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 *Безбородько Сергій Володимирович*
(ПІБ)

академічної групи *122-18ск-2*
(шифр)

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

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

на тему: *Розробка серверної частини системи обліку розселення студентів у гуртожитку НТУ «Дніпровська політехніка»*

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

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18ск-2* (група) *Безбородько С.В.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка серверної частини системи обліку розселення студентів у гуртожитку НТУ «Дніпровська політехніка»*

затверджена наказом ректора НТУ «ДП» від *07.06.2021р.* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

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

Завдання прийняв до виконання _____ *Безбородько С.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 84 с., 14 рис., 6 табл., 3 дод., 22 джерел.

Об'єкт розробки: інформаційна система гуртожитку для ведення обліку та зберігання даних.

Мета кваліфікаційної роботи: створення системи підтримки операцій адміністрації гуртожитку для підвищення продуктивності та скорочення затрат часу на занесення даних та ведення обліку реалізований за допомогою автоматизації процесу ведення бази даних системи та виконання запитів.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється 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 платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

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

Практичне значення полягає у створенні додатка, що надає можливість електронного зберігання даних про гуртожиток та студентів, ведення бази даних та облік поселень та виселень.

Актуальність інформаційної системи визначається попитом на подібні розробки, що оптимізують та спрощують дії щодо ведення бази даних гуртожитку, скорочують час на введення даних на заселення; підвищують ефективність та швидкість роботи адміністрації шляхом електронного ведення обліку, та унеможливлюють помилки при поселенні студентів у кімнати.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, КОМП’ЮТЕР, ОБЛІК, БАЗА ДАНИХ, АВТОМАТИЗАЦІЯ, ПРОЕКТУВАННЯ, ДОДАТОК, МЕНЮ, ВКЛАДКА.

ABSTRACT

Explanatory note: 84 pages, 14 figures, 6 tables, 3 appendix, 22 sources.

Object of development: dormitory information system for accounting and data storage.

The purpose of the qualification work: to create a system to support the operations of the dormitory administration to increase productivity and reduce the time spent on data entry and accounting implemented through the automation of the process of maintaining the system database and queries.

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, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the subject branch is analyzed, the urgency of the task and the purpose of development are defined, the statement of the task is formulated, the requirements to software realization, technologies and software are specified.

The second section analyzes the available solutions, selected platforms for development, designed and developed the program, describes the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance lies in the creation of an application that provides the ability to electronically store data on dormitories and students, maintain a database and record settlements and evictions.

The relevance of the information system is determined by the demand for such developments that optimize and simplify the maintenance of the dormitory database, reduce the time to enter data for settlement; increase the efficiency and speed of the administration through electronic accounting, and prevent mistakes when settling students in rooms.

Keywords: INFORMATION SYSTEM, COMPUTER, ACCOUNTING, DATABASE, AUTOMATION, DESIGN, APPENDIX, MENU, TAB.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

ЦВЗ – цифровий водяний знак;

IDE – Integrated development environment, інтегроване середовище розробки.

ЗМІСТ

ВСТУП

Розроблена система призначена для автоматизації роботи адміністрації гуртожитку НТУ «ДП».

Студентський гуртожиток - це місце тимчасового проживання студентів під час навчального року. Тимчасове проживання студентів змушує адміністрацію вести облік студентів та кімнат. У час коли починається навчальний рік навантаження на адміністрацію гуртожитку підвищується. У результаті можуть ставатися помилки.

Використання проекту обумовлено потребою маніпулювати великими масивами даних. Цей додаток розроблений для вирішення завдань організації діяльності комплексу гуртожитків. Ведення обліку інформації про гуртожиток та мешканців.

Основна мета спроектованої системи – маніпулювання великими масивами даних, прискорення та автоматизація роботи адміністрації. Уникнення можливих помилок та мінімізувати людський фактор. Система не дасть виконати накладання даних які можуть призвести до помилкової ситуації.

Метою кваліфікаційної роботи є розробка серверної частини для web та Android застосунків.

Для досягнення поставленої мети необхідно вирішити основні завдання:

- аналіз організації роботи адміністрації та основні виконувані задачі

та на основі аналізу виділити вимоги до системи;

- проектування алгоритмів, бази даних, програмну реалізацію та перевірку тестуванням й відлагодження створеної системи;

- уточнення вимог до виконуваних процесів;

- автоматизація ведення даних, зниження кількості помилок при занесенні та прискорення роботи адміністрації гуртожитку.

У відповідності до проведеного аналізу поставлені основні функціональні задачі перед системою:

- автоматизація ведення даних;
- зниження проценту помилок при заселенні;
- прискорення роботи адміністратора.

Створюваний системи відображає точки зору: точка зору адміністратора, для якого дуже важлива можливість швидкої та простої зміни інформації, і точка зору співробітника або студента, для яких важливий простий і швидкий перегляд необхідної інформації, та обробка їх запитів. Механізм захисту бази даних від несанкціонованого доступу здійснюється шифруванням паролів для входу в програму.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1.Загальні відомості з предметної галузі

1.1.1 Поняття та актуальність теми

Студентський гуртожиток – це місце тимчасового проживання студентів під час навчання, яке належить навчальним закладам.

Актуальність розробки проекту очевидна. Впродовж багатьох років спостерігається збільшення кількості студентів які бажають поступити в вищий навчальний заклад. Багато студентів не можуть дозволити собі знімати квартири, тому потребують тимчасового проживання під час навчання в студентському гуртожитку. На протязі багатьох років спостережень, гуртожиток на початку кожного навчального року заселяє близько 30% першокурсників, не враховуючи студентів старших курсів. Це обумовлене тим, що мала вартість проживання, зручне розташування, тому приваблює студентів.

Початку навчального року є критичним часом для адміністрації гуртожитку, бо велика кількість студентів починають подавати заявки на заселення. В цей час у адміністрації виникають проблеми з розселенням та наявністю кімнат.

Тому розроблена система спрощую і мінімізує роботу людини. При заселені студентів у гуртожитки, система автоматизує та своєчасно оновлює інформацію, щоб через недостовірність даних про наявність вільних місць та великої кількості числа студентів, бажаючих заселитися, в гуртожитках не склалася ситуація, коли в двомісну кімнату записують троє, а то й четверо студентів. Система надасть достовірну інформацію для користувачів, це своєчасно мінімізує створення помилок та помилкових ситуацій при роботу з гуртожитком.

1.1.2 Огляд наявних програмних аналогів

У відкритому доступі в інтернеті не було знайдено жодного аналога з подібними функціями.

Зі знайдених систем є додаток TSU.Helper, який належить Томському ВНЗ.

Через нього можна подати заявку на виправлення побутових проблем в гуртожитках. Студенти можуть повідомити про поломку в своїй кімнаті або загальнодоступних місцях в гуртожитку. Вхід в TSU.Helper здійснюється через сервіс ТГУ.Аккаунти, при цьому аккаунт повинен бути підтвердженим. У додатку можна залишити заявку, яка буде перевірена і передана на виконання електрику, теслі, сантехніку та іншим. Список заявок переглядає комендант гуртожитку. Він ставить завдання працівникам і контролює процес виконання. Коли заявка буде виконана, студент отримає повідомлення.

1.2.Призначення розробки та галузь застосування

Кваліфікаційна робота призначена для створення системи та додатків, що надають можливість автоматизувати роботу системи гуртожитку, тобто прискорення та полегшення роботи адміністрації.

Система дасть можливість студентам заносити свої дані, та обирати кімнати для проживання. А адміністрації гуртожитку залишиться лише підтвердити дані студента та заявку на заселення. Та зв’язатися з студентам для отримання потрібних документів.

Головна цінність розробленої системи полягає в тому, що вона буде зберігати актуальні дані про мешканців, поверхи, стан заселення, вільні кімнати, де можна через функції пошуку знаходити необхідну інформацію, що прискорить процес заселення і виселення, та не буде створювати проблем

накладки даних.

1.3.Підстава для розробки

Підставою для розробки кваліфікаційної роботи на тему «Розробка серверної частини системи обліку розселення студентів у гуртожитку НТУ «Дніпровська політехніка»» є наказ ректора по Національному технічному університету «Дніпровська політехніка» від 07.06. 2021р. № 317-с.

1.4.Постановка завдання

Задачею кваліфікаційної роботи є розробка серверної частини додатку «Гуртожиток НТУ ДП». Додаток має мати наступні функції:

– виведення інформації про гуртожиток та поверхи;

– виведення розгорнутої інформації про кімнати – кількість місць, кількість вільних місць, опис кімнати;

– реєстрація/авторизація користувачів;

– вибір кімнати та заповнення заявок на заселення до кімнати;

– редагування адміністратором даних.

Додаток має бути зі зрозумілим для користувача інтерфейсом, мати перевірку валідності даних та шифрування паролів. Також дані повинні зберігатися на сервері у базі даних та бути логічно структурованими.

За результатами тестування програми, необхідно зробити висновки з працездатності розробленого додатку та його функцій.

1.5.Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Програмна реалізація проекту має складатися з таких частин, як контролерів входу/реєстрації, контролера головної сторінки, інформації про кімнати, форми для заповнення заявки, контролера сторінки для обробки даних комендантом або адміністратором.

Дані повинні оброблятися на сервері та зберігатися у базі даних, та мати шифрування паролів і cookie.

1.5.2. Вимоги до інформаційної безпеки

Основні вимоги до інформаційної безпеки:

– цілісність даних;

– захист від неавторизованого входу та редагування;

– конфіденційність інформації;

– доступність інформації для всіх авторизованих користувачів.

Для надійної роботи програмного забезпечення необхідно дотримуватися таких факторів:

− використання ліцензійного ПЗ;

− захист від зловмисних програм;

− шифрування даних на сервері;

− встановлення блоків безперебійного живлення.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для програмного додатку потрібен комп’ютер чи ноутбук з рекомендованими параметрами:

– підтримка 64-розрядного процесору та операційної системи;

– операційна система Windows 10 64bit;

– процесор Intel Core i5 3470 3.2 GHz/AMD X8 FX-8350 4 GHz;20

– оперативна пам'ять 8GB ОЗП;

– відеокарта nVidia GTX 650 з 1 Гб відеопам’яті, AMD HD7860 з 1 Гб відеопам’яті;

– DirectX версії 12;

– жорсткий диск мінімум в 72 GB.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програма має бути написана на об’єктно-орієнтованій мові програмування Java для серверної частини та Android додатку і HTML, CSS та JavaScript для клієнтської частини веб-додатку.

Функціонування розробленої системи має бути забезпечено на наступних операційних системах:

- Windows 98, 2000 і вище;
- Windows NT 4.0, XP.

Та у всіх браузерах, по типу Chrome, Internet Explorer, Mozilla Firefox і всіх Android пристроях з версіями не нижче 4.0.1.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Робота забезпечує можливість автоматизації роботи системи гуртожитку, прискорення та полегшення роботи адміністрації. Розроблена система дає можливість студентам самостійно заносити свої дані, обирати кімнати та заповнювати заявки на заселення. Адміністратору необхідно буде лише перевірити дані та розглянути заявку, а саме відхилити або підтвердити та заселити студента в відповідну кімнату.

2.2. Опис застосованих математичних методів

Розроблена система не потребує застосування математичних методів, окрім простих арифметичних та логічних операцій, що використовуються під час роботи серверної частини, та не потребують окремого опису.

2.3. Опис використаних технологій та мов програмування

Для розробки серверної частини під проект було вирішено використати мову програмування Java, а середовищем розробки обрана IntelliJ IDEA 2020. Використані фреймворки Maven та Spring та об’єктно-реляційна система керування базами даних PostgreSQL.

Java - сучасна об'єктно-орієнтована мова програмування. Один з самих популярних мов програмування. Одна з двох мов, що використовується в розробці в Android.

Java є C подібною мовою, тому вона має дуже багато подібного з мовами, що також були написані на основі C. Поняття об'єктно-орієнтована

відноситься до способу написання коду. А саме те що весь код Java знаходиться у класах і запускаються разом. Одним з великих переваг є платформо-незалежність, тобто написавши код на одній платформі можна легко запустити на іншій платформі.

Щоб запустити код на Java треба:

- Java Development Kit;
- Java Runtime Environment;
- Java Virtual Machine.

Java Virtual Machine - це віртуальна машина Java гарантує, що ваші додатки мають доступ до мінімальних ресурсів, для їх запуску. Завдяки Java Virtual Machine програми на Java запускаються так легко на різних платформах. Java Runtime Environment - середовище виконання Java, надає собою контейнер для всіх цих елементів і коду для запуску програми. Java Development Kit - це компілятор, який інтерпретує сам код і виконує його. В Java Development Kit також є інструменти розробника, необхідні для написання коду Java.

Хоча Java і C подібна мова, але вона має також багато відмінностей від них. Одним за таких відмінностей є виведення помилок та попереджень. Усі помилки в Java структуровані і при роботі програми будуть автоматично виводитись до конкретного випадку. Програміст може створити не спеціально ситуацію, яка приведе до не критичної або критичної ситуації. В C та C++ програміст повинен сам буде знайти в якому місці він допустив цю помилку, а Java надасть точне роз'яснення цій помилці, покаже де і коли це відбулося, що воно зачепило і пояснення цій помилці. Також середа розробки не дасть програмісту створити очевидні помилки під час написання коду.

