МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька академія»

Навчально науковий інститут ІТ та бізнесу

Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістра

на тему: **«Управління проєктом розробки системи автоматизованого оформлення довідок для студентів»**

Виконав: студент 2 курсу, групи МУП-2
другого (магістерського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
ОПП «Управління проєктами»

Радчук Вадим Андрійович

Керівник: ***Місай В.В., викладач, фахівець-практик кафедри ІТБ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики Донецького національного університету імені Василя Стуса*

Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних
_____ (проф., д.е.н. Кривицька О.Р.)
Протокол № 5 від «04» грудня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня магістра

Тема: *Управління проєктом розробки системи автоматизованого оформлення довідок для студентів*

Автор: *Радчук Вадим Андрійович*
Науковий керівник: *Місай В.В., викладач, фахівецьпрактик кафедри ІТБ*

Захищена «……»…… 20____ року.

Пояснювальна записка до кваліфікаційної роботи: *с. 91, 6 рис., 33 джерел.*

Ключові слова: *вебсистема, автоматизація, .NET, React, веброзробка, інформаційна система, управління проєктом, OAuth, Azure.*

Короткий зміст праці:
Завданням кваліфікаційної роботи було дослідження та практичне застосування сучасних підходів до управління проєктами під час створення вебсистеми автоматизованого оформлення довідок для студентів Національного університету «Острозька академія». У роботі виконано аналіз традиційних та гнучких методологій управління, зокрема каскадної моделі, Scrum, Kanban та гібридних підходів. Обґрунтовано вибір гібридної методології, яка поєднує переваги каскадного планування ключових етапів із гнучкістю спринтової роботи за Scrum. Такий підхід забезпечив можливість чіткого визначення віх проєкту та одночасну адаптивність у процесі розробки. Практична реалізація охопила повний цикл створення вебсистеми — від аналізу потреб користувачів, розроблення вимог та проєктування архітектури до побудови робочого прототипу та тестування. Бекенд частину реалізовано з використанням ASP.NET Core Web API та Entity Framework Core, що забезпечило надійну обробку даних, авторизацію та роботу з базою. Фронтенд створено на основі React та TypeScript, що дало можливість розробити зручний, швидкий та модульний інтерфейс для студентів і адміністраторів. У системі реалізовано механізми автентифікації через Google OAuth 2.0, рольову модель доступу, захищені маршрути, валідацію форм та захист від надмірної кількості запитів. Завдяки цьому забезпечено високий рівень безпеки, надійності та зручності користування. Результатом роботи стало створення сучасної вебплатформи, що значно спрощує процес отримання студентських довідок, автоматизує взаємодію зі службами університету та скорочує час опрацювання заявок.

ANNOTATION

of a qualification paper

for a master's degree

Theme: *Project Management for the Development of an Automated Certificate Issuance System for Students*

Author: *Radchuk Vadym*

Scientific supervisor: *Misai V.V., lecturer, specialist practitioner at the ITB department*

Defended «……»……...of 2025.

Explanatory note to the qualification work: *p.91, pic. 6, 33 sources.*

Keywords: *web system, automation, .NET, React, web development, information system, project management, OAuth, Azure.*

Summary of the paper:

The objective of the qualification work was to research and practically apply modern project management approaches during the development of a web-based system for automated issuance of student certificates at the National University of "Ostroh Academy." The study includes an analysis of traditional and agile project management methodologies, including the Waterfall model, Scrum, Kanban, and hybrid approaches. A hybrid methodology was justified as the most appropriate, combining the advantages of Waterfall planning for key project milestones with the flexibility of Scrum sprint-based development. This approach ensured clear definition of project phases while maintaining adaptability throughout the development process. The practical implementation covered the full cycle of creating the web system—from analysing user needs, defining requirements, and designing the software architecture to building a functional prototype and conducting testing. The backend was developed using ASP.NET Core Web API and Entity Framework Core, which provided reliable data processing, user authentication, and database operations. The frontend was implemented with React and TypeScript, enabling the creation of a user-friendly, fast, and modular interface for both students and administrators. The system incorporates Google OAuth 2.0 authentication, a role-based access model, protected routes, form validation, and protection against excessive request rates. These mechanisms ensure a high level of security, reliability, and ease of use. As a result, a modern web platform was created that significantly simplifies the process of obtaining student certificates, automates interaction with university services, and reduces the time required to process requests.

ЗМІСТ

ВСТУП

У сучасних умовах цифрової трансформації закладів вищої освіти зростає потреба в автоматизації внутрішніх адміністративних процесів, зокрема тих, що пов'язані з оформленням студентської документації. Традиційна офлайн-процедура отримання довідок вимагає додаткового часу від студентів та створює значне навантаження на працівників деканатів. Студенти змушені особисто звертатися до деканату, часто стояти в чергах, заповнювати паперові бланки, а потім повертатися за готовими документами через кілька днів. Це зумовлює потребу у впровадженні сучасних вебсистем, які забезпечують прозорість, оперативність та зручність взаємодії між студентами й адміністративним персоналом.

Пандемія COVID-19 додатково актуалізувала питання дистанційного надання адміністративних послуг в університетах. Заклади вищої освіти по всьому світу були змушені швидко адаптуватися до нових умов, впроваджуючи цифрові рішення для підтримки освітнього процесу та супутніх адміністративних функцій. Національний університет «Острозька академія», як один із провідних освітніх закладів України, прагне відповідати сучасним вимогам цифровізації та забезпечувати високий рівень сервісу для своїх студентів.

Розробка вебсистеми автоматизованого оформлення довідок для студентів Національного університету «Острозька академія» є актуальним завданням, що поєднує технічні аспекти програмної інженерії, сучасні підходи до управління проєктами та вимоги освітнього середовища. Створення такого програмного продукту потребує застосування ефективної методології управління, детального аналізу вимог, проєктування архітектури, розроблення зручного користувацького інтерфейсу та впровадження сучасних інструментів автентифікації й захисту даних.

Аналіз міжнародного досвіду показує, що провідні університети світу вже давно використовують автоматизовані системи для надання студентських сервісів. Водночас в Україні такі рішення впроваджуються повільніше, що

створює можливості для інновацій та адаптації світових практик до специфіки вітчизняної освітньої системи. Особливої уваги потребує питання інтеграції нових систем з існуючою університетською інфраструктурою та забезпечення відповідності вимогам законодавства України щодо захисту персональних даних.

Метою кваліфікаційної роботи є управління проєктом розробки вебсистеми автоматизованого оформлення довідок для студентів НаУОА та аналіз ефективності застосування гібридної методології управління в поєднанні з сучасним технологічним стеком (.NET, React).

Для досягнення поставленої мети необхідно було вирішити такі завдання:

1. Проаналізувати сучасні теорії та методології управління проєктами, зокрема каскадну, Agile (Scrum, Kanban) та гібридні моделі, визначити їхні переваги та недоліки в контексті освітніх IT-проєктів.
2. Дослідити особливості IT-проєктів, пов'язаних із розробкою вебсистем, включно з вимогами до безпеки, масштабованості та зручності користування, з урахуванням специфіки закладів вищої освіти.
3. Провести аналіз існуючих аналогів систем автоматизації студентських сервісів, виявити їхні сильні та слабкі сторони, визначити функціональні та нефункціональні вимоги до системи автоматизованого оформлення довідок.
4. Розробити концептуальну та технічну архітектуру вебсистеми із застосуванням ASP.NET Core Web API для серверної частини та React для клієнтської частини, визначити сучасні методи інтеграції клієнт–сервер.
5. Реалізувати серверну частину системи із підтримкою багаторівневої авторизації, управління ролями користувачів, ефективної роботи з реляційною базою даних та автоматизованої обробки заявок на довідки.
6. Розробити клієнтську частину системи з інтуїтивним, адаптивним інтерфейсом для різних категорій користувачів (студенти, працівники деканату, адміністратори), що забезпечує зручність використання на різних пристроях.

7. Забезпечити комплексний підхід до безпеки системи: реалізувати валідацію даних на клієнті та сервері, впровадити захист форм за допомогою Google reCAPTCHA, налаштувати механізми rate limiting для запобігання зловживанням, імплементувати систему логування дій користувачів.

8. Провести багаторівневе тестування системи (модульне, інтеграційне, користувацьке), здійснити налагодження виявлених помилок та оцінити ефективність обраної методології управління проєктом на основі аналізу планових і фактичних показників.

9. Розробити рекомендації щодо впровадження системи в університеті та визначити напрями її подальшого розвитку і масштабування.

Об'єкт, предмет та методи дослідження

Об'єктом дослідження є процес управління IT-проєктами зі створення вебсистем для автоматизації адміністративних процесів у закладах вищої освіти, з особливим акцентом на специфіку розробки систем студентського самообслуговування.

Предметом дослідження є гібридні методології управління проєктами (поєднання каскадного планування та Scrum-спринтів), а також сучасні технології розробки вебзастосунків, зокрема ASP.NET Core для backend-розробки, React і TypeScript для frontend-розробки, Google OAuth 2.0 для автентифікації користувачів, Entity Framework Core для роботи з базами даних, та засоби забезпечення безпеки й ефективної клієнт-серверної взаємодії, застосовані під час розробки системи автоматизованого оформлення довідок.
Для досягнення мети використано такі **методи дослідження**:
- теоретичний аналіз і синтез — для всебічного вивчення методологій управління проєктами (Waterfall, Agile, гібридні моделі) та особливостей сучасної веброзробки, систематизації знань про архітектурні патерни та практики програмної інженерії;
- порівняльний аналіз — для об'єктивного вибору оптимальних інструментів і технологій розробки на основі встановлених критеріїв (продуктивність,

безпека, підтримка спільноти, масштабованість), дослідження переваг і недоліків альтернативних рішень;

- методи проєктного менеджменту — декомпозиція роботи (WBS), календарне планування, ідентифікація та аналіз ризиків, організація роботи в Scrum-спринтах із регулярними ретроспективами, контроль виконання завдань та управління змінами;

- емпіричні методи — практичне тестування функціоналу системи (модульне, інтеграційне, end-to-end), перевірка механізмів захисту даних та авторизації, аналіз продуктивності системи під навантаженням, збір зворотного зв'язку від потенційних користувачів;

- статистичний аналіз — для кількісної оцінки виконання планових і фактичних показників проєкту, аналізу відхилень від графіка, визначення ефективності використання ресурсів, побудови метрик якості коду та покриття тестами.

Наукова новизна дослідження

Наукова новизна полягає в обґрунтуванні та практичній апробації ефективного підходу до управління освітнім IT-проєктом обмеженого масштабу, який базується на гібридній моделі управління та використанні сучасного технологічного стеку.

Основні елементи наукової новизни:

1. Обґрунтовано доцільність застосування гібридної методології управління, що поєднує структурованість каскадного планування на рівні проєкту з гнучкістю Scrum на рівні розробки, для освітніх IT-проєктів з обмеженими ресурсами та фіксованими дедлайнами.
2. Розроблено архітектурний підхід до побудови вебсистеми студентського самообслуговування, який інтегрує модульну REST API архітектуру, реактивну клієнтську частину на базі React, OAuth 2.0 авторизацію через корпоративні акаунти та багаторівневу систему захисту даних.

3. Продемонстровано можливість створення масштабованого рішення для автоматизації внутрішніх університетських процесів із використанням відкритих технологій та хмарних сервісів, що може бути адаптоване для інших закладів вищої освіти України.
4. Систематизовано практичні рекомендації щодо управління невеликими IT-командами в освітньому середовищі з урахуванням специфіки роботи над реальним продуктом, що має бути впроваджений в університетську інфраструктуру.

Практичне та теоретичне значення

Практичне значення роботи полягає у створенні готової до впровадження вебсистеми, яка здатна суттєво оптимізувати роботу деканату НаУОА, скоротити час обробки студентських запитів з кількох днів до кількох годин, зменшити паперовий документообіг, мінімізувати ймовірність помилок при оформленні довідок та забезпечити більш ефективну, прозору комунікацію між студентами й адміністративними службами.

Розроблений програмний продукт може бути використаний університетом як повноцінний сервіс, інтегрований в екосистему університетських інформаційних систем, що підвищує якість обслуговування студентів та рівень цифровізації освітнього процесу. Система дозволяє зберігати історію всіх заявок, генерувати статистичні звіти, відстежувати навантаження на деканат та приймати обґрунтовані управлінські рішення щодо оптимізації процесів.

Крім того, архітектурні рішення та програмний код можуть бути використані як основа для розробки аналогічних систем в інших українських університетах або для розширення функціоналу системи на інші адміністративні процеси (замовлення академічних довідок, обхідних листків, електронних студентських квитків тощо).

Теоретичне значення полягає в адаптації гібридних підходів управління проєктами до специфіки веброзробки в освітньому контексті, а також у систематизації знань щодо побудови архітектури сучасних вебсистем на основі

.NET і React з урахуванням вимог безпеки, продуктивності та зручності використання.

Робота доводить ефективність поєднання гнучких практик Scrum (короткі ітерації, адаптивне планування, регулярний зворотний зв'язок) із жорсткішим каскадним плануванням (чітка структура фаз, фіксовані контрольні точки, формальна документація) у межах освітніх IT-проєктів обмежених ресурсів, де команда складається з одного-двох розробників, а термін виконання обмежений академічним календарем.

Дослідження вносить вклад у розуміння особливостей управління проєктами цифрової трансформації в українських університетах, демонструє можливості використання сучасних технологій та методологій для вирішення конкретних практичних завдань освітнього менеджменту.

Гіпотеза дослідження

Гіпотеза дослідження полягає в тому, що використання гібридної методології управління (яка об'єднує спринтову гнучкість Scrum та структурованість каскадного планування) у поєднанні з сучасними вебтехнологіями (.NET, React) забезпечує оптимальний баланс між швидкістю розробки, якістю реалізації та гнучкістю до змін у вимогах при створенні вебсистем для автоматизації університетських процесів.

Передбачається, що така комбінація методологічних і технологічних підходів дозволить:

- зменшити ризики невиконання проєкту в строк завдяки чіткому каскадному плануванню основних фаз;
- швидко реагувати на зміни вимог та пріоритетів завдяки спринтовій організації розробки;
- забезпечити високу якість програмного продукту через регулярне тестування та ретроспективи;
- ефективно використовувати обмежені ресурси малої команди розробки;
- створити масштабовану, безпечну та зручну систему, яка відповідає сучасним стандартам веброзробки.

Структура та обсяг роботи

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить ___ сторінок, включає ___ рисунків, ___ таблиць. Список використаних джерел налічує ___ найменувань.

У першому розділі «Теоретико-методологічні засади управління ІТ-проєктами та створення вебсистем» розглянуто фундаментальні концепції проєктного менеджменту, проаналізовано класичні та гнучкі методології управління (Waterfall, Agile, Scrum, Kanban), досліджено специфіку гібридних підходів. Окрему увагу приділено особливостям ІТ-проєктів у сфері освіти, вимогам до вебсистем студентського самообслуговування, аналізу існуючих рішень та визначенню ключових факторів успіху проєктів цифровізації університетів.

У другому розділі «Практична реалізація проєкту розробки вебсистеми автоматизованого оформлення довідок» детально описано процес створення системи: збір та аналіз вимог стейкхолдерів, розробку концептуальної та технічної архітектури, вибір та обґрунтування технологічного стеку, імплементацію серверної частини (ASP.NET Core Web API, Entity Framework Core, PostgreSQL/SQL Server), створення клієнтської частини (React, TypeScript, Material-UI/Ant Design), інтеграцію Google OAuth 2.0, впровадження механізмів безпеки (валідація, reCAPTCHA, rate limiting, CORS), організацію роботи в спринтах та управління змінами.

У третьому розділі «Оцінка ефективності управління проєктом та результатів розробки» представлено аналіз виконання проєктних показників, порівняння планових і фактичних термінів, оцінку якості використання гібридної методології, результати тестування системи (функціональне, безпеки, продуктивності, юзабіліті), метрики якості коду, висновки щодо успішності реалізації проєкту та рекомендації щодо впровадження системи в університеті й подальшого її розвитку.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Формулювання проблематики та актуальності розробки

У сучасних умовах розвитку цифрових технологій перед закладами вищої освіти постає завдання створення ефективних, доступних та прозорих інформаційних систем, здатних забезпечити високий рівень автоматизації внутрішніх процесів. Особливе значення має цифровізація адміністративних процедур, оскільки саме вони найбільше впливають на якість комунікації між студентами та університетом, а також визначають загальний рівень ефективності організаційної структури навчального закладу. Цифрова трансформація освітніх процесів стала не просто тенденцією, а необхідністю, що диктується вимогами сучасного суспільства, очікуваннями студентів, які виросли в епоху цифрових технологій, та потребою університетів у підвищенні операційної ефективності.

Традиційний процес оформлення студентських довідок передбачає особисте відвідування деканату, заповнення паперових форм, очікування у чергах та тривалу обробку запитів працівниками адміністрації. Цей процес супроводжується значними часовими витратами, високою ймовірністю виникнення помилок, дублюванням даних, перевантаженням персоналу та недостатньою прозорістю для студентів. Студент, якому необхідно отримати довідку про навчання, змушений витрачати час на дорогу до університету, очікування прийому працівника деканату, заповнення бланків вручну, після чого знову повертатися через декілька днів за готовим документом. Якщо при цьому виникають помилки у заповненні або відсутні необхідні дані, процес повторюється спочатку, що призводить до фрустрації з боку студентів та додаткового навантаження на адміністративний персонал.

У пікові періоди навчального року, зокрема перед сесіями, під час оформлення документів для поселення у гуртожиток, подання заяв на стипендію або соціальні виплати, при оформленні академічної мобільності чи влаштуванні

на роботу, навантаження на деканати зростає в кілька разів. Це призводить до утворення черг, збільшення термінів обробки заявок від стандартних трьох-п'яти днів до тижня і більше, а також створює додаткові труднощі як для студентів, так і для персоналу. Працівники деканатів змушені відволікатися від інших важливих обов'язків, що негативно впливає на загальну якість адміністративної роботи університету. Крім того, відсутність системи пріоритизації заявок означає, що термінові запити можуть обробляться так само довго, як і звичайні, що особливо критично у випадках, коли довідка потрібна для подання документів у стислі терміни.

Проблематика традиційного підходу також пов'язана з відсутністю можливості відстеження статусу заявки. Студент не має інформації про те, чи прийнята його заява, на якому етапі опрацювання вона знаходиться, коли приблизно буде готова довідка. Це призводить до необхідності повторних дзвінків або візитів до деканату, що ще більше збільшує навантаження на персонал. Відсутність єдиної бази даних заявок також ускладнює аналіз навантаження, планування роботи та виявлення вузьких місць у процесі обслуговування студентів.

Ще однією важливою проблемою є паперовий документообіг, який не відповідає сучасним екологічним та організаційним стандартам. Зберігання великої кількості паперових документів вимагає фізичного простору, створює труднощі у пошуку архівної інформації, а також несе ризики втрати або пошкодження документів. У разі необхідності отримання статистичної інформації про кількість виданих довідок за певний період, типи найбільш затребуваних документів чи навантаження на різні факультети, працівникам доводиться вручну аналізувати паперові архіви, що є надзвичайно трудомістким процесом.

Університети, що прагнуть модернізувати власну інфраструктуру, дедалі більше схиляються до впровадження інтегрованих вебплатформ, здатних забезпечити єдиний інформаційний простір та автоматизувати документообіг. Такі системи не лише спрощують взаємодію користувачів, але й дають змогу

формувати аналітичні звіти, відстежувати динаміку звернень, виявляти закономірності у поведінці студентів та покращувати стратегічне планування роботи адміністративних підрозділів. Автоматизовані системи дозволяють виконувати багато операцій в автоматичному режимі, мінімізуючи ручне введення даних та пов'язані з цим помилки. Наприклад, інформація про студента може автоматично підтягуватися з університетської бази даних, довідки можуть генеруватися за шаблонами з автоматичним заповненням необхідних полів, а статуси заявок можуть оновлюватися автоматично на кожному етапі обробки.

