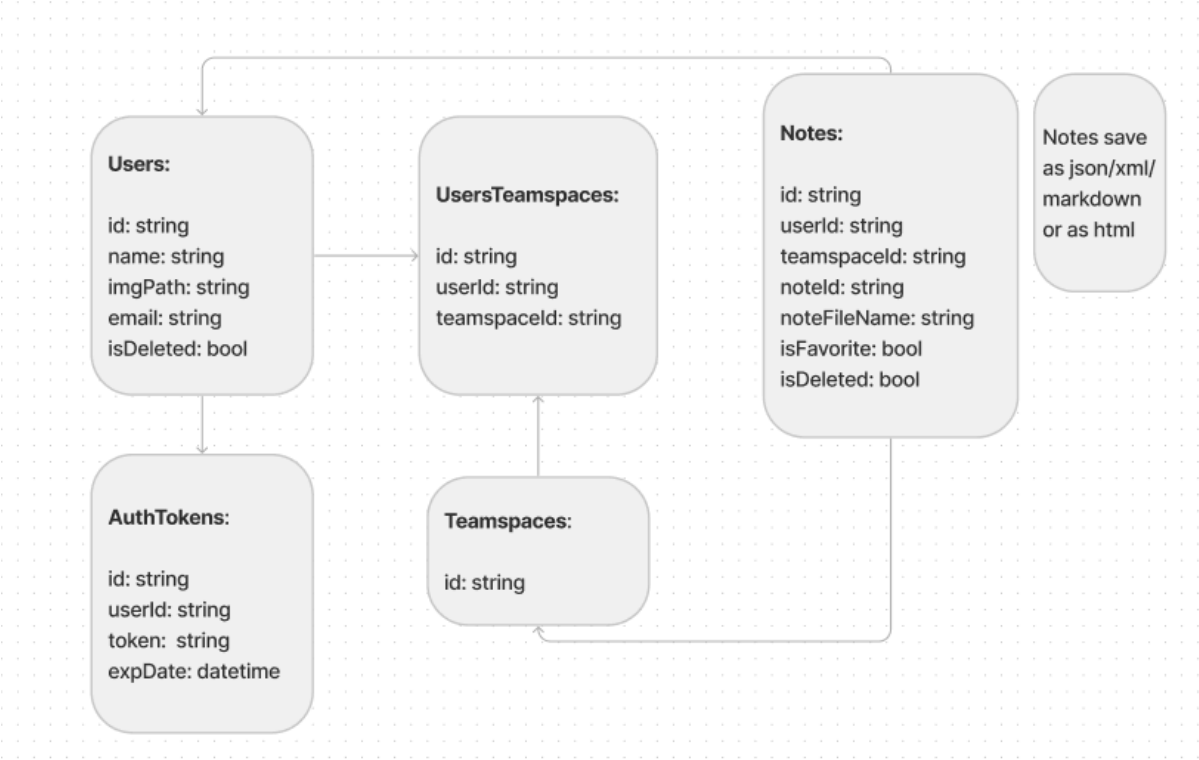
В якості баз даних і технологій зберігання та передачі даних: MS SQL, JSON

Як мови програмування: C#, Transact-SQL

Як основні технології: ASP.NET Core, Entity Framework Core, LINQ

Як додаткові пакети розробки: Newtonsoft.Json, JwtBearer, Swashbuckle

Спочатку роботи було спроектовано таблицю основних сутностей на серверній частині проекту, яка згодом допрацьовувалась у міру розширення проекту. Далі мені довелося виділити технології, з якими я хочу працювати і які найбільше відповідають вимогам мого проекту. Технології вибиралися за принципом ефективності рішень у розробці сучасних проектів, часу витраченого на додаткове освоєння раніше не вивчених технологій та сумісності та зручності використання між собою.



У проекті я зосереджувався на оптимізації, щоб знизити навантаження на сервер у стані роботи проекту.

