Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: **« Комп'ютерна модель автоматизації процесу поселення студентів у гуртожиток »**

Захищено на засіданні
Атестаційної комісії № 42
протокол № __ від __.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії

_____СКОБ Ю.О.

Виконав:

студент 2 курсу, групи КУ– 61
Галузь знань: 15 – Автоматизація та приладобудування
за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології
ГОЛІКОВ Максим Сергійович

Керівник:

к.т.н., доцент кафедри
теоретичної та прикладної
системотехніки

БУЛАВІН Дмитро Олексійович

Рецензент:

д.т.н., доц., професор кафедри
теоретичної і прикладної інформатики
РУККАС Кирило Маркович_____

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновку, списку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 101 сторінку із яких 50 сторінок основної частини, 1 таблицю, 13 рисунків, 29 найменувань списку використаних джерел на 3 сторінках і 5 додатків на 41 сторінках.

Метою кваліфікаційної роботи є прискорення процесу розподілу місць у гуртожитку через оптимізацію цього процесу методами еволюційного підходу.

Об'єкт дослідження - процес автоматизації поселення студентів за допомогою еволюційного підходу.

Предмет дослідження - моделі та еволюційний підхід для ефективного розв'язання задачі автоматизації поселення студентів.

Питання вивчення сучасних систем поселення, їх переваг і недоліків, разом з аналізом потреб студентів та адміністрації гуртожитку, увійшли до області дослідження цієї роботи. На основі цього аналізу було розроблено та впроваджено комп'ютерну модель, яка була реалізована за допомогою мов програмування Java та TypeScript, і з використанням фреймворків Spring Boot та Angular.

Для перевірки працездатності комп'ютерної моделі були використані тестові дані. Вона гідно пройшла перевірку.

Створена комп'ютерна модель може бути дороблена у повноцінну систему, для використання університетом в цілях модернізації та вдосконалення поточної процедури поселення студентів у гуртожиток.

Ключові слова: комп'ютерна модель, автоматизація, оптимізація, еволюційний підхід, генетичний алгоритм, база даних.

ABSTRACT

The explanatory note to the master's attestation work consists of an introduction, 3 sections, a conclusion, a list of sources used and 4 appendices. The total volume of work is 101 pages, of which 50 pages of the main part, 1 table, 13 figures, 29 names of the list of the used sources on 3 pages and 5 appendices on 41 pages.

The aim of the qualification work is to develop a computer model to optimize and accelerate the process of allocating dormitory places, maximizing the use of resources and taking into account the needs of all participants in this process.

The object of research is the process of automating the settlement of students using an evolutionary approach.

The subject of the study is models and an evolutionary approach to effectively solve the task of automating the settlement of students.

The study of modern settlement systems, their advantages and disadvantages, along with the analysis of the needs of students and the dormitory administration, were included in the scope of this paper. Based on this analysis, a computer model was developed and implemented, which was realized with the help of Java and TypeScript programming languages, and using the Spring Boot and Angular frameworks.

Test data was used to verify the performance of the computer model. It passed the test with flying colors.

The created computer model can be further developed into a full-fledged system to be used by the university to modernize and improve the current procedure for settling students in the dormitory.

Keywords: computer model, automation, optimization, evolutionary approach, genetic algorithm, database.

ЗМІСТ

ВСТУП

Сучасна вища освіта надає значення не лише академічним досягненням, але й житловим умовам, які впливають на загальний досвід навчання та студентське життя. Це ставить перед університетами завдання забезпечення оптимальних умов проживання для своїх студентів. Однією з ключових задач у цьому контексті є ефективне розміщення студентів у житлових приміщеннях.

Щорічно при надходженні студентів першого курсу до ВНЗ дирекція гуртожитків повинна надавати їм кімнати в гуртожитках. Традиційні методи розподілу кімнат у гуртожитку можуть призначати їх випадковим чином або просто на основі ідентифікаційних номерів студентів. Такі методи не враховують спосіб життя та уподобання студентів, що може призвести до того, що деякі студенти зроблять запит на зміну кімнати через відмінності в способі життя та уподобаннях між ними та їхніми сусідами по кімнаті.

Оптимізація розміщення студентів має суттєве значення для забезпечення комфорту, соціальної адаптації та успішного навчання. Це питання має комплексний характер, оскільки потребує врахування різних факторів, таких як розміщення студентів однієї статі в кімнаті, соціальна взаємодія на основі стереотипів, середнє навантаження на курс та інші аспекти студентського життя.

Незважаючи на те, що класичні методи оптимізації вже успішно використовуються у різних сферах, вони можуть виявитися неефективними при розв'язанні цієї конкретної задачі з рядом викликів. Велика кількість змінних та обмежень може перевантажити класичні методи, які можуть бути менш адаптованими до таких складних структур. Врахування не лише кількості студентів в кімнатах, але і інших факторів, таких як соціальні аспекти та спеціальні потреби студентів, ускладнює задачу.

Також треба мати на увазі те, що нелінійність залежностей між різними параметрами може ускладнювати використання класичних методів

оптимізації, які базуються на припущенні лінійності взаємозв'язків. Та варто підкреслити, що динамічність ситуації вимагає гнучкості та швидкості в прийнятті рішень

У цьому контексті еволюційні алгоритми, зокрема генетичні алгоритми, можуть бути потужним інструментом для пошуку оптимальних рішень у складних задачах оптимізації. Генетичні алгоритми базуються на природних принципах еволюції та можуть ефективно вирішувати проблеми оптимізації шляхом пошуку найкращих варіантів розміщення студентів.

Особливу увагу також потрібно приділити зручності та комфорту під час самого процесу поселення. Зважаючи на той факт, що система, яка функціонує багато років, в Україні вже не є актуальною у XXI сторіччі, виникає потреба впроваджувати більш сучасні та практичні рішення. Одним з них є створення веб-додатку з розподіленою архітектурою, що дозволить швидко додавати різні нововведення, та як мінімум зробить поселення студентів комфортним та зручним для усіх учасників цього процесу.

Дана робота присвячена дослідженню застосування еволюційного підходу до проблеми оптимізації поселення студентів. Створено програмну реалізацію на базі веб-додатку, розглянуто методику застосування генетичного алгоритму, результати його використання та зроблено висновки щодо його ефективності для забезпечення оптимального житлового розміщення студентської громади.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЕВОЛЮЦІЙНИЙ ПІДХІД ДЛЯ ЗАДАЧ ОПТИМІЗАЦІЇ

1.1 Аналіз комп’ютерних моделей автоматизації поселення студентів у гуртожиток

Дуже важливим для успішного створення програмного забезпечення є процес аналізу різних моделей для більш чіткого розуміння що саме буде для нас вигіднішим та кращим у тій чи іншій ситуації. І одним із способів є системний аналіз. Це методологія прикладної науки, яка ґрунтується на різноманітних системно організованих, структурно взаємопов'язаних та функціонально взаємодіючих евристичних процедурах, методичних прийомах, математичних методах, алгоритмічних програмах і обчислювальних засобах. Вона спрямована на формування комплексних міждисциплінарних знань та процесів різного характеру, для подальших прийняття рішень щодо розвитку та поведінки системи. Ця методологія враховує безліч конфліктуючих критеріїв та цілей, а також ураховує наявність ризикових факторів, неповноту та недостовірність інформації [1].

Для аналізу комп’ютерних моделей автоматизації поселення студентів у гуртожиток використано показники та критерії ефективності функціонування систем та оцінку систем в умовах визначеності на основі векторної оптимізації методом виділення головного критерію [2, 3].

Для оцінки моделей автоматизації поселення студентів у гуртожиток, тобто систем в умовах визначеності, важливо взяти до уваги, що більшість показників при аналізі суперечливі, тому виникає проблема коректності відповідного критерію. Це призводить до необхідності йти на компроміс і вибирати для кожного показника не оптимальне значення, а близьке до нього, але таке, при якому інші показники матимуть прийнятні значення.

Для вирішення проблеми коректності критерію було розроблено методи векторної оптимізації, і один із них було використано для даної задачі, а саме

метод виділення головного критерію. Його сутність полягає у тому, що ОПР (особа, яка приймає рішення) призначає один основний критерій, інші виводяться до складу обмежень [2, 4].

Так як дуже часто процес поселення студентів у гуртожиток являє собою виконання вимог того чи іншого вищого навчального закладу, прописаних у нормативних документах, і варто зауважити що сам процес є здебільшого однаковим з деяким відхиленням у статут та норми кожної з установ, то можна зробити висновок, що побудова таких автоматизованих моделей буде використовуватися за схожими принципами. Тому висловлюючи думку з приводу аналізу таких моделей, необхідно приймати до уваги один з головних критерій - а саме кросплатформність [3, 4].

Кросплатформність, також відома як багатоплатформність чи мультиплатформність, є властивістю програмного забезпечення працювати не лише на одній, але на кількох програмних (включаючи операційні системи) чи апаратних платформах. Це включає в себе технології, які забезпечують можливість оптимального функціонування на різних середовищах.

Існуюча модель автоматизації (рис. 1.1), яка доступна у відкритому доступі зараз, розроблена на HiAsm - інтегрованому середовище розробки програм, у вигляді інтерфейсного додатку для Windows. Вона відповідає умовам, які необхідні для функціонування програмного забезпечення у повному обсязі, а саме автоматизованого процесу поселення студентів у гуртожиток, але зважаючи на той факт, що це додаток розроблений лише для однієї платформи, опираючись на головний критерій нажаль не можливо розцінювати її як модель яка буде працювати не лише на одному пристрої [3, 5].

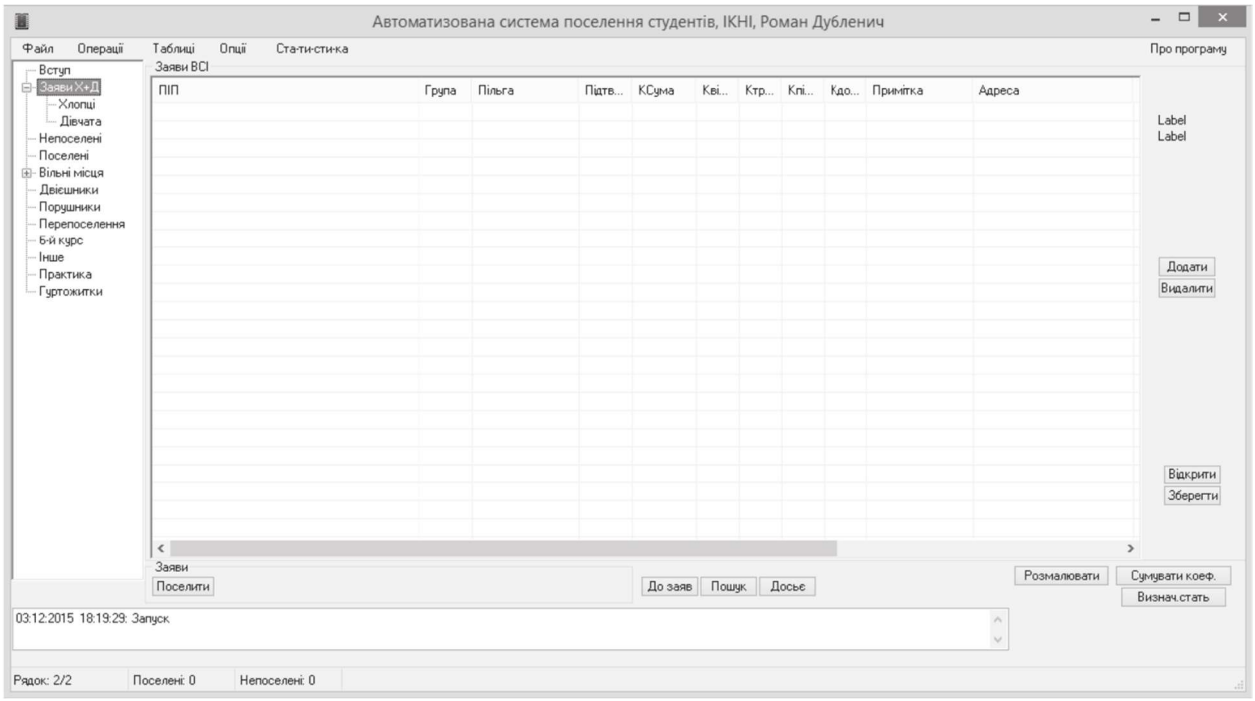

Рисунок 1.1 – Головне вікно існуючої системи «Автоматизована система поселення студентів»

А що саме повинно бути присутнє під час реалізації програмного забезпечення аби воно працювало на різних пристроях та платформах? Однією з технологій та засобів досягнення багатоплатформності є кросплатформність на рівні середовищ виконання, що означає, що виконання програм можливе в різних середовищах завдяки реалізації необхідних можливостей у цих середовищах, що робить програми незалежними від конкретної платформи. Наприклад реалізацію на мові програмування Java [6]. Та було б великою перевагою, якби була модель, яка дозволяла би використовувати її без завантаження на пристрій, а одразу прямо у браузері [3].

