МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНИЙ ПРОЄКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Система для призначення співбесід у IT-компаніях"

Виконавець: студент групи УС-412 Гузей Артур Едуардович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: Олександр ШЕВЧЕНКО

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2023р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Гузея Артура Едуардовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Система для призначення співбесід у IT-компаніях» затверджена наказом ректора від "01" травня 2023 р. за № 623/ст.

2. Термін виконання роботи: 15.05.2023 – 25.06.2023

3. Вихідні данні до роботи: система для призначення співбесід у IT-компаніях, яка допоможе рекрутерам та автоматизує процес найму нових фахівців.

4. Зміст пояснювальної записки: вступ, оцінка переваг автоматизації процесу найму нових фахівців, порівняльна оцінка із відомими системами, вибір технологій та інструментів для реалізації поставленої мети, розробка системи для планування співбесід, налаштування системи для планування співбесід, для її подальшого зручного користування.

5. Перелік обов'язкового графічного матеріалу: контролер кандидата, сторінка кандидата, файл Dockerfile, Файл docker-compose.yml.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Проаналізувати літературу та джерела за темою дипломного проекту.	15.05.2023 – 17.05.2023	
2.	Розроблення та затвердження плану дипломного проекту.	18.05.2023 – 20.05.2023	
3.	Провести консультації з науковим керівником щодо створення першого розділу.	21.05.2023 – 23.05.2023	
4.	Розробка розділу 1	24.05.2023 – 02.06.2023	
5.	Розробка розділу 2	03.06.2023 – 09.06.2023	
6.	Розробка розділу 3	10.06.2023 – 13.06.2023	
7.	Висновки та оформлення пояснювальної записки дипломного проекту.	14.06.2023 – 16.05.2023	
8.	Підписання необхідних документів у встановленому порядку.	17.05.2023 – 19.05.2023	

7. Дата видачі завдання: «15» травня 2023 р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Артур ГУЗЕЙ
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система для призначення співбесід у IT-компаніях» складається зі вступу, трьох розділів, висновку, списку бібліографічних посилань і містить 108 сторінок, 1 таблицю та 45 рис. Список бібліографічних посилань складається з 15 найменувань.

Актуальність. Месенджери й аналогічні системи для призначення співбесід, що використовуються нині, не відповідають вимогам цієї галузі та не вирішують усіх проблем планування. Тому розробка повністю автоматизує процес і вирішує всі проблеми в цій галузі.

Об'єктом дослідження є процес створення системи для планування співбесід у IT-компаніях.

Предметом дослідження є засоби створення системи для планування співбесід у IT-компаніях.

Метою дипломної роботи є проектування та розробка системи для планування співбесід у IT-компаніях, яка допоможе рекрутерам спростити та автоматизувати процес найму нових фахівців.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

- визначення технологій і інструментів для розробки системи;
- проектування архітектури системи та взаємодії між компонентами системи;
- розробка системи для планування співбесід у IT-компаніях;
- налаштування системи для її зручного використання.

Методи дослідження включають у себе:

- методи проектування клієнт-серверної архітектури системи;
- методи розробки серверного додатку;
- методи розробки клієнтської частини системи;
- методи авторизації і аутентифікації користувача у системі;
- методи проектування та створення бази даних системи;
- методи налаштування системи для її зручного використання.

Теоретичною основою дипломної роботи стали сучасні технології та мови програмування, що зроблять систему гнучкою, продуктивною та здатною до масштабування.

Теоретична і практична значимість роботи полягає в тому, що на основі отриманих знань:

1) можна в короткий термін ознайомитися з необхідною теорією використаних технологій, що може допомогти програмісту при складанні власної системи;

2) по зібраному матеріалу можна в короткий термін продублювати процес розробки та проектування системи, що допоможе підвищити навички програмування студента чи програміста;

3) по забраному матеріалу можна продублювати розроблену систему та використовувати її у процесі планування співбесід у IT-компаніях.

На захист виносяться наступні положення:

1) процес проектування та розробки системи;

2) процес налаштування системи для її зручного використання;

3) система для планування співбесід у IT-компаніях.

РЕКРУТИНГ, ДОДАТОК, АВТОМАТИЗОВАНА СИТЕМА, JAVA, SPRING FRAMEWORK, FACEBOOK API, HIBERNATE, DOCKER, OAUTH 2.0, INTELLIJ IDEA, HTML, CSS, JAVASCRIPT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface)	різновид програмного інтерфейсу, який пропонує послуги іншим
ACID	стандарт транзакції
IT-компанія	підприємство інформаційних технологій
CSS (Cascading Style Sheets)	мова стилів
HTML	гіпертекстова мова розмітки
JWT	відкритий стандарт для створення токенів
JS (JavaScript)	мова програмування
OAuth2	протокол відкритого стандарту
Sass (Syntactically Awesome Style Sheets)	скриптова метамова, яка інтерпретується в CSS

ВСТУП

IT-індустрія - це галузь, яка швидко розвивається і постійно змінюється, в якій постійно з'являються нові технології та проекти, що вимагають спеціальних навичок. На сьогоднішньому конкурентному ринку праці компанії постійно прагнуть наймати найкращих фахівців, щоб не відстати від конкурентів. Однак в умовах постійного потоку звільнених і найнятих фахівців процес підбору персоналу може бути складним і тривалим. Однією з областей, де це особливо помітно, є планування співбесід, яке може бути складним і громіздким процесом, особливо для великих IT-компаній. Цей процес може бути тривалим і дорогим, і часто буває важко знайти потрібних кандидатів на роботу. Крім того, як відповідні кандидати знайдені, може бути складно призначити з ними співбесіди та скоординувати дії різних зацікавлених сторін, залучених до процесу найму на роботу. Також, у сучасній IT-індустрії кожен розробник не затримується на одному проекті довше, ніж на 1,5 роки. Це пов'язано з тим, що тривала робота над одним і тим же кодом і технологіями може призвести до вигорання і втрати мотивації співробітника.

Складання графіка співбесід є складним і тривалим процесом з різних причин. По-перше, в умовах стрімкого характеру галузі, що постійно змінюється, рекрутерам необхідно швидко знаходити і зв'язуватися з потенційними кандидатами, що може виявитися непростим завданням, особливо під час роботи з великою кількістю резюме. По-друге, узгодження розкладу кандидата та інтерв'юера може стати логістичним жахіттям, особливо якщо в співбесіді беруть участь кілька інтерв'юерів або кандидат недоступний у певний час. По-третє, ручний характер складання розкладу може призвести до помилок або непорозумінь, що може стати причиною втрачених можливостей як для компанії, так і для кандидата.

Потреба в автоматизованій системі планування інтерв'ю стає все більш актуальною в останні роки. З розвитком технологій з'явилася можливість автоматизувати багато завдань, пов'язаних з процесом найму на роботу. Це може

допомогти заощадити час і зменшити навантаження на рекрутерів, дозволивши їм зосередитися на інших важливих завданнях.

Ці фактори роблять важливим для IT-компаній наявність ефективної та автоматизованої системи планування співбесід. Тому необхідність у системі складання розкладу співбесід стає все більш актуальною.

У цій дипломній роботі ми розглянемо розробку клієнт-серверного додатка для планування співбесід в IT-компаніях. Додаток матиме три основні ролі: кандидат, інтерв'юер і координатор. Кандидат та інтерв'юер зможуть призначати часові інтервали, коли вони вільні для співбесіди, а координатор відповідатиме за бронювання певного часового інтервалу, коли вільні і кандидат, і інтерв'юер, і таким чином призначає час, коли відбудеться співбесіда, а також керувати ролями системи. Система покликана впорядкувати процес планування співбесід, полегшуючи IT-компаніям управління процесом рекрутингу та спростить процес планування співбесід.

Серверна частина додатку являє собою мікросервісний додаток, який легкий для розуміння та подальшого розширення, написаний на мові Java з використанням наступних технологій та інструментів розробки: Spring Boot, Spring Data, Spring Security, Spring Web, Hibernate, PostgreSQL, OAuth2, Facebook API, Postman, Docker.

Клієнтська частина - це звичайні веб-сторінки, написані на HTML, CSS, JavaScript з використанням бібліотеки axios, адже основна увага в цій роботі буде приділена серверній частині додатка, а клієнтська частина буде використовуватися для демонстрації роботи сервера.

Система розроблена таким чином, щоб бути простою у використанні, інтуїтивно зрозумілою та відкритою для розширення і впровадження нових функцій, за потребою певного користувача. Кандидати та інтерв'юери можуть призначати часові інтервали, коли вони доступні для співбесіди. Координатор може переглянути доступні часові інтервали і забронювати час, який підходить як для кандидата, так і для інтерв'юера, а також керувати ролями у системі.

На даний момент, зазвичай, рекрутери для планування співбесід використовують звичайні месенджери для узгодження часу його проведення з

кандидатом та окремо з інтерв’юером. Це відбувається тому що наразі нема зручної системи, яка б могла автоматизувати цей процес, бо наявні схожі системи не відповідають специфіці цієї галузі та не вирішують усі проблеми цього процесу у повному обсязі. Виходячи з цього, можна зробити висновок, що система для планування співбесід у IT-компаніях є новітньою розробкою, яка допоможе рекрутерам та повністю автоматизує процес найму фахівців.

Розроблена нами система планування співбесід має кілька переваг для IT-компаній. По-перше, вона спрощує процес планування співбесід, роблячи його швидшим та ефективнішим. Це може допомогти скоротити час і ресурси, необхідні для процесу найму, що дозволить компаніям зосередитися на інших важливих завданнях.

По-друге, система надає можливість керувати розкладом співбесід і гарантувати, що співбесіди проводяться з правильними кандидатами на правильні посади, також може допомогти компаніям наймати найкращих кандидатів на відкриті вакансії.

РОЗДІЛ 1
АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

1.1. Перевага автоматизації планування співбесід

Автоматизація процесів визначається як використання технологій для зменшення ручної праці та економії часу. По суті, це використання технологій для скорочення людської праці та вивільнення часу для більш важливих завдань. У контексті співбесіди автоматизоване планування може заощадити час і ресурси, а також підвищити точність і ефективність.

Впровадження автоматизованого планування співбесіди може запропонувати численні переваги. Автоматизоване планування може значно скоротити час процесу найму. Наприклад, замість того, щоб вручну координувати час співбесіди між кількома інтерв'юерами та кандидатами, всі завдання з планування можуть бути автоматизовані. Така автоматизація може заощадити час, а також зменшити ризик помилки через ручне введення даних.

Автоматизоване планування співбесід також пропонує підвищену точність. Автоматизація може допомогти забезпечити правильність введення даних, їхню актуальність і точність. Ця точність може допомогти забезпечити правильне планування інтерв'ю та проінформувати всі залучені сторони про правильну дату та час проведення інтерв'ю. Крім того, автоматизоване планування може допомогти зменшити ризик конфліктів у розкладі, оскільки всі сторони бачать однакові дані.

Автоматизація процесу планування співбесід також допомагає покращити досвід кандидатів. Кандидати можуть легко отримати доступ до системи планування та обрати зручний для них час співбесіди. Система також може надавати додаткову інформацію, наприклад, про місце проведення співбесіди та про те, чого очікувати

під час співбесіди. Це допомагає підвищити впевненість кандидата та зменшити будь-яку тривогу, пов'язану з процесом співбесіди.

Переваги автоматизації процесу планування співбесіди стосуються не лише кандидатів та інтерв'юерів. Менеджери з підбору персоналу також можуть отримати вигоду від автоматизації, маючи чіткий огляд розкладу співбесід, і вони можуть швидко побачити, які кандидати були призначені на співбесіду, зменшуючи ризик пропущених співбесід.

Автоматизація планування також допомагає уникнути конфліктів у розкладі та подвійного бронювання, зменшуючи потребу у ручному втручанні. Це означає, що система може впоратися з великою кількістю інтерв'ю, не потребуючи додаткового персоналу для управління процесом. Крім того, автоматизована система може надавати кандидатам та інтерв'юерам інформацію в режимі реального часу, покращуючи комунікацію між двома сторонами.

Нарешті, автоматизований розклад може допомогти заощадити ресурси. Автоматизоване планування може допомогти зменшити кількість часу та грошей, необхідних для ручної координації між кількома сторонами. Автоматизоване планування пропонує ефективний та економічно вигідний спосіб управління процесом планування.

Отже, автоматизоване планування співбесід має багато переваг. Автоматизація може допомогти скоротити час процесу, підвищити точність, полегшити комунікацію та заощадити ресурси. Автоматизоване планування може допомогти гарантувати, що співбесіди заплановані правильно, і що всі сторони знають правильний час і дату співбесіди. Зрештою, автоматизоване планування може допомогти забезпечити безперебійний та ефективний процес співбесіди.

1.2. Порівнялна оцінка із відомими системами

Система планування співбесід в IT-компаніях, розроблена в цій дипломній роботі, є унікальним та інноваційним рішенням, спрямованим на спрощення та оптимізацію процесу планування співбесід як для кандидатів, так і для інтерв'юерів.

Ця система відрізняється від традиційних методів планування співбесід, які вимагають ручної координації між кандидатом, інтерв'юером і менеджером із найму. Ця система автоматизує процес складання розкладу співбесід, скорочуючи час і зусилля, необхідні для узгодження розкладу, і гарантуючи, що кандидати та інтерв'юери зможуть знайти відповідні часові інтервали, що відповідають їхнім можливостям.

Хоча існують й інші подібні системи для планування співбесід, як Calendly і Doodle, їм бракує специфічних функцій і можливостей, які роблять цю систему унікальною. Calendly - це інструмент планування, який дає змогу користувачам призначати зустрічі та наради з іншими людьми, повідомляючи про свою зайнятість через посилання. Однак Calendly не надає платформи для специфічних потреб планування співбесід, як координація кількох сторін, що беруть участь у процесі найму, і не забезпечує функції безпеки, необхідні для планування співбесід у професійному середовищі. Doodle, з іншого боку, є платформою планування, яка дає змогу користувачам створювати та керувати опитуваннями для планування зустрічей і нарад. Однак Doodle обмежений у своїй функціональності та не надає необхідних можливостей для планування співбесід, як управління профілями кандидатів та інтерв'юерів, управління конфліктами під час планування співбесід і забезпечення безпечного доступу до конфіденційних даних.

Порівняно з цими системами, система, розроблена в цій дипломній роботі, являє собою комплексне і спеціалізоване рішення для складання розкладу співбесід в IT-індустрії. Ця система спеціально розроблена для задоволення унікальних вимог процесу найму, з трьома окремими ролями: кандидат, інтерв'юер і координатор. Система надає платформу для управління профілями кандидатів та інтерв'юерів, планування співбесід і розв'язання конфліктів, що можуть виникнути в процесі планування співбесід. Крім того, система забезпечує функції безпеки, необхідні для управління конфіденційними даними, включно з інтеграцією OAuth2 і Facebook API, що гарантує конфіденційність інформації про призначення інтерв'ю.

Загалом, система, розроблена в цій дипломній роботі, пропонує унікальне і комплексне рішення для складання розкладу співбесід в IT-індустрії. Ця система

надає функції та можливості, яких не вистачає в традиційних інструментах складання розкладу та інших подібних системах. Система покликана оптимізувати і спростити процес найму, скоротити час і зусилля, необхідні для складання розкладу співбесід, і гарантувати, що кандидати та інтерв'юери зможуть знайти підходящі тимчасові інтервали, які відповідають їхнім можливостям. Розробка цієї системи необхідна для задоволення специфічних потреб IT-індустрії, а також для підвищення ефективності та результативності процесу найму як для кандидатів, так і для роботодавців.

1.3. Вибір технологій та інструментів

1.3.1. Архітектура системи для планування співбесід

Система планування співбесід в IT-компаніях являє собою клієнт-серверний додаток. У цьому типі архітектури є два типи компонентів: клієнтська сторона і серверна сторона. Клієнтська сторона відповідає за обробку користувацького інтерфейсу і взаємодію з користувачем, тоді як серверна сторона відповідає за управління зберіганням, обробкою і логікою даних.

Для цієї дипломної роботи ми вирішили використовувати архітектуру клієнт-сервер, оскільки вона забезпечує надійне, масштабоване і безпечне розв'язання проблеми складання розкладу співбесід. Вона також дає змогу чітко розділити завдання між компонентами на стороні клієнта і на стороні сервера, що полегшує супровід і модифікацію програми в майбутньому.

Слід зазначити, що архітектура клієнт-сервер забезпечує надійне та масштабоване рішення для створення системи планування співбесід в IT-компаніях. Вибравши цю архітектуру побудови системи, ми зможемо створити високоякісну систему, яка забезпечує безпечний і зручний спосіб планування співбесід в IT-компаніях.

1.3.2. Огляд технологій і інструментів клієнтської частини системи

1.3.2.1. Огляд HTML, CSS, JavaScript

HTML, каскадні таблиці стилів (CSS) і JavaScript є необхідними інструментами для створення клієнтської частини системи планування співбесід. HTML - це мова розмітки, яка використовується для створення структури веб-сторінки. Вона використовується для створення базової структури веб-сторінки, а також для визначення змісту та елементів веб-сторінки. CSS використовується для стилізації елементів і вмісту веб-сторінки. Її використовують для визначення зовнішнього вигляду веб-сторінки та надання їй більш привабливого і наочного вигляду. JavaScript - це мова сценаріїв, яка використовується для надання інтерактивності веб-сторінці. Вона використовується для додавання динамічних елементів на веб-сторінку, таких як форми, меню та анімація.

HTML має низку переваг, які роблять його чудовою мовою для створення структури веб-сторінки. По-перше, він відносно простий у вивченні та використанні. По-друге, він є відкритим стандартом, що означає, що його можна використовувати на будь-якій платформі, не турбуючись про проблеми сумісності. По-третє, він дуже гнучкий і може використовуватися для створення складних веб-сторінок.

CSS також є чудовою мовою для створення стилю веб-сторінки. Вона проста у вивченні та використанні, а також є відкритим стандартом. CSS забезпечує більшу гнучкість в оформленні веб-сторінки, і його можна використовувати для створення складних макетів і візуальних елементів.

JavaScript - це важливий інструмент для додавання інтерактивності на веб-сторінку. Він дає змогу створювати динамічні елементи, як форми, меню та анімацію. JavaScript також є відкритим стандартом і відносно простий у вивченні та використанні.

Поєднання HTML, CSS і JavaScript є потужним інструментом для створення клієнтської частини системи планування співбесід. HTML використовується для

створення структури веб-сторінки, CSS використовується для стилізації веб-сторінки та надання їй візуальної привабливості, а JavaScript використовується для додавання інтерактивності веб-сторінці. Таке поєднання мов дає потужний інструмент для створення зручного та візуально привабливого інтерфейсу для системи.

Недоліків у цих мов небагато, але їх слід мати на увазі під час розроблення системи. По-перше, HTML і CSS можуть бути складними у вивченні та використанні. По-друге, вони не такі потужні, як інші мови, такі як Java або C++. По-третє, JavaScript може повільно виконуватися і його важко налагоджувати.

HTML, CSS і JavaScript є найбільш підходящими інструментами для створення клієнтської частини системи планування співбесід. Їх легко вивчити й використовувати, вони забезпечують потужний і гнучкий інструмент для створення зручного та візуально привабливого інтерфейсу системи. Вони також забезпечують необхідну інтерактивність, яка потрібна для системи. [11, 12, 13]

1.3.2.2. Огляд бібліотеки Axios

Бібліотека Axios - це популярна бібліотека JavaScript, яка використовується для виконання HTTP-запитів з боку клієнта веб-додатків. У контексті нашого проекту ми використовуємо її для зв'язку із серверним додатком мікросервісу. Axios має низку переваг, які роблять його вдалим вибором для нашого випадку використання.

По-перше, Axios підтримує обіцянки, що спрощує обробку асинхронних операцій у JavaScript. Це означає, що ми можемо писати чистіший і лаконічніший код під час виконання HTTP-запитів. Крім того, Axios підтримує синтаксис async/await, що ще більше спрощує роботу з асинхронними операціями.

По-друге, Axios має простий API, який легко використовувати і зрозуміти. Його API побудований на концепції запитів і відповідей, що робить його інтуїтивно зрозумілим для розробників. Крім того, він має багатий набір опцій для налаштування запитів, таких як установка заголовків або методів запиту.

По-третє, Axios забезпечує вбудовану підтримку обробки помилок, таких як мережеві помилки або помилки HTTP. Він також дозволяє нам визначати власну логіку обробки помилок, що робить його більш гнучким і налаштованим.

Axios широко використовується і має велике співтовариство розробників, які його підтримують. Це означає, що він добре підтримується і регулярно оновлюється, що гарантує швидке усунення будь-яких проблем або помилок.

Однак є і деякі потенційні недоліки використання Axios. Одним з основних є те, що Axios може призвести до збільшення розміру коду на стороні клієнта. Це може турбувати користувачів із повільним інтернет-з'єднанням або пристроями з обмеженими ресурсами. Крім того, Axios не підтримує старі браузери, тому нам, можливо, доведеться використовувати запасний варіант для користувачів старих браузерів.

Незважаючи на ці потенційні недоліки, ми вважаємо, що переваги використання Axios переважують недоліки для нашого конкретного випадку використання. Простота і гнучкість API Axios роблять його чудовим вибором для взаємодії з нашим серверним додатком мікросервісу. Крім того, можливість опрацювання помилок і велика спільнота, що підтримує бібліотеку, дають нам упевненість у тому, що це надійне і міцне рішення. [14]

1.3.3. Огляд технологій і інструментів серверної частини системи

1.3.3.1. Огляд мови програмування Java

Однією з ключових переваг Java є його крос-платформна сумісність. Код Java може працювати на різних операційних системах, що означає, що розробникам не потрібно створювати різні версії одного й того самого застосунку для кожної платформи. Це також полегшує підтримку додатка, оскільки розробникам потрібно оновлювати лише одну версію коду.

Ще однією перевагою Java є її безпека. Java-додатки виконуються в "пісочниці", яка ізолює додаток від операційної системи та інших додатків. Це

гарантує, що додаток не завдасть шкоди системі, на якій він працює. Крім того, Java має потужні можливості шифрування, що робить її придатною для додатків, які потребують високого рівня безпеки.

Java також відома своїми великими бібліотеками і фреймворками, які роблять розробку більш швидкою та ефективною. Наприклад, фреймворк Spring широко використовується під час розроблення корпоративних застосунків, надаючи такі можливості, як упровадження залежностей, архітектура MVC і RESTful веб-сервіси. Hibernate - ще один популярний фреймворк, який забезпечує об'єктно-реляційне відображення і спрощує доступ до бази даних у Java-додатках.

Незважаючи на численні переваги, у Java є й недоліки. Одним з основних критичних зауважень на адресу Java є те, що він може бути повільним і вимогливим до пам'яті порівняно з іншими мовами. Однак, завдяки сучасним досягненням у галузі апаратного забезпечення та методів оптимізації, цей недолік стає все менш актуальним.