Впровадження автоматизованої системи є не просто технічним удосконаленням, а важливим елементом цифрової трансформації університету, що впливає на всі аспекти його діяльності. Така система підвищує привабливість університету для абітурієнтів та студентів, які очікують сучасного рівня сервісу, відповідає міжнародним стандартам якості освітніх послуг, покращує імідж закладу як технологічно прогресивного та орієнтованого на потреби студентів. Крім того, досвід впровадження подібних систем у провідних університетах світу демонструє значне зменшення операційних витрат, підвищення задоволеності студентів якістю обслуговування та звільнення часу адміністративного персоналу для виконання більш складних та відповідальних завдань.

Проблематика створення єдиної платформи для оформлення студентських довідок полягає у необхідності забезпечення високої продуктивності системи, здатної обробляти велику кількість одночасних запитів без зниження швидкості відгуку. Важливим аспектом є захист персональних даних студентів відповідно до вимог законодавства України, зокрема Закону України "Про захист персональних даних" та Загального регламенту захисту даних Європейського Союзу. Система повинна забезпечувати шифрування даних при передачі та зберіганні, контроль доступу на основі ролей, журналювання всіх операцій з персональними даними та можливість відстеження історії змін.

Адаптивність інтерфейсу є ще одним критичним аспектом, оскільки студенти очікують можливості користуватися системою з різних пристроїв -

смартфонів, планшетів, ноутбуків та настільних комп'ютерів. Інтерфейс повинен автоматично підлаштовуватися під розмір екрану, забезпечуючи однаково зручний досвід користування незалежно від пристрою. Це особливо важливо, враховуючи, що більшість студентів активно використовують мобільні пристрої для доступу до онлайн-сервісів.

Можливість інтеграції з іншими внутрішніми системами університету, такими як електронний деканат, система управління навчальним процесом, бібліотечна система, система обліку студентів у гуртожитках, є необхідною умовою для забезпечення цілісності даних та уникнення їх дублювання. Інтеграція дозволяє автоматично отримувати актуальну інформацію про студента, його академічні досягнення, статус навчання, що значно прискорює процес формування довідок та зменшує ймовірність помилок.

Гнучкість у подальшому масштабуванні означає, що система повинна бути побудована таким чином, щоб легко розширюватися функціонально без необхідності повного переписування коду. Це передбачає модульну архітектуру, використання сучасних патернів проєктування програмного забезпечення, застосування мікросервісної архітектури або принаймні чітке розділення відповідальності між компонентами системи. У майбутньому може виникнути потреба у додаванні нових типів довідок, інтеграції з державними реєстрами, впровадженні електронного цифрового підпису, створенні мобільного додатку або розширенні функціоналу на інші адміністративні процеси університету.

Саме тому актуальним є розроблення вебсистеми, яка здатна вирішити всі перелічені проблеми та відповідатиме сучасним стандартам цифрової освіти. Така система повинна забезпечувати швидкий та зручний доступ студентів до сервісів оформлення довідок в будь-який час доби та з будь-якого місця, де є доступ до інтернету. Працівники деканату повинні отримати зручний інструмент для централізованого управління заявками, можливість пріоритизації термінових запитів, автоматичного формування документів за шаблонами та ведення статистики. Адміністрація університету має мати доступ до аналітичних інструментів для моніторингу ефективності процесів, виявлення проблемних зон

та прийняття обґрунтованих управлінських рішень щодо оптимізації роботи деканатів.

Розробка такої системи вимагає комплексного підходу, що включає не тільки технічну реалізацію, але й ретельне проєктування користувацького досвіду, забезпечення безпеки, планування інфраструктури та стратегії впровадження. Важливим є також навчання персоналу роботі з новою системою, проведення інформаційної кампанії серед студентів, поступовий перехід від паперового документообігу до цифрового з можливістю паралельної роботи обох систем на перехідному етапі. Успішне впровадження системи автоматизованого оформлення довідок стане важливим кроком у цифровій трансформації Національного університету "Острозька академія" та може слугувати прикладом для інших закладів вищої освіти України.

1.2. Основні користувачі платформи та типові сценарії її застосування

Вебплатформа автоматизованого оформлення студентських довідок призначена для широкого кола користувачів, діяльність яких пов’язана з отриманням, опрацюванням та адмініструванням заявок на довідки. Для забезпечення ефективності та зручності роботи системи було визначено основні категорії користувачів, кожна з яких має власні цілі, потреби та сценарії взаємодії з платформою.

Студенти – основні кінцеві користувачі системи, для яких призначений функціонал створення та відстеження заявок на отримання довідок. Студенти мають змогу швидко оформити запит, обрати необхідний тип документа, переглянути статус обробки та отримати повідомлення про готовність довідки.

Працівники деканату відповідальні за опрацювання заявок. Вони здійснюють перевірку даних, формують довідки, оновлюють статуси та взаємодіють зі студентами в разі потреби уточнень. Ця група має доступ до розширеного функціоналу управління потоком заявок.

Адміністратор системи – технічний користувач який відповідає за підтримку працездатності платформи, керування ролями, контроль захищених маршрутів, налаштування доступів та моніторинг системних логів. Він забезпечує стабільність роботи та відповідність функціоналу вимогам університету.

1.3. Типові сценарії використання систем

Веб-платформа автоматизованого оформлення студентських довідок призначена для широкого кола користувачів, діяльність яких пов'язана з отриманням, опрацюванням та адмініструванням заявок на довідки. Для забезпечення ефективності та зручності роботи системи було визначено основні категорії користувачів, кожна з яких має власні цілі, потреби та сценарії взаємодії з платформою. Розуміння специфіки роботи кожної групи користувачів є критично важливим для проєктування інтерфейсу, визначення функціональних можливостей та розподілу прав доступу. Саме орієнтація на реальні потреби користувачів дозволяє створити систему, яка буде не просто технічно досконалою, але й дійсно корисною та зручною у повсякденному використанні.

Студенти є основними кінцевими користувачами системи, для яких призначений функціонал створення та відстеження заявок на отримання довідок. Це найчисельніша група користувачів платформи, яка включає студентів всіх курсів, факультетів та форм навчання Національного університету "Острозька академія". Студенти звертаються до системи з різною частотою в залежності від їхніх потреб: деякі можуть замовляти довідки раз на семестр, інші - кілька разів на місяць у зв'язку з оформленням стипендій, соціальних виплат, участю у конкурсах чи програмах обміну. Кожен студент має унікальний профіль у системі, який створюється автоматично при першій авторизації через корпоративний Google-акаунт університету.

Типовий сценарій роботи студента з платформою починається з входу до системи через механізм Google OAuth 2.0, що забезпечує безпечну

автентифікацію без необхідності запам'ятовувати додаткові паролі. Після успішної авторизації студент потрапляє на головну сторінку свого особистого кабінету, де відображається актуальна інформація про його поточні заявки, їх статуси та історія попередніх звернень. Інтерфейс студентського кабінету спроєктовано таким чином, щоб мінімізувати кількість кроків, необхідних для створення нової заявки - весь процес може бути завершений буквально за декілька кліків мишкою.

Створюючи нову заявку, студент має змогу обрати необхідний тип довідки зі спеціально структурованого переліку, який включає найпоширеніші документи: довідку про навчання з місцем навчання, довідку про навчання без місця навчання, довідку-виклик, довідку для militar комісаріату, довідку для оформлення соціальних виплат та інші типи документів, що можуть знадобитися студенту під час навчання. Для кожного типу довідки система надає коротку інформаційну довідку про призначення документа, необхідні для його оформлення дані та орієнтовний термін виготовлення. Така прозорість допомагає студентам краще розуміти процес та обирати саме той тип довідки, який відповідає їхнім потребам.

Після вибору типу довідки студент заповнює спеціальну форму, де вказує додаткову інформацію, необхідну для формування документа. Система автоматично підтягує базові дані студента з університетської бази, такі як прізвище, ім'я, по батькові, факультет, спеціальність, курс та форма навчання, що мінімізує ручне введення інформації та зменшує ймовірність помилок. Студенту залишається лише вказати специфічні деталі, наприклад, мету отримання довідки, бажану дату готовності документа або додаткові коментарі для працівників деканату. Форма включає вбудовані механізми валідації, які перевіряють коректність введених даних ще до відправки заявки на сервер, що допомагає уникнути помилок та прискорює процес обробки.

Після відправки заявки студент негайно отримує підтвердження про успішне створення запиту, і заявці автоматично присвоюється унікальний ідентифікатор, за яким можна відстежувати її статус. Система підтримує

декілька станів заявки: "Нова" означає, що заявка щойно створена та очікує на розгляд працівником деканату; "В обробці" вказує на те, що заявку взято в роботу і працівник деканату формує документ; "Готова" означає, що довідка повністю підготовлена і студент може її отримати; "Відхилена" використовується у випадках, коли заявку не може бути виконано з певних причин, які обов'язково вказуються у коментарях. Студент може в будь-який момент зайти до свого кабінету і побачити актуальний статус кожної зі своїх заявок.

Система сповіщень є важливою складовою користувацького досвіду для студентів, оскільки вона забезпечує інформування про всі важливі зміни статусу заявки без необхідності постійно перевіряти систему вручну. Коли статус заявки змінюється, наприклад, переходить зі стану "В обробці" у стан "Готова", студент автоматично отримує сповіщення, яке відображається в інтерфейсі при наступному вході до системи. У перспективі планується розширення системи сповіщень шляхом інтеграції з електронною поштою або SMS-повідомленнями, що дозволить студентам отримувати інформацію навіть коли вони не перебувають активно в системі.

Працівники деканату становлять другу ключову групу користувачів системи, відповідальних за опрацювання заявок студентів. Ця категорія включає співробітників деканатів різних факультетів, методистів, секретарів та інших працівників, які мають повноваження формувати та видавати студентські довідки. Кожен працівник деканату має власний обліковий запис у системі з чітко визначеними правами доступу, які зазвичай обмежені заявками студентів конкретного факультету. Така сегментація забезпечує конфіденційність даних та відповідає організаційній структурі університету.

Робочий інтерфейс працівника деканату суттєво відрізняється від студентського кабінету і орієнтований на ефективне управління великою кількістю заявок. Центральним елементом є таблиця заявок, яка підтримує гнучку систему фільтрації та сортування, що дозволяє швидко знаходити потрібні запити серед сотень інших. Працівник може фільтрувати заявки за

статусом, щоб побачити лише нові заявки, що потребують уваги, або лише ті, що перебувають в обробці. Фільтрація за датою подання допомагає ідентифікувати заявки, які очікують на обробку довше за встановлені нормативи, а пошук за прізвищем студента дозволяє швидко знайти конкретний запит при особистому зверненні студента.

Опрацьовуючи заявку, працівник деканату має доступ до повної інформації про студента, включаючи його академічні дані, історію попередніх звернень та будь-які додаткові коментарі, залишені студентом при створенні заявки. Система дозволяє працівнику змінювати статус заявки на відповідний етап обробки, додавати внутрішні коментарі для колег, які також працюють з цим запитом, та формувати документ на основі попередньо створених шаблонів. Шаблони довідок є стандартизованими формами, які автоматично заповнюються даними студента та специфічною інформацією із заявки, що значно прискорює процес створення документів та гарантує їх уніфікований вигляд відповідно до вимог університету.

Важливою функцією для працівників деканату є можливість комунікації зі студентами через систему. Якщо для завершення обробки заявки потрібна додаткова інформація або виникають питання щодо деталей запиту, працівник може залишити коментар або запит до студента безпосередньо в системі. Студент отримує сповіщення про необхідність надати додаткові дані і може відповісти через інтерфейс своєї заявки, що створює прозорий та документований канал комунікації. Така можливість значно зменшує потребу в телефонних дзвінках або особистих зустрічах для вирішення простих питань, економлячи час як студентів, так і працівників деканату.

Система також надає працівникам деканату інструменти аналітики та звітності, які допомагають оцінювати ефективність роботи та плані навантаження. Працівник може переглянути статистику за певний період: скільки заявок було опрацьовано, який середній час обробки, які типи довідок найбільш затребувані, в які періоди спостерігаються піки звернень. Ця інформація корисна не лише для планування власної роботи, але й для передачі

керівництву деканату чи адміністрації університету для прийняття управлінських рішень щодо оптимізації процесів або перерозподілу ресурсів.

Адміністратор системи є технічним користувачем, який відповідає за підтримку працездатності платформи, керування ролями, контроль захищених маршрутів, налаштування доступів та моніторинг системних логів. На відміну від студентів та працівників деканату, адміністратор не займається безпосередньо обробкою заявок на довідки, натомість його роль полягає у забезпеченні безперебійної роботи всієї інфраструктури, управлінні користувачами та їхніми правами доступу, налаштуванні системних параметрів та реагуванні на технічні проблеми. Адміністратор має найвищий рівень доступу до системи, що дозволяє йому виконувати критичні операції, які недоступні іншим категоріям користувачів.

Одним із ключових завдань адміністратора є управління обліковими записами користувачів та їхніми ролями. Коли новий співробітник приєднується до деканату або змінюється відповідальна особа, адміністратор створює або модифікує обліковий запис, призначаючи відповідну роль та права доступу. Система підтримує гнучку рольову модель, що дозволяє створювати різні типи користувачів з різними наборами дозволів: від обмеженого доступу лише для перегляду заявок до повного доступу з можливістю створення, редагування та видалення будь-яких записів. Адміністратор також може тимчасово блокувати облікові записи, наприклад, коли співробітник йде у відпустку або залишає посаду, що запобігає несанкціонованому доступу до системи.

Моніторинг роботи системи є постійним завданням адміністратора, який використовує спеціалізовані інструменти для відстеження продуктивності, виявлення помилок та аналізу використання ресурсів. Система логування записує всі важливі події: входи користувачів до системи, створення та зміну заявок, зміну статусів, помилки при обробці запитів, спроби несанкціонованого доступу до захищених ресурсів. Адміністратор регулярно аналізує ці логи для виявлення аномалій, потенційних загроз безпеці або проблем з продуктивністю.

Наприклад, якщо система фіксує багато невдалих спроб входу з певної IP-адреси, адміністратор може вжити заходів для блокування підозрілої активності.

Налаштування системних параметрів також входить до обов'язків адміністратора. Він може змінювати конфігураційні налаштування, такі як тайм-аути сесій, параметри підключення до бази даних, ліміти частоти запитів, налаштування інтеграції з зовнішніми сервісами та багато іншого. Адміністратор відповідає за регулярне резервне копіювання даних, що гарантує можливість відновлення системи у разі виникнення критичних збоїв або втрати даних. Він також планує та проводить оновлення системи, встановлює патчі безпеки, тестує нові функції перед їх розгортанням у продуктивному середовищі та забезпечує сумісність з іншими університетськими інформаційними системами.

Робота адміністратора вимагає глибоких технічних знань у галузі веброзробки, баз даних, мережевих технологій та інформаційної безпеки. Він повинен розуміти архітектуру системи, принципи її роботи, потенційні вразливості та методи їх усунення. Адміністратор також є ключовою особою при впровадженні нових функцій або масштабуванні системи для обслуговування зростаючої кількості користувачів. Він тісно співпрацює з розробниками при вирішенні складних технічних проблем, надає рекомендації щодо покращення архітектури та оптимізації продуктивності системи.

Взаємодія між різними категоріями користувачів формує цілісну екосистему, в якій кожна група виконує свою роль для забезпечення ефективної роботи всієї платформи. Студенти створюють заявки, які формують вхідний потік робіт для деканатів. Працівники деканату обробляють ці заявки, забезпечуючи своєчасне надання довідок та підтримуючи якість обслуговування. Адміністратор забезпечує технічну основу, на якій функціонують всі інші процеси, гарантуючи безпеку, стабільність та продуктивність системи. Така багаторівнева структура дозволяє розподілити відповідальність та забезпечити ефективне виконання завдань на кожному рівні, створюючи гармонійну та продуктивну систему автоматизованого оформлення студентських довідок.

1.4. Вимоги до програмного продукту

Вимоги до вебсистеми автоматизованого оформлення довідок визначають функціональні можливості платформи, її продуктивність, надійність, якість та обмеження. Формування вимог є ключовим етапом, що забезпечує відповідність програмного продукту потребам користувачів і технічним умовам університету. Процес збору та систематизації вимог проводився у тісній співпраці з усіма зацікавленими сторонами: студентами, працівниками деканатів, адміністрацією університету та технічними фахівцями. Така комплексна робота дозволила сформувати збалансований набір вимог, які враховують як функціональні потреби користувачів, так і технічні обмеження інфраструктури університету. У межах цього підрозділу наведено систематизований перелік функціональних та нефункціональних вимог, які були враховані під час проєктування і розробки платформи.

Вимоги до системи формувалися на основі аналізу поточних процесів оформлення довідок, опитування потенційних користувачів, вивчення аналогічних рішень у інших університетах та врахування кращих практик розробки вебзастосунків. Особлива увага приділялася пріоритизації вимог за критеріями важливості та терміновості, що дозволило визначити мінімально життєздатний продукт для першого релізу та запланувати функціональність для наступних ітерацій розробки. Всі вимоги були задокументовані у структурованому вигляді з чіткими критеріями їх виконання, що забезпечило можливість об'єктивної перевірки відповідності реалізованої системи заявленим характеристикам.

1.4.1. Перелік функціональних вимог

Функціональні вимоги визначають конкретний спектр можливостей, які повинна реалізовувати вебплатформа, забезпечуючи повноцінне та стабільне виконання користувацьких сценаріїв. Ці вимоги описують, що саме система має

вміти робити, які операції підтримувати та які результати надавати користувачам у відповідь на їхні дії. Функціональні вимоги є основою для проєктування архітектури системи, розробки інтерфейсів користувача та написання програмного коду, що реалізує необхідну бізнес-логіку.

Система повинна підтримувати надійний та безпечний механізм авторизації та автентифікації, що передбачає можливість входу користувача через Google OAuth 2.0 з автоматичним створенням персонального профілю під час першого звернення до платформи. Вибір Google OAuth 2.0 як основного методу автентифікації обумовлений тим, що всі студенти та працівники Національного університету "Острозька академія" вже мають корпоративні Google-акаунти, які використовуються для доступу до електронної пошти та інших університетських сервісів. Це усуває необхідність запам'ятовування додаткових паролів, спрощує процес входу до системи та підвищує безпеку, оскільки автентифікація відбувається через перевірені механізми Google з підтримкою двофакторної автентифікації.

Після успішного входу система повинна коректно ідентифікувати роль користувача, визначаючи на основі даних з бази даних або попередньо налаштованих правил, чи є він студентом, працівником деканату конкретного факультету або адміністратором системи, і на підставі цього надавати доступ лише до тих функцій, які відповідають його рівню повноважень у рамках ролевої моделі. Ролева модель доступу повинна бути реалізована таким чином, щоб користувач не міг отримати доступ до функцій або даних, які не відповідають його ролі, навіть якщо він спробує обійти клієнтські перевірки через пряме звернення до API endpoints. Серверна частина системи має здійснювати перевірку прав доступу на кожному запиті, гарантуючи, що авторизація контролюється не лише на рівні інтерфейсу, але й на рівні бізнес-логіки.

Функціональність управління студентськими заявками має забезпечувати можливість створення нових електронних заявок на отримання довідок різних типів, що доступні у динамічному переліку, який може оновлюватися адміністраторами системи відповідно до актуальних потреб університету.

