НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)
Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення комп’ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Платформа для проведення технічних інтерв’ю (комплексна тема). Система організації та проведення інтерв’ю.*

Індивідуальна частина № 2

Виконав: студент IV курсу, групи *ІП-72 Гнатишин Михайло Степанович*
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. вик. Ковтунець О.В.*
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *ст. вик. Вітковська І.І.*
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

доц., к.т.н., доц. Остапченко К.Б.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислюваної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення комп’ютеризованих систем*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2021 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Гнатишину Михайлу Степановичу
(прізвище, ім’я, по батькові)

1. Тема проєкту *«Платформа для проведення технічних інтерв’ю (комплексна тема). Система організації та проведення інтерв’ю» Індивідуальна частина №1.*

керівник проєкту *Ковтунець Олесь Володимирович, ст. вик.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “11” травня 2021 р. №1139-с

2. Термін подання студентом проєкту «08» *червня 2021 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, розгляд варіантів використання системи, розробка функціональних вимог, постановка задач.

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Аналіз якості програмного забезпечення: аналіз якості розробленого програмного забезпечення, опис процесів тестування.

5. Перелік графічного матеріалу

1) Схема бази даних

2) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «14» *березня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*12.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*13.04.2021*	
3.	*Постановка та формалізація задачі*	*14.04.2021*	
4.	*Аналіз вимог до програмного забезпечення*	*16.04.2021*	
5.	*Алгоритмізація задачі*	*20.04.2021*	
6.	*Моделювання програмного забезпечення*	*25.04.2021*	
7.	*Обґрунтування використовуваних технічних засобів*	*26.04.2021*	
8.	*Розробка архітектури програмного забезпечення*	*27.04.2021*	
9.	*Розробка програмного забезпечення*	*12.05.2021*	
10.	*Налагодження програми*	*13.05.2021*	
11.	*Виконання графічних документів*	*15.05.2021*	
12.	*Оформлення пояснювальної записки*	*16.05.2021*	
13.	*Подання ДП на попередній захист*	*17.05.2021*	
14.	*Подання ДП рецензенту*	*03.06.2021*	
15.	*Подання ДП на основний захист*	*08.06.2021*	

Студент _____ Михайло ГНАТИШИН

(підпис)

Керівник _____ Олесь КОВТУНЕЦЬ

(підпис)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка має загалом 93 сторінки, містить 7 рисунків, 47 таблиць та 2 джерела.

Дана дипломна роботи присвячена розробці програмного забезпечення для системи організації та проведення інтерв'ю в рамках комплексної дипломної роботи. Вона включає в себе розробку частини клієнтського додатку у вигляді односторінкового веб-застосунку та частини серверного застосунку.

В першому розділі «Аналіз вимог до програмного забезпечення» був проведений аналіз варіантів використань системи на основі варіантів використання системи, вказаних в загальній частині дипломного проєкту. На основі цих даних було сформульовано функціональні вимоги до програмного забезпечення в рамках індивідуальної частини дипломного проєкту.

В розділі «Моделювання та конструювання програмного забезпечення» було розглянуто бізнес-процеси системи, які необхідно реалізувати за допомогою програмного забезпечення. Було розроблено детальну архітектуру програмного забезпечення індивідуальної частини, включаючи схеми баз даних та аналіз безпеки системи.

В розділі «Аналіз якості та тестування програмного забезпечення» було складено план тестування, наведено основі сценарії тестування та було виконано перевірку кожного сценарію на розробленому програмному забезпеченні.

КЛЮЧОВІ СЛОВА: СПІВБЕСІДА, ВАКАНСІЯ, РЕКРУТЕР, ІНТЕРВ'ЮЕР, КАНДИДАТ, ІНЖЕНЕР, РЕДАГУВАННЯ ПРОГРАМНОГО КОДУ

ABSTRACT

Structure and scope of thesis. The explanatory note has a total of 93 pages, contains 7 figures, 47 tables and 2 sources.

This thesis is devoted to the development of software for the subsystem of organization and conducting interviews as part of a comprehensive thesis. It includes the development of part of the client application in the form of a one-page web application and part of the server application.

In the first section "Analysis of software requirements" was an analysis of the use of the subsystem based on the use of the system specified in the general part of the thesis project. Based on these data, the functional requirements for the software were formulated within the individual part of the diploma project.

In the section "Modeling and design of software" the business processes of the system that need to be implemented using software were considered. A detailed software architecture of the individual part has been developed, including database schemas and system security analysis.

In the section "Quality analysis and software testing" a testing plan was drawn up, the basis of the testing scenario was given and each scenario was tested on the developed software.

KEY WORDS: INTERVIEW, VACANCY, RECRUITER, INTERVIEWER, CANDIDATE, ENGINEER, SOFTWARE CODE EDITING

Пояснювальна записка
до дипломного проєкту

на тему: Платформа для проведення технічних інтерв’ю (комплексна тема). Система організації та проведення інтерв’ю.

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CRUD (Create Read Update Delete) – термін для позначення базових операцій над даними, що включає в себе створення, зчитування, оновлення та видалення даних.

REST – архітектурний підхід створення програмного інтерфейсу для веб-сервісів.

Чернетка – відгук на кандидата, який є незавершеною версією, не призначеною для аналізу рекрутером.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Змістовний опис і аналіз предметної області

Діаграма варіантів використань зображена на рисунку 1.1.

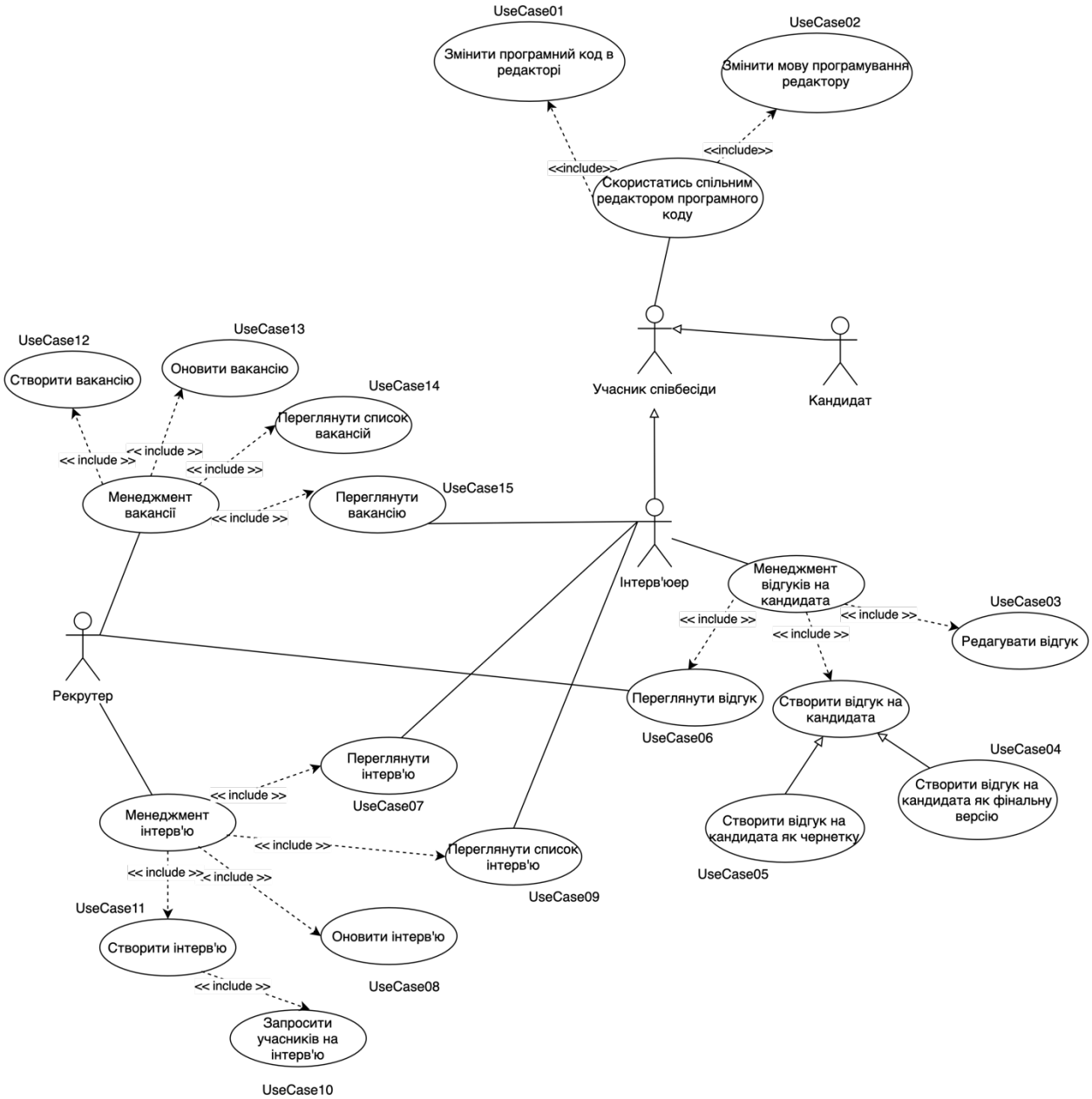


Рисунок 1.1 - Діаграма варіантів використань

Таблиця 1.1 – Варіант використання UseCase01

Назва	Зміна програмного коду в редакторі
Опис	Користувач має можливість змінити текст, що відображається в редакторі коду. При редагуванні коду користувачем усі учасники інтерв'ю отримують сповіщення про зміну та отримують найновішу версію програмного коду.
Учасники	Інтерв'юер, кандидат.
Передумови	Користувач приєднався до інтерв'ю.
Постумови	Програмний код в редакторі коду змінений в всіх учасників
Основний сценарій	а) Користувач під'єднюється до інтерв'ю; б) Система відображає сторінку проведення інтерв'ю; в) Користувач натискає кнопку «Редактор коду»; г) Система відображає вікно з редактором коду; ґ) Користувач вводить в текстовому полі текст, що слугує програмним кодом; д) Система зберігає останню версію текстового поля, надсилає всім учасникам співбесіди останній варіант та відображає його.
Розширення сценарію	

Таблиця 1.2 – Варіант використання UseCase02

Назва	Зміна мови програмування редактору
Опис	Користувач має можливість змінити мову програмування для редактору коду. При зміні мови користувачем усі учасники інтерв'ю отримують про це сповіщення та отримують найновішу версію мови програмування
Учасники	Інтерв'юер, кандидат.
Передумови	Користувач приєднався до інтерв'ю.
Постумови	Мова програмування в редакторі коду змінена в всіх учасників
Основний сценарій	а) Користувач під'єднується до інтерв'ю; б) Система відображає сторінку проведення інтерв'ю; в) Користувач натискає кнопку «Редактор коду»; г) Система відображає вікно з редактором коду; ґ) Користувач обирає необхідну мову програмування для коректної підсвітки програмного коду; д) Система зберігає останній вибір мови програмування, надсилає всім учасникам співбесіди та відображає його.
Розширення сценарію	

Таблиця 1.3 – Варіант використання UseCase03

Назва	Редагування відгуку
Опис	Користувач має можливість оновлювати дані раніше створеного відгуку
Учасники	Інтерв'юер
Передумови	Користувач авторизований в системі як інтерв'юер
Постумови	Оновлена версія відгуку збережена в системі
Основний сценарій	а) Користувач заходить на сторінку перегляду власного відгуку; б) Система відображає сторінку відгуку; в) Користувач натискає кнопку «Редагувати»; г) Система відображає сторінку редагування відгуку; г) Користувач змінює необхідні поля та натискає кнопку «Зберегти»; д) Система перевіряє введені дані та зберігає останню версію відгуку в системі. Система відображає для користувача сторінку перегляду відгуку з найновішими даними;

Продовження таблиці 1.3

Розширення сценарію	д1) Користувач ввів некоректні дані; а) Система відображає відповідні помилки та продовжує відображати сторінку редагування інтерв'ю.

Таблиця 1.4 – Варіант використання UseCase04

Назва	Створення фінальної версії відгуку
Опис	Користувач має можливість зберегти відгук на кандидата, вказавши, що дана версія є фінальною, і її можна використовувати для оцінювання кандидата;
Учасники	Інтерв'юер
Передумови	Користувач авторизований в системі як інтерв'юер
Постумови	Новий відгук на кандидата з приміткою про те, що це фінальна версія збережена в системі
Основний сценарій	а) Користувач заходить на сторінку проведення інтерв'ю та натискає кнопку «Завершити»; б) Система відображає сторінку створення відгуку; в) Користувач заповнює всі необхідні дані та натискає кнопку «Зберегти»; г) Система перевіряє, чи всі дані є коректними та зберігає відгук в

Продовження таблиці 1.4

	системі з відміткою про те, що це є фінальною версією відгуку. Система відображає сторінку перегляду відгуку.
Розширення сценарію	г1) Користувач ввів некоректні дані; а) Система відображає відповідні помилки та продовжує відображати сторінку створення інтерв'ю.

Таблиця 1.5 – Варіант використання UseCase05

Назва	Створення відгуку на кандидата як чернетка
Опис	Користувач має можливість зберегти відгук на кандидата, вказавши, що дана версія є чернеткою, і її ще не можна використовувати для оцінювання кандидата;
Учасники	Інтерв'юер
Передумови	Користувач авторизований в системі як інтерв'юер
Постумови	Новий відгук з приміткою про те, що дана версія є чернеткою, збережений в системі

Продовження таблиці 1.5

Основний сценарій	а) Користувач заходить на сторінку проведення інтерв'ю та натискає кнопку «Завершити»; б) Система відображає сторінку створення відгуку; в) Користувач заповнює всі необхідні дані та натискає кнопку «Зберегти як чернетку»; г) Система зберігає відгук в системі з відміткою про те, що це не є фінальною версією відгуку. Система відображає сторінку перегляду відгуку.
Розширення сценарію	

Таблиця 1.6 – Варіант використання UseCase06

Назва	Перегляд відгуку
Опис	Користувач має можливість переглянути заповнений інтерв'юером відгук на кандидата
Учасники	Інтерв'юер, рекрутер
Передумови	Користувач авторизований в мережі як рекрутер або інтерв'юер та є автором або учасником інтерв'ю, де вже існує заповнений відгук

Продовження таблиці 1.6

Постумови	Користувач переглянув дані, заповнені в відгуці на кандидата
Основний сценарій	а) Користувач відкриває сторінку перегляду інтерв’ю; б) Система відображає сторінку перегляду інтерв’ю; в) Користувач обирає зі списку відгуків необхідний відгук та натискає на нього; г) Система відображає сторінку перегляду відгуку, де містяться всі дані, заповнені інтерв’юером, що є автором даного відгуку.
Розширення сценарію	

Таблиця 1.7 – Варіант використання UseCase07

Назва	Перегляд інтерв’ю
Опис	Користувач має можливість переглянути створене рекрутером інтерв’ю
Учасники	Інтерв’юер, рекрутер
Передумови	Користувач авторизований в мережі як рекрутер або інтерв’юер та є автором або учасником інтерв’ю
Постумови	Користувач переглянув дані про створене рекрутером інтерв’ю

Продовження таблиці 1.7

Основний сценарій	а) Користувач авторизований з роллю рекрутер відкриває сторінку перегляду вакансії; б) Система відображає сторінку перегляду вакансії; в) Користувач обирає зі списку інтерв'ю необхідне інтерв'ю та натискає на нього; г) Система відображає сторінку перегляду інтерв'ю, де містяться всі дані, заповнені рекрутером, що є автором даного відгуку, а також список відгуків, які існують для даного інтерв'ю, та дані про вакансію, до якої належить це інтерв'ю.
Розширення сценарію	а1) Користувач авторизований з роллю інтерв'юер відкриває сторінку облікового запису; а) Система відображає сторінку облікового запису; б) Користувач обирає зі списку інтерв'ю, де він є учасником, необхідне інтерв'ю та натискає на нього; в) Система відображає сторінку перегляду інтерв'ю, де містяться всі дані, заповнені рекрутером, що є автором

Продовження таблиці 1.7

	даного відгуку, а також список відгуків, які існують для даного інтерв'ю, та дані про вакансію, до якої належить це інтерв'ю.

Таблиця 1.8 – Варіант використання UseCase08

Назва	Оновлення інтерв'ю
Опис	Користувач має можливість
Учасники	Рекрутер
Передумови	Користувач авторизований в системі як рекрутер
Постумови	Оновлена версія інтерв'ю зберігається в системі
Основний сценарій	а) Користувач заходить на сторінку перегляду власного інтерв'ю; б) Система відображає сторінку інтерв'ю; в) Користувач натискає кнопку «Редагувати»; г) Система відображає сторінку редагування інтерв'ю; ґ) Користувач змінює необхідні поля та натискає кнопку «Зберегти»; д) Система перевіряє введені дані та зберігає останню версію інтерв'ю в системі. Система відображає для користувача сторінку перегляду інтерв'ю з найновішими даними.

Продовження таблиці 1.8

Розширення сценарію	д1) Користувач ввів некоректні дані; а) Система відображає відповідні помилки та продовжує відображати сторінку редагування інтерв’ю.

Таблиця 1.9 – Варіант використання UseCase09

Назва	Перегляд списку інтерв’ю
Опис	Користувач має можливість переглянути список доступних йому інтерв’ю
Учасники	Інтерв’юер, рекрутер
Передумови	Користувач зареєстрований в системі з роллю інтерв’юер або рекрутер, та має список інтерв’ю, де він є автором або учасником
Постумови	Користувач переглянув список доступних для нього інтерв’ю
Основний сценарій	а) Користувач авторизований з роллю рекрутер відкриває сторінку перегляду вакансії; б) Система відображає сторінку перегляду вакансії; в) Користувач відкриває вкладку інтерв’ю; г) Система відображає список інтерв’ю, де користувач є автором.

Продовження таблиці 1.9

Розширення сценарію	а1) Користувач авторизований з роллю інтерв'юер відкриває сторінку облікового запису; а) Система відображає сторінку облікового запису; б) Користувач відкриває вкладку інтерв'ю; в) Система відображає список інтерв'ю, де користувач є учасником.