Середовищем для Java є програма IntelliJ IDEA. Це одна з популярніших програм для Java, хоча і платна. IntelliJ IDEA - це продукт компанії JetBrains, що надає середу розробки для Java. Незважаючи на те, що IntelliJ IDEA - в першу чергу IDE для Java, вона розуміє і надає інтелектуальну допомогу при

написанні коду на SQL, JPQL, HTML, JavaScript і багатьох інших мовах. Дозволяє редагувати код, написаний не на Java, в середині строкових літералів Java коду. Підтримується багато мов програмування:

- Java;
- JavaScript;
- CoffeeScript;
- HTML / XHTML / HAML;
- CSS / SASS / LESS;
- XML / XSL / XPath;
- YAML;
- ActionScript / MXML;
- Python;
- Ruby;
- Haxe;
- Groovy;
- Scala;
- SQL;
- PHP;
- Kotlin;
- Clojure;
- Ci;
- C ++;
- Go;
- Rust.

IntelliJ IDEA надає великий набір інструментів для створення, управління та редагування проєктів на Java. В середу розробки внесено великий потенціал, який надає програмісту можливість мінімізувати ручне написання коду, так як середа допомагає в написанні. Вона знаходить

підходящі методи, змінні, значення і передає на вибір програмісту. Підключаючи різні бібліотеки і фреймворки середа сама буде їх підгружати до проекту, людині залишається лише пам’ятати їх і вписати код для загрузки в файлах. Великий набір різних форм проектів, для різних потреб. Виділяють 17 різних проектів:

- Java;
- Maven;
- Gradle;
- Java FX;
- Android;;
- IntelliJ Platform Plugin
- Java Enterprise;
- Spring Initializr;
- Quarkus;
- Micronaut;
- MicroProfile;
- Groovy;
- Grails;
- Application Forge;
- Kotlin;
- Web;
- JavaScript.

Ряд мов підтримується за допомогою плагінів сторонніх розробників, так реалізована підтримка OCaml, GLSL, Erlang, Fantom, Haskell, Lua, Mathematica, Perl5, Object Pascal.

Apache Maven - фреймворк для автоматизації збирання проектів на основі опису їх структури в файлах на мові POM, що є підмножиною XML. Спочатку використовувався для Java проектів, але в подальшому розширено і під інші

мови. На теперішній час Maven використовується для побудови і управління проектами, написаними на Java, C #, Ruby, Scala, та іншими мовами. Проект Maven видається співтовариством Apache Software Foundation, де формально є частиною Jakarta Project.

Maven забезпечує декларативну збірку проекту. У файлах опису проекту прописано його специфікація. Всі завдання по обробці файлів, описані в спеціальному файлі специфікації, Maven виконує за допомогою їх обробки вбудованих і зовнішніх плагінів. Серед примітних альтернатив є система автоматичного складання Gradle, побудована на принципах Apache Ant і Maven, але використовує спеціалізований DSL на Groovy замість POM-конфігурації.

Spring Framework - це програмний каркас з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean.

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги «Expert One-on-One Java EE Design and Development» (Wrox Press, жовтень 2002 року).

Фреймворк був відкритий під ліцензією Apache 2.0 license в червні 2003 року. Перша стабільна версія 1.0 випущена в березні 2004.

Spring Framework надає комплексну модель програмування і конфігурації для сучасних корпоративних додатків на основі Java - на будь-якій платформі розгортання. Ключовим елементом Spring є інфраструктурна підтримка на рівні додатків. Spring фокусується на з'єднанні корпоративних додатків, так що команди можуть зосередитися на бізнес логіці рівня додатку, без непотрібних прив'язок до конкретних середовищ розгортання.

Spring Framework складається з кількох модулів, які надають широкий спектр послуг:

- Core technologies є одною з основних модулів Spring. Бібліотека з набором основних бібліотек для роботи з Java. Залежність для неї по замовченню генерується при створенні любого проекту Spring;
- Spring MVC та Spring WebFlux веб-фреймворки. Є збіркою популярніших бібліотек для роботи з Web і забезпечують взаємодію під капотом проекту;
- Доступ до даних: транзакції, підтримка DAO, JDBC, ORM, XML. Управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів. Завдяки цьому модулю програміст обходить рутинну роботу з підключенням баз даних, а зосереджується на логіці взаємодії з цією базою даних;
- Модель-Вигляд-Управління: програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful;
- Аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security;
- Тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів.

Тим часом, особливості ядра Spring застосовні в будь-якому Java-додатку, і існує безліч розширень і удосконалень для побудови веб-додатків на Java Enterprise платформі. З цих причин Spring став одним з самих популярних і визнається програмістами як найкращий фреймворк.

PostgreSQL - вільна об'єктно-реляційна система управління базами даних.

Існує в реалізаціях для безлічі UNIX-подібних платформ, включаючи AIX, різні BSD-системи, HP-UX, IRIX, Linux, macOS, Solaris / OpenSolaris, Tru64, QNX, а також для Microsoft Windows.

Сильними сторонами PostgreSQL вважаються:

- високопродуктивні і надійні механізми транзакцій і реплікації;
- розширювана система вбудованих мов програмування: в стандартному постачанні підтримуються PL / pgSQL, PL / Perl, PL / Python і PL / Tcl; додатково можна використовувати PL / Java, PL / PHP, PL / Py, PL / R, PL / Ruby, PL / Scheme, PL / sh і PL / V8, а також є підтримка завантаження модулів розширення на мові C;
- спадкування;
- можливість індексування геометричних (зокрема, географічних) об'єктів і наявність базується на ній розширення PostGIS;
- вбудована підтримка слабо структурованих даних в форматі JSON з можливістю їх індексації;
- розширюваність (можливість створювати нові типи даних, типи індексів, мови програмування, модулі розширення, підключати будь-які зовнішні джерела даних).

PostgreSQL створена на основі системи управління базами даних Postgres, розробленої як open source проект в Каліфорнійському університеті в Берклі. До розробки Postgres, що почалася в 1986 році, мав безпосереднє відношення Майкл Стоунбрейкер, керівник більш раннього проекту Ingres, на той момент вже придбаного компанією Computer Associates. Назва розшифровується як «Post Ingres», і при створенні Postgres були застосовані багато ранніх напрацювання.

Одна з популярніших баз даних для IntelliJ IDEA. Переваги цієї бази даних вже багато років підтвердили Java програмісти. Велика кількість налаштувань і підтримка різних форматів забезпечую гарну гнучкість для створених проектів.

2.4. Опис структури програми та алгоритмів її функціонування

2.4.1 Опис алгоритмів програми

Програма має 5 рівні доступу користувачів до програми що зображено на Use-case діаграмі(рис.2.1), а саме:

Перший рівень – Включає у себе незареєстрованих користувачів. До їх можливостей відносимо:

- встановити додаток/ зайти на сайт;
- переглянути інформацію про вільні кімнати, що на даний час є у гуртожитку.
- зареєструватися;
- ввійти;

Вони можуть побачити, які кімнати вільні на даний час, кількість мешканців у кімнатах, та кількість наявних вільних місць. У цей рівень попадають усі тільки що зареєстровані користувачі до перевірки адміністратором.

Другий рівень – Гість, він має тіж можливості першого рівня. У цей рівень попадають зареєстровані користувачі котрих адміністратор не знайшов як студентів.

Третій рівень – Студент, він має тіж можливості першого рівня, та має додатково можливості:

- обрати кімнату;
- заповнити заявку на заселення.

У цей рівень попадають усі користувачі яких перевірив адміністратор і знайшов у списку студентів.

Четвертий рівень – Комендант, він успадковує можливості Студента, та наступні властивості:

- розглянути заявку на заселення;
- прийняти або відхилити заявку;

– додати інформацію про нового мешканця у БД;
– виселити мешканця, що проживає;
– видалити інформацію про мешканця, який виселився/якого виселили.

Коменданта заводить тільки адміністратор.

П’ятий рівень – Адміністратор, він має можливості, що і Комендант. Також має додаткові властивості:

– Реєструвати нових користувачів сайту/додатку, надавати права доступу;
– Змінювати інформацію про кімнати – вільні чи заселені;
– Змінювати інформацію про мешканців.

Адміністратор має усі права і має право маніпулювати проектом та базою даних, створювати, редагувати та видаляти по порученню коменданта.

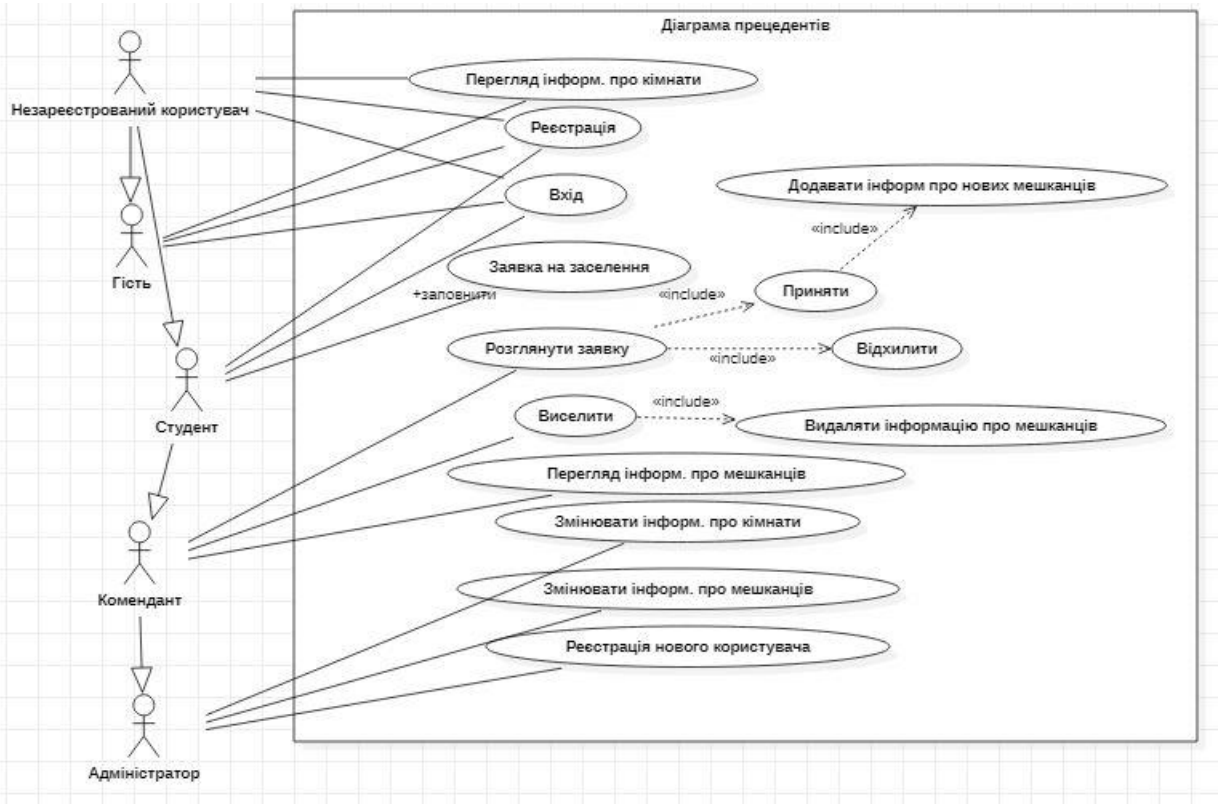

Рис. 2.1. Use-case діаграма

2.4.2 Структура роботи програми

Сучасні програми працюють з великою кількістю різноманітної інформації та технологіями. Для забезпечення роботи проекта усі класи розділені по призначенню:

- Model;
- Repository;
- Configurer;
- Service;
- Controller.

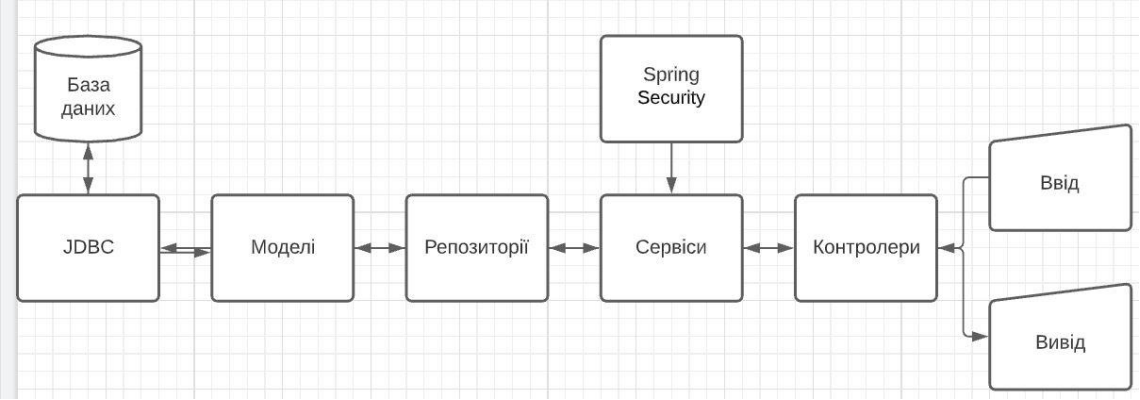

Рис. 2.2. Блок схема роботи програми

Model - це класи які являються моделями для роботи за базою даних. Вони являються об’єктами, що зберігаються і беруться з бази даних проходячи через JDBC.