Студент повинен мати змогу легко орієнтуватися серед типів довідок, кожен з яких супроводжується коротким описом призначення документа та переліком інформації, необхідної для його оформлення. Це допомагає студентам приймати обґрунтовані рішення щодо вибору типу довідки та зменшує кількість помилкових або неповних заявок.

Під час створення заявки студент повинен мати змогу заповнювати всі необхідні дані у відповідній електронній формі, причому система має автоматично підставляти базову інформацію про студента з університетської бази даних, таку як прізвище, ім'я, по батькові, факультет, спеціальність, курс, форма навчання та студентський квиток. Студенту залишається лише коректно ввести специфічну інформацію згідно з вимогами обраного типу довідки, наприклад, мету отримання документа, бажану дату готовності або адресу, куди потрібно надіслати довідку. Форма створення заявки повинна містити вбудовані механізми валідації полів, які перевіряють коректність введених даних у реальному часі, інформуючи користувача про помилки ще до відправки форми на сервер.

Студент за потреби має можливість скасовувати подану заявку до моменту початку її опрацювання працівниками деканату, що забезпечує гнучкість у випадках, коли студент помилково створив заявку, обрав неправильний тип довідки або відпала необхідність у документі. Скасовані заявки повинні позначатися відповідним статусом в історії, але не видалятися з бази даних повністю, що дозволяє вести повний аудит всіх операцій у системі. Крім функції скасування, інтерфейс повинен забезпечувати повний перегляд історії всіх заявок студента, включаючи зазначення їхніх поточних статусів, коментарів від працівників деканату, дати та часу створення, а також інформацію про те, хто і коли вносив зміни до заявки.

Система повинна реалізовувати потужні інструменти опрацювання заявок з боку деканату, які передбачають доступ працівників до переліку заявок, що стосуються студентів їхнього факультету, із забезпеченням повної ізоляції даних між факультетами для дотримання принципу конфіденційності. Інтерфейс

працівника деканату повинен надавати зручні засоби для швидкого перегляду великої кількості заявок через табличне представлення з можливістю налаштування видимих колонок, порядку сортування та збереження персональних налаштувань відображення.

Функціонал має забезпечувати можливість гнучкого сортування заявок за різними критеріями та багаторівневої фільтрації за статусом, що дозволяє швидко знаходити заявки, які потребують уваги, за датою подання для виявлення заявок, що очікують на обробку понад допустимий термін, за типом довідки для групової обробки однотипних запитів, за прізвищем студента для швидкого пошуку конкретної заявки або за будь-якою комбінацією цих критеріїв. Система фільтрації повинна підтримувати збереження часто використовуваних комбінацій фільтрів як персональних пресетів для подальшого швидкого застосування.

Працівники повинні мати право змінювати статуси заявок через зручний інтерфейс, позначаючи їх як нові, що щойно надійшли і ще не розглядалися, такі що перебувають в опрацюванні та активно обробляються відповідальним працівником, готові до видачі з можливістю формування остаточного документа або відхилені з обов'язковим зазначенням причини відмови у коментарях. Зміна статусу повинна автоматично фіксуватися в історії заявки з відміткою про час та ім'я працівника, який здійснив зміну, що забезпечує повну відстежуваність процесу обробки та відповідальність за кожну дію.

Окрім зміни статусів, працівники повинні мати змогу додавати внутрішні коментарі, видимі лише для персоналу деканату, які використовуються для координації роботи між колегами, або зовнішні коментарі, які бачить студент і які можуть містити уточнюючі питання, запити на додаткову інформацію або інструкції щодо отримання готової довідки. Система коментування повинна підтримувати форматування тексту, можливість прикріплення файлів та автоматичне сповіщення студента про появу нового коментаря, адресованого йому.

Також необхідною є можливість формування цифрових документів довідок на підставі даних заявки через використання попередньо створених та затверджених шаблонів документів, які автоматично заповнюються інформацією зі студентського профілю та специфічними даними із заявки. Система шаблонів повинна підтримувати різні формати документів, зокрема PDF для офіційних довідок з можливістю друку або DOCX для документів, які потребують подальшого редагування. Після формування документа працівник має можливість його попереднього перегляду для перевірки коректності всіх даних, внесення необхідних коригувань, завантаження остаточної версії та прикріплення до заявки для подальшого доступу студента або зберігання в архіві системи.

Платформа має включати розвинений модуль адміністрування, що дозволяє управляти обліковими записами користувачів через створення нових акаунтів для працівників деканату або адміністраторів, призначення та зміну ролей користувачів відповідно до їхніх посадових обов'язків, тимчасове блокування або остаточне видалення облікових записів при звільненні співробітників або закінченні навчання студентів. Адміністратор повинен мати змогу детально налаштовувати права доступу для кожної ролі, визначаючи, які саме функції системи доступні користувачам з певною роллю, які дані вони можуть переглядати, редагувати або видаляти, а також встановлювати обмеження на частоту виконання певних операцій для запобігання зловживанням.

Функціонал моніторингу активності повинен надавати адміністратору інструменти для відстеження дій користувачів у системі, перегляду статистики використання різних функцій, аналізу навантаження на систему в різні періоди часу та виявлення незвичайних патернів поведінки, які можуть свідчити про спроби несанкціонованого доступу або зловживання системою. Система повинна вести детальні логи всіх важливих подій, включаючи спроби входу в систему як успішні, так і невдалі, створення та зміну заявок з фіксацією всіх змінених полів, зміну статусів заявок, додавання коментарів, зміну прав доступу

користувачів, системні помилки та винятки, а також будь-які інші критичні операції. Логи повинні зберігатися у структурованому вигляді з можливістю фільтрації, пошуку та експорту для подальшого аналізу.

Адміністратор повинен мати змогу оновлювати перелік доступних типів довідок, додаючи нові типи документів у міру появи нових потреб університету, редагуючи описи та вимоги до існуючих типів довідок або видаляючи застарілі варіанти, які більше не використовуються, забезпечуючи динамічне розширення можливостей сервісу відповідно до актуальних вимог університету без необхідності внесення змін у програмний код. Кожен тип довідки повинен мати налаштування, які визначають, які поля обов'язкові для заповнення, які опціональні, який шаблон використовується для формування документа та хто має право обробляти заявки цього типу.

Система сповіщень має інформувати студента про зміну статусу його заявки в режимі, близькому до реального часу, що дозволяє студенту бути в курсі прогресу обробки його запиту без необхідності постійно перевіряти систему вручну. Повідомлення повинні відображатися у вебінтерфейсі через спеціальний розділ сповіщень, доступний з головної панелі навігації, та бути доступними для перегляду у будь-який момент з можливістю позначення як прочитаних або видалення. Система повинна підтримувати різні типи сповіщень: про зміну статусу заявки, про додавання нових коментарів від працівників деканату, про готовність довідки до отримання, про відхилення заявки з поясненням причин або про необхідність надання додаткової інформації.

Сама система сповіщень має бути побудована таким чином, щоб її можна було легко масштабувати у майбутньому шляхом інтеграції додаткових каналів комунікації, наприклад, автоматичного надсилання електронних листів на корпоративну пошту студента при важливих змінах статусу, SMS-повідомлень для критичних сповіщень, які потребують негайної уваги, push-сповіщень у випадку створення мобільного додатку або інтеграції з месенджерами, такими як Telegram або Viber, якщо студенти надають згоду на отримання повідомлень через ці канали. Архітектура системи сповіщень повинна дозволяти

користувачам налаштовувати свої преференції щодо типів та каналів отримання повідомлень, забезпечуючи гнучкість у комунікації відповідно до індивідуальних потреб.

1.4.2. Перелік нефункціональних вимог

Нефункціональні вимоги визначають якість функціонування веб-платформи та формують загальні критерії її надійності, продуктивності, безпеки й зручності використання. На відміну від функціональних вимог, які описують, що система має робити, нефункціональні вимоги визначають, наскільки добре система має це робити, встановлюючи стандарти якості, які повинні дотримуватися при реалізації всіх функціональних можливостей. Ці вимоги є критично важливими для забезпечення позитивного користувацького досвіду, довгострокової підтримуваності системи та її здатності адаптуватися до змінних умов експлуатації.

Система повинна забезпечувати стабільну та передбачувану продуктивність при різних рівнях навантаження, що проявляється у швидкій реакції інтерфейсу на дії користувача, де час завантаження будь-якої сторінки не має перевищувати двох секунд за умов типового навантаження при використанні сучасного широкосмугового підключення до інтернету. Це вимагає оптимізації розміру передаваних даних, мінімізації кількості HTTP-запитів, використання ефективних алгоритмів рендерингу на стороні клієнта та застосування технік кешування для часто запитуваних ресурсів. Інтерфейс повинен надавати візуальний зворотний зв'язок користувачу під час виконання операцій, які займають більше половини секунди, через індикатори завантаження або анімації, що створює відчуття реактивності навіть при тривалих операціях.

API повинен гарантувати високу оперативність відповіді, що не перевищує п'ятисот мілісекунд для стандартних запитів на отримання даних або виконання простих операцій, таких як зміна статусу заявки або додавання коментаря, забезпечуючи комфортну роботу користувачів і плавний досвід взаємодії без

відчутних затримок. Для складніших операцій, таких як формування документів на основі шаблонів або виконання складних аналітичних запитів, допускається час відповіді до двох секунд. Серверна частина повинна бути оптимізована для мінімізації кількості запитів до бази даних через використання ефективних SQL-запитів, індексування критичних полів та застосування кешування на рівні додатку для часто запитуваних даних, які рідко змінюються.

Платформа має підтримувати одночасну роботу не менше ніж двохсот активних користувачів без зниження стабільності чи продуктивності, причому під активними користувачами розуміються ті, хто активно взаємодіє з системою, виконуючи операції створення заявок, перегляду даних або обробки документів. Це вимагає проєктування архітектури з урахуванням багатопотоковості, ефективного управління ресурсами сервера та оптимізації алгоритмів обробки даних. Система повинна корректно працювати навіть при пікових навантаженнях, які можуть виникати на початку семестру, перед сесіями або в інші періоди масового звернення студентів за довідками, коли кількість одночасних користувачів може перевищувати середні показники у декілька разів.

Архітектура системи повинна бути масштабованою та гнучкою, передбачати можливість легкого розширення функціональності через додавання нових модулів або вдосконалення існуючих та інтеграції нових модулів без суттєвих змін в основному коді завдяки дотриманню принципів модульності, слабкого зв'язування компонентів та використанню стандартних інтерфейсів взаємодії. Це досягається через застосування архітектурних патернів, таких як розділення відповідальності між шарами додатку, використання dependency injection для управління залежностями та застосування event-driven підходів для асинхронної взаємодії між компонентами.

Важливо, щоб платформа могла підтримувати горизонтальне масштабування в умовах збільшення кількості студентів в університеті або зростання частоти запитів при розширенні функціональності системи, що означає можливість додавання нових серверів або інстансів застосунку для

розподілу навантаження без необхідності змін у архітектурі або програмному коді. Використання хмарних рішень, зокрема Microsoft Azure, забезпечує автоматичне керування навантаженням за допомогою механізмів автоскейлінгу, які моніторять поточне використання ресурсів і автоматично збільшують або зменшують кількість інстансів додатку відповідно до поточних потреб, оптимізуючи як продуктивність, так і вартість експлуатації.

Система має бути надійною та забезпечувати безперервність роботи з рівнем доступності не нижче ніж дев'яносто вісім відсотків упродовж року, що допускає сумарний час недоступності не більше приблизно сімдесяти двох годин на рік або близько шести годин на місяць. Цей показник включає як заплановані технічні роботи, так і незаплановані збої, тому критично важливо мінімізувати тривалість обслуговування системи та забезпечити швидке відновлення після інцидентів. У випадку технічних збоїв, таких як відмова серверного обладнання, проблеми з мережевим підключенням або помилки в програмному коді, система повинна мати можливість автоматичного відновлення функціонування без втрати інформації, що особливо важливо для збереження заявок і даних користувачів, які можуть бути створені або змінені безпосередньо перед збоєм.

Для забезпечення високої доступності система повинна використовувати механізми резервування критичних компонентів, балансування навантаження між декількома серверами, автоматичного перемикання на резервні інстанси у разі виявлення проблем з основними та health check endpoints, які регулярно перевіряють стан різних компонентів системи і можуть ініціювати процедури відновлення при виявленні аномалій. Регулярне резервне копіювання бази даних повинно здійснюватися щонайменше один раз на добу, причому бекапи мають створюватися в автоматичному режимі без впливу на продуктивність системи і зберігатися у географічно розподілених сховищах для захисту від локальних катастроф, що забезпечить можливість відновлення критичних даних у разі аварійних ситуацій з мінімальною втратою інформації.

Безпека є одним із найважливіших нефункціональних аспектів системи, оскільки платформа обробляє персональні дані студентів, включаючи їхні імена,

дати народження, адреси, контактну інформацію та академічні записи, які підлягають захисту відповідно до законодавства України та міжнародних стандартів захисту даних. Усі запити між клієнтською частиною, що виконується в браузері користувача, і сервером повинні бути захищені протоколом HTTPS з використанням сучасних версій TLS для шифрування даних під час передачі і запобігання їх перехопленню зловмисниками. Сервер повинен бути налаштований на відмову від з'єднань через незахищений протокол HTTP і автоматичне перенаправлення всіх запитів на HTTPS-версію.

Автентифікація користувачів та авторизація доступу до ресурсів мають реалізовуватися з використанням сучасних стандартів OAuth 2.0 для делегованої автентифікації через Google і JWT-токенів для підтримки сесій користувачів та передачі інформації про їхні права доступу між клієнтом і сервером. JWT-токени повинні мати обмежений термін дії, зазвичай декілька годин для access tokens і декілька днів або тижнів для refresh tokens, що мінімізує ризики у випадку їх компрометації. Токени повинні бути підписані криптографічно стійким алгоритмом і містити всю необхідну інформацію про користувача та його ролі, що дозволяє серверу перевіряти права доступу без додаткових запитів до бази даних на кожному запиті.

Система повинна мати багаторівневий захист від автоматизованих атак через механізми обмеження частоти запитів, які дозволяють обмежити кількість запитів від одного користувача або IP-адреси протягом певного періоду часу, запобігаючи brute-force атакам на паролі, спробам перевантаження системи через DDoS-атаки або зловживанням API через автоматизовані скрипти. Rate limiting повинен застосовуватися диференційовано залежно від типу операції: більш жорсткі обмеження для критичних операцій, таких як спроби входу в систему, і більш м'які для звичайних операцій перегляду даних.

Застосування Google reCAPTCHA на формах, доступних без автентифікації або на критичних формах після автентифікації, додає додатковий рівень захисту, перевіряючи, що запити надсилаються реальними людьми, а не автоматизованими ботами. Всебічна валідація даних повинна виконуватися як на

стороні клієнта для забезпечення зручності користувача через негайний зворотний зв'язок про помилки введення, так і обов'язково на стороні сервера для гарантування того, що в базу даних не потраплять некоректні або зловмисні дані, навіть якщо клієнтська валідація була обійдена. Валідація повинна перевіряти тип даних, формат, діапазон значень, відповідність регулярним виразам для складних форматів та бізнес-правилам, специфічним для домену застосунку.

Чутливі дані, такі як токени автентифікації, секретні ключі для інтеграції з зовнішніми сервісами або інші конфіденційні параметри конфігурації, необхідно шифрувати при зберіганні у базі даних або файлах конфігурації і ніколи не передавати у відкритому вигляді. Для зберігання секретів рекомендується використовувати спеціалізовані сервіси, такі як Azure Key Vault, які забезпечують захищене зберігання і контрольований доступ до секретів. Програмний код повинен бути захищений від поширених типів атак через дотримання безпечних практик програмування: захист від XSS-атак через екранування всіх даних, що виводяться в HTML, і використання Content Security Policy, захист від CSRF-атак через використання anti-forgery tokens і перевірку origin headers, захист від SQL Injection через використання параметризованих запитів або ORM, які автоматично екранують вхідні дані, а також захист від інших типів ін'єкцій і вразливостей, визначених у OWASP Top 10.

Доступ до функцій платформи має визначатися згідно з принципами найменших привілеїв і ролевої моделі доступу, що означає, що кожен користувач повинен мати доступ тільки до тих ресурсів і функцій, які абсолютно необхідні для виконання його обов'язків, і не більше. Права доступу повинні перевірятися на кожному запиті до API як на рівні контролерів, так і на рівні бізнес-логіки, гарантуючи, що навіть якщо зловмисник знайде спосіб обійти клієнтські перевірки, серверна частина все одно запобіжить несанкціонованим діям.

Зручність користування є важливою складовою успіху системи, оскільки навіть найбільш функціональна платформа не буде ефективною, якщо користувачі відчувають труднощі при роботі з нею. Інтерфейс повинен бути інтуїтивним і логічно структурованим, що дозволить студентам швидко та без

труднощів створювати заявки, виконуючи мінімальну кількість дій та не потребуючи детальних інструкцій або навчання. Навігація має бути очевидною, з чітко позначеними розділами і кнопками, які використовують зрозумілу термінологію. Дизайн повинен слідувати усталеним конвенціям веб-інтерфейсів, щоб користувачі могли застосовувати свій попередній досвід роботи з іншими вебсайтами.

Дизайн повинен коректно відображатися на мобільних пристроях різних розмірів, від невеликих смартфонів до великих планшетів, автоматично адаптуючись до ширини екрану і змінюючи компонування елементів для забезпечення зручності користування на сенсорних екранах. Це вимагає застосування responsive design підходів, використання гнучких сіток, media queries і відносних одиниць вимірювання замість фіксованих пікселів. Елементи управління повинні бути достатньо великими для комфортного натискання пальцем, а відстані між інтерактивними елементами мають запобігати випадковим натисканням.

Платформа має бути доступною для користувачів з особливими потребами, дотримуючись принципів веб-доступності, визначених у WCAG. Це включає забезпечення достатнього контрасту між текстом і фоном для користувачів зі зниженим зором, можливість навігації клавіатурою без використання миші, наявність альтернативного тексту для зображень і підтримку програм читання з екрану, коректне використання семантичних HTML-елементів і ARIA-атрибутів для забезпечення правильної інтерпретації структури сторінки асистивними технологіями.

Система повинна бути сумісною з основними сучасними браузерами, включно з Google Chrome, Mozilla Firefox, Microsoft Edge і Safari, забезпечуючи однаковий функціонал і схожий візуальний вигляд незалежно від вибору браузера користувачем. Це вимагає тестування на різних браузерах і використання поліфілів або транспіляції для підтримки сучасних JavaScript-функцій у старіших версіях браузерів. API системи має реалізовувати REST-підхід для забезпечення можливості інтеграції з іншими внутрішніми сервісами

університету, використовуючи стандартні HTTP-методи, статус-коди і формат даних JSON, що робить API зрозумілим і легким для використання розробниками інших систем.

Кодова база, написана на основі .NET для серверної частини та React для клієнтської частини, повинна відповідати загальноприйнятим стандартам і найкращим практикам цих технологій, що дозволяє ефективно проводити командну розробку та підтримку. Код має бути організований модульно, з чіткою структурою папок і файлів, використовувати змістовні назви змінних і функцій, слідувати єдиному стилю кодування в межах всього проєкту і включати коментарі для пояснення складних або неочевидних частин логіки.

