ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Інформаційна система моніторингу вхідних дзвінків абітурієнтів та студентів університету на основі даних АСК ВНЗ та ЄДЕБО»

Ступінь вищої освіти – бакалавр

Спеціальність – 122 «Комп'ютерні науки»

Освітня програма «Комп'ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Гліб СПИЧАК

Керівник: викладач кафедри комп'ютерних наук
Богдан БОЙКО

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.

Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Спичак Гліб Павлович

Тема роботи	Інформаційна система моніторингу вхідних дзвінків абітурієнтів та студентів університету на основі даних АСК ВНЗ та ЄДЕБО
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити інформаційну систему моніторингу вхідних дзвінків абітурієнтів та студентів університету, що інтегрується з даними АСК ВНЗ та ЄДЕБО для оперативної обробки та аналізу комунікаційної активності. Система повинна автоматично ідентифікувати абонента за номером телефону, реєструвати дзвінки та формувати аналітичні звіти для адміністрації університету.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. B. Goode, "Voice over Internet protocol (VoIP)" // IEEE Journals & Magazine. URL: https://ieeexplore.ieee.org/abstract/document/1041060 2. M. Autili, "Web Frameworks for Desktop Apps: an Exploratory Study" // IEEE ESEM. URL: https://dl.acm.org/doi/abs/10.1145/3382494.3422171 3. I. Fette, "The WebSocket Protocol" // RFC, URL: https://www.rfc-editor.org/rfc/rfc6455.html
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп'ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання «27» грудня 2024 р.

Керівник Богдан БОЙКО

Здобувач освітнього ступеня бакалавра Гліб СПИЧАК

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
Основний етап			
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
Завершальний етап			
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Богдан БОЙКО

Здобувач освітнього ступеня бакалавра Гліб СПИЧАК

Спичак Г.П. Інформаційна система моніторингу вхідних дзвінків абітурієнтів та студентів університету на основі даних АСК ВНЗ та ЄДЕБО

У роботі розглянуто розробку інформаційної системи моніторингу вхідних дзвінків абітурієнтів та студентів університету з інтеграцією даних “АСК ВНЗ” та “ЄДЕБО”. Система автоматизує ідентифікацію абонента за номером телефону, збір актуальної інформації з державних баз та відображення її в єдиному інтерфейсі оператора. Розроблено функціональні та нефункціональні вимоги, проведено аналіз існуючих аналогів, спроектовано архітектуру системи та визначено ключові модулі інтеграції.

Ключові слова: телефонія, “АСК ВНЗ”, “ЄДЕБО”, VoIP, інтеграція даних, автоматизація процесів.

Табл.: 4. Рис.: 23. Бібліограф.: 18 найм.

Spychak H.P. Information system for monitoring incoming calls from university applicants and students based on data from the university's ASK VNZ and EDBO

The study explores the development of an information system for monitoring incoming calls from university applicants and students, integrating data from “ASU VNZ” and “EDBO”. The system automates subscriber identification by phone number, retrieves up-to-date information from government databases, and displays it in a unified operator interface. Functional and non-functional requirements were defined, existing solutions were analyzed, the system architecture was designed, and key integration modules were specified.

Keywords: telephony, “ASK VNZ”, “EDBO”, VoIP, data integration, process automation.

Table: 4. Fig.: 23. Bibliography: 18 items.

ЗМІСТ

ВСТУП

Актуальність теми випливає з потреб окремих підрозділів *(прик. студентський офіс, приймальна комісія)*, котрим необхідно у режимі реального часу отримувати усю доступну інформацію про телефонуючого студента або абітурієнта.

З проблеми виникає **мета проєкту** – створення системи збору інформації про студентів/абітурієнтів та відображення у режимі реального часу цієї інформації на стаціонарних комп'ютерах працівників підрозділів.

Завданням дослідження є:

- дослідити перелік доступної інформації та методи її екстракції та обробки системою;
- дослідити перелік необхідної підрозділам інформації та побажання по сортуванню її по важливості;
- дослідити аналогічні рішення та принципи їх роботи;
- розробити бізнес-процеси та логіку роботи мінімальної життєздатної версії застосунку;
- проаналізувати наявні технологічні рішення в університеті-інтеграторі;
- розробити мінімально життєздатний продукт з урахуванням доступних технологічних рішень в університеті-інтеграторі.

Об'єктом дослідження є процес комунікації абітурієнтів та студентів університету через вхідні дзвінки, що включає автоматизований збір, обробку та аналіз інформації з систем АСК ВНЗ та ЄДЕБО, що породжує проблемну ситуацію оперативного доступу до актуальних даних.

Предметом дослідження є інформаційна система моніторингу вхідних дзвінків абітурієнтів та студентів університету, що інтегрується з АСК ВНЗ та ЄДЕБО для автоматизації обробки дзвінків та управління інформацією.

Структура та обсяг пояснювальної записки. Пояснювальна записка складається зі вступу, трьох розділів *(постановка завдання на розробку,*

проєктування та реалізація), висновків, списку посиналь (*18 найменувань*) та *5 додатків*. Пояснювальна записка містить *23 малюнків*, *4 таблиці*. Загальний обсяг пояснювальної записки складає *54 сторінки*, з яких *45 сторінок* – основний зміст записки.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ

1.1 Опис предметної області

У сучасному університеті процес комунікації з абітурієнтами та студентами включає кілька каналів: телефонні дзвінки, електронну пошту, онлайн-чати та особисті звернення. Серед них телефонний зв'язок залишається одним із найпоширеніших і найшвидших способів отримати інформацію та вирішити поточні питання. Працівники університету *(а особливо приймальної комісії та студентського офісу)* щодня обробляють сотні дзвінків, що створює високе навантаження та ризик затримок у наданні відповіді.

Існуючі підходи до обробки телефонних звернень базуються на ручному пошуку даних у різних інформаційних системах: “АСК ВНЗ”, “ЄДЕБО”, Бухгалтерія та різноманітні Microsoft Excel та Microsoft Sharepoint списки. Це призводить до таких проблем:

- Тривалість обробки дзвінка. Працівник витрачає час на пошук даних у розрізнених джерелах.
- Неповнота інформації. Можливість пропустити важливу деталь через відсутність єдиного інтерфейсу.
- Високе навантаження. Зростає вірогідність помилок та зниження якості обслуговування.

Університет наразі використовує такі інформаційні системи:

- АТС MyPBX U510;
- Asterisk;
- АСК ВНЗ;
- ЄДЕБО;
- перехрестна таблиця: таблиця у SQL Server, котра прив’язує ідентифікаційні коди різних систем до одного користувача;
- списки у Microsoft Sharepoint;
- списки Microsoft Excel;

– база даних бухгалтерії;
– Moodle.

На рисунку 1.1 наочно продемострировано суть основної проблеми – усі існуючі компоненти та інформаційні системи відокремлені одна від одної та не мають ніяких зв’язків.

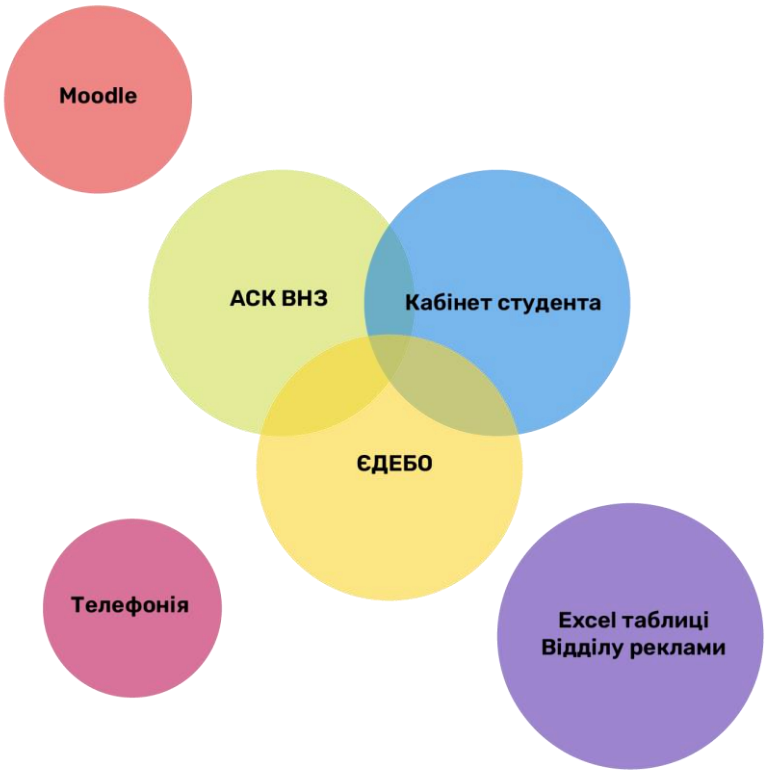

Рисунок 1.1 – Діаграма існуючих інформаційних систем та їх зв’язки між собою

Джерело: розроблено автором

Для підвищення ефективності необхідно створити інформаційну систему моніторингу вхідних дзвінків, яка в режимі реального часу:

1. Автоматично ідентифікує абонента за номером телефону.
2. Підтягує з баз “АСК ВНЗ”, “ЄДЕБО” та інших актуальну інформацію про статус вступника чи студента.
3. Формує єдиний інтерфейс для оператора з усіма необхідними даними та інструментами комунікації.

1.2 Дослідження функціональних вимог системи

На основі опитування основних користувачів системи *(а саме працівників приймальної комісії, студентського офісу та реклами Університету «КРОК»)* були визначені ключові функціональні та нефункціональні вимоги до системи.

Функціональні вимоги:

- ідентифікація абонента за номером телефону;
- отримання та відображення даних з “АСК ВНЗ” та “ЄДЕБО”;
- реєстрація історії дзвінків з часом, тривалістю та результатом;
- формування аналітичних звітів за періодами;
- налаштування шаблонів нотатків.

Нефункціональні вимоги:

- висока доступність та стабільність роботи;
- інтуїтивно зрозумілий інтерфейс;
- забезпечення безпеки передачі та зберігання даних;
- масштабованість та можливість подальшого оновлення.

1.3 Огляд аналогів

Для визначення конкурентного середовища було досліджено низку готових рішень, проте жодне з них не відповідає вимогам університету через високу складність налаштування та відсутність підтримки інтеграції з поточною АТС та державними реєстрами.

JSolutions

JSolutions – українська VoIP-платформа, що пропонує моніторинг дзвінків, маршрутизацію, голосові меню (IVR), базову аналітику, а також модулі, схожі на “АСК ВНЗ” та систему СКУД для внутрішньої аутентифікації співробітників. Недоліки:

- Відсутність готової інтеграції з АТС університету. Потрібна розробка конекторів для підключення до MyPBX510 та Asterisk.