У нашій дипломній роботі ми вирішили використовувати Java для розробки серверної частини нашого додатка з кількох причин. По-перше, крос-платформна сумісність Java гарантує, що наш додаток може працювати на широкому спектрі платформ, що важливо для масштабованості системи. По-друге, функції безпеки Java роблять його придатним для додатка, який працюватиме з конфіденційною інформацією, наприклад, з розкладом співбесід. Нарешті, великі бібліотеки та фреймворки Java забезпечують надійне середовище розробки, що робить розробку системи швидшим та ефективнішим.

Java є популярною мовою програмування для створення додатків корпоративного рівня завдяки своїй крос-платформенній сумісності, функціям безпеки, широким бібліотекам і фреймворкам. Хоча у неї є деякі недоліки, переваги Java роблять її найбільш переконлевим рішенням для розробки нашої системи планування співбесід.

1.3.3.2. Огляд Spring Framework

Spring Framework - це широко використовувана платформа додатків з відкритим вихідним кодом, яка забезпечує комплексну інфраструктурну підтримку для створення додатків корпоративного рівня. У нашій дипломній роботі ми використовували кілька компонентів Spring Framework для створення надійного клієнт-серверного додатка для планування співбесід в IT-компаніях.

Першим і головним компонентом, який ми використовували, є фреймворк Spring Boot, який являє собою потужний і гнучкий інструмент для створення автономних, готових до виробництва додатків. Spring Boot дає нам змогу створити мікросервісну архітектуру для нашого застосунку, яка є популярним шаблоном проєктування для створення масштабованих і стійких застосунків. За допомогою Spring Boot ми можемо легко упакувати наш застосунок в один виконуваний JAR-файл, що значно спрощує розгортання та обслуговування.

Ще один компонент, який ми використовували, - це фреймворк Spring Data, який забезпечує послідовний і простий у використанні інтерфейс для роботи з різними сховищами даних. Ми використовували Spring Data для взаємодії з нашою базою даних PostgreSQL, у якій зберігається вся інформація, пов'язана зі співбесідами та їхнім розкладом. Spring Data допомагає нам позбутися шаблонізованого коду для взаємодії з базою даних і забезпечує надійний і ефективний спосіб управління операціями з базою даних.

Ми також використовували Spring Security, який є потужним і гнучким фреймворком для управління безпекою та контролем доступу в нашому застосунку. За допомогою Spring Security ми можемо легко реалізувати механізми автентифікації та авторизації, щоб захистити наш застосунок від несанкціонованого доступу. Ми використовували Spring Security для реалізації аутентифікації OAuth2, яка дає змогу користувачам входити в систему за допомогою облікового запису Facebook.

Spring Web - це ще один компонент Spring Framework, який ми використовували для створення RESTful API для нашого застосунку. Spring Web

надає гнучкий і потужний спосіб створення веб-додатків з використанням архітектури Model-View-Controller (MVC). Ми використовували Spring Web для реалізації всіх кінцевих точок нашого API, що дає змогу веб-сторінкам на стороні клієнта взаємодіяти з додатком на стороні сервера.

Загалом, Spring Framework надає потужний і гнучкий набір інструментів для створення додатків корпоративного рівня. Однією з головних переваг Spring Framework є його модульність, яка дає нам змогу використовувати тільки ті компоненти, які нам потрібні, і уникати непотрібних залежностей. Іншою перевагою є велика документація та підтримка спільноти, що робить його простим у вивченні та використанні.

Однак одним із потенційних недоліків Spring Framework є його вивчення, особливо для новачків. Spring Framework має велику кількість компонентів і функцій, які можуть виявитися непосильними для розробників, не знайомих із ним. Крім того, деякі компоненти Spring Framework, як Spring Security, можуть бути складними і вимагають глибокого розуміння базових концепцій.

Незважаючи на ці проблеми, ми вважаємо, що Spring Framework є найбільш підходящим рішенням для нашого завдання зі створення клієнт-серверного застосунку для планування співбесід в IT-компаніях. Spring Framework надає потужний і гнучкий набір інструментів, який дає нам змогу з легкістю побудувати масштабований, стійкий і безпечний застосунок. Модульність і велика документація Spring Framework роблять його простим у вивченні та використанні. Крім того, Spring Framework має велике й активне співтовариство, яке забезпечує підтримку та ресурси для розробників, що використовують його.

Ще однією причиною, через яку ми обрали Spring Framework для нашого застосунку, є його сумісність з іншими технологіями та інструментами. Наприклад, ми змогли легко інтегрувати API Facebook у наш застосунок за допомогою компонента Spring Security. Використання Docker дало нам змогу контейнеризувати наш застосунок і з легкістю розгорнути його в різних середовищах. [1, 2, 3, 4]

1.3.3.3. Огляд Hibernate та PostgreSQL

При розробці нашого клієнт-серверного додатка для складання розкладу співбесід, ми вирішили використати Hibernate і PostgreSQL для нашої системи керування базами даних.

PostgreSQL - це популярна система управління реляційними базами даних з відкритим вихідним кодом, яка пропонує широкий спектр можливостей і функціональності. Вона відома своєю надійністю, стабільністю і масштабованістю, що робить її чудовим вибором для додатків корпоративного рівня, таких як наша система.

Переваги PostgreSQL включають:

1. Висока продуктивність: PostgreSQL оптимізовано для високої продуктивності, що робить його популярним вибором для додатків, які потребують швидкого доступу до даних та їх обробки.

2. Відповідність стандарту ACID: PostgreSQL повністю відповідає стандарту ACID, що означає, що транзакції обробляються надійним і послідовним чином. Це гарантує збереження цілісності даних навіть у разі системних збоїв або помилок.

3. Розширені можливості: PostgreSQL пропонує широкий спектр розширених можливостей, включно з підтримкою складних запитів, повнотекстового пошуку та просторових даних. Це робить її чудовим вибором для додатків, що вимагають складної обробки даних.

До недоліків PostgreSQL належать:

1. Складність: Хоча PostgreSQL пропонує широкий спектр можливостей, він може бути складним у встановленні та налаштуванні. Це може вимагати спеціальних знань або досвіду, що може збільшити час розробки.

2. Обмежена підтримка: Хоча PostgreSQL має велике й активне співтовариство користувачів і розробників, він не може запропонувати такий самий рівень підтримки або ресурсів, як комерційні системи управління базами даних. Це може вимагати додаткових ресурсів або досвіду для управління та підтримки.

Hibernate - це Java-фреймворк із відкритим вихідним кодом, який спрощує процес зіставлення Java-об'єктів із реляційними базами даних. Він забезпечує зручний спосіб роботи з базами даних в об'єктно-орієнтованому програмуванні, скорочуючи обсяг повторюваного і схильного до помилок коду, необхідного для управління даними.

Переваги Hibernate включають:

1. Спрощений доступ до бази даних: Hibernate спрощує процес доступу до баз даних і управління ними, зменшуючи кількість необхідного коду. Це полегшує роботу з базами даних в об'єктно-орієнтованому програмуванні без шкоди для продуктивності та функціональності.

2. Переносимість: Hibernate має високу переносимість, що означає, що його можна використовувати з широким спектром баз даних і мов програмування. Це робить його чудовим вибором для додатків, що вимагають гнучкості та масштабованості.

3. Автоматична персистентність: Hibernate автоматично керує персистентністю, тобто зміни об'єктів автоматично зберігаються в базі даних без необхідності написання додаткового коду. Це скорочує кількість часу і зусиль, необхідних для управління даними, що робить його популярним вибором серед розробників.

До недоліків Hibernate належать:

1. Накладні витрати на продуктивність: Hibernate додає деякі накладні витрати на продуктивність, що означає, що він може бути не кращим вибором для додатків, які потребують надзвичайно високої продуктивності або низької затримки.

2. Складність: Hibernate може бути складним у налаштуванні та використанні, вимагаючи спеціальних знань або досвіду. Це може збільшити час розробки, особливо для менш досвідчених розробників.

Ми вирішили використовувати Hibernate і PostgreSQL для нашого клієнт-серверного додатка, оскільки вони пропонують широкий спектр можливостей, високу продуктивність і масштабованість. Хоча для управління і налаштування цих технологій можуть знадобитися спеціальні знання або досвід, ми вважаємо, що ці

технології є найбільш привабливим рішенням для нашої системи. Завдяки їхнім розширеним можливостям і функціональності ми впевнені, що зможемо створити надійну й ефективну систему для планування співбесід в IT-компаніях. [5, 6]

1.3.3.4. Огляд OAuth2 та Facebook API

OAuth2 - це протокол відкритого стандарту, який дає змогу користувачам ділитися своїми приватними ресурсами зі сторонніми додатками, не повідомляючи їм свої паролі або іншу конфіденційну інформацію. Він широко використовується під час розроблення сучасних веб-додатків і дає змогу забезпечити безпеку процесів автентифікації та авторизації.

Однією з головних переваг OAuth2 є його гнучкість. Він дає змогу користувачам використовувати різні провайдери автентифікації, включно з обліковими записами соціальних мереж (наприклад, Facebook і Google), без необхідності вводити кілька імен і паролів. Крім того, OAuth2 забезпечує вищий рівень безпеки, запобігаючи несанкціонованому доступу до конфіденційних даних користувачів. Він також спрощує процес інтеграції сторонніх застосунків у нашу систему, надаючи стандартизований підхід до аутентифікації та авторизації.

З іншого боку, у OAuth2 є і деякі недоліки. Він може бути доволі складним у реалізації та вимагає хорошого розуміння протоколів безпеки. Крім того, він може не підходити для всіх випадків використання, наприклад, для тих, які вимагають більш детального контролю доступу.

Facebook API, з іншого боку, являє собою набір програмних інтерфейсів та інструментів, що надаються Facebook розробникам для створення застосунків, які взаємодіють із платформою соціальної мережі. Він надає розробникам доступ до безлічі даних користувачів, включно з інформацією про їхній профіль, соціальні зв'язки та інші деталі.

Однією з головних переваг Facebook API є простота використання. Він надає розробникам простий та інтуїтивно зрозумілий спосіб інтеграції своїх застосунків із Facebook, даючи їм змогу швидко та легко отримати доступ до даних користувачів.

Крім того, API надає широкий спектр функціональних можливостей, включно з автентифікацією та авторизацією, отриманням призначених для користувача даних і багато іншого.

Однак API Facebook має і деякі недоліки. Одним з основних є конфіденційність користувацьких даних. Хоча API надає розробникам доступ до великої кількості користувацьких даних, він також викликає побоювання з приводу конфіденційності та безпеки даних. Розробники повинні ретельно дотримуватися політики конфіденційності Facebook і забезпечувати належне поводження з даними користувачів.

OAuth2 і Facebook API є найбільш підходящими рішеннями для нашої системи планування співбесід. OAuth2 забезпечує стандартизований підхід до аутентифікації та авторизації, а Facebook API надає розробникам легкий доступ до даних користувачів. Використовуючи ці технології, ми можемо забезпечити безпечний і зручний досвід для наших користувачів, гарантуючи при цьому захист їхньої конфіденційності та легкий вхід користувача у нашу систему. [7, 8]

1.3.3.5. Огляд інструменту тестування Postman

Postman - це популярний і потужний інструмент розробки API, який використовується для тестування та налагодження API. Це комплексна платформа, яка спрощує процес проектування, створення та документування API. Postman надає зручний інтерфейс для надсилання запитів, перевірки відповідей і вивчення API.

Переваги використання Postman полягають у простоті використання, підтримці безлічі протоколів і форматів, можливості створювати і зберігати складні запити, а також інтеграції з популярними інструментами розробки.

Однак одним із потенційних недоліків використання Postman є те, що він може бути ресурсномістким і сповільнювати роботу систем із низьким рівнем пам'яті або обчислювальної потужності. Крім того, його великі функції і можливості можуть виявитися непосильними для недосвідчених користувачів.

Незважаючи на ці недоліки, Postman є найбільш підходящим рішенням для нашого завдання - розроблення клієнт-серверного застосунку для планування співбесід в IT-компаніях. Підтримка Postman безлічі протоколів і форматів робить його універсальним інструментом, який легко інтегрується в наш мікросервісний застосунок. Його здатність автоматизувати повторювані завдання і створювати складні запити буде особливо корисною в цьому контексті. Загалом, Postman - це потужний інструмент, який може спростити розробку і тестування нашого застосунку, що робить його важливим компонентом нашого проекту. [9]

1.3.3.6. Огляд технології Docker

Docker - це популярна технологія контейнеризації, яка широко використовується в сучасній розробці програмного забезпечення. Docker дає змогу розробникам упаковувати свої додатки в контейнери, які є легкими, переносними та легко розгортаються в різних середовищах. Docker забезпечує низку переваг, включно з ізоляцією, масштабованістю та узгодженістю. За допомогою Docker розробники можуть ізолювати свої додатки від базової операційної системи, забезпечуючи їхню стабільну роботу незалежно від середовища, в якому вони розгорнуті. Крім того, контейнери Docker можна легко збільшувати або зменшувати для задоволення мінливих потреб.

Однак у Docker є і деякі недоліки. Наприклад, управління великою кількістю контейнерів може бути складним, особливо коли йдеться про моніторинг, реєстрацію та безпеку. Крім того, контейнерні додатки можуть вимагати більше ресурсів, ніж традиційні додатки, що може призвести до збільшення витрат.

Незважаючи на ці складнощі, Docker - чудовий вибір для нашої системи планування співбесід в IT-компаніях. Docker дає нам змогу упаковувати наші серверні мікросервіси в легковагі контейнери, які можна легко розгортати й керувати ними в різних середовищах. Це гарантує, що наш застосунок працюватиме стабільно та надійно, незалежно від базової інфраструктури. Крім того, Docker

забезпечує нам високий рівень ізоляції та безпеки, що дуже важливо для роботи з конфіденційними даними, такими як розклад співбесід. [10]

1.4. Висновок до розділу 1

Отже, розробка системи планування співбесід в IT-компаніях являє собою значну перевагу в автоматизації процесу найму. Використання ручних методів, як телефонні дзвінки або електронна пошта, може бути виснажливим і забирати багато часу як у рекрутерів, так і в кандидатів, що призводить до неефективності та затримок у процесі найму. Пропонована система покликана автоматизувати цей процес, покращуючи досвід найму як для кандидатів, так і для рекрутерів.

Для того щоб оцінити ефективність пропонованої системи, було проведено порівняльну оцінку з відомими системами. Результати оцінки засвідчили, що пропонована система має низку переваг перед наявними системами, включно з простотою використання, масштабованістю та гнучкістю. Система також забезпечує підвищену безпеку і конфіденційність інформації про кандидатів.

Для розробки пропонованої системи було обрано низку технологій та інструментів. Архітектура системи планування співбесід складається з клієнт-серверного застосунку, де клієнт - це веб-інтерфейс, що дає змогу кандидатам та інтерв'юерам керувати своїм розкладом, а сервер - це мікросервісний застосунок, що керує плануванням і бронюванням співбесід.

На стороні клієнта веб-сторінки було визначено технології та інструменти HTML, CSS і JavaScript, а для виконання API-запитів до сервера визначена бібліотека Axios. На стороні сервера визначено, що додаток буде розроблено з використанням Java, а Spring Framework буде використаний для впровадження залежностей, забезпечення безпеки та веб-розробки. Hibernate буде використовувався як ORM для бази даних PostgreSQL, у якій зберігається інформація про кандидатів і співбесіди. OAuth2 буде використовуватися для аутентифікації та авторизації, а Facebook API - для інтеграції соціального входу.

Нарешті, Postman буде використовувався як інструмент тестування для забезпечення правильного функціонування кінцевих точок API.

Технологію Docker будем використовувати для контейнеризації мікросервісів і забезпечення їхньої переносимості та узгодженості в різних середовищах. Це також дає змогу спростити масштабування і розгортання застосунку.

Отже, слід зазначити, що запропонована система являє собою значну перевагу в автоматизації процесу підбору персоналу, забезпечуючи простоту використання, масштабованість, гнучкість, безпеку і конфіденційність інформації про кандидатів. Використання низки технологій та інструментів, включно з Docker, дає змогу ефективно розробити та розгорнути додаток.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ ДЛЯ ПЛАНУВАННЯ СПІВБЕСІД

2.1. Комунікація між клієнтом та сервером системи

У рамках розробки системи для планування співбесід в IT-компаніях, комунікація між клієнтом і сервером буде здійснюватися за допомогою HTTP запитів. HTTP (Hypertext Transfer Protocol) - це протокол передавання даних, що використовується для комунікації між клієнтом і сервером у мережі інтернет. Він забезпечує стандартизований спосіб передачі запитів і відповідей між різними вузлами в мережі.

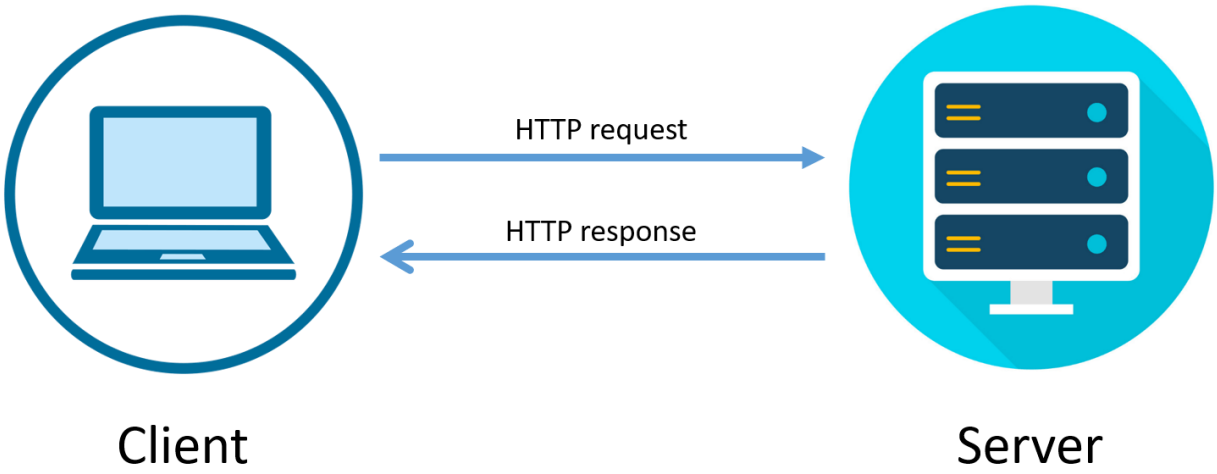


Рис. 2.1. Принцип роботи HTTP-протоколу повертає відповідні відповіді.

У клієнт-серверній архітектурі системи, клієнтом є веб-сайт, розроблений з використанням HTML, CSS і JavaScript. Клієнтська частина взаємодіє з користувачем, надає інтерфейс для введення даних і відображає інформацію, отриману від сервера. Серверна частина представлена мікросервісним додатком, написаним на Java з використанням Spring Boot, Spring Data, Spring Security, Spring Web, Hibernate, PostgreSQL, OAuth2 і Docker. Сервер обробляє запити від клієнта, виконує необхідну логіку і повертає відповідні відповіді.

Під час обміну даними між клієнтом і сервером, клієнт надсилає HTTP-запити на сервер, а сервер відповідає на ці запити за допомогою HTTP- відповідей (рис. 2.1.). У системі для планування співбесід, запити можуть включати операції, такі як створення і редагування тимчасових слотів кандидата і інтерв'юера, бронювання інтерв'ю, управління ролями та інші функціональні можливості системи.

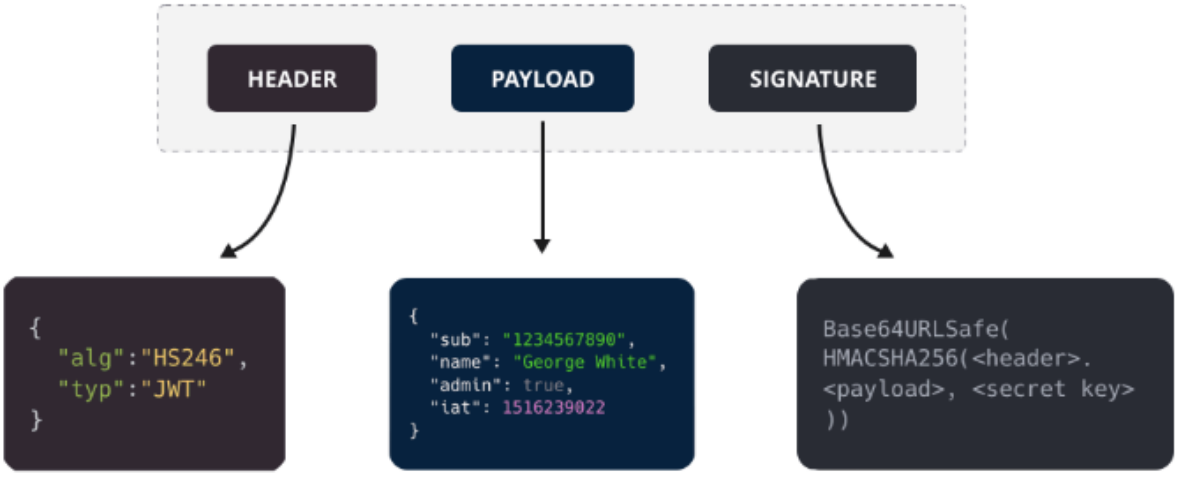


Рис. 2.2. Структура JWT

Для забезпечення безпеки та автентифікації в системі, в кожному запиті присутня авторизація через JWT (JSON Web Token). JWT - це відкритий стандарт (RFC 7519) для створення токенів, які можуть бути перевірені та розшифровані. Токени JWT складаються з трьох частин: заголовка (header), корисного навантаження (payload) і підпису (signature) (рис. 2.2.). Заголовок містить інформацію про тип токена і використовуваний алгоритм шифрування. Корисне

навантаження містить інформацію про користувача та інші дані, необхідні для аутентифікації та авторизації. Підпис використовується для перевірки цілісності токена.

Під час автентифікації в системі, клієнт надає облікові дані, такі як ім'я користувача та пароль, через з'єднання HTTP. Сервер перевіряє дані і в разі успішної автентифікації, генерує JWT токен, який містить інформацію про користувача та інші дозволи. Клієнт зберігає отриманий токен і включає його в заголовок кожного наступного запиту до сервера. Сервер, зі свого боку, перевіряє справжність токена, використовуючи секретний ключ, і виконує запитані операції тільки для автентифікованих і авторизованих користувачів.

У цій системі HTTP запити і відповіді використовуються для передачі даних між клієнтом і сервером. Запити можуть бути відправлені з використанням різних методів HTTP, таких як GET, POST, PUT і DELETE, залежно від типу операції, яку потрібно виконати. Наприклад, при створенні нового тимчасового слота кандидата, клієнт надсилає POST запит на сервер, що містить відповідні дані. Сервер обробляє цей запит, зберігає інформацію про тимчасовий слот у базі даних і повертає відповідну HTTP-відповідь, що вказує на успішне створення слота.

Таким чином, використання HTTP запитів і JWT авторизації забезпечує ефективну комунікацію між клієнтом і сервером у системі для планування співбесід в IT-компаніях. Це дає змогу користувачам взаємодіяти з системою, виконувати різні операції та отримувати відповідні відповіді від сервера, забезпечуючи безпеку та захист даних.