Repository - це інтерфейси репозиторії забезпечують управління над моделями через успадкуванням JpaRepository. Написані методи в цих класах забезпечуть управління об’єктами в базі даних до якої підключен проект. Методи можуть шукати, видаляти, редагувати та зберігати об’єкти. Кожен метод можна налаштувати під конкретну потребу.

Service - це класи в яких написана головна логіка проекту, обробка даних, логіка взаємодії між базою даних і користовачем. Через те, що проект має авторизацію і використовує Spring Security, в сервісах знаходиться і логіка

авторизації та отримання токена користувача для управління додатками. Головна роль цих класів забезпечити логіку усього проекту і не дати доступ постороннім.

Controller - це класи які обробляють запити користувачів, видачу інформації по запитам. Вони передають і отримують інформацію від сервісів.

Configurer - це класи в яких написані правила доступу і управління між усіма компонентами проекту. Головна роль, забезпечити проекту зборку і коректну роботу усіх компонентів між собою, та описати правила які повинен знати кожна технологія в проєкті. На прикладі Spring Security - це видача можливостей кожному користувачу в залежності від їх ієрархії ролей.

На рисунку 2.3 зображена діаграма послідовності взаємодії з програмою.

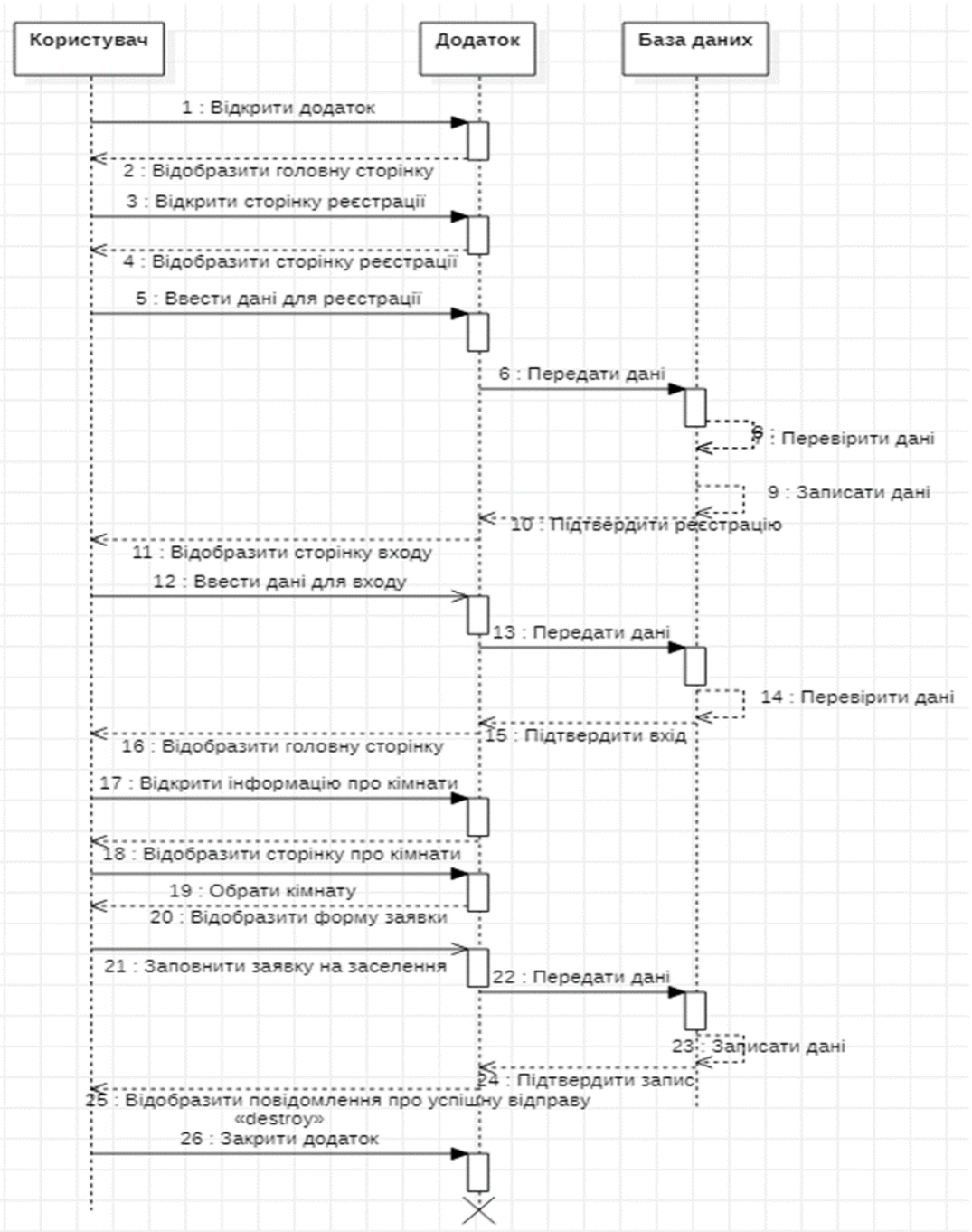

Рис. 2.3. Діаграма послідовностей

2.4.3 Опис структури бази даних системи

Для правильної роботи системи використовуємо базу даних PostgresQL.

Для роботи з базою даних вікористовують:

- JDBC;
- Postgresql;
- Hibernate-validator;
- Spring-boot-starter-data-jpa;
- Lombok.

JDBC - Java DataBase Connectivity платформений стандарт для взаємодії любих Java додатків з любими системами управління базами даних. JDBC основан на концепції драйверів які дозволяють отримати з'єднання з базою даних по спеціальному протоколу URL.

Hibernate-validator - бібліотека забезпечуюча валідацію даних не тільки у Java EE, вле і в Spring проектах. Дає змогу напряму в методах валідувати дані та видавати відповідну інформацію у відповідь. Може бути реалізована, як окремий модуль так і напряму в методах або в моделях для бази даних.

Spring-boot-starter-data-jpa - бібліотека Spring, що включає усі необхідні бібліотеки для роботи з базами даних у проекті Spring. Спрощує і мінімізує підключення багатьох інших залежностей, включає в себе усі необхідні бібліотеки та стандарти для підключення та роботи.

Lombok – java бібліотека з набором команд і анотацій для спрощення написання коду. Дає змогу поставити анотації замість написання подібного коду багато разів.

PostgresQL - набір драйверів та налаштувань для підключення до PostgreSQL. Для нормальної роботи проекту автоматично налаштовує усі параметри для конкретної бази даних.

Для Проекту було вирішено створити базу даних яка буде зберігати інформацію про гуртожиток і видавати її в подальшому(рис.2.4). Для

створення бази даних було вирішено створювати її за допомогою IntelliJ IDEA. В проекті створено папку Model, в ній створені класи моделі. Вони і є моделями для створення та взаємодії з базами даних. Для створення і роботи було створено 6 класів моделей:

- ApplicationsForAccommodation;
- Floors;
- Residents;
- Role;
- Rooms;
- Users.

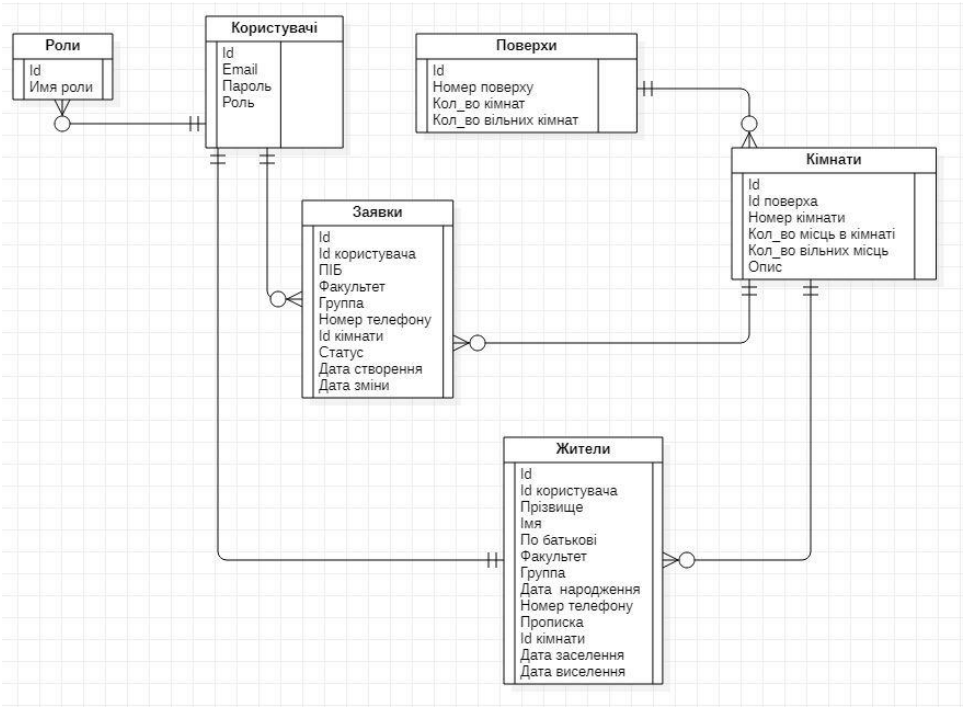

Рис. 2.4. База даних

Для створення і підключення моделей необхідно використовувати анотації Lombok які підтягують автоматичну реалізацію. В подальшому в проекті використання моделей забезпечує прямий зв’язок з базою даних. Для доступу до даних у базі даних створюємо інтерфейси репозиторії:

- ApplicationsForAccommodationRepository;

- FloorsRepository;
- ResidentsRepository;
- RoleRepository;
- RoomsRepository;
- UserRepository.

Репозиторії успадковуються від JpaRepository який є у складі Spring-boot-starter-data-jpa і автоматично реалізує методи зберігання, редагування, видалення та пошуку. Також в репозиторіях можна створити методи для отримання списку на основі будь якого параметру або значення об’єктів в базі даних.

Параметри для роботи з базою даних прописано в файлі application. properties(рис.2.5).

```
spring.datasource.url=jdbc:postgresql://ec2-63-34-97-163.eu-west-1.compute.amazonaws.com:5432/dchfa8ts6gcudv
spring.datasource.username=mksbgazygfntwo
spring.datasource.password=7c2e64975358a781012d394a8429318a67e96f1566a6d7fd9c95b768409caa39
jwt.secret=javamaster

#drop n create table again, good for testing, comment this in production
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

server.port=${PORT:8080}

spring.mvc.view.prefix = /WEB-INF/jsp/
spring.mvc.view.suffix = .jsp
```

Рис. 2.5 – application.properties

2.5. Обґрунтування та організація вхідних та вихідних даних програми

До вхідних даних проекту належать наступні дані, які записуються до бази даних:

Таблиця 2.1

Вхідні дані AdminController

Метод	Дані	Опис
userList	facultyForFilter	Параметр для фільтрації списку по факультету
	groupInForFilter	Параметр для фільтрації списку по групі
	surnameForFilter	Параметр для фільтрації списку по прізвищу
gtUser	userId	Автоматично береться від кожного об’єкта користувача
acceptRejectUser	userId	Автоматично береться від кожного об’єкта користувача якого вибрали
	action	Параметр який дає зрозуміти методу, що треба робити

Таблиця 2.3

Вхідні дані FloorsController

Метод	Дані		Опис
floorAdd	floor	numberFloor	Введений номер кімнати в форму
		numberOfRoomsPerFloor	Введена кількість місць в кімнаті в форму
deleteFloor	floorId	Автоматично береться від кожного об’єкта поверха який вибрали	
	action	Параметр який дає зрозуміти методу, що треба робити	

Таблиця 2.2

Вхідні дані ApplicationsForAccommodationControoler

Метод	Дані		Опис
saveApplicationsForAccommodation	applicationsForAccommodation	nameSurnameLastname	Введене ім'я, прізвище, по батькові студента в форму
		faculty	Введений факультет студента в форму

		groupIn	Введена група студента в форму
		phoneNumber	Введений номер телефону студента в форму
	numberRoom	Номер кімнати в яку подається заявка	
	principal	Об’єкт користувача котрий авторизований у додатку	
makingDecisionOnApplication	applicationId	Автоматично береться від кожного об’єкта заявок який вибрали	
	action	Параметр який дає зрозуміти методу, що треба робити	

Таблиця 2.5

Вхідні дані RoomController

Метод	Дані		Опис
addRoom	floor	numberFloor	Введений номер кімнати в форму
		numberOfRoomsPerFloor	Введена кількість місць в кімнаті в форму
		description	Введений опис кімнати в форму
deleteFloor	floorId	Автоматично береться від кожного об’єкта поверха який вибрали	
	action	Параметр який дає зрозуміти методу, що треба робити	