– Обмежена підтримка державних баз. Платформа не має вбудованого доступу до АСК ВНЗ та ЄДЕБО, що вимагає реалізації додаткових API-шлюзів.
– Нестача аналітичних можливостей. Аналітика обмежується базовими звітами без гнучких фільтрів і візуалізацій, необхідних для адміністрування.
– Застарілий та навантажений інтерфейс (рис. 1.2)
–

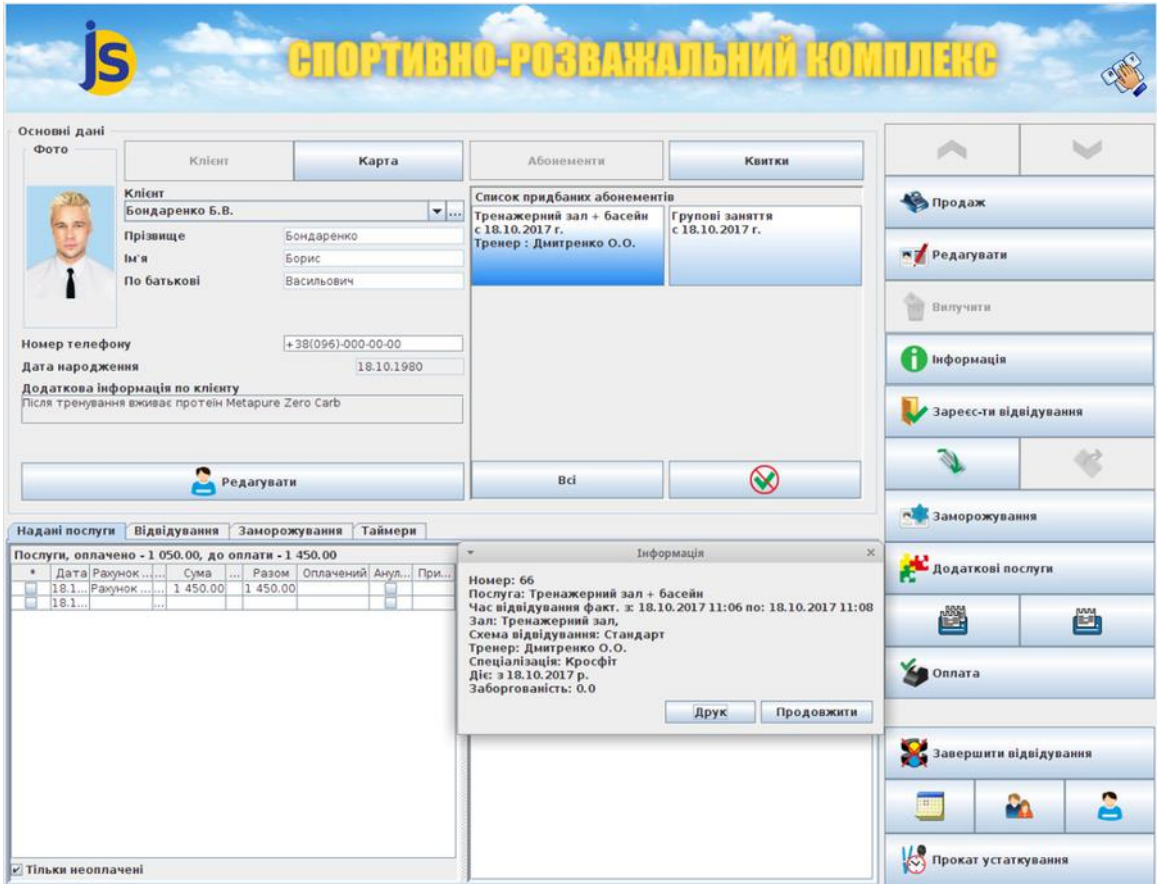

Рисунок 1.2 – Інтерфейс CRM модуля застосунку JSolutions

Джерело: https://www.livebusiness.com.ua/tool/238/

Meritto Education CRM

Meritto Education CRM – CRM-система, орієнтована на освітні установи, що надає функціонал обліку абітурієнтів, управління заявками та подіями. Вона включає у себе багато різноманітних функцій, базові з яких зазначені на рис. 1.3 нижче:

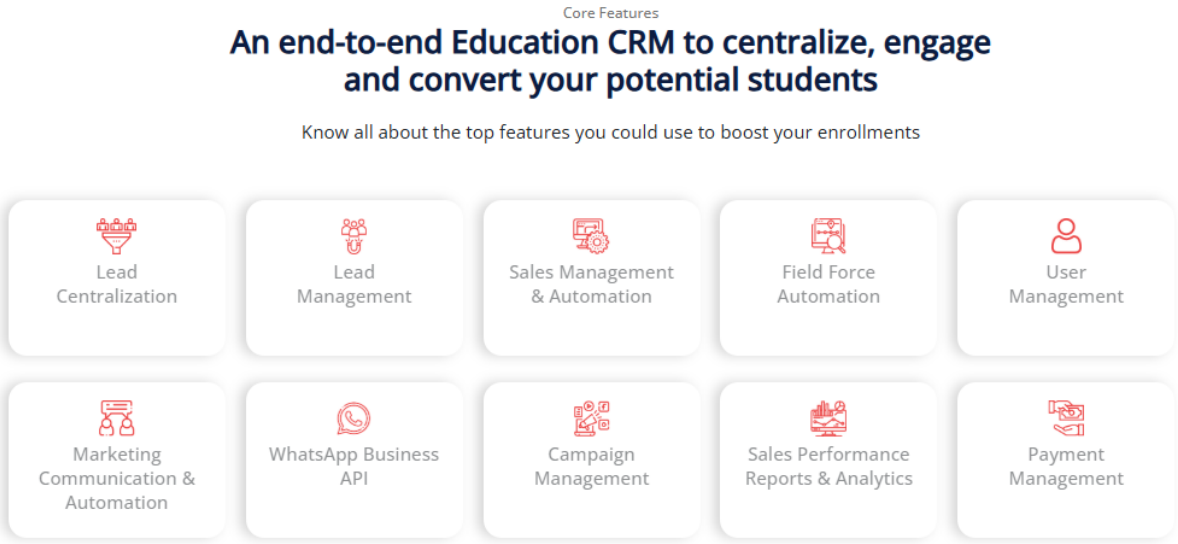

Рисунок 1.3 – Функціонал системи Meritto Education CRM

Джерело: https://www.meritto.com/education-crm/

Попри свою функціональність, вона не підходить для вирішення поставленого завдання через такі недоліки:

– обмежена підтримка телефонії. Інтеграція з VoIP-серверами здійснюється через сторонні модулі, які часто конфліктують із поточною АТС університету. Це унеможливлює стабільну роботу в умовах існуючої телефонної інфраструктури;

– відсутність прямого доступу до державних баз. Для підключення до “ЄДЕБО” та “АСК ВНЗ” необхідно розробляти окремі інтерфейси, що вимагає суттєвих витрат часу й ресурсів, а також технічної експертизи;

– висока складність кастомізації. Налаштування системи під специфіку університету потребує глибокого втручання у код, значної кількості зовнішніх розробок і високих фінансових витрат, що робить її недоцільною для швидкої реалізації.

Odoo VOIP

Odoo – це комплексна open-source платформа для управління бізнесом, яка пропонує широкий спектр додатків, включаючи CRM, продажі, бухгалтерію та інші модулі зазначені на рис. 1.4.

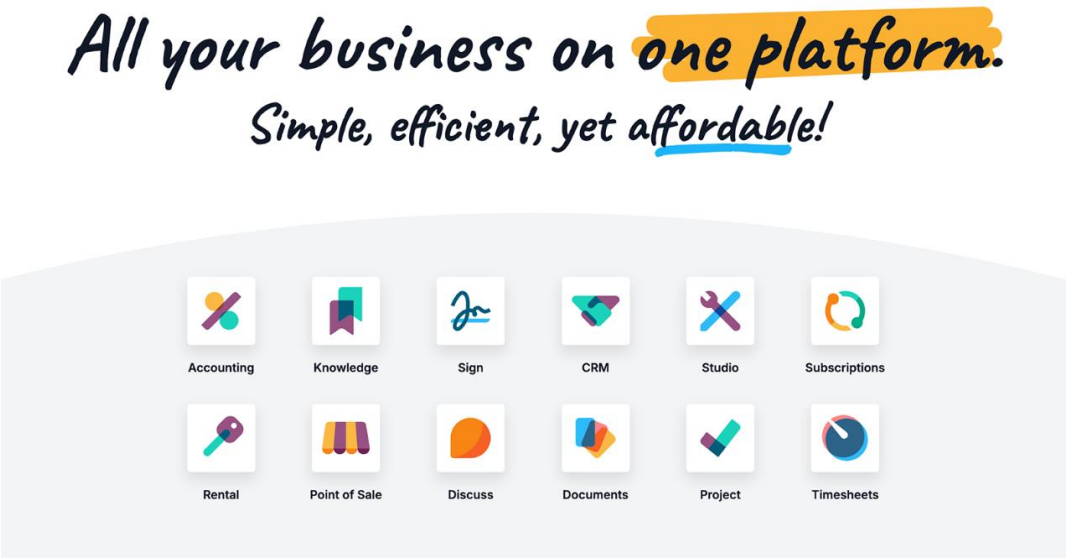

Рисунок 1.4 – Заявлений функціонал системи Odoo

Джерело: https://www.odoo.com/

Завдяки своїй модульній архітектурі, Odoo дозволяє інтегрувати різноманітні функції, що можуть бути корисними для розробки системи моніторингу вхідних дзвінків абітурієнтів та студентів університету.

Основні функціональні можливості Odoo, які можуть бути релевантними:

– інтеграція з VoIP: Odoo підтримує інтеграцію з VoIP-системами, що дозволяє здійснювати та приймати дзвінки безпосередньо з інтерфейсу платформи. Це забезпечує зручність у роботі з клієнтами та спрощує процес комунікації. Приклад того, як виглядає інтеграція VoIP системи в Odoo зазначено на рис. 1.5;

– управління дзвінками та журналювання: система автоматично реєструє всі вхідні та вихідні дзвінки, зберігаючи історію комунікацій з кожним контактом. Це дозволяє відстежувати взаємодію з абітурієнтами та студентами, аналізувати ефективність комунікацій та покращувати обслуговування;
– планування та відстеження активностей: Odoo надає інструменти для планування зустрічей, дзвінків та інших заходів, пов'язаних з контактами. Це допомагає організувати робочий процес та забезпечити своєчасне виконання завдань;
– кастомізація та розширюваність: завдяки модульній структурі, Odoo дозволяє додавати нові функції та адаптувати систему під специфічні потреби організації. Існує можливість розробки власних модулів або використання готових рішень з Odoo Apps Store.

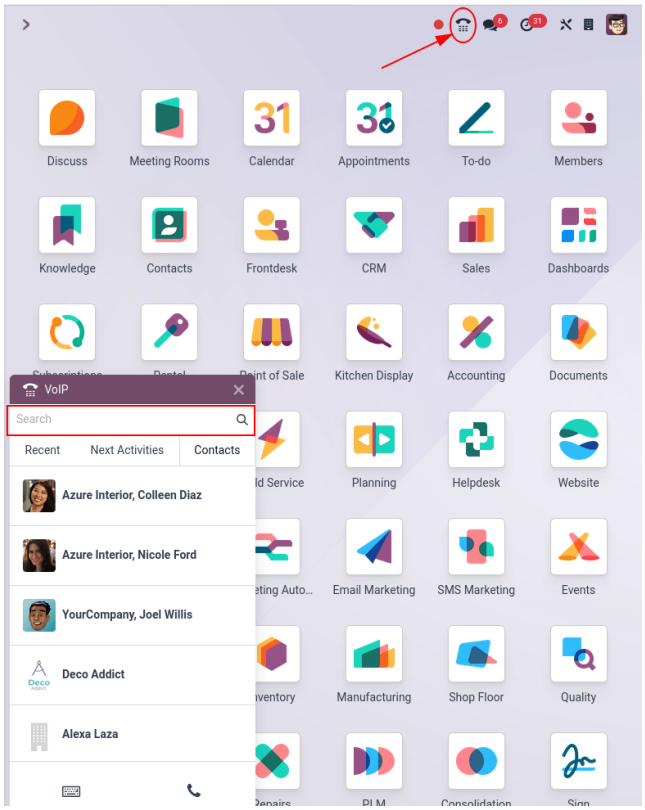

Рисунок 1.5 – Приклад інтерфейсу модуля «VoIP» системи Odoo

Джерело: https://www.odoo.com/documentation

Враховуючи вищезазначене, Odoo може бути ефективним аналогом для розробки системи моніторингу вхідних дзвінків в університеті, надаючи необхідні інструменти для управління комунікаціями та взаємодії з абітурієнтами та студентами.