Таблиця 1.10 – Варіант використання UseCase10

Назва	Запрошення учасників на інтерв'ю
Опис	Користувач має можливість запросити учасників інтерв'ю під час його створення на саме інтерв'ю шляхом надсилання запрошень на електронну пошту
Учасники	Рекрутер
Передумови	Користувач зареєстрований в системі як рекрутер, має створену вакансію
Постумови	Усі учасники інтерв'ю отримали запрошення на інтерв'ю

Продовження таблиці 1.10

Основний сценарій	а) Користувач відкриває сторінку перегляду вакансії; б) Система відображає сторінку перегляду вакансії; в) Користувач натискає кнопку «Нове інтерв’ю»; г) Система відкриває сторінку створення інтерв’ю; г) Користувач заповнює всі необхідні дані про кандидата, та вказує його електронну пошту. Натискає кнопку «Далі»; д) Система перевіряє введені дані та відображає вікно з наступним етапом створення інтерв’ю; е) Користувач заповнює всі необхідні поля, та вводить дані про інтерв’юерів. Натискає кнопку «Далі»; є) Система перевіряє введені дані та відображає вікно з наступним етапом створення інтерв’ю, а саме його переглядом; ж) Користувач натискає кнопку «Зберегти»; з) Система зберігає інтерв’ю в системі та надсилає запрошення на електронну пошту всім учасникам

Продовження таблиці 1.10

	інтерв'ю. Відображає сторінку перегляду інтерв'ю.
Розширення сценарію	д1) Користувач ввів некоректні дані; а) Система відображає відповідні повідомлення про помилку та продовжує відображати поточне вікно; е1) Користувач ввів некоректні дані; а) Система відображає відповідні повідомлення про помилку та продовжує відображати поточне вікно.

Таблиця 1.11 – Варіант використання UseCase11

Назва	Створення інтерв'ю
Опис	Користувач має можливість створити інтерв'ю для існуючої вакансії
Учасники	Рекрутер
Передумови	Користувач зареєстрований в системі як рекрутер, має створену вакансію
Постумови	Нове інтерв'ю для раніше створеної вакансії добавлене в систему

Продовження таблиці 1.11

Основний сценарій	а) Користувач відкриває сторінку перегляду вакансії; б) Система відображає сторінку перегляду вакансії; в) Користувач натискає кнопку «Нове інтерв’ю»; г) Система відкриває сторінку створення інтерв’ю; г) Користувач заповнює всі необхідні дані про кандидата. Натискає кнопку «Далі»; д) Система перевіряє введені дані та відображає вікно з наступним етапом створення інтерв’ю; е) Користувач заповнює всі необхідні поля. Натискає кнопку «Далі»; є) Система перевіряє введені дані та відображає вікно з наступним етапом створення інтерв’ю, а саме його переглядом; ж) Користувач натискає кнопку «Зберегти»; з) Система зберігає інтерв’ю в системі. Відображає сторінку перегляду новоствореного інтерв’ю.

Продовження таблиці 1.11

Розширення сценарію	д1) Користувач ввів некоректні дані; а) Система відображає відповідні повідомлення про помилку та продовжує відображати поточне вікно; е1) Користувач ввів некоректні дані; а) Система відображає відповідні повідомлення про помилку та продовжує відображати поточне вікно.

Таблиця 1.12 – Варіант використання UseCase12

Назва	Створення вакансії
Опис	Користувач має можливість створити нову вакансію
Учасники	Рекрутер
Передумови	Користувач зареєстрований в системі як рекрутер
Постумови	Нова вакансія в системі
Основний сценарій	а) Користувач відкриває сторінку облікового запису; б) Система відображає сторінку перегляду облікового запису; в) Користувач відкриває вкладку зі списком вакансій; г) Система відображає сторінку перегляду уже існуючих вакансій;

Продовження таблиці 1.12

	г) Користувач натискає на кнопку «Нова вакансія»; д) Система відображає сторінку створення вакансії; е) Користувач заповнює всі необхідні дані та натискає кнопку «Зберегти» ; є) Система перевіряє, чи дані коректні та створює нову вакансію в системі. Відображає сторінку перегляду щойно створеної вакансії.
Розширення сценарію	є1) Користувач ввів некоректні дані; а) Система відображає відповідні повідомлення про помилку та продовжує відображати поточне вікно.

Таблиця 1.13 – Варіант використання UseCase13

Назва	Оновлення вакансії
Опис	Користувач має можливість редагувати дані уже існуючої вакансії
Учасники	Рекрутер
Передумови	Користувач зареєстрований в системі як рекрутер
Постумови	Оновлена версія вакансії зберігається в системі

Продовження таблиці 1.13

Основний сценарій	а) Користувач заходить на сторінку перегляду вакансії; б) Система відображає сторінку вакансії; в) Користувач натискає кнопку «Редагувати»; г) Система відображає сторінку редагування вакансії; ґ) Користувач змінює необхідні поля та натискає кнопку «Зберегти»; д) Система перевіряє введені дані та зберігає останню версію вакансії в системі. Система відображає для користувача сторінку перегляду вакансії з найновішими даними.
Розширення сценарію	д1) Користувач ввів некоректні дані; а) Система відображає відповідні помилки та продовжує відображати сторінку редагування вакансії.

Таблиця 1.14 – Варіант використання UseCase14

Назва	Перегляд списку вакансії
Опис	Користувач має можливість
Учасники	Інтерв’юер, кандидат.
Передумови	Користувач зареєстрований в системі з роллю інтерв’юер

Продовження таблиці 1.14

Постумови	Користувач переглянув список доступних для нього вакансій
Основний сценарій	а) Користувач відкриває сторінку облікового запису; б) Система відображає сторінку облікового запису; в) Користувач відкриває вкладку вакансії; г) Система відображає список вакансій, де користувач є автором.
Розширення сценарію	г1) Користувач не створив жодного інтерв'ю; а) Система відображає повідомлення про те, що в користувача ще немає вакансій.

Таблиця 1.15 – Варіант використання UseCase15

Назва	Перегляд вакансії
Опис	Користувач має можливість переглянути власно створену вакансію
Учасники	Рекрутер
Передумови	Користувач авторизований в мережі як рекрутер та має створену вакансію
Постумови	Користувач переглянув дані про створену вакансію

Продовження таблиці 1.15

Основний сценарій	а) Користувач авторизований з роллю рекрутер відкриває сторінку облікового запису; б) Система відображає сторінку облікового запису; в) Користувач обирає зі списку вакансій необхідну вакансію та натискає на неї; г) Система відображає сторінку перегляду вакансії, де містяться всі дані, раніше заповнені користувачем, що є автором даного відгуку, а також список інтерв'ю, які існують для даної вакансії.
Розширення сценарію	

1.2 Аналіз вимог до програмного забезпечення

1.2.1 Розроблення функціональних вимог

Таблиця 1.16 – Опис функціональної вимоги REQ01

Номер	REQ01
Назва	Відображення списку вакансій
Опис	Користувач з роллю рекрутер має можливість переглянути список створених ним вакансій, з інформацією про назву вакансії, її

Продовження таблиці 1.16

	статус, а також час створення та її останнього редагування.

Таблиця 1.17 – Опис функціональної вимоги REQ02

Номер	REQ02
Назва	Відкриття сторінки перегляду вакансії
Опис	Користувач з роллю рекрутер має можливість перейти на сторінку перегляду вакансії, обравши її в списку створених ним вакансій.

Таблиця 1.18 – Опис функціональної вимоги REQ03

Номер	REQ03
Назва	Сторінка перегляду вакансії
Опис	Користувач з роллю рекрутер має можливість переглядати деталі створеної ним вакансії. Йому доступна заповнена ним інформація про назву вакансії, посилання на вакансію на інших ресурсах, опис вакансії, а також список навичок, необхідних для кандидата для заданої вакансії.

Таблиця 1.19 – Опис функціональної вимоги REQ04

Номер	REQ04
Назва	Створення вакансії

Продовження таблиці 1.19

Опис	Користувач з роллю рекрутер має можливість створити нову вакансію, натиснувши кнопку «Нова вакансія» в особистому кабінеті користувача. Після заповнення необхідних даних та їх збереження користувач може створити в системі нову вакансію.

Таблиця 1.20 – Опис функціональної вимоги REQ05

Номер	REQ05
Назва	Введення даних під час створення нової вакансії
Опис	Під час створення вакансії є можливість вводити дані про вакансію в відповідні поля. Серед таких даних є назва, посилання на вакансію на інших ресурсах, опис вакансії, а також список необхідних навичок, які потрібно перевірити в кандидата під час інтерв’ю.

Таблиця 1.21 – Опис функціональної вимоги REQ06

Номер	REQ06
Назва	Закриття вакансії

Продовження таблиці 1.21

Опис	В користувача з роллю рекрутер є можливість закрити вакансію, натиснувши кнопку «Закрити». Дана дія змінить статус самої вакансії.

Таблиця 1.22 – Опис функціональної вимоги REQ07

Номер	REQ07
Назва	Редагування вакансії
Опис	В користувача з роллю рекрутер є можливість редагувати вакансію, натиснувши кнопку «Редагувати» на сторінці перегляду вакансії.

Таблиця 1.23 – Опис функціональної вимоги REQ08

Номер	REQ08
Назва	Редагування даних вакансії
Опис	Під час редагування вакансії є можливість змінювати всі дані про вакансію. Серед таких даних є назва, посилання на вакансію на інших ресурсах, опис вакансії, а також список необхідних навичок, які потрібно перевірити в кандидата під час інтерв’ю.

Таблиця 1.24 – Опис функціональної вимоги REQ09

Номер	REQ09
Назва	Відображення списку інтерв’ю
Опис	Користувач з роллю рекрутер має можливість переглянути список створених ним інтерв’ю для заданої вакансії, з інформацією про ім’я кандидата, загальну оцінку успішності кандидата, загальний рівень професійності кандидата, статус інтерв’ю та його час проведення.

Таблиця 1.25 – Опис функціональної вимоги REQ10

Номер	REQ10
Назва	Створення інтерв’ю
Опис	Користувач з роллю рекрутер має можливість створити нове інтерв’ю, натиснувши кнопку «Нове інтерв’ю» на сторінці перегляду вакансії. Після заповнення необхідних даних та їх збереження користувач може створити в системі нове інтерв’ю.

Таблиця 1.26 – Опис функціональної вимоги REQ11

Номер	REQ11
Назва	Введення даних про кандидата під час створення нового інтерв’ю

Продовження таблиці 1.26

Опис	Під час створення інтерв'ю є можливість вводити дані про кандидата, а саме: ім'я, прізвище, та електронна пошта кандидата. Дана електронна пошта може бути використаною для надсилання запрошення на інтерв'ю.

Таблиця 1.27 – Опис функціональної вимоги REQ12

Номер	REQ12
Назва	Введення даних про інтерв'ю під час створення нового інтерв'ю
Опис	Під час створення інтерв'ю після заповнення даних про кандидата, користувачу з'являється можливість ввести дані про саме інтерв'ю, такі як: імена інтерв'юерів, час початку та кінця інтерв'ю. При заповненні спеціального поля інтерв'юерів користувач може почати вводити ім'я або адресу електронної пошти інтерв'юера, а система сама підкаже користувачів, які задовольняють даному пошуку.

Таблиця 1.28 – Опис функціональної вимоги REQ13

Номер	REQ13
Назва	Перегляд заповнених даних під час створення інтерв’ю
Опис	Під час створення інтерв’ю після заповнення всіх необхідних даних в користувача є можливість переглянути фінальні дані, що стосуються інтерв’ю, а також відповідної вакансії.

Таблиця 1.29 – Опис функціональної вимоги REQ14

Номер	REQ14
Назва	Збереження даних для нового інтерв’ю
Опис	Під час створення інтерв’ю після перегляду його фінальної версії користувач може або натиснути кнопку «Зберегти», та створити в системі нове інтерв’ю, або повернутися назад, змінити певні дані, та зберегти інтерв’ю після цього.

Таблиця 1.30 – Опис функціональної вимоги REQ15

Номер	REQ15
Назва	Відкриття сторінки перегляду інтерв’ю

Продовження таблиці 1.30

Опис	Користувач з роллю рекрутер має можливість перейти на сторінку перегляду інтерв'ю, обравши її в списку інтерв'ю для раніше створеної ним вакансії .

Таблиця 1.31 – Опис функціональної вимоги REQ16

Номер	REQ16
Назва	Сторінка перегляду інтерв'ю
Опис	Користувач з роллю рекрутер має можливість переглядати деталі створеного ним інтерв'ю. Йому доступна заповнена ним інформація про ім'я, прізвище та електронну пошту кандидата, імена та адреси електронних пошт інтерв'юерів, а також час та дата початку інтерв'ю. Також користувач має можливість додатково глянути на інформацію про вакансію.

Таблиця 1.32 – Опис функціональної вимоги REQ17

Номер	REQ17
Назва	Сторінка перегляду інтерв'ю: фінальний результат

Продовження таблиці 1.32

Опис	Користувач з роллю рекрутер після проведення інтерв'ю має можливість переглядати деталі створеного ним інтерв'ю разом з інформацією про результати інтерв'ю, яка включає оцінку успішності кандидата, його професійний рівень, та оцінку кожної з обов'язкових навичок заданої вакансії.

Таблиця 1.33 – Опис функціональної вимоги REQ18

Номер	REQ18
Назва	Редагування інтерв'ю
Опис	В користувача з роллю рекрутер є можливість редагувати створене ним інтерв'ю, натиснувши кнопку «Редагувати» на сторінці перегляду інтерв'ю.

Таблиця 1.34 – Опис функціональної вимоги REQ19

Номер	REQ19
Назва	Редагування даних інтерв'ю
Опис	Під час редагування інтерв'ю є можливість змінювати всі дані про інтерв'ю. Серед таких даних є ім'я, прізвище, та електронна пошта кандидата, а також імена

Продовження таблиці 1.34

	інтерв’юерів, час початку та кінця інтерв’ю.

Таблиця 1.35 – Опис функціональної вимоги REQ20

Номер	REQ20
Назва	Список відгуків
Опис	Користувач з роллю рекрутер чи інтерв’юер після проведення інтерв’ю має можливість переглядати список відгуків на кандидата, з інформацією про ім’я інтерв’юера, що написав відгук, статус відгука, оцінку успішності кандидата, його професійний рівень, час створення відгуку.

Таблиця 1.36 – Опис функціональної вимоги REQ21

Номер	REQ21
Назва	Сторінка перегляду відгуку
Опис	Користувач з роллю інтерв’юер має можливість переглядати деталі створеного ним відгуку. Користувач з роллю рекрутер має можливість переглядати відгук для створеного ним інтерв’ю. Користувачу доступна інформація про ім’я кандидата, його професійний рівень, оцінку успішності, сформовану на основі

Продовження таблиці 1.36

	інтерв'ю, а також детальний текстовий відгук. Також можна переглянути оцінку кожної з необхідних навичок для даної вакансії.

Таблиця 1.37 – Опис функціональної вимоги REQ22

Номер	REQ22
Назва	Створення відгуку як чернетка
Опис	Користувач з роллю інтерв'юер після завершення інтерв'ю має можливість створити відгук на кандидата. Після заповнення необхідних даних та їх збереження в чорновому варіанті за допомогою кнопки «Зберегти як чернетка» користувач може створити в системі новий відгук, який ще не завершений.

Таблиця 1.38 – Опис функціональної вимоги REQ23

Номер	REQ23
Назва	Створення фінального відгуку
Опис	Користувач з роллю інтерв'юер після завершення інтерв'ю має можливість створити відгук на кандидата. Після заповнення необхідних даних та їх збереження в фінальному варіанті за допомогою кнопки «Зберегти»

Продовження таблиці 1.38

	користувач може створити в системі новий відгук, який вже є завершеним.

Таблиця 1.39 – Опис функціональної вимоги REQ24

Номер	REQ24
Назва	Введення даних під час створення нового відгуку
Опис	Під час створення відгуку є можливість вводити дані про успішність кандидата на інтерв’ю, професійний рівень кандидата, а також обрати оцінку для кожної з необхідних для даної вакансії навичок. Також можна заповнити текстове поле відгуку, з метою детально описати про результат кандидата на інтерв’ю.

Таблиця 1.40 – Опис функціональної вимоги REQ25

Номер	REQ25
Назва	Редагування відгуку
Опис	В користувача з роллю інтерв’юер є можливість редагувати відгук, створений ним раніше, натиснувши кнопку «Редагувати» на сторінці перегляду інтерв’ю.

Таблиця 1.41 – Опис функціональної вимоги REQ26

Номер	REQ26
Назва	Редагування даних відгуку
Опис	Під час редагування відгуку є можливість змінювати всі дані про відгук. Серед таких даних є оцінка успішності кандидата, його професійний рівень, текстовий детальний відгук, а також оцінку для кожної з навичок, необхідних для даної вакансії.

Таблиця 1.42 – Опис функціональної вимоги REQ27

Номер	REQ27
Назва	Зміна програмного коду в редакторі
Опис	Під час комунікації з іншими учасниками інтерв’ю користувач має можливість змінювати вміст програмного коду в вікні з редактором коду для всіх учасників інтерв’ю.

Таблиця 1.43 – Опис функціональної вимоги REQ28

Номер	REQ28
Назва	Зміна мови програмування в редакторі

Продовження таблиці 1.43

Опис	Під час комунікації з іншими учасниками інтерв'ю користувач має можливість змінювати обрану мову програмування в редакторі коду для всіх учасників інтерв'ю. Обрана мова програмування впливає на підсвічення синтаксису програмного коду.