Таке програмне рішення у вигляді web-додатку можна реалізувати використовуючи Spring Boot, Angular та PostgreSQL (що буде розглянуто далі). Комп’ютерна модель у такому форматі дозволить використовувати її на різних пристроях та задовольнить головний критерій - кросплатформність. Збереження даних студентів у хмарному сховищі буде більш безпечним ніж зберігання їх на одному комп’ютері [3].

1.2 Еволюційне програмування

Еволюційний підхід до задач оптимізації є аналогією до еволюційних процесів у біосфері і застосовується для вирішення проблем пошуку оптимальних рішень у просторі можливих варіантів. Початковий розвиток цього підходу пов'язаний з теорією «дарвінівського природного відбору» та генетичного розвитку.

Еволюційні алгоритми представляють собою потужний інструмент для вирішення різноманітних складних задач. Цей клас алгоритмів включає:

- генетичні алгоритми;
- еволюційну стратегію;
- еволюційне програмування;
- алгоритми диференціальної еволюції;
- генетичне програмування;

Вони широко застосовуються у різних галузях, включаючи оптимізацію, штучний інтелект, та інженерію [7].

Генетичні алгоритми використовують ідеї еволюції та відбору для ефективного пошуку найкращого вибору у просторі можливих варіантів [8]. Еволюційна стратегія базується на концепції стохастичного пошуку, використовуючи еволюційні операції для покращення кандидатів. Еволюційне програмування включає в себе еволюцію програм або структур, що представляють можливі рішення. Алгоритми диференціальної еволюції опираються на диференціальні операції пошуку оптимальних параметрів [7].

Ці еволюційні підходи дозволяють ефективно вирішувати задачі в різноманітних сферах, враховуючи різноманіття варіантів та невизначеність у вихідних даних. Їх застосування відзначається успішним вирішенням проблем оптимізації та пошуку рішень у складних умовах, що робить їх незамінним інструментом для сучасних наукових та інженерних завдань.

Модель і схема організації їхньої роботи в основному полягає в тому, щоб моделювати природні механізми відбору та рекомбінації генетичної інформації для знаходження оптимальних рішень у просторі пошуку.

Перед появою еволюційних алгоритмів багато задач у сфері оптимізації вирішувалися класичними методами, такими як градієнтний спуск. Однак ці методи часто застрягали в локальних точках мінімізації, та не могли забезпечити оптимальні рішення в задачах зі складною функцією цілі.

Еволюційні алгоритми вирішують ці обмеження, створюючи популяцію кандидатів, яку потім еволюційно перетворюють, дотримуючись цілі пошуку доцільного вирішення проблеми. Цей підхід може бути ефективним, особливо коли функція є неперервною, недиференційованою або має багато локальних мінімумів.

У технологічному контексті еволюційні алгоритми стають ефективним інструментом для оптимізації різноманітних завдань. Наприклад, вони можуть бути використані для:

1) еволюції структури і параметрів нейромереж, щоб отримати оптимальні моделі для конкретних завдань;
2) задач, де потрібно знайти глобальний мінімум або максимум у складних і неоднорідних просторах параметрів;
3) оптимізації параметрів складних моделей, таких як алгоритми машинного навчання, або інші системи з численними налаштуваннями;
4) знаходження найкращих параметрів для функції, яку треба мінімізувати або максимізувати;

Замість традиційних методів, еволюційні алгоритми дозволяють автоматизовано та ефективно знаходити рішення в просторі можливих варіантів, враховуючи при цьому важливі принципи еволюції.

Такий підхід в технологіях відкриває можливості для розв'язання складних завдань, адаптуючись до змінних умов і використовуючи природний відбір як джерело інспірації для оптимізації та вдосконалення штучних систем.

Якщо розглядати в цілому властивості всієї цієї парадигми, однією з ключових - є здатність уникати локальних мінімумів та максимумів, завдяки

імітації принципів еволюції. Це дозволяє їм ефективно обходити простір рішень та знаходити глобально оптимальні конфігурації.

Паралельність є ще однією важливою характеристикою еволюційних алгоритмів, та їх легко виконувати паралельно, оскільки обробка кожного індивіда в популяції може відбуватися незалежно. Це робить їх ефективними на великих обчислювальних кластерах та графічних процесорах, прискорюючи процес пошуку.

Адаптивність – це ключова властивість, яка надає їм можливість змінювати свою стратегію відбору та мутації в процесі пошуку. Це дозволяє їм ефективно пристосовуватися до різноманітних умов та ландшафтів функцій, підвищуючи ймовірність знаходження оптимального рішення.

Здатність працювати з неповними даними та стійкість до шуму робить еволюційні алгоритми особливо корисними в реальних умовах. Вони можуть ефективно оптимізувати задачі навіть при наявності неповних, нелінійних чи невизначених даних, що робить їх універсальними в індустрії та науці.

Неабияка універсальність проявляється у їхній здатності вирішувати різноманітні задачі оптимізації, такі як параметризація моделей, розкладання задач, вибір атрибутів тощо. Ця універсальність робить їх важливим інструментом у великому спектрі доменів та відкриває можливості для розв'язання складних завдань в різних галузях.

Еволюційні алгоритми мають кілька переваг, серед яких:

- гнучкість;

- відсутність потреби в аналітичних похідних;

- у змінних середовищах, де параметри чи умови можуть змінюватися, еволюційні алгоритми можуть адаптуватися до нових умов і продовжувати оптимізацію без повного перетримування;

- реалізація еволюційних алгоритмів може бути відносно простою порівняно з іншими складнішими оптимізаційними методами;

- здатність враховувати багатофункціональність;

- здатність уникнути локальних мінімумів;

- робастність до шуму;

- інтегрованість до різних задач оптимізації;

Також треба зауважити, що існує кілька недоліків у використанні еволюційних алгоритмів, які можуть впливати на їхню ефективність та придатність для конкретних завдань:

- еволюційні алгоритми можуть вимагати значного часу для збіжності до оптимального рішення, особливо в разі складних просторів параметрів або завдань з великою кількістю обмежень;

- результати еволюційних алгоритмів можуть залежати від початкового набору параметрів. Якщо початковий набір обраний несприятливо, це може призвести до знаходження локального мінімуму або максимуму;

- еволюційні алгоритми не завжди гарантують знаходження глобально оптимального рішення, і можуть застрягти в локальних мінімумах або максимумах;

- у великих просторах параметрів еволюційні алгоритми можуть зазнавати труднощів у знаходженні оптимального рішення через великий обсяг простору, який потрібно дослідити;

- на відміну від деяких детермінованих методів оптимізації, еволюційні алгоритми працюють ймовірнісно, і немає гарантії знаходження оптимального рішення за обмежений час.

1.3 Загальне схема еволюційних алгоритмів

Еволюційні алгоритми є важливим напрямком в галузі обчислювального інтелекту, і їх розвиток можна прослідкувати до праці професора Холланда з Університету штату Мічиган, який у 1975 році представив цю інноваційну концепцію у своїй видатній книзі «Адаптація в природних і штучних системах». Відзначається, що ідеї для створення теорії генетичних алгоритмів Холланд отримав під впливом вивчення біологічної літератури, зокрема дарвінівської теорії природного відбору та селекції сільськогосподарських культур.

Генетичні алгоритми базуються на ідеї емуляції природного відбору, який був вперше визначений Чарльзом Дарвіном у його творі «Походження видів» в 1859 році. Основні принципи еволюції, такі як спадковість, мінливість та природний відбір, визначають розвиток живої природи на Землі. Генетичний код організму, закодований у ДНК, визначає його властивості, і ця інформація є ключем до розуміння генетичних алгоритмів [7].

Паралельно з класичною моделлю еволюції Дарвіна, в історії розвитку генетичних алгоритмів виникло значне число інших моделей, таких як модель Ламарка, модель Фріза, модель Гаулда і Елдріджа. Ці моделі, в основі яких лежать різноманітні принципи еволюції, надають можливість розширення та удосконалення класичних генетичних алгоритмів [7].

Вивчення генетичної інформації та спадковості є ключовим аспектом сучасної біології, розкриваючи та пояснюючи механізми, якими передаються характеристики від покоління до покоління. Термін «хромосома» служить ключем до розуміння структур ДНК та розташування генів, які визначають фізичні та біологічні особливості кожної людини.

Кожна клітина організму містить інформацію, закодовану в ДНК, яка організована у вигляді хромосом. Гени, розташовані на хромосомах, визначають властивості, від кольору очей до здатності сприймати смак гіркого. Ці гени можуть знаходитися на конкретних місцях хромосоми, відомих як «локус», і мати різні варіанти, або «алелі».

Генотип визначається сукупністю генів у клітині, і він є основою для формування фенотипу - зовнішніх проявів та характеристик. У процесі розмноження відбувається схрещування, під час якого гени від обох батьків обмінюються та об'єднуються, утворюючи новий генотип для нащадка. Цей процес, також відомий як кросовер або рекомбінація, забезпечує генетичну різноманітність та можливість еволюції видів.

Мутації генів, або зміни в їхній структурі, можуть виникати внаслідок різних факторів. Ці мутації можуть передаватися нащадкам і, в деяких випадках, призводити до нових властивостей чи адаптацій. Якщо такі зміни

корисні, вони можуть стати ключовими для виживання виду та підвищити його пристосованість до змін в середовищі.

Вивчення генетики не лише розширює розуміння спадковості, але й має важливі практичні застосування, такі як генетична терапія та селекція в сільському господарстві. Знання про гени та їх функції відкриває нові можливості для покращення якості життя та здоров'я людей [7].

Загальна схема еволюційних алгоритмів має таки вигляд:

1) Ініціалізація популяції: створення початкової популяції рішень, які відображають можливі варіанти вирішення задачі;

2) Оцінка пристосованості: кожен елемент популяції оцінюється за допомогою функції пристосованості, яка визначає, наскільки добре він вирішує поставлену задачу;

3) Створення нащадків: використовуючи механізми еволюції, створюються нові рішення (нащадки) на основі кращих представників поточної популяції. Це може включати такі операції, як схрещування, мутація та інші види генетичного перетворення. (різні еволюційні алгоритми можуть використовувати різні види еволюційних операторів, таких як оператор мутації, оператор кросовера (схрещування), оператор відбору, а також можуть мати різні параметри, такі як розмір популяції, ймовірність мутації та кросовера тощо);

4) Оцінка пристосованості нащадків: нові елементи популяції оцінюються за допомогою функції пристосованості. (наскільки близьким є задане конструктивне рішення до досягнення поставлених цілей)

5) Відбір: відбираються кращі рішення з поточної популяції та нащадків, які стануть батьками для наступної генерації;

6) Завершення: повторення кроків 3-5 до досягнення заданої точності або кількості ітерацій;

Ці кроки виконуються в циклі, поки не буде досягнуто критерію зупинки. Еволюційні алгоритми можуть бути застосовані до різних задач

оптимізації, таких як пошук глобального мінімуму функції, навчання нейронних мереж, оптимізація параметрів систем управління і тому подібне.