2.2. Визначення бізнес-правил та логіки системи

Перед тим, як почати конструювати систему ми повинні описати вимоги і правила, яких мають дотримуватися кандидати, інтерв'юери та координатори в системі для планування співбесід в IT-компаніях. Розглянемо кожну роль окремо й обґрунтуємо вибір цих вимог.

Кандидат:

Кандидати в системі мають можливість створювати й оновлювати свої часові слоти для проведення співбесід. Наступні вимоги для кандидатів обґрунтовані:

1. Слоти мають бути в майбутньому: Таке обмеження гарантує, що кандидати створюють слоти тільки на майбутні дати, запобігаючи можливості створення слотів у минулому, адже це не має сенсу.

2. Слот має бути 1,5 години або більше з округленням до 30 хвилин: Ця вимога дає змогу створювати достатньо тривалі слоти, щоб забезпечити достатній час для проведення якісної співбесіди. Округлення до 30 хвилин дає змогу зручно планувати розклад та запобігати криві часові слоти.

3. Слот визначається як точна розбіжність дати та часу: Таке визначення слотів дозволяє кандидатам чітко вказувати доступні моменти часу для проведення співбесіди.

Інтерв'юер:

Інтерв'юери також можуть створювати й оновлювати свої слоти для співбесід. Нижче перераховані вимоги, яких повинні дотримуватися інтерв'юери:

1. Створення до кінця п'ятниці (00:00) поточного тижня: Це обмеження гарантує, що інтерв'юери можуть планувати свій розклад тільки на поточний тиждень, забезпечуючи своєчасне планування та уникаючи подальших змін на більш тривалий термін.

2. Слот означає день тижня + часовий діапазон: Такий підхід дає змогу інтерв'юерам вказувати доступні дні тижня та часові інтервали для проведення співбесід, забезпечуючи гнучкість у плануванні.

3. Межі слота округлюються до 30 хвилин: Округлення меж слотів до 30 хвилин полегшує розуміння і використання часових інтервалів у системі.

4. Тривалість слота має бути більшою або дорівнювати 1,5 години: Вимога забезпечує достатній час для проведення співбесіди і дає змогу інтерв'юерам планувати розклад з урахуванням цього часового обмеження.

5. Час початку слота не може бути меншим за 8:00, а час закінчення слота не може бути більшим за 22:00: Ці обмеження визначають робочі години інтерв'юєра і

допомагають установити розумні часові межі для слотів з урахуванням робочого розкладу.

Координатор:

Координатори в системі мають можливість переглядати всі слоти кандидатів та інтерв'юерів, а також створювати та оновлювати бронювання. Для координаторів встановлено такі вимоги:

1. Час бронювання має бути більшим ніж 1,5 години: Вимога забезпечує достатній час для проведення співбесіди та гарантує, що бронювання не може бути занадто коротким.

2. Під час бронювання має зазначатися тема та опис інтерв'ю: Ці додаткові поля дають змогу координаторам вказувати інформацію про мету та характер співбесіди, що полегшує розуміння та планування для всіх учасників.

3. Координатор може призначати або відкликати ролі інтерв'юера або координатора. Це додаткова функціональність, що дає змогу координаторам керувати ролями користувачів. Координатор не може відкликати роль сам у себе. Це дає гарантію, що система ніколи не залишиться без єдиного координатора.

Вибір зазначених вимог і правил ґрунтується на кількох факторах:

1. Зручність використання: Запропоновані вимоги дають змогу зручно використовувати систему для всіх учасників. Вони оптимізовані для планування співбесід в IT-компаніях, враховуючи специфічні вимоги кандидатів та інтерв'юерів.

2. Гнучкість та адаптивність: Вимоги дають змогу кандидатам та інтерв'юерам створювати та оновлювати слоти з урахуванням різноманітних часових обмежень і надають можливість планувати на тиждень наперед. Координатори мають повний контроль над бронюванням та управлінням ролями.

3. Захист від помилок і конфліктів: Обмеження на часові інтервали, округлення меж слотів і перевірка наявності бронювання запобігають можливості помилок і конфліктів під час планування співбесід.

4. Зрозумілість і наочність: Зазначені вимоги чітко та наочно описують умови й обмеження для кандидатів, інтерв'юерів і координаторів, полегшуючи розуміння і використання системи.

У результаті вибору та обґрунтування цих вимог і правил ми створюємо систему для планування співбесід в IT-компаніях, яка забезпечує ефективність, зручність і гнучкість у процесі призначення та проведення співбесід.

2.3. Визначення списку необхідних HTTP-запитів

Для подальшого проектування насамперед слід визначити список необхідних запитів спілкування між клієнтом та сервером, адже коли ми будемо чітко знати якими саме запитами буде спілкуватися клієнт з сервером, нам буде набагато легше спроектувати сутності та побудувати архітектуру клієнта та серверного додатка відповідно.

Слід зазначити, що при розробці списку необхідних HTTP-запитів також буде розглядатися формат відповіді цих запитів. Відповіді можуть бути як успішні, так і не успішні, адже при надсиланні запиту можуть бути помилки. Наприклад, клієнт може надіслати запит із даними, які не відповідають бізнес-логіки нашої системи. У такому випадку відповідь сервера повинна вказати на цю помилку, відповідно із певним статус-кодом цієї відповіді. Тому надалі, при наведенні списку запитів буде розглядатися коректна відповідь сервера та тематичну групу помилок, до якої буде відноситися можлива відповідь сервера. Та наприкінці цього розділу буде наведена таблиця усіх тематичних відповідей не успішних запитів, у якій детально розглянуться усі види відповідей не успішних запитів.

2.3.1. Запити для авторизації та аутентифікації

Тож, почати треба з запитів авторизації та аутентифікації, тому що з цього і починається робота клієнта з сервером.

Так як авторизація та аутентифікація у нас відбуваються за допомогою Facebook API, то для початку нам необхідно створити допоміжний запит для отримання Facebook Token. Цей запит не буде використовуватися клієнтом, адже

клієнт сам буде робити цей запит до Facebook, але цей запит нам буде необхідних при розробці серверної частини для тестування системи без наявного клієнта.

Запит: «*GET /oauth2/facebook/v15.0*».

Відповідь:

{

"clientId": "CLIENT_ID",

"redirectUri": "REDIRECT_URI",

"tokenRequestUrl":

"https://www.facebook.com/v15.0/dialog/oauth?client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=token"

}

За посиланням, представленим у tokenRequestUrl, ми можемо пройти автентифікацію Facebook, і у нас буде перенаправлено на redirectUri з нашим токеном facebook в якості параметра URL-адреси access_token.

Потім із наявним Facebook Token ми повинні наступним запитом отримати JWT (токен нашої системи).

Запит: «*POST /authenticate*».

Параметри даних:

{"facebookToken": "EAAHC..."}

Відповідь:

{

"token": "eyJhb..."

}

Можливі винятки:

- *401 - bad_facebook_token_exception;*
- *401 - bad_credentials.*

Отриманий JWT слід помістити в заголовок в усі посліуючи запити як параметр Authorization зі значенням Bearer.

При використанні JWT в усіх запитах ми можемо зіткнутися з наступними винятковими ситуаціями:

- *401 - not_authenticated_exception;*
- *401 - bad_token_exception;*
- *401 - expired_token_exception;*
- *401 - bad_token_signature_exception;*
- *401 - malformed_token_exception;*
- *401 - unsupported_token_exception;*
- *403 - access_denied_exception.*

2.3.2. Запити, які доступні для будь-якого користувача

Як було визначено раніше, наша система буде використовувати таке поняття як номер тижня. Так як отримання номеру тижня не є конфіденційною інформацією, до цих запитів буде мати доступ будь-який користувач.

Отримати поточний номер тижня:

Запит: «*GET /weeks/current*».

Відповідь:

```
{
  "weekNum": 44
}
```

Отримати наступний номер тижня:

Запит: «*GET /weeks/next*»

Відповідь:

```
{
  "weekNum": 45
}
```

Також кожен користувач має право на отримання власної інформації про себе:

Запит: «*GET /me*».

Відповідь:

```
{
  "email": "example@google.com",
```

"role": "INTERVIEWER",

"id": 123

}

Поле «id» буде присутнє лише для користувачів з ролями INTERVIEWER або COORDINATOR, адже кандидати не будуть зберігаються у нашій системі.

2.3.3. Запити для кандидата

Кандидат може створювати або оновлювати власні слоти, тому нам необхідні запити, які дадуть змогу це зробити:

Створення слота:

Запит: «*POST /candidates/current/slots*».

Параметри даних:

- *time* - дата створення слоту для кандидата;
- *from* - час початку слоту у форматі HH:mm;
- *to* - час закінчення слоту у форматі HH:mm.

Відповідь:

{

"date": "2022-12-06",

"from": "09:00",

"to": "17:00"

}

Можливі групи винятків:

- *Slots Interaction;*
- *Updating Slot.*

Редагування слота:

Запит: «*POST /candidates/current/slots/{slotId}*».

Параметри URL-адреси:

- *slotId* - ідентифікатор слота для редагування.

Параметри даних:

- *date* - дата створення слоту-кандидата;
- *from* - час початку слоту у форматі HH:mm;
- *to* - час закінчення слоту у форматі HH:mm.

Відповідь:


```
{
   "date": "2022-12-06",
   "from": "09:00",
   "to": "17:00"
}
```


Можливі групи винятків:

- *Slots Interaction;*
- *Getting Slots.*

Отримання списку усіх слотів певного кандидату:

Запит: «*GET /candidates/current/slots*».

Відповідь:


```
{
 "candidateSlotDtoList": [
   {
     "date": "22.01.2022",
     "from": "9:00",
     "to": "17:00""
   },
   {
     "date": "23.01.2022",
     "from": "13:00",
     "to": "20:00""
   }
  ]
}
```

2.3.4. Запити для інтерв’юера

Інтерв'юер може створювати або оновлювати власні слоти, тому нам необхідні запити, які дадуть змогу це зробити, крім того, інтерв'юери можуть встановити ліміт бронювання на наступний тиждень (максимальна кількість бронювань, яку координатори зможуть створити для певного інтерв'юера і на певному тижні):

Створення слота:

Запит: «*POST /interviewers/{interviewerId}/slots*».

Параметри URL-адреси:

- *interviewerId* - ідентифікатор поточного інтерв'юера (може бути отриманий з запиту /me).

Параметри даних:

- *week* - номер тижня;
- *dayOfWeek* - день тижня (MON, TUE, WED, THU, FRI);
- *from* - час початку слоту у форматі HH:mm;
- *to* - час закінчення слоту у форматі HH: mm.

Відповідь:

```
{
   "interviewerId": 1,
   "interviewerSlotId": 1,
   "week": 50,
   "dayOfWeek": "TUE",
   "from": "16:00",
   "to": "19:00"
}
```

Можливі групи винятків:

- *Slots Interaction;*
- *User Interaction.*

Оновлення слота:

Запит: «*POST /interviewers/{interviewerId}/slots/{slotId}*».

Параметри URL-адреси:

- *interviewerId* - ідентифікатор поточного інтерв'юера;
- *slotId* - ідентифікатор слоту для редагування.

Параметри даних:

- *week* - номер тижня;
- *dayOfWeek* - день тижня (MON, TUE, WED, THU, FRI);
- *from* - час початку слоту у форматі HH:mm;
- *to* - час закінчення слоту у форматі HH:mm.

Відповідь:


```
{
   "interviewerId": 1,
   "interviewerSlotId": 1,
   "week": 50,
   "dayOfWeek": "TUE",
   "from": "16:00",
   "to": "19:00"
}
```


Можливі групи винятків:

- *Slots Interaction;*
- *User Interaction.*

Отримання всіх слотів певного інтерв’юера:

Запити:

- «*GET /interviewers/{interviewerId}/slots/current*» (для отримання слотів поточного тижня);
- «*GET /interviewers/{interviewerId}/slots/next*» (для отримання слотів наступного тижня).

Параметри URL:

- *interviewerId* - ідентифікатор поточного інтерв'юера.

Відповідь:

```
{
  "interviewerSlotDtoList": [
    {
      "interviewerId": 1,
      "interviewerSlotId": 1,
      "week": 50,
      "dayOfWeek": "TUE",
      "from": "16:00",
      "to": "16:00"
    },
    {
      "interviewerId": 1,
      "interviewerSlotId": 2,
      "week": 50,
      "dayOfWeek": "WED",
      "from": "16:00",
      "to": "16:00"
    }
  ]
}
```


Встановлення ліміту бронювання:

Запит: «*POST /interviewers/{interviewerId}/bookingLimit*».

Параметри URL:

- *interviewerId* - ідентифікатор поточного інтерв'юера (може бути отриманий з кінцевої точки /me).

Параметри даних:

- *bookingLimit* - ліміт бронювань на наступний тиждень.

Відповідь:


```
{
  "userId": 1,
```

"weekNum": 50,

"bookingLimit": 127

}

Можливі групи винятків:

- *User Interaction;*
- *Booking Limit.*

Отримання ліміту бронювання певного інтерв’юера:

Запити:

- «*GET /interviewers/1/booking-limits/current-week*» (для отримання поточного ліміту на поточний тиждень);
- «*GET /interviewers/1/booking-limits/next-week*» (для отримання поточного ліміту на наступний тиждень).

Параметри URL:

- *interviewerId* - ідентифікатор поточного інтерв'юера.

Відповідь:

{

"userId": 1,

"weekNum": 49,

"bookingLimit": 1000

}

2.3.5. Запити для координатора

Координатор може бачити всі слоти кандидатів та інтерв'юерів, оновлювати часовий слот будь-якого інтерв'юера, створювати або оновлювати бронювання, надаючи, крім того, координатор може надавати або відкликати ролі інтерв'юера або координатора за вказаною електронною поштою, яка зареєстрована у користувача у обліковому запису у Facebook. Перший email координатора в системі визначається змінною оточення.

Отримання дашборду (інформація щодо всіх слотів та бронювань усіх користувачів):

Запит: «*GET /weeks/{weekNum}/dashboard*».

Параметри URL-адреси:

- *weekNum* - номер тижня для формування дашборду.

Відповідь:


```
{
  "weekNum": 50,
  "dashboard": {
    "2022-12-15": {
      "interviewerSlots": [],
      "candidateSlots": [],
      "bookings": {}
    },
    "2022-12-14": {
      "interviewerSlots": [
        {
          "interviewerSlotId": 2,
          "from": "16:00",
          "to": "19:00",
          "interviewerId": 2,
          "bookings": []
        }
      ],
      "candidateSlots": [],
      "bookings": {}
    },
    "2022-12-13": {
      "interviewerSlots": [],
      "candidateSlots": [
```

```
      {
        "candidateSlotId": 1,
        "from": "10:00",
        "to": "14:00",
        "candidateEmail": "bielobrov.8864899@stud.op.edu.ua",
        "candidateName": "Артур Бєлобров",
        "booking": []
      }
    ],
    "booking": {}
  },
  "2022-12-12": {
    "interviewerSlots": [
      {
        "interviewerSlotId": 1,
        "from": "15:00",
        "to": "19:00",
        "interviewerId": 2,
        "bookings": [
          1
        ]
      }
    ],
    "candidateSlots": [
      {
        "candidateSlotId": 2,
        "from": "09:00",
        "to": "17:00",
        "candidateEmail": "bielobrov.8864899@stud.op.edu.ua",
        "candidateName": "Артур Бєлобров",
```

```
            "booking": [
               1
            ]
         }
      ],
      "booking": {
         "1": {
            "bookingId": 1,
            "subject": "Співбесіда в понеділок",
            "description": "Співбесіда на позицію Java Developer",
            "interviewerSlotId": 1,
            "candidateSlotId": 2,
            "from": "15:00",
            "to": "16:30"
         }
      }
   },
   "2022-12-18": {
      "interviewerSlots": [],
      "candidateSlots": [],
      "bookings": {}
   },
   "2022-12-17": {
      "interviewerSlots": [],
      "candidateSlots": [],
      "bookings": {}
   },
   "2022-12-16": {
      "interviewerSlots": [],
      "candidateSlots": [],
```

```
        "bookings": {}
      }
    }
}
```


Створення бронювання:

Запит: «*POST /bookings*».

Параметри даних:

- *interviewerSlotId* - ідентифікатор слоту інтерв'юера з вільним часом на потрібний період;
- *candidateSlotId* - ідентифікатор слоту кандидата з вільним часом на потрібний період часу;
- *from, to* - межі бронювання (часовий проміжок) у форматі HH:mm (1:30 год.);
- *subject* - тема бронювання (0 - 255 символів);
- *description* - опис бронювання (до 4000 символів).

Відповідь:


```
{
   "interviewerSlotId": 1,
   "candidateSlotId": 2,
   "from": "15:00",
   "to": "16:30",
   " subject ": "Понеділкова співбесіда",
   "description": "Співбесіда на позицію Java Developer"
}
```


Можливі групи винятків:

- *Slots Interaction;*
- *Booking;*
- *Booking Limit;*
- *User Interaction.*

Оновлення бронювання:

Запит: «*POST /bookings/{booking-id}*».

Параметри URL-адреси:

- *booking-id* - ідентифікатор бронювання, яке потрібно оновити.

Параметри даних:

- *interviewerSlotId* - ідентифікатор слоту інтерв'юера з вільним часом на потрібний період;
- *candidateSlotId* - ідентифікатор слоту кандидата з вільним часом на потрібний період;
- *from, to* - межі бронювання (часовий проміжок) у форматі HH:mm (1:30 год.);
- *subject* - тема бронювання (0 - 255 символів);
- *description* - опис бронювання (до 4000 символів).

Відповідь:

```
{
   "interviewerSlotId": 1,
   "candidateSlotId": 2,
   "from": "15:00",
   "to": "16:30",
   "subject": "Понеділкова співбесіда",
   "description": "Співбесіда на позицію Java Developer"
}
```

Можливі групи винятків:

- *Slots Interaction;*
- *Booking;*
- *Booking Limit;*
- *User Interaction;*
- *Deleting Booking.*

Видалення бронювання:

Запит: «*DELETE /bookings/{booking-id}*».

Параметри URL-адреси:

- *booking-id* - ідентифікатор бронювання для видалення.

Відповідь:


```
{
   "interviewerSlotId": 1,
   "candidateSlotId": 2,
   "from": "15:00",
   "to": "16:30",
   "subject": "Понеділкова співбесіда",
   "description": "Співбесіда на позицію Java Developer"
}
```


Можливі групи винятків:

- *Booking.*

Отримання списку користувачів за ролями:

Запити:

- «*GET /users/interviewers*» - отримати список всіх інтерв'юерів в системі;
- «*GET /users/coordinators*» - отримати список всіх координаторів в системі.

Відповідь:


```
{
   "users": [
      {
         "email": "azofer77@gmail.com",
         "role": "INTERVIEWER",
         "id": 2
      }
   ]
}
```


Надання ролі за електронною поштою:

Запити:

- «*POST /users/interviewers*» - надання ролі інтерв'юера за електронною поштою;
- «*POST /users/coordinators*» - надання ролі координатора за електронною поштою.

Параметри даних:

- *email* - прикріплений до email користувача на Facebook для надання відповідної ролі

Відповідь:

```
{
   "email": "example@gmail.com",
   "role": "INTERVIEWER",
   "id": 3
}
```

Можливі групи винятків:

- *User Interaction.*

Відкликання ролі за електронною поштою:

Запити:

- «*DELETE /users/interviewers/{id}*» - відкликати роль інтерв'юера за ідентифікатором;
- «*DELETE /users/coordinators/{id}*» - відкликати роль координатора за ідентифікатором.

Параметри URL-адреси:

- *id* - id користувача, якого потрібно видалити.

Відповідь:

```
{
   "email": "example@gmail.com",
   "role": "INTERVIEWER",
   "id": 3
}
```

Можливі групи винятків:

- *User Interaction.*

2.3.6. Створення винятків для HTTP-запитів системи

Тож, коли ми визначили список необхідних HTTP-запитів, вказавши url-адресу кожного запиту, їх url-параметри та параметри даних, також ми визначили групи винятків до кожного запиту, то настав час визначити більш детальний огляд кожної групи винятків, їх структуру відповіді, статус-код та повідомлення про причину винятку.

Всі виключення у нашій системі будуть мати наступну структуру відповіді:

```
{
   "errorCode": "snake_case_meaningful_string",
   "errorMessage": "English user friendly message"
}
```

Усі групи винятків та їх детальні відповіді можна побачити на табл. 2.1.

Табл. 2.1.

Групи винятків

Http Response Status	Group (use case)	Error Code	Error Message
400 (Business validation)	Booking	invalid_subject	Provided subject is invalid.
		invalid_description	Provided description is invalid.
	Booking Limit	invalid_booking_limit	Value of booking limit is not correct.
		booking_limit_is_exceeded	Interviewer isn't allowed to have more bookings.
	Slots Interaction	slots_not_intersecting	Provided slots have not free joint time period.
		cannot_edit_this_week	This week can't be edited.
		invalid_boundaries	Time boundaries of slot or booking are invalid.
		invalid_day_of_week	Cannot arrange booking on this day.
		slot_is_booked	Slot you are trying to occur is booked.
		slot_is_overlapping	Slot overlaps already existed one.
		slot_is_in_the_past	New date for this slot is in the past.

Продовження табл. 2.1.

	User Interaction	self_revoking	The user cannot change or delete himself.
		user_already_has_role	This user already has another role.
		not_interviewer	Provided user is not interviewer.
		not_coordinator	Provided user is not coordinator.
401 (Authentication)	Auth	not_authenticated	You are not authenticated to perform this action.
		bad_facebook_token	Invalid OAuth access token - cannot parse access token.
		bad_token	Token does not start with 'Bearer'.
		expired_token	Token has expired.
		bad_token_signature	Given JWT signature does not match locally computed signature.
		malformed_token	Unable to read JWT JSON value.
		unsupported_token	JWT in a particular format/configuration that does not match the format expected by the application.
		bad_credentials	Incorrect credentials.

Продовження табл. 2.1.

403 (Access denied)	Auth	not_authenticated	You are not authenticated to perform this action.
404 (Wrong identifier)	Booking	booking_not_found	Booking by given id was not found.
	Slots Interaction	candidate_slot_not_found	Candidate slot by given id was not found.
		interviewer_slot_not_found	Interviewer slot by given id was not found.
		user_not_found	User not found.
	User Interaction	interviewer_not_found	Invalid interviewer's id in the path.

2.4. Розробка серверної частини системи

2.4.1. Загальна структура серверного додатку

Структура серверного додатка, розроблена на основі фреймворку Spring і його компонентів. Система для планування співбесід в IT-компаніях є мікросервісним застосунком, тому ми використовуватимемо Spring Boot для створення автономного серверного застосунку. Проектування за допомогою Spring дозволить нам легко реалізувати наступні компоненти додатка та взаємодії між ними:

1. Контролери (Controllers). Контролери являють собою основну точку входу для запитів від клієнтської частини системи. У нашому випадку, ми будемо створювати контролери з використанням Spring Web. Вони обробляють вхідні HTTP-запити, взаємодіють з іншими компонентами додатка і повертають результати назад клієнту.