Проте цю систему неможливо інтегрувати в Університеті через необхідність розробки складних конекторів. Odoo гнучкий, але для інтеграції з “АСК ВНЗ”, “ЄДЕБО”, Бухгалтерією та Microsoft Sharepoint потрібно створювати власні модулі та забезпечувати їх постійну підтримку.

1.4 Постановка завдання на кваліфікаційну роботу

Виходячи з аналізу предметної області, функціональних вимог та огляду існуючих аналогічних рішень, основними завданнями роботи є:

1. розробити архітектуру інформаційної системи моніторингу дзвінків;
2. реалізувати модулі інтеграції з аск внз та єдебо;
3. створити десктопний клієнт з графічним інтерфейсом;
4. забезпечити реєстрацію та збереження історії дзвінків;
5. розробити механізми формування аналітичних звітів;
6. провести тестування продуктивності та безпеки системи.

1.5 Висновок до розділу

Аналіз предметної області показав необхідність створення спеціалізованої системи для обробки вхідних дзвінків у вищому навчальному закладі. Огляд аналогів виявив відсутність готових рішень які можна підключити до існуючої системи з мінімальними зусиллями. Поставлені завдання формують основу для подальшого проектування та реалізації системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ

2.1 Моделювання поведінки продукту

У цьому підрозділі викладено етапи формування теоретичної основи майбутньої системи моніторингу дзвінків за допомогою трьох ключових UML-діаграм: діаграми варіантів використання, діаграми діяльності та діаграми послідовності. Спочатку здійснюється ідентифікація всіх зацікавлених сторін і зовнішніх систем, з якими взаємодіятиме наш продукт. До числа **акторів** належать оператор студ. офісу, оператор приймальної комісії та адміністратор, до **прецедентів** – такі сценарії, як «Авторизувати оператора», «Отримання інформації про телефонуючого» та інші допоміжні процеси.

Діаграма варіантів використання (рис. 2.1) дозволяє чітко окреслити межі системи й зрозуміти, які саме функції належать продукту, а які – зовнішнім компонентам. Завдяки цьому одразу видно, які ролі виконують внутрішні модулі та як вони взаємодіють із зовнішніми сервісами під час кожного з прецедентів.

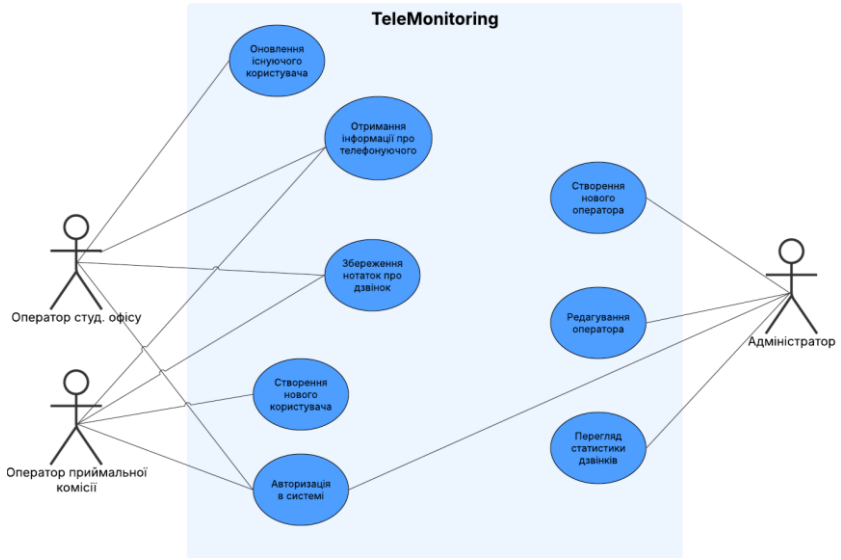

Рис. 2.1 – Діаграма варіантів використання (Use Case Diagram)

Джерело: розроблено автором

Під час побудови діаграми актор «Оператор» було розділено на дві окремі ролі відповідно до їхніх підрозділів. Таке рішення було прийнято через різні дозволи в системі: оператори студофісу можуть лише приймати дзвінки від студентів і не мають права створювати нових користувачів. Отже, їхня роль обмежується лише обробкою вже зареєстрованих абонентів.

Далі, *діаграма діяльності* (рис. 2.2) описує внутрішній робочий процес: від отримання дзвінка оператором, через послідовність дій для перевірки даних абітурієнта та збирання інформації з баз до фінального відображенняданих оператору та до збереження інформації про дзвінок в локальну базу даних.

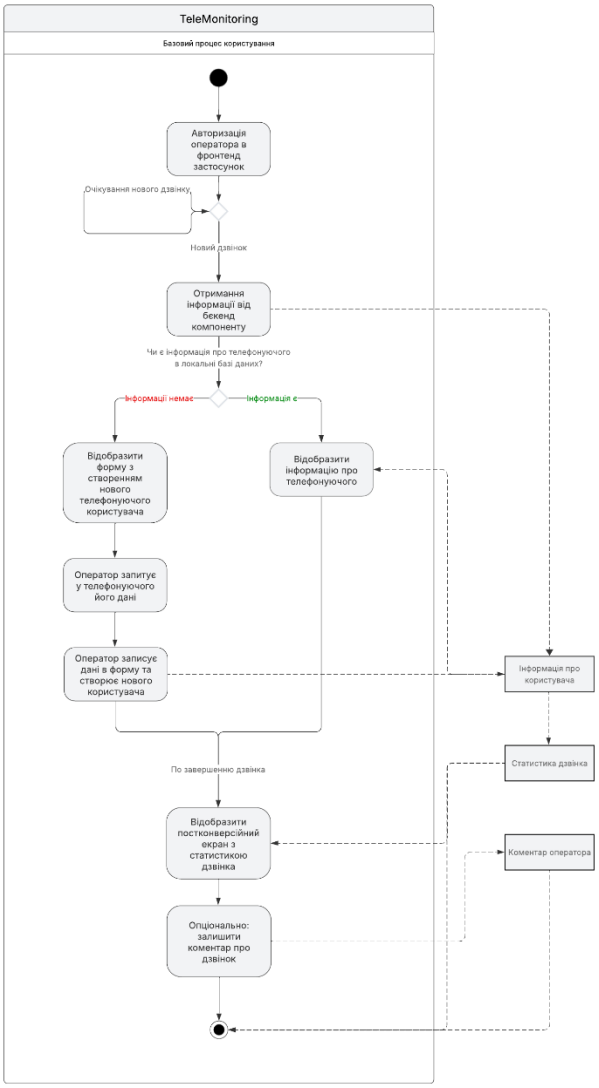

Рисунок 2.2 – Діаграма діяльності

Джерело: розроблено автором

У цій діаграмі показано повний життєвий цикл користувача в системі – від етапу авторизації та очікування вхідного дзвінка до підключення до backend-компонента, отримання даних абітурієнта та їх виводу оператору. Після завершення розмови система автоматично переходить до показу постконверсійного екрану: на ньому відображається статистика дзвінка, а оператору пропонується можливість залишити коментар. Такий кінцевий крок допомагає зібрати оцінний фідбек і зробити процес обробки дзвінків більш аналітичним.

Нарешті, *діаграма послідовності* (рис. 2.3) деталізує часовий порядок обміну повідомленнями між об’єктами системи. Це забезпечує глибоке розуміння внутрішньої взаємодії та дозволяє уточнити точну кількість та суть даних, які передаються між компонентами на ранній стадії проєктування.

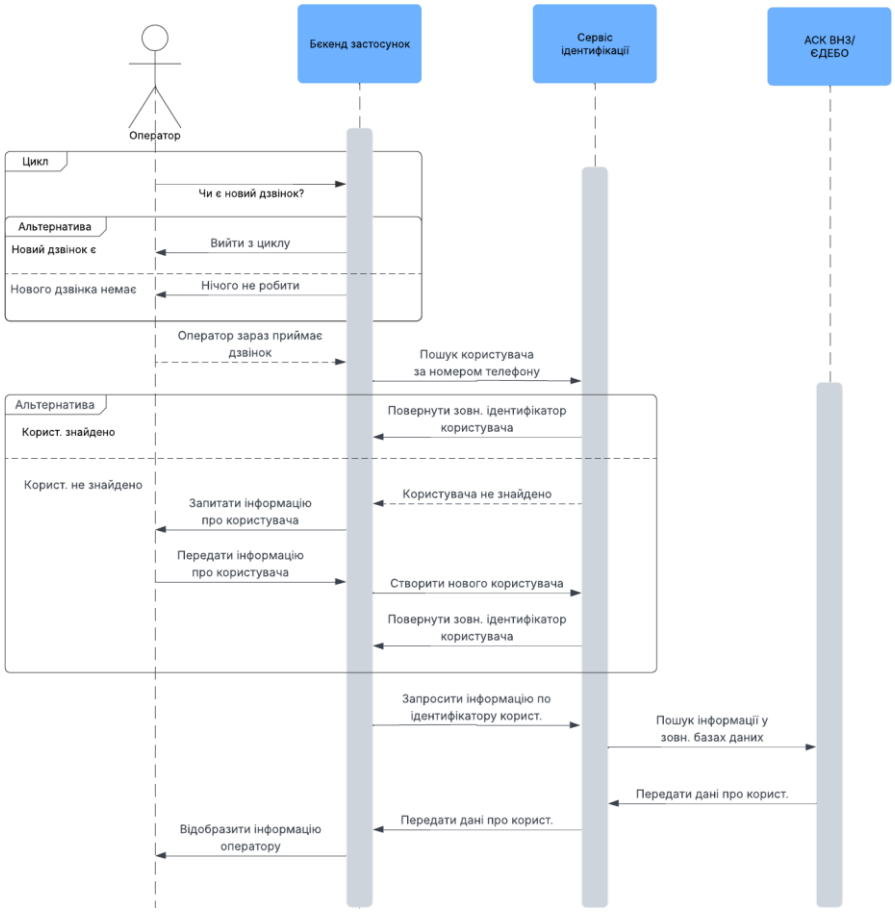

Рисунок 2.3 – Діаграма послідовності

Джерело: розроблено автором

У цій діаграмі детальніше пояснюється механізм отримання інформації про користувача – які саме повідомлення передаються між компонентами і в якому порядку. Зокрема, було додано окремий **Сервіс ідентифікації**, який відповідає за розпізнавання абонента на основі його телефонного номера. Після ідентифікації цей сервіс виконує запити до зовнішніх реєстрів – таких як “АСК ВНЗ”, “ЄДЕБО” або інші аналогічні бази – для отримання унікальних ідентифікаторів та докладних профільних даних.