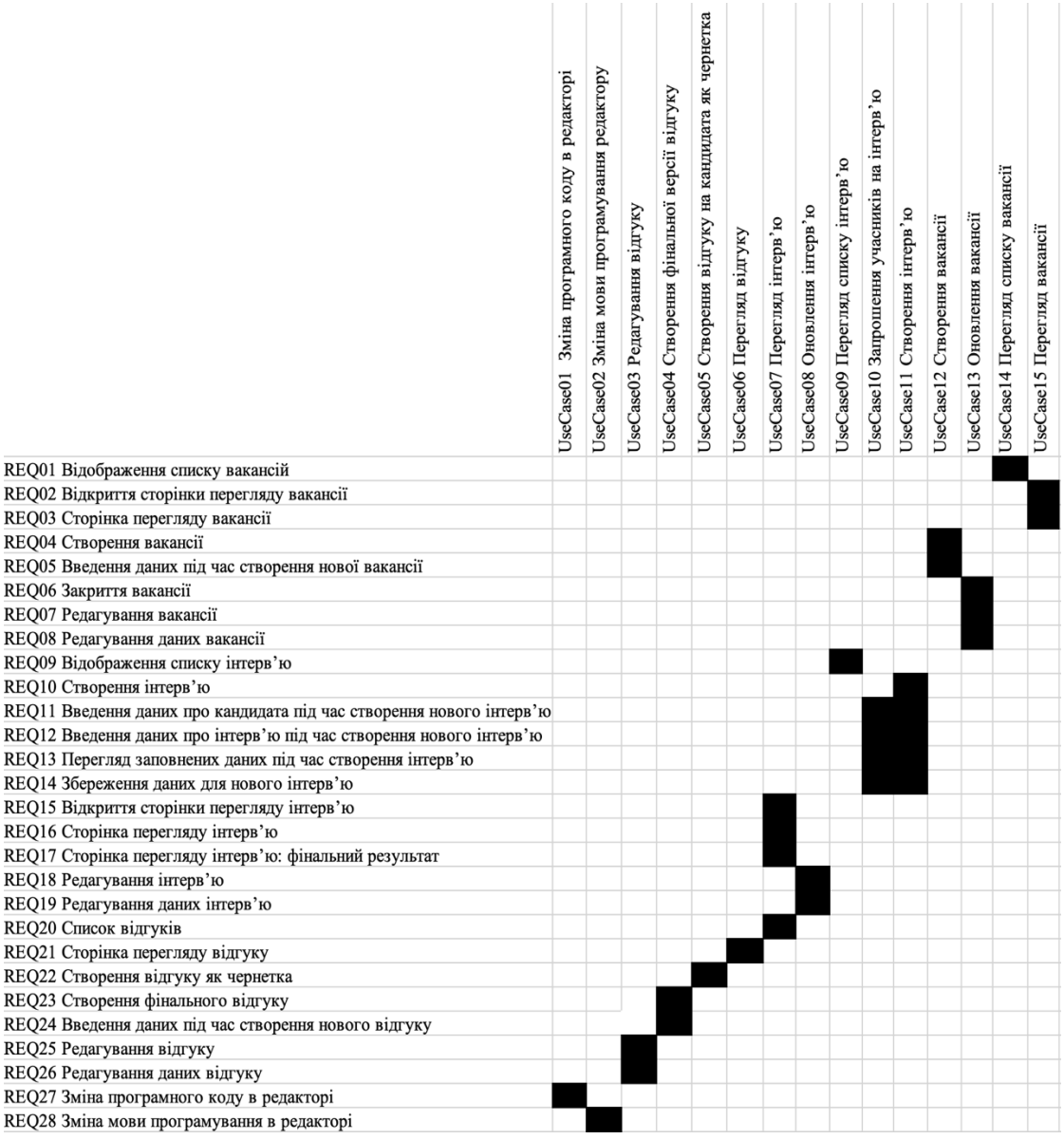


Рисунок 1.2 – Матриця трасування

1.2.2 Постановка комплексу завдань модулю

Для забезпечення функціонування всіх необхідних вимог в рамках індивідуальної частини дипломного проєкту необхідно виконати наступні задачі:

– реалізувати можливість створення вакансії для користувачів з роллю «Рекрутер» ;

– реалізувати можливість оновлення даних вакансії, що була створена раніше;

– реалізувати функціонал для перегляду деталей вакансії;

– реалізувати можливість створення інтерв’ю на основі вакансії;

– реалізувати можливість оновлення даних інтерв’ю, що було створено раніше;

– реалізувати функціонал для перегляду деталей інтерв’ю;

– реалізувати відображення інтерв’ю, які було створено на основі вакансій;

– реалізувати надсилання учасникам інтерв’ю запрошень з деталями інтерв’ю за допомогою електронної пошти;

– реалізувати можливість створювати відгук на кандидата;

– реалізувати можливість змінювати дані відгуку, що був створений раніше;

– реалізувати функціонал для перегляду відгуку на кандидата;

– реалізувати обчислення середніх оцінок на основі всіх відгуків про кандидата;

– реалізувати можливість спільного редагування програмного коду під час інтерв’ю;

– реалізувати можливість вибирати мову програмування для підсвітки синтаксису в спільному редакторі коду.

1.3 Висновки по розділу

В даному розділі було сформульовано основні функціональні вимоги до системи менеджменту і проведення співбесіди, також був визначений перелік задач для дипломного проєкту.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Після проведення в загальній частині диплому аналізу можна визначити, що система організації та проведення співбесіди повинна виконувати наступні задачі:

– створення вакансії (якщо користувач рекрутер);
– оновлення вакансії;
– перегляд деталей вакансії;
– перегляд інтерв’ю, які створено на основі вакансії;
– створення інтерв’ю;
– оновлення даних про інтерв’ю;
– перегляд інформації про інтерв’ю;
– надсилання запрошень учасникам співбесіди за допомогою електронної пошти;
– створення відгуків на кандидата;
– оновлення відгуків на кандидата;
– перегляд відгуків на кандидата;
– обрахування середніх оцінок кандидата на основі відгуків від всіх інтерв’юерів;
– надання можливості спільного редагування програмного коду.

Створення вакансії доступне тільки користувач з роллю рекрутер. При створенні нової вакансії користувач вказує назву, посилання на сторонній ресурс в разі необхідності (наприклад на внутрішню сторінку компанії з описом вакансії або на сайт з пошуку роботи, де є описаною дана вакансія), текстовий опис вакансії, де можуть описуватись вимоги та побажання до кандидата, його потенційна роль в компанії та ін., а також визначається перелік вимог, які будуть оцінювати інтерв’юери в своїх відгуках. Після заповнення всіх необхідних даних

в клієнтському застосунку відбувається перевірка, чи обов’язкові поля, а саме назва вакансії, її опис, а також список вимог до кандидата, не порожні і відбувається запит на сервер. На сервері виконується додаткова перевірка отриманих даних і в разі успіху створюється запис про вакансію. Після цього на основі даної вакансії можна створювати інтерв’ю.

Детальна діаграма даного бізнес процесу наведена на рисунку 2.1

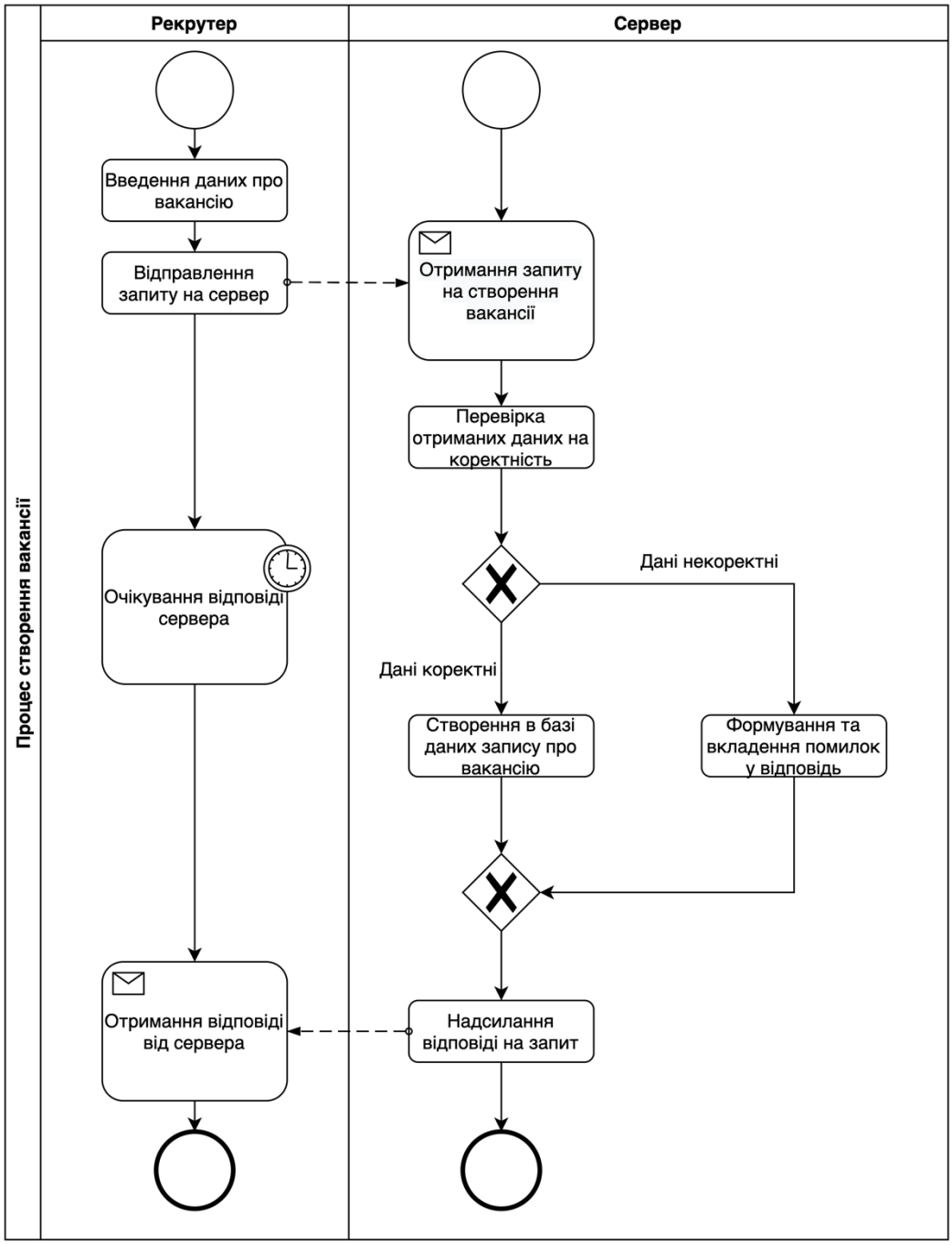


Рисунок 2.1 – діаграма процесу створення співбесіди

Створення інтерв’ю складається з кількох етапів: вибір вакансії, на основі якої буде створюватись інтерв’ю, заповнення даних про кандидата, вибір часу та дати проведення співбесіди, вибір інтерв’юерів серед користувачів системи. Створення співбесіди доступне тільки користувачам з роллю Рекрутер. Спочатку рекрутер повинен авторизуватись в системі та відкрити сторінку з переліком раніше створених вакансій, після чого обрати вакансію для створення співбесіди. На цьому етапі створення інтерв’ю користувач заповнює дані про кандидата, а саме його ім’я, прізвище та електронну пошту. Далі кандидат переходить на наступний крок – вибір дати та часу проведення співбесіди. При цьому він повинен вибрати тільки дату з майбутнього, і дата та час завершення інтерв’ю не повинна бути до дати і часу початку інтерв’ю. Після цього користувач переходить на наступний крок – вибір інтерв’юерів. Йому доступне поле для вводу тексту пошуку, куди користувач може ввести частину імені бажаного інтерв’юера або частину електронної пошти, після чого відбувається запит на сервер, де виконується пошук інтерв’юерів, які підходять під задані умови пошуку і повертаються користувачу. Серед результатів пошуку рекрутер може вибрати бажаних інтерв’юерів. При цьому можна вибрати не більше трьох інтерв’юерів для однієї співбесіди. Коли заповнення даних завершено, відправляється запит на сервер, де перевіряється коректність введення даних, факт існування вибраних інтерв’юерів і в разі успіху створюється запис про інтерв’ю. При цьому формуються запрошення на співбесіди для кандидатів і інтерв’юерів у вигляді електронних листів. В запрошеннях міститься інформація про співбесіду, а саме дата та час проведення, а також посилання для приєднання до співбесіди. Щойно створене інтерв’ю отримує статус «Створено».

Детальна діаграма даного бізнес процесу наведена на рисунку 2.2

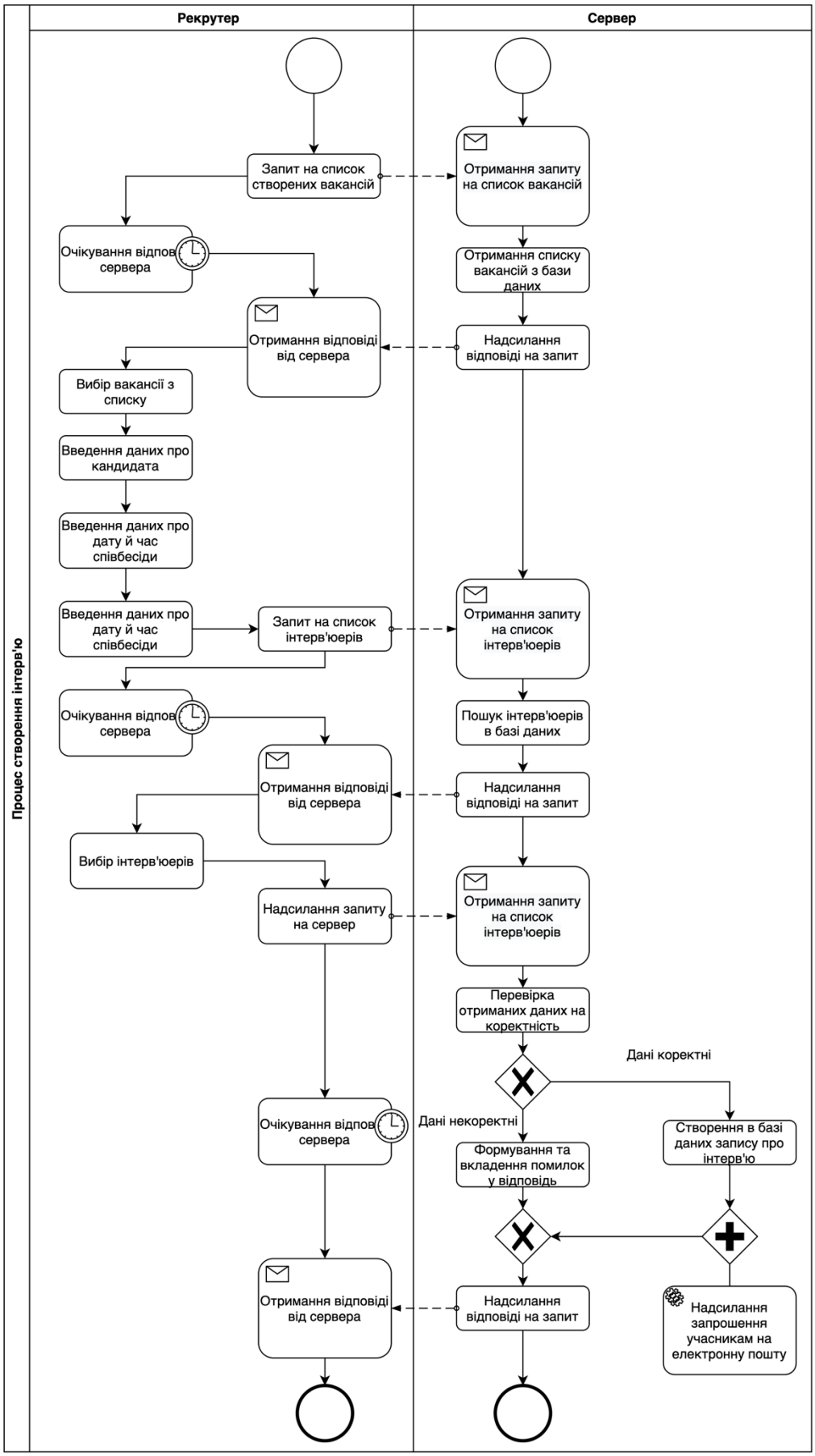


Рисунок 2.2 – Діаграма процесу створення інтерв’ю

Створення відгуку на кандидата починається після закінчення основної частини співбесіди. Коли один з користувачів завершує інтерв'ю – всіх інтерв'юерів клієнтський застосунок перенаправляє на сторінку створення відгуку, при цьому завантажується список вимог, які було перелічено при створенні вакансії, на основі якої було створено інтерв'ю. Користувачам доступно одразу кілька метрик за якими можна оцінити кандидата. Перша метрика - це оцінка професійного рівня кандидата, яка має наступну градацію: Junior, Middle, Senior, Lead. Наступною метрикою є оцінка загального враження від кандидата, яка оцінюється від нуля до 5 балів, з кроком 0.5. Ще однією метрикою оцінки кандидата є оцінка його володіння вимогами, які було перелічено у вакансії. Таким чином інтерв'юер оцінює відповідність кожній вимозі вакансії в градації: початківець, середній рівень, досвідчений. Крім цього інтерв'юеру доступне поле для вводу розгорнутого текстового відгуку. Після заповнення всіх доступних полів відправляється запит на сервер, де дані проходять перевірку та зберігаються. При цьому інтерв'юер може зберегти відгук як чернетку і відредагувати її в майбутньому або зберегти як остаточний варіант. При цьому, якщо всі інтерв'юери створили своїх відгуки у вигляді остаточних версій – співбесіда вважається остаточно завершеною, а рекрутер має змогу провести аналіз всіх відгуків та порівняти їх з відгуками на інших кандидатів, щоб прийняти рішення при найкращого кандидата на ту чи іншу вакансію в компанії.

Детальна діаграма даного бізнес процесу наведена на рисунку 2.3

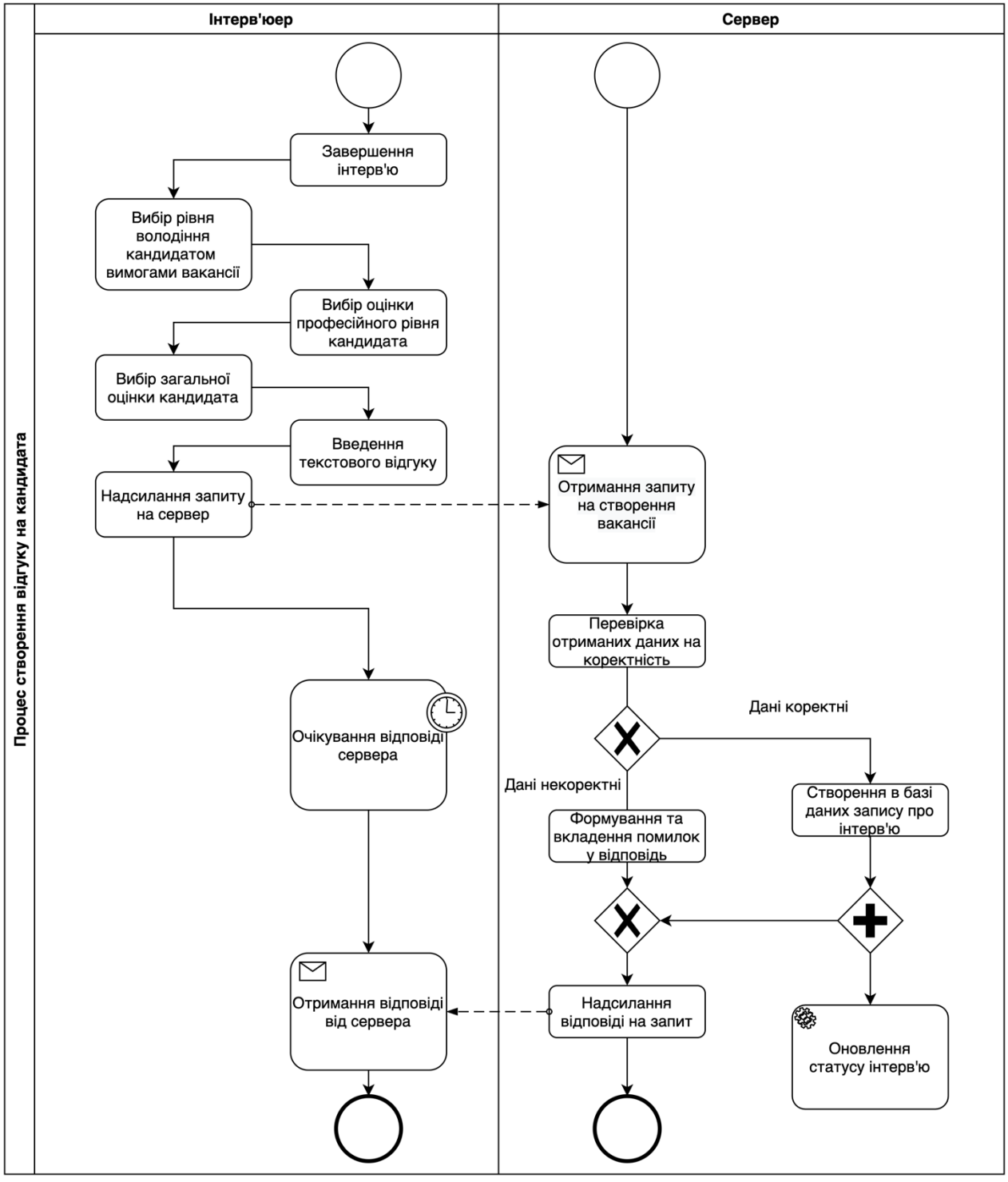


Рисунок 2.3 – процес створення відгуку на кандидата

Процес обрахунку середніх оцінок відгуків на кандидата полягає в калькуляції статистики оцінок одного кандидата за всіма можливими метриками усіма інтерв'юерами, які брали участь в інтерв'ю. При цьому не враховуються метрики, за якими кандидата не було оцінено. Найпростішим обчисленням є обчислення середньої оцінки вражень від кандидата, оскільки для цього потрібно

знайти середнє арифметичне даної метрики у всіх відгуках. Для обрахування середнього професійного рівня використовується підхід з пошуком найнижчої та найвищої серед всіх відгуків оцінки. В даному випадку не можна використовувати звичайний пошук середнього арифметичного, оскільки даний критерій є дуже важливим для оцінки кандидата і не можна допустити його заокруглення в нижчу або вищу сторону. Для пошуку середніх оцінок відповідності вимогам вакансії використовується спосіб, схожий до обрахування середнього професійного рівня, а саме пошук найвищої і найнижчої оцінки за кожною вимогою серед всіх відгуків. На основі цієї статистики будується звіт для рекрутера, який він може використати при ознайомленні з результатами співбесіди, порівнянні різних кандидатів та прийнятті рішення щодо найму конкретного кандидата.