Також її можна зобразити ось такою блок-схемою на рис. 1.2.

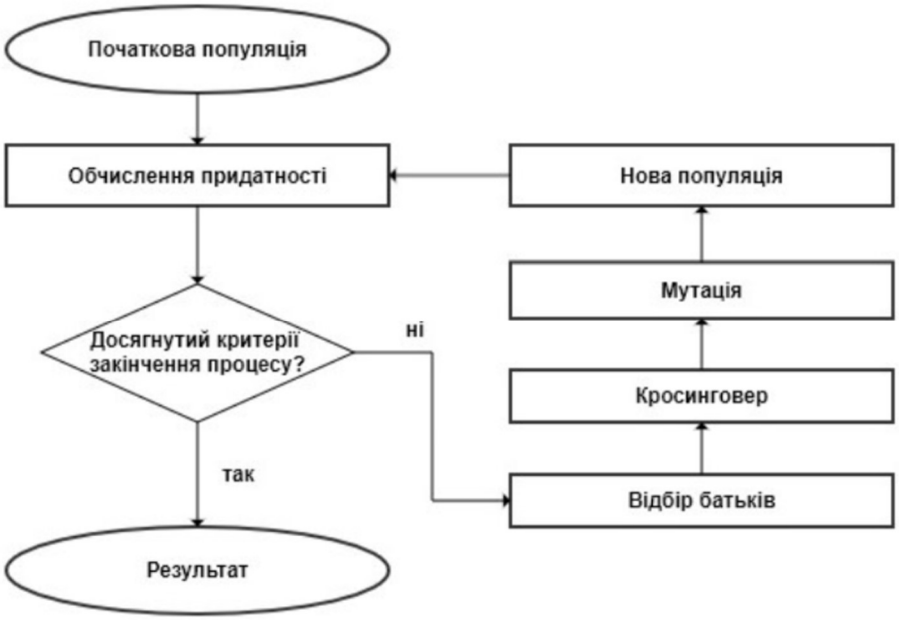

Рисунок 1.2 – Загальна схема еволюційних алгоритмів

1.4 Генетичний алгоритм

Однією з ключових ідей генетичних алгоритмів є використання понять «хромосоми» і «гени». У контексті генетичних алгоритмів - хромосоми представляють можливі рішення, а гени є специфічними параметрами або характеристиками цього рішення. Наприклад, при вирішенні проблеми найкращого маршруту подорожі кожна хромосома може бути представлена набором міст, а гени визначають порядок цих міст.

Також варто звернути увагу на той факт, що генетичний алгоритм взаємодіє з колекцією індивідів, тобто «популяцією», що дає інший підхід ніж інші алгоритми які взаємодіють з одним рішення.

1.4.1 Принцип роботи генетичного алгоритму

Генетичні алгоритми базуються на механізмах природного відбору і реалізують схему «виживання найбільш пристосованих» серед розглянутих

структур, формуючи і змінюючи алгоритм пошуку на основі моделювання еволюції пошуку. У кожному поколінні відбувається процес створення нового набору штучних послідовностей з використанням частини старого набору і додаванням нових частин з «задовільними властивостями» [8].

Алгоритм починається з випадкового набору розв'язків, який називається популяцією. Кожен елемент популяції називається хромосомою і являє собою розв'язок задачі. Хромосоми еволюціонують протягом декількох ітерацій, які називаються поколіннями. В процесі ітерації хромосома оцінюється за допомогою фітнес-функції.

Крім того, в процесі може бути застосована мутація, яка випадковим чином змінює деякі елементи хромосоми. Це сприяє різноманітності та додатковій експлорації простору рішень.

В кінцевому результаті, після кількох ітерацій знаходить оптимальне (або наближене до оптимального) рішення проблеми, яке відповідає найвищій значущості функції придатності серед популяції.

Завдяки функції пристосованості, якщо дивитися на всіх особин популяції, можна виділити:

• максимально пристосовані – це ті, які можуть схрещуватися і в результаті давати потомство;

• мінімально пристосовані, - це ті, які прибираються з популяції і не можуть дати потомства. Такий механізм забезпечує підвищення середньої пристосованості нового покоління порівняно з попереднім.

У класичному генетичному алгоритмі:

• формування початкової популяція відбувається випадковим чином;

• чисельність особин, або розмір популяції, залишається незмінною та піддається змінам протягом всього виконання алгоритму;

• кожна особина генерується як випадковий L-бітний рядок, де L — довжина кодування особини;

• довжина кодування для всіх особин однакова [9].

1.4.2 Алгоритм роботи

На рис. 1.3 зображено схему виконання будь-якого генетичного алгоритму:

Рисунок 1.3 – Блок-схема генетичного алгоритму [10]

Крок алгоритму складається із таких стадій:

1) створення початкової популяції, складеної з деякої кількості хромосом;
2) обчислення для кожної хромосоми її придатності;
3) вибір пари батьківських хромосом за допомогою методів відбору;
4) проведення кросоверу між двома батьківськими хромосомами з ймовірністю, що призводить до отримання двох нащадків;
5) мутація нащадків з ймовірністю;
6) повторення кроків від 3 до 5 до тих пір, поки не буде сформовано нове покоління, яке складається з деякої кількості хромосом;

7) повторення кроків від 2 до 6 до досягнення критерію завершення процесу, і ним може бути визначена кількість поколінь або збіжність функції.

1.4.3 Процес вибору і збереження найкращих рішень для наступного покоління

Існує декілька стратегій відбору: панміксія, інбридинг, аутбридинг та селекція, які відображають ключові аспекти еволюційних процесів і знаходять своє відображення у генетичних алгоритмах.

Панміксія, або універсальне схрещування, представляє собою випадкове схрещування між елементами популяції. У генетичних алгоритмах цей принцип реалізується шляхом комбінування хромосом батьків, забезпечуючи велику різноманітність в наступних поколіннях.

Інбридинг та аутбридинг є аналогіями внутрішньопопуляційного та міжпопуляційного схрещування відповідно. Інбридинг може викликати втрату різноманітності, що може призвести до збільшення ризику утворення недоліків. Аутбридинг, навпаки, сприяє появі нових комбінацій генетичного матеріалу, зберігаючи високий рівень різноманітності.

Селекція в природі призводить до природного відбору, де найбільш пристосовані організми мають більше шансів передати свої гени наступним поколінням. У генетичних алгоритмах селекція вибирає найкращі рішення для подальшого використання в наступних ітераціях.

Турнірний відбір є методом, схожим на природний відбір, де індивіди змагаються за право вижити чи брати участь у схрещуванні. Цей процес може допомогти відібрати найкращі рішення серед популяції.

Метод рулетки в генетичних алгоритмах може відігравати роль аналогії до вибору, що базується на ймовірностях. Індивіди вибираються для схрещування або збереження в залежності від їхнього внеску у покоління.

Використання кожного з них є відзеркаленням у природі та впровадження цих концепцій дозволяє створити ефективні алгоритми оптимізації [11].

1.4.4 Процес винекнення нової комбінації генів

Однією з ключових операцій в генетичних алгоритмах є рекомбінація, яка відіграє важливу роль у створенні нових індивідів шляхом комбінування генетичної інформації від батьків. Рекомбінація може відбуватися в різних формах, і розглянемо деякі з них.

Дискретна рекомбінація визначається конкретним вибором генів від обох батьків. У такому випадку, нащадок отримує цілі гени від одного з батьків, що може призвести до втрати різноманітності. Проте, цей метод дозволяє точно визначити генетичний склад нового індивіда.

Проміжна рекомбінація включає у себе створення нового індивіда шляхом усереднення генетичної інформації від батьків. Цей метод забезпечує більш плавний перехід між значеннями генів і може допомагати у уникненні екстремальних значень.

Лінійна рекомбінація включає в себе процес лінійного комбінування генетичних елементів від батьківських організмів. Цей механізм дозволяє створити нового індивіда, у якого генетична інформація представлена як лінійна комбінація генетичних внесків від обох батьків.

Кросинговер є загальним терміном для рекомбінації в генетичних алгоритмах. Він може включати різноманітні стратегії, такі як:

1) Одноточковий кросинговер: де обмін генетичною інформацією відбувається в одній точці;
2) Двоточковий кросинговер: з обміном в двох точках;
3) Багатоточковий кросинговер: де обмін відбувається в багатьох точках, забезпечуючи більшу різноманітність в генетичній інформації нащадка;

4) Однорідний кросинговер: де кожен ген обирається випадковим чином від одного з батьків;
5) Тріадний кросинговер: що з'єднує три батьківські індивіди для створення нащадка;

Крім того, існують такі варіанти, як перетасувальний кросинговер та кросинговер із зменшенням заміни, які використовують специфічні стратегії для обміну генетичною інформацією.

Перетасувальний кросинговер є методом рекомбінації, який акцентується на обміні генетичною інформацією за допомогою випадкового перетасування генів між батьківськими індивідами. Цей підхід вносить випадковість у процес рекомбінації, забезпечуючи нові комбінації генів, які можуть не відповідати строго структурі батьківських хромосом. Перетасувальний кросинговер дозволяє створювати різноманітні індивіди та допомагає у збереженні генетичної різноманітності в популяції.

Кросинговер із зменшенням заміни є іншою цікавою стратегією. У цьому методі випадково вибираються деякі гени одного з батьків, і їх замінюють генами з іншого батька. Однак кількість замінених генів зменшується з часом, що веде до послідовності, яка стає все більше схожою на одного з батьків. Цей підхід дозволяє зберігати частину генетичної інформації від кращих батьків та, в той же час, дозволяє відбуватися експлорації в генетичному просторі.

Усі ці методи рекомбінації мають свої переваги та недоліки і можуть бути ефективними в різних сценаріях [11-14].

1.4.5 Зміна генетичного коду одного чи декількох індивідів

Ще одним з ключових елементів в генетичному алгоритмі є мутація, яка є необхідною для уникнення локальних екстремумів та передчасної збіжності популяції.

Роль мутації є в тому, що вона вводить випадкові зміни в генетичну інформацію індивідів, що дозволяє алгоритму досліджувати нові області

генетичного простору. Це особливо важливо в ситуаціях, коли популяція може застрягти в локальних екстремумах, і вона служить механізмом, який може допомогти вибратися з таких станів та продовжити пошук оптимального рішення.

Так само існує декілька видів мутації:

1. Мутація для дійсних особин.

Цей вид мутації використовується в задачах оптимізації з дійсними параметрами. Вона полягає в невеликій випадковій зміні значень генів, представлених дійсними числами. Такі зміни дозволяють систематично досліджувати навколишні області генетичного простору.

2. Двійкова мутація.

Використовується у випадках, коли генетична інформація представлена у бінарному вигляді. Тут, зазвичай, випадковим чином обирається певний біт і змінюється його значення. Цей процес допомагає зберегти різноманітність в популяції та уникнути ранньої збіжності.

3. Щільність мутації.

Цей підхід полягає у зміні імовірності мутації з часом. На початку оптимізаційного процесу імовірність мутації може бути високою, але з часом вона зменшується. Це дозволяє алгоритму спочатку досліджувати генетичний простір ширше і зосереджуватися на локальній оптимізації пізніше.