Платформа повинна бути зручною у підтримці завдяки модульній архітектурі, яка спрощує внесення змін, розширення функціоналу та усунення помилок без ризику пошкодження інших частин системи. Модулі повинні бути слабко пов'язані між собою і взаємодіяти через чітко визначені інтерфейси, що дозволяє змінювати внутрішню реалізацію модуля без впливу на інші компоненти. Код API і фронтенду має бути добре документований через коментарі в коді, окрему технічну документацію, що описує архітектуру системи, основні компоненти і потоки даних, а також документацію API у форматі OpenAPI, що сприятиме швидкому розумінню структури проєкту новими розробниками і полегшить процес онбордингу.

Логування подій, помилок і системної активності повинно бути централізованим і структурованим, щоб забезпечити можливість оперативного аналізу та виявлення проблемних ділянок через збір логів з різних компонентів системи в єдиному сховищі, використання структурованих форматів, таких як JSON, для полегшення автоматичного парсингу і аналізу логів, включення контекстної інформації в кожен лог-запис, наприклад, ідентифікатор користувача, ідентифікатор запиту, час події, а також застосування різних рівнів логування для розділення критичних помилок, попереджень і інформаційних повідомлень. У сукупності перелічені нефункціональні вимоги формують цілісний фундамент для стабільної, безпечної, ефективної та зручної у

використанні вебплатформи, яка здатна задовольнити потреби всіх категорій користувачів і залишатися актуальною протягом тривалого часу експлуатації.

1.5. Архітектурні принципи та концепція побудови системи

Архітектурна модель вебсистеми автоматизованого оформлення студентських довідок сформована з урахуванням вимог до масштабованості, надійності, безпеки та зручності подальшого розвитку і підтримки. У процесі проєктування було обрано клієнт-серверну архітектуру, у якій фронтенд і бекенд функціонують як два окремі, незалежні один від одного модулі, що комунікують через стандартизований REST API. Такий підхід дає змогу забезпечити високу гнучкість системи, можливість розгортання складових на різних інфраструктурних платформах та розподіл навантаження між клієнтськими та серверними ресурсами.

Рішення про розділення фронтенду та бекенду як окремих застосунків було прийнято з огляду на декілька важливих факторів. По-перше, це дозволяє розробляти та розгортати кожну частину системи незалежно, що значно прискорює процес впровадження нових функцій та виправлення помилок без необхідності перезапуску всієї системи. По-друге, така архітектура забезпечує можливість використання різних технологічних стеків для клієнтської та серверної частин, обираючи найбільш підходящі інструменти для вирішення специфічних завдань кожного рівня. По-третє, розділення дозволяє незалежно масштабувати фронтенд та бекенд відповідно до фактичного навантаження - наприклад, якщо виникає потреба в обробці великої кількості одночасних запитів, можна збільшити кількість серверних інстансів без змін у клієнтській частині.

Використання REST API як єдиного каналу комунікації між клієнтом і сервером забезпечує стандартизований та добре зрозумілий інтерфейс взаємодії, який базується на принципах архітектурного стилю REST та використовує стандартні HTTP-методи для виконання операцій. Це робить API інтуїтивно

зрозумілим для розробників, які мають досвід роботи з вебтехнологіями, та полегшує майбутню інтеграцію з іншими системами університету. REST API підтримує чітку структуру endpoint-ів, кожен з яких відповідає за конкретний ресурс або операцію, використовує JSON як формат обміну даними, що є універсальним і легким для парсингу, та дотримується принципів stateless комунікації, коли кожен запит містить всю необхідну інформацію для його обробки.

Архітектура орієнтована на подальше масштабування як вертикальне, шляхом збільшення потужності серверів, так і горизонтальне, через додавання нових інстансів застосунку для розподілу навантаження. Передбачена можливість інтеграції з іншими університетськими сервісами, такими як електронний деканат, система управління навчальним процесом, бібліотечна система або система обліку студентів у гуртожитках, через використання стандартизованих протоколів обміну даними та добре документованого API. Система спроєктована таким чином, що розширення функціональності може відбуватися шляхом додавання нових модулів або сервісів без необхідності повної перебудови існуючої архітектури, що забезпечує довгостроковість інвестицій у розробку та можливість поступової еволюції системи відповідно до змінних потреб університету.

Важливим аспектом архітектурного проєктування стало забезпечення балансу між складністю та практичністю. Замість надмірно складної мікросервісної архітектури, яка могла б створити додаткові труднощі в розробці та підтримці для невеликої команди, було обрано модульний монолітний підхід для бекенду з чітким розділенням відповідальності між шарами застосунку. Це дозволяє зберігати простоту розгортання та налагодження, характерну для монолітних застосунків, водночас забезпечуючи структурованість та можливість майбутнього виділення окремих модулів у самостійні сервіси, якщо виникне така потреба в міру зростання системи та команди розробки.

1.5.1. Бекенд: архітектура та принципи побудови

Серверна частина платформи реалізована на основі ASP.NET Core 8 Web API, що забезпечує високу продуктивність, стабільність, гнучкість та відповідність сучасним стандартам для корпоративних застосунків. Вибір ASP.NET Core 8 як основної технології для серверної частини був обумовлений декількома ключовими факторами: по-перше, це кросплатформний фреймворк, який може працювати на Windows, Linux та macOS, що надає гнучкість у виборі інфраструктури для розгортання; по-друге, ASP.NET Core демонструє видатну продуктивність у порівнянні з іншими популярними фреймворками для веб-розробки, що підтверджується численними бенчмарками; по-третє, наявність потужної екосистеми бібліотек, інструментів розробки та активної спільноти розробників забезпечує швидке вирішення технічних проблем та доступ до готових рішень для типових завдань.

Архітектурною основою серверної частини є концепція "Чистої архітектури" (Clean Architecture), яка передбачає чітке відокремлення доменної моделі, що описує бізнес-сутності та їхні взаємозв'язки, бізнес-логіки, яка реалізує правила та процеси обробки даних, та інфраструктурних механізмів доступу до даних, які відповідають за взаємодію з базою даних, зовнішніми API та іншими технічними аспектами. Така організація коду створює концентричну структуру шарів, де внутрішні шари не залежать від зовнішніх, а зовнішні шари знають про внутрішні через абстракції. Це дозволяє уникнути жорстких залежностей між модулями, коли зміни в одній частині системи автоматично вимагають змін в інших частинах, та забезпечує можливість незалежного розвитку окремих компонентів, їх тестування в ізоляції та заміни реалізації без впливу на решту системи.

Практична реалізація Clean Architecture у проєкті передбачає розділення коду на декілька логічних проектів або namespace-ів: Domain layer містить бізнес-сутності, інтерфейси та доменні винятки без залежностей від інфраструктурних деталей; Application layer включає бізнес-логіку, use cases,

валідацію та інтерфейси для роботи з репозиторіями; Infrastructure layer реалізує конкретні механізми доступу до даних, інтеграції з зовнішніми сервісами та інші технічні аспекти; Presentation layer (Web API) обробляє HTTP-запити, виконує маршрутизацію, валідацію вхідних даних та формування відповідей. Така структура забезпечує високу тестованість коду, оскільки бізнес-логіка може тестуватися незалежно від деталей інфраструктури через використання mock-об'єктів замість реальних реалізацій.

У межах роботи з базою даних застосовується репозиторний підхід у поєднанні з шаблоном Unit of Work, що забезпечує впорядкованість і контроль над транзакціями та інкапсулює логіку доступу до даних. Репозиторії надають абстрактний інтерфейс для виконання операцій над сутностями, приховуючи деталі реалізації запитів до бази даних від бізнес-логіки, що дозволяє легко замінити конкретну реалізацію доступу до даних, наприклад, перейти з SQL Server на PostgreSQL або використовувати in-memory базу даних для тестування. Unit of Work координує роботу декількох репозиторіїв у межах однієї транзакції, гарантуючи атомарність складних операцій, які включають зміни в декількох таблицях, та забезпечуючи, що всі зміни або застосовуються повністю, або відкочуються у разі помилки.

Для відображення доменних сутностей у базі даних використовується ORM Entity Framework Core, яка гарантує типобезпечний доступ до SQL Server та мінімізує помилки, пов'язані з ручним написанням SQL-запитів, такі як синтаксичні помилки або проблеми з екрануванням спеціальних символів. Entity Framework Core використовує LINQ для формування запитів, що дозволяє писати їх природною мовою C# з перевіркою типів на етапі компіляції, а також підтримує міграції для версіонування схеми бази даних, що спрощує процес розгортання змін у структурі бази на різних середовищах. ORM автоматично обробляє складні аспекти, такі як відстеження змін у сутностях, lazy loading пов'язаних даних та оптимізація запитів через eager loading для зменшення кількості звернень до бази даних.

Передача даних між клієнтом і сервером організована на основі Data Transfer Objects (DTO), які є спеціальними класами, призначеними виключно для передачі даних без додаткової бізнес-логіки. Використання DTO дозволяє контролювати, які саме поля сутностей передаються клієнту, приховуючи внутрішні деталі реалізації та чутливу інформацію, зменшує обсяг даних, що передаються по мережі, через включення лише необхідних полів, а також забезпечує стабільність API, оскільки зміни у внутрішній структурі доменних моделей не впливають автоматично на зовнішній контракт API. AutoMapper автоматизує процес перетворення між доменними моделями та DTO, забезпечуючи однорідність коду і читабельність через декларативне налаштування мапінгу замість написання повторюваного коду для копіювання значень між об'єктами. Це значно зменшує обсяг boilerplate коду та ризик помилок при ручному копіюванні даних між моделями.

Система автентифікації й авторизації побудована на основі механізмів JWT-токенів (JSON Web Tokens), що гарантує безпечну ідентифікацію користувачів та запобігає несанкціонованому доступу до захищених ресурсів API. JWT-токени є самодостатніми, тобто містять всю необхідну інформацію про користувача та його права у зашифрованому вигляді, що дозволяє серверу перевіряти автентичність запиту без необхідності звертатися до бази даних на кожному запиті. Токени підписуються криптографічним ключем, відомим лише серверу, що унеможливлює їх підробку зловмисниками, та мають обмежений термін дії, зазвичай кілька годин для access token, що мінімізує вікно вразливості у разі їх компрометації.

Для інтеграції з університетським акаунтом впроваджено Google OAuth 2.0, що дозволяє студентам та співробітникам входити до системи, використовуючи свої корпоративні облікові записи Google без необхідності створення окремих паролів для цієї платформи. Процес автентифікації через Google OAuth включає перенаправлення користувача на сторінку входу Google, де він вводить свої облікові дані, отримання authorization code після успішної автентифікації, обмін цього коду на access token та отримання базової інформації

про користувача з Google API. Система автоматично створює локальний профіль користувача при першому вході або оновлює існуючу інформацію, після чого генерує власний JWT-токен для подальшої роботи з API.

Рольова модель доступу реалізована за допомогою Policy-based Authorization у ASP.NET Core, що дозволяє визначати конкретні політики доступу для ролей "студент", "деканат" та "адміністратор" на основі складних правил, що виходять за межі простої перевірки наявності ролі. Політики можуть включати перевірку додаткових claims у токені, таких як належність до конкретного факультету для працівників деканату, перевірку часу дії токена або інших умов. Кожен endpoint API анотується атрибутами, які вказують, яка політика має бути виконана для доступу до цього ресурсу, що забезпечує декларативний та зрозумілий підхід до контролю доступу безпосередньо в коді контролерів.

Журналювання подій, помилок та системних операцій виконується через вбудовані механізми логування ASP.NET Core у поєднанні з додатковими провайдерами, такими як Serilog або NLog, що забезпечує гнучкість у налаштуванні форматів логів, їх збереження та маршрутизації. Логи записуються у структурованому форматі з різними рівнями важливості - від детальної діагностичної інформації до критичних помилок - що дозволяє налаштовувати різний рівень деталізації для різних середовищ: максимальна деталізація в середовищі розробки для полегшення налагодження та мінімальна у продуктивному середовищі для зменшення навантаження. Централізоване збереження логів, наприклад, у Azure Application Insights або інших системах моніторингу, забезпечує контроль за стабільністю системи, швидке виявлення проблем та можливість аналізу патернів помилок для проактивного вдосконалення якості коду.

1.5.2. Фронтенд: архітектурні рішення

Клієнтська частина системи реалізована з використанням React у поєднанні з TypeScript, що надає змогу створювати модульний, передбачуваний і легко підтримуваний інтерфейс. Вибір React як основної бібліотеки для побудови користувацького інтерфейсу обумовлений його зрілістю, широким поширенням у індустрії та наявністю величезної екосистеми готових компонентів і бібліотек. React забезпечує ефективний рендеринг через використання віртуального DOM, що мінімізує кількість операцій з реальним DOM браузера і гарантує високу продуктивність навіть при роботі зі складними інтерфейсами. Додавання TypeScript до проєкту посилює надійність коду через статичну типізацію, яка дозволяє виявляти помилки на етапі компіляції замість виконання, забезпечує кращу підтримку в IDE через автодоповнення та інтелектуальні підказки, а також служить живою документацією коду через явне визначення типів даних, що передаються між компонентами.

Побудова фронтенду базується на компонентному підході, коли інтерфейс розбивається на незалежні, переосмислювані елементи, кожен з яких відповідає за певну логіку або частину представлення та має чітко визначений інтерфейс взаємодії через props. Компоненти можуть бути як функціональними, що є простими функціями, які приймають props та повертають JSX-розмітку, так і більш складними, що включають внутрішній стан, lifecycle методи або side effects через hooks. Такий підхід значно спрощує масштабування системи, оскільки нові функції можуть додаватися через створення нових компонентів без необхідності модифікації існуючих, полегшує тестування завдяки можливості ізольованого тестування кожного компонента окремо від решти системи, та забезпечує ефективне повторне використання компонентів у різних частинах системи або навіть у різних проектах через створення бібліотеки переосмислюваних компонентів.

Структура проєкту організована таким чином, щоб логічно групувати пов'язані компоненти, стилі, утиліти та типи. Основні розділи включають папку

components, де зберігаються переосмислювані UI-компоненти, такі як кнопки, поля вводу, модальні вікна; папку pages, яка містить компоненти сторінок, що відповідають конкретним маршрутам додатку; папку services, де розміщені модулі для взаємодії з API; папку hooks для користувацьких React hooks, що інкапсулюють логіку роботи зі станом або side effects; папку types для TypeScript-визначень інтерфейсів і типів даних; та папку utils для допоміжних функцій і констант. Така організація забезпечує легку навігацію проектом і швидке знаходження потрібного коду.

Для організації маршрутизації використовується React Router, який забезпечує побудову ролевих маршрутів та обмеження доступу до сторінок відповідно до ролі користувача через створення декларативної структури маршрутів, де кожен шлях асоційований з відповідним компонентом сторінки. React Router підтримує вкладені маршрути, що дозволяє створювати складну ієрархію сторінок з загальними елементами, такими як навігаційне меню або хедер, динамічні параметри маршрутів для передачі ідентифікаторів ресурсів, та програмну навігацію через useNavigate hook для переходу між сторінками у відповідь на дії користувача або зміни стану. Захищені маршрути реалізовані через створення компонента-обгортки, який перевіряє наявність валідного токена автентифікації та відповідну роль користувача перед рендерингом запитаної сторінки, перенаправляючи неавторизованих користувачів на сторінку входу або користувачів без відповідних прав на сторінку помилки доступу.

Взаємодія з API здійснюється через бібліотеку Axios, яка надає зручний інтерфейс для виконання HTTP-запитів із використанням інтерсепторів, що дозволяє централізовано обробляти різні аспекти комунікації з сервером. Request interceptors застосовуються для автоматичного додавання JWT-токена до заголовка Authorization кожного запиту, що усуває необхідність вручну додавати токен у кожному місці, де виконується запит до API, а також для додавання інших заголовків, таких як Content-Type або custom headers для відстеження запитів. Response interceptors обробляють відповіді від сервера, дозволяючи централізовано обробляти помилки через перехоплення HTTP-статус кодів

помилок і відображення відповідних повідомлень користувачу, автоматично оновлювати токени автентифікації при отриманні статусу 401 Unauthorized через використання refresh token, та контролювати повторні запити у разі їх тимчасової невдачі, наприклад, при проблемах з мережею.

Управління станом у застосунку реалізоване за допомогою React Hooks, зокрема useState для локального стану компонентів, useEffect для виконання side effects, таких як запити до API або підписка на події, та useContext для глобального стану, що потрібен багатьом компонентам, таким як інформація про поточного користувача або налаштування теми інтерфейсу. Це забезпечує контроль авторизації через зберігання інформації про токен та роль користувача в контексті, управління станом форм, включаючи значення полів, помилки валідації та стан відправки, відображення заявок з можливістю фільтрації, сортування та пагінації, а також управління іншими інтерфейсними елементами, такими як модальні вікна, сповіщення або індикатори завантаження. Для складніших випадків управління станом, де потрібна більш витончена логіка оновлення, використовується useReducer hook, який дозволяє описувати зміни стану через actions і reducer функцію подібно до Redux, але без необхідності підключення зовнішньої бібліотеки.

Інтерфейс розроблено повністю адаптивним з підходом mobile-first, що гарантує коректне відображення та зручність використання на мобільних пристроях з невеликими екранами, планшетах середнього розміру і широкоформатних екранах настільних комп'ютерів. Адаптивність досягається через використання CSS media queries, які застосовують різні стилі залежно від ширини екрану пристрою, гнучких сіток на основі Flexbox або CSS Grid, які автоматично перерозподіляють елементи при зміні розміру вікна, та відносних одиниць вимірювання, таких як відсотки, em або rem, замість фіксованих пікселів. Елементи управління на мобільних пристроях мають достатній розмір для комфортного натискання пальцем відповідно до рекомендацій по доступності, а складні таблиці та списки трансформуються у більш зручні для

мобільних пристроїв формати, наприклад, через використання карток замість табличних рядків.

Додатково передбачено багаторівневі механізми захисту інтерфейсу для запобігання несанкціонованому доступу або помилкових дій користувачів. Guard-компоненти виконують перевірку прав доступу перед рендерингом захищеного контенту і можуть приховувати певні елементи інтерфейсу або цілі сторінки для користувачів, які не мають відповідних привілеїв.

1.5.3. База даних та моделювання

База даних створена на основі Microsoft SQL Server, що забезпечує високу продуктивність, надійність та можливість масштабування. Вибір SQL Server як системи управління базами даних обумовлений його зрілістю, широкою підтримкою з боку ASP.NET Core через Entity Framework Core, наявністю потужних інструментів для адміністрування та моніторингу, а також можливістю використання хмарної версії Azure SQL Database для спрощення розгортання та управління. SQL Server забезпечує підтримку ACID-властивостей транзакцій, що гарантує атомарність, узгодженість, ізольованість та довговічність операцій з даними навіть у випадку збоїв або одночасного доступу багатьох користувачів. Крім того, SQL Server надає потужні можливості для створення індексів, що значно прискорює виконання складних запитів на вибірку даних, підтримку повнотекстового пошуку для ефективного пошуку за текстовими полями, а також механізми резервного копіювання та відновлення даних для забезпечення надійності зберігання критичної інформації.

Структура бази даних спроєктована відповідно до принципів нормалізації, зокрема третьої нормальної форми, що забезпечує мінімізацію надмірності даних через усунення дублювання інформації в різних таблицях та збереження цілісності даних під час виконання операцій читання та запису через використання зовнішніх ключів та каскадних правил. Нормалізація дозволяє уникнути аномалій при вставці, оновленні або видаленні даних, коли зміна в

одному місці повинна автоматично відображатися або контролюватися у пов'язаних записах, та забезпечує логічну організацію інформації, де кожна таблиця відповідає за зберігання конкретної сутності або відношення між сутностями. Для забезпечення швидкості виконання запитів критичні поля таблиць, які часто використовуються в умовах WHERE або JOIN, покриті індексами, що дозволяє базі даних швидко знаходити потрібні записи без необхідності сканування всієї таблиці.