Детальна діаграма даного бізнес процесу наведена на рисунку 2.4

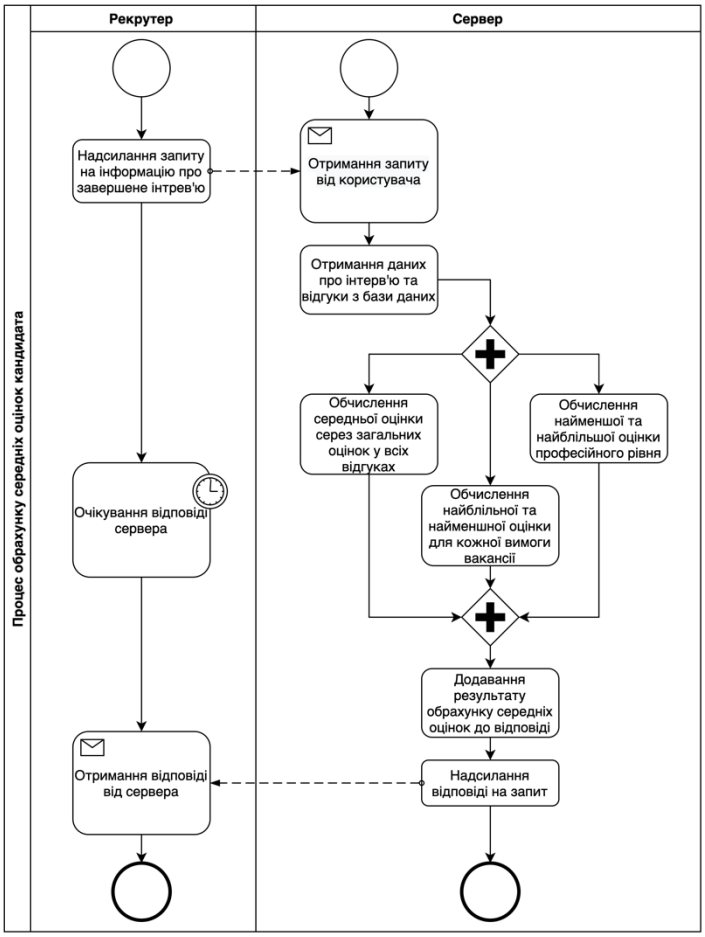


Рисунок 2.4 – процес обчислення середньої оцінки інтерв’ю

2.2 Архітектура програмного забезпечення

Основними підсистемами даної частини дипломного проєкту є: підсистема менеджменту інтерв'ю (повністю), підсистема комунікації між користувачами під час співбесіди (частково) та підсистема користувацького інтерфейсу (частково).

2.2.1 Архітектура підсистеми менеджменту інтерв'ю

Сервіс менеджменту інтерв'ю є однією з найважливіших частин загальної системи, оскільки тут зосереджена основна логіка пов'язана з вакансіями, інтерв'ю та відгуками. Дана підсистема тісно інтегрована з усіма іншими сервісами системи. Вона доступна користувачам через BFF сервіс.

Підсистема менеджменту інтерв'ю реалізована в вигляді REST сервісу, який надає API для інших підсистем на базі протоколу HTTP [1]. Сервіс складається з наступних модулів: модуль роботи з відгуками, модуль інтерв'ю, модуль роботи з вакансіями, модуль спільних функцій, модуль конфігурацій, модуль роботи з базою даних.

Рисунок 2.5 – схема компонентів підсистеми менеджменту інтерв'ю

Таблиця 2.1 – специфікація методів класів підсистеми менеджменту інтерв'ю

Клас	**Метод**	**Призначення**	**Аргументи**
FeedbackService	createEmptyFeedbacksForInterview	Створює новий пустий відгук для кожного з інтерв'юерів, записує в них дані про інтерв'ю, а також необхідні навички, задані в вакансії.	interviewId, interviewersIds, requirements
FeedbackService	createNew	Зберігає заповнений відгук в базу даних.	data
FeedbackService	updateOne	Оновлює відгук згідно редагувань користувача. Оновлює запис в базі даних. Змінює, якщо потрібно, статус відгуку.	data, id
FeedbackService	getOne	Отримує відгук з бази даних за вказаним id.	id
FeedbackService	getOneByInterviewAndAuthor	Отримує відгук з бази даних за вказаним автором відгуку, а також id інтерв'ю.	authorId, interviewId
FeedbackService	getBy	Отримує відгук з бази даних за вказаною умовою.	options
FeedbackService	deleteOne	Видаляє вказаний відгук з бази даних.	id
FeedbackService	search	Виконує пошук по відгуках, повертає отриманий результат.	authorId, interviewId, page, pageSize

Продовження таблиці 2.1

FeedbackService	processSkillsScoresUpdate	Редагує оцінку для обов'язкової для вакансії навички та оновлює ці дані в базі даних.	skills, feedback
InterviewExternalService	completeInterview	Завершує інтерв'ю після того, як всі відгуки, що до нього належать, змінили свій статус на завершений. Зберігає оновлений статус інтерв'ю в базі даних.	id, feedbackId
VacancyRequirementService	getVacancyRequirements	Знаходить необхідні навички для заданої вакансії.	vacancyId
VacancyRequirementService	deleteMany	Видаляє з бази даних задані навички.	requirements
VacancyRequirementService	createMany	Створює навички в форматі необхідних навичок для вакансії.	skills
VacancyService	createNew	Створює нову вакансію та зберігає її в базу даних.	data, authorId
VacancyService	deleteOne	Видаляє існуючу вакансію з бази даних.	id, userId
VacancyService	getOne	Знаходить та повертає вакансію за вказаним id.	id, userId

Продовження таблиці 2.1

VacancyService	Search	Створює пошук серед вакансій за заданими параметрами. Повертає результат пошуку.	params, userId
VacancyService	updateOne	Редагує дані вакансії, згідно з оновленнями заданими користувачем. Оновлює вакансію в базі даних.	id, data, userId
VacancyService	getRequirements	Повертає список необхідних навичок для заданої вакансії.	vacancyId
VacancyService	getUpdatedRequirements	Доповнює список вже існуючих необхідних навичок новими навичками, а також редагує вже існуючі.	currentSkills, skills
VacancyService	getById	Знаходить для заданого користувача вакансію за заданим id.	id, userId
EmailService	sendMessage	Надсилає повідомлення на задану електронну пошту з вказаною темою листа та вказаним контентом.	target, subject, html
ExternalNotifierService	notifyAttendees	Обраховує час тривання співбесіди, надсилає запрошення всім учасникам співбесіди з відповідним для них контентом.	candidate, interviewers, interview

Продовження таблиці 2.1

ExternalNotifierService	notifyInterviewer	Надсилає інтерв'юеру запрошення на інтерв'ю з відповідним посиланням, а також даними про інтерв'ю.	email, name, formattedStartTime, formattedDuration, joiningLink
ExternalNotifierService	notifyCandidate	Надсилає кандидату запрошення на інтерв'ю з відповідним посиланням, а також даними про інтерв'ю.	email, name, interview, userId, formattedStartTime, formattedDuration,
ExternalNotifierService	getExternalJoiningToken	Створює гостьовий токен, який в собі містити задані параметри: id користувача, id інтерв'ю, час закінчення інтерв'ю, а також роль користувача.	userId, role, interview,
ExternalNotifierService	getExternalJoiningLink	Створює посилання для кандидата для переходу на сторінку інтерв'ю.	token
InterviewService	deleteOne	Видаляє задане інтерв'ю з бази даних.	id, userId

Продовження таблиці 2.1

InterviewService	createNew	Створює нове інтерв'ю, використовуючи інформацію з вакансії, для якої це інтерв'ю створюється. Зберігає його в базу даних. Надсилає сповіщення всім учасникам інтерв'ю.	data, userId
InterviewService	updateOne	Оновлює дані інтерв'ю згідно редагувань користувача. Оновлені дані записує в базу даних.	id, data, userId
InterviewService	search	Виконує пошук інтерв'ю за заданими параметрами. Пошук можливий за всіма полями, такими як статус, вакансія, відгуки і т.д. Також фільтрує інтерв'ю за іменем кандидата, статуса і т.д. Повертає знайдений список інтерв'ю, що задовольняють умовам.	searchParams, userId

Продовження таблиці 2.1

InterviewService	calculateInterviewScore	Обраховує фінальну оцінку успішності проходження інтерв'ю кандидата, професійний рівень та оцінку для кожної з обов'язкових навичок для заданої вакансії на основі оцінок, виставлених інтерв'юерами в відгуці. До уваги беруться тільки ті оцінки, які вже виставлені інтерв'юерами.	feedbacks
InterviewService	getOne	Повертає всю інформацію про задане інтерв'ю: загальні дані про інтерв'ю, фінальна оцінка успішності інтерв'ю, дані про вакансію, список учасників інтерв'ю.	id, userId
InterviewService	checkInterviewers	Повертає список інтерв'юерів, що не існує.	equestedInterviewers, existingInterviewers
InterviewService	mapAttendeesDetails	Змінює формат отримання даних учасників інтерв'ю.	usersDetails, attendees
InterviewService	createInterviewers	Створює список учасників інтерв'ю з роллю інтерв'юер.	ids

Продовження таблиці 2.1

InterviewService	getByIdForModification	Повертає дані про інтерв'ю, яке доступне для редагування. Якщо інтерв'ю ще не створене, або його хоче отримати користувач, відмінний від автора самого інтерв'ю, тоді повертає відповідну помилку.	id, userId
InterviewService	getById	Шукає інтерв'ю за його id, або вказаними додатковими вимогами та повертає його.	id, userId, additionalRelations
InterviewService	checkInterviewAccess	Перевіряє, чи інтерв'ю вже створене, та чи користувач має доступ до інтерв'ю: є його автором або учасником.	interview, userId, interviewId
InterviewService	generateInterviewAccessToken	Створює токен доступу до інтерв'ю за відповідними даними про інтерв'ю, а також його тривалістю.	data
InterviewService	getInterviewDuration	Якщо користувач має доступ до інтерв'ю, то обраховує тривалість інтерв'ю, використовуючи час початку і кінця інтерв'ю.	interviewId, userId

Продовження таблиці 2.1

InterviewService	updateStatus	Якщо розпочалося проведення інтерв'ю, то для кожного інтерв'юера створюється пустий відгук з відповідними заповненими даними про інтерв'ю і кандидата. Прикріплює список відгуків до даних про інтерв'ю і зберігає в базі даних.	id: string, status: number
TokenService	getGuestInterviewToken	Створює гостьовий токен, який містить зашифровані задані дані.	data
TokenService	getInterviewAccessToken	Створює токен доступу до інтерв'ю, який містить зашифровані задані дані.	data
TokenService	validateGuestInterviewToken	Перевіряє, чи є гостьовий токен коректний і в разі успіху повертає розшифровані з нього дані. В іншому випадку повертає помилку.	data
UserService	createCandidate	Створює користувача з роллю кандидат.	data
UserService	updateCandidate	Оновлює дані про користувача з роллю кандидат.	Id, data

Продовження таблиці 2.1

UserService	getUsers	Повертає список користувачів з детальною інформацією про них заданими id.	ids, userId

В якості СКБД для даного сервісу використовується реляційна база даних PostgreSQL [2]. Вибір типу бази даних мотивований тим, що даний сервіс зберігає велику кількість різної, чітко структурованої інформації, яка має взаємозалежності. Детальну схему базу даних зображено на графічному матеріалі «Схема база даних».

Таблиця vacancy призначена для зберігання інформації про співбесіду, а саме: ідентифікатор вакансії (id), ідентифікатора користувача що її створив (author_id), назви (title), опису (description), зовнішнього посилання (external_link), дати й часу створення (created_time), дати й часу оновлення (updated_time).

Таблиця vacancy_requirement призначена для зберігання вимог до кандидата які необхідні для певної вакансії. Вона зберігає ідентифікатор вимоги (id), ідентифікатор вакансії (vacancy_id) та назву вимоги (name).

Таблиця interview призначена для зберігання основної інформації про інтерв'ю, а саме: ідентифікатор інтерв'ю (id), ідентифікатор вакансії на основі якої було створено інтерв'ю (vacancy_id), дата й час початку інтерв'ю (start_time), дата й час закінчення інтерв'ю (end_time), прізвище та ім'я кандидата (candidate_name), ідентифікатор статусу інтерв'ю (status_id), дати й часу створення (created_time), дати й часу оновлення (updated_time).

Таблиця interview_status призначена для збереження усіх даних про усі можливі статуси, в яких може перебувати інтерв'ю: ідентифікатор статусу (id), назва статусу (name). Наразі в системі присутні наступні статуси: «Створено

(Created)», «В процесі (In progress)», «Очікування відгуків (Feedback pending)», «Завершено (Completed)».

Таблиця interview_user призначена для того, щоб пов'язати інтерв'ю та його учасників. Для цього в ній зберігається наступна інформація: ідентифікатор запису (id), ідентифікатор користувача (user_id), ідентифікатор ролі користувача на цьому інтерв'ю (role_id), ідентифікатор інтерв'ю (interview_id)

Таблиця interview_user призначена для зберігання ролей користувачів під час інтерв'ю, а саме: ідентифікатор ролі (id), назва ролі (name).

Таблиця feedback призначена для зберігання інформації про відгуки інтерв'юерів на кандидатів, а саме: ідентифікатор відгуку (id), ідентифікатор інтерв'ю (interview_id), ідентифікатор користувача, який створив відгук (author_id), ідентифікатор оцінки професійного рівня кандидата (seniority_id), ідентифікатор статусу відгуку (status_id), текстова частина відгуку (text), оцінка вражень від кандидата (score), дати й часу створення (created_time), дати й часу оновлення (updated_time).

Таблиця feedback_status призначена для збереження даних про усі можливі статуси, в яких може перебувати відгук: ідентифікатор статусу (id), назва статусу (name). Наразі в системі присутні наступні статуси: «Не редаговано (Not edited)», «Чорновий варіант (Draft)», «Завершено (Completed)»

Таблиця feedback_seniority призначена для зберігання даних про усі можливі оцінки професійного рівня кандидата, а саме: ідентифікатор оцінки (id), назва оцінки (name). Наразі в системі присутні наступні оцінки професійного рівня кандидата: «L1 - Junior», «L2 - Middle», «L3 - Senior», «L4 - Lead».

Таблиця feedback_skill_score призначена для зберігання інформації про оцінки володіння кандидатом вимогами вакансії, а саме: ідентифікатор запису про оцінку (id), ідентифікатор відгуку (feedback_id), ідентифікатор вимоги вакансії (requirement_id), ідентифікатор оцінки вимоги (score_id).

Таблиця skill_score призначена для зберігання даних про усі можливі види оцінок володіння кандидатом вимог, а саме: ідентифікатор оцінки (id), назва

оцінки (name). Наразі в системі присутні наступні оцінки володіння вимогами вакансії: «Початківець (Novice)», «Середній (Intermediate)», «Досвідчений (Advanced)».

2.2.2 Архітектура підсистеми комунікації між користувачами під час співбесіди

Дана підсистема відповідальна за підтримки проведення співбесіди та комунікації користувачів під час неї. В рамках індивідуальної частини дипломного проєкту в цій підсистемі було реалізовано механізм для підтримки можливості спільного редагування програмного коду.

Таблиця 2.2 – специфікація методів класів підсистеми комунікації між користувачами під час співбесіди

Клас	**Метод**	**Призначення**	**Аргументи**
EditorService	updateCode	Зберігає в базі даних оновлену версію спільного тексту коду для заданого інтерв'ю.	interviewId, code
EditorService	updateLanguage	Зберігає в базі даних оновлену версію вибраної мови програмування для редактора коду для заданого інтерв'ю.	interviewId, language
LiveEditorGateway	handleCodeUpdate	Зберігає в базі даних оновлений текст програмного коду та надсилає активним учасникам співбесіди нову версію тексту програмного коду.	client, data

Продовження таблиці 2.2

LiveEditorGateway	handleLanguageUpdate	Зберігає в базі даних вибрану мову програмування та надсилає активним учасникам співбесіди вибрану мову.	client, data

2.2.3 Архітектура підсистеми користувацького інтерфейсу

Даний компонент системи представлений клієнтським веб-додатком та служить інтерфейсом для користувачів платформи.

В рамках даної частини дипломного проєкту було розроблено наступні сторінки клієнтського застосунку: сторінка створення вакансії, сторінка редагування вакансії, сторінка перегляду вакансії, сторінка створення інтерв'ю, сторінка перегляду інтерв'ю, сторінка редагування інтерв'ю, сторінка створення відгуку на кандидата, сторінка перегляду відгуку, сторінка редагування відгуку, сторінка, яка включає в себе спільний редактор програмного коду.

2.3 Аналіз безпеки даних

Безпека даних в даній частині дипломного проєкту контролюється в основному механізмами доступу до даних та доступу до сховища даних. Таким чином доступ до даних можуть отримати тільки ті користувачі, яким це передбачено в рамках використання системи.