2.2 Моделювання структури продукту

Система побудована за модульним принципом і включає чотири основні компоненти:

1. Клієнтський застосунок з інтерфейсом.
   Інтерактивний фронтенд, що забезпечує оператору зручну візуалізацію інформації в режимі реального часу та інструменти для обробки дзвінків.
2. Локальна база даних.
   Центральне сховище для всіх записів про сеанси дзвінків, профілі абітурієнтів і статистику.
3. Бекенд-компонент.
   Серверна частина, яка опрацьовує запити фронтенду, координує взаємодію з базою даних, викликає воркерів для фонових операцій і під’єднується до АТС задля моніторингу вхідних дзвінків.
4. Воркери (фонова обробка)
   Набір окремих служб, що виконують асинхронні завдання:
   - Синхронізація даних з АСК ВНЗ та ЄДЕБО,
   - Генерація звітів та статистики,
   - Очищення застарілих записів відповідно до політик зберігання даних.

Через високу динамічність і взаємозалежність цих модулів критично важливо одразу визначити, які дані потрапляють до кожного компоненту, а також як вони обмінюються повідомленням. Для цього:

1. Діаграма класів ілюструє внутрішню структуру локальної бази даних: основні сутності *(Оператор, Адміністраторр, Дзвінок та інші)*, їх атрибути й взаємозв'язки. Це дозволяє чітко уявити модель даних перед початком реалізації;
2. Діаграма компонентів висвітлює точки взаємодії між внутрішніми компонентами *(фронтенд, бекенд, воркери)* та зовнішніми постачальниками *(АСК ВНЗ, ЄДЕБО)*.

Нижче наведена *діаграма класів* (рис. 2.4), яка служить фундаментом для подальшого опису архітектури та забезпечує прозорість у роботі з даними.

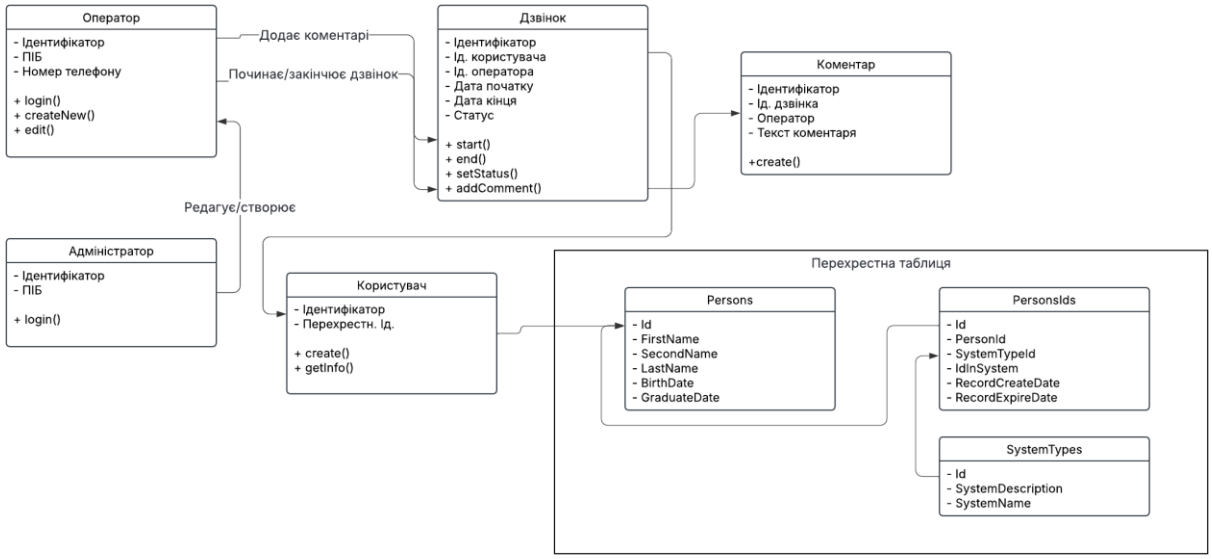

Рис. 2.4 – Діаграма класів

Джерело: розроблено автором

У *додатку А* представлен більш докладний опис усіх класів та їх атрибутів.

Ця діаграма та відповідна таблиця демонструють основну структуру класів, що формують логіку взаємодії користувача з системою. До основних

сутностей належать: Оператор, Дзвінок, Адміністратор, Користувач і Коментар. Кожен з цих класів містить базові атрибути – зокрема, унікальний ідентифікатор, що дозволяє точно ідентифікувати об'єкт у базі даних.

Клас Оператор зберігає інформацію про співробітника: його ПІБ, номер телефону та функціональні методи для створення, редагування чи авторизації. Клас Дзвінок представляє логіку з'єднання між користувачем і оператором та містить посилання на обох учасників, а також поля для фіксації початку, завершення дзвінка, його статусу й коментарів. Компонент Коментар безпосередньо пов’язаний з конкретним дзвінком і оператором, який залишив відповідний запис – це дозволяє вести історію звернень і покращувати якість обслуговування.

Клас Користувач описує особу, яка звертається до системи – студента або абітурієнта – та містить поле для зв’язку з унікальним ідентифікатором із зовнішніх джерел.

Особливу увагу в цій діаграмі приділено механізму взаємодії з зовнішніми інформаційними системами. Для цього реалізовано окремий блок – перехресну таблицю, яка представлена класами Persons, PersonsIds і SystemTypes. Це – вже існуюча в університеті система ідентифікації користувачів. Вона дозволяє зв’язати локальний запис користувача з його відповідними ідентифікаторами в таких системах, як ЄДЕБО чи АСК ВНЗ, через унікальні зовнішні ідентифікатори. Завдяки цьому підходу система може перевіряти автентичність абітурієнта та автоматично підтягувати актуальні дані.

Таким чином, діаграма наочно демонструє, як внутрішні об’єкти взаємодіють між собою та з зовнішніми джерелами, забезпечуючи повноцінну підтримку обробки дзвінків, коментарів та профілів користувачів.

Далі, спираючись на вже створену діаграму класів, було побудована *діаграма об’єктів* (рис. 2.5). Ця діаграма демонструє конкретні екземпляри класів із їхніми даними та ілюструє, в якому форматі й з якими значеннями зберігаються сутності під час роботи системи. Завдяки цьому стає очевидним,

які саме атрибути містить кожен об’єкт, як вони взаємопов’язані й який початковий стан приймають у типовому сценарії.

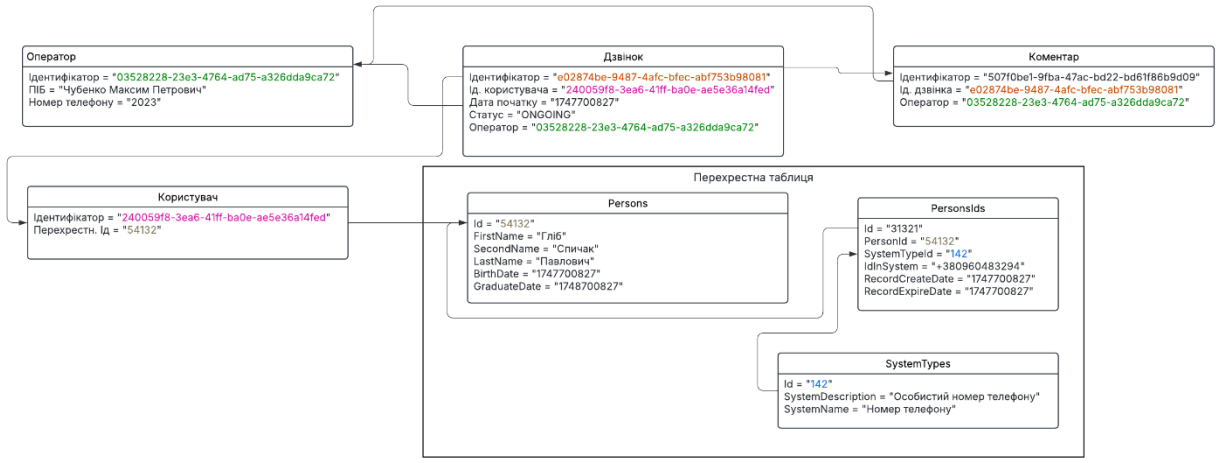

Рис. 2.5 – Діаграма об’єктів

Джерело: розроблено автором

Наступним етапом теоретичного проєктування програмного продукту стало побудова *діаграми компонентів* (рис. 2.6), яка дозволяє візуалізувати фізичну організацію системи та відобразити взаємозв’язки між її основними модулями.

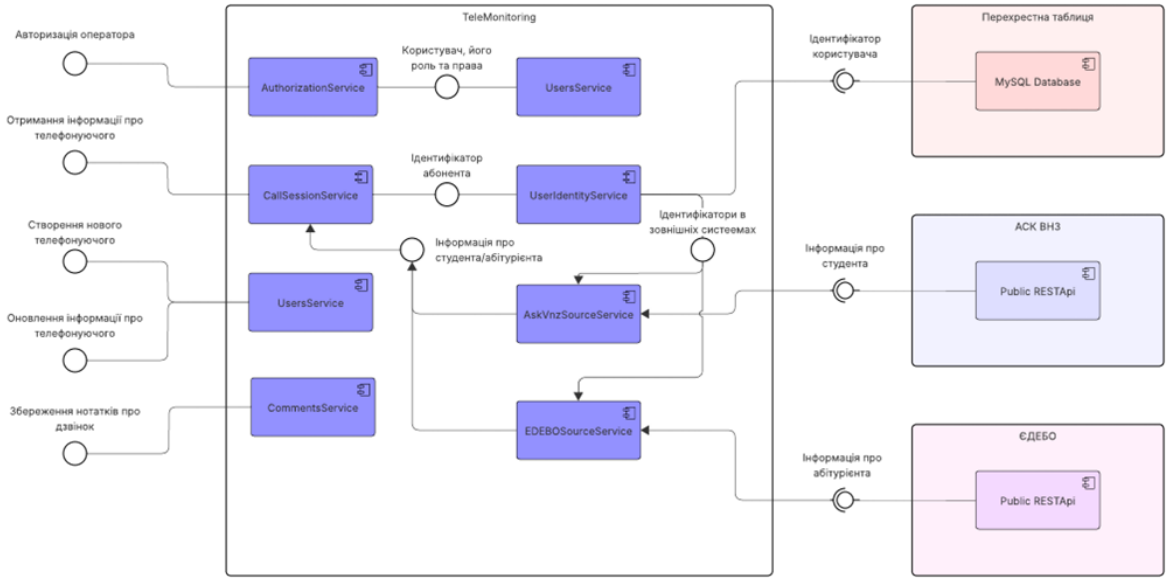

Рис. 2.6 – Діаграма компонентів

Джерело: розроблено автором

На даній діаграмі компонентів відображено структуру та взаємодію основних модулів інформаційної системи моніторингу дзвінків. Архітектура реалізована у вигляді набору взаємопов’язаних сервісів, кожен із яких виконує окрему функцію в межах загального бізнес-процесу. Авторизація операторів здійснюється через компонент AuthorizationService, який перевіряє доступ на основі ролей користувачів. Основна логіка обробки дзвінків зосереджена в CallSessionService, який взаємодіє з UserIdentityService для отримання інформації про телефонуючого. Оператор, після завершення розмови, має можливість залишити коментар, який обробляється через CommentsService і прив’язується до відповідного дзвінка.