Основу моделі становлять таблиці, що відображають ключові сутності системи та їхні взаємозв'язки. Таблиця AspNetUsers містить інформацію про всіх користувачів системи, включаючи студентів, працівників деканату та адміністраторів, зберігаючи дані про їх ідентифікацію через Google OAuth, такі як унікальний ідентифікатор користувача, електронна адреса, ім'я, прізвище, дата створення облікового запису, дата останнього входу в систему, а також зв'язки з іншими таблицями для визначення ролей та прав доступу. Таблиця Roles містить перелік всіх ролей, доступних у системі, таких як Student, FacultyAdmin, FacultyCurator та SystemAdmin, кожна з яких має свій унікальний ідентифікатор та опис призначення, що використовується для реалізації рольової моделі доступу та контролю того, які дії може виконувати користувач з певною роллю.

Таблиця Certificates визначає типи довідок, доступні для вибору студентами при створенні заявки, зберігаючи інформацію про назву кожного типу довідки, її детальний опис, необхідні поля для заповнення студентом, шаблон документа, який використовується для формування остаточної довідки, орієнтовний термін виготовлення, статус активності типу довідки, що дозволяє тимчасово приховувати певні типи без видалення з бази даних, а також мета-інформацію про те, хто та коли створив або останній раз модифікував цей тип довідки. Таблиця Orders містить подані студентами заявки на отримання довідок, зберігаючи зовнішній ключ на користувача, який створив заявку, зовнішній ключ на тип довідки, дату та час створення заявки, поточний статус заявки з можливими значеннями як New, InProgress, Completed, Rejected, додаткову

інформацію, надану студентом при створенні заявки, коментарі від працівників деканату до студента, внутрішні примітки для персоналу, посилання на сформований документ у разі його наявності, та аудит-поля, що фіксують, хто та коли змінював заявку.

Додаткові таблиці включають Faculties для зберігання інформації про факультети університету, що використовується для фільтрації заявок та призначення працівників до конкретних факультетів, JwtTokens для зберігання виданих JWT-токенів з можливістю їх інвалідації при виході користувача або з міркувань безпеки, GoogleTokens для збереження токенів доступу та оновлення від Google OAuth, що дозволяє виконувати запити до Google API від імені користувача, та AuditLog для ведення повного журналу всіх критичних операцій у системі з фіксацією того, хто, коли, що та чому змінив у базі даних. Така модель дозволяє легко розширювати структуру бази даних у разі необхідності додавання нових функцій, наприклад, таблиці для зберігання шаблонів документів, історії змін статусів заявок, системи повідомлень між студентами та деканатом або інтеграції з іншими університетськими системами, зберігаючи логічну цілісність системи через дотримання встановлених зв'язків та обмежень цілісності, та підтримуючи можливість швидкого додавання нових модулів без необхідності рефакторингу існуючих таблиць або міграції великих обсягів даних.

1.5.4. Інтеграція та взаємодія компонентів

Взаємодія між фронтендом і бекендом здійснюється через стандартизований REST API, який працює на основі HTTP-методів GET, POST, PUT, PATCH та DELETE, що дозволяють виконувати операції створення, читання, оновлення та видалення даних відповідно до семантики кожного методу. REST API побудований згідно з принципами архітектурного стилю REST, що включає використання іменників для позначення ресурсів у URL-адресах, застосування відповідних HTTP-статус кодів для індикації результату операції, підтримку stateless комунікації, коли кожен запит містить всю

необхідну інформацію для його обробки без збереження стану на сервері між запитами, та використання стандартних заголовків HTTP для передачі мета-інформації, такої як тип контенту, токени автентифікації або версія API.

Формат обміну даними базується на JSON (JavaScript Object Notation), що забезпечує універсальність завдяки широкій підтримці цього формату у всіх сучасних мовах програмування та платформах, легкість парсингу та серіалізації як на клієнті, так і на сервері, зрозумілість структури даних для людини при налагодженні та документуванні, а також сумісність із більшістю сучасних клієнтських застосунків, включаючи веббраузери, мобільні додатки та інші системи, які потребують доступу до API. JSON дозволяє передавати складні структури даних, включаючи вкладені об'єкти та масиви, зберігаючи при цьому компактність порівняно з XML та іншими форматами.

API підтримує версіонування через включення номера версії безпосередньо в URL-адресу ендпоінтів, наприклад, /api/v1/orders або /api/v2/orders, що дозволяє впроваджувати нові функції, змінювати структуру відповідей або модифікувати бізнес-логіку без порушення роботи існуючих клієнтських застосунків, які продовжують використовувати попередню версію API. Такий підхід забезпечує стабільність під час масштабування системи або зміни бізнес-логіки, дозволяючи поступово мігрувати клієнтські застосунки на нову версію API у зручний час без необхідності одночасного оновлення всіх компонентів системи. Кожна версія API супроводжується детальною документацією у форматі OpenAPI (Swagger), яка описує всі доступні ендпоінти, їхні параметри, формати запитів і відповідей, можливі коди помилок та приклади використання.

Архітектура системи передбачає широкі можливості інтеграції з іншими інформаційними системами університету для створення єдиного інформаційного простору та усунення дублювання даних. Інтеграція з електронним деканатом дозволяє автоматично отримувати актуальну інформацію про студентів, їхній поточний статус навчання, курс, спеціальність та факультет, що виключає необхідність ручного введення цих даних при оформленні довідок. Зв'язок зі

студентським кабінетом забезпечує єдиний вхід користувачів через Single Sign-On механізми та можливість доступу до системи оформлення довідок безпосередньо з особистого кабінету студента без необхідності окремої автентифікації.

Інтеграція з зовнішніми сервісами генерації PDF-документів, такими як спеціалізовані бібліотеки або хмарні сервіси, дозволяє автоматично формувати довідки у стандартизованому форматі на основі попередньо створених шаблонів, забезпечуючи їх професійний вигляд та відповідність вимогам університету. Система також може інтегруватися з сервісами електронного цифрового підпису для автоматичного підписання сформованих довідок уповноваженими особами, що додатково підвищує їхню юридичну значущість та усуває необхідність друку і фізичного підписання документів. Можлива інтеграція з поштовими сервісами для автоматичного надсилання готових довідок на електронну пошту студентів або з системами електронного документообігу університету для автоматичного архівування всіх виданих документів.

1.5.5. Розгортання та інфраструктура

Основним середовищем функціонування програмного комплексу обрано хмарну платформу Microsoft Azure, що дозволяє реалізувати гнучку архітектуру з високим рівнем доступності та відмовостійкості. Базові обчислювальні потужності організовано з використанням керованих сервісів, які підтримують контейнеризацію застосунків, що забезпечує ізоляцію компонентів системи та усуває конфлікти залежностей між різними модулями. Використання хмарних ресурсів дозволяє динамічно керувати навантаженням через налаштування правил автоматичного масштабування, які реагують на зміни у споживанні процесорного часу або оперативної пам'яті, автоматично додаючи або вивільняючи екземпляри сервісів для підтримки стабільної швидкодії навіть у моменти пікової активності користувачів.

Критично важливим елементом інфраструктури є комплексна система безпеки та управління доступом. Для захисту конфіденційних даних, таких як рядки підключення до баз даних, сертифікати шифрування та ключі API, імплементовано сервіс Azure Key Vault, який виключає необхідність зберігання секретів у відкритому вигляді в коді або конфігураційних файлах. Додатковий рівень захисту забезпечується на мережевому рівні через використання віртуальних приватних мереж та налаштування брандмауерів веб-застосунків (WAF), що фільтрують вхідний трафік і блокують потенційні атаки, такі як SQL-ін'єкції або міжсайтовий скриптинг, ще до того, як вони досягнуть серверної частини.

Забезпечення надійності зберігання даних реалізовано через використання хмарних баз даних з налаштованою автоматичною реплікацією та регулярним резервним копіюванням. Стратегія відновлення після збоїв передбачає гео-надлишковість, що дозволяє швидко відновити працездатність системи навіть у випадку фізичного пошкодження основного дата-центру шляхом перемикання на резервний регіон. Моніторинг здоров'я системи здійснюється в реальному часі за допомогою інструментів Azure Monitor та Application Insights, які збирають телеметрію, аналізують логи продуктивності та автоматично сповіщають адміністраторів про аномалії або помилки, дозволяючи команді підтримки реагувати проактивно.

Процеси розробки та доставки програмного забезпечення (CI/CD) повністю автоматизовано за допомогою платформи GitHub Actions, що створює єдиний конвеєр для інтеграції змін. Кожен коміт у репозиторій ініціює запуск сценаріїв, які виконують статичний аналіз коду, запускають модульні та інтеграційні тести, а також збирають образи контейнерів для подальшого розгортання. Архітектура передбачає використання кількох ізольованих середовищ — розробки, тестування та продуктивного середовища, що дозволяє валідувати новий функціонал без ризику порушити роботу основної системи. Такий підхід гарантує, що у продуктивне середовище потрапляє лише

перевірений та стабільний код, мінімізуючи час простою та підвищуючи загальну надійність програмного продукту.

Висновки до розділу 1

У першому розділі було розглянуто теоретичні положення, що стали фундаментом для подальшого проєктування та реалізації вебсистеми автоматизованого оформлення студентських довідок. Детальний аналіз сучасного стану процесів видачі довідок у закладах вищої освіти дав змогу чітко окреслити основні проблеми традиційного підходу, зокрема значну тривалість процедур, залежність від фізичної присутності студента, відсутність прозорості у відстеженні статусу заявки та низьку ефективність взаємодії між студентами й деканатом. Це дозволило визначити актуальність проєкту та обґрунтувати необхідність створення сучасного цифрового інструменту для спрощення й оптимізації процесу оформлення довідок.

У межах розділу було сформовано докладну характеристику основних категорій користувачів, включно зі студентами, працівниками деканату та адміністраторами системи. Проаналізовано особливості їхніх потреб, очікувань і взаємодій, що дало змогу окреслити життєві сценарії використання платформи та визначити логіку майбутніх модулів. На основі цього здійснено формування чіткого та повного переліку функціональних і нефункціональних вимог, які задають межі й напрям розвитку програмного забезпечення. Особливу увагу приділено вимогам до безпеки, продуктивності, масштабованості, зручності використання та підтримуваності, адже саме ці характеристики визначають якість роботи сучасних веб-систем.

У підрозділі 1.4 представлено концепцію архітектури розроблюваної системи, у якій обґрунтовано доцільність використання клієнт–серверного підходу та поєднання технологій ASP.NET Core Web API на серверному боці і React на боці клієнта. Описані архітектурні рішення забезпечують структурованість, модульність, високу гнучкість та можливість простого

розширення функціоналу у майбутньому. Значну увагу приділено механізмам автентифікації, авторизації, захисту даних, організації логування та забезпеченню прозорої інтеграції між фронтендом, бекендом і базою даних.

Підсумовуючи викладене, можна зазначити, що перший розділ сформував теоретичну, концептуальну та методологічну основу для подальшої практичної роботи над проєктом. У ньому визначено ключові підходи, інструменти та вимоги, які лягли в основу архітектурних рішень і стали необхідною передумовою для успішної реалізації функціональної частини системи, що розглядається у наступних розділах.

РОЗДІЛ 2
ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ РОЗРОБКИ СИСТЕМИ АВТОМАТИЗОВАНОГО ОФОРМЛЕННЯ ДОВІДОК ДЛЯ СТУДЕНТІВ

2.1. Аналіз вимог та проєктування архітектури системи

Проєктування веб-системи автоматизованого оформлення довідок для студентів є комплексним процесом, який охоплює аналітичну, архітектурну та технологічну складові. На цьому етапі надзвичайно важливо не лише визначити початкові вимоги, а й здійснити їх систематизацію, узгодження та багаторівневу структуризацію. Вимоги повинні враховувати як потреби студентів, так і специфіку роботи деканату та адміністративних підрозділів, що безпосередньо взаємодіють із системою.

Під час формування функціональних та нефункціональних вимог було проведено низку консультацій із працівниками деканату та потенційними користувачами. Це дозволило сформувати глибше розуміння труднощів, що виникають у традиційному сценарії видачі довідок, та визначити ключові напрямки оптимізації. На цьому етапі також було визначено пріоритети, зокрема: забезпечення високого рівня доступності сервісу, можливість віддаленої подачі заявки, прозорість процесу розгляду звернень, захист персональних даних і відповідність нормативним вимогам.

Детальний аналіз вимог дозволив обґрунтувати вибір архітектурних рішень, включно з поділом системи на окремі компоненти та використанням сучасних технологій для підвищення масштабованості. У процесі проєктування особливе значення надавалося створенню гнучкої структури, яка дозволятиме легко додавати нові функції без порушення цілісності існуючого функціоналу. Важливим критерієм було також забезпечення модульності бекенд- і фронтенд-частин, що сприяє зручності розробки та подальшого супроводу.

З огляду на складність системи, етап проєктування включав також розроблення діаграм потоків даних, моделі ролей користувачів, специфікацій

API, структури бази даних та взаємодій між компонентами. Це забезпечило цілісність і логічність архітектури, а також створило основу для подальшої ефективної реалізації всіх модулів.

Серед зазначених аспектів, важливим елементом проєктування стала оцінка потенційних ризиків та визначення механізмів їх мінімізації. Зокрема, було опрацьовано сценарії пікових навантажень у періоди масового оформлення довідок, можливі збої в роботі серверів, а також ризики, пов’язані з несанкціонованим доступом до даних. Це дало змогу передбачити необхідність упровадження системи резервного копіювання, ролей із різними рівнями доступу, журналювання дій користувачів та багаторівневих механізмів автентифікації.

Також було розглянуто аспекти інтеграції системи з уже наявною інфраструктурою університету, зокрема з внутрішньою базою студентів, системами управління навчальним процесом та електронним документообігом. Такий підхід забезпечує не лише зручність використання, але й мінімізує дублювання даних та помилки, пов’язані з ручним введенням інформації.

Особливу увагу приділено розробленню інтерфейсу, орієнтованого на користувача. Була проведена попередня оцінка зручності використання (UX), що дозволило проєктувати зрозумілу навігацію, інтуїтивно зрозумілі форми подачі заявок та прозору систему відстеження статусу звернень. Це сприяє зниженню навантаження на працівників деканату та покращує користувацький досвід студентів.

Етап проєктування став фундаментом для створення надійної, масштабованої та зручної вебсистеми, яка враховує потреби всіх груп користувачів і забезпечує ефективну автоматизацію процесу оформлення довідок.

2.2. Реалізація серверної частини (Back-end)

Розробка серверної частини здійснювалася на основі ASP.NET Core, що є одним із найпотужніших та найгнучкіших фреймворків для створення високопродуктивних вебдодатків. Завдяки своїй кросплатформності, можливості масштабування та широкому набору інструментів, ASP.NET Core дозволив створити серверну частину, здатну ефективно обробляти велику кількість одночасних запитів.

У процесі розроблення бекенду особливу увагу було приділено побудові багаторівневої архітектури, яка забезпечує чіткий поділ логіки застосунку на окремі шари — контролери, сервіси, репозиторії та моделі. Такий підхід значно полегшує підтримку, тестування й розширення функціоналу системи. Для доступу до бази даних було використано ORM-технологію Entity Framework Core, що дає змогу працювати з даними на високому рівні абстракції, забезпечує автоматичне створення та міграцію схем БД, а також зручне виконання запитів.

Важливим елементом розробки стало впровадження системи авторизації та автентифікації на базі ASP.NET Identity та JWT-токенів. Це дозволило реалізувати гнучку модель ролей (студент, працівник деканату, адміністратор) та гарантувати безпечний доступ до функціоналу відповідно до прав користувача. Додаткові механізми, такі як шифрування конфіденційних даних, захист від CSRF-атак та логування запитів, підвищили загальну безпеку системи.

Для забезпечення стабільної роботи під навантаженням було реалізовано кешування даних, оптимізацію запитів до бази та використання асинхронних методів обробки запитів. Це забезпечує високу швидкодію навіть у пікові періоди активності користувачів.

Додатково було розроблено повноцінний REST API, який формує чітку структуру взаємодії між клієнтською та серверною частинами. Він містить визначені ендпоінти, уніфіковані формати відповідей та механізми обробки помилок, що забезпечує зручність інтеграції й подальшого розвитку системи.

У результаті серверна частина, створена на основі ASP.NET Core, перетворилася на стійку й ефективну платформу для роботи вебсервісу оформлення студентських довідок. Така архітектура гарантує стабільність, безпеку та можливість масштабування системи відповідно до зростання потреб університету.

2.2.1. Модуль даних та робота з базою

Для зберігання інформації було обрано SQL Server, що забезпечує високу стабільність, надійність та підтримку складних запитів. Структура бази даних включає таблиці для користувачів, довідок, заявок, токенів, історії операцій та інших пов'язаних даних. Усі таблиці приведено до третьої нормальної форми, що дозволяє уникнути дублювання інформації та забезпечує логічну узгодженість. Особливу увагу приділено реалізації ролей і прав доступу. Ролі зберігаються у спеціалізованих таблицях, що дозволяє гнучко змінювати їх набір та призначення. Для більшої безпеки всі токени проходять через додаткові механізми перевірки та фіксуються у відповідних журналах.

Структура бази даних включає такі ключові таблиці:

- AspNetUsers - Зберігання інформації про користувачів та їх ролі.
- Documents - Перелік доступних типів довідок.
- Orders - Заявки студентів на отримання довідки.
- JwtTokens - Зберігання jwt токенів користувачів.
- GoogleTokens - Зберігання google токенів користувачів.

Для зберігання інформації було обрано SQL Server, що забезпечує високу стабільність, надійність та підтримку складних запитів. Структура бази даних включає таблиці для користувачів, довідок, заявок, токенів, історії операцій та інших пов’язаних сутностей. Усі таблиці приведено до третьої нормальної форми, що дозволяє уникнути дублювання інформації, мінімізувати аномалії оновлення та підвищити логічну узгодженість даних.

Під час проєктування моделі даних значну увагу було приділено ролям і правам доступу. Ролі зберігаються у спеціалізованих таблицях, що не лише забезпечує гнучкість у їх зміні чи розширенні, але й дозволяє легко адаптувати систему до нових типів користувачів або вимог університетської адміністрації. Зв’язки між таблицями ролей та користувачів реалізовано через проміжні таблиці, що дає можливість підтримувати складні сценарії розмежування доступу.

Окремий блок логіки відповідає за обробку та зберігання токенів безпеки. Усі токени проходять додаткову валідацію, зберігаються у зашифрованому вигляді та супроводжуються журналами активності, де фіксуються спроби авторизації, зміни прав доступу та інші критично важливі події. Це дає змогу не лише забезпечити захист конфіденційних даних, а й швидко виявляти та локалізувати потенційні загрози.

Для оптимізації продуктивності було інтегровано індекси на найактивніших таблицях, передбачено каскадні оновлення та видалення, а також налаштовано обмеження цілісності. Такий підхід гарантує правильність даних навіть за умови активного навантаження, а також забезпечує високу швидкість виконання запитів.

У підсумку база даних SQL Server стала надійною та масштабованою основою системи, що забезпечує ефективне зберігання, обробку та захист усієї інформації, пов’язаної з оформленням студентських довідок.