2. Сервіси (Services). Сервіси відповідають за обробку бізнес-логіки та виконання операцій, пов'язаних із базою даних або іншими зовнішніми сервісами. У

нашому додатку ми будемо використовувати Spring Data для взаємодії з базою даних PostgreSQL.

3. Репозиторії (Repositories). Репозиторії надають абстракцію для роботи з базою даних. У нашому випадку, ми будемо використовувати Spring Data JPA у зв'язці з Hibernate для роботи з базою даних PostgreSQL. Репозиторії дають змогу виконувати операції читання і запису даних у базу даних, а також виконувати запити з використанням мови запитів, підтримуваної JPA.

4. Моделі даних (Data Models). Моделі даних являють собою об'єкти, які відображають сутності системи та їхні зв'язки. У нашій системі для планування співбесід в IT-компаніях моделі даних можуть містити в собі сутності, такі як слоти кандидата чи інтерв'юера, корситувач та інші, а також їхні атрибути та зв'язки між ними.

5. Конфігурація (Configuration). Spring надає можливість налаштування і конфігурації програми з використанням класів конфігурації. Ми можемо використовувати анотації для налаштування різних компонентів Spring, таких як безпека (Spring Security), авторизація (OAuth2), підключення до бази даних та інші.

6. Безпека (Security). Для забезпечення безпеки в нашій системі ми використовуватимемо Spring Security, який надає механізми аутентифікації та авторизації.

Загальна структура серверного додатка містить ці компоненти, які працюють разом для забезпечення функціональності системи для планування співбесід в IT-компаніях. Кожен компонент виконує своє завдання відповідно до певних принципів проєктування і співпрацює з іншими компонентами для досягнення спільної мети розробки системи.

2.4.2. Файлова структура серверного додатку

Структура серверного проекту включає такі основні директорії та файли:

- src: Головна директорія проекту, що містить вихідний код серверної частини.
    - main: Основна директорія з вихідним кодом програми.

- java: Директорія, що містить пакети та класи Java.
    - com.intellistart: Головний пакет програми.
        - config: Пакет, що містить конфігураційні класи програми.
        - controller: Пакет, що містить контролери, які обробляють HTTP-запити.
        - dto: Пакет, що містить класи Data Transfer Object (DTO) для передачі даних між клієнтом і сервером.
        - exception: Пакет, що містить класи винятків, які використовуються в додатку.
        - model: Пакет, що містить класи моделей даних додатка.
        - repository: Пакет, що містить інтерфейси репозиторіїв для взаємодії з базою даних.
        - security: Пакет, що містить класи для реалізації аутентифікації та авторизації.
        - service: Пакет, що містить сервісні класи для обробки бізнес-логіки програми.
        - IntellistartApplication.java: Головний клас програми, точка входу.
    - resources: Директорія з ресурсами додатка.
        - application.properties: Файл конфігурації програми, що містить налаштування бази даних, порту сервера та інші параметри.
- test: Директорія з модульними тестами додатка.

- pom.xml: Файл конфігурації Maven, який використовується для керування залежностями проєкту та налаштування збірки.
- Dockerfile: Файл конфігурації Docker, що описує процес складання Docker-образу програми.

- README.md: Файл з описом проєкту, інструкціями щодо встановлення та використання.

Файлова структура серверної частини системи слідує принципам модульності та хорошої організації коду, що полегшує розуміння і підтримку проекту.

2.4.3. Розробка контролерів серверної частини

Процес розробки контролерів серверної частини системи починається з визначення контролерів та їхніх назв. У цьому проекті виділено такі контролери:

1. Клас GuestController. Клас GuestController буде відповідати за реалізацію запитів із списку визначених нами запитів, які доступні для не авторизованого користувача, , а саме запит «GET weeks/current» та запит «GET weeks/next».

2. Клас CandidateController. Клас CandidateController буде відповідати за реалізацію запитів із списку наших запитів, які відносяться до кандидату.

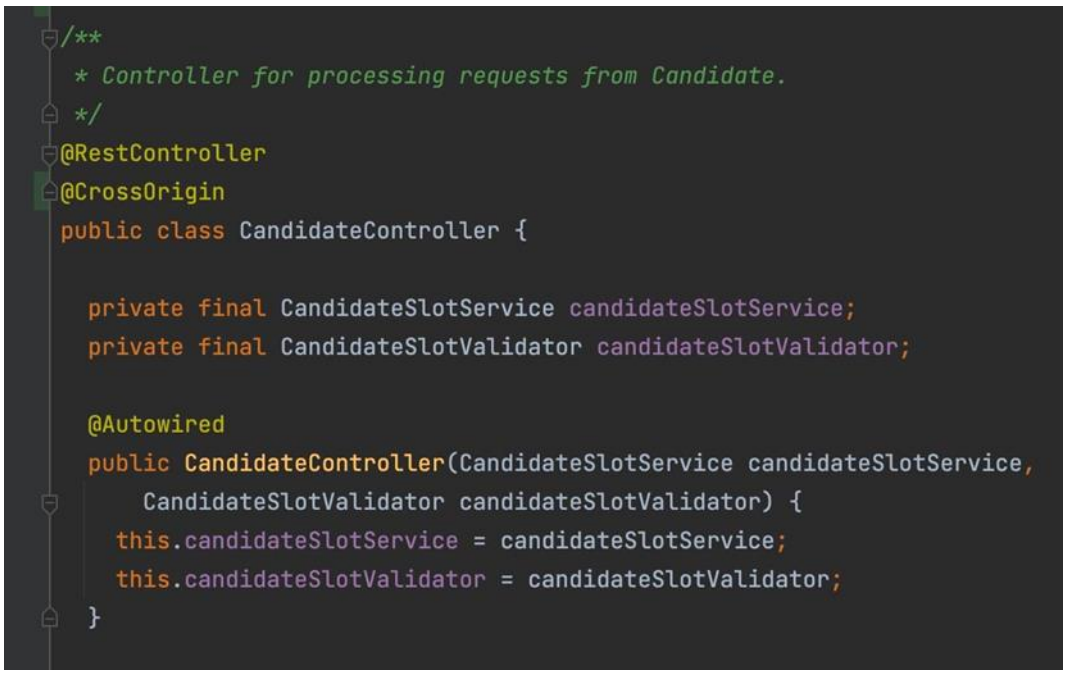
3. Клас InterviewerController. Клас InterviewerController буде відповідати за реалізацію запитів із списку наших запитів, які відносяться до інтерв’юера.

4. Клас CoordinatorController. Клас CoordinatorController буде відповідати за реалізацію запитів із списку наших запитів, які відносяться до координатора.

5. Клас JwtAuthenticationController. Клас JwtAuthenticationController буде відповідати за реалізацію запитів із списку наших запитів, які відносяться до авторизації та аутентифікації.

Тож, розглянемо розробку класа CandidateController.

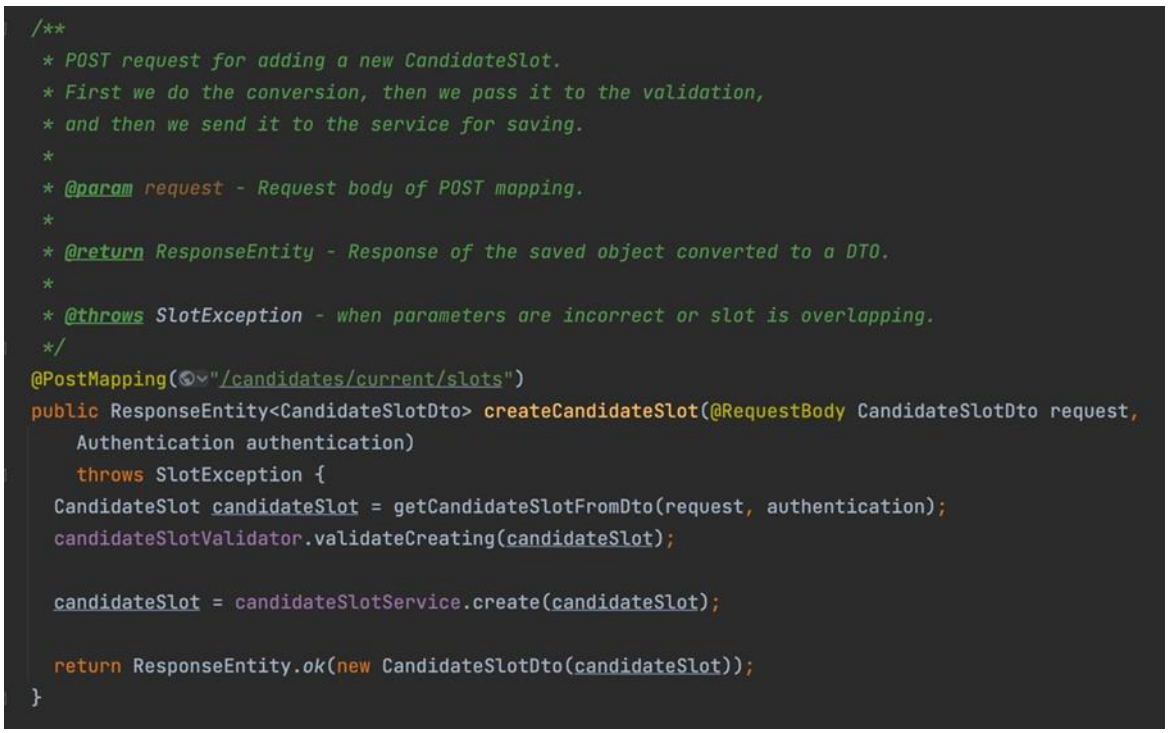
Анотація @RestController вказує, що клас є контролером, який обробляє HTTP-запити. Анотація @CrossOrigin дозволяє крос-доменні запити, що може бути корисно для клієнтської частини програми. Конструктор класу CandidateController ін'єктує сервіси CandidateSlotService і CandidateSlotValidator через механізм впровадження залежностей (@Autowired) (Рис. 2.3.).



```java
/**
 * Controller for processing requests from Candidate.
 */
@RestController
@CrossOrigin
public class CandidateController {

  private final CandidateSlotService candidateSlotService;
  private final CandidateSlotValidator candidateSlotValidator;

  @Autowired
  public CandidateController(CandidateSlotService candidateSlotService,
      CandidateSlotValidator candidateSlotValidator) {
    this.candidateSlotService = candidateSlotService;
    this.candidateSlotValidator = candidateSlotValidator;
  }
```

Рис. 2.3. Контролер кандидата (1)

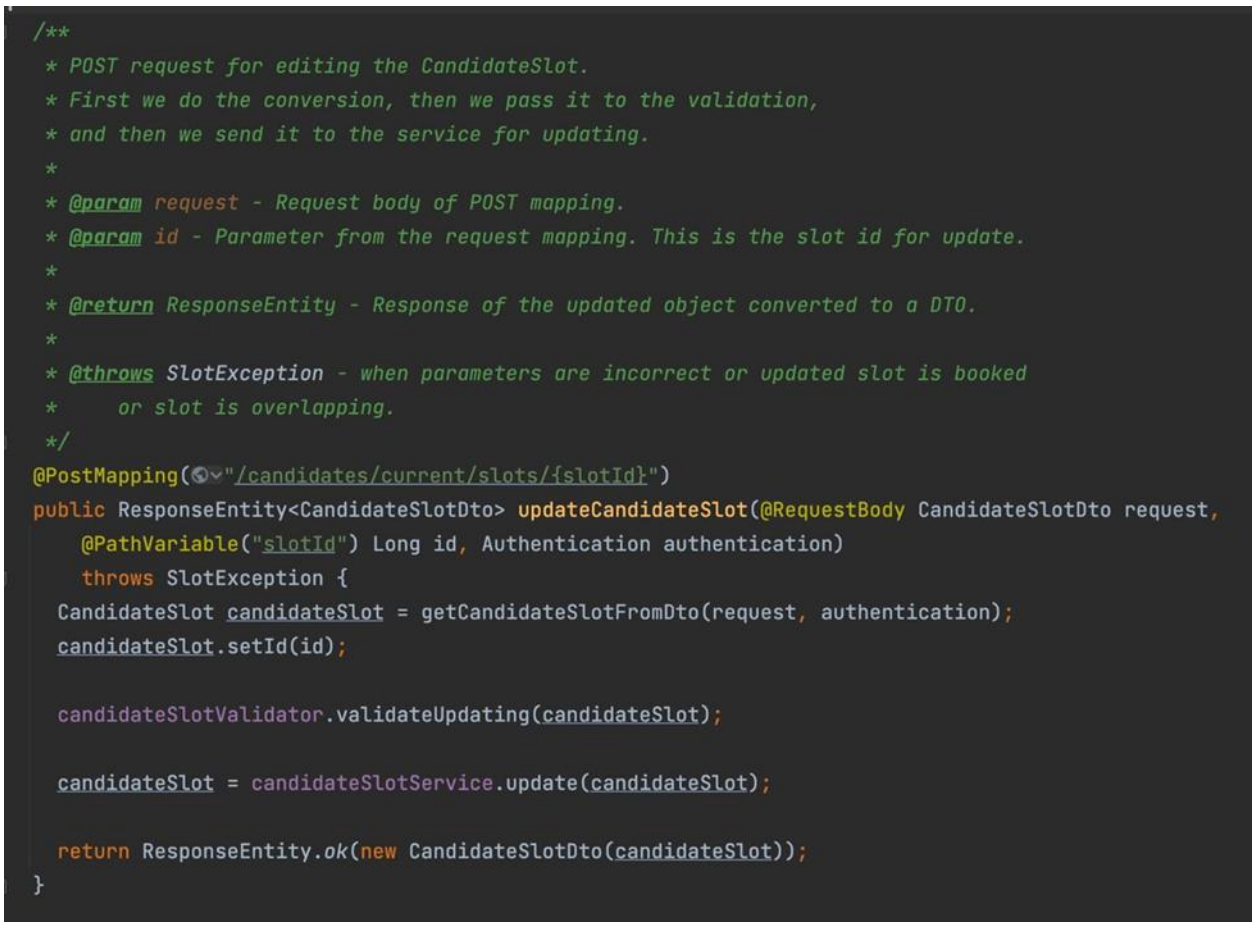
Метод createCandidateSlot обробляє POST-запит для додавання нового CandidateSlot. Анотація @PostMapping вказує на те, що цей метод оброблятиме POST-запити за вказаним шляхом "/candidates/current/slots". Метод отримує CandidateSlotDto з тіла запиту (@RequestBody) і Authentication для аутентифікації користувача. Потім він викликає метод getCandidateSlotFromDto для конвертації CandidateSlotDto в CandidateSlot, виконує валідацію за допомогою candidateSlotValidator і зберігає створений CandidateSlot за допомогою candidateSlotService. Повертається ResponseEntity з HTTP-статусом 200 (OK) і перетвореним об'єктом CandidateSlotDto (Рис. 2.4.).



```
/**
 * POST request for adding a new CandidateSlot.
 * First we do the conversion, then we pass it to the validation,
 * and then we send it to the service for saving.
 *
 * @param request - Request body of POST mapping.
 *
 * @return ResponseEntity - Response of the saved object converted to a DTO.
 *
 * @throws SlotException - when parameters are incorrect or slot is overlapping.
 */
@PostMapping("/candidates/current/slots")
public ResponseEntity<CandidateSlotDto> createCandidateSlot(@RequestBody CandidateSlotDto request,
    Authentication authentication)
    throws SlotException {
  CandidateSlot candidateSlot = getCandidateSlotFromDto(request, authentication);
  candidateSlotValidator.validateCreating(candidateSlot);

  candidateSlot = candidateSlotService.create(candidateSlot);

  return ResponseEntity.ok(new CandidateSlotDto(candidateSlot));
}
```

Рис. 2.4. Контролер кандидата (2)

Метод updateCandidateSlot обробляє POST-запит для редагування CandidateSlot. Анотація @PathVariable вказує, що значення slotId буде витягнуто зі шляху запиту. Решта логіки аналогічна методу createCandidateSlot, за винятком того, що перед збереженням CandidateSlot метод викликає candidateSlotService.update для поновлення наявного слота (рис. 2.5.).



```
/**
 * POST request for editing the CandidateSlot.
 * First we do the conversion, then we pass it to the validation,
 * and then we send it to the service for updating.
 *
 * @param request - Request body of POST mapping.
 * @param id - Parameter from the request mapping. This is the slot id for update.
 *
 * @return ResponseEntity - Response of the updated object converted to a DTO.
 *
 * @throws SlotException - when parameters are incorrect or updated slot is booked
 *     or slot is overlapping.
 */
@PostMapping("/candidates/current/slots/{slotId}")
public ResponseEntity<CandidateSlotDto> updateCandidateSlot(@RequestBody CandidateSlotDto request,
    @PathVariable("slotId") Long id, Authentication authentication)
    throws SlotException {
  CandidateSlot candidateSlot = getCandidateSlotFromDto(request, authentication);
  candidateSlot.setId(id);

  candidateSlotValidator.validateUpdating(candidateSlot);

  candidateSlot = candidateSlotService.update(candidateSlot);

  return ResponseEntity.ok(new CandidateSlotDto(candidateSlot));
}
```

Рис. 2.5. Контролер кандидата (3)

Метод getAllSlotsOfCandidate обробляє GET-запит для отримання всіх слотів поточного кандидата. Анотація @GetMapping вказує, що цей метод обробляє GET-запити за вказаним шляхом "/candidates/current/slots". Метод витягує аутентифікованого користувача з Authentication і використовує його електронну пошту для отримання списку слотів за допомогою candidateSlotService. Потім повертається ResponseEntity з HTTP-статусом 200 (OK) і перетвореним об'єктом CandidateSlotsDto (Рис. 2.6.).

```java
/**
 * GET request for getting all slots of current Candidate.
 *
 * @return ResponseEntity - Response of the list of slots converted to a DTO.
 */
@GetMapping("/candidates/current/slots")
public ResponseEntity<CandidateSlotsDto> getAllSlotsOfCandidate(Authentication authentication) {

  JwtUserDetails jwtUserDetails = (JwtUserDetails) authentication.getPrincipal();

  List<CandidateSlot> candidateSlots = candidateSlotService
      .getAllSlotsByEmail(jwtUserDetails.getEmail());

  return ResponseEntity.ok(getCandidateSlotsDtoFromListOf(candidateSlots));
}
```

Рис. 2.6. Контролер кандидата (4)

```java
/**
 * Converts the candidate slot from the DTO.
 *
 * @param candidateSlotDto - DTO of Candidate slot.
 *
 * @return CandidateSlot object by given DTO.
 */
private CandidateSlot getCandidateSlotFromDto(CandidateSlotDto candidateSlotDto,
    Authentication authentication) throws SlotException {

  JwtUserDetails jwtUserDetails = (JwtUserDetails) authentication.getPrincipal();

  return candidateSlotService.createCandidateSlot(candidateSlotDto.getDate(),
      candidateSlotDto.getFrom(), candidateSlotDto.getTo(), jwtUserDetails.getEmail(),
      jwtUserDetails.getName());
}
```

Рис. 2.7. Контролер кандидата (5)

Приватний метод getCandidateSlotFromDto виконує конвертацію CandidateSlotDto в CandidateSlot з використанням candidateSlotService.createCandidateSlot. Цей метод також вимагає аутентифікації користувача (Authentication), щоб отримати його деталі (Рис. 2.7.).

Приватний метод getCandidateSlotsDtoFromListOf виконує перетворення списку CandidateSlot у список CandidateSlotDto. Він використовує стрім для перетворення кожного CandidateSlot в CandidateSlotDto і збирає їх у список. Потім він створює об'єкт CandidateSlotsDto, встановлює список CandidateSlotDto і повертає його (Рис. 2.8.).

```
/**
 * Created the list of CandidateSlotDto object by given list of CandidateSlot.
 *
 * @param candidateSlotList - List of candidates to convert.
 *
 * @return candidateSlotsDto - List of DTO by given list of CandidateSlot.
 */
private CandidateSlotsDto getCandidateSlotsDtoFromListOf(List<CandidateSlot> candidateSlotList) {
  List<CandidateSlotDto> candidateSlotDtoList = candidateSlotList
      .stream()
      .map(CandidateSlotDto::new)
      .collect(Collectors.toList());

  CandidateSlotsDto candidateSlotsDto = new CandidateSlotsDto();
  candidateSlotsDto.setCandidateSlotDtoList(candidateSlotDtoList);

  return candidateSlotsDto;
}
```

Рис. 2.8. Контролер кандидата (6)

Таким чином у результаті ми маємо контролер для кандидата, який повністю виконує свої функції, дотримуючи усіх правил та стандартів.

Таким же чином було розроблено усі інші визначені нами контролери із визначними запитами.

2.4.4. Розробка безпеки серверної частини

Для забезпечення безпеки серверної частини системи розроблено різні класи та компоненти, які виконують низку завдань, як автентифікація, авторизація та опрацювання винятків. Ці компоненти та їхні ключові функції описано нижче.

SecurityConfig.java: Цей клас представляє конфігурацію безпеки для програми. Він визначає параметри безпеки та налаштовує різні фільтри й обробники. У цьому класі налаштовуються фільтри авторизації, аутентифікації, обробники помилок і правила доступу.

Приклад коду з класу SecurityConfig.java (рис. 2.9.):

```
/** Configuring requests security. */
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

  http.csrf().disable() HttpSecurity
        .cors().and()
      .authorizeRequests().antMatchers( ...antPatterns: "/authenticate", "/oauth2/facebook/v15.0",
          "/weeks/current", "/weeks/next").permitAll() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
      .antMatchers( ...antPatterns: "/candidates/**").hasRole("CANDIDATE")
      .antMatchers( ...antPatterns: "/interviewers/**").hasAnyRole( ...roles: "INTERVIEWER", "COORDINATOR")
      .antMatchers( ...antPatterns: "/bookings", "/bookings/**", "/users", "/users/**",
          "/weeks/{weekNum}/dashboard").hasRole("COORDINATOR")
      .anyRequest().authenticated().and() HttpSecurity

      // Add custom handling for unauthenticated and access denied errors
      .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint) ExceptionHandlingConfigurer<HttpSecurity>
      .and() HttpSecurity
      .exceptionHandling().accessDeniedHandler(jwtAccessDeniedHandler) ExceptionHandlingConfigurer<HttpSecurity>
      .and() HttpSecurity
      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) SessionManagementConfigurer<HttpSecurity>
      .and() HttpSecurity
      // Add a filter to validate the tokens with every request
      .addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class)
      // ExceptionHandler filter
      .addFilterBefore(filterChainExceptionHandler, JwtRequestFilter.class);

  return http.build();
}
```

Рис. 2.9. Метод класу SecurityConfig

У наведеному прикладі визначено правила доступу до різних URL-адрес. Тут вказується, що запити до «/authenticate», «/oauth2/facebook/v15.0», «/weeks/current», «/weeks/next» доступні без автентифікації, тоді як інші запити мають бути автентифіковані до певної ролі системи.

FilterChainExceptionHandler.java: Цей клас представляє обробник винятків для ланцюжка фільтрів безпеки. Він обробляє будь-які винятки, що виникли під час виконання фільтрів, і повертає відповідні HTTP-відповіді з помилками.