2.4 Висновки по розділу

В цьому розділі було виконано моделювання складових індивідуальної частини дипломного проєкту на основі вимог описаних в першому розділі. Було розглянуто вибрану архітектуру підсистем та взаємодію між ними. Також було проаналізовано стан безпеки даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Розглянувши функціональні вимоги та варіанти використання системи, переходимо до визначення планів тестування. Детальний план тестування описаний в таблиці 3.1.

Таблиця 3.1 – План тестування

№	**Розділ**	**Елемент**
1	Вступ	Розглядається опис тестування системи для менеджменту інтерв'ю та їх проведення.
2	Функціонал, що підлягає тестуванню	– Створення вакансії; – Перегляд вакансії; – Редагування вакансії; – Створення інтерв'ю; – Перегляд інтерв'ю; – Запрошення учасників інтерв'ю; – Редагування інтерв'ю; – Створення відгуку на кандидата; – Редагування відгуку на кандидата; – Перегляд відгуку на кандидата; – Введення програмного коду в спільному редакторі; – Вибір мови програмування в спільному редакторі; – Обчислення середніх оцінок кандидата.

Продовження таблиці 3.1

3	Функціонал, що не підлягає тестуванню	
4	Тестові елементи	– Створення вакансії; – Перегляд вакансії; – Редагування вакансії; – Створення інтерв'ю; – Перегляд інтерв'ю; – Запрошення учасників інтерв'ю; – Редагування інтерв'ю; – Створення відгуку на кандидата; – Редагування відгуку на кандидата; – Перегляд відгуку на кандидата; – Введення програмного коду в спільному редакторі; – Вибір мови програмування в спільному редакторі; – Обчислення середніх оцінок кандидата.
5	Тестові підходи	Основний підхід для тестування – Black-box. Він передбачає тільки ввід даних і перевірку вихідних даних на коректність без перевірки внутрішньої роботи системи. Таким чином перевіряється факт досягнення бажаного результату, а не шлях його досягнення.

Продовження таблиці 3.1

6	Критерій успішно/не успішно	– Програма працює без критичних збоїв; – Програма виконує перевірку вхідних даних; – При коректних даних програма віддає очікуваний результат; – Недоліки, які класифікуються як критичні та впливають на роботу програми в цілому повинні бути усуненими; – Допускається 5 недоліків середнього рівня та 10 низького; – Усі вимоги повинні бути покриті тестовими сценаріями.
7	Критерій призупинення/відновлення	Відсутність доступу до програми, відмова сервера, проблеми з інтернет мережею.
8	Вимоги до середовища	Для тестування персонал повинен використати веб-переглядач Google Chrome версії не нижче 90.0.4430.93 та провести перевірку сценаріїв на операційній системі системі macOS версії 11.2.2 та Windows.
9	Вимоги до навичок персоналу	Володіння браузером Google Chrome на рівні вище середнього. Розуміння базових принципів тестування та роботи веб-застосунків.

Продовження таблиці 3.1

10	Задачі тестування	– виявлення некоректної роботи програми та аналіз причин, які до цього призводять; – повторна перевірка несправності після її усунення; – створення тестових сценаріїв;
11	Супровід тестування	– сценарії тестування; – тестовий план; – журнал звітності по кожному сценарію; – генератори даних.
12	Розклад тестування	На кожен тест виділяється 40хв робочого часу.
14	Відповідальні	Автор тестового плану є відповідальним за процес тестування в цілому.
15	Підтвердження	Перехід до наступних етапів тестування може відбутись за відповідним рішенням автора плану.
16	Ризики	Якість розробленого програмного забезпечення може знизитись у випадку зменшення часу та зусиль відведених на перевірку тестових сценаріїв.

3.2 Опис процесів тестування

Тестування відбулося згідно з створеним планом. Результати проведеного тестування відображені в таблиці 3.2.

Таблиця 3.2 – Тестування програмного забезпечення

Процес	**Назва**	**Етап**	**Очікуваний результат**	**Реальний результат**	**Висновок**
Відображення вакансій	Vacancies_list_positive	Зайти під користувачем з роллю рекрутер. Створити вакансію. Відкрити сторінку облікового запису. Перейти на вкладку з вакансіями.	Раніше створена вакансія відображається на сторінці з списком вакансій.	Раніше створена вакансія відображається на сторінці з списком вакансій.	Успіх
Відображення вакансій	Vacancies_list_no_vacancy	Створити нового користувача з роллю рекрутер. Відкрити сторінку з списком вакансій.	Повідомлення про те, що зараз немає жодної вакансії в даного користувача.	Відображається повідомлення про те, що зараз немає жодної вакансії в даного користувача.	Успіх

Продовження таблиці 3.2

Створення вакансії	Vacancy_creation_button	Зайти під користувачем з роллю рекрутер. Відкрити сторінку аккаунту, а далі вкладку зі списком вакансій.	В верхньому правому куті відображається кнопка «Нова вакансія»	В верхньому правому куті відображається кнопка «Нова вакансія»	Успіх
Створення вакансії	Vacancy_creation_opening	Зайти під користувачем з роллю рекрутер. Відкрити сторінку аккаунту, а далі вкладку зі списком вакансій. Натиснути кнопку «Нова вакансія».	Перехід на сторінку створення вакансії	Користувачу відкривається сторінка створення вакансії	Успіх

Продовження таблиці 3.2

Створення вакансії	Vacancy_creation_positive	Відкрити сторінку створення вакансії. Ввести поле назви вакансії. Ввести поле «Посилання на вакансію». Ввести опис вакансії, а також обов’язкові навички для кандидата. Натиснути кнопку «Зберегти».	Нова вакансія створена. Перехід на сторінку з описом щойно створеною вакансією.	Нова вакансія створена. Користувач у відкривається сторінка з описом щойно створеною вакансією.	Успіх
Створення вакансії	Vacancy_creation_negative_absent_title	Відкрити сторінку створення вакансії. Залишити поле назви вакансії пустим. Натиснути кнопку «Зберегти».	Червона рамка навколо поля назви вакансії	З’являється червона рамка навколо поля назви вакансії. Користувач залишається на тій ж сторінці.	Успіх

Продовження таблиці 3.2

Створення вакансії	Vacancy_creation_positive_absent_external_link	Відкрити сторінку створення вакансії. Ввести поле назви вакансії. Залишити поле «Посилання на вакансію» пустим. Ввести опис вакансії, а також обов’язкові навички для кандидата. Натиснути кнопку «Зберегти».	Нова вакансія створена. Перехід на сторінку з описом щойно створеною вакансією.	Нова вакансія створена. Користувач у відкривається сторінка з описом щойно створеною вакансією.	Успіх
Створення вакансії	Vacancy_creation_negative_absent_description	Відкрити сторінку створення вакансії. Залишити поле опису вакансії пустим. Натиснути кнопку «Зберегти».	Червона рамка навколо поля опису вакансії	З’являється червона рамка навколо поля опису вакансії. Користувач залишається на тій ж сторінці.	Успіх

Продовження таблиці 3.2

Створення вакансії	Vacancy_creation_negative_absent_requirements	Відкрити сторінку створення вакансії. Залишити поле обов'язкових навичок кандидата пустим. Натиснути кнопку «Зберегти».	Червона рамка навколо поля обов'язкових навичок кандидата	З'являється червона рамка навколо поля обов'язкових навичок кандидата. Користувач залишається на тій ж сторінці.	Успіх
Відображення списку інтерв'ю	Interview-list_no_interview_positive	Зайти під користувачем з роллю рекрутер. Створити нову вакансію. Перейти на сторінку перегляду вакансії. Відкрити вкладку інтерв'ю.	Повідомлення про те, що для даної вакансії ще немає жодного інтерв'ю.	Відображається повідомлення про те, що для даної вакансії поки що немає жодного інтерв'ю.	Успіх

Продовження таблиці 3.2

Відображення списку інтерв'ю	Interview_list_positive	Зайти під користувачем з роллю рекрутер. Перейти на сторінку перегляду вакансії. Створити інтерв'ю. Перейти на сторінку списку інтерв'ю.	Раніше створене інтерв'ю відображається на сторінці з списком інтерв'ю.	Раніше створене інтерв'ю відображається на сторінці з списком інтерв'ю.	Успіх
Створення інтерв'ю	Interview_creation_button	Зайти під користувачем з роллю рекрутер. Відкрити сторінку перегляду вакансії.	В верхньому правому куті відображається кнопка «Нове інтерв'ю»	В верхньому правому куті відображається кнопка «Нове інтерв'ю»	Успіх
Створення інтерв'ю	Interview_creation_opening	Зайти під користувачем з роллю рекрутер. Відкрити сторінку	Перехід на сторінку створення інтерв'ю	Користувачу відкривається сторінка створення інтерв'ю	Успіх

Продовження таблиці 3.2

		перегляду вакансії. Натиснути кнопку «Нове інтерв’ю»			
Створення інтерв’ю	Interview_creation_positive_candidate_info	Відкрити сторінку створення інтерв’ю. Ввести поле імені кандидата Ввести поле Прізвище кандидата. Ввести пошту кандидата. Натиснути кнопку «Далі»	Відкрита нова вкладка створення інтерв’ю.	Користувачу відкривається нова вкладка для створення інтерв’ю, з полями для деталей інтерв’ю.	Успіх
Створення інтерв’ю	Interview_creation_positive_interview_details	Відкрити другий крок для створення інтерв’ю. Ввести поле інтерв’юери, вказавши імена або пошти потрібних інтерв’юерів.	Відкрита нова вкладка створення інтерв’ю.	Користувачу відкривається нова вкладка для створення інтерв’ю, з узагальненням інформації	Успіх

Продовження таблиці 3.2

		Ввести час та дату початку та кінця інтерв'ю. Натиснути кнопку «Далі»		введеної раніше.	
Створення інтерв'ю	Interview_creation_positive_final_step	Відкрити третій крок для створення інтерв'ю. Натиснути кнопку «Зберегти»	Нове інтерв'ю створене. Перехід на сторінку перегляду інтерв'ю.	Нове інтерв'ю створене. Користувач у відкривається сторінка з переглядом щойно створеного інтерв'ю.	Успіх
Створення інтерв'ю	Interview_creation_negative_absent_first_name	Відкрити перший крок для створення інтерв'ю. Залишити поле імені кандидата пустим. Натиснути кнопку «Далі».	Червона рамка навколо поля імені кандидата.	З'являється червона рамка навколо поля імені кандидата. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Створення інтерв'ю	Interview_creation_negative_absent_last_name	Відкрити перший крок для створення інтерв'ю. Залишити поле прізвища кандидата пустим. Натиснути кнопку «Далі».	Червона рамка навколо поля прізвища кандидата.	З'являється червона рамка навколо поля прізвища кандидата. Користувач залишається на тій ж вкладці.	Успіх
Створення інтерв'ю	Interview_creation_negative_absent_email	Відкрити перший крок для створення інтерв'ю. Залишити поле пошти кандидата пустим. Натиснути кнопку «Далі».	Червона рамка навколо поля пошти кандидата.	З'являється червона рамка навколо поля пошти кандидата. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Створення інтерв'ю	Interview_creation_negative_absent_interviewers	Відкрити другий крок для створення інтерв'ю. Залишити поле інтерв'юерів пустим. Натиснути кнопку «Далі».	Червона рамка навколо поля інтерв'юерів.	З'являється червона рамка навколо поля інтерв'юерів. Користувач залишається на тій ж вкладці.	Успіх
Редагування вакансії	Vacancy_editing_positive_opening	Відкрити сторінку перегляду вакансії. Натиснути кнопку «Редагувати»	Перехід на сторінку редагування вакансії	Користувачу відкривається сторінка редагування вакансії	Успіх
Редагування вакансії	Vacancy_editing_positive	Відкрити сторінку редагування вакансії. Редагувати поле назви, залишивши там коректні дані. Натиснути	Вакансія редагована. Перехід на сторінку з переглядом щойно редагованої вакансією.	Вакансія редагована. Користувачу відкривається сторінка з переглядом щойно	Успіх

Продовження таблиці 3.2

		кнопку «Зберегти».		редагованої вакансією.	
Редагування вакансії	Vacancy_editing_negative_absent_title	Відкрити сторінку редагування вакансії. Витерти поле назви. Натиснути кнопку «Зберегти»	Червона рамка навколо поля назви вакансії	З’являється червона рамка навколо поля назви вакансії. Користувач залишається на тій ж сторінці.	Успіх
Редагування вакансії	Vacancy_editing_negative_absent_description	Відкрити сторінку редагування вакансії. Витерти поле опису вакансії. Натиснути кнопку «Зберегти».	Червона рамка навколо поля опису вакансії	З’являється червона рамка навколо поля опису вакансії. Користувач залишається на тій ж сторінці.	Успіх

Продовження таблиці 3.2

Редагування вакансії	Vacancy_editing_negative_absent_requirements	Відкрити сторінку редагування вакансії. Витерти поле обов'язкових навичок кандидат. Натиснути кнопку «Зберегти».	Червона рамка навколо поля обов'язкових навичок кандидата	З'являється червона рамка навколо поля обов'язкових навичок кандидата. Користувач залишається на тій ж сторінці.	Успіх
Редагування інтерв'ю	Interview_ editing _opening	Відкрити сторінку перегляду інтерв'ю. Натиснути кнопку «Редагувати».	Перехід на сторінку редагування інтерв'ю	Користувач у відкривається сторінка редагування інтерв'ю	Успіх
Редагування інтерв'ю	Interview_editing _positive_candidate _iinfo	Відкрити сторінку редагування інтерв'ю. Редагувати поле імені користувача, залишивши там коректні дані.	Інтерв'ю редаговане. Перехід на сторінку з переглядом щойно редагованого інтерв'ю.	Інтерв'ю редаговане. Користувач у відкривається сторінка з переглядом щойно	Успіх

Продовження таблиці 3.2

		Натиснути двічі кнопку «Далі», а потім «Зберегти».		редагованого інтерв'ю зі зміненим іменем.	
Редагування інтерв'ю	Interview_ editing _negative _absent_first_name	Відкрити перший крок для створення інтерв'ю. Витерти поле імені кандидата. Натиснути кнопку «Далі».	Червона рамка навколо поля імені кандидата.	З'являється червона рамка навколо поля імені кандидата. Користувач залишається на тій ж вкладці.	Успіх
Редагування інтерв'ю	Interview_ editing _negative _absent_last_name	Відкрити перший крок для створення інтерв'ю. Витерти поле прізвища кандидата. Натиснути кнопку «Далі».	Червона рамка навколо поля прізвища кандидата.	З'являється червона рамка навколо поля прізвища кандидата. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Редагування інтерв'ю	Interview_editing_negative_absent_email	Відкрити перший крок для створення інтерв'ю. Витерти поле пошти кандидата. Натиснути кнопку «Далі».	Червона рамка навколо поля пошти кандидата.	З'являється червона рамка навколо поля пошти кандидата. Користувач залишається на тій ж вкладці.	Успіх
Редагування інтерв'ю	Interview_editing_negative_absent_interviewers	Відкрити другий крок для створення інтерв'ю. Витерти поле інтерв'юерів. Натиснути кнопку «Далі».	Червона рамка навколо поля інтерв'юерів.	З'являється червона рамка навколо поля інтерв'юерів. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Редагування інтерв'ю	Interview_editing_negative_absent_interviewers	Відкрити другий крок для створення інтерв'ю. Витерти поле часу початку інтерв'ю. Натиснути кнопку «Далі».	Червона рамка навколо поля часу початку інтерв'ю.	З'являється червона рамка навколо поля часу початку інтерв'ю. Користувач залишається на тій ж вкладці.	Успіх
Редагування інтерв'ю	Interview_editing_negative_absent_interviewers	Відкрити другий крок для створення інтерв'ю. Витерти поле часу кінця інтерв'ю. Натиснути кнопку «Далі».	Червона рамка навколо поля часу кінця інтерв'ю.	З'являється червона рамка навколо поля часу кінця інтерв'ю. Користувач залишається на тій ж вкладці.	Успіх
Проведення інтерв'ю					

Продовження таблиці 3.2

Проведення інтерв'ю	Interview_code_editor_positive	Приєднатися до інтерв'ю. Натиснути на кнопку відкриття редактору коду. Написати певний текст	Текст доступний для перегляду і редагування всім користувачам	Всі користувачі мають доступ до тексту, написаного іншим учасником.	Успіх
Проведення інтерв'ю	Interview_code_editor_positive_highlight	Приєднатися до інтерв'ю. Натиснути на кнопку відкриття редактору коду. Написати певний текст. Вибрати мову програмування	Мова програмування змінюється в всіх користувачів. Текст підсвічений відповідно до синтаксису вибраної мови програмування.	В всіх користувачі в проредагується текст, а також підсвітиться відповідно до обраної мови програмування. Всім користувачам доступна обрана мова програмування.	Успіх

Продовження таблиці 3.2

Створення відгуку	Feedback_creation_positive	Відкрити сторінку створення відгуку. Ввести поле оцінки успішності кандидата. Обрати професійний рівень. Написати текстовий відгук на кандидата. Обрати рівень знання необхідних навичок. Натиснути кнопку «Зберегти».	Відгук збережений. Відкрита сторінка перегляду відгуку.	Відгук збережений. Користувач у відкривається нова сторінка з переглядом відгуку.	Успіх

Продовження таблиці 3.2

Створення відгуку	Feedback_creation_negative_absent_scor	Відкрити сторінку створення відгуку. Залишити пустим поле оцінки успішності кандидата. Натиснути кнопку «Зберегти».	Червона рамка навколо поля оцінки успішності кандидата.	З’являється червона рамка навколо поля оцінки успішності кандидата. Користувач залишається на тій ж вкладці.	Успіх
Створення відгуку	Feedback_creation_negative_absent_seniority	Відкрити сторінку створення відгуку. Залишити пустим поле професійного рівня. Натиснути кнопку «Зберегти».	Червона рамка навколо поля професійного рівня.	З’являється червона рамка навколо поля професійного рівня. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Створення відгуку	Feedback_creation_negative_absent_feedback_text	Відкрити сторінку створення відгуку. Залишити поле текстового відгуку пустим. Натиснути кнопку «Зберегти».	Червона рамка навколо поля текстового відгуку для кандидата.	З’являється червона рамка навколо поля текстового відгуку. Користувач залишається на тій ж вкладці.	Успіх
Створення відгуку	Feedback_creation_negative_absent_required_skills	Відкрити сторінку створення відгуку. Залишити поле оцінки обов’язкової навички пустим. Натиснути кнопку «Зберегти».	Червона рамка навколо поля оцінки обов’язкової навички.	З’являється червона рамка навколо поля оцінки обов’язкової навички. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Створення відгуку	Feedback_creation_positive	Відкрити сторінку створення відгуку. Залишити деякі поля пустими. Натиснути кнопку «Зберегти як чернетка».	Відкрита сторінка перегляду інтерв'ю.	Відгук збережений. Користувачу відкривається нова сторінка з переглядом відгуку. Всі валідації для чернетки відсутні.	Успіх
Редагування відгуку	Feedback_editing_positive	Відкрити сторінку редагування відгуку. Редагувати поле оцінки успішності кандидата, залишивши коректні дані. Натиснути кнопку «Зберегти».	Відгук відредагований. Відкрита сторінка перегляду відгуку.	Відгук відредагований. Користувачу відкривається нова сторінка з переглядом відгуку, з уже зміненим полем оцінки	Успіх

Продовження таблиці 3.2

				успішності кандидата.	
Редагування відгуку	Feedback_editing_negative_absent_scor	Відкрити сторінку редагування відгуку. Витерти поле оцінки успішності кандидата. Натиснути кнопку «Зберегти».	Червона рамка навколо поля оцінки успішності кандидата.	З'являється червона рамка навколо поля оцінки успішності кандидата. Користувач залишається на тій ж вкладці.	Успіх
Редагування відгуку	Feedback_editing_negative_absent_seniority	Відкрити сторінку редагування відгуку. Витерти поле професійного рівня. Натиснути кнопку «Зберегти».	Червона рамка навколо поля професійного рівня.	З'являється червона рамка навколо поля професійного рівня. Користувач залишається на тій ж вкладці.	Успіх

Продовження таблиці 3.2

Редагування відгуку	Feedback_editing_negative_absent_feedback_text	Відкрити сторінку редагування відгуку. Витерти поле текстового відгуку. Натиснути кнопку «Зберегти».	Червона рамка навколо поля текстового відгуку для кандидата.	З'являється червона рамка навколо поля текстового відгуку. Користувач залишається на тій ж вкладці.	Успіх
Редагування відгуку	Feedback_editing_negative_absent_required_skills	Відкрити сторінку редагування відгуку. Витерти поле оцінки обов'язкової навички. Натиснути кнопку «Зберегти».	Червона рамка навколо поля оцінки обов'язкової навички.	З'являється червона рамка навколо поля оцінки обов'язкової навички. Користувач залишається на тій ж вкладці.	Успіх

3.3 Висновки по розділу

В даному розділі було сформульовано план тестування а також описано сценарії тестування за якими відбувалось тестування розробленого програмного забезпечення. В результаті тестування всі сценарії були виконані успішно.