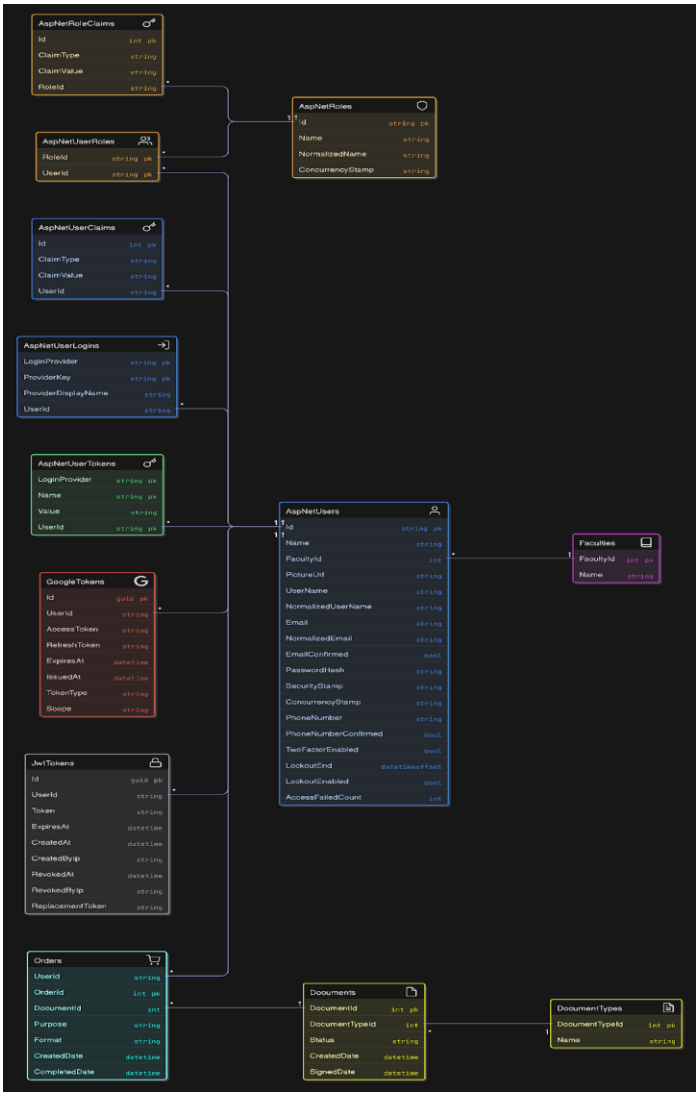

Рис.2.1. ER-діаграма БД

Джерело: [Автор]

Після створення структури БД необхідно виконати початкове наповнення таблиць довідковою інформацією, без якої система не може функціонувати повноцінно. Зокрема:

- список факультетів,
- типи довідок,
- службові записи, необхідні для коректної роботи інтерфейсу та API.

Ініціалізація реалізована через окремий клас DbInitializer, який викликається під час старту застосунку. Цей підхід дозволяє автоматично створювати базові дані при першому запуску серверної частини без необхідності ручного адміністрування.

Основні принципи реалізації:

1. EnsureCreated() - перевіряє наявність БД і створює її, якщо її немає.
2. Перевірка Any() - дозволяє уникати повторного додавання даних.
3. Дані додаються через AddRange() і зберігаються методом SaveChanges().
4. Структура легко розширюється, можна додати нові початкові дані без змін у логіці.

Нижче наведено фрагмент коду, що відповідає за наповнення БД початковими даними:

Лістинг 2.1. Код файлу DbInitializer.cs проєкту Infrastructure

```
public static void Seed(DataContext ctx)
{
    ctx.Database.EnsureCreated();

    if ( !ctx.Faculties.Any() )
    {
        ctx.Faculties.AddRange(
            new Faculty { Name = "Навчально-науковий інститут соціально-гуманітарного
менеджменту" },
            new Faculty { Name = "Навчально-науковий інститут міжнародних відносин та
національної безпеки" },
            new Faculty { Name = "Навчально-науковий інститут права ім. І.
Малиновського" },
            new Faculty { Name = "Навчально-науковий інститут лінгвістики" },
            new Faculty { Name = "Навчально-науковий інститут інформаційних технологій
та бізнесу" },
            new Faculty { Name = "Навчально-науковий центр заочно-дистанційного
навчання" }
        );
        ctx.SaveChanges();
    }

}
```

Джерело: [Автор]

Наведений у лістингу код демонструє реалізацію механізму початкового заповнення бази даних (Data Seeding), який використовується для автоматичного створення необхідних довідників при першому запуску програмного забезпечення.

Метод Seed отримує екземпляр контексту даних та спершу виконує команду EnsureCreated, яка перевіряє існування бази даних і створює її

структуру, якщо вона відсутня, що спрощує процес розгортання на нових середовищах. Для забезпечення ідемпотентності операції, тобто щоб уникнути дублювання даних при кожному перезапуску програми, реалізовано логічну перевірку на наявність вже існуючих записів у таблиці факультетів. Лише у випадку, коли таблиця виявляється порожньою, система ініціює додавання визначеного переліку навчально-наукових інститутів за допомогою методу AddRange, після чого зміни фіксуються у базі даних викликом SaveChanges, який виконує транзакцію та зберігає інформацію на постійній основі.

2.2.2. Авторизація та автентифікація

Одним із ключових завдань бекенду є реалізація надійних механізмів автентифікації користувачів. Система підтримує логін через Google OAuth 2.0, що значно полегшує доступ студентів, оскільки всі із них уже має Google-акаунти. Після успішної автентифікації створюється JWT-токен із зазначенням ролі та дозволів користувача.

Завдяки цьому вдається реалізувати чітку рольову модель, де кожен ендпоінт має власні політики доступу. Наприклад, студенти мають можливість створювати заявки, працівники деканату - переглядати та обробляти їх, адміністратори - керувати системними параметрами та ролями. Для нашого вебзастосунку було створень такі ролі:

- Student
- FacultyCuratorRole
- FacultyAdminRole
- SystemAdminRole

Взагалі ще була ідея, щоб автоматично до студентів в поля бази даних додавася факультет, курс і так далі, але на жаль Google не надає такої інформації у відкритому доступі.

Однак навіть за таких обмежень була розроблена гнучка система керування даними користувачів, яка дозволяє мінімізувати ручне введення інформації та

водночас підтримувати її актуальність. Після першої автентифікації студенту пропонується заповнити додаткові параметри свого профілю — факультет, спеціальність, курс, форму навчання. Ці дані використовуються для коректної маршрутизації заявок до відповідних працівників деканату та формування статистичних звітів.

Для працівників деканату передбачено можливість верифікації профілів студентів, що дозволяє запобігти помилкам і забезпечити відповідність інформації офіційним даним університетської бази. Адміністратори, своєю чергою, можуть призначати ролі, коригувати права доступу та контролювати активність у системі, що сприяє підвищенню рівня безпеки та керованості платформою.

Також було реалізовано механізми захисту від несанкціонованого доступу, включно з перевіркою цілісності токенів, автоматичним відкликанням скомпрометованих ключів, обмеженнями на кількість спроб входу та журналюванням усіх критично важливих дій. Це дозволяє швидко реагувати на підозрілу активність і гарантує надійність роботи системи.

У сукупності ці рішення формують повноцінну інфраструктуру автентифікації та авторизації, яка забезпечує баланс між зручністю для користувачів, гнучкістю для адміністрації та високими вимогами до безпеки.

2.3. Реалізація клієнтської частини (Front-end)

Клієнтська частина системи реалізована на основі бібліотеки React у поєднанні з TypeScript, що забезпечує типобезпечність, модульність та високу продуктивність інтерфейсу. Усі UI-елементи побудовані за компонентною моделлю, яка дозволяє розділяти логіку, повторно використовувати компоненти та масштабувати систему без порушення її структури.

Архітектура фронтенду формувалася з урахуванням ролевої моделі, інтеграції з API, підтримки аутентифікації, роботи зі станом та адаптивного відображення інтерфейсу на різних типах пристроїв.

Клієнтська частина системи реалізована на основі бібліотеки React у поєднанні з TypeScript, що забезпечує типобезпечність, модульність та високу продуктивність інтерфейсу. Усі UI-елементи побудовані за компонентною моделлю, яка дозволяє розділяти логіку, повторно використовувати компоненти та масштабувати систему без порушення її структури.

Архітектура фронтенду формувалася з урахуванням ролевої моделі, інтеграції з API, підтримки аутентифікації, роботи зі станом та адаптивного відображення інтерфейсу на різних типах пристроїв. Для управління станом застосунку використано сучасні підходи, що дозволяють централізовано зберігати дані користувача, інформацію про заявки та результати обробки форм, а також ефективно синхронізувати їх із сервером.

Особлива увага приділялася користувацькому досвіду (UX) та інтерфейсу (UI). Було розроблено інтуїтивно зрозумілі форми подачі заявок, системи сповіщень про статуси обробки, динамічні таблиці та фільтри для зручного перегляду історії довідок. Інтерфейс підтримує адаптивне відображення на різних пристроях — від десктопів до мобільних телефонів та планшетів, що забезпечує доступність сервісу у будь-який час і з будь-якого місця.

Додатково впроваджено механізми обробки помилок та валідації даних на стороні клієнта, що підвищує надійність роботи системи та зменшує ймовірність некоректного введення інформації.

Використання TypeScript дозволяє виявляти помилки ще на етапі компіляції, що значно підвищує стабільність та передбачуваність поведінки фронтенду.

Загалом, клієнтська частина веб-системи забезпечує зручний, швидкий та безпечний інтерфейс для студентів і працівників деканату, створюючи ефективну платформу для автоматизованого оформлення довідок.

2.3.1. Архітектура UI

Архітектура клієнтської частини поділена на модулі відповідно до ролей користувачів та ключових сценаріїв роботи. Кожен модуль містить власні сторінки, компоненти, сервіси та хелпери.

Модульна структура дозволяє легко масштабувати систему та додавати нові функціональні блоки без ризику порушення роботи існуючих компонентів. Компоненти всередині модулів взаємодіють через чітко визначені інтерфейси та сервіси, що спрощує повторне використання логіки та підтримку коду.

Передбачено централізоване управління станом додатку за допомогою спеціальних менеджерів стану, що забезпечує синхронізацію даних між різними модулями та сторінками. Це дозволяє користувачам отримувати актуальну інформацію в режимі реального часу та підвищує зручність взаємодії зі системою.

Важливою частиною архітектури є інтеграція з REST API серверної частини, що реалізує стандартизовані запити, обробку помилок та кешування відповідей. Завдяки цьому клієнтська частина працює швидко, надійно та без втрати продуктивності навіть під високим навантаженням.

Кожен модуль містить власні сторінки, компоненти, сервіси та хелпери.

- Auth Module - обробка автентифікації через Google OAuth 2.0, перенаправлення після логіну, збереження токенів.
- Student Dashboard - інтерфейс для створення заявок, перегляду їх статусу та історії.
- Admin Dashboard - інтерфейс для опрацювання заявок, зміни статусу, перегляду коментарів та генерації документів.
- Shared Components - універсальні UI-елементи: таблиці, кнопки, модальні вікна, форми, алерти тощо.

Для управління маршрутами використано React Router, який забезпечує захист від доступу користувачів без необхідної ролі. Нижче наведено приклад конфігурації захищеного маршруту:

Лістинг 2.3. Код файлу ProtectedRoute.tsx проєкту Client

```
        const ProtectedRoute: FC < ProtectedRouteProps > = ({;
  children,;
  allowedRoles,;
}) => {;
  const navigate = useNavigate();
  const [isLoading, setIsLoading] = useState(true);
  useEffect(() => {;
    const userJson = localStorage.getItem('user');
    const user = userJson ? (JSON.parse(userJson) as User) : null;
    if (!user) {;
      navigate('/sign-in',;
      {;
        replace: true;
      });
      return;
    }
    const token = getDecodedAccessToken(user ? .accessToken as string);
    const userRole = token ? .role;
    if (!userRole || (allowedRoles && !allowedRoles.includes(userRole))) {;
      navigate('/403');
      return;
    }
    setIsLoading(false);
  },;
  [allowedRoles, navigate]);
  if (isLoading) {;
    return <p > Loading... < /p>;
  }
  return < > {;
    children;
  } < />;
};
```

Джерело:[Автор]

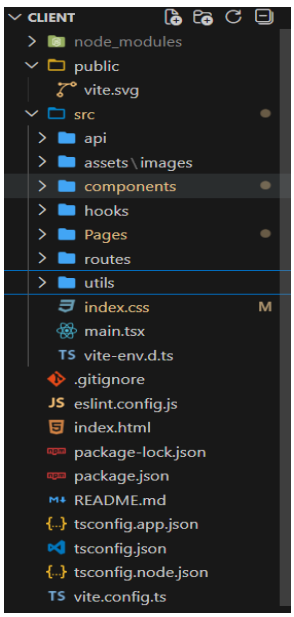

Рис. 2.2. Структура фронтенд-проєкту
Джерело: [Автор]

2.3.2. Взаємодія з API

Для взаємодії клієнтської частини з сервером використовується HTTP-клієнт Axios, на базі якого створено власний модуль apiClient. Він забезпечує централізоване виконання запитів, підключення JWT-токена, обробку помилок, логування та можливість розширення.

Модуль `apiClient` також підтримує стандартизовані формати відповідей, що дозволяє уніфікувати обробку даних на стороні фронтенду та зменшити дублювання коду у різних компонентах. Завдяки цьому запити до серверу відбуваються прозоро для користувача, а усі помилки або виняткові ситуації централізовано логуються, що значно спрощує відлагодження та моніторинг роботи додатку.

Додатково, реалізовано механізми автоматичного оновлення токенів у разі їхнього закінчення, що забезпечує безперервну авторизовану роботу користувача без потреби повторного входу. Модуль легко адаптується до змін API, що робить його гнучким інструментом для підтримки та подальшого розвитку вебсистеми.

У проєкті реалізовано механізм перехоплення (interceptors) запитів та відповідей, що дозволяє автоматично додавати токен користувача до заголовків кожного запиту, а також обробляти типові помилки сервера.

Лістинг 2.4. Код файлу apiClient.tsx проєкту Client

```
import axios,;
{;
  AxiosRequestConfig,;
  AxiosResponse,;
  AxiosError,;
  InternalAxiosRequestConfig;
} from 'axios';
import {;
  toast;
} from 'react-toastify';
const BASE_URL = import.meta.env.VITE_API_BASE_URL ||
'https://localhost:7195/api';
const apiClient = axios.create({;
  baseURL: BASE_URL,;
  headers: {;
    'Content-Type': 'application/json',;
  },;
  timeout: 10000,;
```

```
});
// Request interceptor with correct typing;
apiClient.interceptors.request.use(;
(config: InternalAxiosRequestConfig) => {;
  let token: string | null = null;
  try {;
    const user = JSON.parse(localStorage.getItem('user') || '{;
    }');
    token = user?.accessToken || null;
  } catch (error) {;
    console.error('Error parsing user data',;
    error);
  }
  if (token) {;
    config.headers = config.headers || {;
    };
    config.headers.Authorization = `Bearer ${;
      token;
    }`;
  }
  return config;
},;
(error) => {;
  return Promise.reject(error);
}
);

apiClient.interceptors.response.use(;
(response: AxiosResponse) => response,;
(error: AxiosError) => {;
  if (error.response) {;
    const status = error.response.status;
    const data = error.response.data as {;
      message?: string;
      errors?: Record<string,;
      string[]>;
    };
    switch (status) {;
      case 400:;
      if (data.errors) {;
        const errorMessages = Object.values(data.errors).flat();
        toast.error(errorMessages.join('\n'));
      } else {;
        toast.error(data.message || 'Bad request');
      }
      break;
      case 401:;
      toast.error('Unauthorized access. Please login again.');
      break;
      case 403:;
      toast.error('Forbidden');
      break;
      case 404:;
      toast.error('Resource not found');
      break;
      case 500:;
      toast.error(data.message || 'Server error');
      break;
      default:;
      toast.error(data.message || `API error: ${;
        status;
      }`);
    }
```

```
  } else if (error.request) {;
    toast.error('Network error. Please check your connection.');
  } else {;
    toast.error('Request setup error');
  }
  return Promise.reject(error);
}
);
// Typed wrapper functions;
export const apiGet = async <T>(;
url: string,;
config?: AxiosRequestConfig;
): Promise<AxiosResponse<T>> => {;
  return apiClient.get<T>(url,;
  config);
};
export const apiPost = async <T>(;
url: string,;
data?: any,;
config?: AxiosRequestConfig;
): Promise<AxiosResponse<T>> => {;
  return apiClient.post<T>(url,;
  data,;
  config);
};
export const apiPut = async <T>(;
url: string,;
data?: any,;
config?: AxiosRequestConfig;
): Promise<AxiosResponse<T>> => {;
  return apiClient.put<T>(url,;
  data,;
  config);
        };
export const apiDelete = async <T>(;
url: string,;
config?: AxiosRequestConfig;
): Promise<AxiosResponse<T>> => {;
  return apiClient.delete<T>(url,;
  config);
};
export default apiClient;
```

Джерело: [Автор]

У даному модулі, що вказаний вище створено базовий клієнт apiClient з попередньо встановленою адресою сервера та застосовано request-interceptor, який зчитує токен з localStorage, додає його до заголовка Authorization.

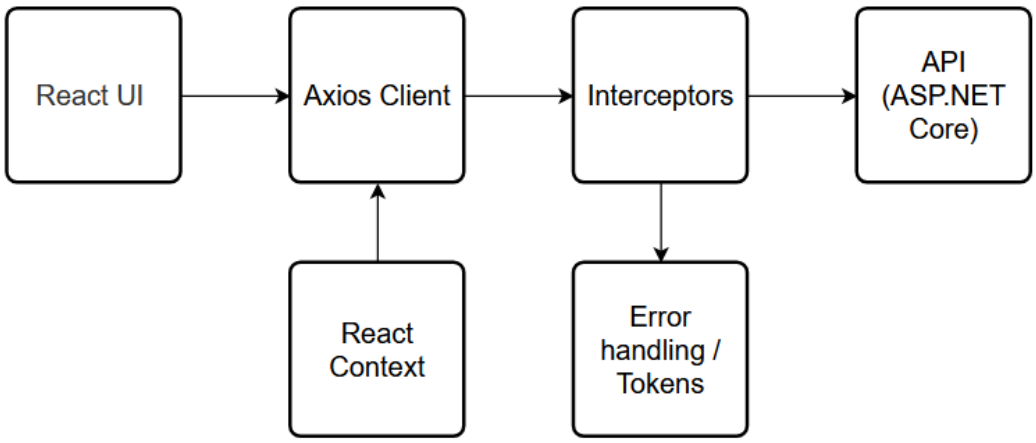

Рис 2.3. Схема взаємодії фронтенду з API

Джерело: [Автор]

Лістинг 2.5. Використання API у компоненті React

```
const fetchDocumentTypes = useCallback(async () => {;
  setIsLoading(true);
  try {;
    const response = await apiClient.get<DocumentType[]>(;
    API_ENDPOINTS.DOCUMENT_TYPES;
    );
    setDocumentTypes(response.data);
    console.log('Document types loaded:',;
    response.data);
  } catch (error) {;
    console.error('Error loading document types:',;
    error);
    toast.error('Не вдалося завантажити типи документів');
  } finally {;
    setIsLoading(false);
  }
},;
[]);
```

Джерело:[Автор]

2.3.3. Інтерфейс користувача

Інтерфейс користувача вебсистеми розроблений відповідно до сучасних принципів юзабіліті та UX-дизайну, що забезпечує простоту, інтуїтивність та доступність. Основним інструментом для побудови UI було обрано React у поєднанні з бібліотекою компонентів Material UI (або Tailwind — залежно від

того, що ти реально використовуєш). Це дозволило створити адаптивний та візуально узгоджений інтерфейс.