Приклад коду з класу FilterChainExceptionHandler.java (рис. 2.10.):

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
    FilterChain filterChain) throws ServletException, IOException {

  try {
    filterChain.doFilter(request, response);
  } catch (SecurityException exception) {
    resolver.resolveException(request, response, null, exception);
  }
 }
}
```

Рис. 2.10. Метод класу FilterChainExceptionHandler

У цьому класі реалізовано метод doFilterInternal, який обробляє винятки, що виникли в ланцюжку фільтрів безпеки, і формує відповідну відповідь із помилкою.

JwtAccessDeniedHandler.java: Цей клас представляє обробник помилки доступу до ресурсу. Якщо у користувача немає необхідних прав доступу, то цей обробник буде викликаний і поверне відповідну відповідь із помилкою доступу.

Приклад коду з класу JwtAccessDeniedHandler.java (рис. 2.11.):

У цьому класі реалізовано метод handle, який обробляє помилку доступу і формує відповідну відповідь із помилкою.

JwtAuthenticationEntryPoint.java: Цей клас представляє точку входу в аутентифікацію. Якщо аутентифікація користувача не вдалася, то цей обробник буде викликаний і поверне відповідну відповідь із помилкою аутентифікації.

```
@Override
public void handle(HttpServletRequest request, HttpServletResponse response,
    AccessDeniedException accessDeniedException) throws IOException, ServletException {

  resolver.resolveException(request, response, null, new SecurityException(
      SecurityExceptionProfile.ACCESS_DENIED));
}
```

Рис. 2.11. Метод класу JwtAccessDeniedHandler

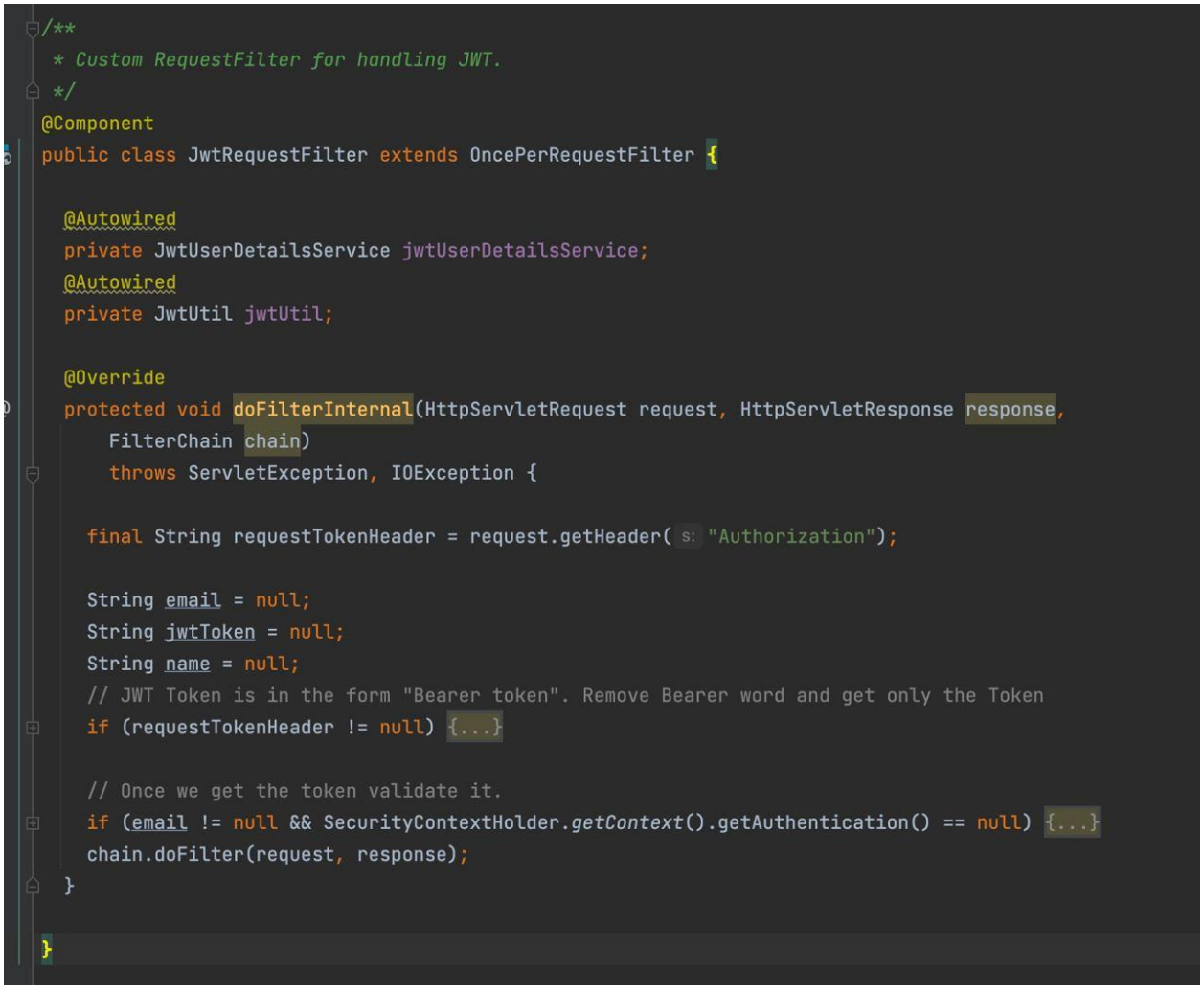
```
@Override
public void commence(HttpServletRequest request, HttpServletResponse response,
    AuthenticationException authException) throws IOException {

  resolver.resolveException(request, response, null,
      new SecurityException(SecurityExceptionProfile.UNAUTHENTICATED));
}
}
```

Рис. 2.12. Метод класу JwtAuthenticationEntryPoint

Приклад коду з класу JwtAuthenticationEntryPoint.java (рис. 2.12.):

У цьому класі реалізовано метод commence, який обробляє помилку аутентифікації та формує відповідну відповідь із помилкою.

JwtRequestFilter.java: Цей клас представляє фільтр для обробки JWT-токенів. Він перевіряє наявність і валідність токена в запиті, а також аутентифікує користувача.



```java
/**
 * Custom RequestFilter for handling JWT.
 */
@Component
public class JwtRequestFilter extends OncePerRequestFilter {

  @Autowired
  private JwtUserDetailsService jwtUserDetailsService;
  @Autowired
  private JwtUtil jwtUtil;

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
      FilterChain chain)
      throws ServletException, IOException {

    final String requestTokenHeader = request.getHeader( s: "Authorization");

    String email = null;
    String jwtToken = null;
    String name = null;
    // JWT Token is in the form "Bearer token". Remove Bearer word and get only the Token
    if (requestTokenHeader != null) {...}

    // Once we get the token validate it.
    if (email != null && SecurityContextHolder.getContext().getAuthentication() == null) {...}
    chain.doFilter(request, response);
  }

}
```

Рис. 2.13. Клас JwtRequestFilter

Приклад коду з класу JwtRequestFilter.java (рис. 2.13.):

У цьому класі реалізовано метод doFilterInternal, який здійснює перевірку токена, аутентифікацію користувача та встановлення контексту безпеки.

JwtUserDetails.java: Цей клас представляє модель користувацьких даних для автентифікації та авторизації. Він містить інформацію про роль користувача, його ідентифікатор, ім'я та інші властивості, необхідні для доступу та обробки запитів.

Приклад коду з класу JwtUserDetails.java (рис. 2.14.):

```
    public String getEmail() { return email; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() { return authorities; }

    @Override
    public String getPassword() { return password; }

    @Override
    public String getUsername() { return email; }

    @Override
    public boolean isAccountNonExpired() { return accountNonExpired; }

    @Override
    public boolean isAccountNonLocked() { return accountNonLocked; }

    @Override
    public boolean isCredentialsNonExpired() { return credentialsNonExpired; }

    @Override
    public boolean isEnabled() { return enabled; }
}
```

Рис. 2.14. Інформація про користувача у класі JwtUserDetails

У цьому класі реалізовано методи інтерфейсу UserDetails, такі як getAuthorities, які повертають ролі користувача.

JwtUserDetailsService.java: Цей клас представляє сервіс для роботи з даними користувачів. Він завантажує інформацію про користувача з бази даних або інших джерел, використовуючи ідентифікатор користувача, і повертає об'єкт JwtUserDetails.

Приклад коду з класу JwtUserDetailsService.java (рис. 2.15.):

```
@Override
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

  User user = userRepository.findByEmail(email);

  Set<GrantedAuthority> authorities = new HashSet<>();

  if (user != null && user.getRole() != null) {
    authorities.add(new SimpleGrantedAuthority( role: "ROLE_" + user.getRole().toString()));
  } else {
    authorities.add(new SimpleGrantedAuthority( role: "ROLE_CANDIDATE"));
  }

  return new JwtUserDetails(email,  name: null, passwordEncoder().encode(email), authorities);
}
}
```

Рис. 2.15. Метод класу JwtUserDetailsService

У цьому класі реалізовано метод loadUserByUsername, який завантажує інформацію про користувача за його поштовою адресою.

Усі перераховані вище класи та компоненти взаємодіють один з одним для забезпечення безпеки серверної частини системи. Вони обробляють аутентифікацію, авторизацію та управління доступом до різних ресурсів залежно від ролі користувача.

2.4.5. Розробка сутностей та бази даних серверної частини

Після розробки безпеки та контролерів у нашій серверній частині системи слід визначити сутності, якими ми будемо оперувати у нашому додатку. Вищначемо наступні сутнсті: User – користувач, CandidateSlot – зарезервований проміжок часу кандидата, InterviewerSlot - зарезервований проміжок часу інтерв’юера, Booking – затверджений проміжок часу координатором для співбесіди, BookingLimit – ліміт на проведення співбесід для інтерв’юера, Period – об’єкт проміжку часу, Week – номер тижня по якому йдуть співбесіди. [15]

Так як ми вирішили використовувати стандарт JPA, то наші сутності, завдяки Hibernate анотаціям, будуть автоматично визначатися між класами Java і таблицями

бази даних, а поля класів будуть атрибутами таблиці бази даних. У зв’язку з цим нам необхідно перед створенням сутностей у вигляді класів визначити зв’язки між ними. Усі сутності та зв’язки між ними можна побачити на рис. 2.16.

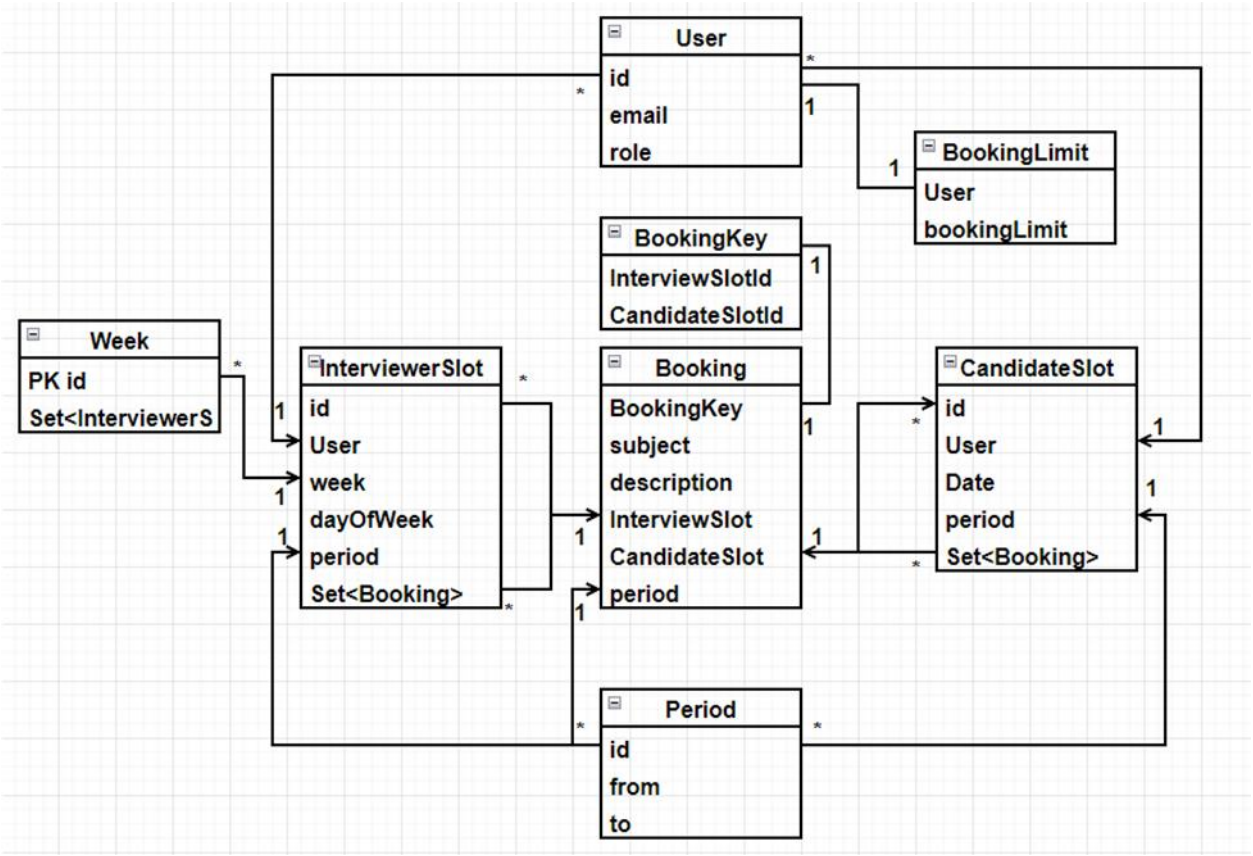


Рис. 2.16. Сутності та зв’язки між ними

Коли ми визначили усі сутності та зв’язки між ними, можна почати будувати ці сутності у наш додаток.

Почнемо будувати сутність CandidateSlot (рис. 2.17.):

Анотація @Entity вказує, що клас CandidateSlot є сутністю, яка буде відображатися в базі даних. Анотація @Table(name = "candidate_slots") задає ім'я таблиці в базі даних, у якій зберігатимуться об'єкти цього класу.

Анотація @Id вказує, що поле id є первинним ключем сутності. Анотація @GeneratedValue(strategy = GenerationType.IDENTITY) визначає стратегію генерації

значень для цього поля, у цьому випадку використовується автоматичне інкрементування.

Поле date типу LocalDate призначене для зберігання дати слота співбесіди.

Анотація @ManyToOne вказує на зв'язок "багато-до-одного" між сутностями CandidateSlot і Period. Анотація @JoinColumn(name = "period_id") задає ім'я стовпця, який буде використовуватися для зв'язку з таблицею Period у базі даних.

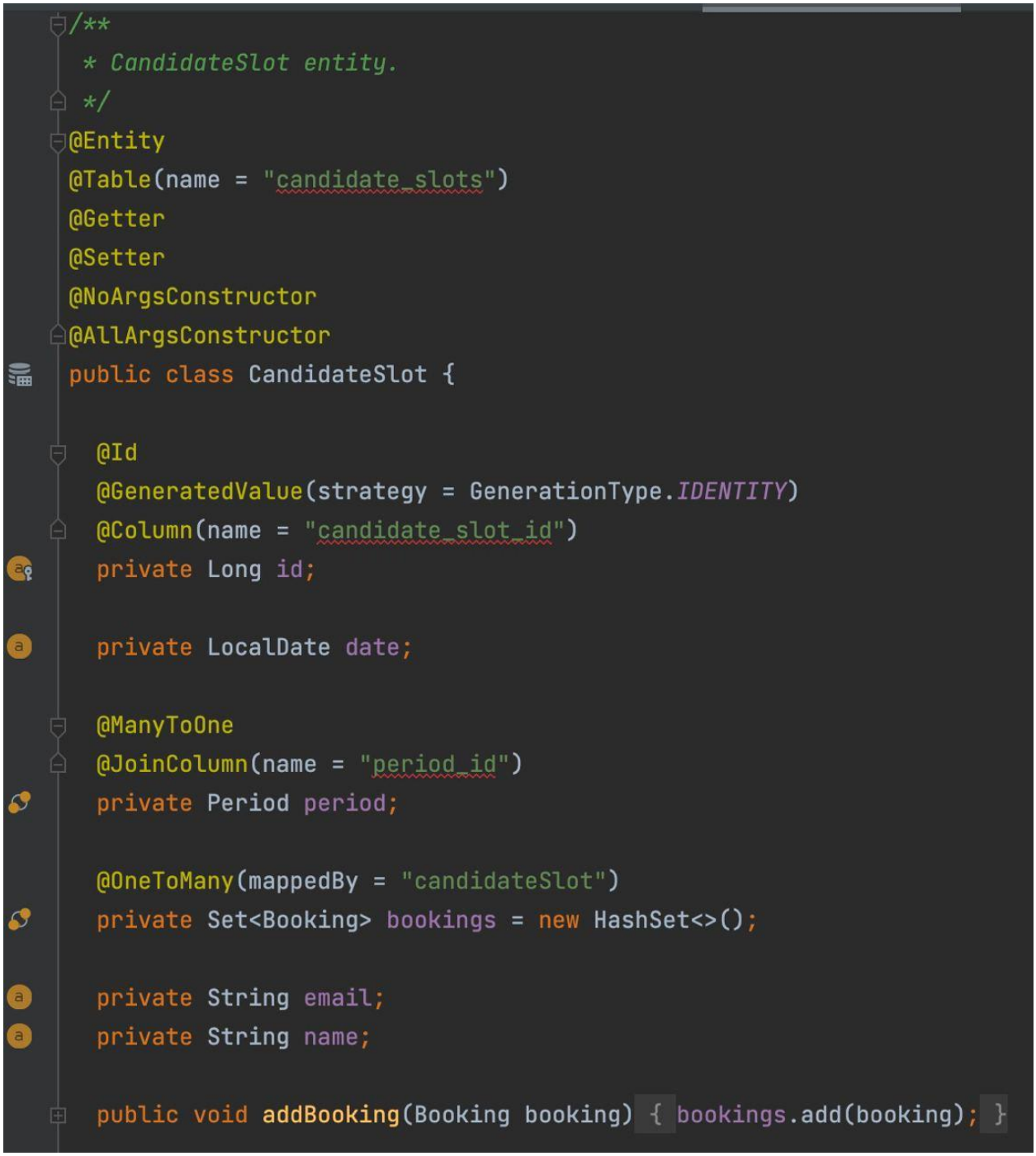
Анотація @OneToMany(mappedBy = "candidateSlot") вказує на зв'язок "один-до-багатьох" між сутностями CandidateSlot і Booking. Поле bookings являє собою колекцію об'єктів Booking, які пов'язані з даним слотом співбесіди.

Поля email та name призначені для зберігання інформації про кандидата, який забронював цей слот.

Метод addBooking(Booking booking) додає нову бронь співбесіди в колекцію bookings.

Таким чином, клас CandidateSlot представляє основну сутність, яка буде зберігатися в базі даних і містить інформацію про доступні слоти для співбесід в IT-компаніях. Він володіє необхідними полями та методами для зручної роботи з даними в рамках розроблюваної системи для планування співбесід.

Таким самим чином створено і реалізовано усі інші зазначені у рис. 2.16. сутності, відповідаючи нашим зв’язкам та вимогам з використанням відповідних анотацій до кожного класу та атрибуту.



```java
/**
 * CandidateSlot entity.
 */
@Entity
@Table(name = "candidate_slots")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class CandidateSlot {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "candidate_slot_id")
    private Long id;

    private LocalDate date;

    @ManyToOne
    @JoinColumn(name = "period_id")
    private Period period;

    @OneToMany(mappedBy = "candidateSlot")
    private Set<Booking> bookings = new HashSet<>();

    private String email;
    private String name;

    public void addBooking(Booking booking) { bookings.add(booking); }
```

Рис. 2.17. Клас CandidateSlot

2.4.6. Підключення бази даних до додатка та розробка репозиторіїв

2.4.6.1. Підключення та налаштування бази даних

У нашому додатку є конфігураційний файл application.yml, який містить налаштування нашого проекту і саме тут і буде відбуватися налаштування та підключення нашої бази даних до нашого серверного додатку. Тож, у цьому файлі ми повинні написати код, як на рис. 2.18.