На початку розробки я задумався над тим, що мені потрібно протягом роботи, і було прийнято рішення написати необхідні сервіси для роботи проекту. Такими сервісами є Email для надсилання листів на пошту користувача для передачі інформації у вигляді коду авторизації, сервіс Hash для хешування коду (тимчасового пароля користувача), сервіс KDF для створення ключа та згодом його безпечного зберігання в базі даних, сервіс Random для формування випадкового рядка, що відправляється на пошту для авторизації користувача або створення випадкового рядка, який буде виступати сіллю для збереження пароля користувача в базі даних.

Далі як об'єкт розробки був взятий конфіг файл, який описував важливі частини структури сервера, такі як шлях до папок користувачів, дані smtp для входу і роботи з поштою з якою згодом будуть відправлятися коди підтвердження на частину користувача, рядок підключення до бази даних, унікальний ключ для генерації jwt токенів.

Спочатку для базового тестування була використана технологія swagger для перевірки базової функціональності серверної частини проекту, яка пізніше була замінена технологією Kestrel для розгортання нашого сервера та можливості працювати разом із клієнтською частиною.

Також була використана технологія Radmin VPN для підключення в одну локальну мережу, перебуваючи при цьому в різних частинах країни. Далі були налаштовані політики CORS для взаємодії клієнтської та серверної частини проекту.

Під час розробки третина часу була проведена за методикою розробки Парне програмування, яка має на увазі рішення завдання спільно "працюючи за одним комп'ютером", ми працювали віддалено використовуючи для зв'язку і демонстрації екрану в реальному часі додаток Discord шляхом пропозиції різних рішень по ходу написання коду і приходу до найбільш оптимізованого та кращого варіанту.

Наступним кроком було налагодити механізм реєстрації та авторизації, проте в процесі планування ми вирішили використовувати більш сучасний підхід для створення зручнішого рішення для користувача і використовувати так звану без парольну систему, в якій користувач як логін вводив би свою пошту, а в якості пароля був б код підтвердження який надходить на його пошту і є одноразовим паролем. Спочатку було прийнято неправильне рішення і витрачено час на вивчення технології ASP.NET Core Identity, однак у цій технології не передбачена безпарольна структура і тому довелося розробляти вручну "кастомний логін", який має на увазі введення вашої робочої пошти в поле реєстрації, потім надсилання на цю пошту коду підтвердження який є паролем.

```
[HttpPost("login")]
[AllowAnonymous]
public IActionResult Login([FromBody] UserDTO user)
{
    Console.WriteLine(user.Email);
    Console.WriteLine(DateTime.UtcNow.ToString());
    if (string.IsNullOrWhiteSpace(user.Email))
    {
        ModelState.AddModelError("Email", "Email is required");
        return BadRequest(ModelState);
    }

    var existingUser = dataContext.Users.FirstOrDefault(u => u.Email == user.Email);

    string ConfirmationCode = _randomService.ConfirmCode(8);
    if (existingUser == null)
    {
        var userId = Guid.NewGuid().ToString();

        var userFolderPath = Path.Combine(_userFolderPath, userId);
        Directory.CreateDirectory(userFolderPath);
        var salt = _randomService.RandomString(16);

        existingUser = new User
        { Id = userId, Email = user.Email, Salt = salt, EmailCode = _kdfService.GetDerivedKey(ConfirmationCode, salt) };
        dataContext.Users.Add(existingUser);
    }
    else
    {
        var salt = _randomService.RandomString(16);

        existingUser.EmailCode = _kdfService.GetDerivedKey(ConfirmationCode, salt);
        existingUser.Salt = salt;
    }

    dataContext.SaveChanges();

    SendConfirmationCodeByEmail(existingUser.Email, ConfirmationCode);

    return Ok(new { status = "Data was accepted" });
}
```

У нашій системі реєстрації також мається на увазі відсутність так званої функції забув пароль, оскільки пароль одноразовий і при кожному використанні він новий, то дана функція не має необхідності.