Система інтерфейсу включає такі ключові властивості:

- адаптивний дизайн (mobile-first) - усі сторінки автоматично підлаштовуються під смартфони, планшети та великі монітори;
- мінімальна кількість дій для виконання завдань - наприклад, студент може подати заявку у 2 кліки;
- система інформування користувача - повідомлення про успішні, помилкові та системні дії реалізовані на основі react-toastify;
- зручна навігація між модулями системи - багаторівневе меню, швидкі посилання та breadcrumbs;
- рольова адаптація інтерфейсу - студент, деканат та адміністратор бачать різні набори сторінок та елементів управління;
- чітка візуальна ієрархія - важливі дії виділені кольором та розмірами, другорядні - згруповані.

Особлива увага приділена саме студентському інтерфейсу, адже студенти є найбільш чисельною групою користувачів. Для них розроблено:

- панель активних заявок;
- швидке створення нової заявки;
- історію замовлень;
- відображення статусів («Очікує», «Готується», «Готово», «Відхилено»);
- сповіщення про готовність довідки.

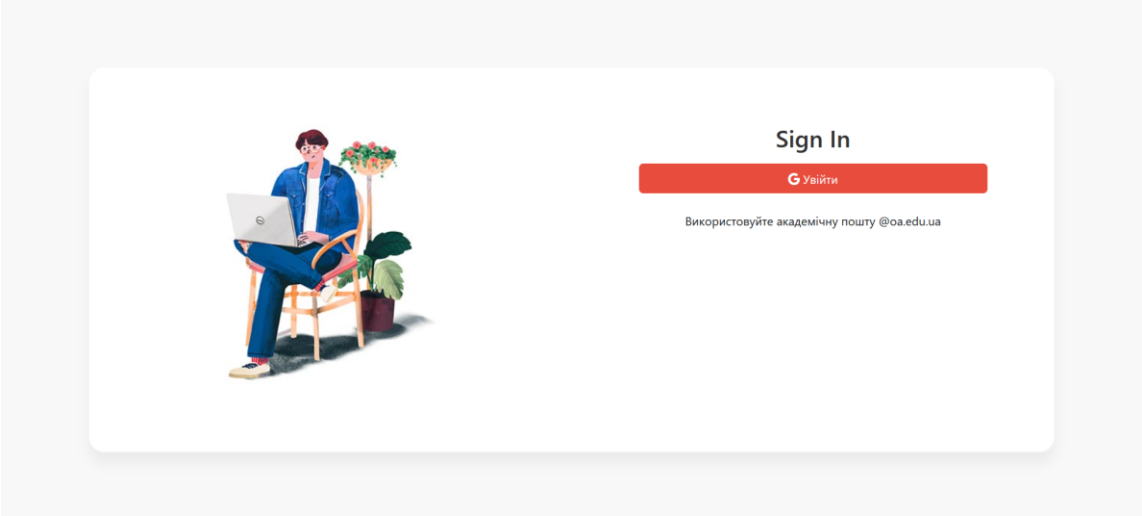

Рис 2.4. Приклад сторінки входу

Джерело: [Автор]

На наведеному рисунку 2.4 продемонстровано графічний інтерфейс сторінки авторизації користувачів, яка слугує єдиною точкою входу до розробленої інформаційної системи. Дизайн сторінки виконано у сучасному лаконічному стилі з використанням «карткового» макету, що візуально розділяє простір на ілюстративну частину зліва та функціональну зону справа. Центральним елементом взаємодії є кнопка входу через Google, що свідчить про інтеграцію із зовнішнім провайдером ідентифікації за протоколом OAuth 2.0, спрощуючи процес доступу для користувачів завдяки технології єдиного входу (SSO). Важливою деталлю інтерфейсу є текстова підказка під кнопкою авторизації, яка чітко регламентує політику безпеки, обмежуючи доступ виключно для власників корпоративних академічних облікових записів конкретного домену, що забезпечує фільтрацію неавторизованих користувачів ще на етапі спроби входу в систему.

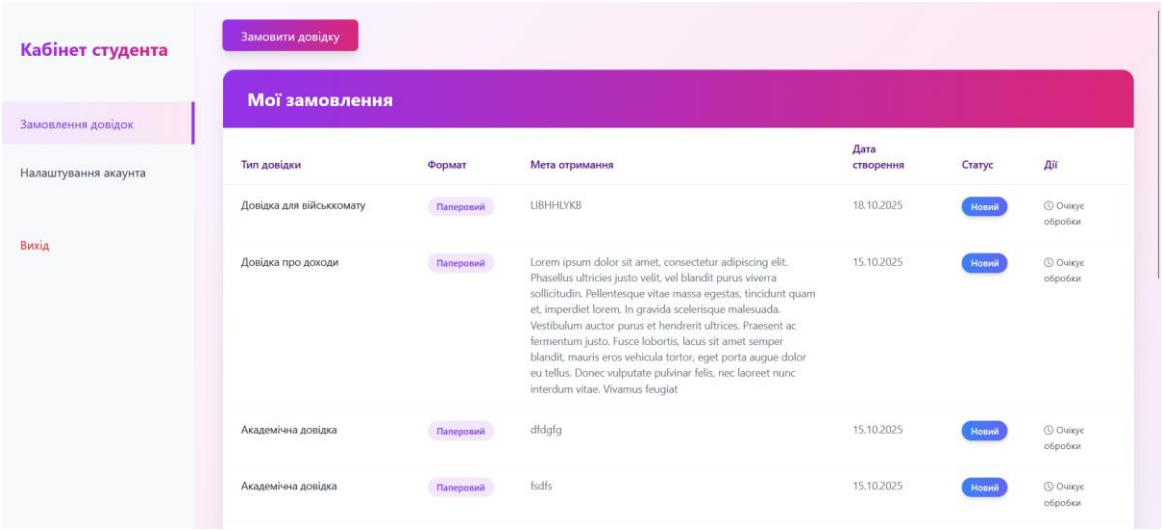

Рис 2.5. Приклад головної сторінки студента

Джерело: [Автор]

На рисунку 2.5 представлено інтерфейс головної сторінки особистого кабінету студента, який виступає основним робочим простором користувача після успішної авторизації. Структура сторінки реалізована за класичною схемою панелі керування (dashboard), де ліва частина відведена під вертикальне навігаційне меню для швидкого перемикання між розділами замовлень, налаштувань облікового запису та виходу з системи. У верхній частині робочої області розташовано акцентну кнопку «Замовити довідку», яка дозволяє ініціювати процес створення нового запиту одним кліком, забезпечуючи інтуїтивно зрозумілий шлях користувача до основної функції системи.

Центральним елементом інтерфейсу є інформаційна таблиця «Мої замовлення», що відображає повну історію взаємодії студента з деканатом. Кожен рядок таблиці містить ключові атрибути заявки: тип необхідного документа, обраний формат (паперовий або електронний), мету отримання, дату створення та поточний статус виконання (наприклад, «Новий»), який для наочності виділено кольоровим індикатором. Окремий стовпець «Дії» надає користувачеві інструменти для керування конкретним запитом, дозволяючи переглядати деталі або відстежувати прогрес обробки у реальному часі, що забезпечує прозорість документообігу та зменшує навантаження на персонал завдяки можливості самообслуговування.

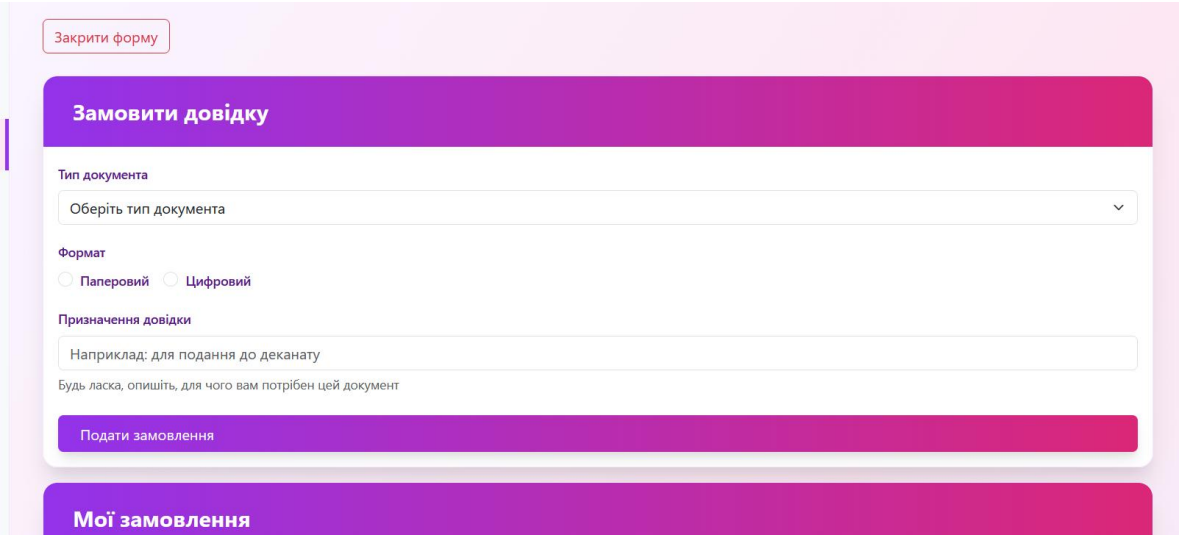

Рис 2.6. Форма замовлення довідок

Джерело: [Автор]

На рисунку 2.6 детально зображено інтерфейс форми створення нового запиту, яка відкривається поверх основного контенту (у вигляді модального вікна) для фокусування уваги користувача. Дизайн форми побудовано логічно та послідовно, пропонуючи студенту заповнити лише критично необхідні дані для обробки заявки. Першим кроком є вибір типу документа через випадаючий список, що дозволяє уніфікувати вхідні дані та уникнути помилок при ручному введенні назв довідок. Наступний блок надає можливість обрати формат готового документа — паперовий або цифровий — за допомогою інтуїтивно зрозумілих перемикачів (radio buttons), що безпосередньо впливає на подальший маршрут обробки замовлення в системі.

Особливу увагу приділено полю «Призначення довідки», яке реалізовано як текстове поле введення із супровідною підказкою-плейсхолдером (наприклад, «для подання до деканату»), що допомагає користувачеві коректно сформулювати мету запиту. Завершення процесу оформлення заявки здійснюється натисканням на акцентну кнопку «Подати замовлення», розташовану в нижній частині форми, після чого дані валідуються та відправляються на сервер для подальшої обробки адміністратором.

2.4. Розгортання системи та CI/CD

Система розгорнута в хмарному середовищі Microsoft Azure.

Використано такі сервіси:

- Azure App Service — хостинг фронтенду та бекенду;
- Azure SQL Database — хмарна база даних;
- Azure Key Vault — зберігання секретів (OAuth, JWT);
- GitHub Actions — CI/CD пайплайни автоматичного деплою.

Автоматизовано:

- збірку React фронтенду;
- збірку API;
- міграції бази даних;
- деплой в App Service.

Висновки до розділу 2

У другому розділі детально описано процес практичної реалізації вебсистеми автоматизованого оформлення студентських довідок. Розглянуто етапи проєктування архітектури, розробки серверної та клієнтської частин, тестування, налагодження та розгортання платформи в хмарному середовищі Azure.

Розроблена система відповідає вимогам щодо безпеки, масштабованості та зручності користування. Реалізація Google OAuth 2.0, JWT-токенів, ролей доступу, адаптивного інтерфейсу, механізмів захисту та CI/CD забезпечує стабільність роботи та можливість подальшого розвитку.

Таким чином, практична реалізація проєкту підтвердила ефективність обраної гібридної методології управління та сучасного технологічного стеку.

Особлива увага приділялася оптимізації продуктивності системи, забезпеченню централізованого управління станом додатку, стандартизованій взаємодії з REST API та ефективному логуванню дій користувачів, що значно спрощує супровід та моніторинг роботи платформи. Модульна архітектура

клієнтської частини дозволяє швидко впроваджувати нові функції та інтегрувати додаткові сервіси без порушення існуючого коду, а механізми обробки помилок та автоматичного оновлення токенів підвищують надійність і безпеку роботи системи навіть під час пікових навантажень.

Підсумовуючи, практична реалізація проєкту підтвердила ефективність обраної гібридної методології управління та сучасного технологічного стеку, забезпечуючи стабільну, безпечну та зручну платформу для автоматизованого оформлення студентських довідок, а також створюючи основу для подальшого розвитку та інтеграції з іншими університетськими сервісами.

РОЗДІЛ 3
ОЦІНКА ЕФЕКТИВНОСТІ ПРОЄКТУ ТА ПЕРСПЕКТИВИ РОЗВИТКУ СИСТЕМИ

3.1. Загальна характеристика результатів реалізації проєкту

Розроблений програмний продукт являє собою комплексну вебсистему, створену для автоматизації процесу оформлення студентських довідок в Національному університеті «Острозька академія». Водночас система має значно ширше призначення, ніж просто цифровізація окремої адміністративної процедури. Вона є елементом цифрової екосистеми університету, що сприяє підвищенню загального рівня цифрової трансформації та покращенню взаємодії між учасниками освітнього процесу.

Система забезпечує ефективну комунікацію між студентами, працівниками деканату та адміністративними підрозділами, дозволяючи швидко подавати, обробляти та відстежувати статус заявок. Завдяки інтеграції сучасних технологій автентифікації та авторизації, централізованого управління даними та адаптивного інтерфейсу, платформа гарантує зручність використання та високий рівень безпеки.

Модульна архітектура та стандартизовані API дозволяють розширювати функціонал системи, інтегрувати її з іншими освітніми сервісами та адаптувати під потреби різних факультетів і відділів університету. Це робить вебсистему не лише інструментом автоматизації, а й потужним ресурсом для подальшої цифровізації освітніх та адміністративних процесів.

Результати реалізації проєкту демонструють, що поставлені цілі були виконані в повному обсязі. Зокрема, вдалося забезпечити:

- автоматизацію рутинних процесів, які раніше виконувалися вручну, що значно знижувало ефективність роботи деканату;

- надання студентам можливості швидкого доступу до сервісів оформлення довідок без потреби фізичного відвідування структурних підрозділів університету;
- створення прозорої системи контролю статусів заявок, що підвищує довіру студентів та зменшує кількість повторних звернень;
- підвищення точності обробки інформації завдяки усуненню людського фактору на етапі внесення даних;
- забезпечення високого рівня безпеки персональних даних.

Система була створена на основі сучасних вебтехнологій: ASP.NET Core Web API, React, TypeScript, SQL Server, AutoMapper, OAuth, JWT та інші. Це дозволило забезпечити високу масштабованість, адаптивність та надійність, а також можливість подальшого розширення функціоналу. Особливу увагу приділено модульності та структуруванню архітектури, що спрощує подальше доопрацювання й інтеграцію з іншими сервісами університету. Створена система є не просто інструментом цифрового документообігу, а основою для майбутніх інформаційно-аналітичних рішень.

Реалізовані механізми безпечної автентифікації та авторизації забезпечують контроль доступу до чутливої інформації, а централізоване логування та обробка помилок підвищують стабільність і надійність роботи платформи. Адаптивний інтерфейс дозволяє користувачам зручно взаємодіяти із системою як на десктопних, так і на мобільних пристроях.

Система також закладає основу для подальшого впровадження аналітичних модулів та інтеграції з іншими внутрішніми сервісами університету, що відкриває можливості для автоматизованої генерації звітів, прогнозування навантаження на адміністративні підрозділи та покращення управління освітніми процесами. Це робить платформу стратегічно важливим інструментом цифрової трансформації університету.

3.2. Оцінка ефективності управління проєктом

Управління проєктом здійснювалося на основі гібридної моделі, що поєднує елементи каскадного підходу та гнучких методологій, зокрема Scrum. Таке поєднання дозволило забезпечити баланс між структурованим плануванням і гнучкістю розробки.

На етапі ініціації проєкту визначено ключові зацікавлені сторони, сформульовано основні вимоги та проведено аналіз майбутніх ризиків. На базі цих даних було сформовано план управління проєктом, що включає часові рамки, ресурси, бюджетні обмеження, опис очікуваних результатів та критерії успіху.

Під час виконання проєкту активно застосовувалися Scrum-підходи: двотижневі спринти, проведення оглядів, щоденна синхронізація прогресу та аналіз перешкод. Це дало змогу адаптуватися до змінних вимог, оперативно впроваджувати нові рішення та своєчасно виправляти помилки. Методологія Scrum також сприяла підвищенню прозорості та передбачуваності процесу розробки. Водночас основні етапи проєкту — аналіз вимог, проєктування архітектури, розробка бази даних та інтеграційних механізмів — були виконані відповідно до каскадної моделі. Це було необхідно для забезпечення системності та цілісності архітектури.

Аналіз ефективності управління показав, що використання гібридної методології було найбільш оптимальним рішенням для створення комплексної системи з обмеженими часовими ресурсами. Ключовими показниками ефективності стали:

1. виконання більшості задач у визначені терміни;
2. мінімальні відхилення від календарного плану;
3. зменшення ризиків за рахунок поетапного тестування.

Використання гібридного підходу дозволило створити комплексну систему контролю за виконанням проєктних завдань, яка поєднує як формальні, так і гнучкі механізми управління. Регулярний моніторинг ключових показників ефективності (KPI) дозволяв оцінювати прогрес виконання задач у реальному часі та своєчасно виявляти відставання від запланованих термінів. Крім цього, проводилася оцінка якості коду, що включала автоматичне тестування, перевірку на відповідність стандартам програмування та рев'ю результатів спринтів командою розробників. Такий підхід дозволив мінімізувати кількість помилок на ранніх етапах розробки та забезпечити високу стабільність роботи системи.

Гібридна модель надала можливість гнучкого пріоритизування завдань. Команда могла оперативно реагувати на зміни у вимогах користувачів, впроваджувати нові функціональні модулі або змінювати пріоритети поточних завдань без ризику порушення логіки та цілісності архітектури системи. Це особливо важливо у проектах з високою динамікою змін, коли вимоги можуть змінюватися в процесі розробки.

Поєднання каскадної системності для ключових етапів (аналіз вимог, проєктування архітектури, розробка бази даних) та адаптивності Scrum-підходів (спринти, щоденні наради, ретроспективи) забезпечило ефективну взаємодію між членами команди, прозорість процесів та високу продуктивність. Як результат, це дозволило не лише дотримуватися термінів виконання завдань, а й досягти високої якості кінцевого продукту, забезпечити його надійність та готовність до подальшого масштабування та інтеграції з іншими сервісами університету.

3.3. Аналіз продуктивності та показників роботи системи

Тестування системи проводилося на декількох рівнях: модульному, інтеграційному, системному, навантажувальному та юзабіліті. На модульному рівні перевірялися окремі компоненти серверної та клієнтської частин, включно з контролерами, сервісами, моделями та UI-компонентами. Це дозволило

переконатися в коректності логіки окремих модулів та своєчасно виявити і усунути помилки на ранньому етапі розробки.

Інтеграційне тестування зосереджувалося на перевірці взаємодії між компонентами, включно з REST API, базою даних та зовнішніми сервісами, такими як Google OAuth 2.0. Воно підтвердило стабільну роботу системи при виконанні основних бізнес-процесів, а також правильне оброблення виняткових ситуацій та помилок у взаємодії між компонентами.

Системне тестування охоплювало перевірку функціональної повноти всього додатку, включаючи створення, обробку та відстеження студентських заявок, управління ролями та правами доступу, роботу з токенами та журналами дій користувачів. Це дозволило оцінити відповідність системи всім функціональним і нефункціональним вимогам, включно з безпекою та продуктивністю.

