Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Полтавця Владислава Олександровича*
(ПІБ)

академічної групи *122-19-4*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-додатку для coworking організації з використанням фреймворку Spring*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-4* (група) *Полтавець В.О.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку для coworking організації з використанням фреймворку Spring*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Спірінцев В.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Полтавець В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: с. 91, рис. 42, 3 дод., 21 джерел.

Об'єкт розробки: веб-додаток для перегляду та бронювання робочих місць для coworking організації.

Мета кваліфікаційної роботи: створення веб-додатку з використанням мови програмування Java та фреймворку Spring, який забезпечує необхідний функціональний потенціал для виконання поставлених задач.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняються постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробку веб-орієнтованої інформаційної системи, описана робота системи, алгоритм і структура його функціонування, а також виклик та завантаження додатку, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню додатку та розраховано час на його створення.

Практичне значення полягає у проектуванні та створені веб-додатку з використанням сучасних методів розробки та технологій, що дозволить користувачу отримувати необхідну інформацію про робочі місця, котрі надає coworking організація та оформлювати замовлення на бронювання обраних робочих місць.

Актуальність розробленого веб-додатку визначається великим інтересом суспільства до популярної та постійно зростаючої сфери coworking. Coworking організації надають можливість працівникам ділити робочий простір, надаючи в оренду робочі місця.

Список ключових слів: COWORKING ОРГАНІЗАЦІЯ, ВЕБ-ДОДАТОК, H2 БАЗА ДАНИХ, JAVA, SPRING, THYMELEAF, HTML, CSS, BOOTSTRAP.

ABSTRACT

Explanatory note: pp. 91, fig. 42, 3 extra, 21 sources.

The object of development: web application for viewing and booking work places for coworking organization.

The purpose of the diploma project is creation of a web application with using Java programming language and Spring framework that gives necessary functionalities for achieving tasks.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, substantiates the relevance of the topic and clarifies the formulation of the problem.

In the first chapter, the subject area is analyzed, the relevance of the task and the purpose of development are determined, the statement of the problem is formulated, the requirements for software implementation, technologies and software are indicated.

In the second section, the available solutions are analyzed, a platform for development is selected, the design and development of a web-oriented information system is carried out, the operation of the system, the algorithm and structure of its functioning, as well as the call and loading of the application are described, the input and output data are determined, the composition of the parameters of the technical means.

In the economic section, the labor intensity of the developed information system is determined, the cost of work on creating an application is calculated and the time for its creation is calculated.

The practical value is to design and create web application using modern development practices and technologies, which provide the necessary information about work places, in coworking organization and give an opportunity to book these places to user.

The relevance of the development web application is determined by great interest of society in the popular and continuously growing coworking sphere. Coworking organizations provide opportunity to workers to share working space with other workers by giving a chance to lease work places.

Keywords List: COWORKING ORGANIZATION, WEB APPLICATION, H2 DATA BASE, JAVA, SPRING, THYMELEAF, HTML, CSS, BOOTSTRAP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JRE – Java Runtime Environment;

JVM – Java Virtual Machine;

IoC – Inversion of Control;

DI – Dependency Injection;

HTTP – HyperText Transfer Protocol;

MVC – Model View Controller;

JPA – Java Persistenr API;

ORM – Object Relational Mapping;

HTML – Hyper Text Markup Language;

CSS – Cascade Style Sheets;

БД – База даних;

СУБД – Система управління базою даних;

SQL – Structure Query Language;

ОС – Операційна система.

ВСТУП

Розроблена інформаційна система призначена для застосування в сфері коворкінгу.

Коворкінг - це спільний робочий простір, який надається організаціями для найманих працівників, фрілансерів та підприємців. Він створює сприятливе середовище для співпраці, обміну ідеями та розвитку бізнесу.

У сфері коворкінгу включається інформаційна система, яка допомагає керувати робочими просторами, розподіляти ресурси, вести облік клієнтів та забезпечувати комунікацію між учасниками спільноти.

Важливою складовою коворкінгу є належна організація робочих просторів, які повинні бути комфортними та функціональними. Крім того, необхідно забезпечити ефективне використання ресурсів, таких як робочі місця, конференц-зали, кухні тощо. Аналіз потреб користувачів дозволяє визначити вимоги до інформаційної системи, такі як резервування робочих місць, календарний планувальник подій, облік оплати послуг та звітність.

Метою даної роботи є розробка системи електронного документування та обліку робочих просторів у коворкінгу.

Для досягнення поставленої мети необхідно вирішити основні завдання:

– аналіз організації коворкінгу та його потреб у робочих просторах;

– уточнення вимог до процесу резервування та обліку робочих місць;

– розробка алгоритмів, бази даних і реалізації програмного забезпечення. Відповідно до проведеного аналізу поставлені основні функціональні задачі перед системою;

– забезпечення зручного та швидкого резервування робочих місць.

Усе це допомагає покращити роботу коворкінгу, забезпечити комфортне середовище для праці та підтримувати продуктивність користувачів. Розроблена інформаційна система може бути використана різними коворкінгами та сприяти їх ефективному управлінню та розвитку.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної області

Основним завданням сучасних coworking організацій, або просто коворкінгів, є гнучка організація робочого простору і прагнення до формування спільноти резидентів та внутрішньої культури. На відміну від роботи вдома, учасники залишаються незалежними та вільними, мають можливість спілкуватися, обмінюватися ідеями та допомагати один одному.

Для початку слід розібратися з найголовнішим терміном у цій сфері, а саме що ж таке коворкінг. Коворкінг (від англ. Co-working – спільна праця / спільно працювати) - це варіант організації робочого простору, за якого працівники збираються у нейтральному середовищі для самостійної роботи або в групах для роботи над спільними проєктами. Так, в одному середовищі працюють пліч-о-пліч фрілансери, стартапери і навіть співробітники невеликих компаній, яким невигідно знімати цілий офіс.

Ідея створити подібний робочий простір, виникла приблизно одночасно з появою фрілансу. Головним недоліком роботи фрілансерів є ізоляція. Люди працюють на дому, тим самим вони частково ізольовані від інших, від колективу.

Головною метою co-working-організацій було вирішити цю проблему самотності, створити колектив, щоб кожен працівник виконував свої завдання, але водночас був частиною колективу.

Зазвичай коворкінг — це приміщення досить великих розмірів (рис. 1.1), в якому мають змогу комфортно розміститися кілька десятків людей з ноутбуками, декілька кімнат для переговорів і навіть окремі офіси.

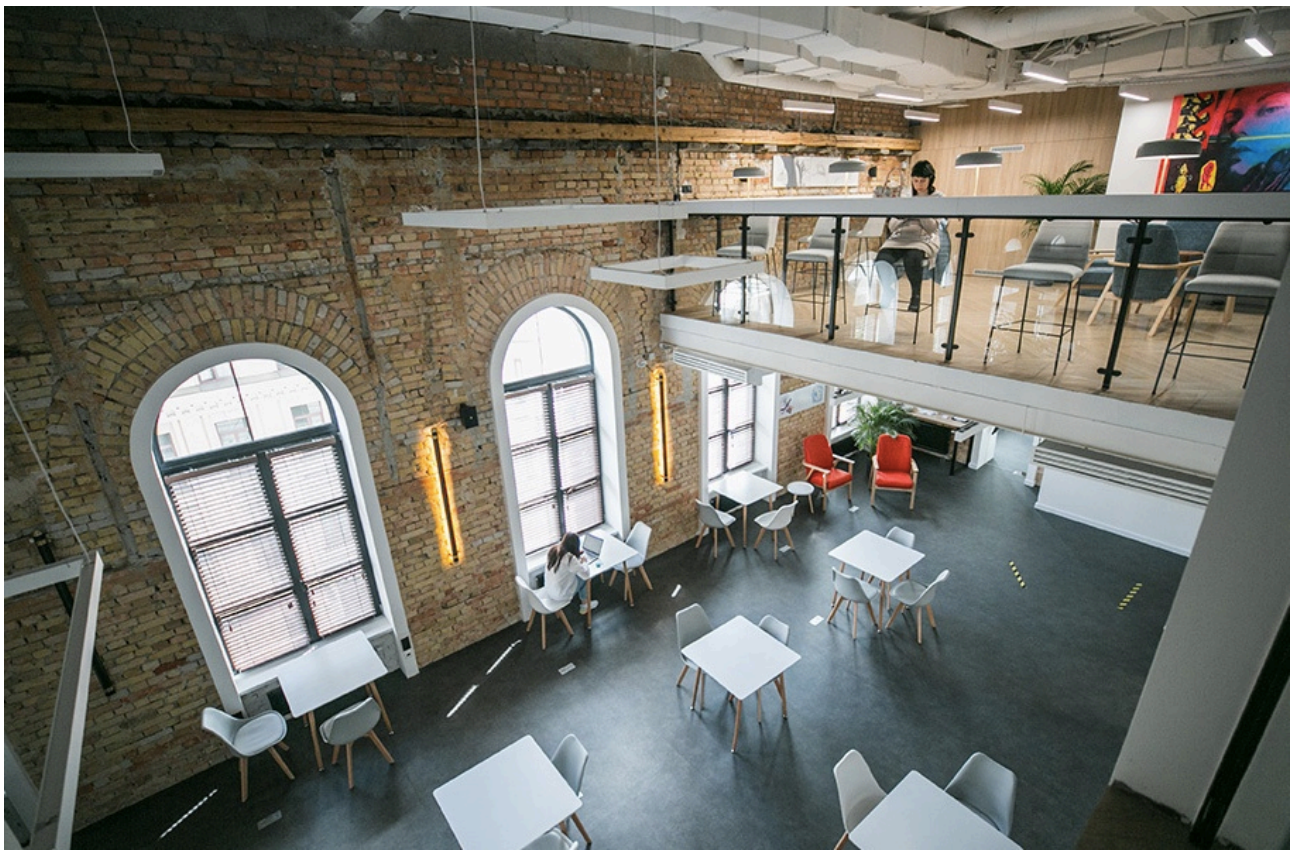


Рис. 1.1. Приклад приміщень, які зазвичай використовують co-working організації

У більшості випадків коворкінг-центри мають стандартний набір умов та зручностей. Серед них можна виділити основні: меблі, доступ до Wi-Fi, принтерів та іншої техніки, можливість випити каву тощо. А деякі додатково мають зони відпочинку, кімнати для медитацій чи навіть спортивні зали. При цьому, важливою особливістю є гнучкість, адже не має необхідності підписувати договір про довгострокову оренду — достатньо обрати тариф (денний чи місячний), який підходить конкретному відвідувачу.

Серед основних форматів роботи у коворкінгах, можна виділити наступні:

– незакріплене місце у open space: тут відвідувачі коворкінгу займають вподобані столи без попереднього бронювання. Відповідно, доведеться прибирати своє робоче місце вкінці дня, а ноутбук, блокнот та інші речі зберігати вдома або у шафках для зберігання речей;

– фіксоване робоче місце: за такої моделі користування окремим столом та тумбою закріплюється за клієнтом на узгоджений термін. Якщо орендар робочого місця має невелику кількість речей для роботи (документи, техніка тощо), він зможе залишати їх тут;

– приватні офіси: це окремий простір для цілої компанії, від невеликої кімнати до цілого поверху. Фактично він ідентичний традиційним офісам, окрім того, що всіма питаннями з організації роботи (наявність меблів та техніки, підключення до інтернету, оплата комунальних послуг, тощо) займається адміністрація коворкінг-центру;

– індивідуальний варіант організації роботи: даний тип ще гнучкіший, ніж інші форми коворкінгу, і може включати, наприклад, змішаний формат, за якого команди частково працюють в окремому офісі, а решту часу — в open space. У відповідь на COVID-19 багато коворкінгових центрів розширили свої пропозиції, аби краще задовольнити потреби своїх клієнтів.

Основними клієнтами коворкінг-центрів є:

– фрілансери;

– стартапери;

– малі бізнеси;

– іноді й великі компанії.

1.2. Призначення розробки та галузь застосування

Як об'єкт впровадження розроблюваного веб-додатку для організації в сфері коворкінгу розглядається веб-сайт “Coworking365”[4], який реалізує усі необхідні функціональні можливості для відображення необхідної інформації про: переваги, котрі надає саме їх організація та тарифи для бронювання цих робочих місць.

Проведений аналіз існуючих технічних рішень сформував основні вимоги, щодо сучасного веб-додатку:

– доступність;

– адаптивність;

– кросплатформеність;

– високий рівень безпеки;

– відсутність необхідності завантаження додатку на пристрій користувача, та інші.

Аналізуючи вищезазначені вимоги, було зроблено висновок про те, що обов'язково буде реалізовано в проекті:

– головна сторінка додатку буде розроблена з максимально зручним та інтуїтивно зрозумілим інтерфейсом для користувача, що включатиме навігаційні елементи для переходу між сторінками (меню), текстову інформацію та допоміжні зображення;

– певний набір сторінок для відображення основної інформації про тарифи на оренду та робочі місця;

– спеціально відведену сторінку для профілю користувача, для відображення інформації про клієнта та його замовлень;

– задля захисту персональних даних, веб-додаток має використовувати засіб для шифрування паролей користувачів, що зменшить вірогідність несанкціонованих входів у профілі клієнтів;

– розподілення по ролях. Відповідно до тієї чи іншої ролі, користувачеві будуть надаватися додаткові дії, котрі він зможе виконувати на сайті;

– форми для входу та створення акаунту користувача, редагування інформації про користувача в профілі, створення запиту на бронювання;

– відведені сторінки для каталогу робочих місць, що містить фотографії та основну інформацію про робочі місця, рейтинг п’яти найпопулярніших робочих місць, статус усіх місць станом на зараз.

Головна сторінка першою постає перед користувачем, тому її дизайну та функціональності необхідно приділити чимало уваги, щоб надати змогу забезпечити користувача достатньою кількістю необхідної інформації.

На рис.1.2. зображено прототип головної сторінки за допомогою онлайн сервісу “Canva”.

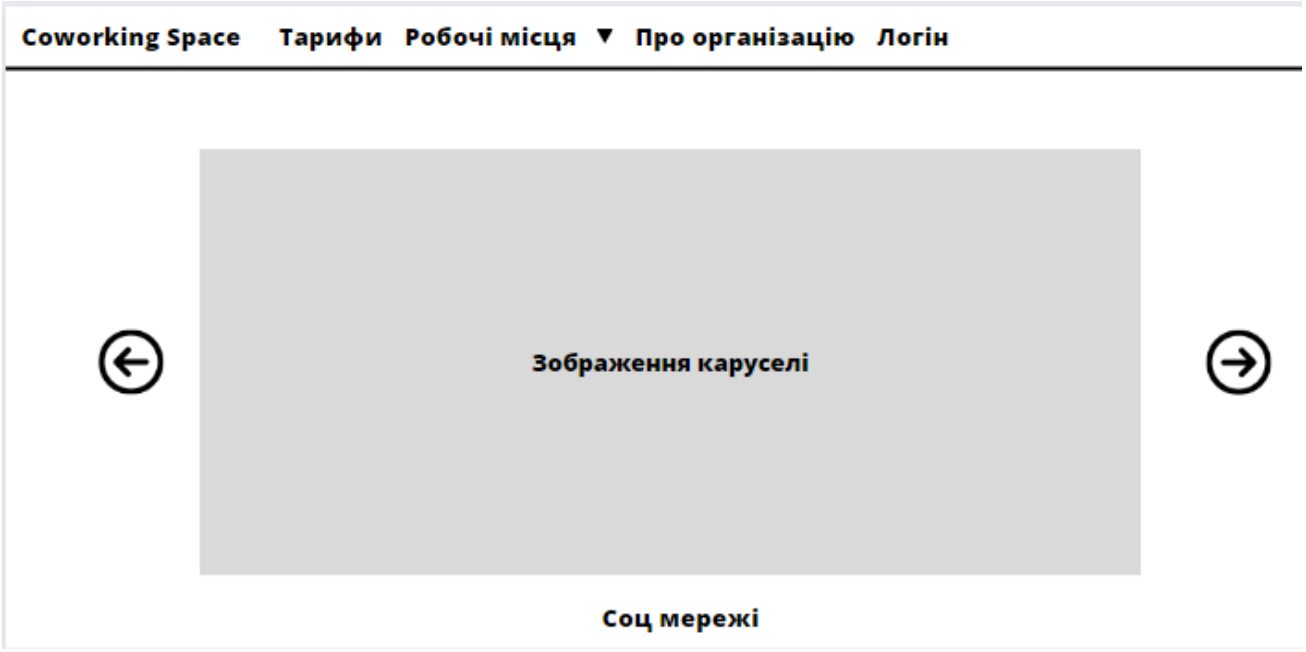


Рис. 1.2. Прототип головної сторінки сервісу

Головна сторінка містить:

– логотип, розроблений звичайною стилізацією тексту;
– система навігації (меню), яка складається з декількох пунктів;
– пункт меню, що відповідає за вхід до акаунту користувача;
– елемент “карусель” для відображення зображень;
– «підвал» або «футер».

Відповідно до прототипу головної сторінки було здійснено її верстку із застосуванням вже готових рішень з фреймворку Bootstrap, котрий використовує наступні мови:

– HTML5;
– CSS3;
– JavaScript.

Розроблена інформаційна система призначена для:

– отримання та відображення актуальних даних з бази даних у зрозумілому для користувача форматі;

– приймання та обробки замовлень на бронювання, що надійшли від користувачів веб-додатку;

– інформування клієнтів щодо успішної оплати та оформлення запитів на бронювання за допомогою електронних листів на поштову скриньку;

– збереження усієї необхідної інформації в базу даних.

1.3. Підстава для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

– ОПП за спеціальністю 122 «Комп'ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;

– завдання на кваліфікаційну роботу на тему: "Розробка веб-додатку для coworking організації з використанням фреймворку Spring ".

1.4. Постановка завдання

В даній кваліфікаційній роботі розглядається створення інформаційного веб-додатку для більш сучасного та комфортнішого відображення інформації та отримання замовлень від клієнтів коворкінгу.

Інформаційний веб-додаток створюється для надання послуг у сфері coworking. Аудиторію інформаційного сервісу становлять працівники будь-якого віку, статті, національності.

Інформаційна система повинна задовольняти наступним основним вимогам:

– інтуїтивно зрозумілий інтерфейс;

– висока швидкість завантаження сторінок;

– налагодженість логіки у алгоритмах роботи додатку;

– простота й повнота управління змістом;

– коректно побудована структура бази даних (БД).

Створення й розробка інформаційної системи повинно включати наступні етапи:

– ознайомитися з предметною областю завдання;

– провести порівняльну характеристику з аналогічними сервісами;

– належно спроектувати структуру та функціонал системи;

– налагодити алгоритми роботи системи;

– підібрати підходящі технології для розробки;

– написати програмний код згідно стандартів розробки.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої в роботі мети в інформаційній системі, що розробляється, повинні бути реалізовані:

– підтримка реєстрації та автентифікації для користувачів;

– можливість створення та збереження нових профілів до бази даних (БД);

– можливість перегляду та зміни контактних даних профіля користувача;

– отримання усієї необхідної інформації про тарифи на оренду та статус доступності робочих місць;

– інформування клієнтів про успішне створення профіля на сайті, успішну оплату бронювання та отримання підтвердження щодо підтвердження бронювання від адміністратора за допомогою електронних листів на поштову скриньку клієнта;

– зміна ролі користувача (з ролі User на Admin);

– доступ до консолі бази даних для користувача з роллю Developer;

– можливість веб-додатку шифрувати паролі задля забезпечення безпеки даних користувачів.

1.5.2. Вимоги до інформаційної безпеки

Організація та управління даними є важливою складовою безпеки. Завдання, пов'язані з організацією даних та контролем їх цілісності, мають бути вирішені на рівні додатків та баз даних.

Для безпечного використання та передачі даних розроблений додаток використовує наступні принципи:

– форми для логіну та реєстрації. Вони використовуються для того, щоб новий користувач не мав змоги зробити замовлення на бронювання;

– розподілення за ролями. Розподілення користувачів за ролями дозволяє надавати можливість виконувати додаткові дії у веб-додатку;

– використання csrf-токенів для захисту від csrf кібер-атак (CSRF (підроблення запитів з іншого сайту, cross-site request forgery) – це тип атаки на веб-сайт, при якому зловмисник, використовуючи шахрайський веб-сайт або скрипт, змушує браузер користувача виконувати дії від його імені на довіреному сайті).

– шифрування паролей. Для цього в проекті використовуєься BCryptPasswordEncoder. Шифрування паролей допомагає захистисти від несанкціонованого доступу до облікових записів користувачів у випадку, якщо база даних з паролями потрапить у руки зловмисників.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для ефективного функціонування веб-орієнтованої інформаційної системи, повинні виконуватися певні вимоги до технічних засобів.

Для клієнтської частини:

– процесор x64 з тактовою частотою не менш 1,4 ГГц;

– оперативної пам'яті 8 Гб;

Для серверної частини:

– процесор x64 з тактовою частотою 2,4 ГГц;

– оперативної пам'яті 32 Гб;

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для ефективного функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати веб-орієнтована підсистема, відповідало наступним вимогам:

– операційна система Unix, Linux, Microsoft Windows XP/7/8/10;

– браузер Інтернет (Microsoft Internet Explorer, MozillaFireFox, Opera, Google Chrome);

Веб-орієнтована інформаційна система має бути реалізована на мовах програмування Java з використанням фреймворку Spring та його компонентів, HTML5 і CSS3 для верстання веб-сторінок, бібліотеки jQuery для написання клієнтських сценаріїв, з використанням веб-серверу Apache TomCat та СУБД H2.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Результатом даної кваліфікаційної роботи має бути додаток, який виконує роль веб сервера з графічним інтерфейсом завдяки якому користувач може отримувати інформацію а також виконувати різні дії.

Основне призначення веб-додатку:

– отримання та відображення актуальних даних з бази даних у зрозумілому для користувача форматі;

– приймання та відображення замовлень на бронювання від користувачів;

– отримувати актуальні дані з бази даних (БД);

– відображувати дані з БД у зрозумілому для користувача форматі;

– приймати замовлення на бронювання;

– відображати замовлення для адміністраторів задля їх підтвердження;

– оброблювати замовлення;

– інформувати стосовно успішної реєстрації, оплати замовлення та підтвердження замовлення за допомогою електронних листів на пошту клієнта;

– збереження усіх замовлень у базі даних.

Для досягнення поставленої задачі додаток повинен вміти отримувати та оновлювати інформацію з використанням даних отриманих з бази даних, оброблювати великі об’єми даних за допомогою використання чітко налагоджених механізмів та алгоритмів обробки, а також видавати потрібну інформацію для користувача через графічний інтерфейс у зрозумілому вигляді.

2.2. Опис застосованих математичних методів

Під час проектування та розробки інформаційної системи математичні методи не використовувалися, були проведені лише базові математичні дії такі як додавання, віднімання та множення.