4. Інші види мутації:

1) Додавання випадкового гену: додає новий ген в індивіда, розширюючи генетичний простір;

2) Видалення випадково обраного гену: зменшує розмір генетичного коду, що може бути корисним для пристосування популяції до змінних умов;

3) Обмін місцями: міняє місцями два гени в індивіда, щоб зберегти корисні комбінації [11-14].

1.4.6 Відбір особин до нової популяції

Розглянемо деякі з найпоширеніших методів відбору:

1. Відбір Усіченням.

Відбір усіченням є методом, при якому особини в популяції вибираються на основі їхнього пристосування. Ті особини, які мають вище значення пристосованості, мають більше шансів потрапити до нової популяції. Цей метод сприяє збереженню кращих рішень і спрямований на концентрацію генетичного матеріалу вищої якості.

2. Елітарний Відбір.

Елітарний відбір передбачає пряме включення найкращих особин з поточної популяції в нову без будь-яких змін. Це гарантує, що кращі рішення залишаються в популяції, уникнувши «втрати» цінної генетичної інформації.

3. Відбір Витісненням.

В цьому методі нові особини вибираються для популяції на заміну існуючим. Слабкіші особини можуть бути викинуті, що сприяє збільшенню концентрації добре пристосованих особин в популяції.

4. Метод Больцмана (Метод Відпалу).

Метод Больцмана використовує ймовірнісний підхід для вибору особин. Ймовірність вибору особини залежить від її пристосованості. Цей метод дозволяє більшій кількості особин мати шанс потрапити в нову популяцію, навіть якщо вони мають менше значення пристосованості [11-14].

1.4.7 Види генетичних алгоритмів

1. Канонічний Генетичний Алгоритм.

Канонічний генетичний алгоритм (ГА) - це базовий та загальний варіант генетичного алгоритму, який включає в себе основні етапи еволюційного оптимізаційного процесу. Основні кроки включають в себе випадковий вибір початкової популяції, оцінку пристосованості кожної особини, використання операторів кросинговера та мутації для створення нащадків, а також відбір особин для формування нової популяції.

3. Метод Уривчастої Рівноваги.

Метод уривчастої рівноваги вказує на стратегію вибору параметрів алгоритму, яка постійно змінюється в ході еволюції. Він включає в себе зміни ймовірностей кросинговера та мутації, а також інші параметри, залежно від етапу еволюційного процесу.

4. Гібридний Алгоритм.

Гібридний генетичний алгоритм поєднує принципи генетичних алгоритмів з іншими методами оптимізації або метаевристиками. Для прикладу візьмемо використання злиття генетичних алгоритмів з методами оптимізації на основі виборки чи алгоритмами імітації виживання найкращих.

5. CHC - Cross-population Selection, Heterogeneous Recombination, and Cataclysmic Mutation (міжпопуляційний відбір, гетерогенна рекомбінація та катаклізмична мутація).

Цей метод вказує на підхід до оптимізації, який використовує різні методи відбору, рекомбінації та мутації для покращення роботи генетичного алгоритму.

По-перше, «Cross-population selection» визначає стратегію відбору, де особини обираються з різних популяцій. Це сприяє збільшенню різноманітності та обміну корисною інформацією між підгрупами. Такий підхід покращує здатність алгоритму адаптуватися до різноманітних умов та ефективно досліджувати простір можливих рішень.

«Heterogeneous recombination» - означає використання різних методів рекомбінації. Замість стандартної однорідної рекомбінації, цей підхід застосовує комбінацію різних методів, що дозволяє краще враховувати специфіку завдань. Наприклад, можливість використання частин генетичного коду з одного батьківського елемента та іншої частини з іншого. Нарешті, «Cataclysmic mutation» підкреслює використання інтенсивної мутації для створення радикальних змін у популяції. Це може бути корисним в ситуаціях, де потрібно стрімко змінити характеристики рішення для швидкого пристосування до нових умов або для виходу з локальних оптимумів.

6. Генетичний Алгоритм з Нефіксованим Розміром Популяції.

Генетичний алгоритм з нефіксованим розміром популяції є інноваційним підходом, де розмір популяції може динамічно змінюватися протягом еволюційного процесу. Це відкриває нові перспективи для адаптації генетичних алгоритмів до специфіки оптимізаційних завдань та сприяє більш ефективному пошуку оптимальних рішень.

Однією з ключових особливостей цього підходу є гнучкість щодо розміру популяції. Традиційні генетичні алгоритми використовують фіксований розмір популяції протягом усього еволюційного процесу. У іншому випадку, генетичний алгоритм з нефіксованим розміром популяції може динамічно збільшувати або зменшувати кількість особин у популяції залежно від обставин.

Такий підхід особливо корисний у випадках, коли оптимізаційна задача демонструє змінювану складність або динамічність у часі. Наприклад, при швидко змінних умовах або в разі, коли необхідно пристосувати розмір популяції для ефективного вирішення конкретного етапу оптимізації. Цей підхід також дозволяє економити обчислювальні ресурси. Наприклад, у періоди зменшення складності задачі можна зменшити розмір популяції, щоб уникнути надмірного обчислювального навантаження. З іншого боку, у більш складних фазах еволюції розмір популяції можна збільшити для більш широкого обходження простору рішень [11-14].

Висновки за розділом 1

У ході аналізу предметної області було проаналізовано систему для автоматизованого поселення студентів у гуртожиток, яка була у відкритому доступі в інтернеті, та зроблено висновок щодо недоліків такої системи.

Варто зазначити, що було обрано сильне та інноваційне рішення, якого у відкритому доступі не знайшлося – використати еволюційний підхід для поставленої задачі.

Визначено, що генетичний алгоритм є ефективним методом оптимізації, який може бути ідеально використаний для задачі поселення студентів у гуртожиток через його здатність швидко знаходити оптимальні рішення в складних просторах можливих варіантів.

- Подання індивідів: у генетичному алгоритмі кожен студент може бути представлений як «індивід» з хромосомою, де кожен ген кодує певну характеристику студента;
- Функція пристосованості (fitness function): оцінює, наскільки оптимально певний розподіл студентів у гуртожитку задовольняє вимоги та критерії;
- Схрещування і мутація: генетичний алгоритм використовує операції схрещування (комбінування характеристик батьківських особин) та мутації (випадкові зміни в хромосомах) для створення нових індивідів, які представляють потенційні розподіли студентів у гуртожитку;
- Еволюційний процес: генетичний алгоритм працює у формі ітераційного процесу, де кожна наступна генерація індивідів стає все кращою за попередню, оскільки кращі рішення мають більші шанси на просування до наступної генерації.

Завданням генетичного алгоритму є знаходження такого розподілу студентів у гуртожитку, який максимізує задоволення вимог та критеріїв, що вказані у функції придатності.

Генетичні алгоритми ефективно працюють зі складними задачами оптимізації, тому вони можуть бути важливим інструментом для оптимізації розподілу студентів у гуртожитку відповідно до конкретних обмежень та умов.

РОЗДІЛ 2
РОЗРОБКА КОМП'ЮТЕРНОЇ МОДЕЛІ З РОЗПОДІЛЕНОЮ АРХІТЕКТУРОЮ

2.1 Аналіз потреб та вимог до комп'ютерної моделі

Аналіз вимог, відомий також як інженерія вимог - це процес встановлення очікувань щодо нового чи вдосконаленого продукту з боку користувачів. Це завдання програмної інженерії, яке заповнює прогалину між системною інженерією та проектуванням системи. Слід відзначити, що саме аналіз вимог дозволяє інженеру-програмісту визначити призначення програмного забезпечення та побудувати моделі даних, функціональних та поведінкових аспектів, які буде обробляти програмне забезпечення. Це дає розробнику програмного забезпечення уявлення про інформацію, функції та поведінку системи.

Цей етап важливий у процесі розробки програмного забезпечення, оскільки він охоплює завдання, спрямовані на визначення потреб або умов, яким повинен відповідати новий або змінений продукт, враховуючи можливо суперечливі вимоги зацікавлених сторін, таких як бенефіціари або користувачі.

Вимоги повинні бути реалізованими, вимірюваними, тестовими, пов'язаними з визначеними бізнес-потребами чи можливостями. Вони можуть бути як функціональними, так і нефункціональними.

Функціональні вимоги, також відомі як вимоги функціональності, встановлюють функціональність програмного забезпечення, яку розробники мають забезпечити, щоб користувачі могли виконувати свої завдання відповідно до бізнес-вимог. Іноді їх називають вимогами поведінки, оскільки вони містять вимоги з традиційним «повинна». Наприклад, «Програма повинна відправляти користувачу вітання під час успішного входження на свою сторінку». Якщо зібрати це до купи, то функціональні вимоги конкретизують, що саме розробнику потрібно реалізувати [15].

2.1.1 Вивчення потреб користувачів (студентів, робітників)

Технології та інновації завжди визначають ритм нашого життя, вивчення потреб користувачів стає важливим елементом нормального функціонування будь-якої організації, приділяючи особливу увагу конкретним аспектам, а саме потребам студентів і керівництва гуртожитків, було приділено пріоритетну увагу розумінню і задоволенню їх очікувань, щоб створити сприятливе і ефективне середовище для навчання і життя. Це допомагає визначити задоволеність і комфорт учасників освітнього процесу та керуючого персоналу з метою підвищення їх досвіду і поліпшення умов роботи і навчання.

Вивчаючи потреби студентів, потрібно звернути увагу на конкретну складову – час, який студент витрачає на увесь процес поселення, починаючи з отримання усіх необхідних документів, і закінчуючи останнім етапом, а саме отримання документу, що посвідчує приналежність студента до якоїсь конкретної кімнати у конкретному гуртожитку. Особливо відстояти черги, подекуди навіть із 30-40 осіб, що також займає багато часу.

Так само і для робітників університету – вони витрачають купу часу для своїх поставлених задач, за які кожен з них відповідає, наприклад аналіз вільних кімнат та гендерної кон'юнктури для адміністрації гуртожитку, підписання багатьох документів з боку деканату, видача паперових бланків для заповнення з боку паспортиста та багато чого іншого. Також зважаючи на інші задачі, які можуть бути поставлені у кожного з робітників університету, одразу виникає аргументування потреби у зменшенні навантаження та делегуванні задач за допомогою комп’ютеризованого інструменту.

Потрібно підкреслити, що перераховані не усі аспекти та проблеми з потребами, адже для кожного учасника процесу потрібен індивідуальний підхід, і зазначені проблеми, такі як час та завантаженість, є основними.

У результаті ознайомлення з цими потребами користувачів були отримані цікаві висновки про чинники, які визначають якість їхнього життя та професійну діяльність. Це вказує на важливість створення комфортних та

інноваційних умов для студентів та робітників, що сприяє їхньому успішному навчанню та розвитку. Також визначено ключові аспекти, які впливають на ефективність роботи кожного з учасників процесу, включаючи покращення обслуговування та інфраструктури.

Враховуючи вищезазначене, важливо активно залучати відгуки та пропозиції від користувачів у подальших дослідженнях та покращеннях. Це сприятиме адаптації середовища до швидкозмінних потреб та очікувань, створюючи гармонійний та продуктивний простір для всіх учасників освітнього процесу. Виходячи з цього, можна впевнено стверджувати, що подальші заходи та зусилля щодо покращення умов проживання та навчання сприятимуть покращенню якості життя та ефективності праці всіх людей, які залучені до цього середовища [16].