Важливим елементом архітектури є UserIdentityService, який виконує ідентифікацію користувача за вхідним номером телефону, а також зв'язує внутрішній запис із зовнішніми джерелами через перехресну таблицю. Для збору додаткової інформації про особу система звертається до двох зовнішніх джерел – АСК ВНЗ (через AskVnzSourceService) та ЄДЕБО (через EDEBOSourceService) – використовуючи відкриті REST API. Взаємодія між компонентами побудована таким чином, щоб дані зберігались централізовано, а кожен окремий сервіс мав чітку відповідальність і був максимально ізольований.

Ця модульна архітектура дозволяє масштабувати систему, легко замінювати або розширювати окремі компоненти (наприклад, додати нові зовнішні джерела даних), забезпечує високу підтримуваність, розподіл навантаження та спрощує інтеграцію з іншими системами університету або державних реєстрів.

2.3 Опис архітектури продукту

Проєкт розробки системи моніторингу вхідних дзвінків до університету базується на сучасних підходах до архітектури програмного забезпечення. З метою формалізації, структурованості та зручності візуалізації, було використано методологію 4+1 View Model. Цей підхід дозволяє представити

архітектуру ПЗ через п’ять взаємопов’язаних точок зору: логічну, процесну, фізичну, розгортання та сценарії використання. Такий поділ забезпечує не лише технічне розуміння системи, а й зручність для спілкування між розробниками, замовниками та кінцевими користувачами.

У логічній моделі визначено основні програмні сутності системи та їхні взаємозв’язки. Система складається з кількох базових компонентів:

- *користувач* (Студент/Абітурієнт) – особа, яка телефонує в університет;
- *оператор* – авторизований співробітник, який приймає дзвінки;
- *дзвінок* – сесія між користувачем і оператором;
- *коментар* – відгук оператора після завершення дзвінка;
- *зовнішні ідентифікатори* (PersonsIds) – дані, отримані з ЄДЕБО та АСК ВНЗ.

Ці класи зображено на діаграмі класів *(див. рис. 2.4)*.

З точки зору обробки процесів система розподілена на кілька незалежних компонентів, які взаємодіють між собою:

- *фронтенд* реагує на події дзвінків у реальному часі та відображає інформацію користувачу застосунку;
- *бекенд* обробляє логіку дзвінків, користувачів та аутентифікацію;
- *фоновий воркер* регулярно синхронізує дані з ЄДЕБО/АСК ВНЗ.
- *ідентифікаційний сервіс* виконує пошук зовнішніх профілів абітурієнтів за номером телефону.

Фізично система складається з кількох компонентів, які можуть бути розгорнуті на окремих серверах або контейнерах:

- *клієнтський застосунок* встановлюється на робочі комп’ютери операторів;
- *API-сервер* розгортається на хмарному або локальному сервері;
- *фоновий воркер* виконується як окремий Node.js-процес або контейнер;

– *база даних* розміщується на сервері із забезпеченням бекапів та безпеки.

Кожен компонент може масштабуватись окремо, відповідно до навантаження, що робить систему гнучкою для розширення.

У рамках точки зору архітектури розгортання система передбачає:

– *десктопні клієнти* для операторів;
– *серверна частина*, що приймає WebSocket-з'єднання та REST-запити;
– *базу даних*, яка є централізованим джерелом даних для всієї системи.

Компоненти об'єднані внутрішньою мережею (локально або у хмарі), з можливістю розділення за контекстами доступу та ролями.

Основними сценаріями взаємодії з системою є:

– *авторизація оператора* – перевірка доступу перед початком роботи;
– *обробка дзвінка* – отримання даних користувача з баз, показ історії;
– *завершення дзвінка* – збереження коментаря та фіксація статусу;
– *синхронізація даних* – регулярне оновлення інформації про користувачів;
– *аналіз історії дзвінків* – формування звітності на основі журналу звернень.

2.4 Висновки

У другому розділі було здійснено комплексне проєктування програмного продукту з використанням сучасних методів моделювання. Це дало змогу сформувати цілісне бачення майбутньої системи як з боку її поведінки, так і з боку внутрішньої структури.

На першому етапі виконано моделювання поведінки продукту: побудовано діаграми варіантів використання, діяльності та послідовності, які дозволили ідентифікувати основні сценарії взаємодії користувача з системою, зовнішніми сервісами (АСК ВНЗ, ЄДЕБО) та внутрішніми модулями. Це

забезпечило чітке розуміння функціональних можливостей системи та логіки її роботи.

У підрозділі з моделювання структури було описано складові елементи системи: фронтенд частину, серверну логіку, локальну базу даних та фонові служби. За допомогою діаграми класів визначено основні об’єкти предметної області, їхні атрибути та зв’язки, а за допомогою діаграми компонентів – взаємодії між частинами програмного забезпечення та зовнішніми джерелами даних.

На основі архітектурного опису визначено загальну концепцію побудови системи, її компоненти та внутрішню реалізацію. Такий багаторівневий підхід дозволяє гнучко розвивати продукт, масштабувати його та адаптувати під змінні вимоги користувачів і зовнішнього середовища.

У результаті виконаної роботи створено повноцінну архітектурну та логічну модель системи, що стане основою для її безпосередньої реалізації в третьому розділі, присвяченому розробці програмного продукту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ

3.1 Реалізація фронтенд компонента застосунку

Фронтенд компонент відповідає за відображення оператору повної інформації про поточний дзвінок – ім’я, ідентифікатор, статус, історію звернень та інші дані, отримані з усіх доступних джерел

Програма встановлюється як нативний десктоп-додаток на Windows, виконує автентифікацію оператора через бекенд та підтримує постійне real-time з’єднання для надходження подій про нові дзвінки

Після отримання від бекенду події *«новий дзвінок»* фронтенд миттєво запитує детальні дані абітурієнта та виводить їх у зручному графічному інтерфейсі без потреби ручного оновлення сторінки

При виборі інструментів було обрано наступні пріоритети:

- *компактність і швидкість старту* – невеликий розмір бінарного файлу та миттєвий запуск;
- *кросплатформеність* – однаковий досвід на Windows, macOS і Linux;
- *легкість програмування* – можливість застосувати HTML, CSS і JavaScript-середовище;
- висока продуктивність – мінімальні затримки в оновленні інтерфейсу.

На основі цих вимог обрано:

1. Tauri. Tauri генерує малі й швидкі бінарні файли, оскільки сам застосунок пишеться на Rust, а UI відображається через вбудований WebView Забезпечує безпеку за замовчуванням: жорстка Content Security Policy (CSP), управління доступом до команд через permissions API та відсутність вбудованого HTTP-сервера, якщо він не потрібен Підтримує будь-який веб-фреймворк, що компілюється до HTML/CSS/JS, без додаткових змін у пакувальнику;

2. Svelte. Svelte компілює компоненти у максимально оптимізований JavaScript, позбавлений шару віртуального DOM, що забезпечує відсутність накладних витрат під час оновлення інтерфейсу Завдяки реактивним деклараціям стан оновлюється автоматично при зміні залежностей, що спрощує підтримку складних зв'язків у даних;

3. Socket.IO для комунікації з бекенд частиної в режимі реального часу. Socket.IO надає двосторонній подійний API, де клієнт і сервер можуть вільно надсилати й отримувати події. Запроваджує механізм відновлення стану після тимчасових розривів, включно з обробкою пропущених пакетів. Дозволяє передавати складні об'єкти у форматі JSON без додаткового кодування, що спрощує обмін даними.

Після остаточного затвердження технологічного стеку для фронтенду був розпочат процес безпосереднього проєктування користувацького інтерфейсу. Врахувавши специфіку робочих процесів та прагнення мінімізувати час на навчання персоналу, було вирішено зосередитися лише на трьох ключових сторінках (далі в тексті – екран).

Перший *екран авторизації* (рис 3.1) дозволяє авторизуватись оператору та гарантує безпечний вхід у систему. Авторизація відбувається за допомогою зовнішнього сервісу *Microsoft Entra Id*, котрий на даний момент є головним методом авторизації для усіх співробітників та студентів університету.

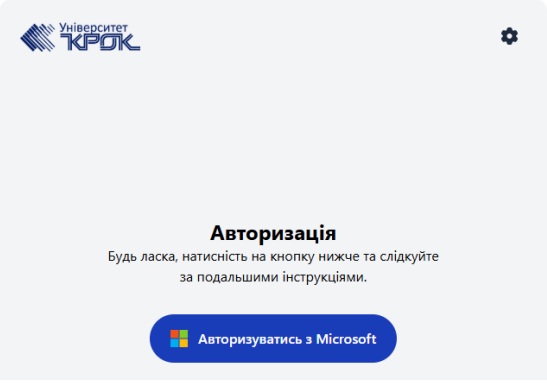

Рисунок 3.1 – Екран авторизації оператора

Джерело: розроблено автором

Другий екран застосунку *(див. рис. 3.2 та рис. 3.3)* виконує ключову функцію в роботі оператора, оскільки саме на ньому відображається повна інформація про поточний вхідний дзвінок. У реальному часі на екран виводяться персональні дані абітурієнта або студента, які система автоматично отримує із зовнішніх джерел. Окрім основної ідентифікаційної інформації, оператор також бачить історію попередніх звернень цього користувача – коли саме відбувалися дзвінки, які питання піднімалися, хто з операторів їх обробляв, а також залишені коментарі.

Важливо зазначити, що цей екран є динамічним у плані розвитку, оскільки він має стати центром усієї взаємодії з даними про абонента. Надалі передбачається його поступове розширення – доопрацювання в частині глибшої інтеграції з іншими внутрішніми та зовнішніми інформаційними системами університету, а також додавання нових блоків даних, аналітичних віджетів і функціоналу фільтрації.