Так само однією з переваг нашої програми з боку розробки є з'єднання створення нового користувача та оновлення коду для заходу вже на існуючого користувача, так як більша частина дій повторювалася б у даних методах, то було прийнято рішення поєднати 2 методи в один через те, що вони не несуть додаткового навантаження на сервер і у разі їхнього поділу проводилися б ті самі операції, так як процедура оновлення даних у існуючого користувача і створення нового дуже схожі.

Метод авторизації передбачає перевірку одержуваних даних на існування поточного користувача, зіставлення отриманого ним коду на той, що

зберігається в базі даних у зашифрованому вигляді, створення JWT токена, збереження його в базі даних і повернення токена на клієнтську частину в разі успіху всіх попередніх операцій.

```
[HttpPost("authorize")]
[AllowAnonymous]
public async Task<IActionResult> Authorize([FromBody] LoginDTO login)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = dataContext.Users.FirstOrDefault(u => u.Email == login.Email);
    if (user == null)
    {
        ModelState.AddModelError("Email", "User with this email does not exist");
        return BadRequest(ModelState);
    }

    if (user.EmailCode != _kdfService.GetDerivedKey(login.EmailCode, user.Salt))
    {
        ModelState.AddModelError("EmailCode", "Incorrect email code");
        return BadRequest(ModelState);
    }

    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(_key);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
    new Claim(ClaimTypes.NameIdentifier, user.Id)
        }),
        Expires = DateTime.UtcNow.AddHours(1),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    var tokenString = tokenHandler.WriteToken(token);
    var checkOldToken = dataContext.AuthTokens.FirstOrDefault(t => t.userId == Guid.Parse(user.Id));
    if (checkOldToken == null)
    {
        checkOldToken = new AuthToken
        {
            Id = Guid.NewGuid(),
            userId = Guid.Parse(user.Id),
            Token = tokenString,
            Used = 0
        };
        dataContext.AuthTokens.Add(checkOldToken);
    }
    else
    {
        checkOldToken.Token = tokenString;
    }
    dataContext.SaveChanges();
    return Ok(new { Token = tokenString });
}
```

Метод SendBoard відповідає за отримання та збереження інформації про дошку. Спочатку він перевіряє, що в даних форми є інформація про дошку (ключ "board"). Потім JSON-рядок з інформацією про дошку перетворюється на об'єкт BoardDTO. Після цього метод шукає користувача в базі даних за його електронною поштою, вказаною в BoardDTO, і перевіряє валідність токена, що міститься в заголовку запиту. Якщо нотатка з NoteId не знайдена, створюється нова нотатка. Інакше оновлюються властивості існуючої нотатки. Дані про дошку зберігаються в JSON-файлі з ім'ям note.Id.json. Якщо разом із даними

передані файли, перший із них зберігається на сервері. Всі зміни зберігаються в базі даних, і метод повертає відповідь з повідомленням про успішне збереження даних або помилку у разі невдачі.

Метод SendList працює аналогічно до методу SendBoard, але обробляє дані списку. Він перевіряє, що у формі є дані списку (ключ "list"), потім перетворює JSON-рядок з інформацією про список на об'єкт ListDTO, шукає користувача по email з ListDTO, перевіряє валідність токена користувача, створює нову замітку або оновлює існуючу, зберігає дані списку у файл note.Id.json, зберігає перший файл із форми даних, якщо він є, та зберігає всі зміни в базі даних. Наприкінці метод повертає повідомлення про успішне збереження чи помилку.