2.3. Опис використаних технологій та мов програмування

Дана комп'ютерна система розроблена за допомогою наступних інформаційних технологій:

– мова програмування Java – основна мова програмування, яка була використана для написання backend-частини додатку;

– фреймворк Spring – основний фреймворк для написання серверного коду;

– Lombok – зовнішня бібліотека, для скорочення шаблонного коду;

– HTML5 – мова, використана для створення каркасу сторінки;

– CSS3 – використана для стилізації сторінок написаних на HTML;

– Javascript – мова програмування, для створення сценаріїв для веб-сторінок;

– Bootstrap – front-end фреймворк для декорування сторінок;

– Thymeleaf – серверний двигун для об'єдання HTML сторінок з Java кодом;

– Apache Maven – використовувався для збірки проекту;

– Hibernate – ORM бібліотека для більш зручної роботи з базою даних;

– H2 – вбудована у проект база даних.

Java є високорівневою, основаною на класах, об'єктно-орієнтованою мовою програмування призначена для того, щоб мати якомога менше залежностей від реалізації. Ключові властивості даної мови програмування наведені нижче:

– об’єктно-орієнтована: Java – це об’єктно-орієнтована мова програмування, що означає що вона сфокусована на створенні багаторазових об’єктів, котрі містять у собі дані та дії;

– платформено-незалежність: програми, написанні на Java, можна запускати на будь-якій платформі, якщо вона має Java Virtual Machine (JVM). Ця властивість досягається за допомогою принципу «написано один раз, запускається усюди». Це означає, що скомпільований програмний код одного разу може бути запущений на різних операційних системах без необхідності компілювати його вдруге;

– синтаксис: синтаксис даної мови програмування схожий на інші C-подібні мови програмування такі як C++ та C#. Це означає, що програмний код виглядає структуровано та легкий для читання;

– керування пам’ятю: Java автоматично керує пам’ятю за допомогою збиральника сміття (Garbage collector). В такому випадку, розробникам не потрібно виділяти та вивільняти пам’ять вручну. Використання garbage collector’у зменшує шанси на витік пам’яті;

– бібліотеки та фреймворки: Java має доволі велику екосистему фреймворків та бібліотек, котрі забезпечують готові функціональні можливості, які роблять пришвидшують розробку та роблять її більш ефективною. Деякі популярні фреймворки: Spring, Hibernate, JavaFX та інші.

– компоненти платформи: Java забезпечує великий набір API та бібліотек для різноманітних цілей, таких як: робота у мережі, з’єднання з базами даних (JDBC), графічні інтерфейси користувача (Swing, JavaFX) та інші;

– багатопоточність: Java підтримує багатопоточність, що дозволяє розробникам створювати додатки, котрі можуть виконувати декілька завдань одночасно. Багатопоточність дозволяє збільшити продуктивність та реагування програми;

– популярність використання: Java широко використовується для розробки різного роду додатків. Типи додатків, які розробляються на Java:

корпоративне програмне забезпечення, веб-додатки, мобільні застосунки (під Android OS), десктопні додатки, наукові застосункита інші.

В цілому, Java – це універсальна та широко прийнята мова програмування, відома своєю незалежністю платформи, сильними об'єктно-орієнтованими можливостями та великою екосистемою бібліотек та фреймворків.

Згідно з рейтингами, розміщеними на форумі програмістів dou.ua[5], Java займає лідируючі позиції. Так, 14% програмістів використовують її для роботи (рис. 2.1).

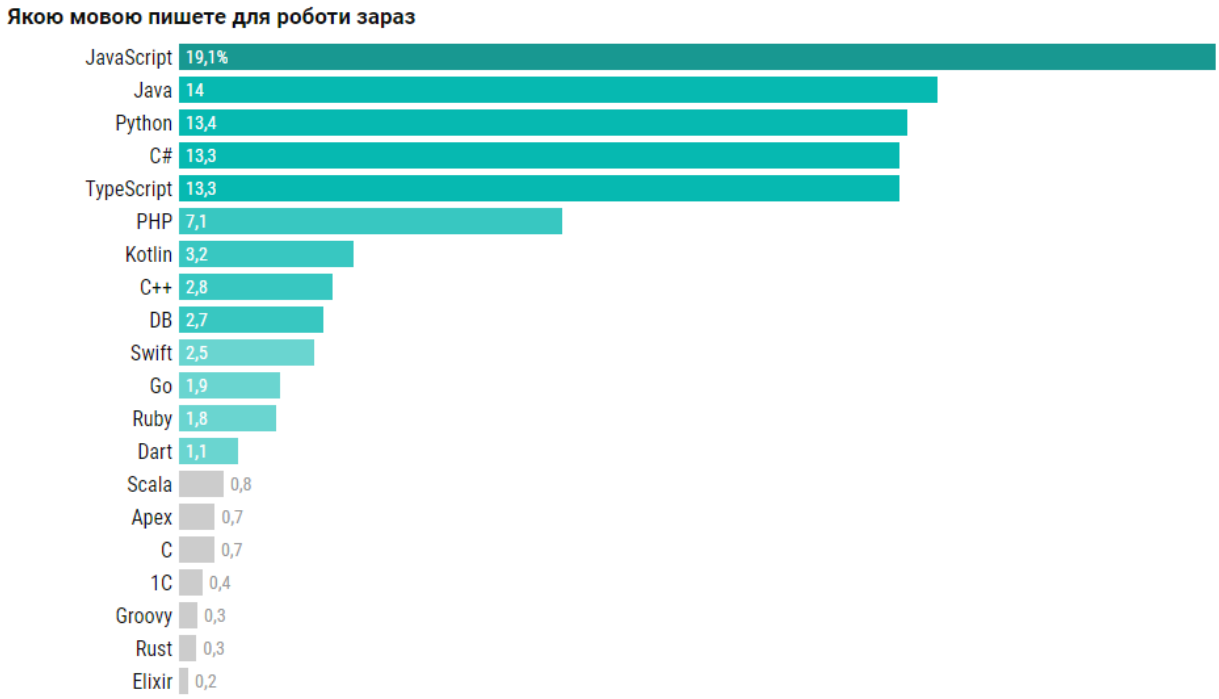


Рис. 2.1. Статистика використання мов програмування при роботі на початок 2023-го року

Java також є лідером у написанні backend-частини додатків (рис. 2.2). Дані також узяті з форуму для програмістів dou.ua [5].

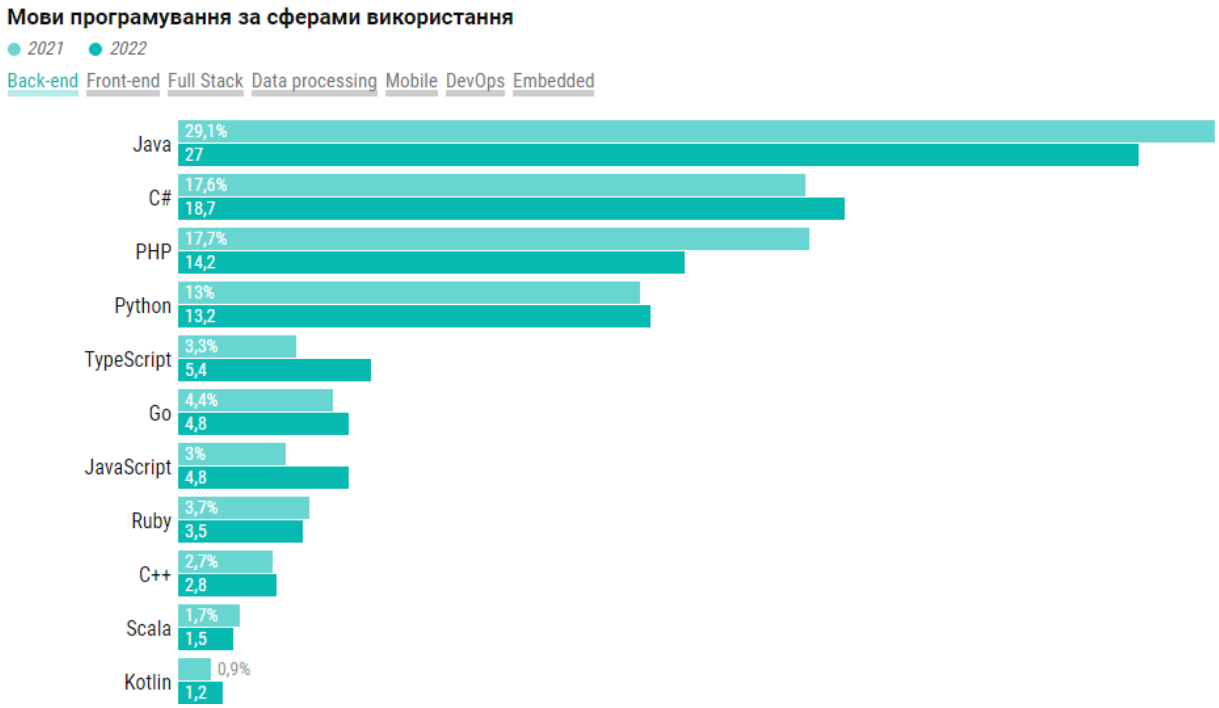


Рис. 2.2. Статистика використання мов програмування у backend-частині додатків на початок 2023-го року.

Spring Framework - платформа Java, яка забезпечує комплексну інфраструктурну підтримку розробки Java-додатків. Spring фреймворк обробляє інфраструктуру, щоб розробник мав змогу зосередитися на написанні програмного коду. Spring фреймворк складається з багатьох модулів (рис. 2.3), але ключовим є Spring Core Container. Даний фреймворк формується на двох основних принципах: Inversion of Control (IOC) та Dependency Injection (DI), які реалізуються у контейнері.

Основним елементом додатку є бін (англ. Bean). Контейнер керує життєвим циклом бінів у додатку, що дає йому змогу створювати, знищувати та виконувати інші дії з бінами.

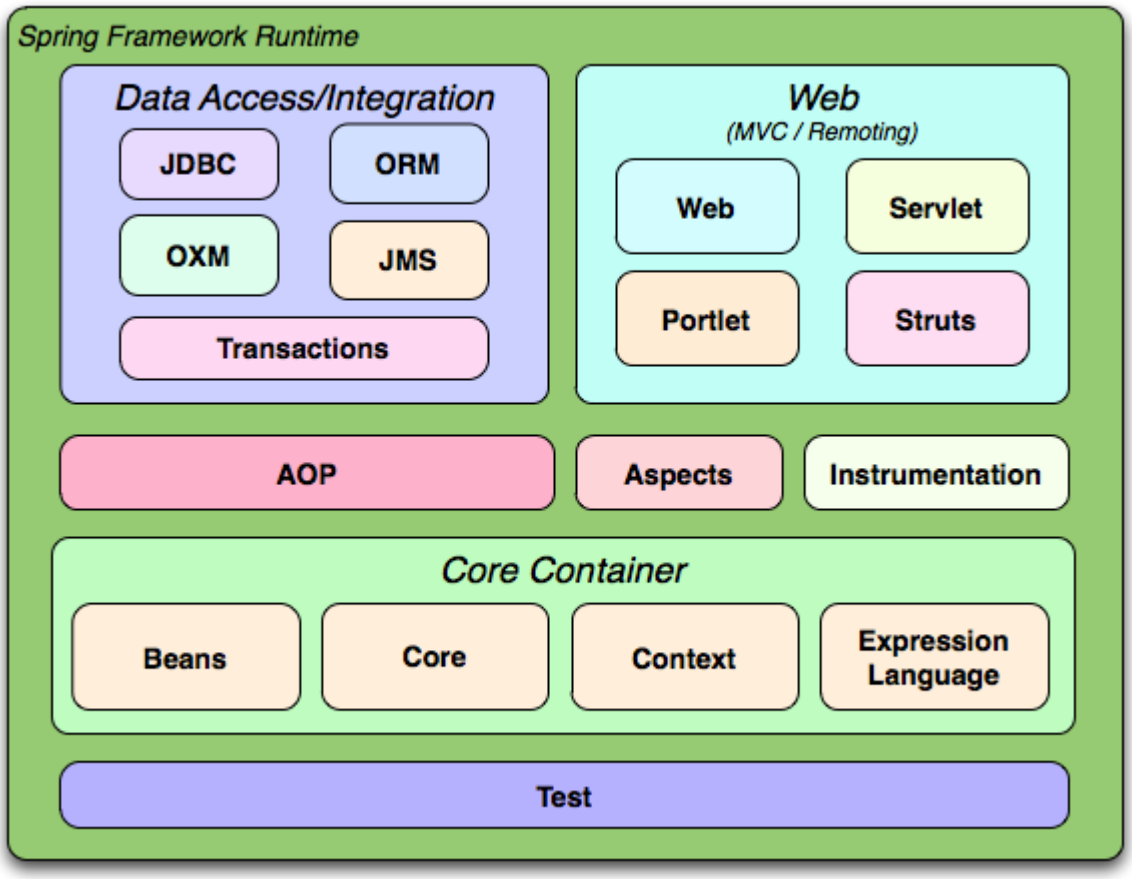


Рис. 2.3. Компоненти фреймворку Spring

Для реалізації необхідного функціоналу були використані модулі фреймворку Spring. Нижче перераховано всі використані фреймворки, які входять до фреймворку Spring.

Фреймворк Spring Web model-view-controller (MVC) (рис. 2.3) розроблений на основі DispatcherServlet, який відправляє запити обробникам, з налаштовуваними обробниками, роздільною здатністю, локаллю та роздільною здатністю теми, а також підтримкою завантаження файлів. Обробник за замовчуванням базується на анотаціях @ Controller і @ RequestMapping, пропонуючи широкий спектр гнучких методів обробки. Даний фреймворк використовує протокол HyperText Transfer Protocol (HTTP) для роботи.

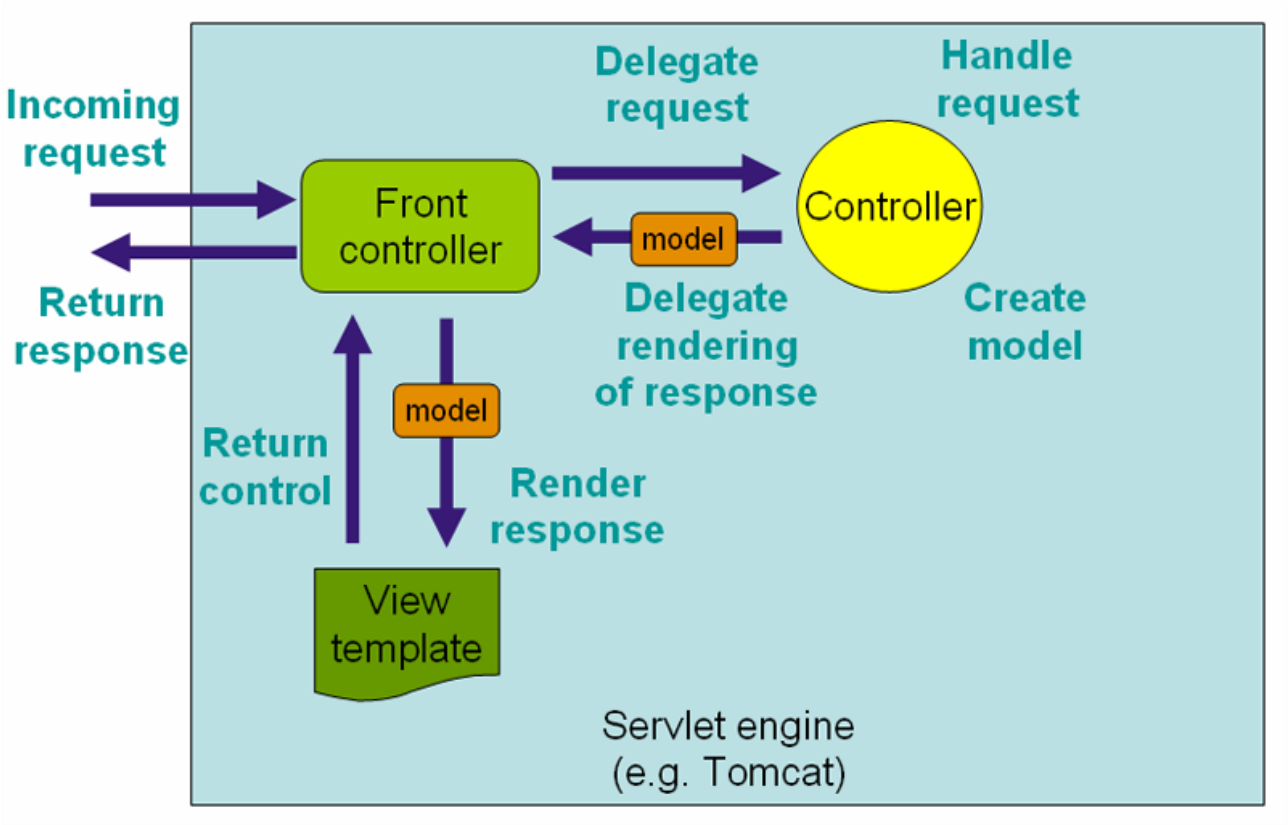


Рис. 2.4. MVC та Front Controller патерн в Spring Web

Модуль Spring Web дає можливість створювати додатки і з графічним інтрфейсом, і без нього.

Для того, щоб використовувати Spring Web додатки необхідна окрема інфраструктура – контейнер сервлетів. Одним з найпопулярніших контейнерів сервлетів є Apache Tomcat. Цей інструмент використовується для розгортання та запуску веб-додатків, що написані з використанням сервлетів та мови програмування Java.

Spring Security - це фреймворк, який фокусується на забезпеченні автентифікації та авторизації Java-додатків. Як і всі проекти Spring, реальна потужність Spring Security знаходиться в тому, наскільки легко вона може бути розширена для задоволення індивідуальних потреб.

Основні функції, які виконує Spring Security фреймворк:

– всебічна та розширювана підтримка автентифікації та авторизації;

– захист від атак, таких як фіксація сеансу, клікджекінг, підробка запиту на перехресний сайт та інших;

– інтеграція з Servlet API;

– додаткова інтеграція з фреймворком Spring Web MVC.

Spring Data JPA, частина більшого сімейства Spring Data, дозволяє легко реалізувати репозиторії на базі JPA. Цей модуль займається розширеною підтримкою шарів доступу до даних на основі JPA. Це полегшує створення Spring-додатків, які використовують технології доступу до даних.

Spring Data JPA має на меті значно покращити реалізацію шарів доступу до даних за рахунок зменшення зусиль до суми, яка насправді потрібна. Розробнику необхідно написати інтерфейси репозиторію (рис. 2.5), включаючи власні методи пошуку, і Spring забезпечить реалізацію автоматично.

```
@Repository
public interface PeopleRepository extends JpaRepository<Person, Long> {
    Optional<Person> findByUsername(String username);

    Optional<Person> findByEmail(String email);

    void deleteById(Long id);
}
```

Рис. 2.5. Приклад створеного репозиторію

Spring Boot - фреймворк на основі Java з відкритим вихідним кодом, що використовується для створення мікросервісів. Spring Boot забезпечує хорошу платформу для розробників Java для розробки автономного та виробничого Spring додатка, який можна запросто запустити. Розробник має можливість почати роботу з мінімальними конфігураціями без необхідності налаштування всієї конфігурації Spring.

Фреймворк Spring Boot створювався для вирішення наступних задач:

– щоб уникнути використання складної конфігурації XML у Spring;

– для полегшення процесу розробки додатків;

– щоб скоротити час розробки і запускати додаток самостійно;

– запропонувати більш простий спосіб початку роботи з додатком.

Spring Boot має вбудований повноцінний контейнер Apache Tomcat, що дозволяє розробнику сфокусуватися на процесі розробки, а не на збірці проекту. Даний фреймворк має велику кількість підмодулів (стартерів).

Нижче наведено список стартерів, які були використані при написанні кваліфікаційної роботи.

Spring Boot Web: підмодуль Spring Boot, призначений для прискорення розробки веб-додатків. Містить вбудований контейнер сервлетів – Apache Tomcat.

Spring Boot Security: підмодуль Spring Boot, який спрощує конфігурацію безпеки веб-додатків.

Spring Boot Data JPA: підмодуль Spring Boot, який спрощує роботи з базами даних.

Spring Boot Thymeleaf: підмодуль для роботи з технологією Thymeleaf, що дозволяє генерувати динамічні веб-сторінки для Spring Web на основі HTML.

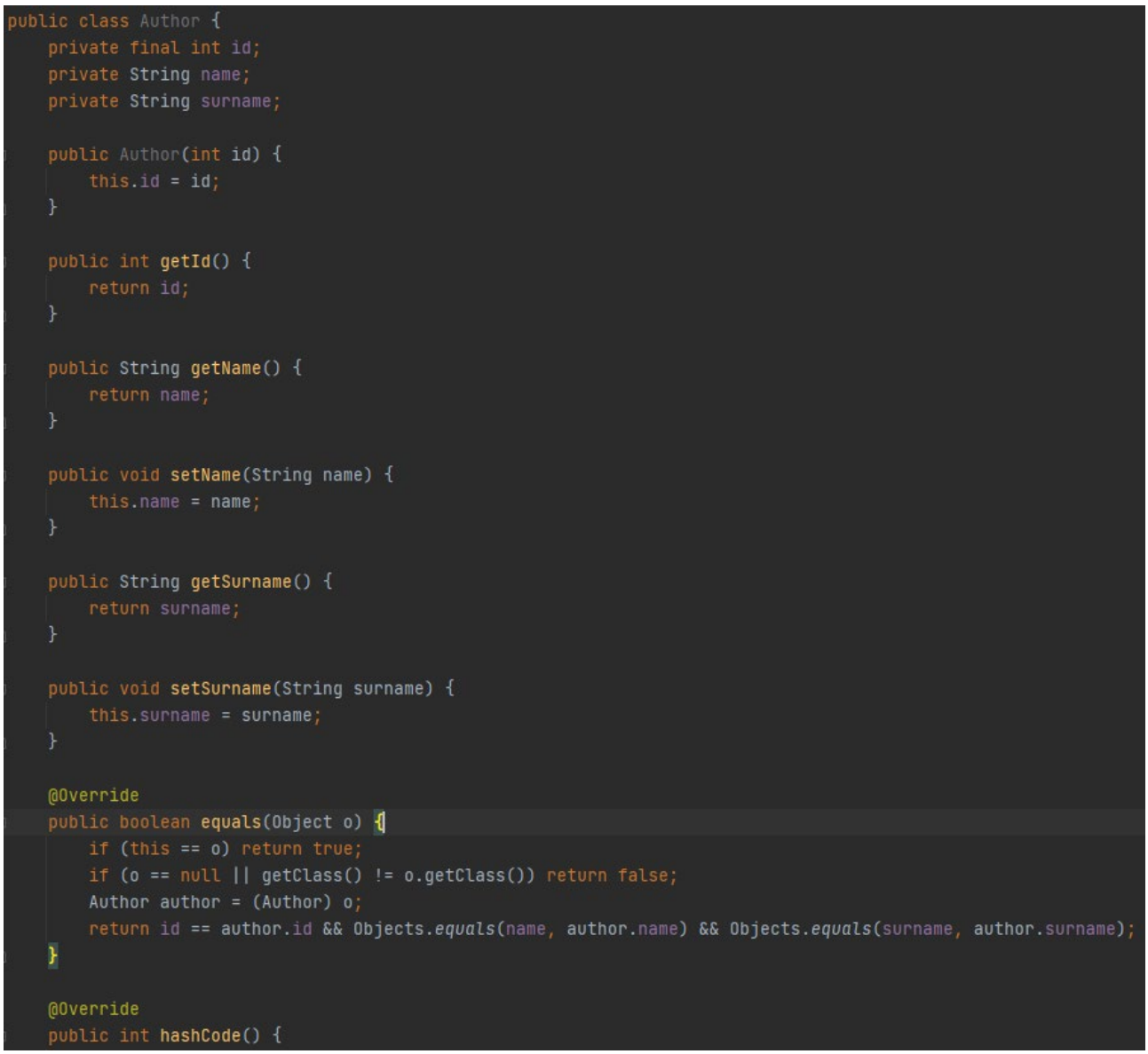
Для прискорення розробки веб-додатку була використана бібліотека Lombok.