ВИСНОВКИ

В даній індивідуальній частині дипломного проєкту було розроблено систему організації та проведення інтерв'ю в рамках платформи для проведення технічних інтерв'ю. Основною задачею якої є спрощення та автоматизація процесів роботи з кандидатами, співбесідами та відгуками на кандидатів.

Після виконання аналізу предметної області в рамках загальної частини було виконано формулювання варіантів використання системи організації та проведення інтерв'ю, визначено основних акторів, які взаємодіють з системою. На основі варіантів використання розроблено перелік функціональних вимог, а також сформульовано основні задачі індивідуального дипломного проєкту.

Для виконання поставлених задач описано основні бізнес-процеси, які вони передбачають. Для реалізації бізнес-процесів в програмі було розроблено деталізовану архітектуру компонентів програмного забезпечення, визначених в загальній частині дипломного проєкту, а також визначено типи та схеми баз даних для цих компонентів.

Після розроблення програмного забезпечення сформульовано план тестування та сценарії тестування, які покликані охопити всі функціональні вимоги. Було проведено тестування програмного забезпечення згідно з сценаріями тестування, в результаті чого всі тести були завершені успішно.

ПЕРЕЛІК ПОСИЛАНЬ

1) What is a REST API? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.redhat.com/en/topics/api/what-is-a-rest-api
2) PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. – Режим доступу до ресурсу: https://www.postgresql.org/

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008230369

Дата перевірки:
08.06.2021 15:15:16 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2021 15:47:06 EEST

ID користувача:
77149

Назва документа: **Hnatyshyn_bachelor_ip72**

Кількість сторінок: **130** Кількість слів: **13883** Кількість символів: **113859** Розмір файлу: **1.99 MB** ID файлу: **1008303705**

4.64%
Схожість

Найбільша схожість: **4.13% з джерелом з Бібліотеки** (ID файлу: **1008303694**)

2.67% Джерела з Інтернету	85	Сторінка 132
4.52% Джерела з Бібліотеки	178	Сторінка 132

0.03% Цитат

Цитати	1	Сторінка 133

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПЛАТФОРМА ДЛЯ ПРОВЕДЕННЯ ТЕХНІЧНИХ ІНТЕРВ’Ю (КОМПЛЕКСНА ТЕМА). СИСТЕМА ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ІНТЕРВ’Ю.

Опис програми

КПІ.ІП-7207.045440.09.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.В. Ковтунець

Нормоконтроль:

_____ І.І. Вітковська

Виконавець:

_____М.С. Гнатишин

Київ – 2021 року

Тексти програмного коду

Система організації та проведення інтерв'ю

(Найменування програми (документа))

DVD-R

(Вид носія даних)

46 арк, 321 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2021

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { Interview, InterviewDocument } from '../database/schemas/interview.schema';

@Injectable()
export class EditorService {
  constructor(
    @InjectModel(Interview.name)
    private interviewModel: Model<InterviewDocument>,
  ) {}

  async updateCode(interviewId: string, code: string): Promise<void> {
    await this.interviewModel.updateOne({ interviewId }, { $set: { 'editor.code': code } }).exec();
  }

  async updateLanguage(interviewId: string, language: string): Promise<void> {
    await this.interviewModel
      .updateOne({ interviewId }, { $set: { 'editor.language': language } })
      .exec();
  }
}

export enum MESSAGES {
  CODE_UPDATE = 'editor/code-update',
  LANGUAGE_UPDATE = 'editor/language-update',
}
```

```
import { SubscribeMessage, WebSocketGateway } from '@nestjs/websockets';
import { Socket } from 'socket.io';
import {
  CodeUpdateMessagePayload,
  LanguageUpdateMessagePayload,
} from './messages-payload.interface';
import { MESSAGES } from './messages.enum';
import { EditorService } from './editor.service';

@WebSocketGateway()
export class LiveEditorGateway {
  constructor(private editorService: EditorService) {}

  @SubscribeMessage(MESSAGES.CODE_UPDATE)
  async handleCodeUpdate(client: Socket, data: CodeUpdateMessagePayload) {
    client.to(data.interviewId).emit(MESSAGES.CODE_UPDATE, { code: data.code });
    await this.editorService.updateCode(data.interviewId, data.code);
  }

  @SubscribeMessage(MESSAGES.LANGUAGE_UPDATE)
  async handleLanguageUpdate(client: Socket, data: LanguageUpdateMessagePayload) {
    client.to(data.interviewId).emit(MESSAGES.LANGUAGE_UPDATE, { language: data.language });
    await this.editorService.updateLanguage(data.interviewId, data.language);
  }
}
```

```
import { Injectable } from '@nestjs/common';
import { DataMapper } from '../../shared/data-mapper';
import { FeedbackSkillScore } from '../../database/entities/feedback-skill-score.entity';
import { NewFeedbackSkillsDto } from '../request-models/new-feedback-skills.dto';
import { SkillsScoresDto } from '../response-models/skills-scores.dto';

@Injectable()
export class FeedbackSkillsMapper extends DataMapper<
  FeedbackSkillScore,
  NewFeedbackSkillsDto,
  SkillsScoresDto
> {
  public mapDtoToEntity(dto: NewFeedbackSkillsDto & { feedbackId: string }): FeedbackSkillScore {
    const item = new FeedbackSkillScore();
    item.feedback_id = dto.feedbackId;
    item.requirement_id = dto.requirementId;
    item.score_id = dto.score;
    return item;
  }

  public mapEntityToDto(entity: FeedbackSkillScore): SkillsScoresDto {
    return {
      id: entity.id,
      requirementId: entity.requirement_id,
      score: entity.score.name,
      scoreId: entity.score.id,
```

```
      requirementName: entity.requirement.name,
    };
  }
}

import { Injectable } from '@nestjs/common';
import { DataMapper } from '../../shared/data-mapper';
import { Feedback } from '../../database/entities/feedback.entity';
import { CreateFeedbackDto } from '../request-models/create-feedback.dto';
import { FeedbackSkillsMapper } from './feedback-skills.mapper';
import { FeedbackDto } from '../response-models/feedback.dto';

@Injectable()
export class FeedbackMapper extends DataMapper<Feedback, CreateFeedbackDto, FeedbackDto> {
  constructor(private skillsMapper: FeedbackSkillsMapper) {
    super();
  }

  public mapDtoToEntity(dto: CreateFeedbackDto): Feedback {
    const feedback = new Feedback();
    feedback.score = dto.score;
    feedback.author_id = dto.authorId;
    feedback.text = dto.text;
    feedback.interview_id = dto.interviewId;
    feedback.seniority_id = dto.seniority;
    return feedback;
  }
```

```
  public mapEntityToDto(entity: Feedback): FeedbackDto {
    return {
      id: entity.id,
      interviewId: entity.interview_id,
      title: entity.interview?.vacancy?.title || '',
      text: entity.text,
      authorId: entity.author_id,
      createdTime: entity.created_time,
      updatedTime: entity.updated_time,
      seniority: entity.seniority.name,
      seniorityId: entity.seniority.id,
      score: entity.score,
      status: entity.status.name,
      statusId: entity.status.id,
      candidateName: entity.interview?.candidate_name || '',
      skillScores: entity.skill_scores
        ? entity.skill_scores.map(this.skillsMapper.mapEntityToDto)
        : undefined,
    };
  }
}

import { Injectable } from '@nestjs/common';
import { CreateFeedbackDto } from '../request-models/create-feedback.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Feedback } from '../../database/entities/feedback.entity';
import { Repository, Connection } from 'typeorm';
import { FeedbackMapper } from './feedback.mapper';
import { UpdateFeedbackDto } from '../request-models/update-feedback.dto';
```

```
import { FeedbackSkillScore } from '../../database/entities/feedback-skill-score.entity';
import { NotFoundException } from '../../shared/exceptions';
import { FeedbackWithScoresDto } from '../response-models/feedback-with-scores.dto';
import { FeedbackDto } from '../response-models/feedback.dto';
import { FeedbackSkillsMapper } from './feedback-skills.mapper';
import { FindConditions } from 'typeorm/find-options/FindConditions';
import { NewFeedbackSkillsDto } from '../request-models/new-feedback-skills.dto';
import { FeedbackSeniority } from '../../database/entities/feedback-seniority.entity';
import { VacancyRequirement } from '../../database/entities/vacancy-requirement.entity';
import { FeedbackStatus } from '../../database/entities/feedback-status.entity';
import { InterviewExternalService } from '../../interview-external/services/interview-external.service';

export enum FEEDBACK_STATUS {
  NOT_EDITED = 1,
  DRAFT,
  FINISHED,
}

@Injectable()
export class FeedbackService {
  constructor(
    @InjectRepository(Feedback)
    private feedbackRepository: Repository<Feedback>,
```

```
  @InjectRepository(FeedbackSkillScore)
  private feedbackSkillsRepository: Repository<FeedbackSkillScore>,
  @InjectRepository(FeedbackSeniority)
  private seniorityRepository: Repository<FeedbackSeniority>,
  @InjectRepository(FeedbackStatus)
  private feedbackStatusRepository: Repository<FeedbackStatus>,
  private feedbackMapper: FeedbackMapper,
  private feedbackSkillsMapper: FeedbackSkillsMapper,
  private connection: Connection,
  private interviewService: InterviewExternalService,
) {}

createEmptyFeedbacksForInterview(
  interviewId: string,
  interviewersIds: string[],
  requirements: VacancyRequirement[],
): Feedback[] {
  return interviewersIds.map((userId) => {
    const feedback = new Feedback();

    feedback.skill_scores = requirements.map((requirement) => {
      const skillScore = new FeedbackSkillScore();
      skillScore.requirement_id = requirement.id;
      return skillScore;
    });

    feedback.author_id = userId;
    feedback.interview_id = interviewId;
    feedback.status_id = 1;
```

```
    return feedback;
  });
 }

 async createNew(data: CreateFeedbackDto): Promise<FeedbackWithScoresDto> {
  const queryRunner = this.connection.createQueryRunner();
  await queryRunner.connect();
  const feedback = this.feedbackMapper.mapDtoToEntity(data);

  try {
   await queryRunner.startTransaction();
   await this.feedbackRepository.save(feedback);

   const feedbackSkills = data.skills.map((score) => {
    return this.feedbackSkillsMapper.mapDtoToEntity({ ...score, feedbackId: feedback.id });
   });
   await this.feedbackSkillsRepository.save(feedbackSkills);
   await queryRunner.commitTransaction();
  } catch (e) {
   await queryRunner.rollbackTransaction();
   throw e;
  }

  return this.getOne(feedback.id);
 }
```

```
    async updateOne(data: UpdateFeedbackDto, id: string): Promise<FeedbackWithScoresDto> {
      const feedback = await this.getBy({ id });

      feedback.text = data.text || feedback.text;
      feedback.score = data.score || feedback.score;

      if (data.seniority) {
        feedback.seniority = await this.seniorityRepository.findOneOrFail(data.seniority);
        feedback.seniority_id = data.seniority;
      }

      if (feedback.status_id !== FEEDBACK_STATUS.FINISHED) {
        const statuses = await this.feedbackStatusRepository.find();
        const finished = statuses.find((status) => status.id === FEEDBACK_STATUS.FINISHED);
        const draft = statuses.find((status) => status.id === FEEDBACK_STATUS.DRAFT);

        feedback.status = data.isCompleted ? finished : draft;

        if (feedback.status.id === FEEDBACK_STATUS.FINISHED) {
          await this.interviewService.completeInterview(feedback.interview_id, feedback.id);
        }
      }
```

```
        const [skillsScoresToSave, skillsScoresToCreate] =
  this.processSkillsScoresUpdate(
          data.skills || [],
          feedback,
        );

        const queryRunner = this.connection.createQueryRunner();
        await queryRunner.connect();

        try {
          await queryRunner.startTransaction();

          if (skillsScoresToSave.length > 0) {
            await this.feedbackSkillsRepository.save(skillsScoresToSave);
          }

          if (skillsScoresToCreate.length > 0) {
            await this.feedbackSkillsRepository.create(skillsScoresToCreate);
          }

          console.log(feedback.status_id);

          await this.feedbackRepository.save(feedback);

          await queryRunner.commitTransaction();
        } catch (e) {
          await queryRunner.rollbackTransaction();
          throw e;
        }
```

```
    return this.feedbackMapper.mapEntityToDto(feedback) as
FeedbackWithScoresDto;
  }

  async getOne(id: string): Promise<FeedbackWithScoresDto> {
    const feedback = await this.getBy({ id });
    return this.feedbackMapper.mapEntityToDto(feedback) as
FeedbackWithScoresDto;
  }

  async getOneByInterviewAndAuthor(
    authorId: string,
    interviewId: string,
  ): Promise<FeedbackWithScoresDto> {
    const feedback = await this.getBy({ author_id: authorId, interview_id:
interviewId });
    return this.feedbackMapper.mapEntityToDto(feedback) as
FeedbackWithScoresDto;
  }

  private async getBy(options: FindConditions<Feedback>):
Promise<Feedback> {
    const feedback = await this.feedbackRepository.findOne(options, {
      relations: [
        'skill_scores',
        'skill_scores.requirement',
        'skill_scores.score',
        'seniority',
```

```
      'status',
      'interview',
      'interview.vacancy',
    ],
  });
  if (!feedback) {
    throw new NotFoundException('Feedback', JSON.stringify(options));
  }
  return feedback;
}

async deleteOne(id: string): Promise<void> {
  const feedback = await this.getBy({ id });
  await this.feedbackRepository.remove(feedback);
}

async search(
  authorId: string,
  interviewId: string,
  page = 1,
  pageSize = 10,
): Promise<FeedbackDto[]> {
  const queryOption: FindConditions<Feedback> = {};
  if (authorId) {
    queryOption.author_id = authorId;
  }
  if (interviewId) {
    queryOption.interview_id = interviewId;
  }
```

```
    const feedbacks = await this.feedbackRepository.find({
      where: queryOption,
      relations: ['seniority', 'status'],
      take: pageSize,
      skip: (page - 1) * pageSize,
      order: { created_time: 1 },
    });
    return feedbacks.map(this.feedbackMapper.mapEntityToDto);
  }

  private processSkillsScoresUpdate(
    skills: NewFeedbackSkillsDto[],
    feedback: Feedback,
  ): [FeedbackSkillScore[], FeedbackSkillScore[]] {
    const skillsScoresToSave = [];
    const skillsScoresToCreate = [];
    skills.forEach(({ score, requirementId }) => {
      const item = feedback.skill_scores.find((_) => _.requirement_id ===
requirementId);
      if (item) {
        item.score_id = score;
        skillsScoresToSave.push(item);
        return;
      }
      const newRecord = this.feedbackSkillsMapper.mapDtoToEntity({
        score,
        requirementId,
        feedbackId: feedback.id,
      });
```

```
            skillsScoresToCreate.push(newRecord);
          });
          return [skillsScoresToSave, skillsScoresToCreate];
        }
      }

      import { Injectable } from '@nestjs/common';
      import { InjectRepository } from '@nestjs/typeorm';
      import { Interview } from '../../database/entities/interview.entity';
      import { Repository } from 'typeorm';
      import { FEEDBACK_STATUS } from '../../feedback/services/feedback.service';
      import { INTERVIEW_STATUS } from '../../interview/services/interview.models';

      @Injectable()
      export class InterviewExternalService {
        constructor(
          @InjectRepository(Interview)
          private interviewRepository: Repository<Interview>,
        ) {}

        async completeInterview(id: string, feedbackId: string): Promise<void> {
          const interview = await this.interviewRepository.findOne(id, { relations: ['feedbacks'] });
          const areAllFeedbacksCompleted =
            interview.feedbacks.filter(
              (feedback) => feedback.status_id === FEEDBACK_STATUS.FINISHED || feedback.id === feedbackId,
```

```
      ).length === interview.feedbacks.length;
     if (areAllFeedbacksCompleted) {
      interview.status_id = INTERVIEW_STATUS.COMPLETED;
      await this.interviewRepository.save(interview);
     }
    }
   }

   import { Injectable } from '@nestjs/common';
   import * as mailjet from 'node-mailjet';
   import { ConfigService } from '@nestjs/config';
   import { EmailConfig } from '../../core/config';

   @Injectable()
   export class EmailService {
    private client;

    constructor(private configService: ConfigService) {
     const emailConfig = configService.get<EmailConfig>('email');
     this.client = mailjet.connect(emailConfig.key, emailConfig.secret);
    }

    async sendMessage(target: string, subject: string, html: string): Promise<void>
{
     await this.client.post('send', { version: 'v3.1' }).request({
      Messages: [
       {
        From: {
         Email: '7585955@ukr.net',
```