```yaml
jpa:
  show-sql: true
  generate-ddl: true
  hibernate:
    ddl-auto: ${HIBERNATE_DDL_AUTO}
  database: postgresql

datasource:
  platform: postgres
  url: "jdbc:postgresql://postgresql:${POSTGRES_PORT}/${POSTGRES_DB}"
  username: ${POSTGRES_USER}
  password: ${POSTGRES_PASSWORD}
  driver-class-name: org.postgresql.Driver
  initialization-mode: always
  continue-on-error: true
```

Рис. 2.18. Налаштування та підключення бази даних

Ретельно розглянем цей код:

- jpa:
  - show-sql: true: Цей рядок вказує, що під час виконання запитів до бази даних буде виводитися відповідний SQL-код у логування. Це корисно для налагодження і розуміння того, які запити генерує JPA.

- generate-ddl: true: Тут вказується, що JPA має генерувати DDL-скрипти (Data Definition Language) для створення таблиць бази даних на основі визначення сутностей JPA. DDL-скрипти використовуються для створення та зміни схеми бази даних.
- hibernate:
  - ddl-auto: ${HIBERNATE_DDL_AUTO}: Тут налаштовується параметр ddl-auto для Hibernate, який є реалізацією JPA. Значення цього параметра береться зі змінної оточення HIBERNATE_DDL_AUTO. ddl-auto визначає стратегію автоматичного оновлення схеми бази даних. Значення HIBERNATE_DDL_AUTO буде задано в іншому місці, змінній оточення.
- database: postgresql: Тут вказується, що використовується база даних PostgreSQL. Це важливо для правильного налаштування драйвера JDBC (Java Database Connectivity) та інших специфічних параметрів.

- datasource:
  - platform: postgres: Вказується, що використовується платформа PostgreSQL. Це впливає на вибір певних функцій і параметрів під час роботи з базою даних.
  - url: "jdbc:postgresql://postgresql:${POSTGRES_PORT}/${POSTGRES_DB}": У цьому рядку задається URL-адреса бази даних PostgreSQL, до якої відбувається підключення. Вона формується з використанням змінних оточення POSTGRES_PORT і POSTGRES_DB. POSTGRES_PORT і POSTGRES_DB також матимуть значення, задані в іншому місці, змінних оточення.
  - username: ${POSTGRES_USER}: Тут вказується ім'я користувача для підключення до бази даних. Знову ж таки, значення береться зі змінної оточення POSTGRES_USER.
  - password: ${POSTGRES_PASSWORD}: Цей рядок задає пароль для підключення до бази даних. Значення береться зі змінної оточення POSTGRES_PASSWORD.

- driver-class-name: org.postgresql.Driver: Тут вказується клас драйвера JDBC, який буде використовуватися для підключення до бази даних PostgreSQL.
- initialization-mode: always: Тут вказується, що ініціалізація бази даних повинна відбуватися завжди під час старту програми. Це означає, що Hibernate буде створювати таблиці та заповнювати їх тестовими даними під час кожного запуску програми.
- continue-on-error: true: Цей параметр вказує, що процес ініціалізації бази даних повинен продовжуватися в разі помилок. Якщо встановлено значення false, у разі виникнення помилки ініціалізації ініціалізацію буде перервано.

Це основні налаштування у файлі application.yml, що використовує JPA і PostgreSQL. Вони визначають різні аспекти взаємодії з базою даних, включно з виведенням SQL-коду, генерацією DDL-скриптів, налаштуваннями Hibernate, під'єднанням до бази даних та ініціалізацією схеми бази даних.

2.4.6.2. Розробка репозиторіїв

Репозиторії надають абстракцію для роботи з базою даних, тому для того щоб проводити операції створення, оновлення, читання або видалення наших сутностей, нам потрібно створити класи репозиторій та налаштувати потрібні нам операції з БД.

Для нашого серверного додатку необхідно створити наступні репозиторії: CandidateSlotRepository, BookingLimitRepository, BookingRepository, InterviewerSlotRepository, PeriodRepository, UserRepository, WeekRepository.

Створемо CandidateSlotRepository (рис. 2.19.).

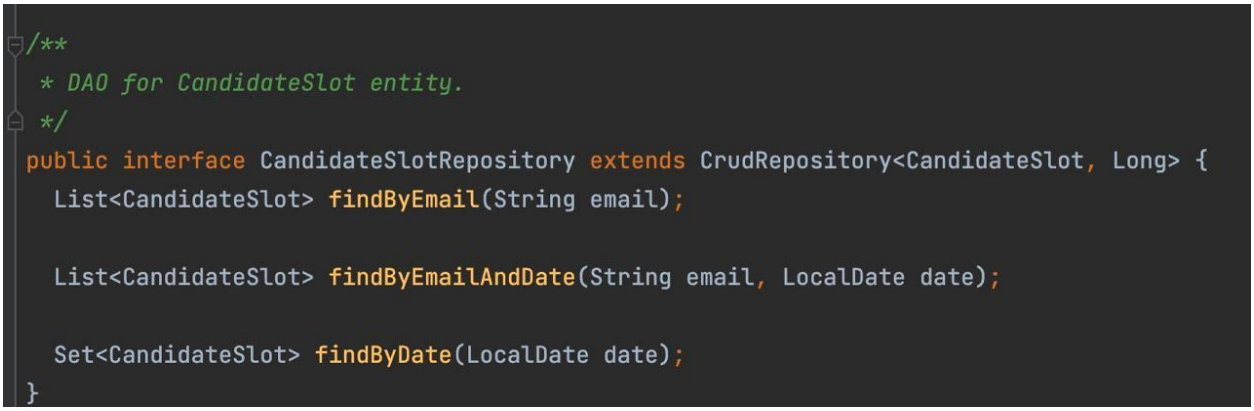


Рис. 2.19. Клас CandidateSlotRepository

У наведеному коді інтерфейс "CandidateSlotRepository" розширює інтерфейс "CrudRepository" і вказує, що сутність "CandidateSlot" має тип ідентифікатора "Long". "CrudRepository" - це інтерфейс із Spring Data, який надає основні методи для виконання операцій CRUD (Create, Read, Update, Delete) над сутністю.

Далі, в інтерфейсі "CandidateSlotRepository" визначено кілька додаткових методів, які дають змогу виконувати пошук сутностей "CandidateSlot" за різними критеріями:

1. Метод "findByEmail" приймає параметр "email" і повертає список об'єктів "CandidateSlot", пов'язаних із зазначеною адресою електронної пошти. Цей метод дає змогу знайти всі слоти кандидата за його адресою електронної пошти.

2. Метод "findByEmailAndDate" приймає два параметри - "email" і "date", і повертає список об'єктів "CandidateSlot", пов'язаних із зазначеною адресою електронної пошти і датою. Цей метод дає змогу знайти всі слоти кандидата за його адресою електронної пошти та датою.

3. Метод "findByDate" приймає параметр "date" і повертає набір об'єктів "CandidateSlot", пов'язаних із зазначеною датою. Цей метод дає змогу знайти всі слоти кандидатів за вказаною датою.

Розроблений інтерфейс "CandidateSlotRepository" дає змогу легко виконувати різноманітні запити до бази даних, використовуючи стандартні методи з "CrudRepository" і додаткові методи, визначені в інтерфейсі. Це забезпечує зручність

і гнучкість під час роботи з даними та спрощує розробку бізнес-логіки системи планування співбесід в IT-компаніях.

Таким самим чином розроблено усі інші репозиторії, які зазначені раніше із потрібними додатковими методами визначеними в інтерфейсі, якщо це необхідно.

2.4.7. Розробка бізнес-логіки системи

За бізнес-логіку нашої системи будуть відповідати наступні класи-сервіси та допоміжні класи валідації даних нашої системи: CandidateSlotService, CandidateSlotValidator, BookingService, BookingLimitService, InterviewerSlotService, InterviewerSlotDtoValidator, PeriodService, PeriodValidator, TimeService, UserService, WeekService.

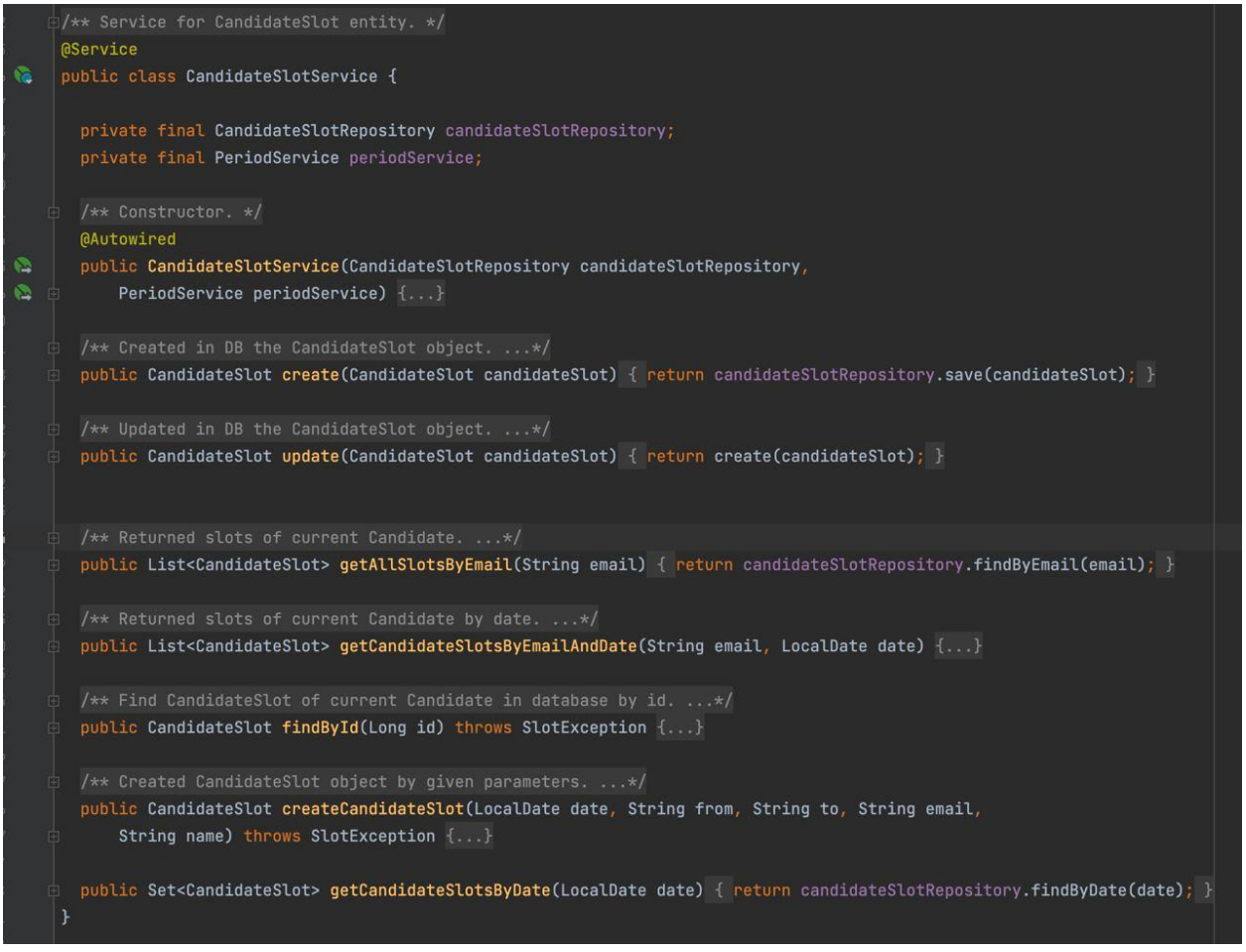
Розглянемо розробку класу CandidateSlotService та CandidateSlotValidator.

Клас CandidateSlotService (рис. 2.20.) є сервісом для роботи із сутністю CandidateSlot. Він надає набір функцій для створення, оновлення та отримання даних, пов'язаних з об'єктами CandidateSlot.

Основні функції та ідеї класу:

- Конструктор класу приймає два аргументи - CandidateSlotRepository і PeriodService, які інжектуються за допомогою автоматичної пров'язки залежностей (Autowired).
- Метод create дає змогу створити новий об'єкт CandidateSlot і зберегти його в базі даних. Повертає збережений об'єкт.
- Метод update дає змогу оновити дані об'єкта CandidateSlot у базі даних. Він викликає метод create для збереження оновлених даних. Повертає оновлений об'єкт.
- Метод getAllSlotsByEmail повертає список усіх слотів Candidate, пов'язаних із зазначеною адресою електронної пошти.
- Метод getCandidateSlotsByEmailAndDate повертає список слотів Candidate, пов'язаних із зазначеною адресою електронної пошти і датою.

- Метод findById знаходить і повертає об'єкт CandidateSlot за заданим ідентифікатором. Якщо слот із зазначеним ідентифікатором не знайдено, викидається виняток SlotException.
- Метод createCandidateSlot створює об'єкт CandidateSlot на основі заданих параметрів, як дата, час початку і закінчення слота, адреса електронної пошти та ім'я. Повертає створений об'єкт. Якщо відбувається помилка при створенні слота, викидається виняток SlotException.
- Метод getCandidateSlotsByDate повертає набір слотів Candidate, пов'язаних із зазначеною датою.



```java
/** Service for CandidateSlot entity. */
@Service
public class CandidateSlotService {

  private final CandidateSlotRepository candidateSlotRepository;
  private final PeriodService periodService;

  /** Constructor. */
  @Autowired
  public CandidateSlotService(CandidateSlotRepository candidateSlotRepository,
      PeriodService periodService) {...}

  /** Created in DB the CandidateSlot object. ...*/
  public CandidateSlot create(CandidateSlot candidateSlot) { return candidateSlotRepository.save(candidateSlot); }

  /** Updated in DB the CandidateSlot object. ...*/
  public CandidateSlot update(CandidateSlot candidateSlot) { return create(candidateSlot); }


  /** Returned slots of current Candidate. ...*/
  public List<CandidateSlot> getAllSlotsByEmail(String email) { return candidateSlotRepository.findByEmail(email); }

  /** Returned slots of current Candidate by date. ...*/
  public List<CandidateSlot> getCandidateSlotsByEmailAndDate(String email, LocalDate date) {...}

  /** Find CandidateSlot of current Candidate in database by id. ...*/
  public CandidateSlot findById(Long id) throws SlotException {...}

  /** Created CandidateSlot object by given parameters. ...*/
  public CandidateSlot createCandidateSlot(LocalDate date, String from, String to, String email,
      String name) throws SlotException {...}

  public Set<CandidateSlot> getCandidateSlotsByDate(LocalDate date) { return candidateSlotRepository.findByDate(date); }
}
```

Рис. 2.20. Клас CandidateSlotService

Клас, з назвою "CandidateSlotValidator" (рис. 2.21.), є валідатором для об'єкта "CandidateSlot". Він містить кілька методів для перевірки та валідації різних аспектів об'єкта "CandidateSlot".

Ось основні ключові моменти та функції класу:

- Конструктор: клас має конструктор, який приймає два параметри типу CandidateSlotService і PeriodService. Він використовується для впровадження залежностей цих сервісів у клас.
- Метод "validateCreating": цей метод перевіряє об'єкт CandidateSlot для визначення того, чи повинен слот перебувати в майбутньому, і чи не перекривається він з іншими слотами. Якщо перевірка не проходить, викидається виняток типу SlotException.
- Метод "validateUpdating": цей метод використовується для перевірки об'єкта CandidateSlot під час його оновлення. Він перевіряє, чи існує слот, чи не є він забронюваним, а також виконує перевірку, як у методі "validateCreating".
- Метод "validateSlotInFuture": цей метод перевіряє, чи перебуває дата в об'єкті CandidateSlot у майбутньому. Якщо дата в минулому, викидається виняток типу SlotException.
- Метод "validateOverlapping": цей метод перевіряє, чи не перекривається поточний слот з іншими слотами. Він отримує список усіх слотів, пов'язаних з адресою електронної пошти і датою об'єкта CandidateSlot. Потім відбувається перевірка на перекриття слотів. Якщо виявлено перекриття, викидається виняток типу SlotException.
- Метод "validateSlotIsBookingAndTheSlotExists": цей метод перевіряє, чи існує вказаний номер слота в базі даних і чи не забронюваний він. Він отримує номер слота і виконує перевірку. Якщо номер не знайдено або слот заброньовано, викидається виняток типу SlotException.
- Впровадження залежностей: клас використовує впровадження залежностей за допомогою анотації @Autowired для отримання екземплярів CandidateSlotService і PeriodService.

Головна ідея цього класу полягає в тому, щоб надати методи для валідації об'єкта CandidateSlot у різних сценаріях, як створення нового слота й оновлення наявного слота. Він виконує перевірки, щоб переконатися, що слот перебуває в майбутньому, не перекривається з іншими слотами і не забронюваний. У разі порушення будь-якої з цих умов викидається виняток для обробки помилок.

Таким же самим чином було розроблено усі інші сервіси для виконання нашої бізнес-логіки відповідно до вимог системи.

```java
/** Validator for CandidateSlot. */
@Service
public class CandidateSlotValidator {
  private final CandidateSlotService candidateSlotService;
  private final PeriodService periodService;

  @Autowired
  public CandidateSlotValidator(CandidateSlotService candidateSlotService,
      PeriodService periodService) {...}

  /** Validate CandidateSlot object for what the slot should be in the future, ...*/
  public void validateCreating(CandidateSlot candidateSlot) throws SlotException {...}

  /** Validate CandidateSlot object for all slot creation checks, ...*/

  public void validateUpdating(CandidateSlot candidateSlot) throws SlotException {...}

  /** Validate that date in CandidateSlot in the future. ...*/
  private void validateSlotInFuture(CandidateSlot candidateSlot) throws SlotException {...}

  /** Validate that the slot does not overlap with other slots. ...*/
  private void validateOverlapping(CandidateSlot candidateSlot) throws SlotException {...}

  /** Validate for the given number of slot is exist in DB, ...*/
  private void validateSlotIsBookingAndTheSlotExists(Long id)
      throws SlotException {...}
}
```

Рис. 2.21. Клас CandidateSlotValidator

2.5. Розробка клієнтської частини системи

2.5.1. Файлова структура клієнтського додатку

Структура нашого сайту включає такі основні директорії та файли:

- interview-planning-aplication: Головна папка нашого клієнтського додатка.
    - css: В цій директорії знаходяться файли, що відповідають за стилізацію і зовнішній вигляд веб-сторінок нашого клієнтського додатка. Тут можуть міститися файли CSS (Cascading Style Sheets), які визначають правила форматування елементів HTML, такі як кольори, шрифти, розміри, відступи тощо. Використовуючи файли CSS, ми можемо надати нашому додатку привабливий та спеціально оформлений вигляд.
    - html: У цій директорії розташовуються файли HTML (HyperText Markup Language), які визначають структуру та зміст веб-сторінок нашого клієнтського додатка. Вони використовуються для створення різних елементів, таких як заголовки, абзаци, таблиці, форми, зображення та інші. Файли HTML визначають, як вміст сторінки повинен бути відображений та організований.
    - js: У цій директорії знаходяться файли JavaScript, які відповідають за функціональність та динамічність нашого клієнтського додатка. JavaScript є мовою програмування, яка використовується для створення інтерактивних елементів на веб-сторінках. Вона дозволяє додавати різноманітні функції, обробники подій, валідацію форм, надсилання запитів на сервер, анімацію та інші можливості, які роблять наш додаток більш живим та зручним для користувачів.
- index.html: Це головний файл, який представляє собою вхідну точку нашого клієнтського додатка. Він відображається як головна сторінка нашого сайту.

2.5.2. Розробка головної сторінки

Головна сторінка index.html являє собою HTML-сторінку, яка слугує частиною сайту для планування інтерв'ю. При розробці коду цієї сторінки, можна виділити кілька ключових моментів і функцій цього файлу:

Заголовок і стилі: У файлі визначено стилі для елементів сторінки, як колір фону, шрифт і розмір тексту, а також зовнішній вигляд контейнера і кнопки входу (рис. 2.22.).

```
    <style>
        body {
            background-color: #F7F7F7;
            font-family: 'Open Sans', sans-serif;
            font-size: 16px;
            line-height: 1.5;
        }

        #container {
            width: 800px;
            margin: 0 auto;
            padding: 20px;
            background-color: #FFFFFF;
            border-radius: 10px;
        }

        #signin {
            text-align: center;
        }

        #signin a {
            padding: 10px;
            background-color: #3B5998;
            color: #FFFFFF;
            text-decoration: none;
            border-radius: 5px;
        }
    </style>
</head>
```

Рис. 2.22. Стилі сторінки index.html

Розмітка контенту: Усередині елемента з ідентифікатором "container" міститься заголовок сторінки "Interview planning application" і елемент "signin" з кнопкою "Sign in with Facebook" (рис. 2.23.).

```
<body>
    <div id="container">
        <h1>Interview planning application</h1>
        <div id="signin">
            <a href="#" onclick="loginWithFacebook()">Sign in with Facebook</a>
        </div>
    </div>
    <script>…
    </script>
```

Рис. 2.23. Розмітка сторінки index.html

Ініціалізація Facebook JavaScript SDK: Файл підключає Facebook JavaScript SDK, використовуючи зовнішнє посилання. У функції fbAsyncInit відбувається ініціалізація SDK із зазначенням параметрів, включно з ідентифікатором програми. Потім викликається FB.AppEvents.logPageView() для відстеження подій на сторінці (рис. 2.24.).

```
// Initialize the Facebook JavaScript SDK
window.fbAsyncInit = function () {
    FB.init({
        appId: '240435898644410',
        cookie: true,
        xfbml: true,
        version: 'v2.8'
    });
    FB.AppEvents.logPageView();
};
```

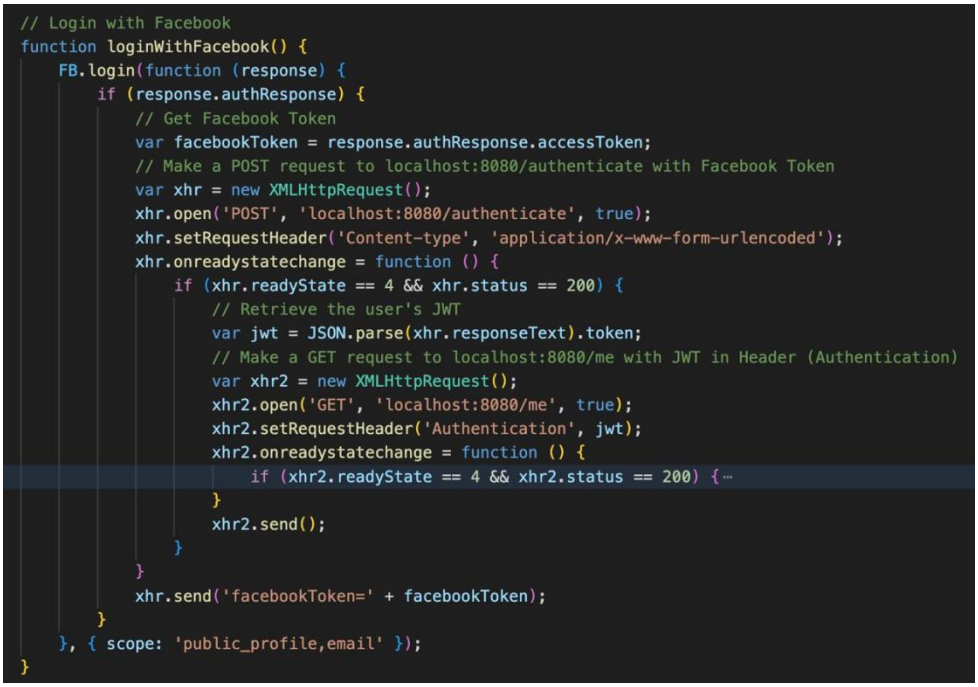
Рис. 2.24. Підключення Facebook JavaScript SDK

Завантаження SDK асинхронно: У цьому коді (рис. 2.25.) визначено функцію, яка асинхронно завантажує Facebook SDK, додаючи відповідний скрипт до HTML-документа.

```
// Load the SDK asynchronously
(function (d, s, id) {
    var js, fjs = d.getElementsByTagName(s)[0];
    if (d.getElementById(id)) { return; }
    js = d.createElement(s); js.id = id;
    js.src = "//connect.facebook.net/en_US/sdk.js";
    fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));
```

Рис. 2.25. Асинхронне завантаження SDK

Вхід за допомогою Facebook: Функція loginWithFacebook() (рис. 2.26.) викликається при натисканні кнопки "Sign in with Facebook". Вона відкриває вікно авторизації Facebook, і в разі успішної аутентифікації відбувається виконання певних дій.



```
// Login with Facebook
function loginWithFacebook() {
    FB.login(function (response) {
        if (response.authResponse) {
            // Get Facebook Token
            var facebookToken = response.authResponse.accessToken;
            // Make a POST request to localhost:8080/authenticate with Facebook Token
            var xhr = new XMLHttpRequest();
            xhr.open('POST', 'localhost:8080/authenticate', true);
            xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
            xhr.onreadystatechange = function () {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    // Retrieve the user's JWT
                    var jwt = JSON.parse(xhr.responseText).token;
                    // Make a GET request to localhost:8080/me with JWT in Header (Authentication)
                    var xhr2 = new XMLHttpRequest();
                    xhr2.open('GET', 'localhost:8080/me', true);
                    xhr2.setRequestHeader('Authentication', jwt);
                    xhr2.onreadystatechange = function () {
                        if (xhr2.readyState == 4 && xhr2.status == 200) {…
                    }
                    xhr2.send();
                }
            }
            xhr.send('facebookToken=' + facebookToken);
        }
    }, { scope: 'public_profile,email' });
}
```

Рис. 2.26. Функція loginWithFacebook()

Аутентифікація за допомогою Facebook: У разі успішної аутентифікації функція loginWithFacebook() отримує токен Facebook (facebookToken) і відправляє його на сервер (POST-запит) для аутентифікації. Для цього використовується об'єкт XMLHttpRequest. Потім отриманий JWT (токен JSON-веб-токена) використовується для виконання GET-запиту до сервера, щоб отримати дані користувача (userData).

Залежно від ролі користувача відбувається перенаправлення на відповідну сторінку (рис. 2.27.).