Lombok – одна з найпопулярніших бібліотек для розробки на мові програмування Java. Вона базується на використанні анотацій з бібліотеки Java, що допомагають скоротити повторюваний код, такий як написання getter-ів та setter-ів, перевизначення методів toString(), equals(), hashCode() та інших, а також написання конструкторів.

```java
import lombok.Data;

@Data
public class Author {
    private final int id;
    private String name;
    private String surname;
}
```

Рис. 2.6. Вигляд програмного коду з використанням бібліотеки Lombok



```java
public class Author {
    private final int id;
    private String name;
    private String surname;

    public Author(int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

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

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Author author = (Author) o;
        return id == author.id && Objects.equals(name, author.name) && Objects.equals(surname, author.surname);
    }

    @Override
    public int hashCode() {
```

Рис. 2.7. Вигляд програмного коду без використання бібліотеки Lombok

H2 Connector - це набір інтерфейсів, які дозволяють веб-застосунку працювати з базою даних H2. Він включає в себе необхідні методи підключення, безпеки, цілісності даних і запиту бази даних.

HTML - стандартизована мова розмітки для перегляду веб-сторінок у браузері. Вона також може використовувати додаткові технології, такі як JavaScript для створення сценаріїв і динамічних подій у документах, і CSS для зміни візуальних атрибутів HTML-елементів.

JavaScript (JS) – динамічна, об’єктно-орієнтована, прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-

сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

CSS (*англ*. Cascading Style Sheets) — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних. Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Thymeleaf - сучасний рушій шаблонів Java на стороні сервера як для веб-середовища, так і для окремих середовищ. Основна мета Thymeleaf полягає в розробці елегантних шаблонів HTML, які можуть бути правильно відображені в браузерах, а також працювати як статичні прототипи, що дозволяє посилити співпрацю в командах розробників.

Bootstrap - це безкоштовний фреймворк розробки з відкритим вихідним кодом для створення веб-сайтів та веб-додатків. Розроблений для забезпечення адаптивної розробки веб-сайтів для мобільних пристроїв, Bootstrap надає колекцію синтаксису для шаблонів.

Як фреймворк Bootstrap включає основи для адаптивної веб-розробки, тому розробникам потрібно лише вставити код у заздалегідь визначену систему сітки. Фреймворк Bootstrap побудований на Hypertext Markup Language (HTML), каскадних таблицях стилів (CSS) і JavaScript. Веб-розробники, які використовують Bootstrap, можуть створювати веб-сайти набагато швидше, не витрачаючи час.

Apache Maven - це інструмент управління та розуміння програмними проектами. Виходячи з концепції об'єктної моделі проекту (POM), Maven може керувати побудовою проекту, звітуванням та документацією з центральної частини інформації.

Hibernate - Java фреймворк, який спрощує розробку програми Java для взаємодії з базою даних. Це інструмент з відкритим кодом, легкий, ORM

(Object Relational Mapping). Hibernate реалізує специфікації JPA (Java Persistence API) для збереження даних.

Інструмент ORM спрощує створення даних (рис. 2.8), маніпулювання даними та доступ до даних. Це техніка програмування, яка відображає об'єкт до даних, що зберігаються в базі даних та внутрішньо використовує JDBC API.

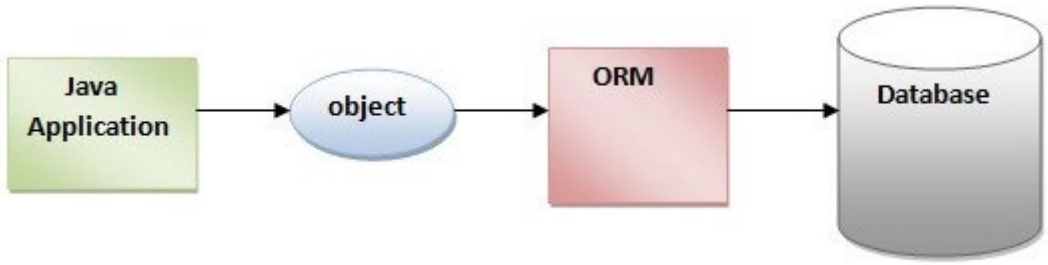


Рис. 2.8. Схема збереження об’єкту в базу даних за допомогою ORM

Основні переваги використання фреймворку Hibernate:

1. відкритий програмний код та легкість: hibernate має відкритий програмний код під ліцензією LGPL і є легким;

2. швидка робота: продуктивність hibernate є високою, оскільки кеш внутрішньо використовується у фреймворці. Існує два типи кешу в hibernate: кеш першого рівня і кеш другого рівня. Кеш першого рівня включений за замовчуванням.

3. незалежні запити до бази даних: HQL (Hibernate Query Language) - об'єктно-орієнтована версія SQL. Він генерує базу незалежних запитів. Тому розробнику не потрібно писати конкретні запити до бази даних.

4. автоматичне створення таблиць: Hibernate framework надає можливість автоматично створювати таблиці бази даних. Тому немає необхідності створювати таблиці в базі даних вручну.

5. надає статистику запитів та стан бази даних: Hibernate підтримує Query cache і надає статистику про запит і стан бази даних.

Останньою використаною технологією є система управління базою даних (СУБД) H2.

H2 - легка база даних Java з відкритим кодом. Вона може бути вбудованою в Java додаток або працювати в режимі клієнт-сервер. В основному, база даних H2 може бути налаштована для запуску як вбудована база даних, що означає, що дані не будуть зберігатися на диску. Через вбудовану базу даних вона не використовується для розробки виробництва, але в основному використовується для розробки та тестування.

Основні характеристики H2 бази даних:

– потужний та швидкий двигун;

– підтримка стандартів SQL та JDBC API. Також може використовувати PostgreSQL ODBC driver;

– підтримка кластеризації та багатоваріантності;

– сильні функції безпеки.

Для роботи з базою даних користувач повинен мати веб-браузер та H2 консольний сервер.

2.4. Опис структури системи та алгоритмів її функціонування

Елементи створеного веб-додатку логічно розбиті на директорії:

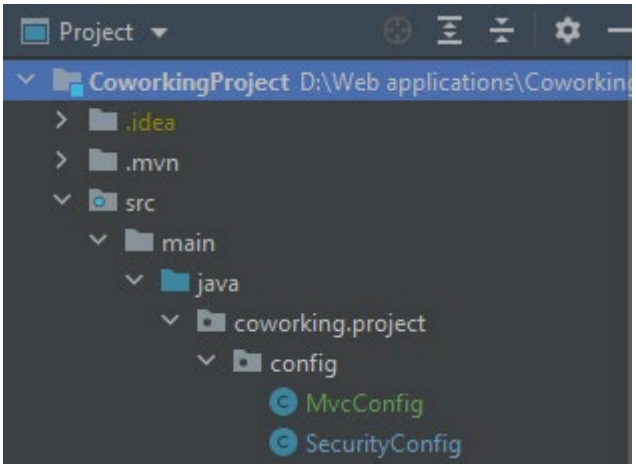


Рис. 2.9. Директорія Config

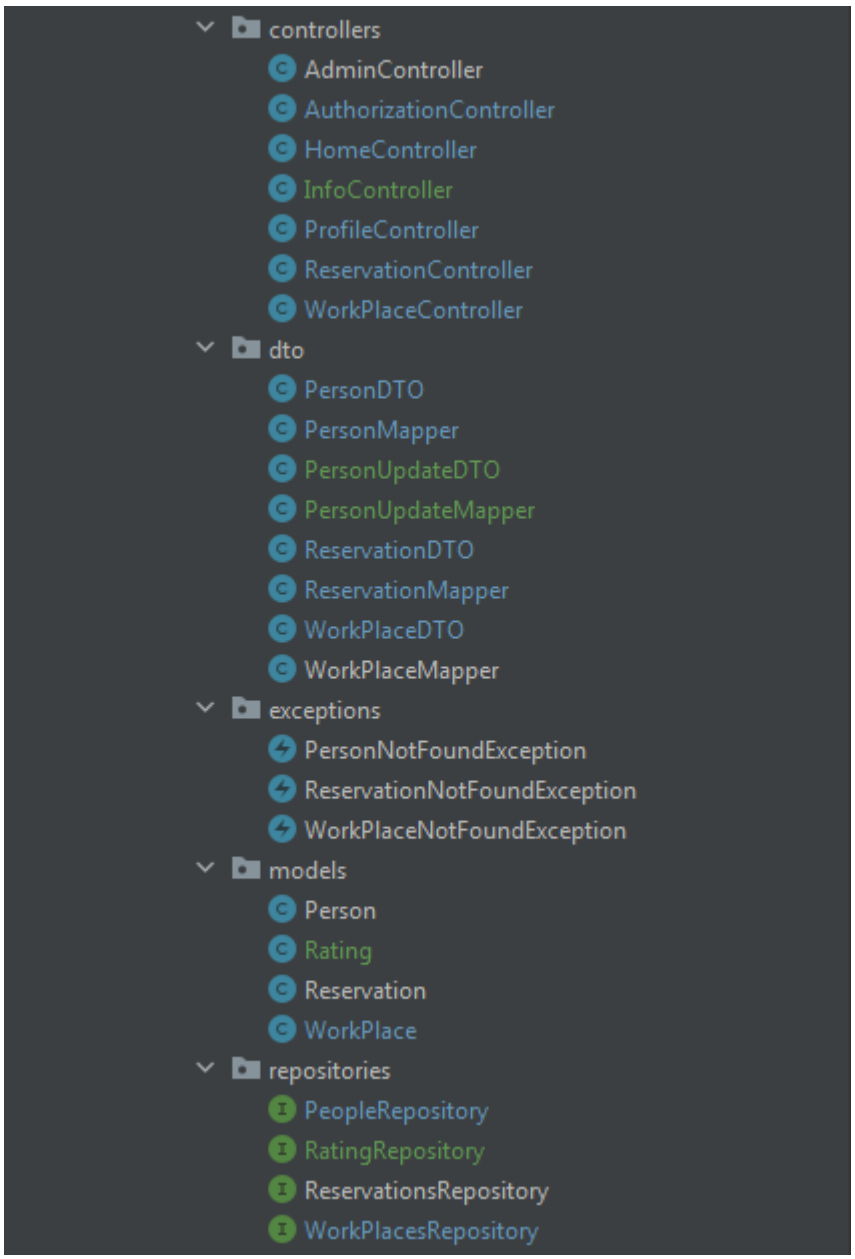


Рис. 2.10. Директорії Controllers, DTO, Exceptions, Models, Repositories

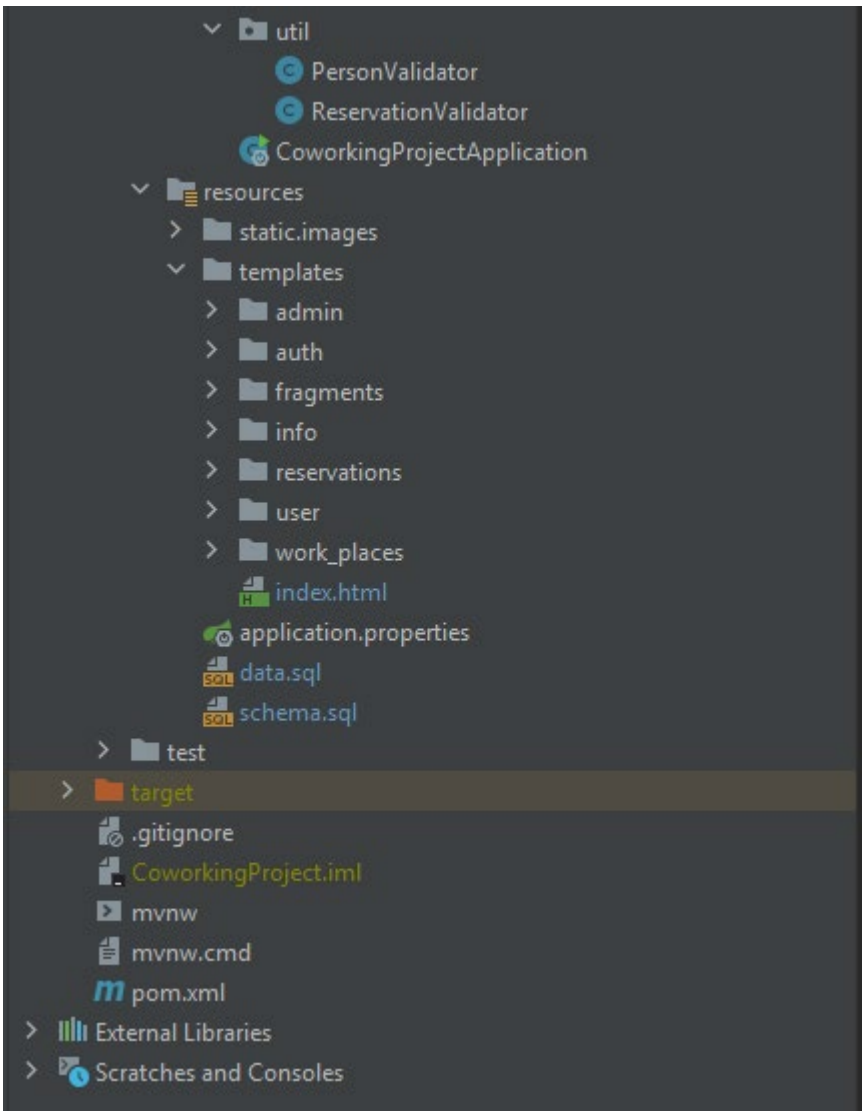


Рис. 2.11. Інші директорії та файли проекту

За структурою створений веб-додаток можна поділити на дві частини:

– директорії з Java класами;

– директорії з іншими файлами (конфігурації, html-сторінок).

Класи розташовані в папці controllers містять інформація про те, як сервер оброблює запит користувача на певну адресу і яку html-сторінку має повернути після обробки запиту.

Класи у директорії config повністю відповідають за конфігурацію веб-додатку: налаштування безпеки та можливість доступу до статичних даних додатку.

Класи з каталогу dto, так звані DTO моделі (Data Transfer Object), відповідають за безпечне «спілкування» між контролерами та веб-сторінками.

Класи розташовані у папці exceptions використовуються в програмному коді при виникненні непередбачуваної ситуації (коли користувач намагається знайти робоче місце, користувача, замовлення яких не існує).

Класи з директорії models описують таблиці з бази даних.

У каталозі repositories зберігаються інтерфейси, котрі надають можливість працювати з базою даних. Це досягається за допомогою фреймворку Spring Data.

Папка util містить у собі два класи, котрі використовуються для валідації даних, котрі користувач вводить до форм.

Каталог resources містить підкаталог static/images котрий відповідає за збереження зображень, котрі використовуються веб-додатком.

Також каталог resources містить у собі інший підкаталог – templates. Даний каталог зберігає у собі усі написанні веб-сторінки, котрі структуровано розташовані по підкаталогам.

Не менш важливим є файл pom.xml. Даний файл відповідає за завантаження необхідних бібліотек до проекту під час його зборки.

Алгоритм, котрий використовує розроблений веб-додаток, для обробки вхідних запитів (рис. 2.12).

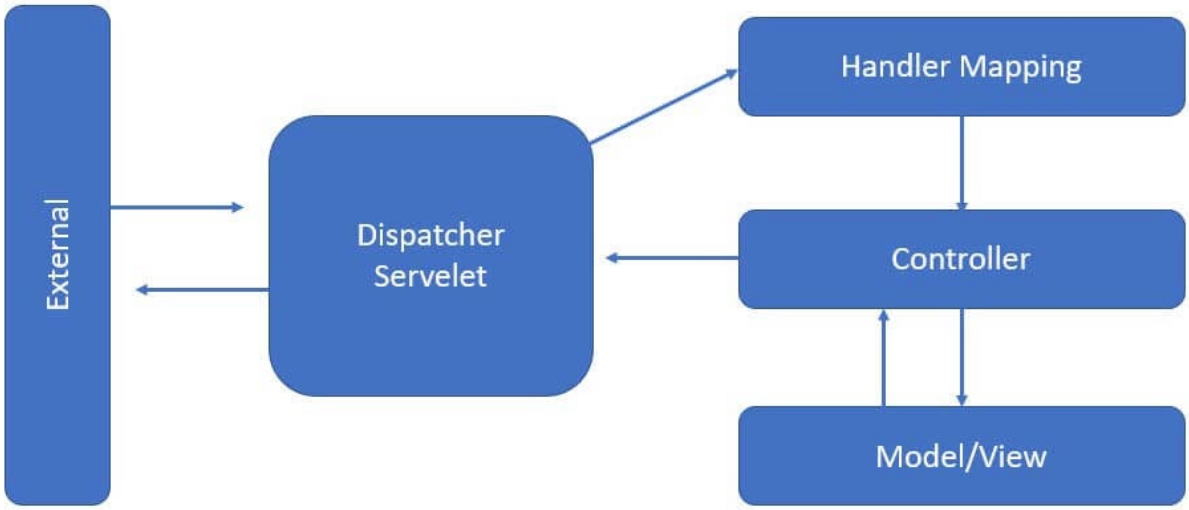


Рис. 2.12. Алгоритм обробки запитів

Усі запити від клієнтів потрапляють до особливого класу – DispatcherServlet, який є вхідною точкою всіх запитів.

DispatcherServlet використовує метод HandlerMapping, який визначає до якого контролеру необхідно відправити запит, що надійшов.

Після того як запит надійшов до необхідного контролеру, веб-додаток передає параметри запиту до відповідного сервісу, який уже використовує потрібний репозиторій для маніпулювання даними з бази даних.

Після успішного виконання необхідних дій, сервіс повертає результат контролеру, котрий в свою чергу повертає необхідну веб-сторінку.

Алгоритм роботи з БД

Для роботи та виконання базових дій (читання, збереження, редагування та видалення даних з БД) було використано фреймворк Spring Data. Він значно пришвидшує роботу та дозволяє не писати типові запити до БД. Розробнику необхідно створити репозиторій, котрий є інтерфейсом, та унаслідувати його від інтерфейсу JpaRepository. Для додавання кастомних методів необхідно у створеному репозиторії задати їх сигнатуру, згідно встановлених правил. І далі

фреймворк самостійно реалізує даний метод та згенерує відповідний запит до БД.

```
@Repository
public interface PeopleRepository extends JpaRepository<Person, Long> {
    Optional<Person> findByUsername(String username);

    Optional<Person> findByEmail(String email);

    void deleteById(Long id);
}
```

Рис. 2.13. Приклад створеного репозиторію

Для збереження даних використовується СУБД H2. Структура схеми БД системи (рис 2.14.) має наступні сутності та їх атрибути:

1. Таблиця користувача (Person). Атрибути:
   - ID користувача;
   - електронна адреса;
   - ім’я користувача в справжньому вигляді;
   - пароль;
   - роль;
   - рік народження;
2. Таблиця робоче місце (Work_place). Атрибути:
   - ID робочого місця;
   - назва робочого місця;
   - опис;
   - ціна за годину оренди;
   - статус зайнятості місця.
3. Таблиця бронювання (Reservation). Атрибути:
   - ID броні;
   - ID робочого місця;
   - ID користувача;

- день оренди;
- час початку оренди;
- час кінця оренди;
- статус оплати;
- статус підтвердження бронювання;
- загальна сума бронювання.

4. Таблиця рейтингу (Rating). Атрибути:
   - ID рейтингу;
   - ID робочого місця;
   - кількість орендарів.

Далі наведено ключі, за рахунок яких пов’язані сутності БД.

1. Користувачі (ID користувача):
   - ID користувача – первинний ключ.
2. Робочі місця (ID робочого місця):
   - ID робочого місця – первинний ключ.
3. Бронювання (ID бронювання, ID робочого місця, ID користувача):
   - ID бронювання – первинний ключ;
   - ID робочого місця – зовнішній ключ;
   - ID користувача – зовнішній ключ.
4. Рейтинг (ID бронювання, ID робочого місця):
   - ID бронювання – первинний ключ;
   - ID робочого місця – зовнішній ключ.

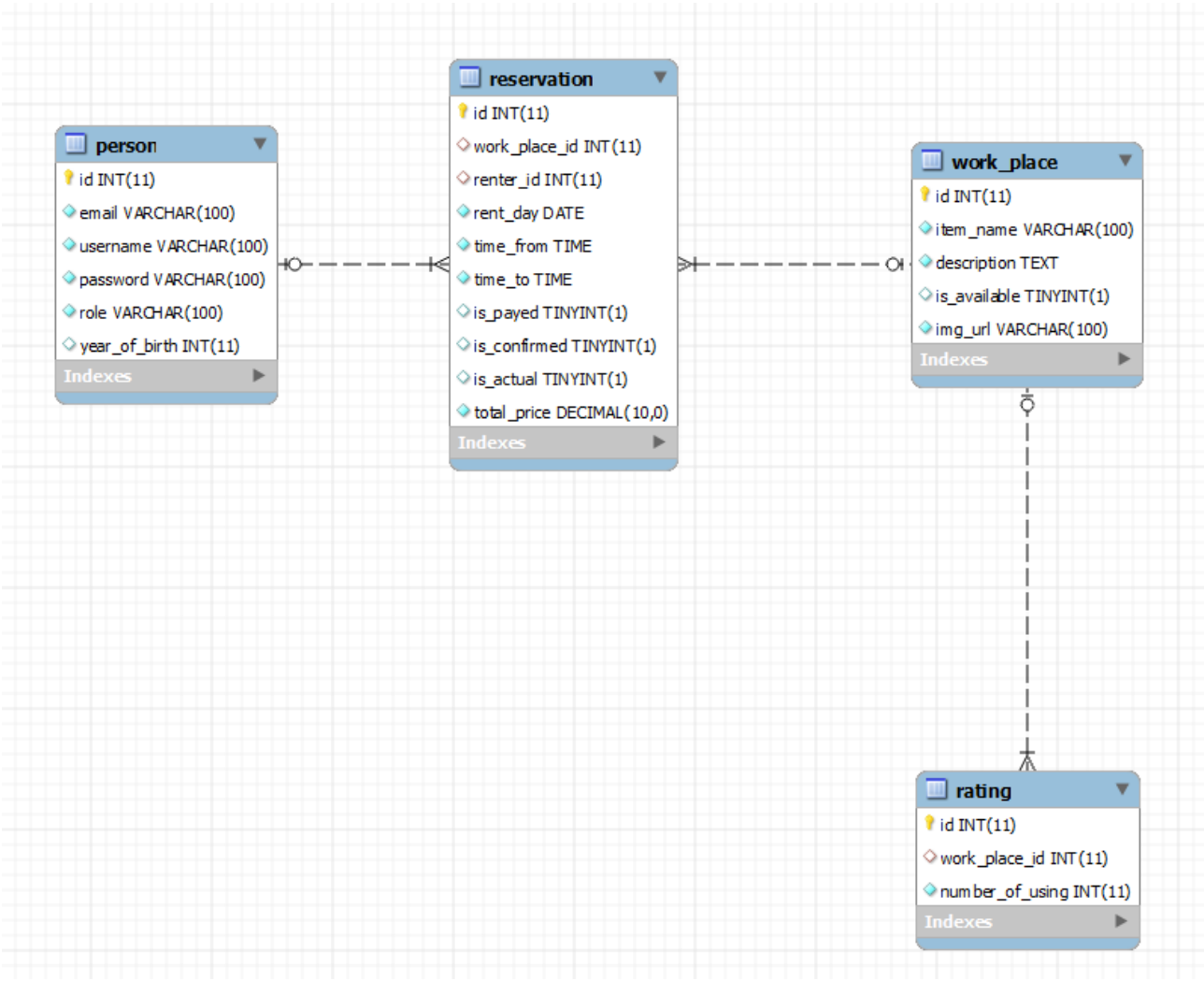


Рис. 2.14. Схема розробленої БД

2.5. Обґрунтування та організація вхідних та вихідних даних програми

В даній кваліфікаційній роботі головними вхідними даними що постійно оброблюються є дані, які були отримані від користувача. Такі данні приходять з форм, котрі розміщені у графічному інтерфейсі веб-додатку. Додаток отримує дані користувача при оформленні замовлення або при оплаті замовлення на бронювання. При обробці даних, які надійшли від користувача при створенні нового запиту на бронювання робочого місця в базі даних шукається робоче місце с таким же ідентифікайним кодом, який був обраний при бронюванні робочого місця, якщо робоче місце з таким номером вільне, то додаток збирає

дані з форми та оформлює запит на бронювання. На виході ми отримаємо створений запит з коректно обраною датою та часом бронювання, статусом на оплату та підтвердження від адміністратора та загальну суму бронювання.