```
            Name: 'Interview platform',
          },
          To: [
            {
              Email: target,
            },
          ],
          Subject: subject,
          HTMLPart: html,
        },
      ],
    });
  }
}

import { Injectable } from '@nestjs/common';
import { EmailService } from './email.service';
import { Interview } from '../../database/entities/interview.entity';
import { TokenService } from './token.service';
import { GuestTokenDto } from '../request-models/guest-token.dto';
import { getFormattedDuration } from './helpers';
import { ConfigService } from '@nestjs/config';
import { getCandidateEmailTemplate, getInterviewerEmailTemplate } from
'./email.templates';
import { UserDetailsDto } from '../response-models/user-details.dto';
import { INTERVIEW_USER_ROLES } from './interview.models';

@Injectable()
export class ExternalNotifierService {
```

```
  private readonly WEB_APP_URL: string;

  constructor(
    private emailService: EmailService,
    private tokenService: TokenService,
    private configService: ConfigService,
  ) {
    this.WEB_APP_URL = configService.get('services.webApp');
  }

  async notifyAttendees(
    candidate: UserDetailsDto,
    interviewers: UserDetailsDto[],
    interview: Interview,
  ): Promise<void> {
    const startTime = interview.start_time;
    const durationInMilliseconds =
      new Date(interview.end_time).valueOf() - new
Date(interview.start_time).valueOf();
    const formattedDuration = getFormattedDuration(durationInMilliseconds);
    const formattedStartTime = new Date(startTime).toLocaleString();
    await this.notifyCandidate(
      candidate.email,
      `${candidate.firstName} ${candidate.lastName}`,
      interview,
      candidate.id,
      formattedStartTime,
      formattedDuration,
    );
```

```
  const internalLink = this.getInternalJoiningLink(interview.id);

  await Promise.all(
    interviewers.map((interviewer) =>
      this.notifyInterviewer(
        interviewer.email,
        `${interviewer.firstName} ${interviewer.lastName}`,
        formattedStartTime,
        formattedDuration,
        internalLink,
      ),
    ),
  );
}

async notifyInterviewer(
  email: string,
  name: string,
  formattedStartTime: string,
  formattedDuration: string,
  joiningLink: string,
) {
  const emailTemplate = getInterviewerEmailTemplate(
    name,
    formattedStartTime,
    formattedDuration,
    joiningLink,
  );
```

```
        await this.emailService.sendMessage(email, 'Interview Invite',
emailTemplate);
      }

      async notifyCandidate(
        email: string,
        name: string,
        interview: Interview,
        userId: string,
        formattedStartTime: string,
        formattedDuration: string,
      ): Promise<void> {
        const joiningToken = await this.getExternalJoiningToken(
          userId,
          INTERVIEW_USER_ROLES.CANDIDATE,
          interview,
        );
        const joiningLink = this.getExternalJoiningLink(joiningToken);
        const emailTemplate = getCandidateEmailTemplate(
          name,
          formattedStartTime,
          formattedDuration,
          joiningLink,
        );
        await this.emailService.sendMessage(email, 'Interview Invite',
emailTemplate);
      }

      private getInternalJoiningLink(id: string): string {
```

```
    return `${this.WEB_APP_URL}interview/${id}`;
  }

  private getExternalJoiningToken(
    userId: string,
    role: INTERVIEW_USER_ROLES = INTERVIEW_USER_ROLES.CANDIDATE,
    interview: Interview,
  ): Promise<string> {
    const payload: GuestTokenDto = {
      userId,
      interviewId: interview.id,
      interviewEnd: interview.end_time,
      role,
    };
    return this.tokenService.getGuestInterviewToken(payload);
  }

  private getExternalJoiningLink(token: string): string {
    return `${this.WEB_APP_URL}external/join-interview?token=${token}`;
  }
}

import { Score } from '../response-models/interview-scoring.dto';
import { Feedback } from '../../database/entities/feedback.entity';
import { FeedbackSeniority } from '../../database/entities/feedback-seniority.entity';
import { FeedbackSkillScore } from '../../database/entities/feedback-skill-score.entity';
```

```
export const getFormattedDuration = (duration: number): string => {
  const hours = Math.floor(duration / (60 * 60 * 1000));
  const minutes = duration / (60 * 1000) - hours * 60;
  let result = '';
  if (hours > 0) {
    result += `${hours} hour${hours > 1 ? 's' : ''} `;
  }
  if (minutes > 0) {
    result += `${minutes} minute${minutes > 1 ? 's' : ''}`;
  }
  return result;
};

export const getMinAndMaxAmongScores = (scores: Score[]): { min: Score; max: Score } => {
  return scores.reduce(
    (result, curr: Score) => {
      if (!result.min || curr.id < result.min.id) {
        result.min = curr;
      }
      if (!result.max || curr.id > result.max.id) {
        result.max = curr;
      }
      return result;
    },
    { min: null, max: null },
  );
};
```

```
export const extractScoringCategories = (
  feedbacks: Feedback[],
): {
  seniorities: FeedbackSeniority[];
  skillScores: FeedbackSkillScore[][];
  scores: number[];
} =>
  feedbacks.reduce(
    (acc, curr) => {
      if (curr.seniority) {
        acc.seniorities.push(curr.seniority);
      }
      if (curr.score) {
        acc.scores.push(curr.score);
      }
      if (curr.skill_scores) {
        const skills = curr.skill_scores.filter((skill) => !!skill.score);
        acc.skillScores.push(skills);
      }
      return acc;
    },
    {
      seniorities: [],
      skillScores: [],
      scores: [],
    },
  );


import { DataMapper } from '../../shared/data-mapper';
```

```
import { Interview } from '../../database/entities/interview.entity';
import { CreateInterviewDto } from '../request-models/create-interview.dto';
import { InterviewDto } from '../response-models/interview.dto';
import { Injectable } from '@nestjs/common';
import { INTERVIEW_STATUS } from './interview.models';

@Injectable()
export class InterviewMapper extends DataMapper<Interview, CreateInterviewDto, InterviewDto> {
  public mapDtoToEntity(dto: CreateInterviewDto): Interview {
    const interview = new Interview();
    interview.status_id = INTERVIEW_STATUS.CREATED;
    interview.start_time = dto.startTime;
    interview.end_time = dto.endTime;
    interview.vacancy_id = dto.vacancyId;
    interview.author_id = dto.authorId;
    interview.candidate_name = `${dto.candidateDetails.firstName} ${dto.candidateDetails.lastName}`;
    return interview;
  }

  public mapEntityToDto(entity: Interview): InterviewDto {
    return {
      id: entity.id,
      status: entity.status?.name || 'Created',
      statusId: entity.status?.id || entity.status_id,
      startTime: entity.start_time,
      endTime: entity.end_time,
      candidateName: entity.candidate_name,
```

```
        attendees: [],
      };
    }
  }

  import { Injectable } from '@nestjs/common';
  import { Repository } from 'typeorm';
  import { Interview } from '../../database/entities/interview.entity';
  import { InjectRepository } from '@nestjs/typeorm';
  import { CreateInterviewDto } from '../request-models/create-interview.dto';
  import { UserService } from './user.service';
  import { InterviewUser } from '../../database/entities/interview-user.entity';
  import { VacancyService } from '../../vacancy/services/vacancy.service';
  import { InterviewMapper } from './interview.mapper';
  import { InterviewAttendeeDto } from '../response-models/interview-
atendee.dto';
  import { InterviewDto } from '../response-models/interview.dto';
  import {
    BadInputParameters,
    InvalidActionException,
    NotAllowedActionException,
    NotFoundException,
  } from 'src/modules/shared/exceptions';
  import { SearchInterviewsDto } from '../request-models/search-interviews.dto';
  import { InterviewLightDto } from '../response-models/interview-light.dto';
  import { UpdateInterviewDto } from '../request-models/update-interview.dto';
  import { UserDetailsDto } from '../response-models/user-details.dto';
  import { ExternalNotifierService } from './external-notifier.service';
  import { InterviewTokenDto } from '../request-models/interview-token.dto';
```

```
import { TokenService } from './token.service';
import { TokenDto } from '../response-models/token.dto';
import { FEEDBACK_STATUS, FeedbackService } from '../../feedback/services/feedback.service';
import { INTERVIEW_STATUS, INTERVIEW_USER_ROLES } from './interview.models';
import { Feedback } from '../../database/entities/feedback.entity';
import { extractScoringCategories, getMinAndMaxAmongScores } from './helpers';
import { InterviewScoringDto } from '../response-models/interview-scoring.dto';

@Injectable()
export class InterviewService {
  constructor(
    @InjectRepository(Interview)
    private interviewRepository: Repository<Interview>,
    @InjectRepository(InterviewUser)
    private interviewUserRepository: Repository<InterviewUser>,
    private userService: UserService,
    private vacancyService: VacancyService,
    private mapper: InterviewMapper,
    private notifierService: ExternalNotifierService,
    private tokenService: TokenService,
    private feedbackService: FeedbackService,
  ) {}

  async deleteOne(id: string, userId: string): Promise<void> {
    const interview = await this.getByIdForModification(id, userId);
```

```
    await this.interviewRepository.softRemove(interview);
  }

  async createNew(data: CreateInterviewDto, userId: string): Promise<InterviewDto> {
    const { candidateDetails, interviewersId, ...interviewData } = data;
    const candidate = await this.userService.createCandidate(candidateDetails);
    const interviewers = await this.userService.getUsers(interviewersId, userId);
    const vacancy = await this.vacancyService.getOne(interviewData.vacancyId, userId);
    await this.checkInterviewers(interviewersId, interviewers);
    const interviewCandidate = new InterviewUser();
    interviewCandidate.user_id = candidate.id;
    interviewCandidate.role_id = INTERVIEW_USER_ROLES.CANDIDATE;
    const interviewInterviewers = this.createInterviewers(interviewers.map(({ id }) => id));
    const interviewUsers = [interviewCandidate, ...interviewInterviewers];
    const interview = this.mapper.mapDtoToEntity({
      ...data,
      vacancyId: vacancy.id,
      authorId: userId,
    });
    interview.attendees = interviewUsers;
    await this.interviewRepository.save(interview);
    await this.notifierService.notifyAttendees(candidate, interviewers, interview);
    const resultDto = this.mapper.mapEntityToDto(interview);
```

```
        resultDto.attendees = this.mapAttendeesDetails([candidate, ...interviewers],
interviewUsers);
        return resultDto;
      }

      async updateOne(id: string, data: UpdateInterviewDto, userId: string):
Promise<InterviewDto> {
        const interview = await this.getByIdForModification(id, userId);
        interview.start_time = data.startTime || interview.start_time;
        interview.end_time = data.endTime || interview.end_time;
        const candidate = interview.attendees.find(
          (user) => user.role_id === INTERVIEW_USER_ROLES.CANDIDATE,
        );
        const attendeesIds = interview.attendees.map(({ user_id }) => user_id);
        const attendeesDetails = await this.userService.getUsers(attendeesIds,
userId);
        let candidateDetails = attendeesDetails.find(({ id }) => id ===
candidate.user_id);
        let interviewersDetails = attendeesDetails.filter((details) => details !==
candidateDetails);
        if (data.candidateDetails) {
          candidateDetails = await this.userService.updateCandidate(
            candidate.user_id,
            data.candidateDetails,
          );
          interview.candidate_name = candidateDetails.firstName + ' ' +
candidateDetails.lastName;
        }
        if (data.interviewersId) {
```

```
        const interviewers = interview.attendees.filter(
          (user) => user.role_id === INTERVIEW_USER_ROLES.INTERVIEWER,
        );
        const interviewersToCreate = data.interviewersId.filter(
          (id) => !interviewers.find((user) => user.user_id === id),
        );
        const interviewersToDelete = interviewers.filter(
          (user) => !data.interviewersId.includes(user.user_id),
        );
        const newInterviewersDetails = await this.userService.getUsers(interviewersToCreate, userId);
        await this.checkInterviewers(interviewersToCreate, newInterviewersDetails);
        const currentInterviewers = interviewers.filter(
          (user) => !interviewersToDelete.includes(user),
        );
        const newInterviewers = this.createInterviewers(interviewersToCreate);
        interviewersDetails = interviewersDetails
          .filter(({ id }) => !interviewersToDelete.find(({ user_id }) => user_id === id))
          .concat(newInterviewersDetails);
        interview.attendees = [candidate, ...newInterviewers, ...currentInterviewers];
        await this.interviewUserRepository.remove(interviewersToDelete);
      }
      await this.interviewRepository.save(interview);
      const resultDto = this.mapper.mapEntityToDto(interview);
      resultDto.attendees = this.mapAttendeesDetails(
        [candidateDetails, ...interviewersDetails],
```

```
        interview.attendees,
      );
      return resultDto;
    }

    async search(searchParams: SearchInterviewsDto, userId: string): Promise<InterviewLightDto[]> {
      const pageSize = searchParams.pageSize || 10;
      const page = searchParams.page || 1;
      const query = this.interviewRepository
        .createQueryBuilder('interview')
        .innerJoinAndSelect('interview.status', 'status')
        .innerJoinAndSelect('interview.vacancy', 'vacancy')
        .leftJoinAndSelect('interview.feedbacks', 'feedbacks')
        .leftJoinAndSelect('feedbacks.seniority', 'seniority')
        .leftJoinAndSelect('feedbacks.skill_scores', 'skill_scores')
        .leftJoinAndSelect('skill_scores.score', 'score')
        .leftJoinAndSelect('skill_scores.requirement', 'requirement')
        .innerJoin('interview.attendees', 'attendee')
        .where('(interview.author_id = :userId OR attendee.user_id = :userId)', { userId })
        .take(pageSize)
        .skip((page - 1) * pageSize);
      if (searchParams.query) {
        query.andWhere('vacancy.title ILIKE :searchTerm', { searchTerm: `%${searchParams.query}%` });
      }
      if (searchParams.candidateName) {
        query.andWhere('interview.candidate_name ILIKE :candidateName', {
```

```
        candidateName: `%${searchParams.candidateName}%`,
      });
    }
    if (searchParams.status) {
      query.andWhere('interview.status_id = :status', { status: searchParams.status
});
    }
    if (searchParams.vacancyId) {
      query.andWhere('interview.vacancy_id = :vacancy', { vacancy:
searchParams.vacancyId });
    }
    const result = await query.getMany();
    return result.map((item) => ({
      id: item.id,
      title: item.vacancy.title,
      candidate: item.candidate_name,
      startTime: item.start_time,
      endTime: item.end_time,
      status: item.status.name,
      statusId: item.status.id,
      createdTime: item.created_time,
      updatedTime: item.updated_time,
      scoring: this.calculateInterviewScore(item.feedbacks),
    }));
  }

  private calculateInterviewScore(feedbacks: Feedback[]): InterviewScoringDto
{
    const startedFeedbacks = feedbacks.filter(
```

```
      ({ status_id }) => status_id !== FEEDBACK_STATUS.NOT_EDITED,
    );
    if (startedFeedbacks.length === 0) {
      return null;
    }
    const { seniorities, skillScores, scores } =
extractScoringCategories(startedFeedbacks);
    const avgScore = scores.reduce((sum, curr) => sum + curr, 0) / scores.length;
    const arrangedSkillsScores = skillScores.reduce((acc, curr) => {
      curr.forEach((skillScore) => {
        acc[skillScore.requirement.name] = acc[skillScore.requirement.name] || [];
        acc[skillScore.requirement.name].push(skillScore.score);
      });
      return acc;
    }, {});
    const minAndMaxSkillsScores =
Object.keys(arrangedSkillsScores).reduce((acc, key) => {
      return { ...acc, [key]:
getMinAndMaxAmongScores(arrangedSkillsScores[key]) };
    }, {});

    const minAndMaxSeniorities = getMinAndMaxAmongScores(seniorities);
    return {
      score: avgScore,
      skills: minAndMaxSkillsScores,
      seniority: minAndMaxSeniorities,
    };
  }
```

```
  async getOne(id: string, userId: string): Promise<InterviewDto> {
    const interview = await this.getById(id, userId, [
      'feedbacks',
      'feedbacks.seniority',
      'feedbacks.skill_scores',
      'feedbacks.skill_scores.score',
      'feedbacks.skill_scores.requirement',
    ]);
    const scoring = this.calculateInterviewScore(interview.feedbacks);
    const vacancy = await this.vacancyService.getOne(interview.vacancy_id,
interview.author_id);
    const userIds = interview.attendees.map(({ user_id }) => user_id);
    const userDetails = await this.userService.getUsers(userIds, interview.id);
    const responseDto = this.mapper.mapEntityToDto(interview);
    responseDto.vacancy = vacancy;
    responseDto.scoring = scoring;
    responseDto.attendees = userDetails.map(
      ({ firstName, lastName, email, id, photo }): InterviewAttendeeDto => ({
        firstName,
        lastName,
        email,
        id: id,
        role: interview.attendees.find(({ user_id }) => user_id ===
id)?.role?.name,
        roleId: interview.attendees.find(({ user_id }) => user_id === id)?.role?.id,
        photo,
      }),
    );
    return responseDto;
```

```
    }

    private checkInterviewers(
      requestedInterviewers: string[],
      existingInterviewers: UserDetailsDto[],
    ): void {
      if (requestedInterviewers.length !== existingInterviewers.length) {
        const actualIds = existingInterviewers.map((interviewer) => interviewer.id);
        const wrongIds = requestedInterviewers.filter((id) =>
!actualIds.includes(id));
        throw new BadInputParameters('These interviewers doesnt exist: ' +
wrongIds.join(', '));
      }
    }

    private mapAttendeesDetails(
      usersDetails: UserDetailsDto[],
      attendees: InterviewUser[],
    ): InterviewAttendeeDto[] {
      return usersDetails.map(
        ({ firstName, lastName, email, id, photo }): InterviewAttendeeDto => ({
          firstName,
          lastName,
          email,
          id: id,
          role: attendees.find(({ user_id }) => user_id === id)?.role?.name,
          roleId: attendees.find(({ user_id }) => user_id === id)?.role?.id,
          photo,
        }),
```

```
      );
    }

    private createInterviewers(ids: string[]): InterviewUser[] {
      return ids.map((id) => {
        const interviewer = new InterviewUser();
        interviewer.user_id = id;
        interviewer.role_id = INTERVIEW_USER_ROLES.INTERVIEWER;
        return interviewer;
      });
    }
    private async getByIdForModification(id: string, userId: string):
Promise<Interview> {
      const interview = await this.getById(id, userId);
      if (interview.author_id !== userId) {
        throw new NotAllowedActionException('Forbidden resource');
      }
      if (interview.status_id !== INTERVIEW_STATUS.CREATED) {
        throw new InvalidActionException(
          'Can not update interview if status is has been already started',
        );
      }
      return interview;
    }

    private async getById(
      id: string,
      userId: string,
      additionalRelations: string[] = [],
```