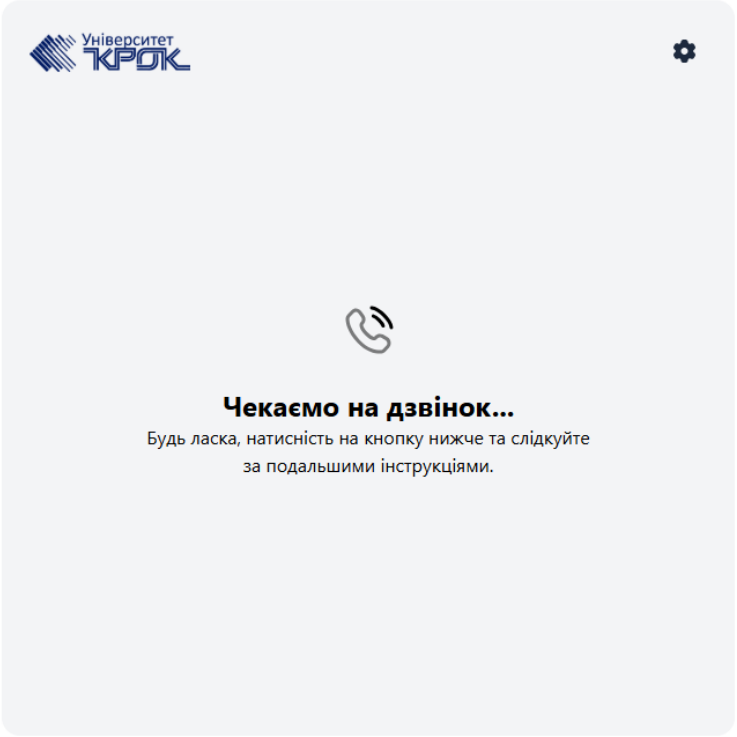

Рисунок 3.2 – Головний екран застосунку

Джерело: розроблено автором

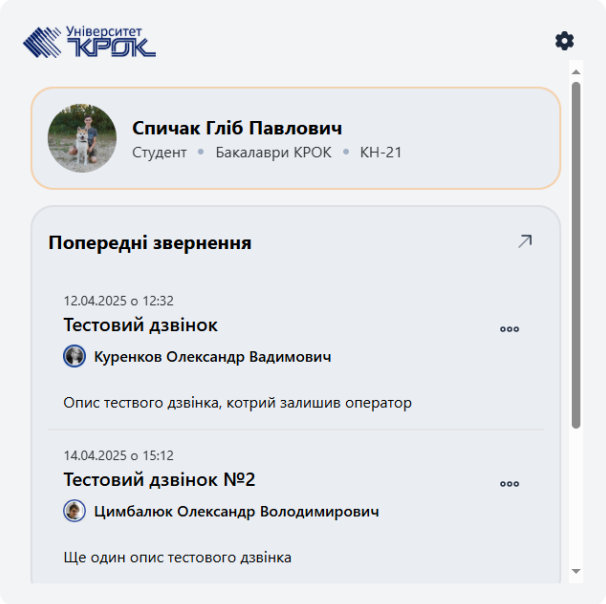

Рисунок 3.3 – Екран з інформацією про телефонуючого

Джерело: розроблено автором

Третій, *постконверсійний екран* (рис. 3.4) автоматично з’являється одразу після завершення розмови та надає оператору зручну форму для швидкого введення коментаря; ці відгуки одразу фіксуються й стають доступними всім іншим операторам у момент наступного дзвінка відповідного абітурієнта.

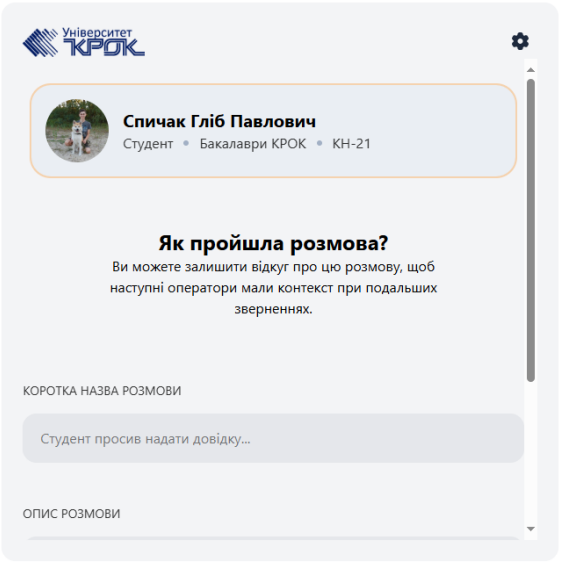

Рисунок 3.4 – Постконверсійний екран

Джерело: розроблено автором

Такий мінімалістичний підхід до структури інтерфейсу не лише пришвидшує орієнтацію у застосунку, але й знижує навантаження на користувача, дозволяючи зосередитися на головному – якості взаємодії з абітурієнтом.

3.2 Реалізація бекенд компонента застосунку

При визначенні технологій для створення бекенд-компонента системи було обрано наступні критерії для пошуку технологічного стеку для розробки: швидка розробка, легка підтримка та гнучке масштабування. Крім того, важливо було забезпечити:

– потужну базу для створення модульних та тестованих сервісів, який спрощує організацію коду та впровадження найкращих практик *(Dependency Injection, middleware, аутентифікація тощо)*;

– підтримку сучасного *TypeScript-стека*, щоб уніфікувати мову розробки фронтенд та бекенд компонентів системи;

– зручні засоби для роботи з реляційною базою даних, зокрема підтримку складних транзакцій і міграцій, але без громіздких конфігурацій або шаблонного коду.

Відтак було обрано наступний технологічний стек:

– Node.js. Легковага платформа що дозволяє обслуговувати велику кількість одночасних з’єднань із мінімальним споживанням пам’яті;

– NestJS. Прогресивний фреймворк поверх Express/Koa, побудований із використанням TypeScript та архітектурних патернів (модулі, контролери, сервіси, DI-контейнер). Забезпечує чітку структуру проєкту, вбудовану підтримку мідлварів, інтерсепторів та гвардів для авторизації й аутентифікації;

– Drizzle ORM. Легкий і водночас потужний ORM-інструмент із повною підтримкою TypeScript-типів і декларативним синтаксисом для побудови запитів. Drizzle дозволяє писати безпечні SQL-запити без втрати

контролю над продуктивністю й накладає мінімальні накладні витрати в порівнянні з важкішими ORM;

– PostgreSQL. Стабільна й перевірена реляційна СУБД зі широкими можливостями для роботи зі складними структурами даних та транзакціямі. Підтримує масштабування, реплікацію та забезпечує високий рівень цілісності й надійності.

Комбінація цих технологій дозволила прискорити розробку, зберегти простоту архітектури й водночас отримати масштабовану, добре типізовану та тестовану бекенд-систему, готову до розширення й інтеграції з іншими сервісами.

При розробці серверної частини застосунку було суворо дотримування офіційних рекомендацій NestJS щодо архітектурних патернів і неймінг-конвенцій. Кожен модуль, сервіс, контролер і гвард отримали зрозумілі та однозначні імена – наприклад, AuthenticationProviderService відповідає за ініціалізацію й перевірку облікових даних (приклад коду на рис 3.5), а UsersService містить бізнес-логіку, пов’язану з управлінням профілями користувачid.

```
@Injectable()
export class AuthenticationProviderService {
    constructor(
    // Deps
    private readonly jwtService: JwtService,
    private readonly usersService: UsersService,
    private readonly tokenService: TokenService,
    private readonly configModule: ConfigurationModule,
  ) {}

  /**
   * Check the validity of a jwt authorization token
   *
   * @param token - JWT token that we need to check
   * @returns isValid - whatever this token is valid or no
   */
  public async isTokenValid(token: string): Promise<boolean> {
    try {
      this.jwtService.verify(token, this.configModule.get("JWT_CONFIGURATION"));
      return true;
    } catch {
      return false;
    };
  };
```

Рисунок 3.5 – Приклад коду з сервісу AuthenticationProviderService

Джерело: розроблено автором

На рівні протоколів HTTP-запитів застосовано middleware AuthGuard *(приклад коду на рис 3.6)*, який перехоплює кожен запит і перевіряє наявність валідного JWT-токена або сесії перед передачею управління до захищених маршрутів.

```
import { Injectable, CanActivate, ExecutionContext, UnauthorizedException } from '@nestjs/common';
import { Request } from "express";
import { Observable } from 'rxjs';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    // Deps
    private readonly authenticationService: AuthenticationProviderService;
  ) {}

  canActivate(
    context: ExecutionContext,
  ): Promise<boolean> {
    const request: Request = context.switchToHttp().getRequest();

    // Checking if our request have Authentication header
    if (request.get("Authentication") == null) return false;

    return this.authenticationService.isTokenValid(request.get("Authentication"));
  }
}
```

Рисунок 3.6 – Приклад коду з гуарду AuthGuard

Джерело: розроблено автором

Окрему увагу приділено фоновій обробці даних. Зокрема, для організації черги завдань застосовано bullmq – потужний інструмент на основі Redis, що підтримує різні пріоритети, повтори та затримки виконання. За допомогою bullmq реалізовано планувальник завдань (cron-джоб), який щоденно ініціює операцію оновлення локальної бази даних. Під час виконання цього завдання система отримує зі зовнішньої БД (“АСК ВНЗ” або “ЄДЕБО”) свіжі дані про кожного користувача та ставить у чергу відповідні роботи на оновлення профілів.

Крім того, бекенд включає:

– Модулі для конфігурації (ConfigModule) з підтримкою змінних оточення та валідаторами;
– Фільтри виключень (ExceptionFilters) для єдиної обробки помилок;

- Гварди (Guards) для доступу тільки для авторизованих клієнтів (AuthGuard) і обмеження швидкості запитів (ThrottlerGuard) задля захисту від DDoS-атак;
- Пайплайни (Pipes) для трансформації та валідації вхідних DTO з клас-валідаторами.

Загалом, обраний стек і архітектурні рішення дозволяють отримати масштабований, безпечний і легко підтримуваний бекенд, де чітко розподілені зони відповідальності, а фонові процеси організовані через надійну чергову систему.

Одним із ключових елементів під час розробки серверної частини застосунку була побудова та організація бази даних, яка забезпечує збереження, обробку та доступ до всієї інформації, що використовується в системі. Особливу увагу було приділено вибору технологічного стеку для реалізації цієї частини. У якості системи управління базами даних (СУБД) було обрано **PostgreSQL**, яка є потужним і надійним об’єктно-реляційним рішенням з широкими можливостями для роботи з транзакціями, зв’язками між таблицями, а також підтримкою складної логіки запитів.

Для роботи з базою даних на рівні застосунку була використана сучасна ORM-бібліотека **Drizzle ORM**, що забезпечує типобезпечний підхід до побудови SQL-запитів за допомогою мовних конструкцій TypeScript. Бібліотека дозволяє описувати структуру таблиць у вигляді окремих об’єктів, що робить розробку більш передбачуваною, зручною для тестування й підтримки. На відповідному *рисунку 3.7* нижче представлена схема вже реалізованої бази даних, а також на *рисунку 3.8* представлен зразок фрагмента коду, який демонструє створення однієї з основних сутностей у системі – наприклад, таблиці користувачів або дзвінків – за допомогою синтаксису Drizzle ORM.

Однією з переваг цієї бібліотеки є наявність CLI-інструменту, який після опису моделей у коді автоматично генерує SQL-міграції.

Повний перелік моделей, які були створені для даного проєкту, а також їх опис у вигляді коду, наведено у **Додатку Д**.