Також додаток обробляє механізм симуляції оплати бронювання, одразу після оформлення запиту. Після чого на виході ми отримаємо повідомлення на електронну адресу про вдалу оплату бронювання та відправлення запиту до адміністратора для підтвердження.

Щодо вхідних даних які система отримає безпосередньо від користувача завдяки графічному інтерфейсу, до таких належать наступні данні: для реєстрації (електрона адреса, пароль, ім’я користувача, вік) та авторизації(електрона адреса та пароль).

До вихідних даних відносяться поточні данні робочих місць (ціна, назва, опис, зображення робочого місця, статус), данні користувача(ім’я, вік, електронна адреса), замовлення користувача (сума замовлення, номер робочого місця, статус оплати).

2.6. Опис розробленого програмного продукту

2.6.1. Використані технічні засоби

При розробці та тестуванні роботи програми був використаний персональний комп’ютер від компанії Hewlett-Packard (HP) на основі операційної системи Windows 7 з наступними характеристиками:

- Процесор Intel(R) Core(TM) 2 ;
- Оперативна пам'ять 3 ГБ.

2.6.2. Використані програмні засоби

Додаток було написано в середовищі розробки компанії JetBrains - Intellij Idea Ultimate 2020 року на основі ліцензії для студентів, що надає інтегровані

інструменти для розробки та налагодження додатків на основі Java. У процесі тестування веб додатку був також використаний веб браузер від компанії Google – Google Chrome. База даних H2 є підключеною безпосередньо до самого проекту. Також для роботи, тестування деяких запитів до бази даних, її налагодження та перегляду було використано програмне забезпечення PG Admin – додаток для розробки та адміністрування СУБД PostgreSQL.

2.6.3. Виклик та завантаження програми

Для запуску створеного веб-додатку необхідне певне програмне забезпечення – JRE. Для повноцінної та коректної роботи додатку потрібен файл зі скомпільованим програмним кодом. Запуск програми виконується через консоль, за допомогою команди «java -jar» (рис. 2.15). В якості параметра передається ім’я файлу - CoworkingSpace.jar із скомпільованим кодом.

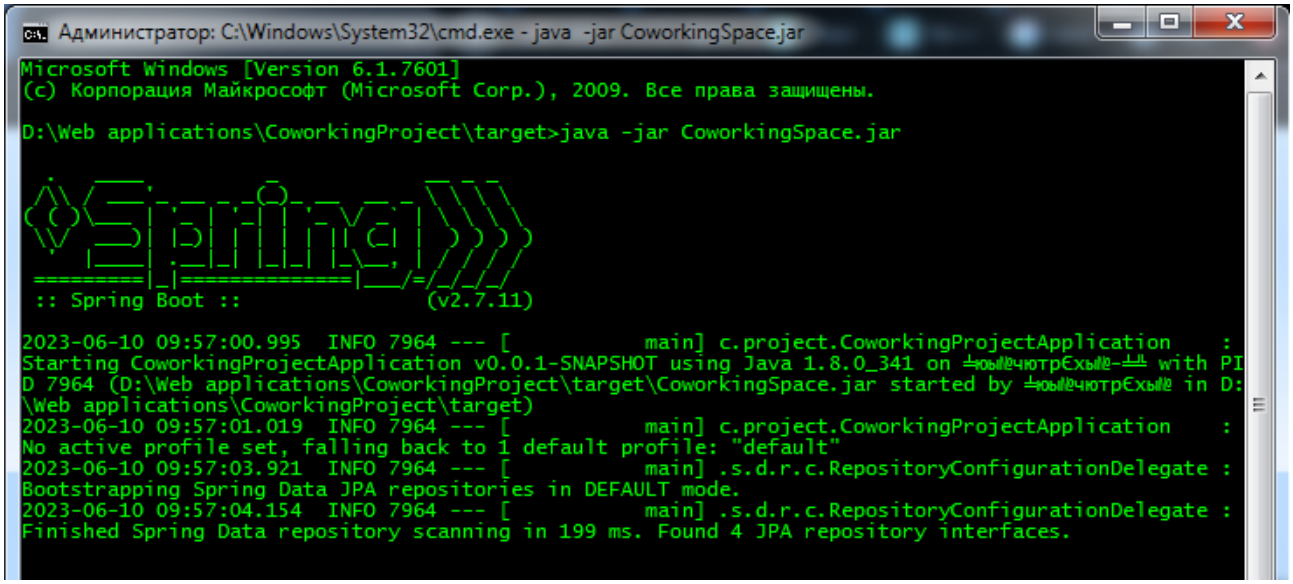


Рис. 2.15. Процес запуску веб-додатку

2.6.4. Опис інтерфейсу користувача

Інтерфейс користувача веб-додатку відображається за допомогою сторінок у інтернет браузері.

Після переходу на веб адресу додатку з’являється головна сторінка (рис. 2.16). Зверху знаходиться навігаційне меню, котре відображається на кожній сторінці додатку. Головна сторінка містить елемент «карусель» для відображення фотографій клієнтів коворкінгу. Додатково на усіх сторінках є «футер» з посиланнями на соціальні мережі.

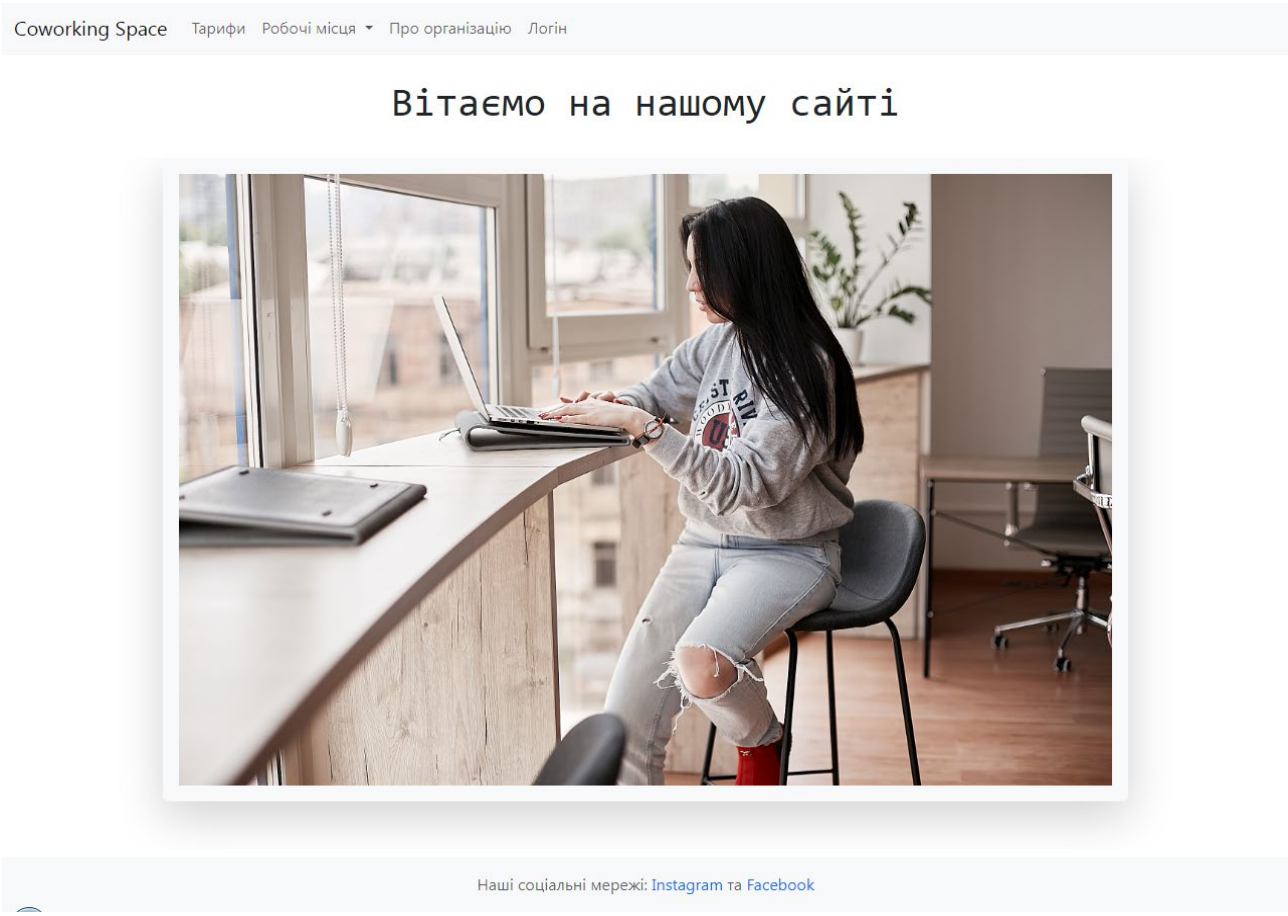


Рис. 2.16. Головна сторінка (вигляд для не авторизованого користувача)

Навігаційне меню (рис. 2.17) складається з наступних пунктів:

– логотипу організації (стилізований текст «Coworking Space»);

– пункту «Тарифи»;

– випадаючого списку «Робочі місця», котрий містить у собі наступні пункти:

1) пункт «Каталог»;

2) пункт «Топ-5»;

3) пункт «Статус»;

– пункту «Про організацію»;
– пункту «Логін».

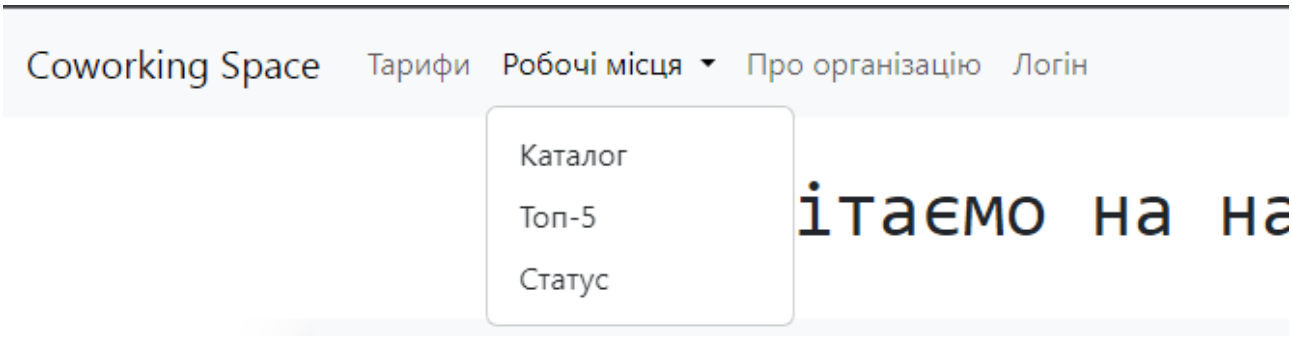


Рис. 2.17. Навігаційне меню

Неавторизований користувач може переглядати наступні сторінки:

– тарифи (рис. 2.18);
– робочі місця:
  1) каталог (рис. 2.19);
  2) топ-5 (рис. 2.20);
  3) статус (рис. 2.21);
– про організацію (рис. 2.22);
– логін (рис. 2.23).