Метод SendGallery відповідає за збереження галереї зображень. Він перевіряє, що об'єкт GalleryDTO не дорівнює null, потім шукає користувача по email з GalleryDTO та перевіряє валідність токена. Якщо нотатка з NoteId не знайдена, створюється нова нотатка. Інакше оновлюються властивості існуючої нотатки. Метод створює потрібні папки для зберігання даних і, якщо у галереї є зображення у форматі base64, зберігає їх як файли. Дані галереї зберігаються у файлі note.Id.json. Усі зміни зберігаються у базі даних, і метод повертає повідомлення про успішне збереження чи помилку.

```
[HttpPost("sendGallery")]
[Authorize]
public async Task<IActionResult> SendGallery([FromForm] GalleryDTO gallery)
{
    try
    {
        if (gallery == null)
        {
            return BadRequest("Отсутствуют данные.");
        }

        var user = dataContext.Users.FirstOrDefault(u => u.Email == gallery.email);
        if (user == null)
        {
            return BadRequest("Пользователь не найден.");
        }
        var authHeader = HttpContext.Request.Headers["Authorization"].FirstOrDefault();
        if (!tokenCheck(user.Id, authHeader))
        {
            return BadRequest("Invalid token");
        }
        var noteId = Guid.Parse(gallery.noteId);
        var note = dataContext.Notes.FirstOrDefault(n => n.Id == noteId);

        if (note == null)
        {
            note = new Note
            {
                Id = noteId,
                userId = user.Id,
                isDeleted = false,
                teamspaceId = Guid.Empty,
                isFavorite = false,
                name = gallery.title,
                iconPath = gallery.iconPath,
                routerLink = gallery.currentLink,
                noteFileName = $"{noteId}.json"
            };

            dataContext.Notes.Add(note);
        }
        else
        {
            note.name = gallery.title;
        }

        var userFolderPath = Path.Combine(AppConfig.UserFolderPath, user.Id.ToString());
        Directory.CreateDirectory(userFolderPath);

        var imagesFolderPath = Path.Combine(userFolderPath, "images");
        Directory.CreateDirectory(imagesFolderPath);

        foreach (var card in gallery.content)
        {
            if (!string.IsNullOrEmpty(card.base64Image))
            {
                var imageBytes = Convert.FromBase64String(card.base64Image);
                var fileSavePath = Path.Combine(imagesFolderPath, $"{card.Id}.png");

                await System.IO.File.WriteAllBytesAsync(fileSavePath, imageBytes);

                card.base64Image = null;
                card.description = $"{card.Id}.png";
            }
        }

        var jsonFilePath = Path.Combine(userFolderPath, note.noteFileName);
        Directory.CreateDirectory(Path.GetDirectoryName(jsonFilePath));
        System.IO.File.WriteAllText(jsonFilePath, JsonConvert.SerializeObject(gallery));

        dataContext.SaveChanges();

        return Ok(new { message = "Галерея успешно сохранена." });
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Произошла ошибка: {ex.Message}");
    }
}
```

Метод SendTable відповідає за збереження даних таблиці. Він перевіряє, що у формі є дані таблиці (ключ "table"), потім перетворює JSON-рядок з інформацією про таблицю на об'єкт TableDTO, шукає користувача по email з TableDTO, перевіряє валідність токена, створює нову замітку або оновлює існуючу, зберігає дані таблиці у файл note.Id.json зберігає перший файл з форми даних, якщо він є, і зберігає всі зміни в базі даних. Наприкінці метод повертає повідомлення про успішне збереження чи помилку.