Таблиця 2.4

Вхідні дані RegistrationController

Метод	Дані		Опис
addUser	userForm	username	Введений логін користувача в форму
		password	Введений пароль користувача в форму
		passwordConfirm	Введене підтвердження пароля користувача в форму
	name		Введене ім’я користувача в форму
	lastname		Введене прізвище користувача в форму
	surname		Введене по батькові користувача в форму
	faculty		Введений факультет користувача в форму
	groupIn		Введена група користувача в форму
	phoneNumber		Введений номер телефону користувача в форму
	registration		Введену місце проживання користувача в форму

До вихідних даних належать всі дані, що виводяться з бази даних під час роботи проекту:

Таблиця 2.6

Вихідні об’єкти

Об’єкти	Параметри	Опис
ApplicationsForAccommodation	id	Id заявки
	nameSurnameLastname	Ім’я, прізвище, по батькові користувача що подав заявку
	faculty	факультет користувача що подав заявку
	groupIn	група користувача що подав заявку
	phoneNumber	Номер телефону користувача що подав заявку
	creationDate	Дата створення заявки
	status	Статус заявки
	dateOfChange	Дата розгляду заявки
Floors	id	Id поверху
	numberFloor	Номер поверху
	numberOfRoomsPerFloor	Кількість місць в поверху
	numberOfFreeRoomsOnTheFloor	Кількість вільних кімнат на поверху
Residents	id	Id користувача
	name	Ім’я користувача
	surname	Прізвище користувача
	lastname	По батькові користувача
	faculty	Факультет користувача
	groupIn	Група користувача
	phoneNumber	Номер телефону користувача
	registration	Місце регістрації користувача
Rooms	id	Id кімнати
	numberRoom	Номер кімнати
	numberOfSeatsInTheRoom	Кількість місць в кімнаті
	numberOfFreePlacesInTheRoom	Кількість вільних місць в кімнаті
	description	Опис кімнати
Users	id	Id користувача
	username	Логін користувача

Вихідні дані можуть бути як поодинокі так і у масивах, але формат виводу у web застосунок може бути різним. Так як доступ до об’єктів прямий і може підтягуватися за рахунок зв’язків.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для розробки ІС використовувався комп’ютер з наступними параметрами:

– операційна система Windows 10;

– процесор Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz;

– оперативна пам'ять 8GB ОЗП;

– відеокарта nVidia GeForce 8600/9600GT, ATI/AMD;

– Radeon HD2600/3600;

– DirectX версії 9.0c;

– жорсткий диск мінімум в 15 GB.

2.6.2. Використані програмні засоби

Запропонована програма написана на строго типізованій об’єктно-орієнтовані мові програмування Java в середовищі програмування IntelliJ IDEAUltimate, база даних PostgresSQL.

2.6.3. Виклик та завантаження програми

Для запуску даного сервера необхідно зайти в середу розробки IntelliJ IDEA. Коли вперше запускається сервер, перед цим треба підключити базу даних до проекту. Та змінити в application.properties змінити значеня

spring.jpa.hibernate.ddl-auto=update на spring.jpa.hibernate.ddl-auto=create. Після цього зайти в клас HotelStudentApplication. В класі HotelStudentApplication зліва натиснути на зелений трикутник. Після старту дочекатися поки запуститься проєкт, як на рисунку 2.6.

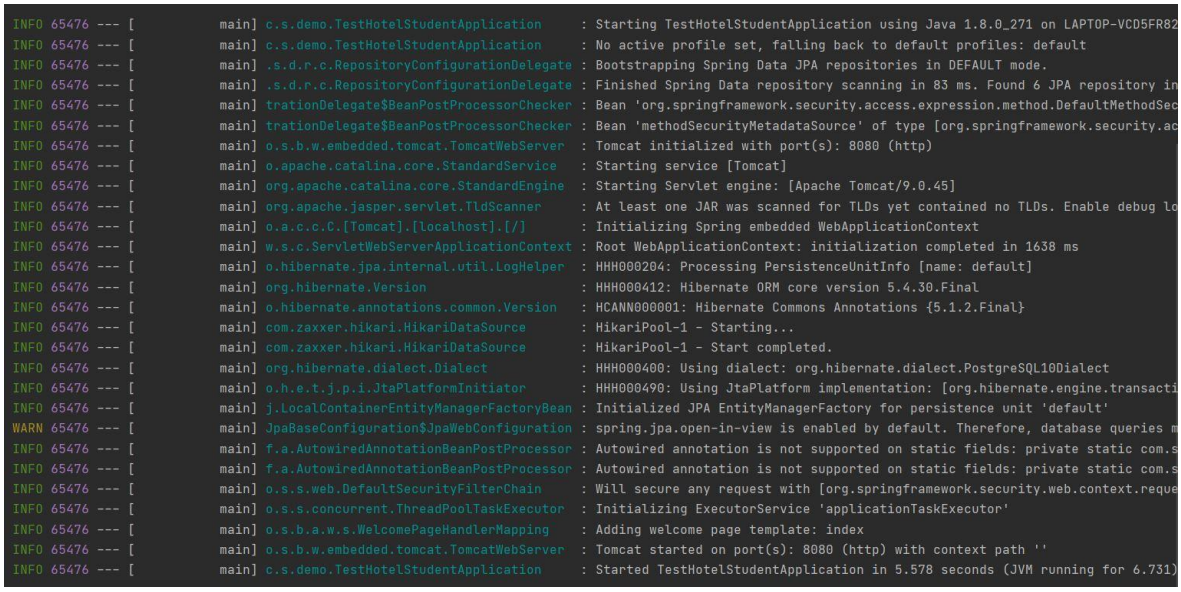

Рис. 2.6 – Start server

2.6.4. Опис інтерфейсу користувача

В програмі можна виконувати такі дії:

- Регістрація;
- Авторизація;
- Перегляд інформації про гуртожиток, користувачів, поверхи, кімнати і заявки;
- Створення кімнат, поверхів;
- Видалення кімнат, поверхів;
- Подання заявок на заселення в кімнати;
- Виселення студентів з кімнат.

2.6.5. Порядок роботи з розробленою системою

Запускаємо проект, після запуску:

1. Переходимо на початковий URL localhost:8080/. Визивається метод який перенаправляє нас на localhost:8080/first і визиває метод first.

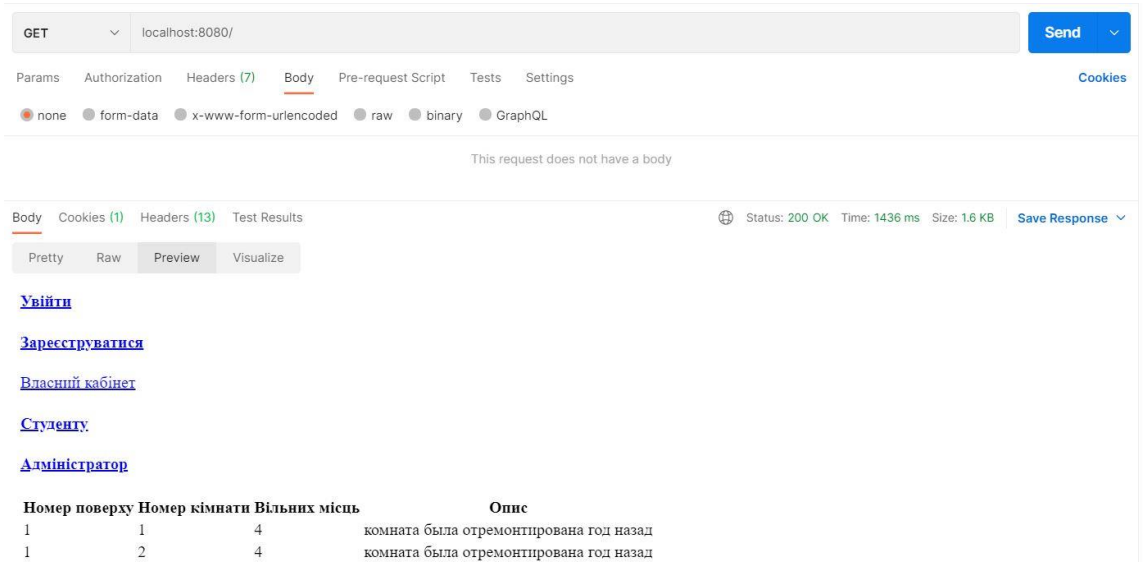

Рис. 2.7. Шлях localhost:8080/

Для усіх користувачів дозволяється отримати інформацію про вільні кімнати через метод @GET first.

2. Якщо ви не маєте облікового запису то переходимо на localhost:8080/registration

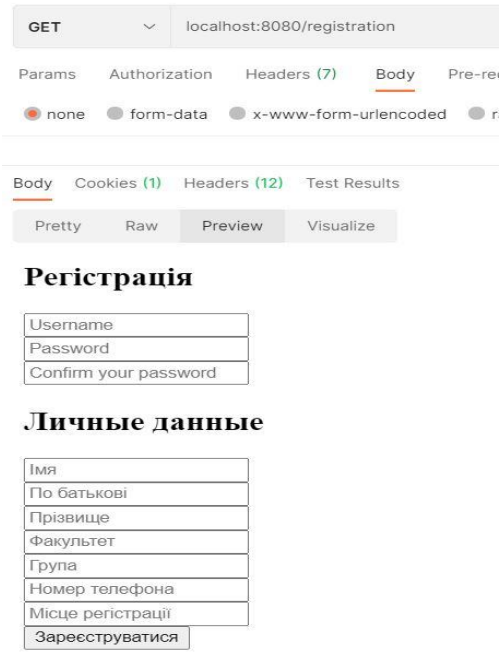

Рис. 2.8. Шлях localhost:8080/ registration

Після заповнення форми натиснувши кнопку “Зареєструватися” уся введена інформація передасться у метод @POST addUser і збережеться у базі даних. Після збереження автоматично перейде на localhost:8080/first.

3. Після успішної регістрації або ,як що у вас є обліковий запис то переходимо на localhost:8080/login

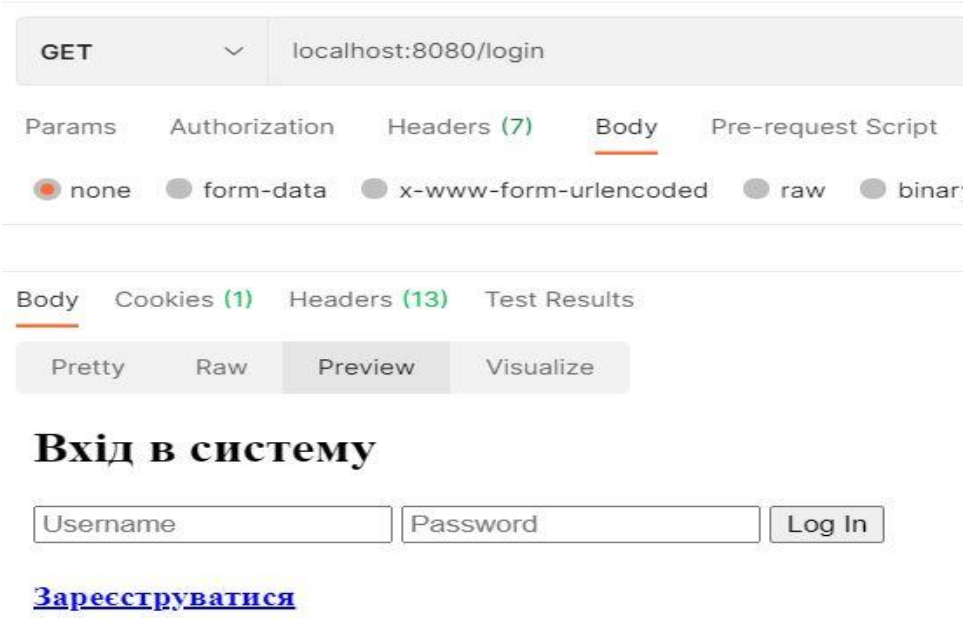

Рис. 2.9. Шлях localhost:8080/ login

Для входу необхідно ввести свій логін та пароль та натиснути ла кнопку входу, після цього визивається метод Spring Security. Коли система перевіре введену інформацію то створить сесію користувача і перейде на localhost:8080/first. Якщо користувач не є студентом, комендантом чи адміністротором то його можливості на цьому закінчуються.

2.6.5.1. Порядок роботи з розробленою системою студентам

1. Якщо студент і обліковий запис заведений у систему як студент. То можливості користувача збільшені від звичайних користувачів. В його можливостях є подання розширеній список за переходом на localhost:8080/news визивається метод @GET studentNews і виводить розширену інформацію для студентів.