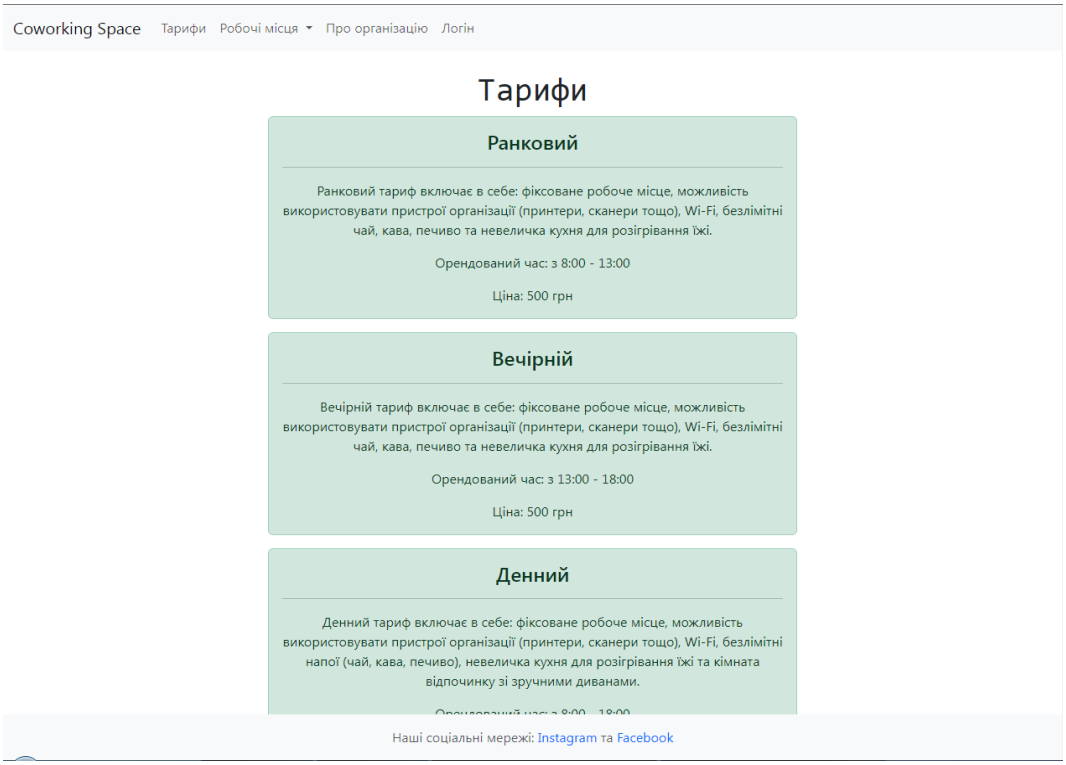


Рис. 2.18. Доступні тарифи на оренду робочих місць

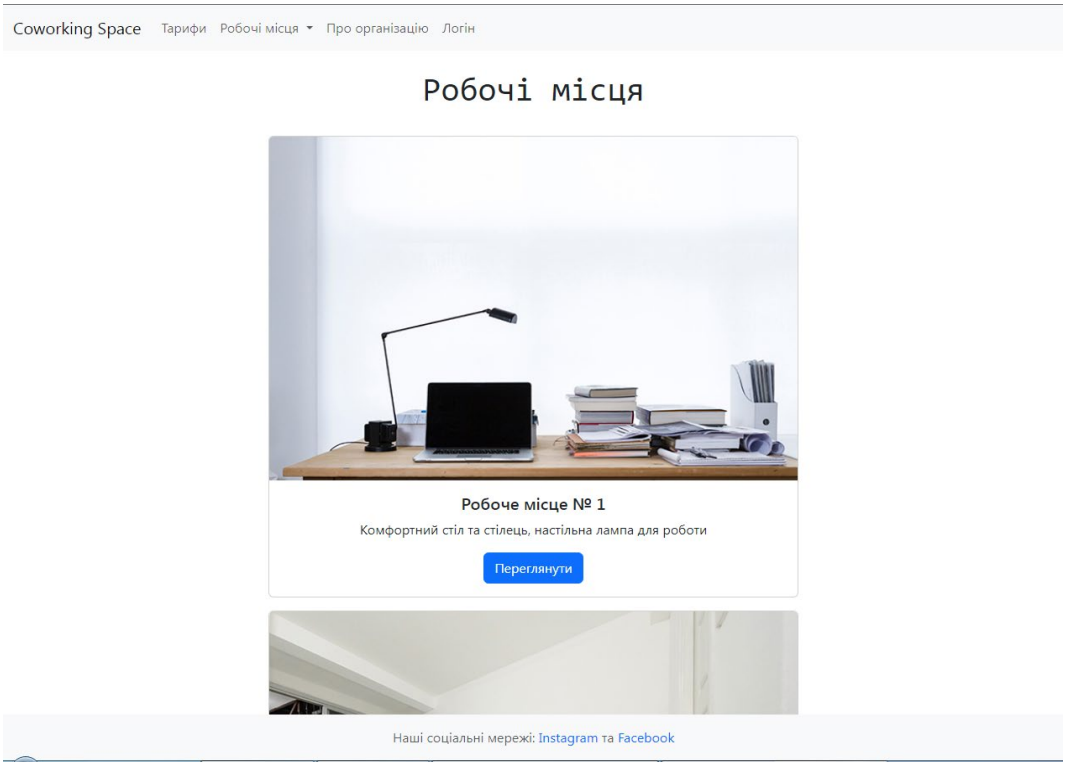


Рис. 2.19. Каталог усіх робочих місць

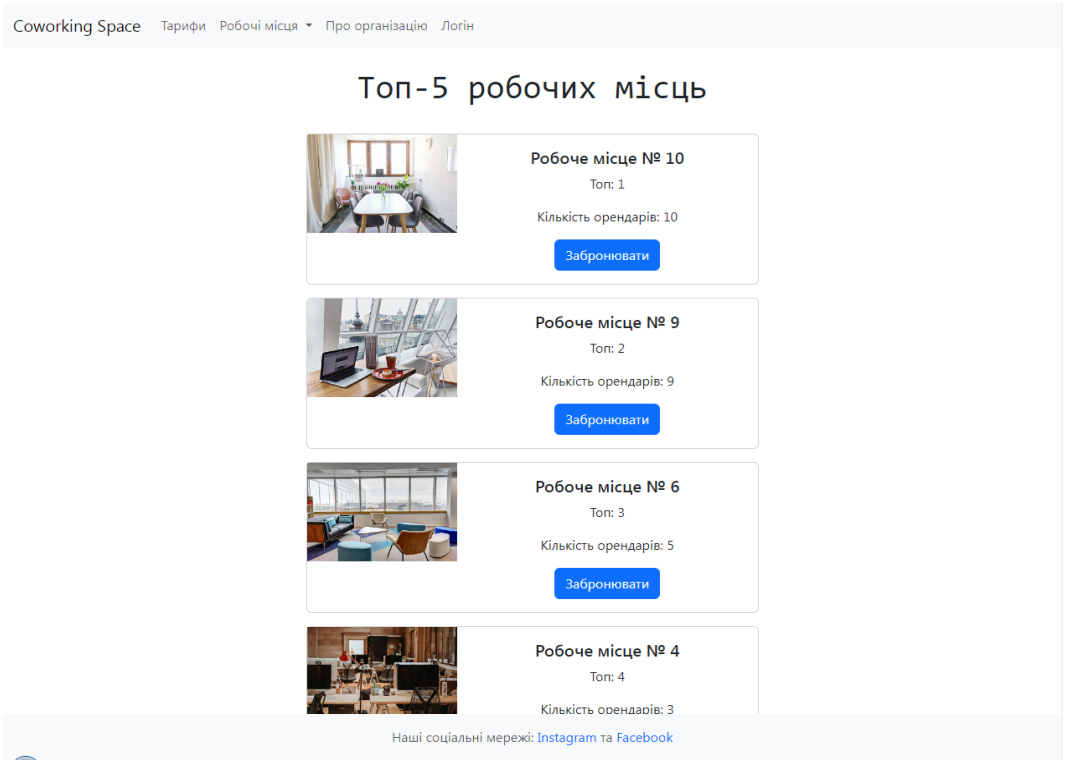


Рис. 2.20. Топ-5 найпопулярніших робочих місць серед відвідувачів

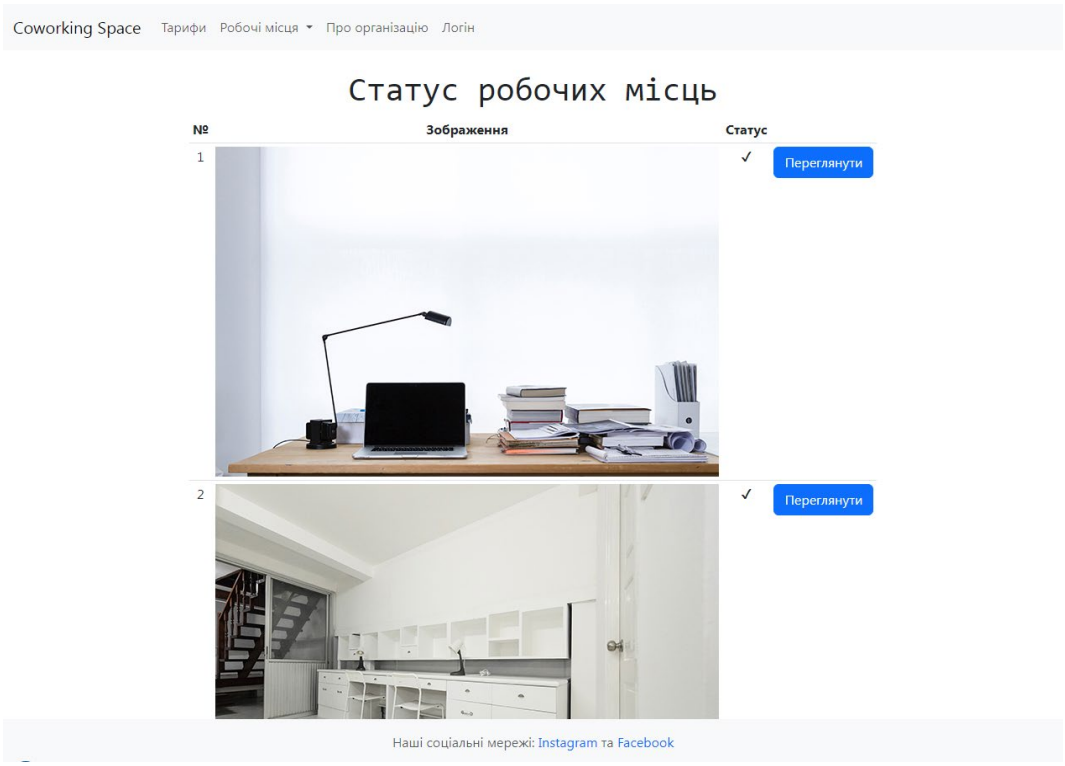


Рис. 2.21. Статус робочих місць станом на зараз

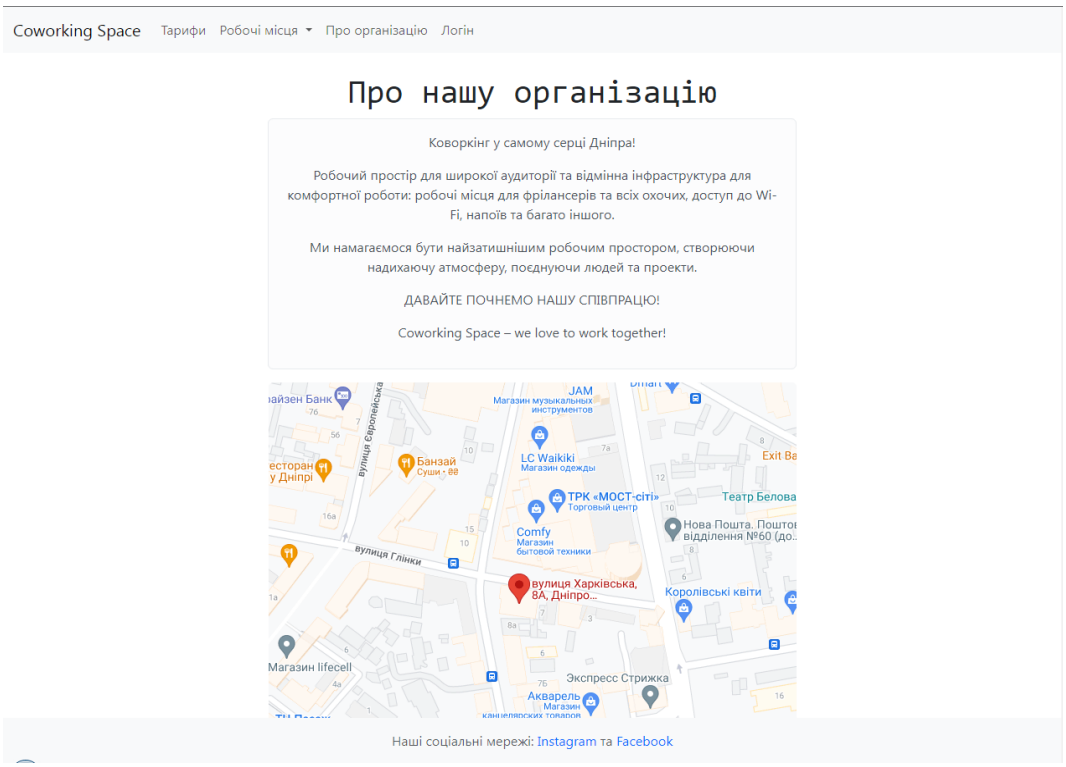


Рис. 2.22. Інформаційна сторінка про організацію

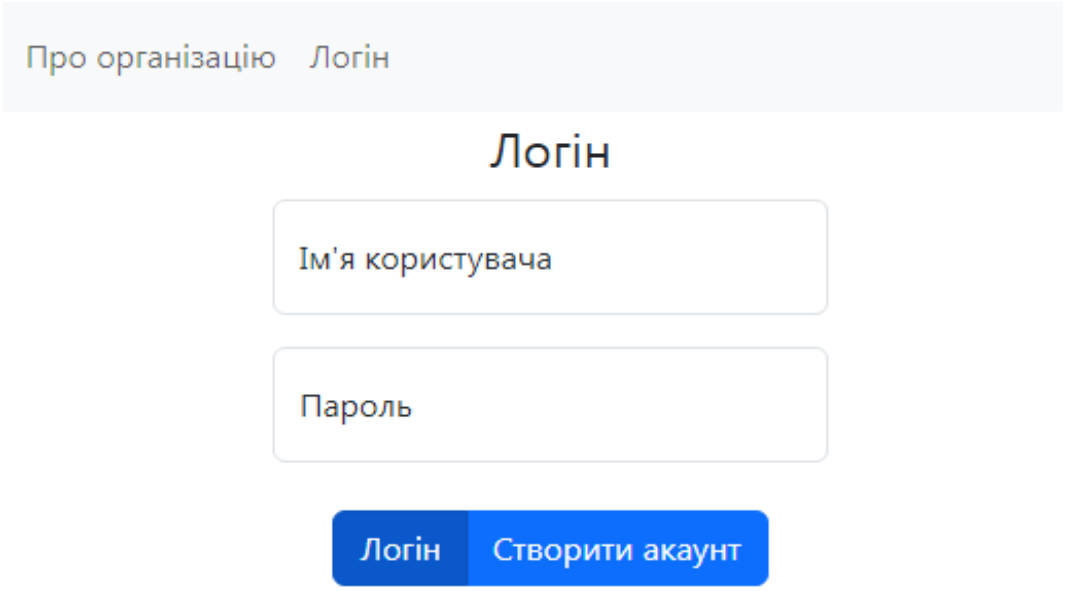


Рис. 2.23. Сторінка для логіну

На сторінці логіну не авторизований користувач має змогу увійти до свого облікового запису, а новий – створити новий акаунт (рис. 2.24).

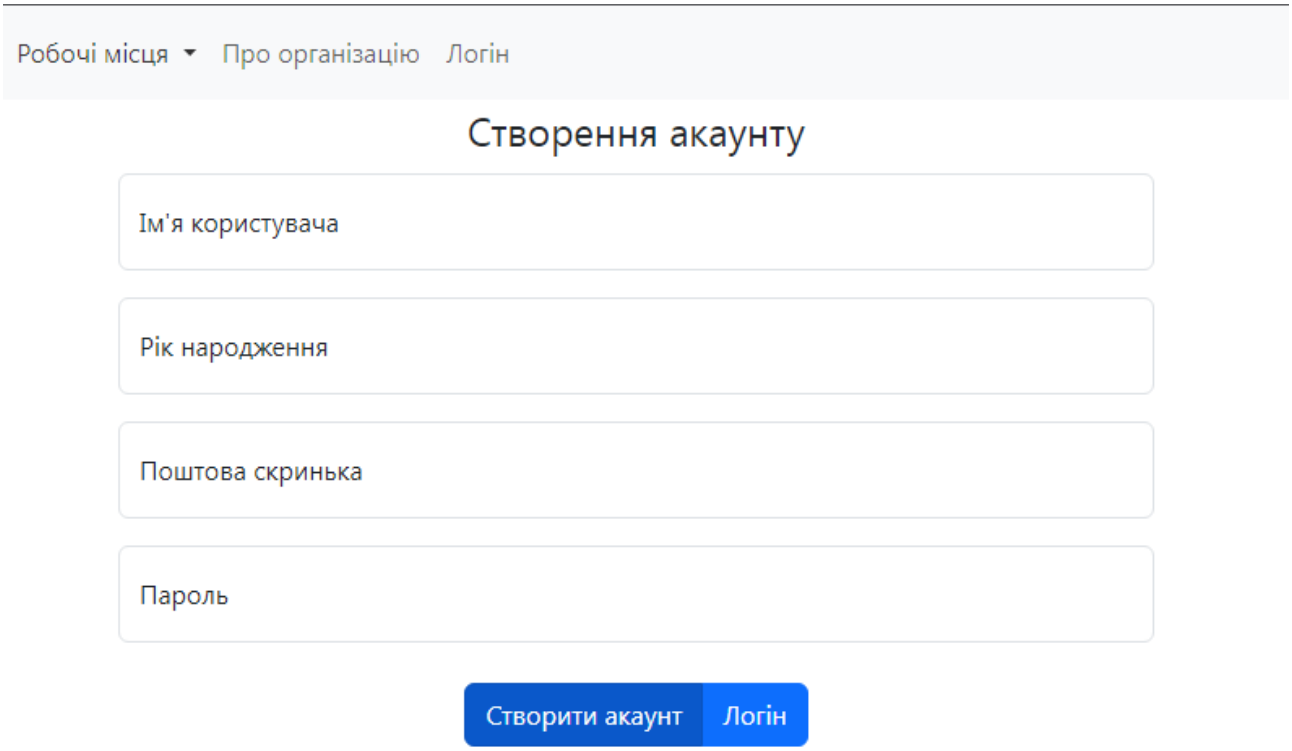


Рис. 2.24. Сторінка створення акаунту

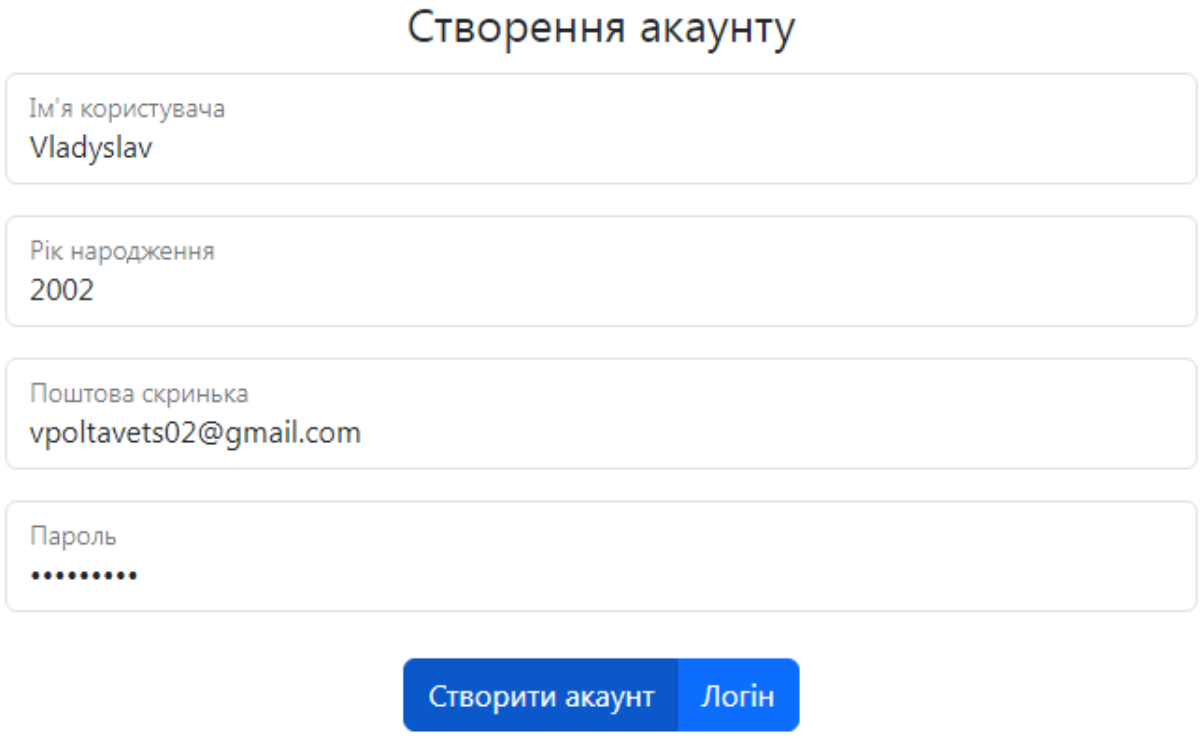


Рис. 2.25. Заповнена форма створення акаунту

Після створення нового профілю на сайті організації, користувачу необхідно буде увійти в тільки но створений обліковий запис. Далі користувач

потрапляє на головну сторінку, але вже для авторизованого користувача (рис. 2.26). Клієнт може побачити привітання, адресоване йому, та пункт «Логін» змінився на «Профіль».

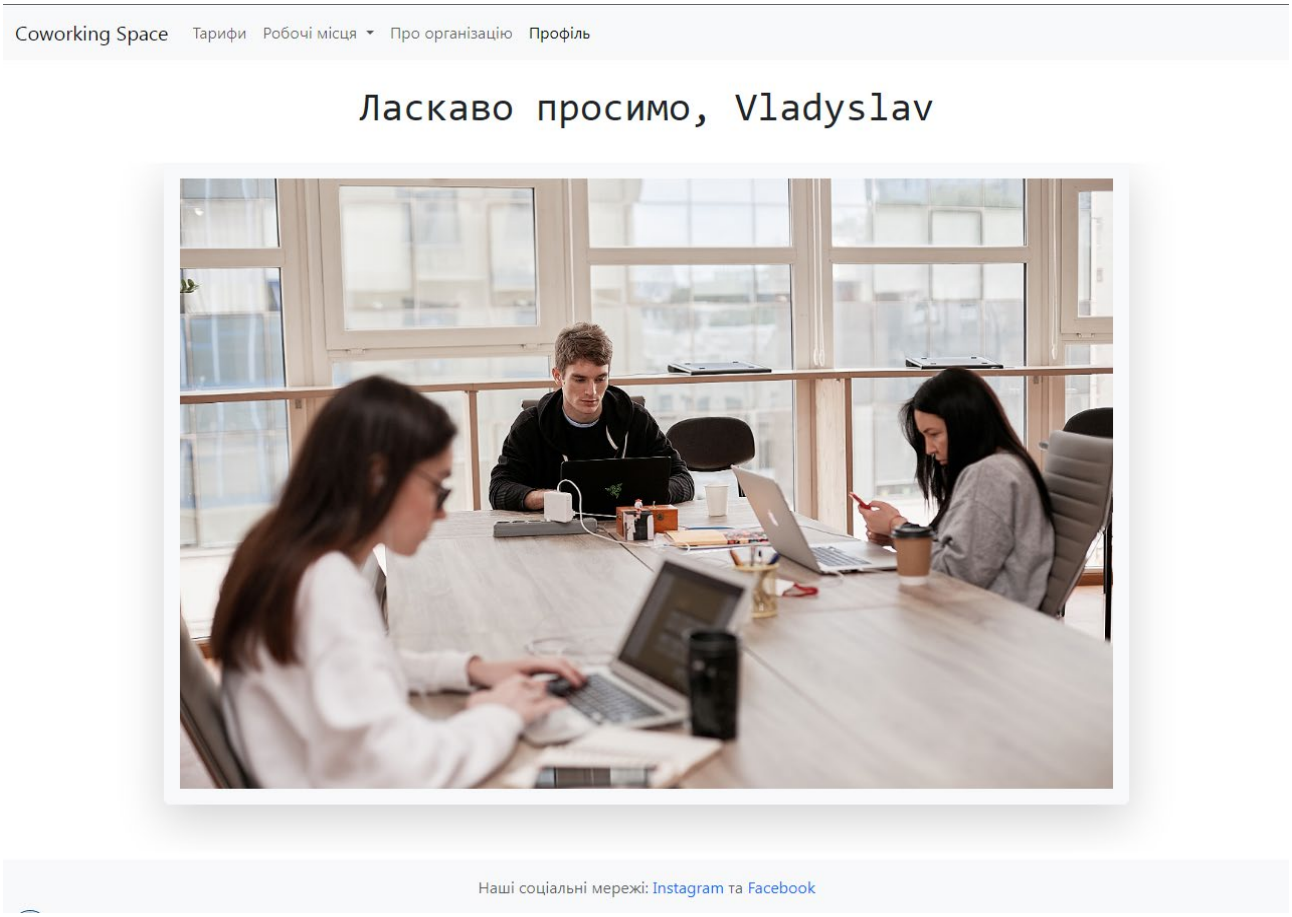


Рис. 2.26. Головна сторінка (вигляд для авторизованого користувача)

В свою чергу авторизований користувач має змогу виконувати наступні дії:

– переглядати інформацію про певне робоче місце (рис. 2.27);

– переглядати та змінювати інформацію про обліковий запис у профілі (рис. 2.28-2.30);

– створювати та сплачувати запити на оренду робочих місць (рис. 2.31, 2.32).