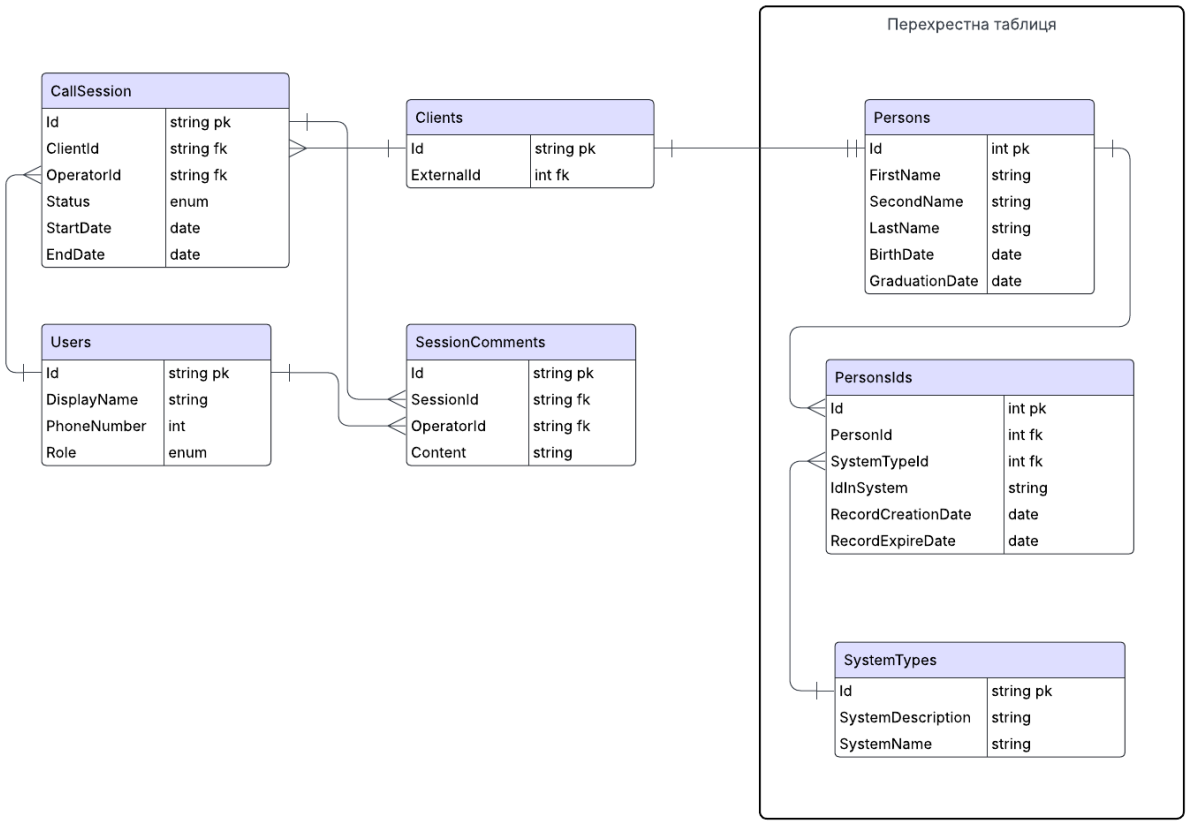

Рисунок 3.7 – Схема реалізованої бази даних.

Джерело: розроблено автором.

```
import { date, pgTable, uuid, pgEnum } from "drizzle-orm/pg-core";

// CallSession status enum definiton
export const statusEnum = pgEnum('status', ['INCOMING', 'ONGOING', 'REJECTED', 'COMPLETED']);

// CallSession table definition
export const callSessionTable = pgTable("call_session", {
  id: uuid().primaryKey().defaultRandom(),
  clientId: uuid().notNull(),
  operatorId: uuid().notNull(),
  status: statusEnum().notNull().default("INCOMING"),
  startDate: date().notNull().defaultNow(),
  endDate: date(),
})
```

Рисунок 3.8 – приклад реалізації моделі "Дзвінок" в коді.

Джерело: розроблено автором.

3.3 Тестування застосунку

Для забезпечення надійності та працездатності створеного програмного забезпечення було проведено комплексне тестування всіх його основних компонентів: фронтенду, бекенду та фонових сервісів. У процесі тестування основна увага приділялася перевірці стабільності виконання бізнес-логіки, коректності обробки вхідних даних, відповідності результатів очікуваним значенням, а також перевірці відмовостійкості системи.

Було застосовано кілька рівнів тестування:

- unit-тестування для окремих функцій та методів у бекенд- і фронтенд-компонентах;
- інтеграційне тестування REST API та взаємодії між модулями;
- ручне тестування інтерфейсу для перевірки UX/UI поведінки та валідації логіки;
- фонове тестування воркерів для перевірки черги завдань та взаємодії з зовнішніми джерелами.

Інструменти, що використовувалися:

- vitest і jsdom – для unit-тестування фронтенду;
- @nestjs/testing з Jest – для перевірки бекенду;
- postman – для ручного тестування HTTP-запитів.

У *таблиці 3.1* представлен приклад результатів unit-тестування бекенд компоненту системи. Повна таблиця з коротким описом процесу та загальних результатів тестування представлена у *додатку Б.*

Таблиця 3.1 – приклад результатів unit-тестування бекенд компоненту системи.

Назва модуля	*Метод*	*Опис тесту*	*Очікуваний результат*	*Результат*
AuthorizationService	isTokenValid(token)	Перевіряє автентичність токену авторизації	Метод повертає true	Успішно

Таблиця 3.1 – приклад результатів unit-тестування бекенд компоненту системи.

Назва модуля	*Метод*	*Опис тесту*	*Очікуваний результат*	*Результат*
			для дійсного токена, false – для недійсного.	
	createTokenFor(user)	Створює новий токен авторизації для користувача	Створюється валідний токен, який проходить перевірку методом isTokenValid().	Успішно
CallSession Service	handleAsteriskEvent(event)	Отримує подію від Asterisk як аргумент. Головна бізнес-логіка застосунку описана тут.	Відповідні зміни стану сесії зберігаються, логіка виконана згідно з очікуванням.	Успішно
	setStatus(id, status)	Встановити статус для розмови.	Статус оновлюється в базі та доступний для перевірки через інші методи.	Успішно
	getCallsFor(user)	Отримує користувача/оператора як аргумент; повертає список усіх розмов з цим користувачем/оператором.	Повертається коректна вибірка записів з відповідною фільтрацією.	Успішно

У наступній *таблиці 3.2* також представлен приклад результатів з інтегнраційного тестування бекенд застосунку. В даному випадку тестування відбувалось за допомогою застосунку Postman та за мету ставилось перевірити публічні API шляхи застосунку, котрими користується фронтент компонент система задля отримання або модифікації інформації. Повна таблиця представлена в *додатку В*.

Таблиця 3.2– приклад результатів інтеграційного тестування бекенд компоненту системи.

Endpoint	*Метод*	*Опис тесту*	*Очікувана відповідь*
/auth/login	POST	Авторизація оператора	Відповідь зі статусом 200 OK, об’єкт з токеном авторизації
/calls	GET	Список усіх дзвінків	Відповідь зі статусом 200 OK, масив об’єктів дзвінків
/call	POST	Створити новий дзвінок	Відповідь зі статусом 201 Created, об’єкт створеного дзвінка
/call/:callId	GET	Інформація про дзвінок по ідентифікатору	Відповідь зі статусом 200 OK, об’єкт дзвінка з деталями (ідентифікатор, статус тощо)
/call/:callId/status	POST	Змінити статус дзвінку	Відповідь зі статусом 200 OK, підтвердження оновлення статусу

Таблиця 3.2– приклад результатів інтеграційного тестування бекенд компоненту системи.

Endpoint	*Метод*	*Опис тесту*	*Очікувана відповідь*
/call/:callId/comments	GET	Список усіх коментарів дзвінка	Відповідь зі статусом 200 OK, масив об’єктів коментарів

Остання *таблиця 3.3* показує приклад результатів ручного тестування UI/UX фронтент компоненту системи. Повна таблиця представлена в *додатку Г*.

Таблиця 3.3 – приклад результатів ручного тестування UI/UX фронтенд компоненту системи.

Екран	*Що перевірялось*	*Очікуваний результат*	*Статус*
Авторизаційний екран (рис. 3.1)	Візуальне відображення форми входу, інтеграція з Microsoft Entra ID	Користувач може авторизуватись через корпоративний акаунт. Після успішної авторизації виконується перенаправлення до головного екрана системи.	Пройдено
Екран вхідного дзвінка (рис. 3.2)	Автоматичне відображення персональних даних абітурієнта/студента при дзвінку	У режимі реального часу відображається ПІБ, попередні звернення, дати дзвінків, відповідальні	Пройдено

Таблиця 3.3 – приклад результатів ручного тестування UI/UX фронтенд компоненту системи.

Екран	*Що перевірялось*	*Очікуваний результат*	*Статус*
		оператори, пов’язані коментарі.	
	Завантаження інформації із зовнішніх джерел	Дані успішно підтягуються з баз зовнішніх систем без затримок або помилок.	Пройдено
	Інтерактивність елементів інтерфейсу (розгортання історії звернень, навігація)	Кнопки та елементи історії звернень працюють належним чином. Перегляд деталей дзвінка доступний без перезавантаження сторінки.	Пройдено
	Готовність до розширення (додавання нових блоків, інтеграція з іншими системами)	UI побудований модульно. Технічно готовий до інтеграції нових віджетів, розширення функціоналу, реалізації фільтрів.	Пройдено
Екран завершення дзвінка (рис. 3.4)	Доступність форми коментарів після завершення розмови	Після завершення дзвінка відображається форма введення	Пройдено

Таблиця 3.3 – приклад результатів ручного тестування UI/UX фронтенд компоненту системи.

Екран	*Що перевірялось*	*Очікуваний результат*	*Статус*
		коментаря. Дані успішно зберігаються, коментарі стають доступними іншим операторам у наступних зверненнях.	
	Збереження введених коментарів	Усі введені коментарі фіксуються в базі даних і коректно асоціюються з відповідним дзвінком і користувачем.	Пройдено

3.4 Висновки

Фронтенд-модуль на базі Tauri, Svelte і Socket.IO успішно поєднав мінімалістичний дизайн із високою швидкістю відображення даних у реальному часі, що значно покращує оперативність реакції операторів на вхідні дзвінки.

Бекенд, реалізований за допомогою NestJS, Drizzle ORM і PostgreSQL, продемонстрував високу гнучкість і зрозумілу модульну архітектуру, що полегшує розширення функціоналу й підтримку коду. Завдяки чіткій

структурі сервісів і контролерів, реалізація проєкту буде зрозуміла усім, а налаштовані механізми аутентифікації й авторизації забезпечують безпечний доступ до API.

Застосування BullMQ і cron-планувальника для фонового оброблення даних гарантує своєчасне оновлення профілів абітурієнтів, регулярне створення аналітичних звітів і очищення застарілих записів без навантаження основного потоку запитів.

Загалом, інтеграція цих компонентів дала змогу побудувати відмовостійку і легко підтримувану систему моніторингу дзвінків, готову до подальшого впровадження в реальне середовище університетського кол-центру.

ВИСНОВКИ

У результаті виконаної кваліфікаційної роботи було успішно реалізовано інформаційну систему моніторингу вхідних дзвінків абітурієнтів та студентів університету, яка інтегрується з державними базами даних “АСК ВНЗ” та “ЄДЕБО”. Система дозволяє в автоматизованому режимі ідентифікувати абонента за номером телефону, оперативно отримувати інформацію з зовнішніх джерел і відображати її оператору в уніфікованому інтерфейсі.

У межах роботи проведено детальний аналіз предметної області та функціональних вимог, вивчено недоліки наявних аналогів і розроблено власну архітектурну модель із використанням методології 4+1. Було побудовано ключові діаграми для моделювання поведінки та структури продукту, визначено основні компоненти системи, включно з фронтендом, бекендом, локальною базою даних та фоновими службами.