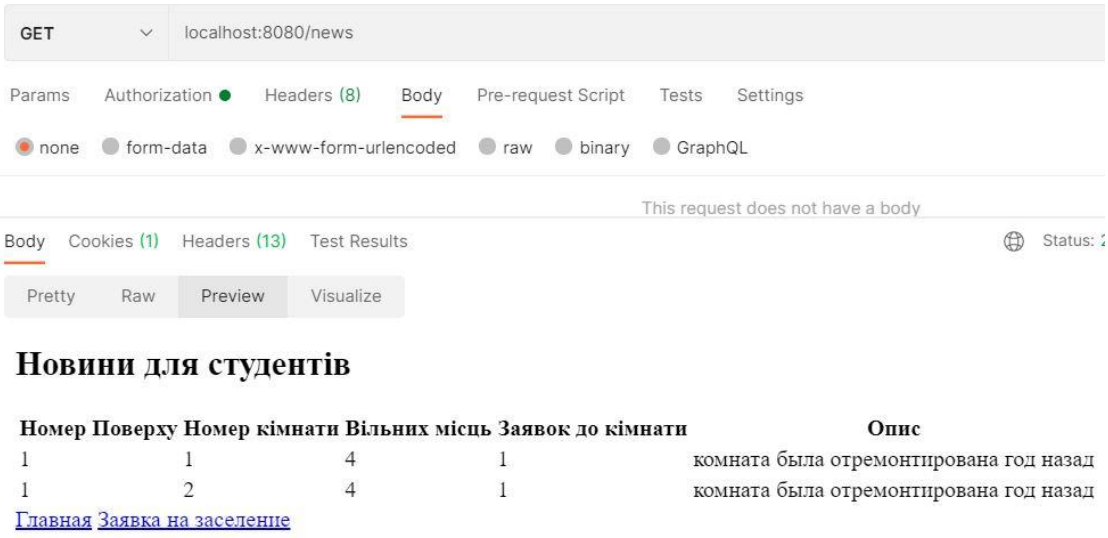

Рис. 2.10. Шлях localhost:8080/news

2. З інформації що надає метод @GET studentNews можна вирішити з кімнатою і перейти в localhost:8080/applicationsForAccommodation який

визиває метод @GET applicationsForAccommodation для отримання форми заповнення заявок на заселення.

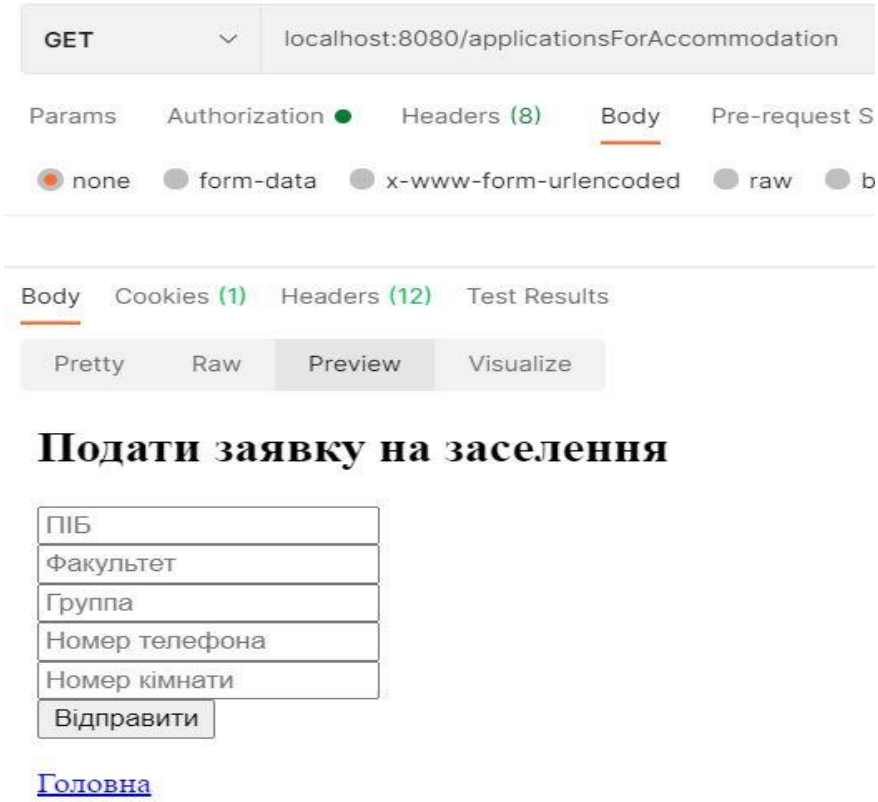

Рис. 2.11. Шлях localhost:8080/applicationsForAccommodation

Після того як форма заявки буде заповнена і натиснута кнопка відправки інформації на заселення то починає працювати метод @ POST saveApplicationsForAccommodation. Метод обробляє інформацію, додає дату та час заповнення заявки і прикріпляє її до користувача який заповнював заявку.

2.6.5.2. Порядок роботи з розробленою системою коменданта та адміністратора

1. Комендант та адміністратор можуть мати доступ до усіх попередніх методів. Але вони мають і власні методи до котрих ніхто окрім них не

має дуступа. Для перходу на головну сторінку вводимо localhost:8080/admin, метод для відображення @GET userList.

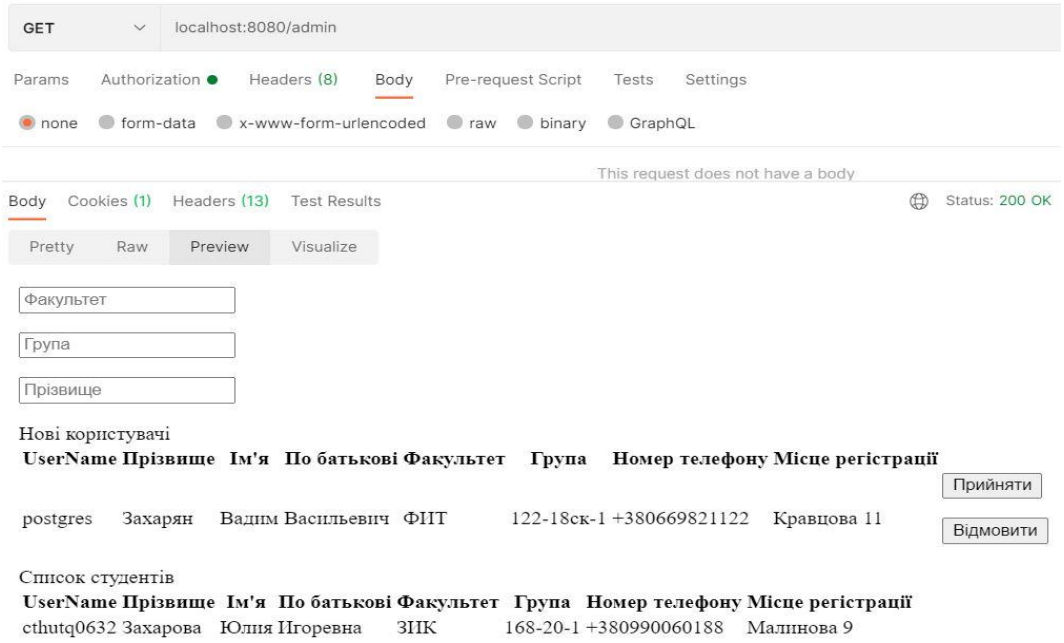

Рис. 2.12. Шлях localhost:8080/admin

Метод userList може виводити увесь список користувачів які чекають вирішення їх статусу, але і сортувати цей список по факультету, групі і прізвищам. Одним з основних методів для вирішення статусу є @POST acceptRejectUser. При роботі цього методу передається значення яке дає змогу визначити чи зробити користувача студентом або звічайним користувачем.

2. Для управління заявками відведено шлях localhost:8080/applications. Метод @GET applications для відображення усіх списків заявок користувачів.

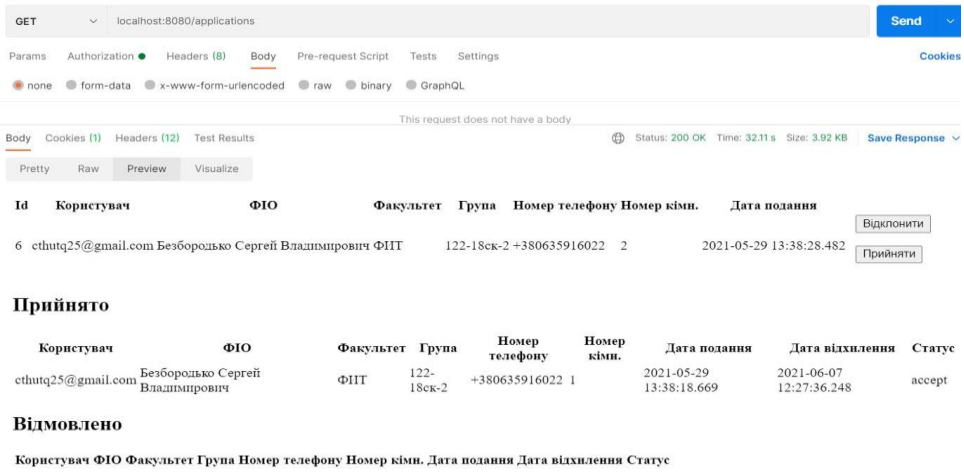

Рис. 2.13. Шлях localhost:8080/applications

Для управління заявками відведено метод @POST makingDecisionOnApplication, метод приймає id заявки і значення для прийняття або відхилення заявки на заселення.

3. Для роботи з студентами перейдемо на localhost:8080/controlResidents. Метод @GET showAllResident виводить усіх студентів з бази даних, також може фільтрувати по 5 параметрам.

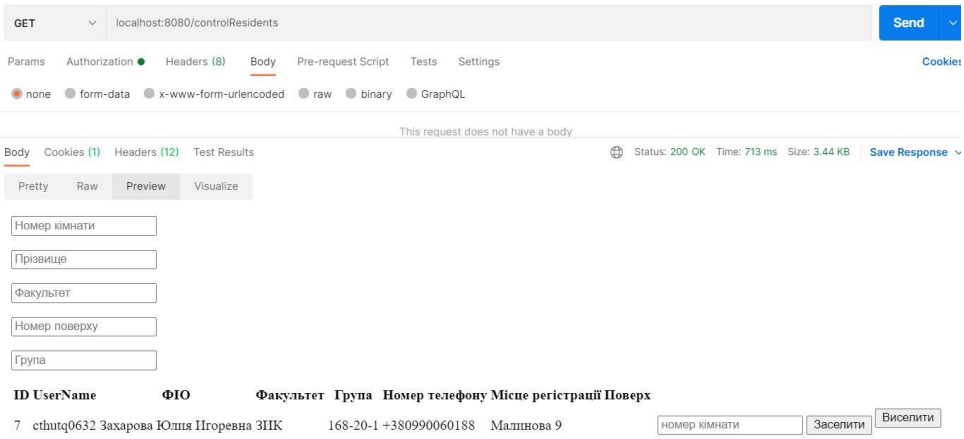

Рис. 2.14. Шлях localhost:8080/controlResidents

Метод @POST controlResident отримує значення id студента для подальшому управлінням ним. Далі передають значення для управління і номера кімнати, якщо треба заселити.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

– Передбачуване число операторів – 1215;

– Коефіцієнт складності програми – 1,5;

– Коефіцієнт корекції програми в ході її розробки – 0,05;

– Годинна заробітна плата програміста, грн/год – 85.

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:

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

де:

to - витрати праці на підготовку й опис поставленої задачі (приймається 50);

ti - витрати праці на дослідження алгоритму рішення задачі;

ta - витрати праці на розробку блок-схеми алгоритму;

tп - витрати праці на програмування по готовій блок-схемі;

totл - витрати праці на налагодження програми на ЕОМ;

tд - витрати праці на підготовку документації.

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

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \quad (3.2)$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

$$Q = 1215 \cdot 1,5 \cdot (1+ 0,05) = 1914 \text{ людино-годин.} \quad (3.3)$$

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

$$t_u = \frac{QB}{(75..85)K}, \text{ людино-годин,} \quad (3.4)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; B=1.2 ... 1.5;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{1914*1,2}{80*0,8} = 36, \text{ людино-годин.} \quad (3.5)$$

Витрати на складання програми по готовій блок-схемі:

$$t_\alpha = \frac{Q}{(20..25)K} \text{ людино-годин.} \quad (3.6)$$

$$t_a = \frac{1914}{20*0,8} = 119, \text{ людино-годин.} \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25)K} \text{ людино-годин.} \quad (3.8)$$

$$t_n = \frac{1914}{25*0,8} = 96, \text{ людино-годин.} \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{\text{отл}} = \frac{Q}{(4..5)K} \text{ людино-годин.} \quad (3.10)$$

$$t_{\text{отл}} = \frac{1914}{4*0,8} = 598, \text{ людино-годин.} \quad (3.11)$$

за умови комплексного налагодження завдання:

$$t^k_{\text{отл}} = 1,2 * t_{\text{отл}}; \quad (3.12)$$

$$t_{\text{отл}} = 598*1,2 = 717,6 \quad (3.13)$$

Витрати праці на підготовку документації:

$$t_д = t_{др}+t_{до}, \text{ людино-годин,} \quad (3.14)$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15..20)K}, \text{ людино-годин.} \quad (3.15)$$

$$t_{др} = \frac{1914}{15*0,8} = 160 \text{ людино-годин,} \quad (3.16)$$

tдо - трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин.} \quad (3.17)$$

$$t_{do} = 0{,}75 \cdot 160 = 120 \quad (3.18)$$

$$t_{д} = 160+120 = 280 \text{ людино-годин.} \quad (3.19)$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 36+119+96+717+160+280 = 1408 \text{ людино-годин.} \quad (3.20)$$

3.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ , грн,} \quad (3.21)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР} \text{ , грн,} \quad (3.22)$$

де:

t - загальна трудомісткість, людино-годин;

Спр - середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 1408 \cdot 40 = 56320 \text{грн.} \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = tотл \times C_М \text{ , грн,} \quad (3.24)$$

де:

tотл - трудомісткість налагодження програми на ЕОМ, год.