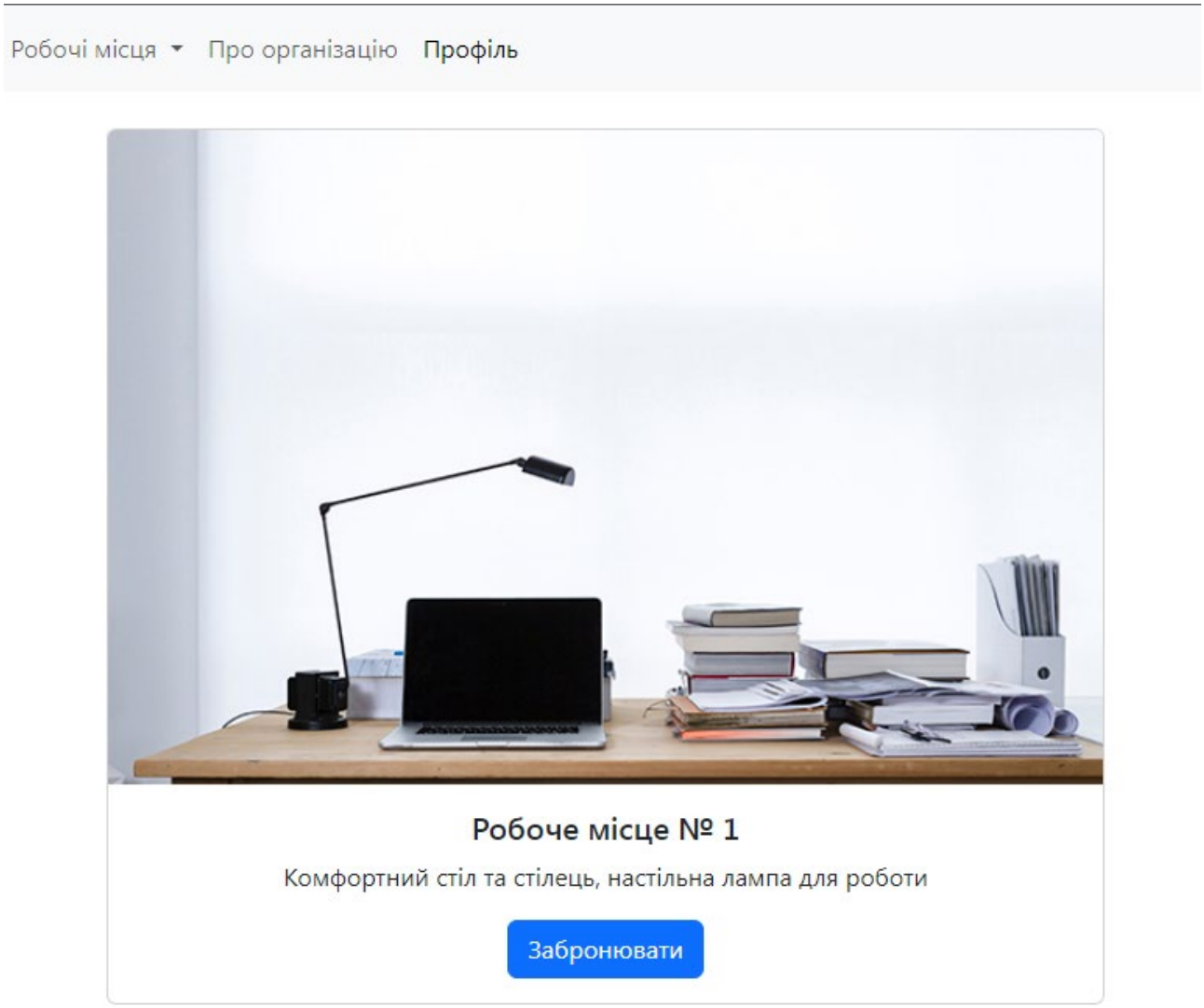


Рис. 2.27. Сторінка огляду робочого місця

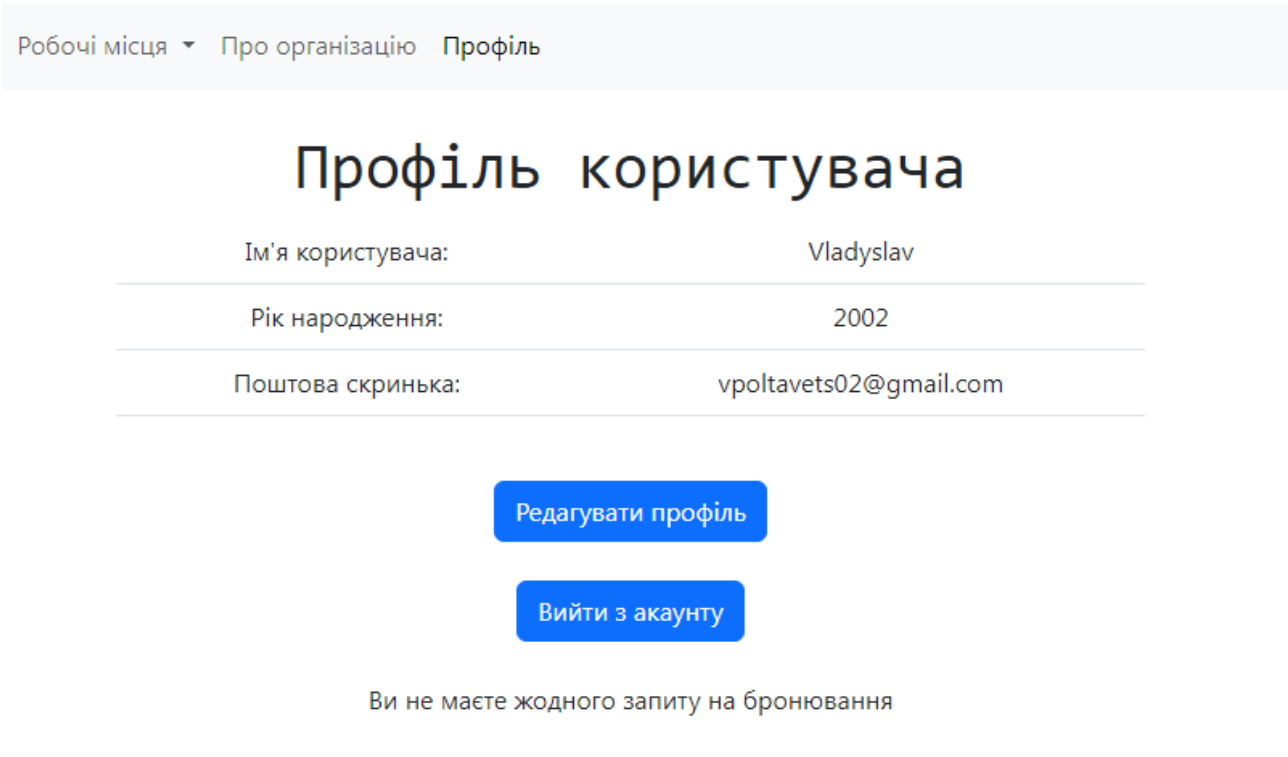


Рис. 2.28. Перегляд профілю користувача

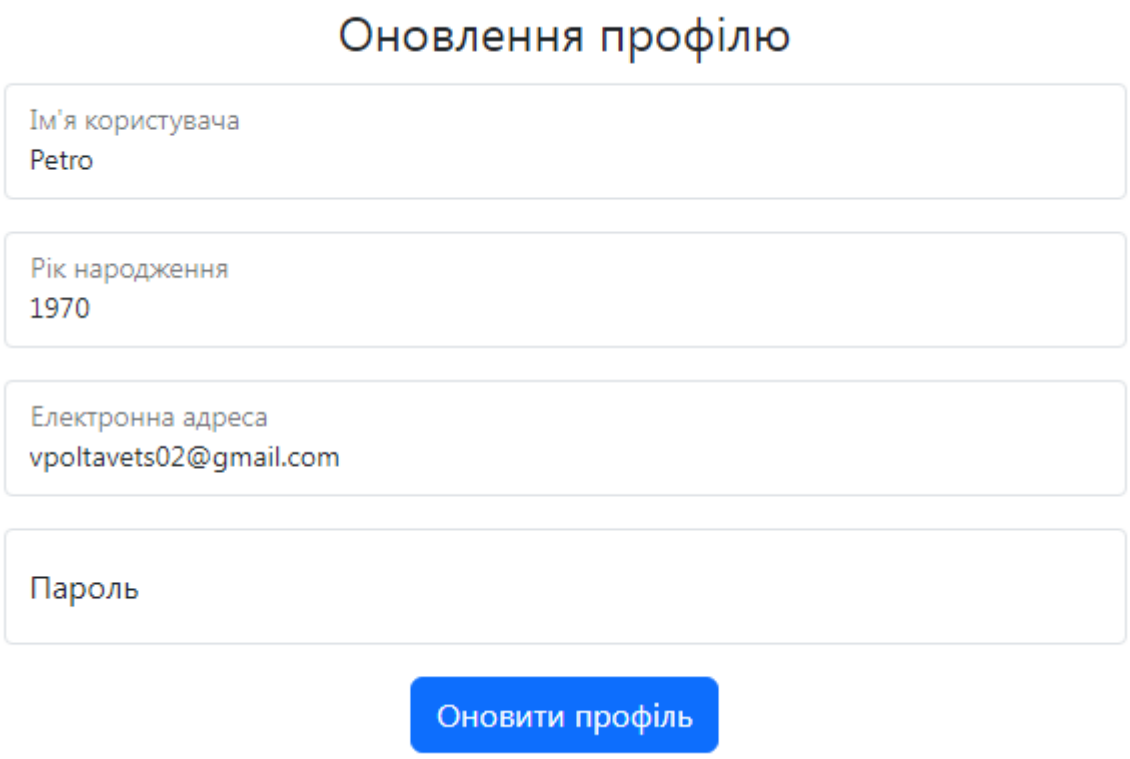


Рис. 2.29. Сторінка оновлення даних користувача

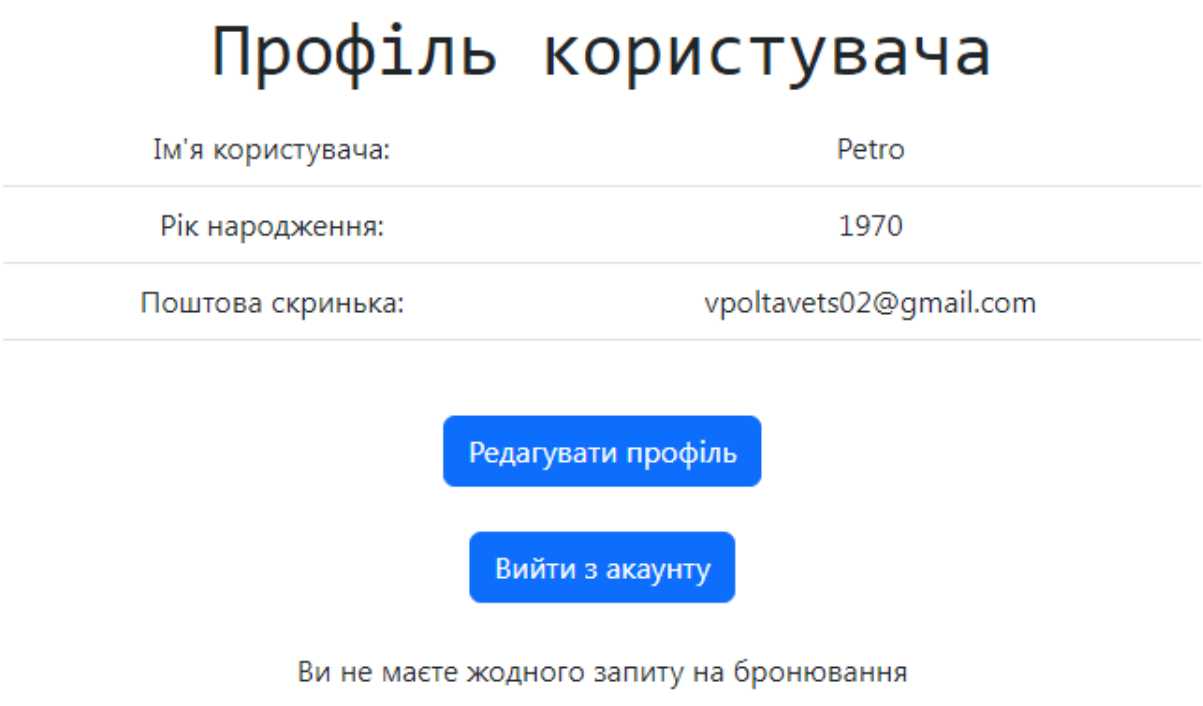


Рис. 2.30. Профіль після оновлення даних

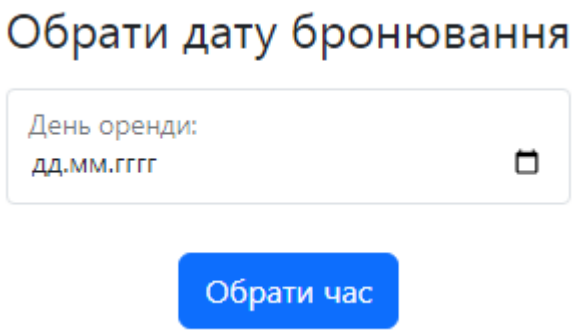


Рис. 2.31. Вибір дати для створення запиту на бронювання

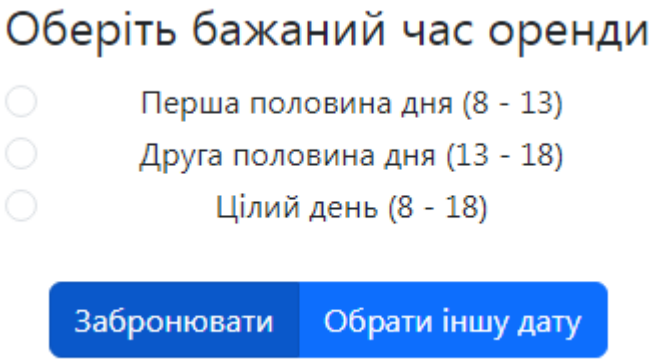


Рис. 2.32. Відображення доступного часу для бронювання

Після створення запиту на бронювання, цей запит з’являється у профілі користувача для сплати (рис. 2.33).

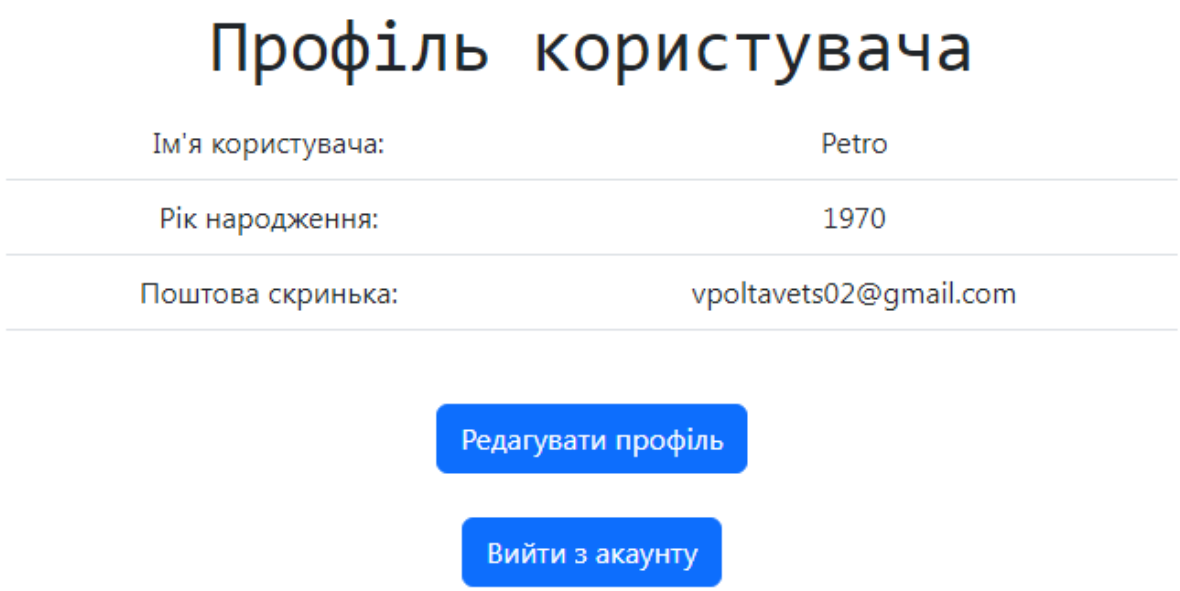


Рис. 2.33. Створений запит на бронювання у профілі користувача

Після сплати запиту, статус запиту зміниться на «Запит сплачено» (рис. 2.34).

Рис. 2.34. Відображення сплаченого запиту на бронювання

Далі сплачений запит відображається у користувачів з роллю «Admin» або «Developer» і вони підтверджують запит на бронювання (рис. 2.35).

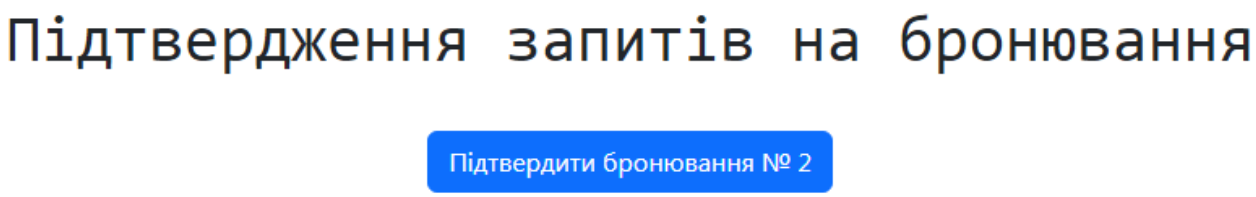


Рис. 2.35. Сторінка для підтвердження запитів на бронювання

Після отримання підтвердження від адміністратора, користувач отримує електронний лист про підтвердження запиту на бронювання (рис 2.36).

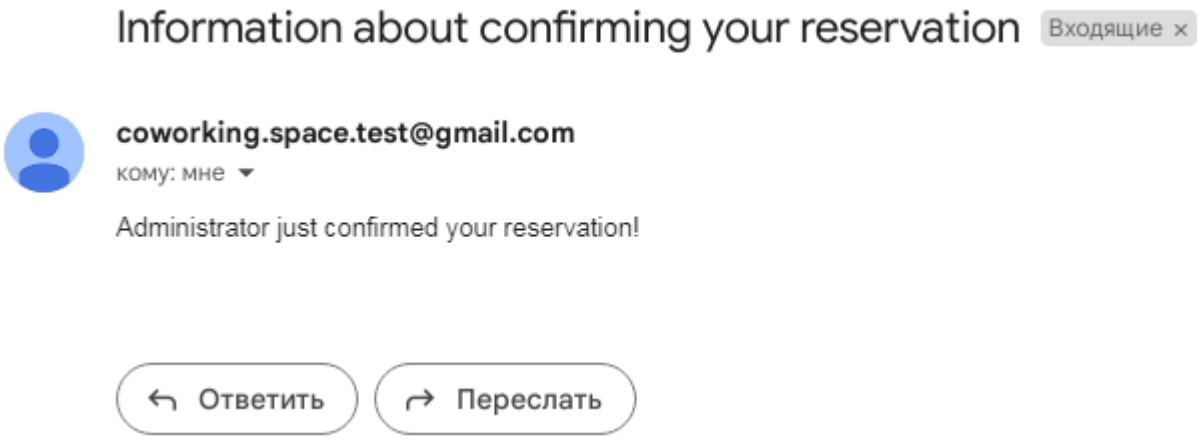


Рис. 2.36. Електронний лист про підтвердження запиту

Якщо наступає час оренди робочого місця, то веб-додаток самостійно змінює статус робочого місця на «Зайняте» (рис. 2.37).

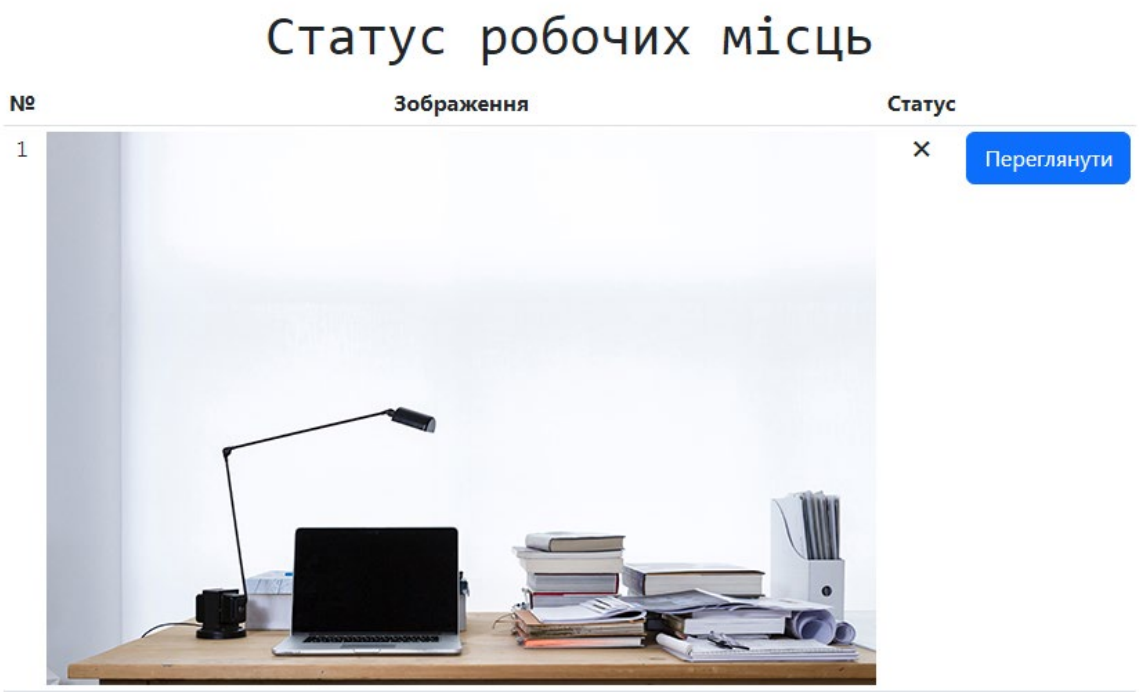


Рис. 2.37. Статус зайнятого робочого місця

Користувачі з ролями «Admin» та «Developer» мають додаткові функції у профілі (рис. 2.38). Перша з них підтвердження запитів на бронювання (рис. 2.35), друга – зміна ролі для користувачів з роллю «User» на «Admin» (рис. 2.39).

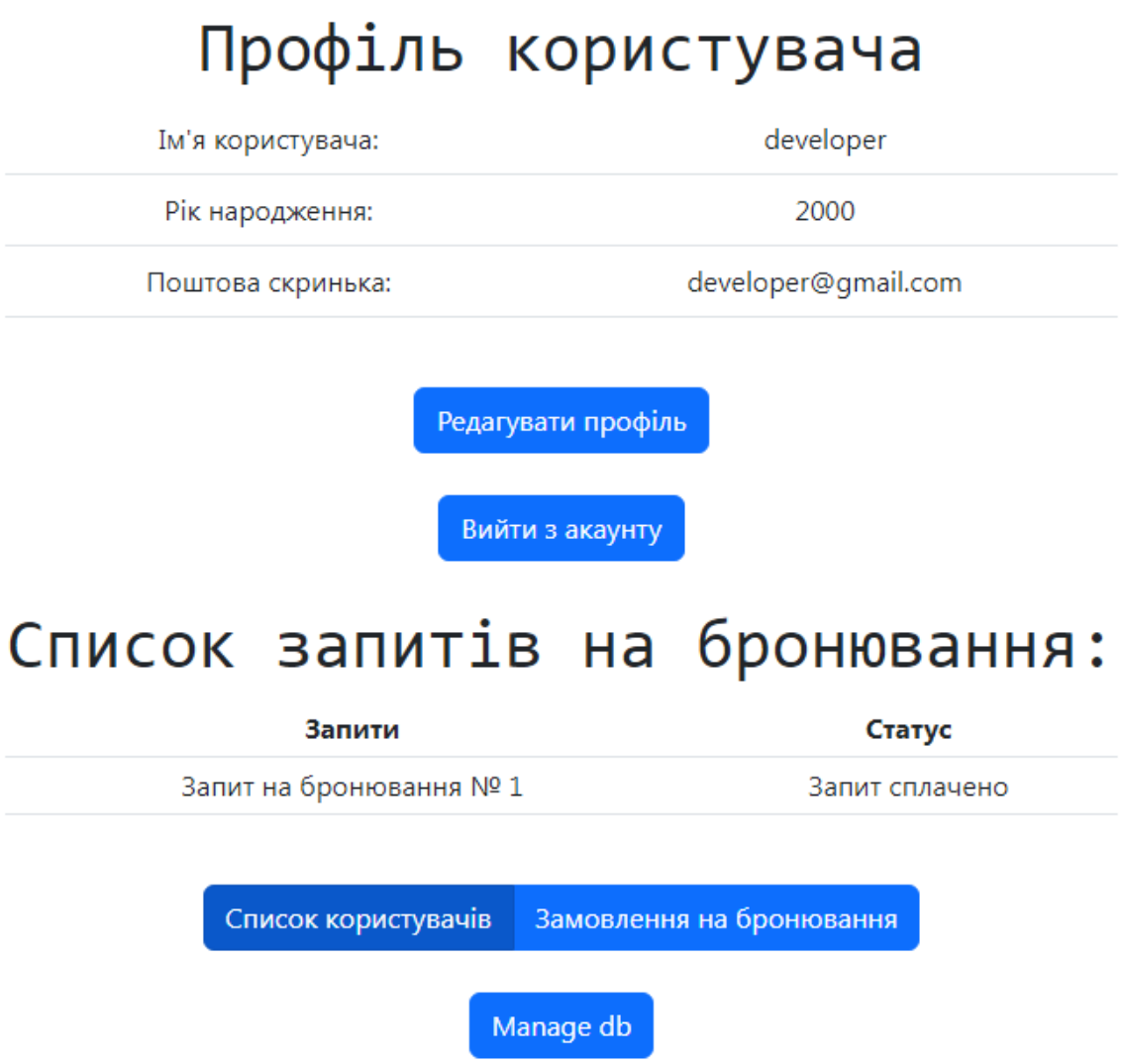


Рис. 2.38. Профіль користувача з роллю «Developer»

Рис. 2.39. Можливість зміни ролі для користувачів

Користувач з роллю «Developer» має на одну функціональну можливість більше ніж «Admin». «Developer» має доступ до консолі бази даних H2 та може вносити корективи в разі виникнення збоїв (рис. 2.40).

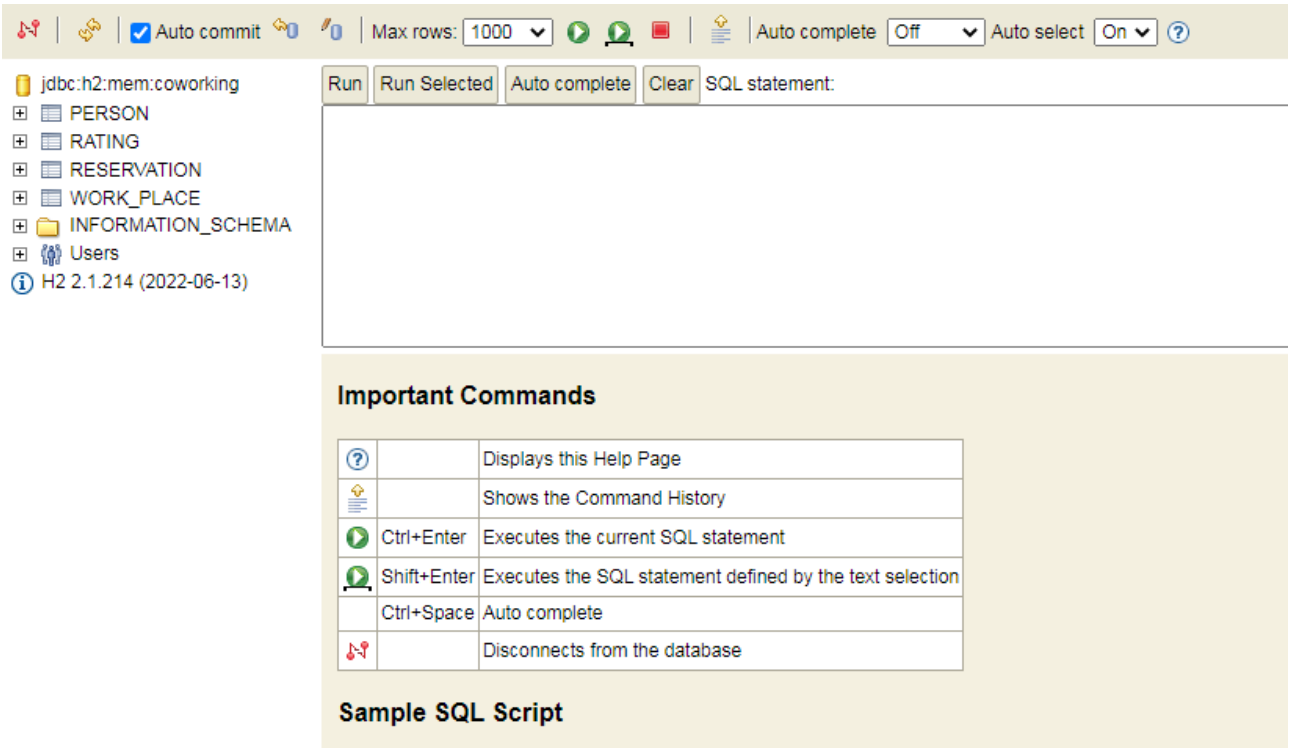


Рис. 2.40. Консоль бази даних

РОЗДІЛ 3 ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості розробки інформаційної системи

Початкові дані:

1. Передбачуване число операторів програми – 2050.
2. Коефіцієнт складності програми – 1,6.
3. Коефіцієнт корекції програми в ході її розробки – 0,06.
4. Годинна заробітна плата Java розробника – 168 грн/год.
5. Коефіцієнт збільшення витрати праці внаслідок недостатнього опису задачі – 1,2.
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,6.
7. Вартість машино-години ЕОМ – 13 грн/год.