```
  ): Promise<Interview> {
    const interview = await this.interviewRepository.findOne(id, {
      relations: ['status', 'attendees', 'attendees.role', ...additionalRelations],
    });
    this.checkInterviewAccess(interview, userId, id);
    return interview;
  }
  private checkInterviewAccess(interview, userId, interviewId): void {
    if (!interview) {
      throw new NotFoundException('Interview', interviewId);
    }
    if (
      interview.author_id !== userId &&
      !interview.attendees.find(({ user_id }) => user_id === userId)
    ) {
      throw new NotAllowedActionException('Forbidden resource');
    }
  }

  async generateInterviewAccessToken(data: InterviewTokenDto): Promise<TokenDto> {
    const interviewDuration = await this.getInterviewDuration(data.interviewId, data.userId);
    const payload = {
      ...data,
      interviewDuration,
    };
    const accessToken = await this.tokenService.getInterviewAccessToken(payload);
```

```
        return { accessToken };
      }

      private async getInterviewDuration(interviewId: string, userId: string): Promise<number> {
        const interview = await this.interviewRepository.findOne(interviewId, {
          relations: ['attendees'],
          select: ['id', 'start_time', 'end_time'],
        });
        this.checkInterviewAccess(interview, userId, interviewId);
        return Math.ceil(
          (new Date(interview.end_time).valueOf() - new Date(interview.start_time).valueOf()) / 1000,
        );
      }

      async updateStatus(id: string, status: number): Promise<void> {
        const interview = await this.interviewRepository.findOne(id, { relations: ['attendees'] });
        interview.status_id = status;
        if (status === INTERVIEW_STATUS.IN_PROGRESS) {
          const requirements = await this.vacancyService.getRequirements(interview.vacancy_id);
          const interviewers = interview.attendees
            .filter(({ role_id }) => role_id === INTERVIEW_USER_ROLES.INTERVIEWER)
            .map(({ user_id }) => user_id);
          interview.feedbacks = this.feedbackService.createEmptyFeedbacksForInterview(
```

```
            interview.id,
            interviewers,
            requirements,
          );
        }
        await this.interviewRepository.save(interview);
      }
    }

    import { HttpService, Injectable } from '@nestjs/common';
    import { ConfigService } from '@nestjs/config';
    import { GuestTokenDto } from '../request-models/guest-token.dto';
    import { TokenDto } from '../response-models/token.dto';
    import { InterviewTokenWithDurationDto } from '../request-models/interview-token-with-duration.dto';
    import { AccessTokenDto } from '../request-models/access-token.dto';
    import { ParsedGuestTokenDto } from '../request-models/parsed-guest-token.dto';
    import { NotAllowedActionException } from '../../shared/exceptions';

    @Injectable()
    export class TokenService {
      private readonly BASE_URL: string;

      constructor(private httpService: HttpService, private configService: ConfigService) {
        this.BASE_URL = configService.get('services.auth');
      }
```

```
  async getGuestInterviewToken(data: GuestTokenDto): Promise<string> {
    const response = await this.httpService
      .post<TokenDto>(this.BASE_URL + 'api/guest-access/generate', data)
      .toPromise();
    return response.data.accessToken;
  }

  async getInterviewAccessToken(data: InterviewTokenWithDurationDto):
Promise<string> {
    const response = await this.httpService
      .post<TokenDto>(this.BASE_URL + 'api/interview-access/generate', data)
      .toPromise();
    return response.data.accessToken;
  }

  async validateGuestInterviewToken(data: AccessTokenDto):
Promise<ParsedGuestTokenDto> {
    const response = await this.httpService
      .post<ParsedGuestTokenDto>(this.BASE_URL + 'api/guest-
access/validate', data)
      .toPromise();
    if (!response.data.isValid) {
      throw new NotAllowedActionException();
    }
    return response.data;
  }
}

import { HttpService, Injectable } from '@nestjs/common';
```

```
import { CandidateDetailsDto } from '../request-models/new-candidate-details.dto';
import { UserDetailsDto } from '../response-models/user-details.dto';
import { ConfigService } from '@nestjs/config';
import { UpdatedCandidateDetailsDto } from '../request-models/updated-candidate-details.dto';

@Injectable()
export class UserService {
  private readonly BASE_URL: string;
  private readonly CANDIDATE_ROLE_ID: string = '04649e48-fba2-46c5-b975-b340bde08919';

  constructor(private httpService: HttpService, private configService: ConfigService) {
    this.BASE_URL = configService.get('services.user');
  }
  async createCandidate(data: CandidateDetailsDto): Promise<UserDetailsDto> {
    const body = {
      ...data,
      roleId: this.CANDIDATE_ROLE_ID,
    };

    const response = await this.httpService
      .post<UserDetailsDto>(this.BASE_URL + 'api/user-details', body)
      .toPromise();
    return response.data;
  }
```

```
  async updateCandidate(id: string, data: UpdatedCandidateDetailsDto):
Promise<UserDetailsDto> {
    const response = await this.httpService
      .put<UserDetailsDto>(`${this.BASE_URL}api/user-details/${id}`, data)
      .toPromise();
    return response.data;
  }

  async getUsers(ids: string[], userId: string): Promise<UserDetailsDto[]> {
    const response = await this.httpService
      .get<UserDetailsDto[]>(this.BASE_URL + 'api/user-details', {
        params: { ids },
        headers: {
          'X-API-USER': userId,
        },
      })
      .toPromise();
    return response.data;
  }
}

import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { VacancyRequirement } from '../../database/entities/vacancy-
requirement.entity';
import { Repository } from 'typeorm';
import { VacancyRequirementMapper } from './vacancy-requirement.mapper';
```

```
@Injectable()
export class VacancyRequirementService {
 constructor(
  @InjectRepository(VacancyRequirement)
  private vacancyRequirementRepository: Repository<VacancyRequirement>,
  private mapper: VacancyRequirementMapper,
 ) {}

 getVacancyRequirements(vacancyId: string): Promise<VacancyRequirement[]> {
  return this.vacancyRequirementRepository.find({ vacancy_id: vacancyId });
 }
 async deleteMany(requirements: VacancyRequirement[]): Promise<void> {
  await this.vacancyRequirementRepository.remove(requirements);
 }
 createMany(skills: string[]): VacancyRequirement[] {
  return skills.map((skill) => this.mapper.mapDtoToEntity(skill));
 }
}

import { Injectable } from '@nestjs/common';
import { DataMapper } from '../../shared/data-mapper';
import { VacancyRequirement } from '../../database/entities/vacancy-requirement.entity';

@Injectable()
export class VacancyRequirementMapper extends DataMapper<VacancyRequirement, string, string> {
 public mapEntityToDto({ name }: VacancyRequirement): string {
```

```
    return name;
  }
  public mapDtoToEntity(name: string): VacancyRequirement {
    const requirement = new VacancyRequirement();
    requirement.name = name;
    return requirement;
  }
}

import { Injectable } from '@nestjs/common';
import { DataMapper } from '../../shared/data-mapper';
import { Vacancy } from '../../database/entities/vacancy.entity';
import { CreateVacancyDto } from '../request-models/create-vacancy.dto';
import { VacancyDto } from '../response-models/vacancy.dto';
import { VacancyRequirementMapper } from './vacancy-requirement.mapper';

@Injectable()
export class VacancyMapper extends DataMapper<Vacancy, CreateVacancyDto, VacancyDto> {
  constructor(private vacancyRequirementMapper: VacancyRequirementMapper) {
    super();
  }

  public mapEntityToDto(entity: Vacancy): VacancyDto {
    return {
      id: entity.id,
      title: entity.title,
      description: entity.description,
```

```
            createdTime: entity.created_time,
            updatedTime: entity.updated_time,
            authorId: entity.author_id,
            externalLink: entity.external_link,
            requirements:
entity.requirements.map(this.vacancyRequirementMapper.mapEntityToDto),
          };
        }

        public mapDtoToEntity(dto: CreateVacancyDto): Vacancy {
          const vacancy = new Vacancy();
          vacancy.author_id = dto.authorId;
          vacancy.description = dto.description;
          vacancy.external_link = dto.externalLink || '';
          vacancy.title = dto.title;
          vacancy.requirements                                    =
dto.requirements.map(this.vacancyRequirementMapper.mapDtoToEntity);
          return vacancy;
        }
      }

      import { Injectable } from '@nestjs/common';
      import { CreateVacancyDto } from '../request-models/create-vacancy.dto';
      import { Vacancy } from '../../database/entities/vacancy.entity';
      import { Repository } from 'typeorm';
      import { InjectRepository } from '@nestjs/typeorm';
      import { VacancyMapper } from './vacancy.mapper';
      import { VacancyDto } from '../response-models/vacancy.dto';
```

```
import { NotAllowedActionException, NotFoundException } from '../../shared/exceptions';
import { SearchVacanciesDto } from '../request-models/search-vacancies.dto';
import { UpdateVacancyDto } from '../request-models/update-vacancy.dto';
import { VacancyRequirement } from '../../database/entities/vacancy-requirement.entity';
import { VacancyRequirementService } from './vacancy-requirement.service';

@Injectable()
export class VacancyService {
  constructor(
    @InjectRepository(Vacancy)
    private vacancyRepository: Repository<Vacancy>,
    private mapper: VacancyMapper,
    private vacancyRequirementService: VacancyRequirementService,
  ) {}

  async createNew(data: CreateVacancyDto, authorId: string): Promise<VacancyDto> {
    const vacancy = this.mapper.mapDtoToEntity({ ...data, authorId });
    await this.vacancyRepository.save(vacancy);
    return this.mapper.mapEntityToDto(vacancy);
  }

  async deleteOne(id: string, userId: string): Promise<void> {
    const vacancy = await this.getById(id, userId);

    await this.vacancyRepository.softRemove(vacancy);
  }
```

```
  async getOne(id: string, userId: string): Promise<VacancyDto> {
    const vacancy = await this.getById(id, userId);
    return this.mapper.mapEntityToDto(vacancy);
  }

  async search(params: SearchVacanciesDto, userId: string):
Promise<VacancyDto[]> {
    const pageSize = params.pageSize || 10;
    const page = params.page || 1;
    const query = this.vacancyRepository
      .createQueryBuilder('vacancy')
      .where({ author_id: userId })
      .innerJoinAndSelect('vacancy.requirements', 'requirements')
      .take(pageSize)
      .skip((page - 1) * pageSize);
    if (params.query) {
      query.andWhere('(title ILIKE :query OR description ILIKE :query)', {
        query: `%${params.query}%`,
      });
    }
    const result = await query.getMany();
    return result.map(this.mapper.mapEntityToDto.bind(this.mapper));
  }

  async updateOne(id: string, data: UpdateVacancyDto, userId: string):
Promise<VacancyDto> {
    const vacancy = await this.getById(id, userId);
    vacancy.title = data.title || vacancy.title;
```

```
    vacancy.description = data.description || vacancy.description;
    vacancy.external_link = data.externalLink || vacancy.external_link;
    if (data.requirements) {
      vacancy.requirements = await this.getUpdatedRequirements(
        vacancy.requirements,
        data.requirements,
      );
    }
    await this.vacancyRepository.save(vacancy);
    return this.mapper.mapEntityToDto(vacancy);
  }

  getRequirements(vacancyId: string): Promise<VacancyRequirement[]> {
    return
this.vacancyRequirementService.getVacancyRequirements(vacancyId);
  }

  private async getUpdatedRequirements(
    currentSkills: VacancyRequirement[],
    skills: string[],
  ): Promise<VacancyRequirement[]> {
    const requirementsToCreate = skills.filter((skill) => {
      return !currentSkills.find((requirement) => requirement.name === skill);
    });
    const requirementsToDelete = currentSkills.filter((requirement) => {
      return !skills.includes(requirement.name);
    });
    const existingRequirements = currentSkills.filter((requirement) => {
      return !requirementsToDelete.includes(requirement);
```

```
      });
      const newRequirements = this.vacancyRequirementService.createMany(requirementsToCreate);
      await this.vacancyRequirementService.deleteMany(requirementsToDelete);
      return existingRequirements.concat(newRequirements);
    }

    private async getById(id: string, userId: string): Promise<Vacancy> {
      const vacancy = await this.vacancyRepository.findOne(id, { relations: ['requirements'] });
      if (!vacancy) {
        throw new NotFoundException('Vacancy', id);
      }
      if (vacancy.author_id !== userId) {
        throw new NotAllowedActionException();
      }
      return vacancy;
    }
  }
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПЛАТФОРМА ДЛЯ ПРОВЕДЕННЯ ТЕХНІЧНИХ ІНТЕРВ’Ю (КОМПЛЕКСНА ТЕМА). СИСТЕМА ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ІНТЕРВ’Ю.

Схема бази даних

КПІ.ІП-7207.045440.10.99.СБД

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.В. Ковтунець

Нормоконтроль:

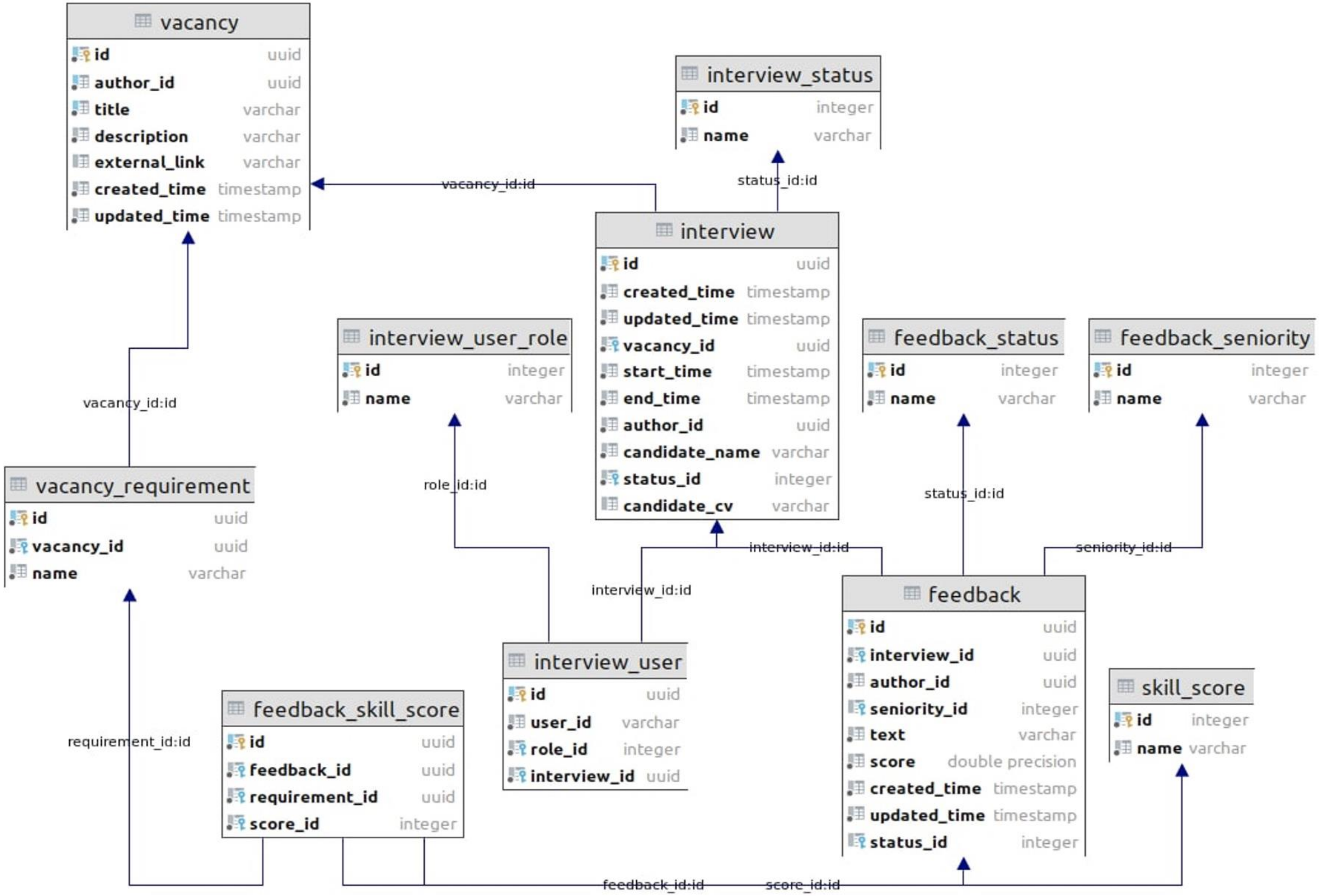
_____ І.І. Вітковська

Виконавець:

_____М.С. Гнатишин

Київ – 2021 року

КПІ.ІП-7207.045440.10.99.СБД



					КПІ.ІП-7207.045440.10.99.СБД			
Зм.	Арк.	№ документа	Підпис	Дата		Літера	Маса	Масштаб
Розробив		Гнатишин М.С.			Схема бази даних			
Перевірив		Ковтунець О.В.						
Т. кон.						Аркуш		Аркушів
Н. кон.		Вітковська І.І.			Платформа для проведення технічних інтерв'ю (комплексна тема). Система організації та проведення інтерв'ю.	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-72		
Затвердив		Ковтунець О.В.						

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

ПЛАТФОРМА ДЛЯ ПРОВЕДЕННЯ ТЕХНІЧНИХ ІНТЕРВ’Ю (КОМПЛЕКСНА ТЕМА). СИСТЕМА ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ІНТЕРВ’Ю

Креслення вигляду екранних форм

КПІ.ІП-7207.045440.11.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.В. Ковтунець

Нормоконтроль:

_____ І.І. Вітковська

Виконавець:

_____М.С. Гнатишин

Київ – 2021 року

Interview details

Add interviewers

Interviewers

Select date and time for the interview

Start date and time

dd.mm.yyyy, --:--

End date and time

dd.mm.yyyy, --:--

BACK NEXT

INTERVIEW

INTERVIEW

SAVE

Vacancy title

External link for vacancy

Type vacancy description...

REQUIREMENTS

Add required skills...

Please, add some information about candidate

First Name

first name

Last Name

last name

Work Email Address

candidateEmail@company.com

BACK NEXT

C++ Software Engineer (updated version)

Estimated seniority

Please estimate the candidate's skills

1	15	2	25	3	35	4	45	5

Type your feedback here...

SAVE AS DRAFT SAVE

skills

C/C++

OpenCV

Python

ONNX

PyTorch

English

Зм.	Арк.	№ документа	Підпис	Дата	Кресленяя вигляду екранних форм		
Розробив		Гнатишин М.С.				Літера	Маса	Масштаб
Перевірив		Ковтуньєль О.В.						
Т. кон.						Аркуш	Аркушів	
Н. кон.		Бітковська І.І.			Платформа для проведення технічних інтерв'ю (комплексна тема). Система організації та проведення інтерв'ю.	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-72		
Затвердив		Ковтуньєль О.В.						

КПІ.ІП-7207.045440.11.99.КЕ