Метод SendPage відповідає за збереження даних сторінки. Він перевіряє, що у формі є дані сторінки (ключ "page"), потім перетворює JSON-рядок з інформацією про сторінку на об'єкт PageDTO, шукає користувача по email з PageDTO, перевіряє валідність токена, створює нову замітку або оновлює існуючу, зберігає дані сторінки у файл note.Id.json зберігає перший файл з форми даних, якщо він є, і зберігає всі зміни в базі даних. Наприкінці метод повертає повідомлення про успішне збереження чи помилку.

```
[HttpPost("sendPage")]
[Authorize]
public async Task<IActionResult> SendPage([FromForm] IFormCollection formData)
{
    try
    {
        if (!formData.TryGetValue("page", out var pageJson))
        {
            return BadRequest("Отсутствуют данные 'page'.");
        }

        var page = JsonConvert.DeserializeObject<PageDTO>(pageJson);

        var user = dataContext.Users.FirstOrDefault(u => u.Email == page.Email);
        if (user == null)
        {
            return BadRequest("Пользователь не найден.");
        }
        var authHeader = HttpContext.Request.Headers["Authorization"].FirstOrDefault();
        if (!tokenCheck(user.Id, authHeader))
        {
            return BadRequest("Invalid token");
        }
        var noteId = Guid.Parse(page.NoteId);
        var note = dataContext.Notes.FirstOrDefault(n => n.Id == noteId);

        if (note == null)
        {
            note = new Note
            {
                Id = noteId,
                userId = user.Id,
                isDeleted = false,
                teamspaceId = Guid.Empty,
                isFavorite = false,
                name = page.Title,
                iconPath = page.iconPath,
                routerLink = page.currentLink,
            };

            dataContext.Notes.Add(note);
        }
        else
        {
            note.name = page.Title;
        }

        note.noteFileName = $"{note.Id}.json";
        var jsonFilePath = Path.Combine(_userFolderPath, note.userId.ToString(), note.noteFileName);
        Directory.CreateDirectory(Path.GetDirectoryName(jsonFilePath));
        System.IO.File.WriteAllText(jsonFilePath, pageJson.ToString());
        if (formData.Files.Count > 0)
        {
            var file = formData.Files[0];
            var fileExtension = Path.GetExtension(file.FileName);
            var fileSavePath = Path.Combine(_userFolderPath, note.userId.ToString(), $"{note.Id}{fileExtension}");

            using (var stream = new FileStream(fileSavePath, FileMode.Create))
            {
                await file.CopyToAsync(stream);
            }
        }

        dataContext.SaveChanges();

        return Ok(new { message = "Заметка и опциональный файл успешно сохранены." });
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Произошла ошибка: {ex.Message}");
    }
}
```

Метод GetPage призначений для отримання сторінки нотатки. Він аутентифікує користувача по email і токену, шукає замітку NoteId, перевіряє наявність відповідного JSON-файлу на сервері і зчитує його вміст. Поля email, noteId і

currentLink видаляються з об'єкта JSON перед відправкою даних клієнту. У разі помилки повертається відповідний статус та повідомлення.

Метод GetGallery відповідає за отримання галереї зображень. Він перевіряє наявність даних запиту, аутентифікує користувача та перевіряє токен. Потім шукає замітку NoteId і перевіряє наявність JSON-файлу. Зчитується вміст файлу, дані перетворюються на об'єкт GalleryDTO, і для кожного зображення в галереї завантажується його base64-кодування. Результат повертається клієнту. У разі помилок повертається відповідний статус та повідомлення.

```
[HttpPost("getGallery")]
[Authorize]
public async Task<IActionResult> GetGallery([FromBody] GetNoteDTO request)
{
    try
    {
        if (request == null || string.IsNullOrEmpty(request.Email) || string.IsNullOrEmpty(request.NoteId))
        {
            return BadRequest("Отсутствуют данные.");
        }

        var user = dataContext.Users.SingleOrDefault(u => u.Email == request.Email);
        if (user == null)
        {
            return NotFound("Пользователь не найден.");
        }

        var parsedNoteId = Guid.Parse(request.NoteId);
        var note = dataContext.Notes.SingleOrDefault(n => n.Id == parsedNoteId);

        if (note == null)
        {
            return NotFound("Заметка не найдена.");
        }
        var authHeader = HttpContext.Request.Headers["Authorization"].FirstOrDefault();
        if (!tokenCheck(user.Id, authHeader))
        {
            return BadRequest("Invalid token");
        }
        var userFolderPath = Path.Combine(AppConfig.UserFolderPath, user.Id.ToString());
        var jsonFilePath = Path.Combine(userFolderPath, $"{note.Id}.json");

        if (!System.IO.File.Exists(jsonFilePath))
        {
            return NotFound("JSON-файл с данными не найден.");
        }

        using (var streamReader = new StreamReader(jsonFilePath))
        {
            var jsonData = await streamReader.ReadToEndAsync();
            var galleryDto = JsonConvert.DeserializeObject<GalleryDTO>(jsonData);

            var imagesFolderPath = Path.Combine(userFolderPath, "images");
            foreach (var card in galleryDto.content)
            {
                var fileSavePath = Path.Combine(imagesFolderPath, $"{card.Id}.png");
                if (System.IO.File.Exists(fileSavePath))
                {
                    var imageBytes = await System.IO.File.ReadAllBytesAsync(fileSavePath);
                    card.base64Image = Convert.ToBase64String(imageBytes);
                }
            }

            return Ok(galleryDto);
        }
    }
    catch (FormatException ex)
    {
        return BadRequest("Неверный формат ID заметки.");
    }
    catch (FileNotFoundException ex)
    {
        return NotFound("Не удалось найти файл.");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Произошла ошибка: {ex.Message}");
    }
}
```