```
if (xhr2.readyState == 4 && xhr2.status == 200) {
    // Get user data from the JSON response
    var userData = JSON.parse(xhr2.responseText);
    // Redirect to one of the pages depending on the role field
    switch (userData.role) {
        case 'CANDIDATE':
            window.location.href = 'html/candidate.html';
            break;
        case 'COORDINATOR':
            window.location.href = 'html/coordinator.html';
            break;
        case 'INTERVIEWER':
            window.location.href = 'html/interviewer.html';
            break;
    }
}
```

Рис. 2.27. Код перенаправлення на відповідну сторінку після успішної аутентифікації

У результаті ми отримаємо зовнішній вигляд нашої головної сторінки як показано на рис. 2.28.

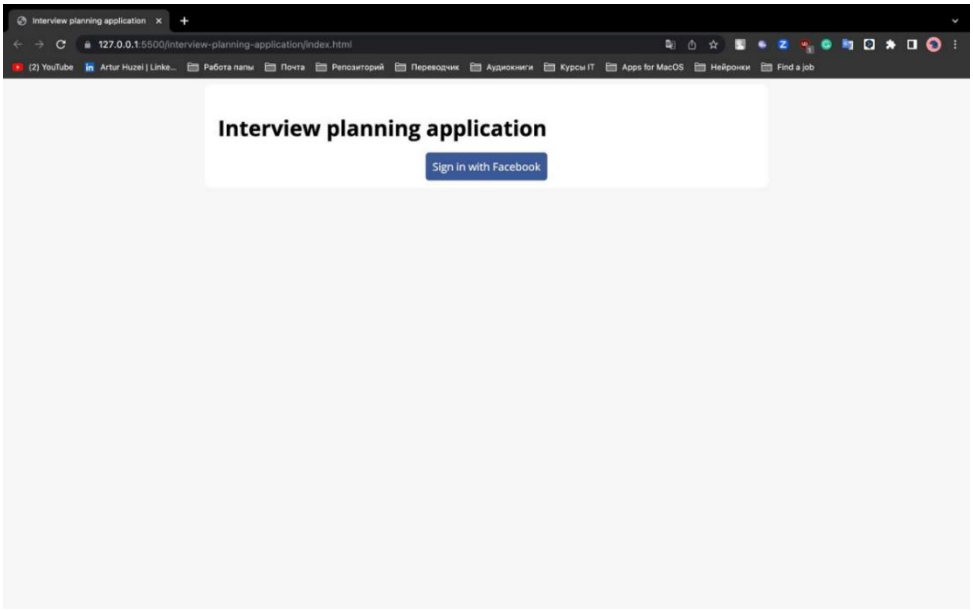


Рис. 2.28. Зовнішній вигляд index.html

2.5.3. Розробка сторінок для кожної ролі системи

Розробимо сторінку для кандидата candidate.html. Цей файл являє собою HTML-сторінку для додатка планування інтерв'ю, з функціональністю, пов'язаній із кандидатами. Ось ключові моменти та головні ідеї розробки цього файлу:

Заголовок і стилі: У файлі визначено заголовок сторінки "Schedule Interview" і посилання на зовнішні стилі, що містяться у файлах css_candidate.css. Стилі описують зовнішній вигляд елементів сторінки, щоб створити узгоджений дизайн (рис. 2.29.).

```
<head>
    <title>Schedule Interview</title>
    <base href="/interview-planning-application/index.html">
    <link rel="stylesheet" href="css/css_candidate.css">
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/xhr2/0.2.0/xhr2.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <script src="js/candidate_js.js"></script>
</head>
```

Рис. 2.29. Заголовок сторінки кандидата candidate.html

JavaScript-бібліотеки: Файл підключає кілька зовнішніх JavaScript-бібліотек, як xhr2.js, jQuery і axios. Ці бібліотеки використовуються для виконання асинхронних запитів до сервера і спрощення роботи з DOM і мережевими запитами (рис. 2.29.).

Розмітка контенту (рис. 2.30.): Усередині елемента з класом "container" міститься розмітка контенту сторінки. Вона складається з двох основних блоків: інформація про кандидата і таблиця слотів.

```
<body>
    <div class="container">
        <div class="top-left">
            <h2>Candidate Information</h2>
            <div class="info">…
            </div>
        </div>
        <div class="calendar-table">
            <h2>Schedule Interview</h2>
            <div class="calendar">…
            </div>
            <div class="buttons">…
            </div>
        </div>
        <div class="create-form" id="create-form">
            <h2>Create Slot</h2>
            <form>…
            </form>
        </div>
        <div class="edit-form" id="edit-form">
            <h2>Edit Slot</h2>
            <form>…
            </form>
        </div>
        <button type="button" class="update-slots" onclick="updateSlots()">Update Current Slots</button>
    </div>
</body>
```

Рис. 2.30. Розмітка контенту candidate.html

Інформація про кандидата: У блоці з класом "top-left" виводиться інформація про кандидата, така як електронна пошта і роль. Дані будуть заповнені пізніше за допомогою JavaScript.

Таблиця слотів: У блоці з класом "calendar-table" знаходиться розмітка для відображення розкладу інтерв'ю. Він містить таблицю із заголовком і слотами для інтерв'ю. Заголовок таблиці містить стовпці "ID", "Date", "From" і "To". Слоти для інтерв'ю будуть додані динамічно під час завантаження даних.

Форми для створення і редагування слотів: У блоці з класом "create-form" знаходиться форма для створення нового слота, а в блоці з класом "edit-form" - форма для редагування наявного слота. Обидві форми містять поля для введення дати, часу початку і часу закінчення інтерв'ю, а також кнопки для виконання дій (створення, редагування і скасування). Функції JavaScript, пов'язані з цими формами, дають змогу відображати і приховувати форми, а також виконувати відповідні дії.

Оновлення поточних слотів: Наприкінці сторінки знаходиться кнопка "Update Current Slots", яка викликає функцію JavaScript updateSlots(). Ця функція виконує запит до сервера для оновлення поточного розкладу інтерв'ю.

Далі розглянемо розробку коду сторінки кандидата candidate_js.js.

Цей файл являє собою клієнтську частину веб-додатка, написану мовою JavaScript. У ньому міститься кілька функцій, які виконують різні операції із сервером, використовуючи бібліотеку axios для надсилання HTTP-запитів.

Основні функції та їхнє призначення:

- window.onload (рис. 2.31.): Ця функція буде виконана під час завантаження сторінки. Вона відправляє GET-запит на http://localhost:8080/me, передаючи в заголовку авторизаційний токен JWT (JSON Web Token) для аутентифікації користувача. Потім, у разі успішного виконання запиту, функція оновлює елементи на сторінці з «id» «email» і «role» значеннями, отриманими з відповіді.

```
window.onload = function () {
    axios
        .get("http://localhost:8080/me", {
            headers: {
                Authorization: `Bearer ${jwt}`,
            },
        })
        .then((response) => {
            document.getElementById("email").innerText = response.data.email;
            document.getElementById("role").innerText = response.data.role;
        })
        .catch((error) => {
            alert(error);
        });
    getCandidateSlots();
};
```

Рис. 2.31. Функція window.onload()

- getCandidateSlots() (рис. 2.32.): Ця функція надсилає GET-запит на http://localhost:8080/candidates/current/slots, також передаючи авторизаційний токен у заголовку. При отриманні відповіді, вона обробляє дані і створює HTML-елементи

для кожного слота в списку response.data.candidateSlotDtoList. Потім вона додає ці елементи в батьківський елемент з id "slots" на сторінці.

```
function getCandidateSlots() {
    axios
        .get("http://localhost:8080/candidates/current/slots", {
            headers: {
                Authorization: `Bearer ${jwt}`,
            },
        })
        .then((response) => {
            const slotsDiv = document.getElementById("slots");
            slotsDiv.innerHTML = "";
            response.data.candidateSlotDtoList.forEach(function (slot) {
                const slotDiv = document.createElement("div");
                slotDiv.className = "slot";
                slotDiv.innerHTML = `
                <div class="id-slot">${slot.id}</div>
                <div class="date">${slot.date}</div>
                <div class="from">${slot.from}</div>
                <div class="to">${slot.to}</div>
                `;
                slotsDiv.appendChild(slotDiv);
            });
        })
        .catch((error) => {
            alert(error);
        });
}
```

Рис. 2.32. Функція getCandidateSlots()

- updateSlots(): Ця функція викликає getCandidateSlots() для оновлення списку слотів на сторінці.
- showCreateForm() (рис. 2.33.): Ця функція відображає форму створення нового слота, встановлюючи властивість CSS display елемента з id "create-form" у значення "block".
- hideCreateForm() (рис. 2.33.): Ця функція приховує форму створення нового слота, встановлюючи властивість CSS display елемента з id "create-form" у значення "none".

```
function showCreateForm() {
    document.getElementById("create-form").style.display = "block";
}

function hideCreateForm() {
    document.getElementById("create-form").style.display = "none";
}
```

Рис. 2.33. Функції відкриття та закриття форм створення слотів.

- createSlot() (рис. 2.34.): Ця функція збирає дані з полів форми створення слота (дата, час початку і закінчення) і відправляє POST-запит на http://localhost:8080/candidates/current/slots, передаючи дані в тілі запиту і авторизаційний токен в заголовку. У разі успішного виконання запиту, функція виводить повідомлення про успішне створення слота, викликає getCandidateSlots() для оновлення списку і приховує форму створення.

```
function createSlot() {
    const date = document.getElementById("date").value;
    const from = document.getElementById("from").value;
    const to = document.getElementById("to").value;

    if (!date || !from || !to) {
        alert("Please fill out all fields");
        return;
    }

    axios
        .post("http://localhost:8080/candidates/current/slots", {
            date: date,
            from: from,
            to: to,
        }, {
            headers: {
                Authorization: `Bearer ${jwt}`,
            },
        })
        .then((response) => {
            alert("Slot created successfully");
            getCandidateSlots()
            hideCreateForm();
        })
        .catch((error) => {
            const response = error.response.data;
            alert(response.errorMessage);
        });
}
```

Рис. 2.34. Функція createSlot()

- showEditForm(): Ця функція відображає форму редагування слота, встановлюючи властивість CSS display елемента з id "edit-form" у значення "block".
- hideEditForm(): Ця функція приховує форму редагування слота, встановлюючи властивість CSS display елемента з id "edit-form" у значення "none".
- editSlot() (рис. 2.35.): Ця функція аналогічна createSlot(), але надсилає POST-запит на http://localhost:8080/candidates/current/slots/${slotId}, де ${slotId} - ідентифікатор редагованого слота. Вона оновлює дані слота з новими значеннями дати і часу, виводить повідомлення про успішне оновлення, викликає getCandidateSlots() і приховує форму редагування.

```
function editSlot() {
    const date = document.getElementById("edit-date").value;
    const from = document.getElementById("edit-from").value;
    const to = document.getElementById("edit-to").value;
    const slotId = document.getElementById("slot-id").value;

    if (!date || !from || !to || !slotId) {
        alert("Please fill out all fields");
        return;
    }