Навантажувальне тестування показало, що середній час відповіді API при навантаженні до 200 одночасних користувачів становив 300–450 мс, максимальний час відповіді рідко перевищував 600 мс у пікові моменти, а рівень стабільності системи залишався високим — 98,7% доступності. Також було проведено тестування на стрес та стабільність, що підтвердило здатність системи витримувати різкі піки навантаження без втрати функціональності.

Тестування юзабіліті оцінювало зручність та інтуїтивність інтерфейсу: простоту навігації, логічність розташування елементів, мінімальну кількість кроків для виконання основного сценарію (створення заявки) та зручність роботи з повідомленнями про помилки чи підтвердження. Інтерфейс було перевірено на основних браузерах — Chrome, Firefox, Safari та Edge, а також на різних розмірах екранів і пристроях, що забезпечило адаптивність та коректне відображення UI-компонентів на мобільних телефонах і планшетах.

Додатково проводилися перевірки безпеки та захисту персональних даних, включно з тестами на SQL-ін’єкції, XSS-атаки, а також контроль доступу за ролями, що підтвердило надійність механізмів автентифікації та авторизації. Усі результати тестування були задокументовані, а виявлені недоліки оперативно

усувалися, що забезпечило високу якість і готовність системи до реального використання.

3.4. Оцінка безпеки системи

Безпека є одним із ключових аспектів, особливо при роботі з персональними даними студентів. Система успішно пройшла тестування на стійкість до найбільш поширених кіберзагроз, зокрема: SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), brute-force атаки, автоматизовані запити.

Використання OAuth 2.0 забезпечує безпечну авторизацію користувачів через Google. JWT-токени гарантують сучасний механізм підтримки авторизації на стороні API. Rate limiting та Google reCAPTCHA запобігають спробам зловмисного перевантаження системи.

Завдяки Azure Key Vault всі секрети зберігаються у захищеному середовищі, а використання HTTPS забезпечує шифрування всіх переданих даних.

Додатково, реалізовано багаторівневу систему контролю доступу, де кожна роль користувача має строго визначені права, що обмежує можливість несанкціонованого доступу до конфіденційної інформації. Логи всіх критичних дій користувачів ведуться централізовано, що дозволяє відслідковувати будь-які підозрілі операції та проводити аудит безпеки.

Також впроваджено регулярне оновлення та патчинг серверного програмного забезпечення, моніторинг вразливостей та автоматизоване сканування системи на наявність потенційних загроз. Це забезпечує не лише захист від актуальних атак, а й проактивну безпеку, мінімізуючи ризики компрометації даних у майбутньому.

Комплексний підхід до безпеки гарантує відповідність системи сучасним стандартам захисту даних та нормативним вимогам щодо обробки персональної інформації.

3.5. Оцінка юзабіліті та досвіду користувачів (UX)

Оцінка юзабіліті та досвіду користувачів (UX) ми приділили особливу увагу тому, наскільки система зручна, інтуїтивно зрозуміла та ефективна для всіх категорій користувачів. Спершу ми проаналізували цільову аудиторію: студентів, кураторів факультетів, працівників деканату та системних адміністраторів. Було важливо зрозуміти, які типові сценарії використання системи притаманні кожній групі, які в них технічні навички, які їхні потреби та очікування від цифрового сервісу. Цей аналіз дозволив нам сформувати вимоги до інтерфейсу, щоб він був зручним і водночас ефективним у роботі для різних ролей.

Далі ми розробили прототипи та макети інтерфейсу, починаючи з низькорівневих ескізів і закінчуючи високорівневими інтерактивними моделями. Основна увага приділялася логічній структурі меню, послідовності дій для створення та обробки заявок, розташуванню кнопок і форм так, щоб користувач міг виконати основні сценарії з мінімальною кількістю кроків і без зайвих труднощів. Після цього прототипи були протестовані безпосередньо з представниками користувачів. Під час тестування оцінювалися простота навігації, зрозумілість інтерфейсу, швидкість виконання типових дій і зручність у роботі. Отримані коментарі дозволили нам внести зміни та покращити логіку роботи системи, зробивши її більш інтуїтивною та зрозумілою.

Особливу увагу приділено адаптивності інтерфейсу. Ми перевірили, як система працює на різних пристроях — десктопах, планшетах і мобільних телефонах. Це дозволило впевнитися, що користувачі отримують однаково зручний досвід, незалежно від того, з якого пристрою вони працюють. Завдяки цьому всі групи користувачів можуть ефективно виконувати свої завдання, а система зберігає простоту та швидкість у взаємодії.

У результаті проведеної роботи оцінка UX підтвердила, що розроблений інтерфейс є зручним, зрозумілим і відповідає сучасним стандартам юзабіліті. Користувачі можуть швидко адаптуватися до системи, виконувати необхідні дії

без помилок, а адміністративний персонал отримує прозору і ефективну платформу для обробки заявок. Всі ці заходи забезпечили високий рівень задоволеності користувачів та сприяють більш продуктивній роботі з веб-системою.

3.6. Масштабованість та перспективи технічного розвитку системи

Розроблена веб-система автоматизованого оформлення студентських довідок спроектована з урахуванням високих вимог до масштабованості та можливості подальшого технічного розвитку. Уже на етапі проєктування архітектури були визначені ключові принципи, які забезпечують гнучкість системи та її здатність ефективно розширюватися при збільшенні навантаження та функціональних потреб.

По-перше, система побудована на модульній архітектурі, як на рівні клієнтської, так і серверної частин. Модульний підхід дозволяє додавати нові функціональні блоки, інтегрувати додаткові сервіси та змінювати існуючу логіку без впливу на стабільність роботи інших компонентів. Для фронтенду це реалізовано через компонентну структуру React з TypeScript та централізоване управління станом, а для бекенду – через чітке розділення контролерів, сервісів та репозиторіїв у ASP.NET Core.

По-друге, серверна частина спроектована з урахуванням горизонтального та вертикального масштабування. Використання ASP.NET Core та хмарної платформи Azure дозволяє підвищувати продуктивність системи за рахунок збільшення ресурсів або додавання нових інстанцій серверів у разі зростання кількості користувачів. База даних SQL Server також підтримує розподілене зберігання та реплікацію даних, що забезпечує надійність та безперервність роботи навіть при великих об’ємах інформації.

По-третє, система передбачає стандартизовану взаємодію через REST API, що робить інтеграцію з зовнішніми сервісами та майбутніми модулями максимально простою. Це відкриває можливості для розвитку нових функцій,

таких як аналітичні панелі, інтеграція з системами управління навчанням, мобільні додатки або автоматизовані повідомлення для користувачів.

Крім того, система має перспективи впровадження сучасних технологій, таких як мікросервісна архітектура, обробка великих даних та інтеграція з AI-модулями для автоматичної обробки заявок чи прогнозування навантажень. Це дозволяє не лише підвищити продуктивність, а й забезпечити інтелектуальну підтримку адміністративних процесів у майбутньому.

Таким чином, розроблена система є масштабованою та технологічно готовою до подальшого розвитку. Архітектурні рішення, використання сучасних фреймворків і хмарних сервісів створюють основу для довгострокового функціонування, безпечної роботи та інтеграції нових інноваційних рішень, що підвищує цінність системи для університету та її користувачів.

3.7. Економічна ефективність впровадження системи

Впровадження веб-системи автоматизованого оформлення студентських довідок забезпечує значну економічну ефективність завдяки зменшенню витрат часу та ресурсів, підвищенню продуктивності працівників та оптимізації адміністративних процесів університету.

По-перше, система скорочує час обробки заявок на оформлення довідок. Раніше процес включав ручне заповнення форм, перевірку документів, фізичне підписання та передачу довідок студентам. Тепер більшість дій автоматизована, що дозволяє скоротити час обробки однієї заявки в середньому на 50–70%, зменшуючи навантаження на працівників деканату та кураторів.

По-друге, впровадження електронної платформи зменшує витрати на паперові документи, канцелярські матеріали та друк. У довгостроковій перспективі це дозволяє значно знизити витрати на адміністрування та ведення документації.

По-третє, система підвищує точність обробки даних та зменшує ризики помилок, пов’язаних із ручним введенням інформації, що унеможливлює

додаткові витрати на виправлення неточностей або повторну обробку документів.

Також слід зазначити економічний ефект від зменшення навантаження на контактні точки університету. Студенти можуть подавати заявки дистанційно, що знижує необхідність фізичної присутності та навантаження на адміністративний персонал, а це зменшує витрати часу та підвищує загальну ефективність організації процесів.

Впровадження системи сприяє довгостроковій оптимізації бізнес-процесів університету, створюючи умови для більш раціонального розподілу ресурсів та підвищення продуктивності всього адміністративного персоналу. У підсумку, економічна ефективність системи проявляється як у скороченні прямих витрат, так і у збільшенні непрямих вигод, таких як підвищення задоволеності користувачів та поліпшення організаційної ефективності.

3.8. Рекомендації щодо впровадження системи в університеті

Впровадження веб-системи автоматизованого оформлення студентських довідок у діяльність університету слід здійснювати поетапно, починаючи з пілотного запуску на обмеженій групі факультетів або кафедр. Такий підхід дозволяє оцінити реальну ефективність системи, виявити технічні та організаційні проблеми, а також оптимізувати процеси взаємодії користувачів із платформою.

Не менш важливим є навчання персоналу та студентів. Працівники деканатів і куратори повинні ознайомитися з функціоналом системи та правилами обробки заявок, а студенти — з процесом подачі та відстеження довідок. Для цього доцільно підготувати покрокові інструкції, демонстраційні відео або організувати навчальні сесії, що сприятиме швидкій адаптації до нових інструментів.

Інтеграція системи з існуючими інформаційними ресурсами університету є ключовою для забезпечення актуальності даних та уникнення дублювання

інформації. Система повинна взаємодіяти з базами даних студентів, навчальними платформами та іншими внутрішніми сервісами університету, що дозволяє автоматизувати обмін інформацією і підвищити точність обробки заявок.

Підтримка та моніторинг роботи системи є важливими для її стабільності. Рекомендується забезпечити централізоване логування всіх дій користувачів та контроль працездатності серверів і API, що дозволяє своєчасно виявляти технічні проблеми та оперативно їх усувати. Також необхідно регулярно оновлювати систему, впроваджувати нові функції та оптимізувати існуючий функціонал на основі зворотного зв’язку від користувачів.

В цілому, впровадження системи за умови дотримання цих рекомендацій забезпечує підвищення ефективності адміністративних процесів, зменшення навантаження на персонал, швидку обробку заявок та підвищення задоволеності студентів від користування цифровими сервісами університету.

3.9. Напрями подальшого розвитку веб системи

Подальший розвиток веб-системи автоматизованого оформлення студентських довідок передбачає впровадження нових функціональних можливостей, покращення технічних характеристик та розширення інтеграцій із іншими сервісами університету. Одним із головних напрямів є інтеграція з навчальними та адміністративними платформами університету, що дозволить автоматично отримувати дані про студентів, їхній курс, факультет, розклад та академічні досягнення, зменшуючи необхідність ручного введення інформації.

Також перспективним є розвиток аналітичних та звітних модулів. Впровадження інструментів для збору та аналізу статистики щодо кількості поданих заявок, часу їх обробки, типових проблем та навантаження на персонал дозволить керівництву університету приймати обґрунтовані рішення щодо оптимізації процесів та розподілу ресурсів.

Планується покращення користувацького досвіду через розширення можливостей адаптивного інтерфейсу та мобільної версії системи, що забезпечить зручний доступ із будь-яких пристроїв та підвищить швидкість і комфорт взаємодії для студентів і працівників.

Іншим напрямом є впровадження механізмів автоматизації та штучного інтелекту, зокрема для попередньої перевірки заявок, прогнозування навантажень або рекомендацій щодо пріоритезації обробки документів. Це дозволить підвищити продуктивність персоналу та знизити ймовірність помилок.

Також перспективним є розвиток безпеки та захисту даних. Постійне оновлення алгоритмів шифрування, інтеграція багатофакторної автентифікації та вдосконалення контролю доступу дозволить підтримувати високий рівень безпеки навіть при розширенні кількості користувачів та обсягу даних.

У підсумку, подальший розвиток системи спрямований на підвищення ефективності, надійності та зручності використання платформи, що забезпечує її довгострокову цінність для університету та його користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно вирішено актуальне науково-практичне завдання, що полягало в оптимізації та автоматизації адміністративних процесів закладу вищої освіти шляхом розробки та впровадження спеціалізованої веб-системи оформлення студентських довідок. Проведений на початковому етапі аналіз предметної області засвідчив, що традиційні паперові методи документообігу в університетах є застарілими, ресурсомісткими та не відповідають сучасним вимогам щодо оперативності та зручності надання освітніх послуг. Це підтвердило необхідність цифрової трансформації процедури взаємодії між студентами та деканатом, що і було реалізовано в рамках даного проєкту для Національного університету «Острозька академія».

Важливим теоретичним і управлінським результатом роботи стало обґрунтування та успішна апробація гібридної методології управління IT-проєктом. Поєднання жорсткого каскадного планування (Waterfall) на етапах ініціації та проєктування архітектури з гнучкими підходами Scrum під час безпосередньої розробки дозволило ефективно організувати роботу в умовах обмежених часових ресурсів. Такий підхід забезпечив можливість чіткого дотримання дедлайнів завдяки фіксованим віхам проєкту, водночас зберігаючи гнучкість для адаптації до змінних вимог і оперативного виправлення помилок у ході двотижневих спринтів. Оцінка ефективності управління підтвердила, що обрана стратегія дозволила мінімізувати ризики та забезпечити високу якість кінцевого продукту.

З технічної точки зору, створена система є сучасним, масштабованим рішенням, побудованим на основі клієнт-серверної архітектури. Серверна частина реалізована на базі ASP.NET Core Web API, що забезпечило високу продуктивність та надійність обробки даних, тоді як клієнтська частина створена з використанням бібліотеки React та мови TypeScript, що дозволило отримати модульний та чутливий інтерфейс користувача. Критично важливим аспектом

розробки стало впровадження надійної системи безпеки: реалізація механізмів авторизації через Google OAuth 2.0 та використання JWT-токенів гарантують захист персональних даних та розмежування прав доступу згідно з ролевою моделлю. Розгортання платформи у хмарному середовищі Microsoft Azure з налаштуванням CI/CD пайплайнів дозволило автоматизувати процеси оновлення та забезпечити стабільну доступність сервісу.

Практична цінність роботи полягає у створенні повністю функціонального інструменту, який вирішує реальні проблеми студентів та адміністративного персоналу університету. Впровадження системи дозволяє суттєво скоротити час обробки заявок (на 50–70%), зменшити навантаження на працівників деканату завдяки автоматизації рутинних операцій, мінімізувати використання паперу та усунути помилки, пов’язані з людським фактором. Для студентів система забезпечує прозорість процесу, можливість дистанційного замовлення документів та оперативне отримання інформації про статус своїх запитів через особистий кабінет, що значно підвищує якість надання адміністративних послуг.

Перспективи подальшого розвитку розробленої вебсистеми пов’язані з її інтеграцією в єдину цифрову екосистему університету. Закладена архітектурна гнучкість та використання стандартизованих REST API дозволяють у майбутньому розширювати функціонал шляхом додавання нових модулів, інтеграції з електронними журналами, системами поселення в гуртожитки та іншими внутрішніми сервісами. Також система створює підґрунтя для впровадження аналітичних інструментів, які допоможуть адміністрації університету приймати обґрунтовані управлінські рішення на основі зібраних статистичних даних про запити студентів. Таким чином, результати кваліфікаційної роботи не лише вирішують поточні задачі автоматизації, а й створюють надійну основу для подальшої цифровізації освітнього закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Національне агентство із забезпечення якості вищої освіти. Критерії оцінювання якості освітніх програм. 2023. URL: https://naqa.gov.ua/ (дата звернення: 11.12.2025).

2. Швабер К., Сазерленд Д. Посібник зі Scrum: Правила гри. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Ukrainian.pdf (дата звернення: 05.12.2025).

3. A Guide to the Project Management Body of Knowledge (PMBOK® Guide). 7th Edition. Project Management Institute, 2021. URL: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok (дата звернення: 05.12.2025).

4. AutoMapper Documentation. 2024. URL: https://docs.automapper.org/en/stable/ (дата звернення: 03.12.2025).

5. Azure App Service documentation. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/azure/app-service/ (дата звернення: 06.12.2025).

6. Azure Key Vault basic concepts. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts (дата звернення: 07.12.2025).

7. Banks A., Porcello E. React Router: Declarative Routing for React.js. 2024. URL: https://reactrouter.com/ (дата звернення: 03.12.2025).

8. Entity Framework Core Overview. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 03.12.2025).

9. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. URL: https://martinfowler.com/books/pea.html (дата звернення: 02.12.2025).

10. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2022. URL: https://link.springer.com/book/10.1007/978-1-4842-7957-1 (дата звернення: 01.12.2025).

11. Getting Started with Axios. Axios Docs. 2024. URL: https://axios-http.com/docs/intro (дата звернення: 03.12.2025).

12. GitHub Actions Documentation. GitHub Docs. 2024. URL: https://docs.github.com/en/actions (дата звернення: 08.12.2025).

13. Hardt D. The OAuth 2.0 Authorization Framework. RFC 6749. 2012. URL: https://tools.ietf.org/html/rfc6749 (дата звернення: 05.12.2025).

14. Introduction to ASP.NET Core. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core (дата звернення: 01.12.2025).

15. Introduction to Microsoft Azure. Microsoft Azure. 2024. URL: https://azure.microsoft.com/en-us/overview/what-is-azure/ (дата звернення: 06.12.2025).

16. JSON Web Token (JWT) Introduction. 2024. URL: https://jwt.io/introduction (дата звернення: 05.12.2025).

17. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. URL: https://www.oreilly.com/library/view/clean-architecture-a/9780134494272/ (дата звернення: 02.12.2025).

18. Material UI: The React component library. 2024. URL: https://mui.com/core/ (дата звернення: 04.12.2025).

19. OWASP Top 10: The Ten Most Critical Web Application Security Risks. OWASP Foundation. 2024. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 09.12.2025).

20. React – The library for web and native user interfaces. 2024. URL: https://react.dev/ (дата звернення: 02.12.2025).

21. SQL Server Technical Documentation. Microsoft Learn. 2024. URL: https://learn.microsoft.com/en-us/sql/sql-server/ (дата звернення: 04.12.2025).

22. TypeScript: The Handbook. 2024. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 02.12.2025).

23. Using OAuth 2.0 to Access Google APIs. Google Identity. 2024. URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 05.12.2025).

24. Beck K. et al. Manifesto for Agile Software Development. 2001. URL: https://agilemanifesto.org/ (дата звернення: 05.12.2025).

25. Chacon S., Straub B. Pro Git. Apress, 2014. URL: https://git-scm.com/book/en/v2 (дата звернення: 08.12.2025).

26. Cross-Origin Resource Sharing (CORS). MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 09.12.2025).

27. Docker Documentation. Docker Inc. 2024. URL: https://docs.docker.com/ (дата звернення: 06.12.2025).

28. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 02.12.2025).

29. Jest: Delightful JavaScript Testing. Meta. 2024. URL: https://jestjs.io/ (дата звернення: 07.12.2025).

30. Miller J. Hybrid Project Management: What It Is and How to Implement It. Project Management Institute. 2022. URL: https://www.pmi.org/learning/library/hybrid-project-management-what-how-12888 (дата звернення: 05.12.2025).

31. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. 2020. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 11.12.2025).

32. Swagger. API Documentation & Design Tools for Teams. SmartBear. 2024. URL: https://swagger.io/ (дата звернення: 03.12.2025).

33. xUnit.net: The free, open source, community-focused unit testing tool for the .NET Framework. 2024. URL: https://xunit.net/ (дата звернення: 07.12.2025).