2.1.2 Визначення функціональних вимог

Функціональні вимоги для комп'ютерної моделі автоматизації процесу поселення студентів у гуртожиток є конкретними, і було виділено наступні, так як вони є найважливішими саме у контексті даної задачі:

1) Авторизація та аутентифікація: користувач повинен ввести логін та пароль для того, щоб ввійти у систему, що й буде відрізняти яка його роль у системі (студент, працівник університету, адміністратор);
2) Реєстрація користувачів у системі: користувач може зареєструватися як студент на спеціальній формі реєстрації, але зареєструвати його під іншою роллю повинен тільки адміністратор;
3) Управління даними користувачів: адміністратор повинен мати доступ до операцій редагування, додавання, видалення та перегляду інформації про студентів та робітників;
4) Алгоритм поселення студентів: після успішної реєстрації студентів та виконання усіх вимог для поселення, використати генетичний алгоритм на основі конкретних даних про всіх студентів;

5) Перегляд інформації про поселення: після успішного виконання генетичного алгоритму, студенти повинні мати вільний доступ до перегляду куди та з ким їх поселила програма;
6) Безпека даних: користувачі не повинні мати доступ до ресурсів інших користувачів, але водночас адміністратор повинен мати доступ до ресурсів кожного з користувачів;

Дивлячись на перераховані вимоги, можна побачити, що вони не описують як ці функції будуть виконуватися, і це «поглиблення» у функціонування цих вимог буде розглянуто далі. Також варто зазначити, що ці вимоги не є кінцевими [16, 17].

2.2 Проектування архітектури комп’ютерної моделі

Як вже зазначалося раніше, під час аналізу комп’ютерних моделей було виявлено додатки, які орієнтовані на конкретну операційну систему. Тому було вирішено розробляти веб-застосунок з розподіленою архітектурою типу клієнт-сервер.

Розподілена архітектура — це концепція розробки програмного забезпечення, у якій системні компоненти розташовані на різних фізичних машинах або в різних логічних просторах і зазвичай з’єднані через мережу. Ця архітектура розроблена для покращення масштабованості, доступності та ефективності системи.

Трирівнева архітектура (англ. three-tier або multitier architecture) - це підтип розподіленої архітектури, який використовується для розробки програмних систем. Ця архітектура розділяє програмну систему на три рівні або компоненти, кожен з яких відповідає конкретним функціональним обов`язкам. Трирівнева архітектура включає наступні рівні:

1) Представлення (інтерфейсний рівень, Presentation Layer): на цьому рівні знаходиться інтерфейс користувача (User Interface), який відповідає за відображення інформації та обробку введення користувача. У його обов'язки входить передача команд, які

необхідно обробити на прикладному рівні, отримання відповідей від прикладного рівня для читання та представлення інформації користувачеві.

2) Бізнес-логіка (логічний рівень, Business Logic Layer): другий рівень, який відповідає за обробку бізнес-логіки та бізнес-процесів програмної системи. Цей рівень діє як мозок системи та створює контрольований і безпечний спосіб зв'язку між рівнем презентації та рівнем даних. Коли користувач виконує дію, наприклад надсилає форму у веб-додатку, прикладний рівень аналізує інформацію та виконує операції для взаємодії з рівнем даних.
3) Доступ до даних (рівень доступу до даних, Data Access Layer): на цьому рівні реалізована взаємодія з базою даних або іншими джерелами даних. Забезпечує взаємодію із збереженням та отриманням даних для подальшої обробки на рівні бізнес-логіки.

Схематично таку архітектуру можна зобразити на рис. 2.1 [18].

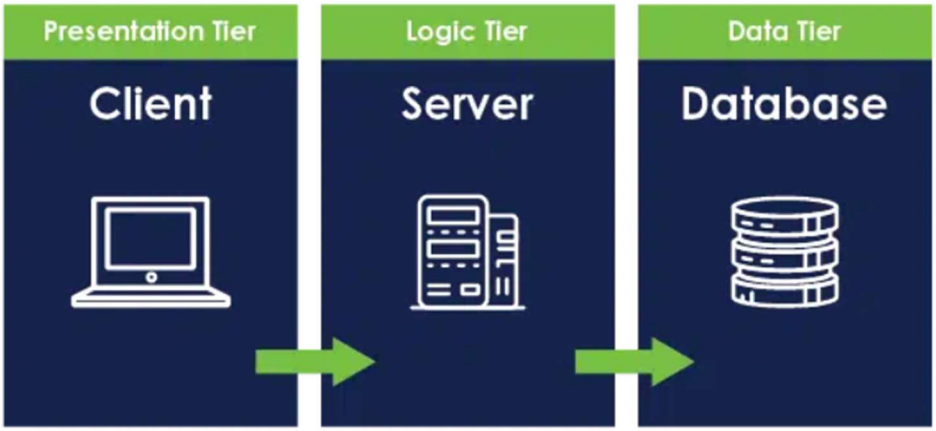

Рисунок 2.1 – Модель трирівневої архітектури

Основні переваги трирівневої архітектури включають:

1) Модульність: кожен рівень може бути розроблений та змінений незалежно від інших.

2) Підтримка різних платформ: інтерфейс може бути розроблений для різних платформ, оскільки бізнес-логіка та доступ до даних залишаються незалежними.
3) Віддалена підтримка викликів: кожен рівень може бути розташований на різних комп'ютерах або серверах, що забезпечує віддалену підтримку дзвінків.

Важливо відзначити, що трирівнева архітектура може бути розгорнута в розподіленому середовищі, де кожен рівень може бути розташований на різних серверах або навіть у різних мережах.

Фрагмент структури проекту зображений на рис. 2.2.

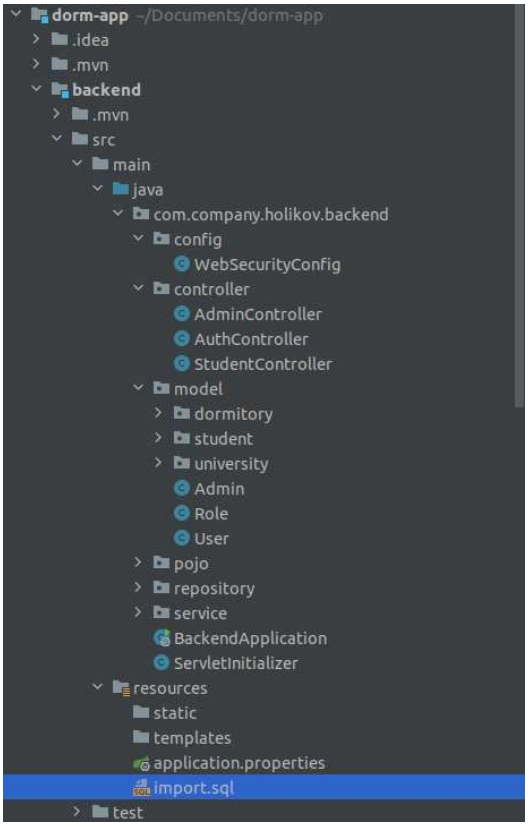

Рисунок 2.2 – Фрагмент структури проекту

Блок з реалізацією еволюційного підходу для задачі поселення студентів розташований на рівні бізнес логіки. Він взаємодіє з базою даних і на вхід йому подаються дані студентів, а саме курс, соціотип та гендер. Після відпрацювання алгоритму, на виході висвітлені результати роботи алгоритму,

а саме кінцевий варіант поселення студенті по кімнатах. Вони будуть подаватися в спрощеному форматі та відображатимуться у консолі. Детальна реалізація буде розглянута у розділі 3.

Також слід відзначити, що такий підхід дозволить у майбутньому доробити модель у повноцінну систему, завдяки вищезгаданим характеристикам. Наприклад, передавати інформацію про поселення студентів, після виконання блоку генетичного алгоритму, на рівень представлення, що дозволить якісно та коректно надати користувачам розуміння у якій кімнаті та з ким вони будуть жити.

Тому для розробки комп'ютерної моделі автоматизації процесу поселення студентів у гуртожитку було обрано саме трирівневу архітектуру.

2.2.1 Обґрунтування вибору компонентів для розробки веб-додатку

Під час розробки комп'ютерної моделі згідно поставленої задачі, важливо обрати правильні компоненти та інструменти для забезпечення ефективності, надійності та безпеки додатку. На цьому етапі було розглянуто основні компоненти для розробки комп'ютерної моделі процесу поселення студентів у гуртожиток: Spring Boot [19, 20], база даних PostgreSQL [21, 24], Maven [22], ORM (Java Persistence API з Hibernate) [20], Spring MVC [19, 20], Spring Security [19, 20] та Angular [23], та обґрунтування їх вибору.

1) Spring Boot є ідеальним вибором для розробки програм завдяки своїй простоті та швидкості налаштування. Він забезпечує інтегрований веб-сервер Tomcat, який полегшує розгортання програми. Крім того, Spring Boot також надає багато корисних бібліотек і функцій для розробки, щоб заощадити час і зусилля.

2) PostgreSQL є найкращим вибором для бази даних цього проекту з кількох причин:

- Надійність: PostgreSQL відомий своєю стабільністю та надійністю, що забезпечить безпеку даних студентів та гуртожитків.

- Розширюваність і гнучкість: PostgreSQL дозволяє розширювати та налаштовувати його функціональність, додаючи власні типи даних, функції та розширення. Це дозволяє створювати власні рішення для ваших унікальних потреб (розглянуто далі в документі).

- Сумісність з Java і Hibernate: PostgreSQL має хорошу сумісність з Java і пов'язаними рішеннями ORM, такими як Hibernate, полегшуючи взаємодію з базою даних із веб-додатку.

3) ORM насправді є концепцією, де об’єкт Java може бути представлений як дані в базі даних (і навпаки). Реалізація - Java Persistence API (JPA) із Hibernate, що дозволяє використовувати Object Relational Mapping (ORM) для легкої взаємодії з базою даних. Hibernate надає зручний спосіб виконувати операції з базою даних без написання складних запитів SQL.

4) Hibernate - це основа для роботи з базами даних у програмах Java. Він забезпечує зручний спосіб взаємодії з базами даних за допомогою об’єктно-реляційного відображення (ORM). Основна мета Hibernate - забезпечити ефективний, простий і гнучкий доступ до конкретних даних у базі даних.

5) Spring MVC (Model-View-Controller) є частиною широко використовуваної Spring Framework, яка дозволяє легко та ефективно розробляти веб-програми на Java. Spring MVC використовує архітектурний шаблон Model-View-Controller для організації та керування веб-запитами.

6) Використання Spring Security є розумним вибором для захисту та автентифікації користувачів у програмі. Це забезпечує безпеку особистої та конфіденційної інформації.

7) Maven є найпопулярнішим інструментом для складання проекту та керування залежностями в середовищі Java. Це спростить процес додавання бібліотек і забезпечить уніфікований процес збірки.

8) Angular — потужний фреймворк для розробки інтерфейсу користувача, який дозволяє створювати динамічні та інтерактивні веб-сайти.

Spring Boot є найкращим вибором серед інших фреймворків для розробки веб-додатків на Java. Він пропонує багато переваг, які спрощують написання коду та дозволяють зосередитися на логіці додатків. Однією з ключових переваг Spring Boot є концепція «конвенції над конфігурацією». Це означає, що багато параметрів за замовчуванням уже вбудовані, і не потрібно витрачати час на визначення багатьох параметрів конфігурації. Отже, можна фокусуватися на розробці коду, який вирішує конкретні завдання.

Іншою важливою особливістю цього фреймворку є його вбудований контейнер для обробки HTTP-запитів. Це дозволяє розробникам запускати свої програми як окремі виконувані файли, що значно полегшує розгортання та спрощує процес розгортання.

Крім того, Maven - інструмент для управління проектами та залежностями - дозволяє легко додавати бібліотеки та інші модулі до проекту, і він має ідеальну гармонію разом із Spring Boot. Це робить розробку більш гнучкою та дозволяє швидко впроваджувати нові функції та оновлення без зайвих зусиль.