    axios
        .post(`http://localhost:8080/candidates/current/slots/${slotId}`, {
            date: date,
            from: from,
            to: to,
        }, {
            headers: {
                Authorization: `Bearer ${jwt}`,
            },
        })
        .then((response) => {
            alert("Slot updated successfully");
            getCandidateSlots();
            hideEditForm();
        })
        .catch((error) => {
            const response = error.response.data;
            alert(response.errorMessage);
        });
}
```

Рис. 2.35. Функція editSlot()

Файл використовує бібліотеку axios для виконання асинхронних HTTP-запитів до сервера, а також використовує DOM для маніпуляції елементами на сторінці та оновлення їхнього вмісту. Файл також залежить від змінної jwt, яка містить JWT-токен, що використовується для аутентифікації користувача.

Також додано стилі до нашої сторінки таким чином, що результат нашої розробки ми можемо побачити на рис. 2.36. та рис. 2.37.

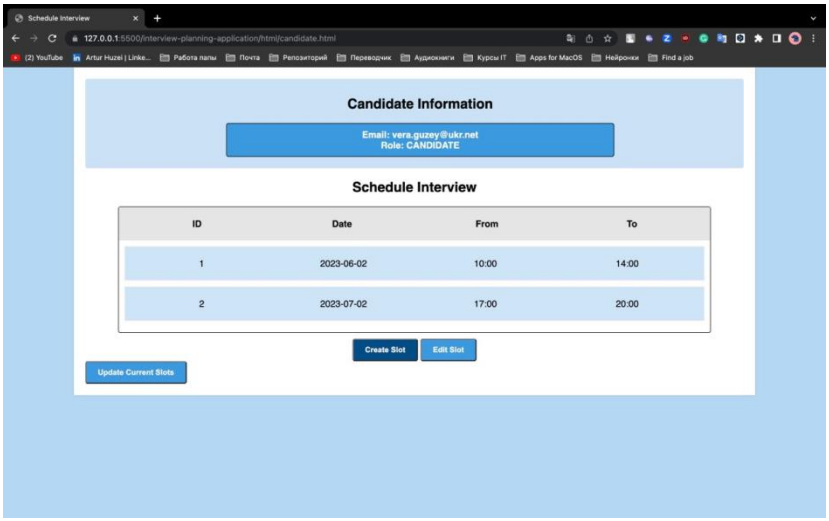


Рис. 2.36. Сторінка кандидата (1)

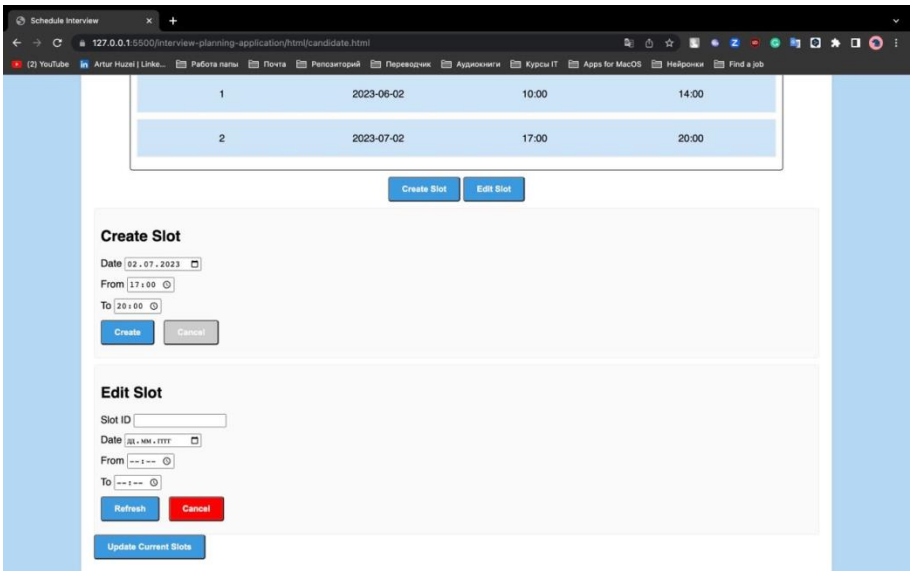


Рис. 2.37. Сторінка кандидата (2)

За такими самими принципами було реалізовано сторінки для інтерв’юера та координатора відповідно до логіки цих ролей та їх запитів на сервер та отримано наступні результати сторінки інтерв’юера (рис. 2.38. та рис. 2.39.) та сторінки координатора (рис. 2.40., 2.41 та 2.42.).

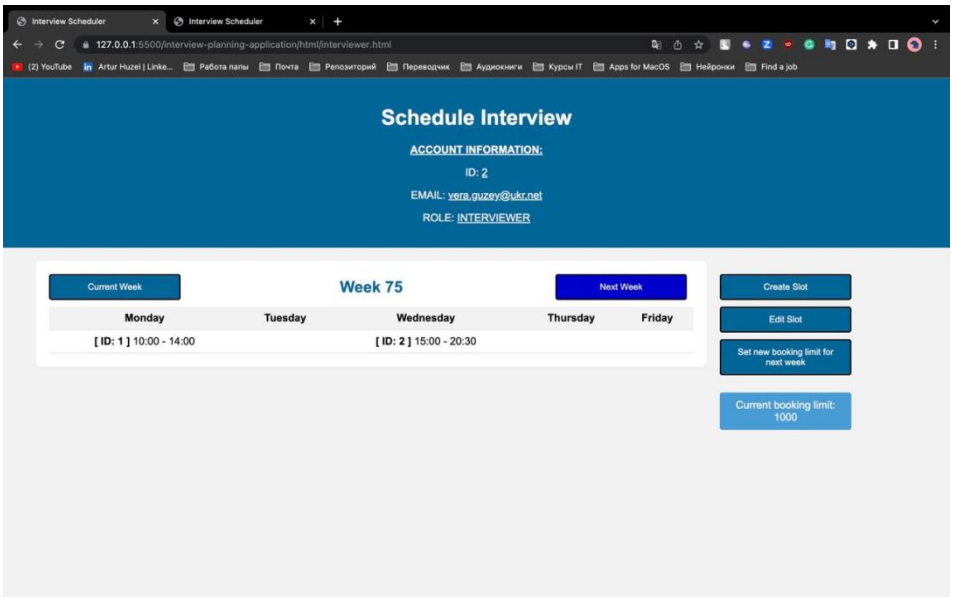


Рис. 2.38. Сторінка інтерв’юера (1)

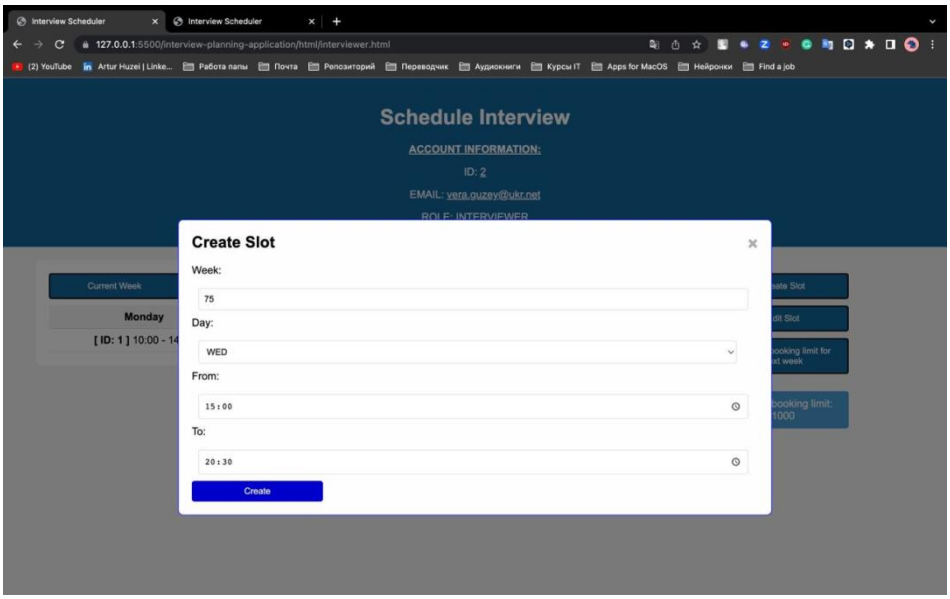


Рис. 2.39. Сторінка інтерв’юера (2)

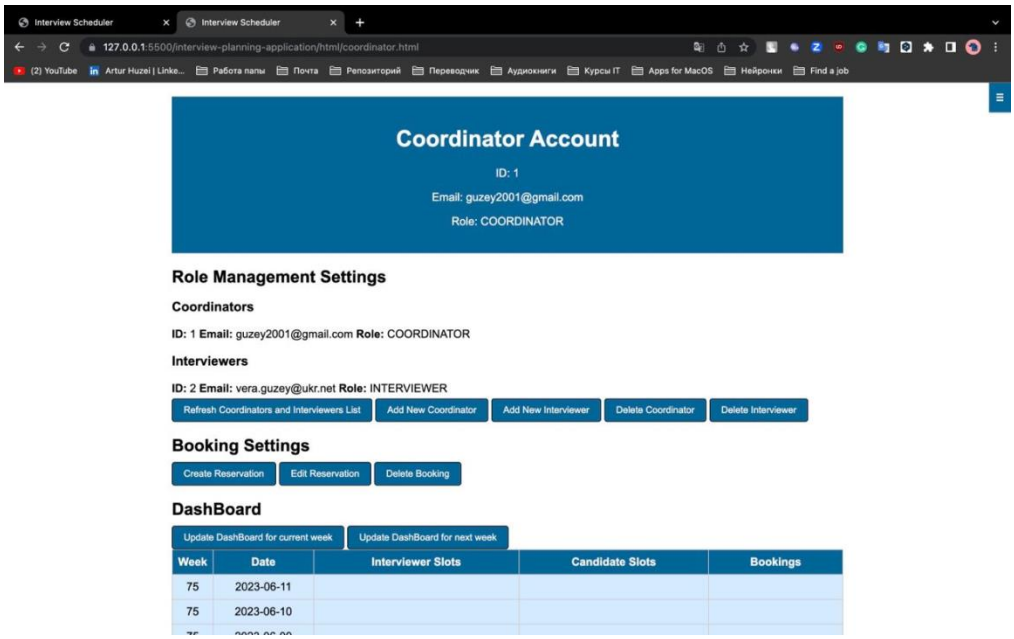


Рис. 2.40. Сторінка координатора (1)

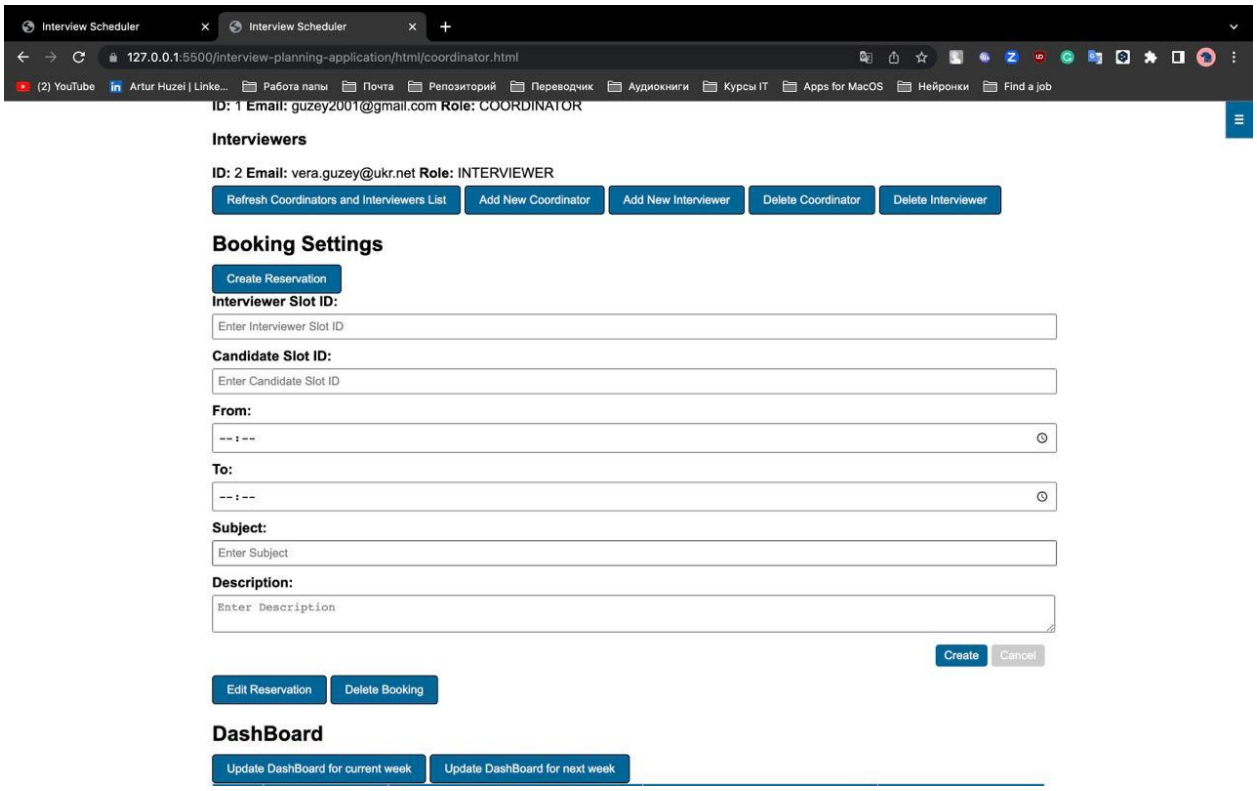


Рис. 2.41. Сторінка координатора (2)

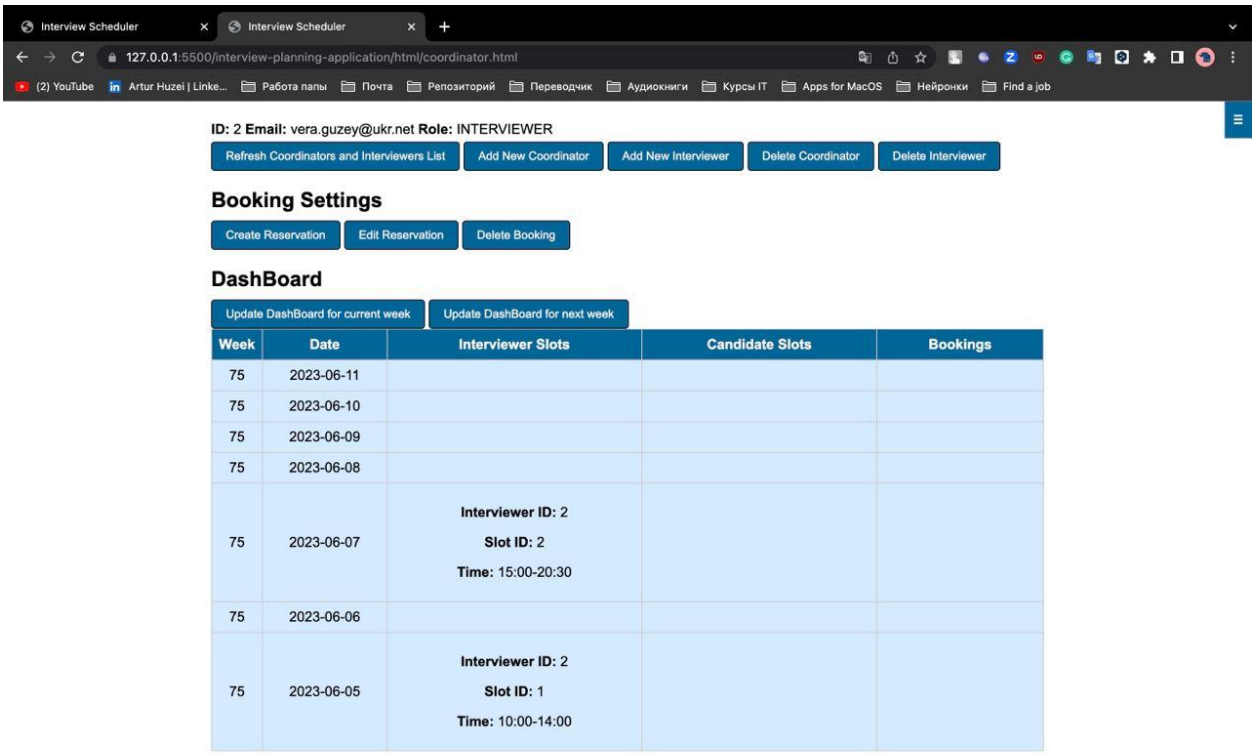


Рис. 2.42. Сторінка координатора (3)

2.6. Висновок до розділу 2

У цьому розділі було розглянуто основні аспекти розробки системи для планування співбесід в IT-компаніях. Ми вивчили комунікацію між клієнтом і сервером системи, визначили правила і бізнес-логіку системи, а також обговорили список необхідних HTTP-запитів. Потім ми перейшли до розробки серверної та клієнтської частин системи, включно зі структурою файлів, контролерами, безпекою, базою даних і бізнес-логікою.

Основною метою розробки системи було створення інструменту, який допоможе рекрутерам швидко й ефективно призначати співбесіди в компанії. Для досягнення цієї мети ми обрали сучасні технології та інструменти розробки, такі як Java, Spring Boot, Spring Data, Spring Security, Spring Web, Hibernate, PostgreSQL,

OAuth2 і Docker. Ці технології забезпечують високу продуктивність, надійність і безпеку системи.

Під час розробки системи ми приділили особливу увагу комунікації між клієнтом і сервером. Ми використовували HTTP-протокол для передачі даних і визначили необхідні HTTP-запити. Були розроблені запити для авторизації та автентифікації, запити, доступні для будь-якого користувача, а також запити, специфічні для кожної ролі в системі: кандидата, інтерв'юера та координатора. Це забезпечує гнучкість і функціональність системи, даючи змогу кожній ролі виконувати свої завдання.

Визначення правил і бізнес-логіки системи було суттєвим кроком у розробці. При розробці було враховано основні вимоги та потреби рекрутерів, а також надали гнучкі налаштування, як обмеження на кількість інтерв'ю, які може провести інтерв'юер за тиждень. Це дає змогу оптимізувати процес планування та розподілу співбесід у компанії.

Розробка серверної частини системи включала створення загальної структури застосунку, файлової структури, контролерів, безпеки та бази даних. Ми використовували Spring Boot і Spring Data для спрощення розробки та інтеграції цих компонентів. Було здійснено зв'язок із базою даних PostgreSQL, і розроблено репозиторії для доступу до даних. Також було розроблено бізнес-логіку системи, яка забезпечує основний функціонал системи та виконання бізнес-правил.

Розробка клієнтської частини системи включала створення файлової структури і розробку головної сторінки, а також сторінок для кожної ролі системи. Ми використовували HTML, CSS і JavaScript для створення клієнтського інтерфейсу, який забезпечує зручну взаємодію користувачів із системою. Головна сторінка надає зручну аутентифікацію у систему, а сторінки для кожної ролі надають відповідні функції та можливості.

У результаті нашої роботи ми розробили систему для планування співбесід в IT-компаніях, яка допомагає рекрутерам швидко та ефективно призначати співбесіди. Ми обрали сучасні технології розробки, визначили правила та бізнес-логіку системи, розробили необхідні HTTP-запити, створили серверну та клієнтську

частини системи. Отримана система має високу продуктивність, надійність і безпеку, а також надає зручний інтерфейс для користувачів.

РОЗДІЛ 3
ІНСТРУКЦІЯ ПО НАЛАШЛУВАННЮ СИСТЕМИ

Отже, ми маємо готову систему для планування співбесід у IT-компаніях, але для того щоб почати користуватися нею, треба зробити певні налаштування системи, адже система має певні індивідуальні параметри, які у кожної компанії, яка буде користуватися цією системою, будуть різні. Наприклад, такі як параметри користувача бази даних, порт, на якому буде розгортатися наш додаток або ж ключ додатку для користування входу через Facebook та інше.

Клієнтська частина нашої системи не підлягає налаштуванню з таких причин:

1. У нашій системі клієнтська частина реалізована як приклад використання серверного додатку, яка водночас може не підлягати побажанням кінцевої компанії, яка буде користуватися нею. Ці побажання можуть бути такі як: відсутність бажаного дизайну сайту, стилізації під власний бренд компанії, інша структура відображення та надання даних системи, представлення клієнтської частини не у якості сайту, а, наприклад, у якості мобільного додатку та інше.

2. Клієнтська частина не вимагає певних налаштувань, крім того як налаштувати розміщення розробленої нами системи у відкритий доступ мережі інтернет, наприклад, за допомогою Google Cloud або Amazon Web Services.

У той же час, серверна частина повинна бути ретельно налаштована перед її використанням за інструкціями, які будуть наведені у цьому розділі.

3.1. Створення додатка на платформі Facebook for developers

У цілях безпеки Facebook API не дозволяє будь-кому звертатися до їх системи, а тим паче отримувати конфіденційну інформацію облікового запису користувачів соціальної мережі, як у нашому випадку, електрону адресу користувача.

Тому, розглянемо покрокову інструкцію зі створення додатка на платформі Facebook for Developers для інтеграції його в нашу систему для планування співбесід в IT-компаніях. Це дасть нам змогу використовувати Facebook API для входу та авторизації користувачів у нашій системі. Крім того, ми також налаштуємо доступ до електронної пошти користувачів, оскільки вона використовується у нашій системі.

Крок 1: Створення програми на платформі Facebook for Developers.

1. Перейдіть на веб-сайт Facebook for Developers: https://developers.facebook.com/.
2. Увійдіть у свій акаунт Facebook або зареєструйтеся, якщо у вас ще немає акаунта.
3. Перейдіть у "Мої додатки" (My Apps) і натисніть на кнопку "Створити додаток" (Create App).
4. Введіть ім'я додатка та виберіть категорію. Натисніть на кнопку "Створити ідентифікатор додатка" (Create App ID).
5. Пройдіть перевірку безпеки (Security Check) і заповніть необхідну інформацію про застосунок.
6. Натисніть на кнопку "Зберегти зміни" (Save Changes).

Крок 2: Налаштування основних параметрів програми.

1. У лівій панелі виберіть "Налаштування" (Settings).
2. У розділі "Основні" (Basic) заповніть такі поля:

- Домени додатка (App Domains): Вкажіть домен вашого додатка, наприклад, "interview-planning-aplication.com".
- Адреси електронної пошти (Email): Вкажіть адресу електронної пошти для отримання повідомлень про ваш додаток.

- Політика конфіденційності (Privacy Policy URL): Вкажіть URL-адресу сторінки з політикою конфіденційності вашого додатка.
- Умови використання (Terms of Service URL): Вкажіть URL-адресу сторінки з умовами використання вашого додатка.

3. Натисніть на кнопку "Зберегти зміни" (Save Changes).

Крок 3: Налаштування функцій програми та доступу до електронної пошти.

1. У лівій панелі виберіть "Продукти" (Products).
2. У розділі "Вхід через Facebook" (Facebook Login) натисніть на кнопку "Налаштування" (Set Up).
3. У розділі "Налаштування OAuth" (OAuth Settings) заповніть такі поля:

- Довірена URL-адреса перенаправлення OAuth (Valid OAuth Redirect URIs): Вкажіть URL-адресу, на яку користувачі будуть перенаправлені після авторизації через Facebook.
- Довірена URL-адреса скасування OAuth (Deauthorize Callback URL): Вкажіть URL-адресу для обробки події скасування авторизації користувача.

4. У розділі "Запитувані дозволи" (Permissions) виберіть необхідні дозволи для вашого додатка, включно з доступом до електронної пошти (email).
5. Натисніть на кнопку "Зберегти зміни" (Save Changes).

Крок 4: Отримання ключів і секретів програми

1. У лівій панелі виберіть "Налаштування" (Settings).
2. У розділі "Основні" (Basic) ви знайдете таку інформацію, яка нам знадобиться для розробки авторизації та аутентифікації в нашій системі:

- Ідентифікатор застосунку (App ID): скопіюйте цей ідентифікатор, він буде використовуватися в подальших налаштуваннях нашого серверного застосунку.
- Секрет програми (App Secret): натисніть на кнопку "Показати" (Show) поруч із секретом програми, щоб відобразити його. Скопіюйте секрет, він також буде використовуватися в подальших налаштуваннях серверного додатка.

Отже, ми успішно створили додаток на платформі Facebook for Developers і налаштували його для інтеграції в нашу систему. Тепер ми можемо використовувати значення ідентифікатора застосунку та секрету застосунку, отримані на кроці 4, для

розробки авторизації та автентифікації в нашій системі. Зверніть увагу, що доступ до електронної пошти користувачів має бути надано з використанням запитуваних дозволів у налаштуваннях додатка, щоб ми могли отримувати електронну пошту користувачів і використовувати її в нашій системі.

3.2. Інтеграція Docker та налаштування змінних оточення

Розглянемо покрокову інструкцію з інтеграції Docker із серверною частиною системи та налаштування необхідних змінних оточення. Інтеграція Docker забезпечує зручне розгортання та керування мікросервісним додатком, а використання змінних оточення дає змогу налаштувати його роботу в різних оточеннях.

Крок 1: Встановлення Docker

1. Завантажте та встановіть Docker з офіційного сайту: https://www.docker.com/get-started.

2. Перевірте успішне встановлення Docker, виконайте команду «docker –version» в командному рядку. У вас має бути видно версію встановленого Docker.

Крок 2: Налаштування Dockerfile

1. Створіть файл із назвою "Dockerfile" у кореневій папці серверної частини системи.

2. Відкрийте файл Dockerfile у текстовому редакторі та додайте такий код (рис. 3.1.):

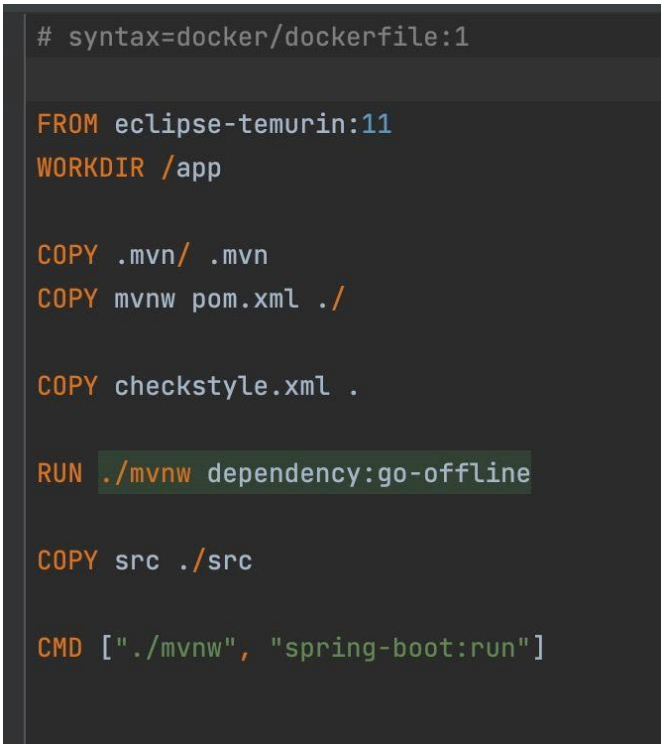


Рис. 3.1. Файл Dockerfile

3. Збережіть файл Dockerfile.

Крок 3: Налаштування docker-compose.yml

1. Створіть файл із назвою "docker-compose.yml" у кореневій папці серверної частини системи.
2. Відкрийте файл docker-compose.yml у текстовому редакторі та додайте наступний код (рис. 3.2.):
3. Збережіть файл docker-compose.yml.

Рис. 3.2. Файл docker-compose.yml

Крок 4: Налаштування змінних оточення в api.env

1. Створіть файл із назвою "api.env" у кореневій папці серверної частини системи.
2. Відкрийте файл api.env у текстовому редакторі та заповніть необхідні змінні оточення відповідно до наступного списку:
   - APPLICATION_PORT - порт, на якому буде запущено додаток.
   - JWT_SECRET - визначає секретність роботи для призначення JWT.
   - JWT_VALIDITY - термін дії JWT у секундах.
   - JWT_CACHING - тривалість кешування JWT у секундах.

- FIRST_COORDINATOR_EMAIL - email першого координатора, прикріплений до акаунту facebook, який буде автоматично додано до БД.
- FACEBOOK_CLIENT_ID - ідентифікатор клієнта додатку, наданий facebook.
- FACEBOOK_SECRET - секрет додатку, наданий facebook.
- FACEBOOK_REDIRECT_URI - URI, на який ви будете перенаправлені після oauth2. Налаштовується додатком facebook.
- HIBERNATE_DDL_AUTO - сплячий режим запуску DDL:
    - validate: перевіряє схему, не вносить змін до бази даних.
    - update: оновлює схему.
    - create: створює схему, знищуючи попередні дані.
    - create-drop: вилучає схему, коли SessionFactory явно закривається, зазвичай при зупинці програми.
    - none: нічого не робить зі схемою, не вносить змін до бази даних.
- DATABASE_PORT - порт, на якому буде запущено базу даних.
- POSTGRES_USER - ім'я користувача postgresql за замовчуванням.
- POSTGRES_PASSWORD - пароль користувача postgresql за замовчуванням.
- POSTGRES_DB - ім'я бази даних postgresql.
- REDIS_HOSTNAME - ім'я хосту сервера для підключення redis DB.
- REDIS_PORT - порт вказаного хосту для підключення redis DB.

Приклад заповнення змінних оточення (рис 3.3.):

```
APPLICATION_PORT=8080
JWT_SECRET=your-jwt-secret
JWT_VALIDITY=3600
JWT_CACHING=300
FIRST_COORDINATOR_EMAIL=your-email@example.com
FACEBOOK_CLIENT_ID=your-facebook-client-id
FACEBOOK_SECRET=your-facebook-secret
FACEBOOK_REDIRECT_URI=your-facebook-redirect-uri
HIBERNATE_DDL_AUTO=create-drop
DATABASE_PORT=5432
POSTGRES_USER=your-postgres-user
POSTGRES_PASSWORD=your-postgres-password
POSTGRES_DB=your-postgres-database
REDIS_HOSTNAME=your-redis-hostname
REDIS_PORT=6379
```

Рис. 3.3. Приклад заповнення файлу api.env

3.3. Висновок до розділу 3

У цьому розділі було представлено інструкцію з налаштування системи для планування співбесід в IT-компаніях. Було розглянуто ключові кроки, необхідні для налаштування системи та її успішного функціонування.

Було наведено інструкцію про створення застосунку на платформі Facebook for developers, у якому було описано дії з інтеграції Facebook API у систему. Цей крок включав створення додатка на платформі Facebook for developers, отримання та налаштування необхідних ключів доступу, а також визначення дозволів, необхідних для аутентифікації та авторизації користувачів через Facebook.

Також було інструкцію "Інтеграція Docker і налаштування змінних оточення". У ньому було представлено інструкції з використання Docker для пакування та розгортання мікросервісного додатка. Було також розглянуто налаштування змінних оточення, що дає змогу системі працювати в різних середовищах.

Унаслідок налаштувань повністю реалізовано автентифікацію й авторизацію користувачів за допомогою Facebook API і OAuth2 та застосовано контейнеризацію з використанням Docker для зручного розгортання та масштабування системи.

ВИСНОВКИ

У результаті нашої роботи ми розробили систему для планування співбесід в IT-компаніях, використовуючи сучасні технології розробки.

У першому розділі було розглянуто вибір технологій для розробки системи планування співбесід в IT-компаніях. Ми провели аналіз різних технологій та інструментів і вибрали найбільш підходящі для нашої системи. Ми порівняли цю систему з іншими схожими системами й обґрунтували переваги обраних технологій. Обрані технології, такі як Java, Spring Boot, Spring Data, Spring Security, Spring Web, Hibernate, PostgreSQL, OAuth2 і Docker, забезпечують високу продуктивність, надійність і безпеку системи.

У другому розділі ми розглянули процес розробки системи планування співбесід в IT-компаніях. Ми детально описали проектування системи та її архітектуру. Було розроблено серверну та клієнтську частини системи, визначено файлову структуру, контролери, безпеку та базу даних. Ми приділили особливу увагу комунікації між клієнтом і сервером, визначили необхідні HTTP-запити та розробили бізнес-логіку системи. Результатом розробки є система, яка допомагає рекрутерам швидко та ефективно призначати співбесіди, забезпечуючи гнучкість і функціональність для кожної ролі в системі.

У третьому розділі було представлено інструкцію з налаштування системи для планування співбесід в IT-компаніях. Ми розглянули ключові кроки налаштування системи та її успішного функціонування. Було описано інтеграцію з Facebook API та використання Docker для контейнеризації та розгортання мікросервісного застосунку. Завдяки цим налаштуванням, система забезпечує автентифікацію та авторизацію користувачів за допомогою Facebook API та OAuth2, а також гнучке розгортання та масштабування за допомогою Docker.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ТА ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spring Boot. [Electronic resource]. Access mode: https://spring.io/projects/spring-boot (lastaccess: 28.04.2023). – Title from the screen;
2. Spring Data. [Electronic resource]. Access mode: https://spring.io/projects/spring-data (lastaccess: 29.04.2023). – Title from the screen;
3. Spring Security. [Electronic resource]. Access mode: https://spring.io/projects/spring-security (lastaccess: 01.05.2023). – Title from the screen;
4. Spring Web. [Electronic resource]. Access mode: https://spring.io/projects/spring-web (lastaccess: 01.05.2023). – Title from the screen;
5. Hibernate. [Electronic resource]. Access mode: https://hibernate.org/ (lastaccess: 04.05.2023). – Title from the screen;
6. PostgreSQL. [Electronic resource]. Access mode: https://www.postgresql.org/ (lastaccess: 04.05.2023). – Title from the screen;
7. OAuth2. [Electronic resource]. Access mode: https://oauth.net/2/ (lastaccess: 04.05.2023). – Title from the screen;
8. Facebook API. [Electronic resource]. Access mode: https://developers.facebook.com/docs/graph-api (lastaccess: 04.05.2023). – Title from the screen;
9. Postman. [Electronic resource]. Access mode: https://www.postman.com/ (lastaccess: 04.05.2023). – Title from the screen;
10. Docker. [Electronic resource]. Access mode: https://www.docker.com/ (lastaccess: 05.05.2023). – Title from the screen;
11. HTML. [Electronic resource]. Access mode: https://developer.mozilla.org/en-US/docs/Web/HTML (lastaccess: 05.05.2023). – Title from the screen;
12. CSS. [Electronic resource]. Access mode: https://developer.mozilla.org/en-US/docs/Web/CSS (lastaccess: 07.05.2023). – Title from the screen;

13. JavaScript. [Electronic resource]. Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript (lastaccess: 07.05.2023). – Title from the screen;

14. Axios Library. [Electronic resource]. Access mode: https://github.com/axios/axios/ (lastaccess: 07.05.2023). – Title from the screen;

15. Гузей А.Е. Система для призначення співбесід у IT- компаніях / Гузей А.Е., Холявкіна Т.В.// Проблеми інформатизації та управління: зб. наук. праць. К.: НАУ, 2022. №4(72). С. 63-67.