Метод GetTemplate призначений для отримання шаблону. Він аутентифікує користувача по email та токену, шукає відповідний JSON-файл шаблону на сервері та зчитує його вміст. Поля noteId і currentLink додаються до об'єкта JSON. Якщо нотатка з NoteId не існує, створюється нова нотатка. Потім дані зберігаються у JSON-файл. У разі помилки повертається відповідний статус та повідомлення.

Метод DelPage призначений для позначення нотатки як видаленої. Він аутентифікує користувача по email і токену, шукає замітку по NoteId і встановлює для неї isDeleted прапор. У разі успіху повертається повідомлення про видалення, а у разі помилки – відповідний статус та повідомлення.

```
[HttpPost("delPage")]
[Authorize]
public IActionResult DelPage([FromBody] GetNoteDTO request)
{
    try
    {
        var user = dataContext.Users.FirstOrDefault(u => u.Email == request.Email);
        if (user == null)
        {
            return BadRequest("User not found.");
        }
        var authHeader = HttpContext.Request.Headers["Authorization"].FirstOrDefault();
        if (!tokenCheck(user.Id, authHeader))
        {
            return BadRequest("Invalid token");
        }
        Console.WriteLine(request.NoteId);
        var note = dataContext.Notes.FirstOrDefault(n => n.Id == Guid.Parse(request.NoteId));
        if (note == null)
        {
            return BadRequest("Note not found.");
        }
        note.isDeleted = true;
        dataContext.SaveChanges();

        return Ok("Note was deleted");

    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred: {ex.Message}");
    }
}
```

Метод RecoverPage відновлює раніше видалену нотатку. Він аутентифікує користувача по email та токену, шукає замітку по NoteId та знімає прапорець isDeleted. У разі успіху повертається повідомлення про відновлення, а у разі помилки – відповідний статус та повідомлення.

Метод FullDelPage призначений для повного видалення нотатки. Він аутентифікує користувача по email і токену, шукає замітку з NoteId і повністю видаляє її з бази даних. У разі успіху повертається повідомлення про видалення, а у разі помилки – відповідний статус та повідомлення.

Метод FavPage призначений для додавання замітки до обраних. Він аутентифікує користувача за допомогою email та токена, знаходить замітку за NoteId, після чого позначає її як обрану, встановлюючи властивість isFavorite значення true. У разі успіху повертається повідомлення про успішне додавання до обраних, а у випадку помилки - відповідний статус і повідомлення.

Метод UnFavPage призначений для видалення замітки з обраних. Він аутентифікує користувача за допомогою email та токена, знаходить замітку за NoteId, після чого знімає позначку обраної замітки, встановлюючи властивість isFavorite значення false. У разі успіху повертається повідомлення про успішне видалення з обраних, а у випадку помилки - відповідний статус і повідомлення.

Метод GetUserNotes призначений для отримання всіх заміток користувача. Він перевіряє наявність email у запиті, аутентифікує користувача за допомогою email та токена, знаходить всі не видалені замітки користувача і повертає їх у вигляді списку об'єктів NoteDTO. У разі відсутності заміток повертається повідомлення про їх відсутність, а у випадку помилки - відповідний статус і повідомлення.