Нарешті, велика спільнота розробників Spring Boot і обширна документація роблять його привабливим вибором для розробки проектів. Це дозволяє швидше вирішувати проблеми та обмінюватися досвідом між розробниками.

Переходячи до PostgreSQL – об’єктно-реляційної системи управлінням баз даних - потрібно зазначити, що це є найкращим вибором серед інших реляційних баз даних, завдяки своїм високим стандартам безпеки, надійності та розширюваності. Однією з ключових особливостей PostgreSQL є його велика активна спільнота, яка постійно вносить внески у розвиток та удосконалення системи.

Ще однією перевагою є його розширені можливості для обробки географічної інформації та роботи з геоданими. Завдяки вбудованим розширенням, таким як PostGIS, ця система управлінням може ефективно

взаємодіяти з геопросторовими даними, що робить його популярним в області геоінформаційних систем та аналізу геоданих.

Треба підкреслити, що PostgreSQL має відкритий код. Це означає, що присутня можливість переглядати, змінювати та розповсюджувати програмне забезпечення без обмежень. Відкритість коду сприяє створенню спільноти розробників, яка спільно працює над вдосконаленням системи, а також забезпечує повний контроль над вашою базою даних. Крім того, він підтримує ряд розширень та додаткових модулів, що дозволяє вам адаптувати базу даних під конкретні потреби вашого проекту. Це може включати в себе різноманітні функції для оптимізації продуктивності, роботи з різними типами даних чи підтримки специфічних протоколів зв'язку.

Також варто зазначити, що завдяки тому, що PostgreSQL дозволяє використовувати об'єктно-реляційний підхід, що сприяє роботі з об'єктами та класами, що є типовим для об'єктно-орієнтованих мов програмування, він ідеально підходить для використання разом із Java.

Angular — це потужний фреймворк, який використовується для створення сучасних веб-додатків. Його архітектура базується на компонентах, які спрощують організацію коду та створення багаторазових елементів інтерфейсу.

Однією з головних переваг Angular є двостороння прив’язка даних. Це означає, що зміни в моделі автоматично відображаються в інтерфейсі користувача і навпаки. Це полегшує маніпулювання даними та зменшує кількість коду, який потрібно написати для синхронізації даних.

Angular також надає потужний механізм для обробки HTTP-запитів, що робить його ідеальним вибором для взаємодії на стороні сервера. Інтегрована система модулів допомагає організувати код і надає можливість додавати нові функції та розвивати програму.

В цілому, цей вибір компонентів обумовлений кількома факторами, такими як їх продуктивність, ефективність, масштабованість, та найголовніше – це простота використання.

2.2.2 Конфігурація та налаштування проекту

Щоб успішно поєднати три технології, а саме Spring Boot, Angular та PostgreSQL, важливо визначити зручні способи взаємодії між ними. Це включає в себе визначення REST, конфігурації та налаштування самого проекту.

Для того аби створити взаємодію між Spring Boot та PostgreSQL, треба за допомогою інструмента для управління проектами Maven у спеціальному файлі додати залежності, які автоматично завантажаться у проект з віддаленого репозиторію. Вони включають у себе фреймворк для роботи з реляційними базами даних у середовищі мови програмування Java, а саме Hibernate, та власне об’єктно-реляційну базу даних Postgres.

Після цього потрібно коректно задати конфігурацію у файлі application.properties, у якому необхідно створити підключення вказавши URL-адресу, логін та пароль до бази даних. Варто зазначити, що у цьому файлі потрібно налаштувати спосіб генерації схеми, діалект та інші параметри необхідні для правильної роботи. Це зображено на рис. 2.3.

```
spring.jpa.database=POSTGRESQL
spring.datasource.url=jdbc:postgresql://localhost:5432/settling
spring.datasource.username=postgres
spring.datasource.password=123
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.show-sql=true
#spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.sql.init.data-locations= classpath:/import.sql
spring.sql.init.mode=always

#spring.jpa.defer-datasource-initialization=true
#spring.sql.init.mode=always
```

Рисунок 2.3 – Конфігурація Spring Boot

Для генерації запитів з серверу до бази даних потрібно створити клас, вказавши анотацію @Repository, який буде розширювати інший клас JpaRepository<T, E>. Після цього, за допомогою ін’єкції цього компоненту

викликати необхідні методи. Фрагменті цього функціоналу зображена на рис. 2.4.

```
@Service
public class UserServiceImpl implements UserService {
    7 usages
    private final UserRepository userRepository;
    @Autowired
    public UserServiceImpl(UserRepository userRepository) { this.userRepository = userRepository; }
    no usages
    @Override
    public List<User> findAll() { return userRepository.findAll(); }
    no usages
    @Override
    public User findByLogin(String login) { return userRepository.findByLogin(login); }
    no usages
    @Override
    public User findByEmail(String email) { return userRepository.findByEmail(email); }
```

Рисунок 2.4 – Фрагмент запитів до бази даних

Як вже вказано раніше, Spring Boot та Angular завдяки своїм перевагам, були обрані для реалізації трирівневої архітектури, і аби налаштувати їх взаємодіють між собою, за допомогою REST (Representational State Transfer - передача репрезентативного стану) - це архітектурний стиль, який використовує HTTP (Hyper Text Transfer Protocol - протокол передачі гіпертекстових документів) для передачі та взаємодії з ресурсами, та CORS (Cross-origin resource sharing - Перехресне використання ресурсів) - це технологія браузерів, яка дає доступ веб-додаткам до ресурсів з будь-якого іншого домену. Наприклад, з домену «http://localhost:4200/get-data» можна зробити запит на отримання даних з домену «http://localhost:8080/data». Для цього, з боку сервера слід створити клас помітивши його анотацією @RestController, а з боку клієнта просто робити запити з екземпляра HttpClient за допомогою методів get, post, push тощо.

Якщо неправильно налаштувати конфігурацію CORS, то з будь-яких інших доменів зі сторони клієнта можна буде отримати інформацію з домену сервера, що потягне за собою порушення безпеки, наприклад витік даних. В наслідок цього, ця технологія у контексті спільного використання ресурсів є важливою для безпеки та захисту даних.

2.2.3 Визначення основних етапів автоматизованого процесу поселення

Під час визначення основних етапів автоматизованого процесу поселення студентів у гуртожиток, треба визначити той факт, що створюючи комп'ютерну модель, було обрано загальну структуру та майбутній напрям функціонування моделі.

Основні етапи є наступними:

1) реєстрація студента;
2) авторизація усіх задіяних у цьому процесі користувачів;
3) студент надсилає запити робітникам університету на підтвердження виконання тих чи інших вимог, в залежності від спеціалізації робітника;
4) робітники приймають рішення щодо підтвердження виконання конкретних вимог студентом;
5) якщо студентом усі вимоги виконані, то він допускається до поселення у гуртожиток
6) запуск генетичного алгоритму для поселення студентів
7) додаткова перевірка на гендер (за потреби)

Тепер більш детально розглянемо етапи 3 та 4. Під робітниками мається на увазі користувачі з ролями:

- медсестра;
- економіст;
- заступник декана;

Кожен з цих робітників відповідає лише за свої конкретні задачі: медсестра за підтвердження проходження студентом медогляду, економіст за оплату місця у гуртожитку, а заступник декана за перевірку, що користувач є студентом.

Цей процес можна описати на концептуальному рівні зобразивши на діаграмі прецедентів на рис. 2.5 [25, 26].

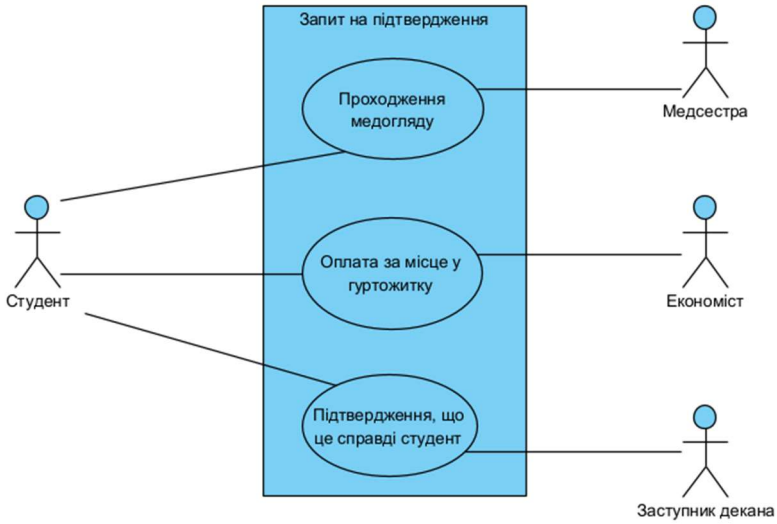

Рисунок 2.5 – Діаграма прецедентів взаємодії студенту та робітників

2.3 Розробка бази даних

Створення бази даних є важливою стадією в галузі інформаційних технологій, тому що вона дозволяє ефективно управляти, організовувати і зберігати значний обсяг даних. Вона представляє собою систематизовану колекцію інформації, яка легко доступна та піддається обробці. ЇЇ значущість полягає у можливості забезпечити консистентність, цілісність та доступність даних для користувачів та систем.

Основні принципи розробки бази даних включають створення належної структури, визначення взаємозв`язків між різними таблицями, нормалізацію даних для уникнення дублювання і забезпечення оптимального використання простору для зберігання. Крім того, коректне визначення прав доступу до даних є ключовим для гарантування безпеки та запобігання несанкціонованому доступу за допомогою SQL-ін'єкцій. Успішна розробка також передбачає адаптацію до потреб користувачів і зменшення зусиль для вирішення тієї чи іншої проблеми під час подальшої розробки комп’ютерної моделі.

Як вже зазначалося вище, PostgreSQL був обраний інструментом для зберігання даних про студентів, та потрібно зазначити, що він є найкращим вибором серед інших реляційних баз даних.

2.3.1 Визначення структури та розробка схеми БД

Структура бази даних - це організований спосіб зберігання та організації даних в базі даних. Вона визначає, як саме дані будуть представлені, зберігатися і взаємодіяти одне з одним. Основні елементи структури бази даних включають таблиці, поля та відносини між таблицями.

Варто одразу визначити головні сутності з якими будуть відбуватися основні операції:

- student (таблиця з даними про студентів);
- users (таблиця для збереження користувачів);
- role (таблиця для визначення ролі користувача на сайті);
- employee (таблиця з даними про робітників університету);
- administrator (таблиця з даними адміністраторів);
- dormitory (таблиця для збереження даних про гуртожиток);

Інші таблиці є допоміжними, щоб краще структурувати та впорядкувати дані. Особливо для таблиці student, де такі атрибути, як спеціальність та факультет (speciality та department відповідно у цій таблиці), є первинними ключами для таблиць які містять потрібні дані.

Повну схему бази даних можна побачити на рис. Д.1.

У купі знадобилося 13 таблиць.

З огляду завантаженості даними у цій схемі, треба зробити висновок, що для розробки комп’ютерної моделі згідно поставленої задачі, цих таблиць вистачить у повному обсязі, але для розробки повноцінної системи, треба буде додати таблиці та налаштувати зв’язки між ними.

Також наведено частину коду для створення цих таблиць на рис. 2.6.

```
create table if not exists role
(
    id   integer generated always as identity primary key,
    name varchar(50) not null
);

create table if not exists users
(
    id       integer generated always as identity primary key,
    login    varchar(50) unique not null,
    password varchar(100)       not null,
    email    varchar(50) unique not null,
    role integer,

    constraint users_id_fk foreign key (role) references role (id)
);
```

Рисунок 2.6 – Створення таблиць role, users

Беручи за основу концептуальну модельну декларацію, можна побудувати діаграму «Enhanced Entity-Relationship» - розширена сутнісно-зв'язкова модель. Це розширення стандартної моделі сутнісно-зв'язкових (Entity-Relationship) діаграм, яке додає додаткові концепції та можливості для більш точного та повного опису взаємозв'язків між об'єктами в базі даних [26].