Годинну заробітну плату Java розробника було розраховано за допомогою даних, узятих з сайту «Української спільноти програмістів (DOU.ua)» [21]. Середня заробітна плата Java розробника з досвідом роботи до року, по Україні, складає приблизно 800 американських доларів на місяць. Станом на зараз, початок червня 2023 року, один американський долар дорівнює 36,90 грн, виходячи з цього, середня заробітна плата у гривнях складає 29520 грн. При звичайному восьмигодинному робочому графіку (приблизно 176 годин на місяць) середня погодинна заробітна плата буде складати 167,7 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_o – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів:

$$Q = q \cdot C \cdot (1 + p) \quad (3.2)$$

де q - передбачуване число операторів (2050),

C - коефіцієнт складності програми (1,6),

p - коефіцієнт корекції програми в ході її розробки (0,06).

$$Q = 2050 \cdot 1{,}6 \cdot (1 + 0{,}06) = 3478$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин} \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з

даної спеціальності.

$$t_u = \frac{3478 \cdot 1{,}2}{80 \cdot 0{,}6} = 86{,}95 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин} \quad (3.4)$$

де Q – умовне число операторів програми,

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), людино-годин:

$$t_a = \frac{2050}{20 \cdot 0{,}6} = 171 \text{ людино-годин}$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин} \quad (3.5)$$

$$t_a = \frac{2050}{25 \cdot 0{,}6} = 137 \text{ людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}, \text{ людино-годин} \quad (3.6)$$

$$t_{отл} = \frac{2050}{5 \cdot 0{,}6} = 683 \text{ людино-годин}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}5 \cdot t_{отл}, \text{ людино-годин} \quad (3.7)$$

$$t_{отл}^{к} = 1{,}5 \cdot 683 = 1025 \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, \text{ людино-годин} \quad (3.8)$$

де $t_{др}$- трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15..20)\cdot к}, \text{ людино-годин} \quad (3.9)$$

$$t_{др} = \frac{2050}{15 \cdot 0{,}6} = 228 \text{ людино-годин}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин} \quad (3.10)$$

$$t_{до} = 0{,}75 \cdot 228 = 171 \text{ людино-годин}$$

$$t_{д} = 228 + 171 = 399 \text{ людино-годин}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 86{,}95 + 171 + 137 + 1025 + 399 = 1868{,}95 \text{ людино-годин}$$

У результаті ми розрахували, що в загальній складності необхідно 1868,95 людино-годин для розробки даного веб-додатку.

3.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ $К_{по}$включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{по} = З_{зп} + З_{мв}, \text{грн} \quad (3.11)$$

де $З_{зп}$ заробітна плата виконавців, яка визначається за формулою:

$$З_{зп} = t \cdot С_{пр}, \text{грн} \quad (3.12)$$

де t – загальна трудомісткість людино-годин,

$С_{пр}$ – середня годинна заробітна плата програміста, грн/год.

$$З_{зп} = 1868{,}95 \cdot 168 = 313983{,}6 \text{ грн}$$

$З_{мв}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ.

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{грн} \quad (3.13)$$

де $t_{отл}$– трудомісткість налагодження програми на ЕОМ, год,

$С_{мч}$– вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 1868{,}95 \cdot 13 = 24296{,}35 \text{ грн}$$

$$К_{по} = 313983{,}6 + 24296{,}35 = 338279{,}95 \text{ грн}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$Т = \frac{t}{В_к \cdot F_р}, \text{ міс} \quad (3.14)$$

де $В_к$ – число виконавців,

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні $F_p = 176$ годин).

$$Т = \frac{1868{,}95}{1 \cdot 176} = 10{,}62 \text{ міс}$$

Висновок: На розробку даного веб-додатку піде 1868,95 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 10,62 місяців при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення веб-додатку складатимуть 338279,95 грн.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення веб-додатку, який може бути використаний користувачами як на комп'ютерах так і на мобільних пристроях, матиме зручний, інтуїтивно зрозумілий інтерфейс та необхідні функціональні можливості.

В результаті виконання кваліфікаційної роботи був розроблений веб-додаток для coworking організації. Не дивлячись на вже існуючі та конкурентні додатки, розроблений додаток є актуальним, через наявність певних переваг перед аналогами таких як відсутність реклами, можливість перегляду та бронювання окремих робочих місць на певну дату та час.

Під час виконання кваліфікаційної роботи були виконанні наступні задачі:

1. Розробка алгоритму для вирішення задачі.
2. Написання програмного коду для веб-додатку.
3. Розробка моделі бази даних.

Створений веб-додаток надає наступні функціональні можливості:

1. Можливість реєстрації для нових користувачів та авторизації для вже існуючих.
2. Можливість перегляду та редагування профілю користувача.
3. Користувач має список замовлень та може оплачувати їх.
4. Розподілення користувачів за ролями.
5. Можливість перегляду стану робочих місць, станом на зараз, та рейтинг найпопулярніших робочих місць.

Веб-додаток розроблений з використанням мови програмування Java та WEB-орієнтованого фреймворку Spring.

В ході кваліфікаційної роботи було вирахувано трудомісткість та порахована приблизна вартість розробленого додатку. Загальна вартість програмного продукту становить 338279,95 грн, а час створення - 1868,95 годин або 10,62 місяці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Coworking / URL: https://en.wikipedia.org/wiki/Coworking
2. Коворкінг від А до Я: все, що потрібно знати про гнучкий робочий простір / URL: https://peremoga.space/kovorkinh-vid-a-do-ia-vse-shcho-potribno-znaty-pro-hnuchkyj-robochyj-prostir-peremoga-space/
3. Коворкінг (Co-working) – що це таке, навіщо потрібен та як працює / URL: https://termin.in.ua/kovorkinh-co-working/
4. Coworking365 / URL: https://coworking365.ua/
5. Рейтинг мов програмування 2023 / URL: https://dou.ua/lenta/articles/language-rating-2023/
6. Java (Programing language) / URL: https://en.wikipedia.org/wiki/Java_(programming_language)
7. Overview of Spring Framework / URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html
8. Web MVC Framework / URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html
9. Spring Security / URL: https://spring.io/projects/spring-security
10. Spring Data JPA / URL: https://spring.io/projects/spring-data-jpa
11. Spring Boot Introduction / URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm
12. Project Lombok / URL: https://projectlombok.org/
13. HTML / URL: https://uk.wikipedia.org/wiki/HTML
14. Java Script / URL: https://uk.wikipedia.org/wiki/JavaScript
15. CSS / URL: https://uk.wikipedia.org/wiki/CSS
16. Thymeleaf / URL: https://www.thymeleaf.org/
17. What is Bootstrap? / URL: https://www.techtarget.com/whatis/definition/bootstrap
18. Apache Maven / URL: https://maven.apache.org/

19. Hibernate Framework / URL: https://www.javatpoint.com/hibernate-tutorial#:~:text=Hibernate%20is%20a%20Java%20framework,Persistence%20API)%20for%20data%20persistence.

20. H2 Database – Introduction / URL: https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm

21. dou.ua зарплати / URL: https://jobs.dou.ua/salaries/?period=2022-12&position=Junior%20SE&technology=Java&experience=0-1&english=1-3

ДОДАТОК А

КОД ПРОГРАМИ

schema.sql

```
create table person
(
   id           long auto_increment primary key,
   email        varchar(100) unique not null,
   username     varchar(100) unique not null,
   password     varchar            not null,
   role         varchar            not null,
   year_of_birth int check (year_of_birth > 1915)
);

create table work_place
(
   id          long auto_increment primary key,
   item_name   varchar(100) not null,
   description  text        not null,
   is_available boolean default true,
   img_url     varchar     not null
);

create table reservation
(
   id           long auto_increment primary key,
   work_place_id int references work_place (id) on delete cascade,
   renter_id    int references person (id) on delete cascade,
   rent_day     date check (rent_day >= current_date) not null,
   time_from    time check (time_from >= '08:00:00')  not null,
   time_to      time check (time_to <= '18:00:00')    not null,
   is_payed     boolean default false,
   is_confirmed  boolean default false,
   is_actual    boolean default true,
   total_price   decimal                    not null,
   foreign key (work_place_id) references work_place (id),
   foreign key (renter_id) references person (id)
);

create table rating
(
   id             long auto_increment primary key,
   work_place_id   long references work_place (id) on delete cascade,
   number_of_using int not null default 0,
   foreign key (work_place_id) references work_place (id)
);
```

data.sql

```
insert into person(email, username, password, role, year_of_birth)
values                                    ('developer@gmail.com',                    'developer',
'$2a$10$jG5C252jUtKbdm/AGGl4yub/qbIStoOOm80bUIqo4MHddNabD0lJy',
      'ROLE_DEVELOPER', 2000);

insert into person(email, username, password, role, year_of_birth)
values                                      ('admin@gmail.com',                      'admin',
'$2a$10$ZxrvhkIoyFCQ7WXqE4tXJuq5AZYKHDTakWIBM8YFcSF4Jlhr2C2Km',
      'ROLE_ADMIN', 2000);

insert into person(email, username, password, role, year_of_birth)
```

```
values ('demo06478@gmail.com', 'user',
'$2a$10$2jKKw/RZhuLSX8DnjVK3QeFxHCjZQzphUTfTP5/chWYdcFEewA3OS',
    'ROLE_USER', 2000);

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 1', 'Комфортний стіл та стілець, настільна лампа для роботи', '/images/workPlace1.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 2', 'Комфортний стіл та стілець для роботи', '/images/workPlace2.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 3', 'Комфортний стіл та стілець для роботи', '/images/workPlace3.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 4', 'Комфортний стіл та стілець для роботи', '/images/workPlace4.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 5', 'Комфортний стіл та стілець для роботи', '/images/workPlace5.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 6', 'Комфортний стіл та стілець для роботи', '/images/workPlace6.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 7', 'Комфортний стіл та стілець для роботи', '/images/workPlace7.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 8', 'Комфортний стіл та стілець для роботи', '/images/workPlace8.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 9', 'Комфортний стіл та стілець для роботи', '/images/workPlace9.jpg');

insert into work_place(item_name, description, img_url)
values ('Робоче місце № 10', 'Комфортний стіл та стілець для роботи', '/images/workPlace10.jpg');

insert into reservation(work_place_id, renter_id, rent_day, time_from, time_to, total_price, is_payed, is_confirmed)
values (1, 3, current_date, current_time + 1 minute, current_time + 2 minute, 100, true, true);

insert into rating(work_place_id) values (1);
insert into rating(work_place_id, number_of_using) values (2, 1);
insert into rating(work_place_id, number_of_using) values (3, 1);
insert into rating(work_place_id, number_of_using) values (4, 3);
insert into rating(work_place_id, number_of_using) values (5, 1);
insert into rating(work_place_id, number_of_using) values (6, 5);
insert into rating(work_place_id) values (7);
insert into rating(work_place_id) values (8);
insert into rating(work_place_id, number_of_using) values (9, 9);
insert into rating(work_place_id, number_of_using) values (10, 10);
```

application.properties

```
#h2 db properties
spring.h2.console.enabled=true
spring.datasource.generate-unique-name=false
spring.datasource.name=coworking
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.show_sql=true
spring.mvc.hiddenmethod.filter.enabled=true
#spring mail
spring.mail.host=smtp.gmail.com
spring.mail.port=${port}
spring.mail.username=${username}
spring.mail.password=${password}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

MvcConfig.java

```
package coworking.project.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      if (!registry.hasMappingForPattern("/static/**")) {
         registry.addResourceHandler("/static/**")
               .addResourceLocations("/static/");
      }
   }
}
```

SecurityConfig.java

```
package coworking.project.config;

import coworking.project.services.PersonDetailsService;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.ProviderManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

import java.util.Collections;
import java.util.List;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
   private final PersonDetailsService personDetailsService;

   public SecurityConfig(PersonDetailsService personDetailsService) {
      this.personDetailsService = personDetailsService;
   }

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .csrf()
            .ignoringAntMatchers("/h2-console/**")
            .and()
            .headers()
            .frameOptions()
            .sameOrigin()
            .and()
            .authorizeRequests()
            .antMatchers("/h2-console/**")
            .hasRole("DEVELOPER")
            .antMatchers("/admin")
```

```
            .hasAnyRole("ADMIN", "DEVELOPER")
            .antMatchers("/", "/error", "/auth/signup", "/auth/login", "/workPlaces",
                  "/workPlaces/rating", "/workPlaces/status", "/images/**", "/info/pricing",
                  "/info/aboutUs")
            .permitAll()
            .anyRequest()
            .hasAnyRole("USER", "ADMIN", "DEVELOPER")
            .and()
            .formLogin()
            .loginPage("/auth/login")
            .loginProcessingUrl("/process_login")
            .defaultSuccessUrl("/", true)
            .failureUrl("/auth/login?error")
            .and()
            .logout()
            .logoutUrl("/logout")
            .logoutSuccessUrl("/");
        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(personDetailsService);
        authProvider.setPasswordEncoder(getPasswordEncoder());

        List<AuthenticationProvider> providers = Collections.singletonList(authProvider);

        return new ProviderManager(providers);
    }

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

AdminController.java

```
package coworking.project.controllers;

import coworking.project.dto.PersonMapper;
import coworking.project.models.Person;
import coworking.project.services.AdminService;
import coworking.project.services.PeopleService;
import coworking.project.services.ReservationService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.stream.Collectors;

@Controller
@RequestMapping("/admin")
public class AdminController {
    private final AdminService adminService;
    private final PersonMapper personMapper;
    private final PeopleService peopleService;
    private final ReservationService reservationService;
```

```
    public AdminController(AdminService adminService, PersonMapper personMapper, PeopleService peopleService,
ReservationService reservationService) {
        this.adminService = adminService;
        this.personMapper = personMapper;
        this.peopleService = peopleService;
        this.reservationService = reservationService;
    }

    @GetMapping
    public String getAdminPage(Model model) {
        model.addAttribute("users", adminService.findAll()
                .stream()
                .map(personMapper::convertToPersonDTO).collect(Collectors.toList()));
        return "admin/admin";
    }

    @GetMapping("/reservations")
    public String getReservations(Model model) {
        model.addAttribute("reservationsToConfirm", reservationService.findPayedReservations());
        return "admin/reservations";
    }

    @GetMapping("/reservations/{id}")
    public String confirmReservation(@PathVariable("id") Long id) {
        adminService.confirmReservation(id);
        return "redirect:/admin/reservations";
    }

    @PatchMapping("/{id}")
    public String setAdminRole(@PathVariable Long id) {
        Person person = peopleService.findById(id);
        person.setRole("ROLE_ADMIN");
        peopleService.update(id, person);
        return "redirect:/admin";
    }
}
```

AuthorizationController.java

```
package coworking.project.controllers;

import coworking.project.dto.PersonDTO;
import coworking.project.dto.PersonMapper;
import coworking.project.models.Person;
import coworking.project.services.AuthorizationService;
import coworking.project.util.PersonValidator;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.validation.Valid;

@Controller
@RequestMapping("/auth")
public class AuthorizationController {
    private final PersonMapper personMapper;
    private final PersonValidator personValidator;
    private final AuthorizationService authorizationService;
```

```
    public AuthorizationController(PersonMapper personMapper, PersonValidator personValidator,
AuthorizationService authorizationService) {
        this.personMapper = personMapper;
        this.personValidator = personValidator;
        this.authorizationService = authorizationService;
    }

    @GetMapping("/login")
    public String getLoginPage(Model model) {
        model.addAttribute("title", "Логін");
        return "auth/login";
    }

    @GetMapping("/signup")
    public String getSignupPage(@ModelAttribute("person") PersonDTO person, Model model) {
        model.addAttribute("title", "Створення акаунту");
        return "auth/signup";
    }

    @PostMapping("/signup")
    public String getRegisteredPerson(@ModelAttribute("person") @Valid PersonDTO personDTO,
                                      BindingResult bindingResult) {
        Person person = personMapper.convertToPerson(personDTO);
        personValidator.validate(person, bindingResult);
        if (bindingResult.hasErrors())
            return "auth/signup";
        authorizationService.save(person);
        return "redirect:/home";
    }
}
```

HomeController.java

```
package coworking.project.controllers;

import coworking.project.services.ProfileService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {
    private final ProfileService profileService;

    public HomeController(ProfileService profileService) {
        this.profileService = profileService;
    }


    @GetMapping()
    public String getHomePage(Model model) {
        model.addAttribute("title", "Coworking Space");
        model.addAttribute("user", profileService.getPerson());
        return "index";
    }
}
```

InfoController.java

```
package coworking.project.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
@RequestMapping("/info")
public class InfoController {

    @GetMapping("/pricing")
    public String getPricingPage(Model model) {
        model.addAttribute("title", "Тарифи");
        return "info/pricing";
    }

    @GetMapping("/aboutUs")
    public String getAboutUsPage(Model model) {
        model.addAttribute("title", "Про організацію");
        return "info/aboutUs";
    }
}
```

ProfileController.java

```
package coworking.project.controllers;

import coworking.project.dto.PersonDTO;
import coworking.project.dto.PersonMapper;
import coworking.project.dto.PersonUpdateDTO;
import coworking.project.dto.PersonUpdateMapper;
import coworking.project.models.Person;
import coworking.project.services.PeopleService;
import coworking.project.services.ProfileService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.validation.Valid;

@Controller
@RequestMapping("/profile")
public class ProfileController {
    private final PersonMapper personMapper;
    private final PeopleService peopleService;
    private final ProfileService profileService;
    private final PersonUpdateMapper personUpdateMapper;

    public ProfileController(PersonMapper personMapper, PeopleService peopleService, ProfileService profileService,
PersonUpdateMapper personUpdateMapper) {
        this.personMapper = personMapper;
        this.peopleService = peopleService;
        this.profileService = profileService;
        this.personUpdateMapper = personUpdateMapper;
    }

    @GetMapping()
    public String getUserProfile(Model model) {
        Person person = profileService.getPerson();
        model.addAttribute("person", personMapper.convertToPersonDTO(person));
        return "user/profile";
    }

    @GetMapping("/edit")
    public String editUserProfile(Model model) {
```

```
        Person person = profileService.getPerson();
        model.addAttribute("person", personUpdateMapper.convertToPersonUpdateDTO(person));
        return "user/edit";
    }

    @PatchMapping("/edit")
    public String saveChanges(@ModelAttribute("person") @Valid PersonUpdateDTO personUpdateDTO,
                              BindingResult bindingResult) {
        if (bindingResult.hasErrors())
            return "user/edit";
        Person person = personUpdateMapper.convertToPerson(personUpdateDTO);
        peopleService.update(person.getId(), person);
        return "redirect:/profile";
    }
}
```

ReservationController.java

```
package coworking.project.controllers;

import coworking.project.services.EmailService;
import coworking.project.services.ReservationService;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/reservations")
public class ReservationController {
    private final EmailService emailService;
    private final ReservationService reservationService;

    public ReservationController(EmailService emailService, ReservationService reservationService) {
        this.emailService = emailService;
        this.reservationService = reservationService;
    }

    @PatchMapping("/{id}")
    public String payReservation(@PathVariable("id") Long id) {
        reservationService.payReservation(id);
        emailService.sendSuccessfulPaymentMessage(reservationService.findById(id).getRenter().getEmail());
        return "redirect:/profile";
    }
}
```

WorkPlaceController.java

```
package coworking.project.controllers;

import coworking.project.dto.ReservationDTO;
import coworking.project.dto.ReservationMapper;
import coworking.project.dto.WorkPlaceMapper;
import coworking.project.models.Reservation;
import coworking.project.services.RatingService;
import coworking.project.services.ReservationService;
import coworking.project.services.WorkPlaceService;
import coworking.project.util.ReservationValidator;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.stream.Collectors;
```

```
@Controller
@RequestMapping("/workPlaces")
public class WorkPlaceController {
    private final RatingService ratingService;
    private final WorkPlaceMapper workPlaceMapper;
    private final WorkPlaceService workPlaceService;
    private final ReservationMapper reservationMapper;
    private final ReservationService reservationService;
    private final ReservationValidator reservationValidator;

    public WorkPlaceController(RatingService ratingService, WorkPlaceMapper workPlaceMapper, WorkPlaceService
workPlaceService, ReservationMapper
            reservationMapper, ReservationService reservationService, ReservationValidator reservationValidator) {
        this.ratingService = ratingService;
        this.workPlaceMapper = workPlaceMapper;
        this.workPlaceService = workPlaceService;
        this.reservationMapper = reservationMapper;
        this.reservationService = reservationService;
        this.reservationValidator = reservationValidator;
    }

    @GetMapping()
    public String getWorkPlaces(Model model) {
        model.addAttribute("title", "Каталог робочих місць");
        model.addAttribute("workPlaces", workPlaceService.findAll()
                .stream()
                .map(workPlaceMapper::convertToWorkPlaceDTO)
                .collect(Collectors.toList()));
        return "work_places/main";
    }

    @GetMapping("/{id}")
    public String getWorkPlace(@PathVariable("id") Long id, Model model) {
        model.addAttribute("title", "Робоче місце № " + id);
        model.addAttribute("workPlace", workPlaceMapper.convertToWorkPlaceDTO(workPlaceService.findById(id)));
        return "work_places/show";
    }

    @GetMapping("/{id}/reservation")
    public String getReservationForm(@PathVariable("id") Long id, Model model,
                                     @ModelAttribute("reservation") ReservationDTO reservationDTO) {
        model.addAttribute("title", "Вибір дати");
        model.addAttribute("workPlace",
                workPlaceMapper.convertToWorkPlaceDTO(workPlaceService.findById(id)));
        return "reservations/chooseDate";
    }

    @GetMapping("/{id}/reservation/new")
    public String createNewReservation(@PathVariable("id") Long id, Model model,
                                       @ModelAttribute("reservation") @Valid ReservationDTO reservationDTO
            , BindingResult bindingResult) {
        model.addAttribute("title", "Створення запиту на бронювання");
        model.addAttribute("workPlace",
                workPlaceMapper.convertToWorkPlaceDTO(workPlaceService.findById(id)));
        reservationValidator.validate(reservationDTO, bindingResult);
        if (bindingResult.hasErrors())
            return "/reservations/chooseDate";
        reservationMapper.updateReservationDTO(reservationDTO, id, reservationDTO.getRentDay());
        model.addAttribute("reservation", reservationDTO);
        return "reservations/createReservation";
    }
```

```
    @PostMapping("/{id}/reservation/new")
    public String saveReservation(@PathVariable("id") Long id, @ModelAttribute("reservation") ReservationDTO
            reservationDTO) {
        Reservation reservation = reservationMapper.convertToReservation(reservationDTO);
        reservationMapper.setOtherValues(reservation, reservationDTO, id);
        reservationService.save(reservation);
        return "redirect:/";
    }

    @GetMapping("/status")
    public String getStatistic(Model model) {
        model.addAttribute("title", "Статус робочих місць");
        model.addAttribute("list", workPlaceService.findAll());
        return "work_places/work_places_now";
    }

    @GetMapping("/rating")
    public String getRating(Model model) {
        model.addAttribute("title", "Топ-5");
        model.addAttribute("rating", ratingService.findTop5ByOrderByNumberOfUsingDesc());
        return "work_places/rating";
    }
}
```