См - вартість машино-години ЕОМ, 7 грн/год.

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

$$З_{МВ} = 717 \times 7 = 5019 \text{ грн.} \quad (3.25)$$

$$Кпо = 56320 + 5019 = 61339 \text{ грн.} \quad (3.26)$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p} \text{міс.} \quad (3.27)$$

де:

Bk - число виконавців;

Fp - місячний фонд робочого часу (при 50 годинному робочому тижні Fp=225 годин).

$$T = \frac{1480}{1*225} = 6{,}5 \text{міс.} \quad (3.28)$$

Визначено трудомісткість розробленої інформаційної системи (1480 людгод), проведений підрахунок вартості роботи по створенню програми (61339 грн.) та розраховано час на його створення (6,5 міс).

ВИСНОВКИ

Було розроблено серверну частини для web та Android застосунків. Були виконані такі завдання основні завдання:

- аналіз організації роботи адміністрації та основні виконувані задачі та на основі аналізу виділити вимоги до системи;

- проектування алгоритмів, бази даних, програмну реалізацію та перевірку тестуванням й відлагодження створеної системи;

- уточнення вимог до виконуваних процесів;

- автоматизація ведення даних, зниження кількості помилок при занесенні та прискорення роботи адміністрації гуртожитку.

В процесі виконання було виявлено, що мало інформації є у вільному доступі по даній темі. Проаналізовані джерела не дали майже ніякої інформації. Подібні програми написані на інших мовах і з іншим підходом.

Створена система відображає точки зору: точка зору адміністратора, для якого дуже важлива можливість швидкої та простої зміни інформації, і точка зору співробітника або студента, для яких важливий простий і швидкий перегляд необхідної інформації, та обробка їх запитів. Механізм захисту бази даних від несанкціонованого доступу здійснюється шифруванням паролів для входу в програму.

Запропонована програма написана на об'єктно-орієнтованій мові програмування Java в середовищі програмування розробки Intellij IDEA. Проведені тестування та дослідження сервера показали його практичність, безвідказність , тобто можливість обробляти дані. В економічному розділі визначено трудомісткість розробленої інформаційної системи 1480 людино-годин, проведений підрахунок вартості роботи по створенню програми 61339 гривень та розраховано час на його створення 6,5 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп'ютерні науки /, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : НТУ «Дніпровська політехніка», 2020. – 65 с.

2. Динеш Раджпут Spring. Все паттерны проектирования. — СПб.: Питер, 2019. — 320 с.: ил. — (Се-рія «Бібліотека програміста»).

3. Олег Докука, Игорь Лозинский Практика реактивного программирования в Spring 5. – М.: ДМК Пресс, 2019. – 508 с.

4. Глобальная сеть рефератов – URL: https://otherreferats.allbest.ru/radio/00312640_0.html/ . Дата звернення 26.02.2020

5. Файловий архів студентів – URL: https://studfiles.net/preview/4452619// Дата звернення 26.02.202

6. Mercan Topkara, Cuneyt M. Taskiran, and Edward J.Delp.2005.Natural language watermarking.InProceedings of the SPIE Conference on Secu-rity, Steganography, and Watermarking of Multimedia Contents, volume 5681, pages 441–452, San Jose, CA.

7. Spring Boot – URL: https://spring.io/projects/spring-boot

8. Spring Initializr – URL: https://start.spring.io/

9. Spring Initializer Reference Guid – URL: https://docs.spring.io/initializr/docs/current/reference/html/

10. Maven Repository URL: https://mvnrepository.com/

11. Introduction to the Dependency Mechanism – URL: https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

12. Spring Security Reference URL: – https://docs.spring.io/spring-security/site/docs/current/reference/html5/

13. IntelliJ IDEA Ultimate – URL: https://plugins.jetbrains.com/idea

14. Spring Rest API URL: https://spring.io/guides/gs/rest-service/

15. Spring Common Application Properties – URL: https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

16. Документація до PostgreSQL 13.3 The PostgreSQL Global Development Group Перевод на українську мову, 2015-2021

17. PostgreSQL Documentation – URL: https://www.postgresql.org/docs/

18. Ресурси Intellij IDEA – URL: https://www.jetbrains.com/ru-ru/idea/resources/

19. Spring Boot Starter Data JPA – URL: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa

20. Validation in Spring Boot – URL: https://www.baeldung.com/spring-boot-bean-validation

21. Validation, Data Binding, and Type Conversion – URL: https://docs.spring.io/spring-framework/docs/4.1.x/spring-framework-reference/html/validation.html