Всі методи для роботи з нотатками мають властивість [Authorize] за допомогою якого всі запити, що стосуються нотаток, вимагають відправки токена авторизації в заголовках запиту.

```
[HttpPost("getUserNotes")]
[Authorize]
public IActionResult GetUserNotes([FromBody] GetAllNotes obj)
{
    try
    {
        if (string.IsNullOrWhiteSpace(obj.Email))
        {
            return BadRequest("Email is required.");
        }

        Console.WriteLine("Trying to get notes for email:", obj.Email);

        var user = dataContext.Users.FirstOrDefault(u => u.Email == obj.Email);
        if (user == null)
        {
            return BadRequest("User not found.");
        }
        var authHeader = HttpContext.Request.Headers["Authorization"].FirstOrDefault();
        if (!tokenCheck(user.Id, authHeader))
        {
            return BadRequest("Invalid token");
        }
        var userNotes = dataContext.Notes.Where(n => n.userId == user.Id && n.isDeleted == false).ToList();
        if (userNotes.Count == 0)
        {
            return NotFound("No notes found for this user.");
        }

        var notesDTO = new List<NoteDTO>();
        foreach (var note in userNotes)
        {
            var noteDTO = new NoteDTO
            {
                id = note.Id.ToString(),
                name = note.name,
                isFavorite = note.isFavorite,
                iconPath = note.iconPath,
                currentLink = note.routerLink
            };
            notesDTO.Add(noteDTO);
        }

        return Ok(notesDTO);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred: {ex.Message}");
    }
}
```

Метод GetDeletedUserNotes призначений для отримання всіх видалених заміток користувача. Він перевіряє наявність email у запиті, аутентифікує користувача за допомогою email та токена, знаходить всі видалені замітки користувача і повертає їх у вигляді списку об'єктів NoteDTO. У разі відсутності видалених заміток повертається повідомлення про їх відсутність, а у випадку помилки - відповідний статус і повідомлення.

Метод tokenCheck призначений для перевірки валідності токену авторизації. Він спочатку шукає токен в базі даних за допомогою userId, перетворюючи його в Guid. Якщо токен не знайдено, метод повертає false. Далі, метод

перевіряє заголовок авторизації на наявність та правильний формат. Якщо заголовок порожній або не починається з "Bearer ", повертається false. Якщо заголовок має правильний формат, метод витягує токен з нього, обрізаючи "Bearer " та зайві пробіли. Після цього, витягнутий токен порівнюється з токеном, що зберігається в базі даних. Якщо токени не співпадають, метод повертає false. У випадку успішної перевірки, коли токени співпадають, метод повертає true, підтверджуючи валідність токену авторизації.

Більшість класів представлена у проекті є DTO класами передачі даних між серверної і клієнтської частинами сайту. Також присутні класи сутностей що зберігаються в базі даних, які використовуються для збереження та отримання даних з бази даних за допомогою технологій Entity Framework.

```
public class BoardDTO
{
    [JsonProperty("email")]
    public string Email { get; set; }

    [JsonProperty("noteId")]
    public string NoteId { get; set; }

    [JsonProperty("title")]
    public string Title { get; set; }

    [JsonProperty("lists")]
    public List<ListBoard> Content { get; set; } = new List<ListBoard>();

    [JsonProperty("currentLink")]
    public string currentLink { get; set; }

    [JsonProperty("iconPath")]
    public string iconPath { get; set; }
}
```

Перелік використаних технологій

№	Назва технології
1	Visual Studio Розробник: Microsoft
2	ASP.NET Core Розробник: Microsoft
3	ADO.NET Entity Framework Розробник: Microsoft
4	Kestrel Розробник: Microsoft
5	SHA-256 Розробник: NIST (Національний інститут стандартів і технології)
4	LINQ Розробник: Microsoft
5	Postman Розробник: Ankit Sobti, Abhinav Asthana, and Abhijit Kane
6	Radmin VPN Розробник: Famatech Corp.