Фрагмент цих зв’язків поданий у канонічній нотації EER-моделі на рис. 2.7 [27].

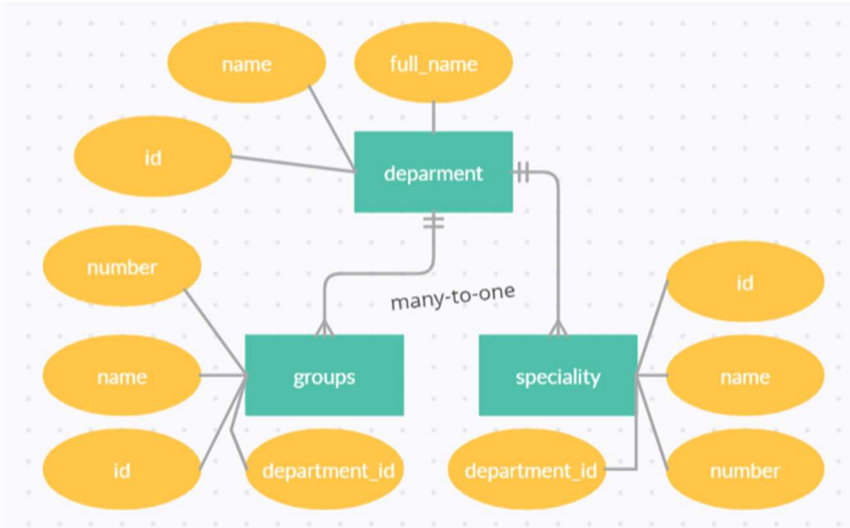

Рисунок 2.7 – Фрагмент канонічній нотації EER-моделі для таблиць department, groups та speciality

На малюнку можна побачити, що для цих таблиць використовується тип асоціації Many-to-one (багато-до-одного) - це відношення в базі даних, де багато записів в одній таблиці відповідають одному запису в іншій таблиці. З цього отримуємо, що один факультет має декілька спеціальностей та груп, і водночас багато груп та спеціальностей можуть бути на одному факультеті.

Висновки за розділом 2

У даному розділі проведено аналіз потреб та вимог до комп'ютерної моделі з розподіленою архітектурою для автоматизації процесу поселення студентів у гуртожиток. Вивчення потреб користувачів, зокрема студентів та робітників, дало можливість зрозуміти основні вимоги до її функціональності.

Визначено функціональні вимоги, які повинні бути задоволені веб-додатком. Це включає в себе не лише індивідуальні потреби користувачів, але й ефективність та надійність комп’ютерної моделі в цілому. З урахуванням цих вимог визначено основні етапи автоматизованого процесу поселення, які будуть реалізовані у веб-додатку.

У розділі також розглянуто проектування архітектури комп'ютерної моделі, де обґрунтовано вибір компонентів для розробки веб-додатку. Це включає вибір технологій, мов програмування та інших ключових елементів для розробки. Також було розглянуто налаштування проекту і його конфігурацію.

Особлива увага була приділена розробці бази даних, де визначено структуру та розроблено схему, яка відповідає потребам даної моделі. Це є важливим кроком у забезпеченні ефективного зберігання та обробки інформації, необхідної для виконання функцій веб-додатку. Разом з тим обговорено створення запитів з серверу до сховища PostgreSQL.

Загальний аналіз вимог, проектування та розробка бази даних надає зрозумілу основу для подальшої імплементації комп’ютерної моделі.

РОЗДІЛ 3
ВИКОРИСТАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ЗАДАЧІ ОПТИМІЗАЦІЇ ПОСЕЛЕННЯ СТУДЕНТІВ

3.1 Аспекти розробки генетичного алгоритму

З наукової точки зору, еволюційний підхід у вигляді реалізації генетичного алгоритму для оптимізації поселення студентів є ефективним методом вирішення задачі оптимізації простору проживання. Він відкриває перспективу пошуку ідеального балансу між різними критеріями, такими як гендер, наявність соціальних об'єктів, а також середньозваженість за курсом студентів. Такий підхід дозволяє ефективно управляти обмеженими ресурсами та забезпечує комфортне житло для кожного студента, сприяючи його успішному навчанню та особистому розвитку.

У цьому контексті важливо досліджувати та розробляти нові стратегії застосування генетичних алгоритмів для оптимізації умов проживання студентів, зокрема для відповіді на підвищені вимоги у сфері вищої освіти. Цей підхід може бути спрямований на створення збалансованих так званих «громад» у кімнаті, що враховують не лише соціологічні аспекти проживання, але й особистість та учбові інтереси кожного студента, призвести до виникнення новаторських рішень, спрямованих на поліпшення якості життя студентів та підтримку їхнього успішного навчання.

У наш час, де збільшується кількість студентів, пошук оптимальних умов для їхнього проживання стає нагальним питанням. З метою вирішення цього завдання виникає необхідність розробки ефективних стратегій розподілу студентів по кімнатам, що враховують різноманітні потреби студентів.

Генетичний алгоритм, інспірований природнім еволюційним процесом, може відіграти важливу роль у цьому контексті. Він дозволить даній комп’ютерній моделі автоматично адаптуватися та знаходити оптимальні рішення, враховуючи перелічені критерії [28].

Далі розглядається впровадження цієї концепції у процес поселення, та детальне пояснення щодо функціонування та вибору критерії для алгоритму.

3.1.1 Визначення параметрів

Для якісного впровадження та реалізації еволюційного підходу, в комп'ютерну модель, перш за все, важливо визначити параметри генетичного алгоритму для оптимізації поселення студентів.

Один із таких параметрів - це функція придатності або функція пристосованості (fitness function). Вона визначає, наскільки кожен індивідуум (комбінація студентів у кімнаті) відповідає визначеним критеріям, таким як курс, гендер і соціотип. Функція придатності має надавати вищі значення тим комбінаціям, які краще відповідають установленим вимогам. За це відповідає обчислення загального показника пристосованості для даного розподілу студентів по кімнатах на основі заданих критеріїв, і кожен з них з різними вагами вносить свій власний вклад в одне число, що оцінює, наскільки вдалим є розподіл студентів з точки зору рівномірності за гендерною належністю, балансу соціотипів та розподілу за курсами.

Ще одним важливим параметром є розмір популяції. Оптимальний розмір популяції повинен бути достатнім для врахування різноманітності студентської групи та забезпечення можливості знаходження оптимального рішення. Занадто мала популяція може привести до ранньої зупинки алгоритму на підоптимальному рішенні, тоді як занадто велика популяція може збільшити обчислювальну складність.

Іншою есенційною частиною є ймовірність мутації. Вона дозволяє змінювати певні характеристики індивідів для додавання різноманітності та уникнення застрягання в локальних мінімумах. Визначення оптимальної ймовірності мутації важливо для збереження різноманітності популяції та прискорення збіжності алгоритму.

Одним із значущих аспектів, є також врахування взаємодії між критеріями. І зважаючи на конкретику щодо окремого співіснування людей з

різними статями, потрібно вказати більшу вагу саме для критерію гендеру, щоб поселити жінок до жінок а чоловіків до чоловіків.

Варто зазначити, що визначення параметрів генетичного алгоритму для задачі поселення студентів у кімнати гуртожитку є доволі кропітким та дуже відповідальним завданням, яке вимагає збалансованого врахування критеріїв та оптимізації функції придатності.

Тому правильне налаштування параметрів може значно полегшити процес та забезпечити максимально задовільні умови проживання для всіх студентів у гуртожитку.

3.1.2 Визначення цілей та обмежень

Першим і найважливішим кроком є чітке визначення цілей, яким має відповідати генетичний алгоритм. Їх формулювання у цьому розумінні окреслює стратегічні напрямки для оптимізації процесу поселення.

Однією з головних для розробки генетичного алгоритму - є максимальна задоволеність студентів умовами життя та соціальним середовищем. Це означає, що алгоритм має ефективно враховувати особисті переваги та особливості кожного студента та забезпечувати йому комфортні та сприятливі умови навчання, а конкретно взаємній підтримці та спільному отримання освіти. Зокрема, генетичному алгоритму необхідно враховувати гендерні аспекти, щоб створити сприятливі «групи» для різних категорій студентів. Так само і для аспекту курсу, на якому вони навчаються, щоб забезпечити різнобарвний рівень академічного досвіду серед студентів.

Окрім максимального задоволення учасників навчального процесу, важливо встановити об’єктивні критерії, які забезпечують справедливий і збалансований розподіл ресурсів гуртожитку. Це обмеження враховує всі інші перешкоди, пов’язані із забезпеченням рівного доступу до житла всіх студентів, уникнення сильного соціального чи гендерного дисбалансу та забезпечення безпеки та комфорту.

Також обмеженнями є кількість кімнат та місць у них.

3.2 Імплементація алгоритму у програний код

Один із загальних сценаріїв застосування генетичних алгоритмів - це задачі комбінаторної оптимізації. Програмний код, представлений у Додатку Г, демонструє використання генетичного алгоритму для розв'язання задачі розподілу студентів по кімнатах.

Спрощено відображаючи природні закономірності, генетичний алгоритм створює та еволюціонує популяцію рішень, у якій кращі адаптовані рішення мають більше шансів бути відібраними для подальшого розмноження та мутації. Кожне рішення представляє собою набір кімнат, в кожній з яких розміщені студенти з урахуванням їхніх особливостей, таких як гендер, курс та соціотип.

Головна ідея генетичного алгоритму, що демонструється - це використання випадкових мутацій та схрещування для створення нових поколінь рішень, які успішніше вирішують поставлену задачу. Крім того, в алгоритмі реалізована адаптація параметрів (імовірності мутації та кросовера) відповідно до середньої пристосованості популяції. Це дозволяє алгоритму адаптуватися до змінних умов та швидше знаходити оптимальні рішення.

Генетичний алгоритм включає в себе п'ять основних етапів: ініціалізація, відбір, схрещування, мутація та оцінка пристосованості. Кожен з цих етапів розглянуто далі.

Почнемо з огляду етапу ініціалізації. Але для початку, щоб розпочати роботу алгоритму, треба визначити вхідні дані, щоб на їх основі створити початкову популяцію.

Спосіб їх отримання визначається запитами до бази даних, щоб отримати список студентів та кімнат з їх кількістю місць. Успішне виконання запитів надають дані про усіх студентів, з котрих беруться параметри, які будуть використовуватися в якості критеріїв поселення, а саме їх стать, курс на якому вони навчаються та їх тип особистості. Також потрібна конкретна кількість кімнат з вільними місцями та кількість цих місць, щоб задати обмеження. Було обрано 28 кімнат, по 4 вільних місця у кожній. Студентів 112.

Після чого у кімнати у хаотичному порядку записуються усі студенти, та заповнюється окремий список, який і буде початковою популяцією. Та варто підкреслити, що геном є структурою даних, яка визначає розподіл студентів по кімнатах. Тобто геном можна ідентифікувати набір кімнат, а кожна кімната представляє собою список студентів з їхніми характеристиками (гендер, курс, соціотип). Розмір популяції складає 100 умовних одиниць.

Після створення популяції, обчислюється пристосованість кожного з індивідів. Після цього вони сортуються у порядку спадання та відбувається вибір першої половини популяції як потенційних батьків.