22. Serving Web Content with Spring MVC – URL: https://spring.io/guides/gs/serving-web-content/

Додаток А

КОД ПРОГРАМИ

```
@Getter
@Setter
@Entity
@Table(name = "users")
@Data
public class Users  {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column
    @Size(min=5, message = "Не меньше 5 знаков")
    private String username;
    @Column
    @Size(min = 5,message = "Не меньше 5 знаков")
    private String password;
    @Transient
    private String passwordConfirm;
    @ManyToOne
    @JoinColumn(name = "role_id")
    private Role role;
    @OneToOne(mappedBy = "users",cascade = CascadeType.ALL)
    @JsonIgnore
    private Residents residents;
    @OneToMany(mappedBy = "users")
    @JsonIgnore
    private Set<ApplicationsForAccommodation> applicationsForAccommodation;
    public Users() {
```

```
    }
}

@Setter
@Getter
@Entity
@Table(name = "applications_for_accommodation")
public class ApplicationsForAccommodation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Column(name = "nameSurnameLastname")
    private String nameSurnameLastname;
    @Column(name = "faculty")
    private String faculty;
    @Column(name = "groupIn")
    private String groupIn;
    @Column(name = "phoneNumber")
    private String phoneNumber;
    @Column
    private Date creationDate;
    @Column(name = "status")
    private String status;
    @Column
    private Date dateOfChange;
    @ManyToOne
    @JsonIgnore
    @PrimaryKeyJoinColumn
    @JoinColumn(name = "users_id")
```

```
    private Users users;
    @ManyToOne
    @JsonIgnore
    @PrimaryKeyJoinColumn
    @JoinColumn(name = "rooms_id")
    private Rooms rooms;
}

@Getter
@Setter
@Entity
@Table(name = "floors")
public class Floors {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Column
    @NonNull
    private short numberFloor;
    @Column
    @NonNull
    private int numberOfRoomsPerFloor;
    @Column
    private int numberOfFreeRoomsOnTheFloor;
    @OneToMany(mappedBy = "floors")
    @JsonIgnore
    private Set<Rooms> rooms;
    public Floors() {
    }
```

```
    public Floors(short numberFloor, int numberOfRoomsPerFloor, int
numberOfFreeRoomsOnTheFloor, Set<Rooms> rooms) {
        this.numberFloor = numberFloor;
        this.numberOfRoomsPerFloor = numberOfRoomsPerFloor;
        this.numberOfFreeRoomsOnTheFloor = numberOfFreeRoomsOnTheFloor;
        this.rooms = rooms;
    }
    @Override
    public String toString() {
        return "Floors{" +
                "id=" + id +
                ", numberFloor=" + numberFloor +
                ", numberOfRoomsPerFloor=" + numberOfRoomsPerFloor +
                ", numberOfFreeRoomsOnTheFloor=" +
numberOfFreeRoomsOnTheFloor +
                ", rooms=" + rooms +
                '}';
    }
}

@Getter
@Setter
@Entity
@Table(name = "rooms")
public class Rooms {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Column
```

```
    private long numberRoom;
    @Column
    private short numberOfSeatsInTheRoom;
    @Column
    private short numberOfFreePlacesInTheRoom;
    @Column
    private String description;
    @OneToMany(mappedBy = "rooms")
    @JsonIgnore
    @PrimaryKeyJoinColumn
    private Set<Residents> residents;
    @ManyToOne
    @JsonIgnore
    @PrimaryKeyJoinColumn
    @JoinColumn(name = "floor_id")
    private Floors floors;

    @OneToMany(mappedBy = "rooms")
    @JsonIgnore
    @PrimaryKeyJoinColumn
    private Set<ApplicationsForAccommodation> applicationsForAccommodation;
    public Rooms() {
    }
    public Rooms(long numberRoom, short numberOfSeatsInTheRoom, String
description) {
        this.numberRoom = numberRoom;
        this.numberOfSeatsInTheRoom = numberOfSeatsInTheRoom;
        this.description = description;
```

```
    }
}
@Entity
@Getter
@Setter
@Table(name = "t_role")
@Data
public class Role  {
    @Id
    private Long id;
    @Column
    private String name;
    public Role() {
    }
    public Role(Long id, String name) {
        this.id = id;
        this.name = name;
    }
}

@Entity
@Table(name = "residents")
public class Residents {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Column
    private String name;
    @Column
```

```
    private String surname;
    @Column
    private String lastname;
    @Column
    private String faculty;
    @Column(name = "groupIn")
    private String groupIn;
    @Column(unique = true)
    private String phoneNumber;
    @Column
    private String registration;
    @Column
    private Date dateCheckIn;
    @Column
    private Date dateCheckOut;
@OneToOne(cascade = CascadeType.ALL)
@JsonIgnore
@JoinColumn(name = "users_id")
    private Users users;
@ManyToOne(cascade = CascadeType.ALL)
@JsonIgnore
@JoinColumn(name = "rooms_id")
private Rooms rooms;
    public Residents() {
    }
    public Residents(String name, String surname, String lastname, String faculty,
String groupIn, String phoneNumber, String registration, Users users) {
        this.name = name;
        this.surname = surname;
```

```
        this.lastname = lastname;
        this.faculty = faculty;
        this.groupIn = groupIn;
        this.phoneNumber = phoneNumber;
        this.registration = registration;
        this.users = users;
    }
}

@Repository
public interface UserRepository extends JpaRepository<Users,Long> {
    Users findByUsername(String username);
    List<Users> findByResidents_FacultyContaining(String Faculty);
    List<Users> findByResidents_GroupInContaining(String groupIn);
    List<Users> findByResidents_SurnameContaining(String surname);
}

@Repository
public interface RoleRepository extends JpaRepository<Role,Long> {
    Role findByName(String name);
}

@Repository
public interface ResidentsRepository extends JpaRepository<Residents,Long> {
    Residents findByUsers(Users users);
    Residents findByPhoneNumber(String phoneNumber);
    List<Residents> findResidentsByRooms_Id(long roomId);
    List<Residents> findResidentsByRooms_NumberRoom(long numberRoom);
    List<Residents> findResidentsByFacultyContaining(String faculty);
```

```
    List<Residents> findResidentsByGroupInContaining(String groupIn);
    List<Residents> findResidentsBySurnameContaining(String surname);
}

@Repository
public interface RoomsRepository extends JpaRepository<Rooms,Long> {
    Rooms findByNumberRoom(long numberRoom);
    List<Rooms> findRoomsByFloors_Id(long floorId);
    Long deleteByNumberRoom(long numberFloor);
}

@Repository
public interface FloorsRepository extends JpaRepository<Floors,Long> {
    Floors findByNumberFloor(short numberFloor);
    Floors findByRooms(Rooms rooms);
}

@Repository
public interface ApplicationsForAccommodationRepository extends
JpaRepository<ApplicationsForAccommodation,Long> {
    List<ApplicationsForAccommodation> findByUsers_Username(String
username);
    List<ApplicationsForAccommodation> findByRooms_Id(long roomsId);
    ApplicationsForAccommodation findById(long applicationId);
}

@Service
public class UserService  {
```

```
    @PersistenceContext
    private EntityManager em;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private RoleRepository roleRepository;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;
    @Autowired
    private ResidentsRepository residentsRepository;
    @Autowired
    private RoomService roomService;


    public Users findByUsername(String username) {
        return userRepository.findByUsername(username);
    }



    public Users findByUsernameAndPassword(String username, String password)
{
        Users users = findByUsername(username);
        if (users != null) {
            if (bCryptPasswordEncoder.matches(password, users.getPassword())) {
                return users;
            }
        }
        return null;
    }
```

```
    public List<Residents> allResidents() {
        return residentsRepository.findAll();
    }

    public List<Residents> residentList(String username) {
        System.out.println(username);
        return em.createQuery("SELECT r FROM Residents r WHERE
r.users.username =:username", Residents.class)
                .setParameter("username", username).getResultList();
    }
    public boolean saveResident(String name,String lastname,String surname,
                                String faculty,String groupIn,String phoneNumber,
                                String registration,String username){
        Users user =userRepository.findByUsername(username);
        Residents residentFromDb =
residentsRepository.findByPhoneNumber(phoneNumber);
        if(residentFromDb!=null){
            return false;
        }
        Residents resident=new
Residents(name,surname,lastname,faculty,groupIn,phoneNumber,registration,user)
;
        residentsRepository.save(resident);
        return true;
    }
    public boolean saveUser(Users user){
        Role role = roleRepository.findByName("ROLE_UNVERIFIED");
        Users userFromDb =userRepository.findByUsername(user.getUsername());
```

```
        if (userFromDb!=null){
            return false;
        }
        user.setRole(role);
user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return true;
    }

    public Users findUserById(Long userId) {
        Optional<Users> userFromDb = userRepository.findById(userId);
        return userFromDb.orElse(new Users());
    }
    public List<Users> allUsers() {
        return userRepository.findAll();
    }
    public boolean deleteUser(Long userId) {
        if (userRepository.findById(userId).isPresent()) {
            userRepository.deleteById(userId);
            return true;
        }
        return false;
    }
    public List<Users> userList(Long idMin) {
        return em.createQuery("SELECT u FROM Users u WHERE u.id >
:paramId", Users.class)
                .setParameter("paramId", idMin).getResultList();
    }
public List<Residents> studentList() {
```

```
    return em.createQuery("SELECT r FROM Residents r JOIN r.users.role rl
WHERE rl.name= 'ROLE_STUDENT' ", Residents.class)
            .getResultList();
}
public boolean moveOutMoveTo(String action, Long residentId, Long
numberRoom){
    Residents residentFromDb = residentsRepository.findById(residentId).get();
        if (action.equals("moveOut")&&numberRoom==null){
            residentFromDb.setRooms(null);
            residentsRepository.save(residentFromDb);
            return true;
        }
        if (action.equals("moveTo")&&
roomService.presenceOfSuchRoom(numberRoom)){
            if (!roomService.fullRoom(numberRoom)) {
                residentFromDb.setRooms(roomService.giveRoom(numberRoom));
                residentsRepository.save(residentFromDb);
                return true;
            }
        }
        return false;
}
public boolean acceptRejectAccount(String action,Long userId){
        Users userFromDb = userRepository.findById(userId).get();
        if (userFromDb!=null){
            if (action.equals("accept")) {
                userFromDb.setRole(null);
                Role role = roleRepository.findByName("ROLE_STUDENT");
                userFromDb.setRole(role);
```

```
            userRepository.save(userFromDb);
            return true;
        }
        if (action.equals("reject")) {
            userFromDb.setRole(null);
            Role role = roleRepository.findByName("ROLE_GUEST");
            userFromDb.setRole(role);
            userRepository.save(userFromDb);
            return true;
        }
    }
    return false;
}
  @Transactional
  public List<Users> unverifiedUsers() {
    return em.createQuery("SELECT u FROM Users u JOIN u.role ur WHERE
ur.name ='ROLE_UNVERIFIED'", Users.class)
            .getResultList();
  }
  public List<Users> studentUsers() {
    return em.createQuery("SELECT u FROM Users u JOIN u.role ur WHERE
ur.name ='ROLE_STUDENT'", Users.class)
            .getResultList();
  }
    public List<Users> studentUsersListFilterByFaculty(String faculty){
    return userRepository.findByResidents_FacultyContaining(faculty);
  }
    public List<Users> studentUsersListFilterByGroupIn(String groupIn){
    return userRepository.findByResidents_GroupInContaining(groupIn);
```

```
    }
    public List<Users> studentUsersListFilterBySurname(String surname){
        return userRepository.findByResidents_SurnameContaining(surname);
    }
    public List<Residents> residentsListFilterByRoom(Long numberRoom) {
        return em.createQuery("SELECT r FROM Residents r WHERE
r.rooms.numberRoom =:numberR ", Residents.class)
                .setParameter("numberR", numberRoom).getResultList();
    }
    public List<Residents> residentsListFilterByFloor(Short numberFloor) {
        return em.createQuery("SELECT r FROM Residents r WHERE
r.rooms.floors.numberFloor =:numberFloor ", Residents.class)
                .setParameter("numberFloor", numberFloor).getResultList();
    }
    public List<Residents> residentsListFilterByFaculty(String faculty) {
        return residentsRepository.findResidentsByFacultyContaining(faculty);
    }
    public List<Residents> residentsListFilterByGroupIn(String groupIn) {
        return residentsRepository.findResidentsByGroupInContaining(groupIn);
    }

    public List<Residents> residentsListFilterBySurname(String surname) {
        return residentsRepository.findResidentsBySurnameContaining(surname);
    }
}
@Service
public class RoomService {

    @Autowired
```

```
    private EntityManager em;

    @Autowired
    private FloorsRepository floorsRepository;

    @Autowired
    private RoomsRepository roomsRepository;

    @Autowired
    private ResidentsRepository residentsRepository;

    public boolean chekResident(long numberRoom){
        if
(residentsRepository.findResidentsByRooms_NumberRoom(numberRoom).size()
>0){
            return false;
        }
        return true;
    }
    @Transactional
    public boolean deleteByNumber(long numberRoomForDell){
            long
param=roomsRepository.deleteByNumberRoom(numberRoomForDell);
        return true;
    }

    public Rooms giveRoom(long numberRoom){
        Rooms roomFromDb =
roomsRepository.findByNumberRoom(numberRoom);
        return roomFromDb;
```

```
    }
    public List<Rooms> allRooms(){return roomsRepository.findAll();}
    public boolean saveRoom(Rooms room, short numberFloor){
        Floors floor = floorsRepository.findByNumberFloor(numberFloor);
        Rooms roomFromDb =
roomsRepository.findByNumberRoom(room.getNumberRoom());
        if (floor==null){
            return false;
        }
        if (roomFromDb!=null){
            return false;
        }
        room.setFloors(floor);
        roomsRepository.save(room);
        return true;
    }
    public List<Rooms> roomList(Long idMin) {
        return em.createQuery("SELECT r FROM Rooms r WHERE r.id >
:paramId", Rooms.class)
                .setParameter("paramId", idMin).getResultList();
    }
    public boolean deleteRoom(Long roomId){
        if (residentsRepository.findResidentsByRooms_Id(roomId).size()>0){
            return false;
        }
        if (roomsRepository.findById(roomId).isPresent()){
            roomsRepository.deleteById(roomId);
            return true;
        }
```

```
        return false;
    }
    public List<Rooms> roomFreeList() {
        return em.createQuery("SELECT r FROM Rooms r WHERE
r.numberOfFreePlacesInTheRoom > 0 order by r.numberRoom", Rooms.class)
                .getResultList();
    }
    public List<Rooms> roomFoolPlaceList() {
        return em.createQuery("SELECT r FROM Rooms r WHERE
r.numberOfFreePlacesInTheRoom > r.numberOfSeatsInTheRoom order by
r.numberRoom", Rooms.class)
                .getResultList();
    }
public boolean checkTheNumberOfFreeSeats(){
        List<Rooms> roomsList = roomsRepository.findAll();
    for (Rooms o:roomsList) {
        Rooms room = roomsRepository.findById(o.getId()).get();
        room.setNumberOfFreePlacesInTheRoom((short)
(o.getNumberOfSeatsInTheRoom()-
residentsRepository.findResidentsByRooms_Id(o.getId()).size()));
        System.out.println(o.getId()+" "+o.getNumberOfSeatsInTheRoom()+"
"+residentsRepository.findResidentsByRooms_Id(o.getId()).size()+"
"+o.getNumberOfFreePlacesInTheRoom());
        roomsRepository.save(room);
    }
        return true;
}
public boolean presenceOfSuchRoom(long numberRoom){
        if (roomsRepository.findByNumberRoom(numberRoom) == null){
```

```
            return false;
        }
        return true;
}
public boolean fullRoom(long numberRoom){
        int numb =
roomsRepository.findByNumberRoom(numberRoom).getNumberOfFreePlacesInT
heRoom();
    if
(roomsRepository.findByNumberRoom(numberRoom).getNumberOfFreePlacesIn
TheRoom()==0){
        return  true;
    }
        return false;
}
}

@Service
public class FloorsService {
    @Autowired
    private RoomService roomService;
    @Autowired
    private FloorsRepository floorsRepository;
    @Autowired
    private RoomsRepository roomsRepository;
    @PersistenceContext
    private EntityManager em;
    public boolean checkRoomsAtFloor(long floorId ){
        List<Rooms> rooms = roomsRepository.findRoomsByFloors_Id(floorId);
```

```
        if(rooms.size()>0){
            return false;
        }
        return true;
    }
    public boolean saveFloor(Floors floor){
        Floors floorFromDb =
floorsRepository.findByNumberFloor(floor.getNumberFloor());
        if (floorFromDb!=null){
            return false;
        }
        floorsRepository.save(floor);
        return true;
    }
    public List<Floors> allFloors(){return floorsRepository.findAll();}
    public boolean deleteFloor(Long floorId) {
        if (floorsRepository.findById(floorId).isPresent()){
            floorsRepository.deleteById(floorId);
            return true;
        }
        return false;
    }
    public List<Floors> florList(Long idMin) {
        return em.createQuery("SELECT f FROM Floors f WHERE f.id > :paramId",
Floors.class)
                .setParameter("paramId", idMin).getResultList();
    }
    public boolean checkFreeRooms(){
        List<Floors> floorsList = floorsRepository.findAll();
```

```
        for (Floors floors:floorsList) {
            Floors floor = floorsRepository.findById(floors.getId()).get();
floor.setNumberOfFreeRoomsOnTheFloor(floor.getNumberOfRoomsPerFloor()-
roomService.roomFoolPlaceList().size());
            floorsRepository.save(floor);
        }
        return true;
    }