PersonDTO.java

```
package coworking.project.dto;

import coworking.project.models.Reservation;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.Id;
import javax.validation.constraints.Email;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Size;
import java.util.List;

@Getter
@Setter
@NoArgsConstructor
@ToString
public class PersonDTO {
    private Long id;

    @Email
    private String email;

    @NotEmpty
    @Size(min = 2, max = 100)
    private String username;

    @Size(min = 8)
    private String password;

    private String role;

    @Min(value = 1915)
    private Integer yearOfBirth;

    private List<Reservation> reservations;
```

```
}
```

PersonMapper.java

```
package coworking.project.dto;

import coworking.project.models.Person;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
public class PersonMapper {
   private final ModelMapper modelMapper;

   public PersonMapper(ModelMapper modelMapper) {
      this.modelMapper = modelMapper;
   }

   public Person convertToPerson(PersonDTO personDTO) {
      return modelMapper.map(personDTO, Person.class);
   }

   public PersonDTO convertToPersonDTO(Person person) {
      return modelMapper.map(person, PersonDTO.class);
   }
}
```

PersonUpdateDTO.java

```
package coworking.project.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.validation.constraints.Email;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Size;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class PersonUpdateDTO {
   @NotEmpty
   @Size(min = 2, max = 100)
   private String username;
   @Size(min = 8)
   private String password;
   @Email
   @NotEmpty
   private String email;
   @Min(value = 1915)
   private Integer yearOfBirth;
}
```

PersonUpdateMapper.java

```
package coworking.project.dto;

import coworking.project.models.Person;
import coworking.project.services.ProfileService;
import org.modelmapper.ModelMapper;
import org.springframework.security.crypto.password.PasswordEncoder;
```

```
import org.springframework.stereotype.Component;

@Component
public class PersonUpdateMapper {
    private final ModelMapper modelMapper;
    private final ProfileService profileService;
    private final PasswordEncoder passwordEncoder;


    public PersonUpdateMapper(ModelMapper modelMapper, ProfileService profileService, PasswordEncoder
passwordEncoder) {
        this.modelMapper = modelMapper;
        this.profileService = profileService;
        this.passwordEncoder = passwordEncoder;
    }

    public Person convertToPerson(PersonUpdateDTO personUpdateDTO) {
        Person person = modelMapper.map(personUpdateDTO, Person.class);
        person.setId(profileService.getPerson().getId());
        person.setRole(profileService.getPerson().getRole());
        person.setPassword(passwordEncoder.encode(personUpdateDTO.getPassword()));
        return person;
    }

    public PersonUpdateDTO convertToPersonUpdateDTO(Person person) {
        return modelMapper.map(person, PersonUpdateDTO.class);
    }
}
```

ReservationDTO.java

```
package coworking.project.dto;

import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;

import java.time.LocalDate;

@Data
@NoArgsConstructor
public class ReservationDTO {
    private String time;
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate rentDay;
}
```

ReservationMapper.java

```
package coworking.project.dto;

import coworking.project.models.Person;
import coworking.project.models.Rating;
import coworking.project.models.Reservation;
import coworking.project.models.WorkPlace;
import coworking.project.services.ProfileService;
import coworking.project.services.RatingService;
import coworking.project.services.ReservationService;
import coworking.project.services.WorkPlaceService;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.Collections;
```

```
import java.util.List;

@Component
public class ReservationMapper {
    private final ModelMapper modelMapper;
    private final ProfileService profileService;
    private final WorkPlaceService workPlaceService;
    private final ReservationService reservationService;

    public ReservationMapper(ModelMapper modelMapper, ProfileService profileService, WorkPlaceService
workPlaceService, ReservationService reservationService) {
        this.modelMapper = modelMapper;
        this.profileService = profileService;
        this.workPlaceService = workPlaceService;
        this.reservationService = reservationService;
    }

    public Reservation convertToReservation(ReservationDTO reservationDTO) {
        return modelMapper.map(reservationDTO, Reservation.class);
    }

    public ReservationDTO convertToReservationDTO(Reservation reservation) {
        return modelMapper.map(reservation, ReservationDTO.class);
    }

    public void updateReservationDTO(ReservationDTO reservationDTO, Long workPlaceId, LocalDate date) {
        List<Reservation> reservations = reservationService.findAllByWorkPlaceAndRentDay(workPlaceId, date);
        if (!reservations.isEmpty()) {
            if (reservations.size() == 2) {
                reservationDTO.setTime("full");
            }
            if (reservationDTO.getTime() == null) {
                for (Reservation reservation : reservations) {
                    if (reservation.getTimeFrom().getHour() == 8 && reservation.getTimeTo().getHour() == 13)
                        reservationDTO.setTime("first");
                    if (reservation.getTimeFrom().getHour() == 13 && reservation.getTimeTo().getHour() == 18)
                        reservationDTO.setTime("second");
                    if (reservation.getTimeFrom().getHour() == 8 && reservation.getTimeTo().getHour() == 18)
                        reservationDTO.setTime("full");
                }
            }
        }
    }

    public void setOtherValues(Reservation reservation, ReservationDTO reservationDTO, Long workPlaceId) {
        Person person = profileService.getPerson();
        WorkPlace workPlace = workPlaceService.findById(workPlaceId);
        reservation.setIsPayed(false);
        reservation.setIsConfirmed(false);
        reservation.setIsActual(true);
        reservation.setWorkPlace(workPlace);
        reservation.setRenter(person);
        setTime(reservationDTO, reservation);
        saveData(person, workPlace, reservation);
    }

    private void setTime(ReservationDTO reservationDTO, Reservation reservation) {
        if (reservationDTO.getTime().equals("first")) {
            reservation.setTimeFrom(LocalTime.of(8, 0));
            reservation.setTimeTo(LocalTime.of(13, 0));
            reservation.setPriceTotal(500.0);
        } else if (reservationDTO.getTime().equals("second")) {
            reservation.setTimeFrom(LocalTime.of(13, 0));
```

```
            reservation.setTimeTo(LocalTime.of(18, 0));
            reservation.setPriceTotal(500.0);
        } else {
            reservation.setTimeFrom(LocalTime.of(8, 0));
            reservation.setTimeTo(LocalTime.of(18, 0));
            reservation.setPriceTotal(400.0);
        }
    }

    private void saveData(Person person, WorkPlace workPlace, Reservation reservation) {
        if (person.getReservations().isEmpty())
            person.setReservations(Collections.singletonList(reservation));
        else
            person.getReservations().add(reservation);
        if (workPlace.getReservations().isEmpty())
            workPlace.setReservations(Collections.singletonList(reservation));
        else
            workPlace.getReservations().add(reservation);
    }
}
```

WorkPlaceDTO.java

```
package coworking.project.dto;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class WorkPlaceDTO {
    private Long id;
    private String itemName;
    private String description;
    private String imgUrl;
}
```

WorkPlaceMapper.java

```
package coworking.project.dto;

import coworking.project.models.WorkPlace;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
public class WorkPlaceMapper {
    private final ModelMapper modelMapper;

    public WorkPlaceMapper(ModelMapper modelMapper) {
        this.modelMapper = modelMapper;
    }

    public WorkPlace convertToWorkPlace(WorkPlaceDTO workPlaceDTO) {
        return modelMapper.map(workPlaceDTO, WorkPlace.class);
    }

    public WorkPlaceDTO convertToWorkPlaceDTO(WorkPlace workPlace) {
        return modelMapper.map(workPlace, WorkPlaceDTO.class);
    }
}
```

PersonNotFoundException.java

```
package coworking.project.exceptions;
```

```
public class PersonNotFoundException extends RuntimeException { }
```

ReservationNotFoundException.java

```
package coworking.project.exceptions;

public class ReservationNotFoundException extends RuntimeException { }
```

WorkPlaceNotFoundException.java

```
package coworking.project.exceptions;

public class WorkPlaceNotFoundException extends RuntimeException { }
```

Person.java

```
package coworking.project.models;

import lombok.*;

import javax.persistence.*;
import javax.validation.constraints.*;
import java.util.List;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "PERSON")
public class Person {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Email
    @NotEmpty
    @Column(name = "email", unique = true)
    private String email;

    @NotEmpty
    @Size(min = 2, max = 100)
    @Column(name = "username", unique = true)
    private String username;

    @Size(min = 8)
    @Column(name = "password")
    private String password;

    @Column(name = "role")
    private String role;

    @Min(value = 1915)
    @Column(name = "year_of_birth")
    private Integer yearOfBirth;

    @OneToMany(mappedBy = "renter", fetch = FetchType.EAGER)
    private List<Reservation> reservations;

    @Override
    public String toString() {
        return "Person{" +
                "id=" + id +
                ", email='" + email + '\'' +
                ", username='" + username + '\'' +
```

```
                ", password='" + password + '\'' +
                ", role='" + role + '\'' +
                ", yearOfBirth=" + yearOfBirth +
                '}';
    }
}
```

Rating.java

```
package coworking.project.models;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "RATING")
public class Rating {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "number_of_using")
    private Integer numberOfUsing;

    @OneToOne
    @JoinColumn(name = "work_place_id")
    private WorkPlace workPlace;
}
```

Reservation.java

```
package coworking.project.models;

import lombok.*;
import org.springframework.format.annotation.DateTimeFormat;

import javax.persistence.*;
import java.sql.Date;
import java.sql.Time;
import java.time.LocalDate;
import java.time.LocalTime;

@Getter
@Setter
@AllArgsConstructor
@Entity
@NoArgsConstructor
@Table(name = "RESERVATION")
public class Reservation {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "time_from")
    @DateTimeFormat(pattern = "HH:mm")
```

```
    private LocalTime timeFrom;

    @Column(name = "time_to")
    @DateTimeFormat(pattern = "HH:mm")
    private LocalTime timeTo;

    @Column(name = "rent_day")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate rentDay;

    @Column(name = "is_payed")
    private Boolean isPayed;

    @Column(name = "is_confirmed")
    private Boolean isConfirmed;

    @Column(name = "is_actual")
    private Boolean isActual;

    @Column(name = "total_price")
    private Double priceTotal;

    @ManyToOne
    @JoinColumn(name = "work_place_id")
    private WorkPlace workPlace;

    @ManyToOne
    @JoinColumn(name = "renter_id")
    private Person renter;

    @Override
    public String toString() {
        return "Reservation{" +
                "id=" + id +
                ", timeFrom=" + timeFrom +
                ", timeTo=" + timeTo +
                ", rentDay=" + rentDay +
                ", isPayed=" + isPayed +
                ", isConfirmed=" + isConfirmed +
                ", priceTotal=" + priceTotal +
                '}';
    }
}
```

WorkPlace.java

```
package coworking.project.models;

import lombok.*;

import javax.persistence.*;
import java.util.List;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "WORK_PLACE")
public class WorkPlace {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @Column(name = "item_name")
    private String itemName;

    @Column(name = "description")
    private String description;

    @Column(name = "is_available")
    private Boolean isAvailable;

    @Column(name = "img_url")
    private String imgUrl;

    @OneToMany(mappedBy = "workPlace", fetch = FetchType.EAGER)
    private List<Reservation> reservations;

    @OneToOne(mappedBy = "workPlace")
    private Rating rating;

    @Override
    public String toString() {
        return "WorkPlace{" +
                "id=" + id +
                ", itemName='" + itemName + '\'' +
                ", description='" + description + '\'' +
                '}';
    }
}
```

PeopleRepository.java

```
package coworking.project.repositories;

import coworking.project.models.Person;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface PeopleRepository extends JpaRepository<Person, Long> {
    Optional<Person> findByUsername(String username);

    Optional<Person> findByEmail(String email);

    void deleteById(Long id);
}
```

RatingRepository.java

```
package coworking.project.repositories;

import coworking.project.models.Rating;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface RatingRepository extends JpaRepository<Rating, Long> {
    List<Rating> findTop5ByOrderByNumberOfUsingDesc();
}
```

ReservationsRepository.java

```
package coworking.project.repositories;
```

```
import coworking.project.models.Reservation;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
import java.util.List;

@Repository
public interface ReservationsRepository extends JpaRepository<Reservation, Long> {
   @Query(value = "SELECT * FROM  reservation WHERE is_payed = true AND is_confirmed = false", nativeQuery
= true)
   List<Reservation> findPayedReservationsToConfirm();

   @Query(value = "SELECT * FROM reservation WHERE is_payed=true AND is_confirmed=true " +
         "AND rent_day=CURRENT_DATE() AND is_actual=true", nativeQuery = true)
   List<Reservation> findAllConfirmedReservations();

   @Query(value = "SELECT * FROM reservation r WHERE r.work_place_id = :work_place_id " +
         "AND r.rent_day = :rent_day", nativeQuery = true)
   List<Reservation> findAllByRentDay(@Param("work_place_id") Long workPlaceId,
                              @Param("rent_day") LocalDate date);
}
```

WorkPlacesRepository.java

```
package coworking.project.repositories;

import coworking.project.models.WorkPlace;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface WorkPlacesRepository extends JpaRepository<WorkPlace, Long> {
   @Query(value = "SELECT * FROM work_place WHERE is_available = true", nativeQuery = true)
   List<WorkPlace> findAvailableWorkPlaces();
}
```

PersonDetails.java

```
package coworking.project.security;

import coworking.project.models.Person;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.Collections;

public class PersonDetails implements UserDetails {
   private final Person person;

   public PersonDetails(Person person) {
      this.person = person;
   }

   @Override
   public Collection<? extends GrantedAuthority> getAuthorities() {
      return Collections.singletonList(new SimpleGrantedAuthority(person.getRole()));
```

```
    }

    @Override
    public String getPassword() {
        return this.person.getPassword();
    }

    @Override
    public String getUsername() {
        return this.person.getUsername();
    }

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

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

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

    @Override
    public boolean isEnabled() {
        return true;
    }

    public Person getPerson() {
        return person;
    }
}
```

AdminService.java

```
package coworking.project.services;

import coworking.project.exceptions.ReservationNotFoundException;
import coworking.project.models.Person;
import coworking.project.models.Reservation;
import coworking.project.repositories.PeopleRepository;
import coworking.project.repositories.ReservationsRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
public class AdminService {
    private final EmailService emailService;
    private final PeopleRepository peopleRepository;
    private final ReservationsRepository reservationsRepository;

    public AdminService(EmailService emailService, PeopleRepository peopleRepository, ReservationsRepository
reservationsRepository) {
        this.emailService = emailService;
        this.peopleRepository = peopleRepository;
        this.reservationsRepository = reservationsRepository;
    }
```

```
    public List<Person> findAll() {
        return peopleRepository.findAll();
    }

    @Transactional
    public void confirmReservation(Long id) {
        Reservation reservation = reservationsRepository.findById(id).orElseThrow(ReservationNotFoundException::new);
        reservation.setIsConfirmed(true);
        reservationsRepository.save(reservation);
        emailService.sendSuccessfulConfirmedMessage(reservation.getRenter().getEmail());
    }
}
```

AuthorizationService.java

```
package coworking.project.services;

import coworking.project.models.Person;
import coworking.project.repositories.PeopleRepository;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class AuthorizationService {
    private final EmailService emailService;
    private final PasswordEncoder passwordEncoder;
    private final PeopleRepository peopleRepository;


    public AuthorizationService(EmailService emailService, PasswordEncoder passwordEncoder, PeopleRepository peopleRepository) {
        this.emailService = emailService;
        this.passwordEncoder = passwordEncoder;
        this.peopleRepository = peopleRepository;
    }

    @Transactional
    public void save(Person person) {
        person.setRole("ROLE_USER");
        person.setPassword(passwordEncoder.encode(person.getPassword()));
        peopleRepository.save(person);
        emailService.sendSignUpEmail(person.getEmail());
    }
}
```

EmailService.java

```
package coworking.project.services;

import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class EmailService {
    private final JavaMailSender javaMailSender;

    public EmailService(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }

    public void sendSignUpEmail(String to) {
        SimpleMailMessage message = new SimpleMailMessage();
```

```
        message.setFrom("coworking.email.test@gmail.com");
        message.setTo(to);
        message.setSubject("You are successfully registered on our site!");
        message.setText("Thank you for registration on our site!");
        javaMailSender.send(message);
    }

    public void sendSuccessfulPaymentMessage(String to) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("coworking.email.test@gmail.com");
        message.setTo(to);
        message.setSubject("Information about payment");
        message.setText("You successfully paid for your reservation! Thank u.");
        javaMailSender.send(message);
    }

    public void sendSuccessfulConfirmedMessage(String to) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("coworking.email.test@gmail.com");
        message.setTo(to);
        message.setSubject("Information about confirming your reservation");
        message.setText("Administrator just confirmed your reservation!");
        javaMailSender.send(message);
    }
}
```

PeopleService.java

```
package coworking.project.services;

import coworking.project.exceptions.PersonNotFoundException;
import coworking.project.models.Person;
import coworking.project.repositories.PeopleRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

@Service
public class PeopleService {
    private final PeopleRepository peopleRepository;


    public PeopleService(PeopleRepository peopleRepository) {
        this.peopleRepository = peopleRepository;
    }

    public Person findById(Long id) {
        return peopleRepository.findById(id).orElseThrow(PersonNotFoundException::new);
    }

    public Optional<Person> findByUsername(String username) {
        return peopleRepository.findByUsername(username);
    }

    public Optional<Person> findByEmail(String email) {
        return peopleRepository.findByEmail(email);
    }

    @Transactional
    public void update(Long id, Person updatedPerson) {
        peopleRepository.save(updatedPerson);
    }
```

```
    @Transactional
    public void delete(Long id) {
        peopleRepository.deleteById(id);
    }
}
```

PersonDetailsService.java

```
package coworking.project.services;

import coworking.project.models.Person;
import coworking.project.repositories.PeopleRepository;
import coworking.project.security.PersonDetails;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import java.util.Optional;

@Component
public class PersonDetailsService implements UserDetailsService {
    private final PeopleRepository peopleRepository;

    public PersonDetailsService(PeopleRepository peopleRepository) {
        this.peopleRepository = peopleRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<Person> person = peopleRepository.findByUsername(username);
        if (!person.isPresent())
            throw new UsernameNotFoundException("Користувача з таким ім`ям не існує");
        return new PersonDetails(person.get());
    }
}
```

ProfileService.java

```
package coworking.project.services;

import coworking.project.models.Person;
import coworking.project.security.PersonDetails;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;

@Service
public class ProfileService {
    private final PeopleService peopleService;

    public ProfileService(PeopleService peopleService) {
        this.peopleService = peopleService;
    }

    public Person getPerson() {
        Person person = getCurrentUser();
        return (person.getId() == null) ? person : peopleService.findById(person.getId());
    }

    private Person getCurrentUser() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if (authentication.getPrincipal().equals("anonymousUser"))
            return new Person();
```

```
        PersonDetails personDetails = (PersonDetails) authentication.getPrincipal();
        return personDetails.getPerson();
    }
}
```

RatingService.java

```
package coworking.project.services;

import coworking.project.models.Rating;
import coworking.project.repositories.RatingRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
public class RatingService {
    private final RatingRepository ratingRepository;

    public RatingService(RatingRepository ratingRepository) {
        this.ratingRepository = ratingRepository;
    }

    public List<Rating> findTop5ByOrderByNumberOfUsingDesc() {
        return ratingRepository.findTop5ByOrderByNumberOfUsingDesc();
    }

    @Transactional
    public void update(Rating rating) {
        ratingRepository.save(rating);
    }
}
```

ReservationService.java

```
package coworking.project.services;

import coworking.project.exceptions.ReservationNotFoundException;
import coworking.project.models.Rating;
import coworking.project.models.Reservation;
import coworking.project.repositories.ReservationsRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDate;
import java.util.List;

@Service
public class ReservationService {
    private final RatingService ratingService;
    private final ReservationsRepository reservationsRepository;

    public ReservationService(RatingService ratingService, ReservationsRepository reservationsRepository) {
        this.ratingService = ratingService;
        this.reservationsRepository = reservationsRepository;
    }

    public Reservation findById(Long id) {
        return reservationsRepository.findById(id).orElseThrow(ReservationNotFoundException::new);
    }

    public List<Reservation> findPayedReservations() {
        return reservationsRepository.findPayedReservationsToConfirm();
    }
```

```
    public List<Reservation> findAllConfirmedReservations() {
        return reservationsRepository.findAllConfirmedReservations();
    }

    public List<Reservation> findAllByWorkPlaceAndRentDay(Long workPlaceId, LocalDate date) {
        return reservationsRepository.findAllByRentDay(workPlaceId, date);
    }

    @Transactional
    public void payReservation(Long id) {
        Reservation reservation = reservationsRepository.findById(id).orElseThrow(ReservationNotFoundException::new);
        reservation.setIsPayed(true);
        Rating rating = reservation.getWorkPlace().getRating();
        rating.setNumberOfUsing(rating.getNumberOfUsing() + 1);
        ratingService.update(rating);
        reservationsRepository.save(reservation);
    }

    @Transactional
    public void save(Reservation reservation) {
        reservationsRepository.save(reservation);
    }
}
```

ScheduleService.java

```
package coworking.project.services;


import coworking.project.models.Reservation;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

import java.time.LocalTime;
import java.util.List;

@Service
public class ScheduleService {
    private final WorkPlaceService workPlaceService;
    private final ReservationService reservationService;

    public ScheduleService(WorkPlaceService workPlaceService, ReservationService reservationService) {
        this.workPlaceService = workPlaceService;
        this.reservationService = reservationService;
    }

    @Scheduled(fixedDelay = 60_000)
    public void checkActualityOfInformation() {
        List<Reservation> reservations = reservationService.findAllConfirmedReservations();
        for (Reservation reservation : reservations) {
            if (reservation.getTimeFrom().isBefore(LocalTime.now())) {
                reservation.getWorkPlace().setIsAvailable(false);
            }
            if (LocalTime.now().isAfter(reservation.getTimeTo())) {
                reservation.setIsActual(false);
                reservation.getWorkPlace().setIsAvailable(true);
            }
            reservationService.save(reservation);
            workPlaceService.update(reservation.getWorkPlace());
        }
    }
}
```

WorkPlaceService.java

```
package coworking.project.services;

import coworking.project.exceptions.WorkPlaceNotFoundException;
import coworking.project.models.WorkPlace;
import coworking.project.repositories.WorkPlacesRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
public class WorkPlaceService {
   private final WorkPlacesRepository workPlacesRepository;

   public WorkPlaceService(WorkPlacesRepository workPlacesRepository) {
      this.workPlacesRepository = workPlacesRepository;
   }

   public List<WorkPlace> findAll() {
      return workPlacesRepository.findAll();
   }

   public List<WorkPlace> findAvailableWorkPlaces() {
      return workPlacesRepository.findAvailableWorkPlaces();
   }

   public WorkPlace findById(Long id) {
      return workPlacesRepository.findById(id).orElseThrow(WorkPlaceNotFoundException::new);
   }

   @Transactional
   public void update(WorkPlace workPlace) {
      workPlacesRepository.save(workPlace);
   }
}
```

ДОДАТОК Б

ВІДГУК КЕРВІНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_Полтавець.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_ Полтавець.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_ Полтавець.ppt	Презентація кваліфікаційної роботи