Фронтенд реалізовано з використанням сучасного стеку *(Tauri, Svelte, Socket.IO)*, що забезпечує високу швидкість та зручність у користуванні. Бекенд побудовано на базі NestJS із використанням Drizzle ORM і PostgreSQL, що дозволяє легко масштабувати систему й дотримуватися принципів модульності. Для асинхронної обробки даних реалізовано воркери на базі BullMQ з підтримкою cron-завдань.

Уся система була протестована за допомогою сучасних інструментів *(Vitest, Jest, @nestjs/testing)*, що дозволило перевірити її стабільність, точність логіки та надійність у роботі. Завдяки цьому створено гнучке, відмовостійке та практичне рішення, яке може бути впроваджено у робоче середовище кол-центру навчального закладу з метою покращення якості обслуговування студентів і абітурієнтів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 15.05.2025).
2. Tauri Documentation. URL: https://tauri.app/start/ (дата звернення: 15.05.2025).
3. Svelte Documentation. URL: https://svelte.dev/docs (дата звернення: 15.05.2025).
4. Socket.IO Documentation. URL: https://socket.io/docs/v4/ (дата звернення: 15.05.2025).
5. Єдина державна електронна база з питань освіти (ЄДЕБО). URL: https://info.edbo.gov.ua/ (дата звернення: 15.05.2025).
6. Margolin O. UML-діаграми для моделювання програмного забезпечення. Evergreen Blog. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 15.05.2025).
7. Brown S. The C4 Model for Software Architecture. InfoQ. URL: https://www.infoq.com/articles/C4-architecture-model/ (дата звернення: 15.05.2025).
8. Diaconu A. WebSockets Explained: What They Are and How They Work. Ably Blog. URL: https://ably.com/topic/websockets (дата звернення: 15.05.2025).
9. Testing Svelte Applications. Svelte Docs. URL: https://svelte.dev/docs/svelte/testing (дата звернення: 15.05.2025).
10. Arnab K. WebSockets in NestJS: Real-Time Applications. Medium. URL: https://arnab-k.medium.com/websockets-in-nestjs-real-time-applications-992d1a91a494 (дата звернення: 15.05.2025).
11. NestJS Official Guide: Testing. URL: https://docs.nestjs.com/fundamentals/testing (дата звернення: 15.05.2025).

12. Mozilla WebSocket API (MDN). URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 15.05.2025).
13. АСУ «ВНЗ». URL: https://vuz.osvita.net/ (дата звернення: 15.05.2025).
14. SvelteKit Documentation. URL: https://svelte.dev/docs/kit/introduction (дата звернення: 15.05.2025).
15. NodeJS Documentation. URL: https://nodejs.org/dist/latest/docs/api/ (дата звернення: 15.05.2025).
16. MDN – Progressive Web Apps (PWA). URL: https://developer.mozilla.org/docs/Web/Progressive_web_apps (дата звернення: 15.05.2025).
17. PostgreSQL Documentation. URL: PostgreSQL: Documentation: 17: PostgreSQL 17.5 Documentation (дата звернення: 15.05.2025).
18. DrizzleORM Documentation. URL: https://orm.drizzle.team/docs/overview (дата звернення: 15.05.2025).

ДОДАТКИ

ДОДАТОК А

Цей додаток містить опис описаної у рис. 2.4 діаграми класів. Деякі класи було випущено з цієї таблиці через їх простоту.

Таблиця А.1 – опис атрибутів діаграми класів

Назва класу	*Назва поля*	*Тип поля*	*Короткий опис*
Оператор	Ідентифікатор	UUID	Головний унікальний ідентифікатор об'єкту
	ПІБ	Текст	Прізвище, ім'я та по-батькові оператора
	Номер телефону	Число (4 цифри)	Внутрішній номер телефону оператора. В системі PBX внутрішні номера – 4-х значні.
Дзвінок	Ідентифікатор	UUID	Унікальний ідентифікатор дзвінка
	Ід. користувача	UUID	Посилання на ідентифікатор телефонуючого (класс Користувач)
	Ід. оператора	UUID	Посилання на ідентифікатор оператора (класс Оператор)
	Дата початку	Дата	Дата початку розмови (обов'язкова)
	Дата кінця	Дата	Дата кінця розмови (опціональна)
	Статус	Перелік	Поточний статус розмови. Можливі варіанти: *INCOMING*, *ONGOING*, *REJECTED*, *COMPLETED*
Користувач	Ідентифікатор	UUID	Унікальний ідентифікатор користувача
	Перехрестн. Ідентифікатор	Число	Посилання на ідентифікатор користувача у перехрестній таблиця (класс Persons)

Таблиця А.1 – опис атрибутів діаграми класів

Назва класу	*Назва поля*	*Тип поля*	*Короткий опис*
Persons	Id	Integer PK	Унікальний ідентифікатор
	FirstName	String	Ім’я
	SecondName	String	Прізвище
	LastName	String	По-батькові
	BirthDate	Date	Дата народження
	GraduateDate	Date	Дата випуску
PersonIds	Id	Integer PK	Унікальний ідентифікатор запису
	PersonId	Integer FK	Посилання на табл. Persons
	SystemTypeId	Integer FK	Посилання на табл. SystemTypes
	IdInSystem	String	Ідентифікатор в відповідній системі (SystemTypes)
	RecordCreateDate	Date	Дата створення запису
	RecordExpireDate	Date	Дата закінчення дії цього запису
SystemTypes	Id	Integer PK	Унікальний ідентифікатор системи
	SystemDescription	String	Опис зовнішньої систеи
	SystemName	String	Назва цієї зовнішньої системи

ДОДАТОК Б

Цей додаток описує unit-тестування бекенд компоненту системи. Тестування проводилось за допомогою бібліотеки *@nestjs/testing* та за методологіями та правилами тестування, котрі рекомендує офіційна документація фреймворку NestJS.

Таблиця Б.1 – приклад результатів unit-тестування бекенд компоненту системи.

Назва модуля	*Метод*	*Опис тесту*	*Результат*
AuthorizationService	isTokenValid(token)	Перевіряє автентичність токену авторизації	
	createTokenFor(user)	Створює новий токен авторизації для користувача	
CallSessionService	handleAsteriskEvent(event)	Отримує подію від Asterisk як аргумент. Головна бізнес-логіка застосунку описана тут.	
	setStatus(id, status)	Встановити статус для розмови.	
	getCallsFor(user)	Отримує користувача/оператора як аргумент; повертає список усіх розмов з цим користувачем/оператором.	
	startCall(client)	Розпочати дзвінок	
	endCall(id)	Завершити дзвінок	
UsersService	get(id)	Отримати користувача по його ідентифікатору	
	modify(id, newData)	Оновити інформацію про оператора	
	createNew(data)	Створити нового оператор	
	delete(id)		
CommentsService	getFor(user)		
	createNew(callId, message)		
	delete(id)		

Таблиця Б.1 – приклад результатів unit-тестування бекенд компоненту системи.

Назва модуля	*Метод*	*Опис тесту*	*Результат*
UserIdentityService	getIdsFor(id)	Отримати усі зовніші ідентифікатори для цього користувача.	
	createNew(data)	Створити нового користувача	
	assignExternalId(id, extId)	Прив’язати зовнішній ідентифікатор (до перехрестної таблиці) до цього користувача.	
AskVnzSourceService	getDataFor(id)	Отримати дані про користувача по його ідентифікатору.	
EDEBOSourceService	getDataFor(id)	Отримати дані про користувача по його ідентифікатору.	

ДОДАТОК В

Цей додаток містить таблиці з результатами інтеграційного тестування API застосунку. Для цього було використано інструмент Postman, за допомогою якого було надіслано низку різноманітних запросів з різноманітними даними на бекенд сервер та оцінено результати.

Таблиця В.1– приклад результатів інтеграційного тестування бекенд компоненту системи.

Endpoint	*Метод*	*Опис тесту*	*Очікувана відповідь*
/auth/login	POST	Авторизація оператора	
/calls	GET	Список усіх дзвінків	
/call	POST	Створити новий дзвінок	
/call/:callId	GET	Інформація про дзвінок по ідентифікатору	
/call/:callId/status	POST	Змінити статус дзвінку	
/call/:callId/comments	GET	Список усіх коментарів дзвінка	
/call/:callId/end	POST	Завершити дзвінок	
/users	GET	Список усіх операторів	
/user/:id	GET	Інформація про оператора	
/user/:id/calls	GET	Список дзвінків відповідного користувача	

ДОДАТОК Г

Цей додаток містить таблицю з результатами ручного тестування UI/UX фронтенд компоненту системи. Для цього було спочатку описано усі бажані результати реагування застосунку на ті чи інші дії користувача і потім вручну було перевірено реальні результати самого застосунку.

Таблиця Г.1 – приклад результатів ручного тестування UI/UX фронтенд компоненту системи.

Екран	*Що перевірялось*	*Очікуваний результат*	*Статус*

ДОДАТОК Д

У цьому додатку міститься код усіх моделей застосунку, котрі були створенні за допомогою бібліотеки Drizzle ORM.

```
// CallSession status enum definiton
export const statusEnum = pgEnum('status', ['INCOMING', 'ONGOING', 'REJECTED', 'COMPLETED']);

// CallSession table definition
export const callSessionTable = pgTable("call_session", {
  id: uuid().primaryKey().defaultRandom(),
  clientId: uuid().notNull(),
  operatorId: uuid().notNull(),
  status: statusEnum().notNull().default("INCOMING"),
  startDate: date().notNull().defaultNow(),
  endDate: date(),
});

// CallSession relations
export const callSessionRelations = relations(callSessionTable, ({ one, many }) => ({
  client: one(clientsTable, {
    fields: [callSessionTable.clientId],
    references: [clientsTable.id]
  }),
  operator: one(usersTable, {
    fields: [callSessionTable.operatorId],
    references: [usersTable.id]
  }),
}));
```

Рисунок. Д.1. – Реалізація класу «Дзвінок» у коді застосунку.

Джерело: розроблено автором.

```
// Users role enum definition
export const userRoleEnum = pgEnum('user_role', ['OPERATOR', 'ADMINISTRATOR']);

// Users table definition
export const usersTable = pgTable('users', {
  id: uuid().primaryKey().defaultRandom(),
  displayName: varchar().notNull(),
  phoneNumber: integer().notNull(),
  role: userRoleEnum(),
});

// Users relations
export const usersRelations = relations(usersTable, ({ many }) => ({
  callSessions: many(callSessionTable)
}));
```

Рисунок Д.2. – Реалізація класу «Оператор» та «Адміністратор» у коді застосунку.

Джерело: розроблено автором.

```
// Client table definition
export const clientsTable = pgTable('clients', {
  id: uuid().primaryKey().defaultRandom(),
  externalId: integer().notNull().unique()
});

export const clientsRelations = relations(clientsTable, ({ one, many }) => ({
  callSessions: many(callSessionTable)
}));
```

Рисунок Д.3. – Реалізація класу «Користувач» у коді застосунку.

Джерело: розроблено автором.

```
// SessionComment table definition
export const sessionCommentsTable = pgTable('session_comments', {
  id: uuid().primaryKey().defaultRandom(),
  sessionId: uuid().notNull(),
  operatorId: uuid().notNull(),
  content: varchar(),
});
```

Рисунок Д.4. – Реалізація класу «Коментар» у коді застосунку.

Джерело: розроблено автором.