    public Floors foundFloor(Rooms room){
        Floors floor= floorsRepository.findByRooms(room);
        return floor;
    }
}
@Service
public class ApplicationsForAccommodationService {
    @Autowired
    private ApplicationsForAccommodationRepository
applicationsForAccommodationRepository;
    @PersistenceContext
    private EntityManager em;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private RoomsRepository roomsRepository;
    public boolean chekSeatAvailability(long numberRoom){
if(roomsRepository.findByNumberRoom(numberRoom).getNumberOfFreePlacesI
nTheRoom()>0){
            return true;
        }
```

```
        return false;
    }
    public boolean save(ApplicationsForAccommodation
applicationsForAccommodation,
                        String username,
                        long numberRoom) {
List<ApplicationsForAccommodation>listApplicationsForAccommodations =
applicationsForAccommodationRepository.findByUsers_Username(username);
        if (listApplicationsForAccommodations.size()>=3){
            return false;
        }
        Rooms roomFromDb =
roomsRepository.findByNumberRoom(numberRoom);
        Users userFromDb = userRepository.findByUsername(username);
        Date dateCreate = new Date();
        applicationsForAccommodation.setUsers(userFromDb);
        applicationsForAccommodation.setRooms(roomFromDb);
applicationsForAccommodation.setCreationDate(dateCreate);
applicationsForAccommodationRepository.save(applicationsForAccommodation);
        return true;
    }
    public List<ApplicationsForAccommodation> applicationList(Long idMin) {
        return em.createQuery("SELECT a FROM ApplicationsForAccommodation a
WHERE a.id > :paramId", ApplicationsForAccommodation.class)
                .setParameter("paramId", idMin).getResultList();
    }
    public List<ApplicationsForAccommodation>
allApplicationsForAccommodations() {
        return applicationsForAccommodationRepository.findAll();
```

```
    }
    public List<ApplicationsForAccommodation> unverifiedApplicationList() {
        return em.createQuery("SELECT a FROM ApplicationsForAccommodation a
WHERE  a.status is null", ApplicationsForAccommodation.class)
                .getResultList();
    }
public List<ApplicationsForAccommodation> personalApplicationsList(String
username) {
    return em.createQuery("SELECT a FROM ApplicationsForAccommodation a
WHERE  a.users.username =:userName", ApplicationsForAccommodation.class)
            .setParameter("userName", username).getResultList();
}
    public List<ApplicationsForAccommodation> acceptApplicationList() {
        return em.createQuery("SELECT a FROM ApplicationsForAccommodation a
WHERE  a.status ='accept' ", ApplicationsForAccommodation.class)
                .getResultList();
    }
    public List<ApplicationsForAccommodation> deflectApplicationList() {
        return em.createQuery("SELECT a FROM ApplicationsForAccommodation a
WHERE  a.status ='deflect' ", ApplicationsForAccommodation.class)
                .getResultList();
    }
    public boolean makingDecisionOnApplication(Long applicationId,String
action){
        Date dateOfChange = new Date();
        if
(applicationsForAccommodationRepository.findById(applicationId).isPresent()){
            if (action.equals("deflect")) {
                ApplicationsForAccommodation ap =
```

```
applicationsForAccommodationRepository.findById(applicationId).get();
                ap.setStatus(action);
                ap.setDateOfChange(dateOfChange);
applicationsForAccommodationRepository.save(ap);
                return true;
            }
            if (action.equals("accept")){
                ApplicationsForAccommodation ap =
applicationsForAccommodationRepository.findById(applicationId).get();
                ap.setStatus(action);
                ap.setDateOfChange(dateOfChange);
                applicationsForAccommodationRepository.save(ap);
                return true;
            }
        }
        return false;
    }
}
@Controller
public class RoomController {
    @Autowired
    private RoomService roomService;
    @Autowired
    private FloorsRepository floorsRepository;

    @GetMapping("/room")
    public String room(Model model){
        model.addAttribute("roomAdd",new Rooms());
       model.addAttribute("allRoom",roomService.allRooms());
```

```
        return "room";
    }
    @PostMapping("/addRoom")
        public String addRoom(@ModelAttribute("roomAdd") Rooms room,
                              @RequestParam short numberFloor,
                              Model model){
        if (!roomService.saveRoom(room,numberFloor)){
            if (floorsRepository.findByNumberFloor(numberFloor)==null){
                model.addAttribute("floorError","Такого этажа нет");
                return "room";
            }
            model.addAttribute("roomError","Такая комната уже существует");
            return "room";
        }
        roomService.saveRoom(room,numberFloor);
        roomService.checkTheNumberOfFreeSeats();
        return "redirect:/room";
    }
    @PostMapping("/room")
    private String  deleteRoom(@RequestParam(required = true, defaultValue = "" )
Long roomId,
                               @RequestParam(required = true, defaultValue = "" ) String
action,
                               RedirectAttributes redirectAttributes) {
        if (action.equals("delete")){
            if (!roomService.deleteRoom(roomId)){
redirectAttributes.addFlashAttribute("forAdminMessage","Щоб видалити
кімнату, спершу виселіть і проінформуйте мешканців");
                return "redirect:/room";
```

```
            }
            roomService.deleteRoom(roomId);
        }
        return "redirect:/room";
    }
    @GetMapping("/room/gt/{roomsId}")
    private String  gtFloor(@PathVariable("roomId") Long roomId, Model model) {
        model.addAttribute("allRoom", roomService.roomList(roomId));
        return "room";
    }
}
@Controller
public class RegistrationController {
    @Autowired
    private UserService userService;
    @GetMapping("/registration")
    public String registration(Model model) {
        model.addAttribute("userForm", new Users());
        return "registration";
    }
    @PostMapping("/registration")
    public String addUser(@Valid @ModelAttribute ("userForm") Users userForm,
                          @RequestParam String name,
                          @RequestParam String lastname,
                          @RequestParam String surname,
                          @RequestParam String faculty,
                          @RequestParam String groupIn,
                          @RequestParam String phoneNumber,
                          @RequestParam String registration,
```

```
                    BindingResult bindingResult, Model model) throws Exception{
        if(bindingResult.hasErrors()){
            return "registration";
        }
        if (!userForm.getPassword().equals(userForm.getPasswordConfirm())){
            model.addAttribute("passwordError", "Пароли не совпадают");
            return "registration";
        }
        if (!userService.saveUser(userForm)){
            model.addAttribute("usernameError", "Пользователь с таким именем
уже существует");
            return "registration";
        }
        userService.saveUser(userForm);

userService.saveResident(name,lastname,surname,faculty,groupIn,phoneNumber,r
egistration,userForm.getUsername());
        return "redirect:/";
    }
    }
@Controller
public class FloorsController {
    @Autowired
    private FloorsService floorsService;
    @Autowired
    private FloorsRepository floorsRepository;
    @GetMapping("/floors")
    public String floor(Model model){
        model.addAttribute("floorAdd",new Floors());
```

```
        model.addAttribute("allFloors", floorsService.allFloors());
        return "floors";
    }
    @PostMapping("/floors")
    public String floorAdd(@Valid @ModelAttribute("floorAdd") Floors floor,
                           Model model
    ){
        if (!floorsService.saveFloor(floor)){
            model.addAttribute("floorError","Такой этаж уже существует");
            return "redirect:/floors";
        }
            floorsService.saveFloor(floor);
        floorsService.checkFreeRooms();
        return "redirect:/floors";
    }
        @GetMapping("/floors/gt/{floorsId}")
        private String  getFloor(@PathVariable("floorId") Long florId, Model model)
{
            model.addAttribute("allFloors", floorsService.florList(florId));
            return "floors";
        }
    @PostMapping("/floorDelete")
    private String  deleteFloor(@RequestParam(required = true, defaultValue = "" )
Long floorId,
                                @RequestParam(required = true, defaultValue = "" ) String
action,
                                Model model, RedirectAttributes redirectAttributes) {
        if (action.equals("delete")){
            if (!floorsService.checkRoomsAtFloor(floorId)){
```

```
            model.addAttribute("forAdmin","Вы не можите удалить этаж пока не разберетесь с камнатами, жильцами и заявками на заселение ");
redirectAttributes.addFlashAttribute("forAdmin","Вы не можите удалить этаж пока не разберетесь с камнатами, жильцами и заявками на заселение ");
            return "redirect:/floors";
        }
        floorsService.deleteFloor(floorId);
    }
    return "redirect:/floors";
  }
}
@Controller
public class Control {
  @Autowired
  private FloorsService floorsService;
  @Autowired
  private RoomService roomService;
  @Autowired
  private ApplicationsForAccommodationService
applicationsForAccommodationService;
  @Autowired
  private UserService userService;
  @GetMapping("/")
  public String index(){
    roomService.checkTheNumberOfFreeSeats();
    floorsService.checkFreeRooms();
    return "redirect:/first";
  }
  @GetMapping("/first")
```

```
    public String first(Model model){
model.addAttribute("freeRoomForAll",roomService.roomFreeList());
        return "first";
    }
    @GetMapping("/personalArea")
    public String personalArea(Model model,Principal principal){
        System.out.println(principal.getName());

model.addAttribute("userInfo",userService.residentList(principal.getName()));
model.addAttribute("userApplicatonsList",applicationsForAccommodationService.
personalApplicationsList(principal.getName()));
        return "personalArea";
    }
}
@Controller
public class AuthorizedUserController {
    @Autowired
    private RoomService roomService;
    @GetMapping("/news")
    public String studentNews(Model model){
model.addAttribute("freeRoomForAuthorized",roomService.roomFreeList());
        return "news";
    }
}
@Controller
public class ApplicationsForAccommodationControoler {
    @Autowired
    private ApplicationsForAccommodationService
applicationsForAccommodationService;
```

```
    @Autowired
    private RoomService roomService;
    @GetMapping("/applicationsForAccommodation")
    public String applicationsForAccommodation( Model model){
model.addAttribute("applicationsForAccommodationForm",new
ApplicationsForAccommodation());
        return "applicationsForAccommodation";
    }
    @PostMapping("/applicationsForAccommodation")
    public String
saveApplicationsForAccommodation(@ModelAttribute("applicationsForAccomm
odationForm")ApplicationsForAccommodation applicationsForAccommodation,
@RequestParam long numberRoom,
                                     BindingResult bindingResult, Model model,
Principal principal){
        if(bindingResult.hasErrors()){
            return "applicationsForAccommodation";
        }
        if (!roomService.presenceOfSuchRoom(numberRoom)){
            model.addAttribute("roomError","Такої кімнати не існує");
            return "applicationsForAccommodation";
        }
        if
(!applicationsForAccommodationService.chekSeatAvailability(numberRoom)){
            model.addAttribute("roomError","У кімнаті нема місць, перегляньте і
виберіть іншу кімнату");
            return "applicationsForAccommodation";
        }
        if
```

```
(!applicationsForAccommodationService.save(applicationsForAccommodation,principal.getName(),numberRoom)){
            model.addAttribute("applicationError","Ви перевищили кількість заявок (5 штук)");
            return "applicationsForAccommodation";
        }
applicationsForAccommodationService.save(applicationsForAccommodation, principal.getName(), numberRoom);
        return "redirect:/news";
    }
    @GetMapping("/applications")
    public String applications(Model model){
        model.addAttribute("unverifiedApplications", applicationsForAccommodationService.unverifiedApplicationList());
model.addAttribute("deflectApplications",applicationsForAccommodationService.deflectApplicationList());
model.addAttribute("acceptApplications",applicationsForAccommodationService.acceptApplicationList());
        return "applications";
    }
    @GetMapping("/applications/gt/{applicationId}")
    private String  gtFloor(@PathVariable("roomId") Long applicationId, Model model) {
        model.addAttribute("allRoom", applicationsForAccommodationService.applicationList(applicationId));
        return "applications";
    }
    @PostMapping("/applications")
    private String  makingDecisionOnApplication(@RequestParam(required = true,
```

```
defaultValue = "" ) Long applicationId,
                        @RequestParam(required = true, defaultValue = "" ) String
action)
applicationsForAccommodationService.makingDecisionOnApplication(application
Id, action);
        return "redirect:/applications";
    }
}
@Controller
public class AdminController {
    @Autowired
    private UserService userService;
    @Autowired
    private RoomService roomService;
    @GetMapping("/admin")
    private String userList(Model model,
                        @RequestParam (value = "facultyForFilter", required = false)
String facultyForFilter,
                        @RequestParam(value = "groupInForFilter",required =
false)String groupInForFilter,
                        @RequestParam(value = "surnameForFilter",required =
false)String surnameForFilter) {
        model.addAttribute("unverifiedUsers", userService.unverifiedUsers());
        model.addAttribute("studentUsers", userService.studentUsers());
        if (facultyForFilter!=null) {
model.addAttribute("studentUsers",userService.studentUsersListFilterByFaculty(f
acultyForFilter));
        }
        if (groupInForFilter!=null) {
```

```
model.addAttribute("studentUsers",userService.studentUsersListFilterByGroupIn(
groupInForFilter));
        }
        if (surnameForFilter!=null) {
model.addAttribute("studentUsers",userService.studentUsersListFilterBySurname(
surnameForFilter));
        }
        return "admin";
    }
    @GetMapping("/admin/gt/{userId}")
    private String  gtUser(@PathVariable("userId") Long userId, Model model) {
        model.addAttribute("allUser", userService.userList(userId));
        return "admin";
    }
    @PostMapping("/admin")
    private String  acceptRejectUser(@RequestParam(required = true, defaultValue
= "" ) Long userId,
                                     @RequestParam(required = true, defaultValue = "" ) String
action,
                                     Model model) {
        userService.acceptRejectAccount(action,userId);
        return "redirect:/admin";
    }
    @GetMapping("/controlResidents")
    public String showAllResident(Model model,
                                  @RequestParam(value = "numberRoomForFilter",required =
false) Long numberRoomForFilter,
                                  @RequestParam(value = "numberFloorForFilter",required =
false) Short numberFloorForFilter,
```

```
                    @RequestParam(value = "facultyForFilter",required = false)
String facultyForFilter,
                    @RequestParam(value = "groupInForFilter",required = false)
String groupInForFilter,
                    @RequestParam(value = "surnameForFilter",required = false)
String surnameForFilter){
    roomService.checkTheNumberOfFreeSeats();
model.addAttribute("allStudent",userService.studentList());
    if (numberRoomForFilter != null) {
model.addAttribute("allStudent",userService.residentsListFilterByRoom(numberR
oomForFilter));
    }
    if (numberFloorForFilter != null) {
model.addAttribute("allStudent",userService.residentsListFilterByFloor(numberFl
oorForFilter));
    }
    if (facultyForFilter != null) {
model.addAttribute("allStudent",userService.residentsListFilterByFaculty(facultyF
orFilter));
    }
    if (groupInForFilter != null) {
model.addAttribute("allStudent",userService.residentsListFilterByGroupIn(groupI
nForFilter));
    }
    if (surnameForFilter != null) {
model.addAttribute("allStudent",userService.residentsListFilterBySurname(surna
meForFilter));
    }
model.addAttribute("allStudent",userService.residentsListFilterByRoom());
```

```
        return "controlResidents";
    }
    @PostMapping("/controlResidents")
    public String controlResident(String action, Long residentId, Long
numberRoom, RedirectAttributes redirectAttributes){
redirectAttributes.addFlashAttribute("roomFulError","Удостовертесь что в
комнате есть места");
        if (!userService.moveOutMoveTo(action, residentId, numberRoom)){
redirectAttributes.addFlashAttribute("roomError","Такої кімнати не існує");
            return "redirect:/controlResidents";
        }       userService.moveOutMoveTo(action,residentId,numberRoom);
        return "redirect:/controlResidents";
    }
}
@Configuration
public class MvcConfig implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
registry.addViewController("/login").setViewName("login");
registry.addViewController("/").setViewName("index");
registry.addViewController("/first").setViewName("first");
    }
}
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка серверної частини системи обліку розселення студентів у гуртожитку НТУ «Дніпровська політехніка»»

студента групи 122-18ск-2 Безбородька Сергія Володимировича

Керівник економічного розділу **О.Г.Вагонова**

Зав. каф. ПЕП та ПУ, д.е.н

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Безбородько.doc	Пояснювальна записка до проекту. Документ Word.
Диплом_Безбородько.pdf	Пояснювальна записка до проекту в форматі PDF
Програма	
ProgramBezborodko.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Безбородько.ppt	Презентація проекту