Метод відбору є простим і він має назву «турнірний відбір». Основна ідея полягає в тому, що чим краще пристосовані особини, тим вони мають більше шансів передавати свої «гени» наступному поколінню, і, таким чином, покращувати якість рішення.

Потім відбувається схрещування, з використання концепції того, що генерується випадкове число у конкретному проміжку і порівнюється з визначеним значення кросоверу. Ця ймовірність контролює, чи буде застосовано кросинговер, або ж обидва батьки просто скопіюються в якості нащадків.

І варто підкреслити, що використовується одноточковий кросовер. Його сенс полягає в тому, що випадковим чином обирається одна точка розрізу в генетичних послідовностях (у цьому випадку, список кімнат). Усе, що знаходиться до цієї точки, передається першому нащадку від одного з батьків, а все, що знаходиться після точки, передається від іншого батька іншому нащадку.

Після кросовера відбувається мутація. І так само використовується ймовірнісний контроль. Тип мутації класифікується як «мутація обміну» або «обмін місцями». Суть цієї мутації полягає в тому, що два студенти обмінюються своїми місцями між різними кімнатами, щоб зберегти корисні комбінації.

Варто зазначити, що особливістю є те, що розроблено адаптивність кросовера та мутації. Це дозволяє алгоритму адаптуватися до змінюючихся умов задачі та підтримувати баланс між розвідкою (exploration) і експлуатацією (exploitation) простору пошуку.

Якщо алгоритм виявляє, що кросовер добре працює на поточному етапі оптимізації, ймовірність кросовера може бути збільшена. З іншого боку, якщо мутація виявляється більш ефективною в більш пізніх етапах, ймовірність мутації може бути збільшена. Це дозволяє адаптивно вибирати операції залежно від того, які саме операції краще пристосовані для поточних умов пошуку. Тобто якщо середнє значення пристосованості близько до 1, то ймовірність кросовера збільшиться, а в іншому випадку вона зменшиться.

Такий підхід допомагає алгоритму краще пристосовуватися до характеристик задачі і поліпшує його здатність знаходження оптимального рішення в змінному середовищі

Ці параметри задані у табл. 3.1 [29]:

Таблиця 3.1

Використані параметри

Значення	Мутація	Кросовер
Мінімальне	0.01	0.6
Максимальне	0.5	0.9
Початкове	0.1	0.8

Переходячи до функції пристосованості, яка призначена для оцінки заданого розподілу студентів по кімнатах, треба зауважити, що на вхід подається дані, які є різними з огляду на кожен етап алгоритму.

Функція calculateFitness, фрагмент якої можна побачити на рис. 3.1, використовує три критерії для оцінки ефективності розподілу студентів по кімнатах в житловому приміщенні. Кожен критерій оцінює різні аспекти

розподілу, такі як стать, тип особистості (інтроверт чи екстраверт) та курс студентів. Вагові коефіцієнти для кожного критерію визначають їхню важливість у визначенні загальної пристосованості.

```
static void calculateFitness(RoomAssignment assignment) {
    List<List<Student>> rooms = assignment.getRooms();
    double fitness = 0.0;

    // Критерій 1: жінки до жінок, чоловіки до чоловіків
    int genderMismatch = 0;
    for (List<Student> room : rooms) {
        boolean isFemaleRoom = false; //  позначає, що це кімната для жінок
        boolean isMaleRoom = false;   //  позначає, що це кімната для чоловіків
        for (Student student : room) {
            if (student.getGender() == 1) { // Якщо студент - жінка
                isFemaleRoom = true;
            } else { // У іншому випадку (чоловік)
                isMaleRoom = true;
            }
            // Якщо і жінки та чоловіки присутні у кімнаті, це не відповілає критерію
            if (isFemaleRoom && isMaleRoom) {
                genderMismatch++;
                break; // Виходимо з циклу, так як вже є невідповідність
            }
        }
    }
```

Рисунок 3.1 – Фрагмент функції пристосованості

Ось як кожен критерій враховується у визначенні загальної пристосованості:

Критерій 1 - жінки до жінок, чоловіки до чоловіків:

- Для кожної кімнати функція перевіряє, чи всі студенти в кімнаті належать до одного статі (чоловіки або жінки).
- Якщо кімната містить як чоловіків, так і жінок, то збільшується лічильник genderMismatch.

Отже, genderMismatch відображає кількість кімнат, де є студенти обох статей.

Критерій 2 - баланс між інтровертами та екстравертами:

- Для кожної кімнати функція підраховує кількість інтровертів та екстравертів.
- Якщо різниця між кількістю інтровертів та екстравертів у кімнаті дорівнює 0, то збільшується sociotypeMismatch.
- Також перевіряється, чи у половині кімнат співвідношення інтровертів та екстравертів рівне (якщо так, то ще один sociotypeMismatch).

Отже, sociotypeMismatch відображає невідповідності у співвідношенні інтровертів та екстравертів.

Критерій 3 - розміщення студентів певних курсів у одну кімнату:

- Функція рахує кількість студентів для кожного курсу та середню різницю між курсами.
- Для кожного студента в кімнаті перевіряється, чи його курс відхиляється від середньої різниці більше, ніж на один.
- Якщо так, збільшується courseMismatch.

Отже, courseMismatch відображає різницю в розподілі студентів за курсами в кімнатах.

Загальна пристосованість (fitness):

- Призначується загальна пристосованість, яка обчислюється за формулою: fitness = genderMismatch * 0.8 + sociotypeMismatch * 0.1 + courseMismatch * 0.1.
- Кожен критерій має ваговий коефіцієнт, який визначає його вплив на загальну пристосованість.

Якщо поглянути на ваги, то можна чітко бачити, що найбільш вагомим є саме розподіл за гендером, так як це є ключовим у контексті комфорту проживання студентів.

Також варто підкреслити, що чим менше значення пристосованості, тим більш ефективний вважається розподіл студентів.

Кількість ітерацій алгоритму складає 10000.

На виході цього алгоритму, як вже зазначалося раніше, у консоль виводяться дані про те, як алгоритм розселив студентів по кімнатах, що дає змогу дати оцінку його якості роботи.

Фрагмент результату після відпрацювання алгоритму зображений на рис. 3.2 [29].

```
Кімната 1:
Гендер: чоловік, Курс: 6, Соціотип: інтроверт
Гендер: чоловік, Курс: 5, Соціотип: інтроверт
Гендер: чоловік, Курс: 6, Соціотип: інтроверт
Гендер: чоловік, Курс: 5, Соціотип: екстраверт

Кімната 2:
Гендер: чоловік, Курс: 3, Соціотип: інтроверт
Гендер: чоловік, Курс: 3, Соціотип: інтроверт
Гендер: чоловік, Курс: 5, Соціотип: інтроверт
Гендер: чоловік, Курс: 3, Соціотип: екстраверт

Кімната 3:
Гендер: чоловік, Курс: 5, Соціотип: інтроверт
Гендер: чоловік, Курс: 3, Соціотип: інтроверт
Гендер: чоловік, Курс: 6, Соціотип: інтроверт
Гендер: чоловік, Курс: 1, Соціотип: інтроверт

Кімната 4:
Гендер: жінка, Курс: 3, Соціотип: інтроверт
Гендер: жінка, Курс: 4, Соціотип: екстраверт
Гендер: жінка, Курс: 6, Соціотип: інтроверт
Гендер: жінка, Курс: 1, Соціотип: інтроверт

Кімната 5:
Гендер: чоловік, Курс: 6, Соціотип: інтроверт
Гендер: чоловік, Курс: 3, Соціотип: інтроверт
Гендер: чоловік, Курс: 6, Соціотип: інтроверт
Гендер: чоловік, Курс: 1, Соціотип: інтроверт
```

Рисунок 3.2 – Фрагмент результату після роботи алгоритму

Як можна бачити, алгоритм впорався досить непогано. Але варто зазначити, що так як алгоритм працює за допомогою імовірності, то існує варіант розселення, коли у деякі кімнати поселилися студенти різних гендерів (рис. 3.3).

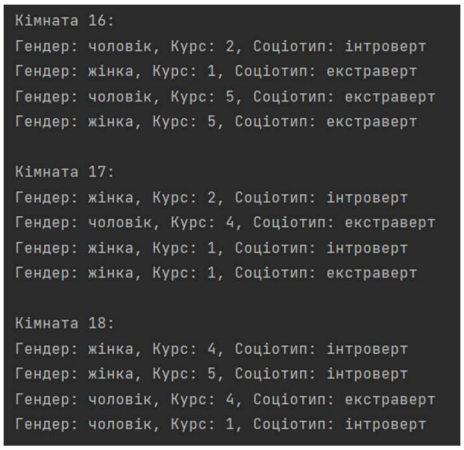

Рисунок 3.3 – Студенти з різними гендерами у кімнаті

Для вирішення цієї проблеми було реалізовано алгоритм, який шукає особу, яка підходить до поточної комбінації у кімнаті, і міняє їх місцями з особинами з іншої кімнати, аби усі студенти були поселені коректно.

Висновки за розділом 3

У даному розділі було детально розглянуто аспекти використання еволюційного підходу для вирішення задачі оптимізації поселення студентів., а саме реалізацією генетичного алгоритму. Цей підхід виявляється дуже ефективним і перспективним у контексті оптимізаційних завдань, пов'язаних із розподілом обмежених ресурсів, таких як житловий простір в університетських гуртожитках.

Встановлено, що важливою частиною цього процесу було визначення оптимальних значень для певних параметрів, таких як розмір популяції, ймовірність хромосомного мутування та інші. Це дозволяє забезпечити ефективну роботу алгоритму. Також, проаналізувавши процес визначення цілей та обмежень для генетичного алгоритму, було враховано реальні умови задачі, такі як поселення студентів одного гендеру, їх курс та соціотип. Цільова функція повинна враховувати індивідуальні потреби студентів і максимізувати їх задоволеність умовами проживання.

Зокрема, дивлячись на імплементацію генетичного алгоритму у програмний код, потрібно зробити висновок, що цей етап є ключовим для успішного використання алгоритму в реальних умовах. Важливо враховувати оптимізації та оптимальні структури даних для швидкої та ефективної роботи алгоритму при обробці великої кількості даних та особливо приділити увагу коректності розселення за головним критерієм, а саме гендеру.

Загальна концепція використання генетичного алгоритму для задачі оптимізації поселення студентів є перспективною і варта подальших досліджень. Результати подальших експериментів та реалізацій можуть принести нові важливі відкриття і вдосконалення в області оптимізаційних завдань для університетських гуртожитків.

ВИСНОВКИ

Представлена кваліфікаційна робота спрямована на створення комп’ютерної моделі автоматизації процесу поселення студентів у гуртожиток.

У першому розділі проведено аналіз сучасних комп’ютерних моделей автоматизації поселення студентів, що визначило необхідність розвитку більш ефективних та гнучких систем. Еволюційний підхід, і зокрема генетичні алгоритми, були обрані як потенційно ефективний інструмент для вирішення цієї задачі.

Другий розділ присвячений розробці комп’ютерної моделі з розподіленою архітектурою. Аналіз потреб користувачів дозволив визначити функціональні вимоги, а проектування архітектури включило в себе обґрунтування вибору компонентів, конфігурацію проекту та розробку бази даних. Ці кроки стали основою для подальшої реалізації генетичного алгоритму.

У третьому розділі надано детальний опис використання генетичного алгоритму для оптимізації процесу поселення студентів. Аспекти розробки включають визначення параметрів та цілей, а імплементація алгоритму у програмний код забезпечила його функціональність та ефективність.

Усі ці розділи об'єднуються в комплексний підхід до вирішення проблеми поселення студентів, що використовує передові технології та методи. Завдяки використанню генетичних алгоритмів вдалося досягти оптимальності та гнучкості в процесі прийняття рішень.

Ця робота слугує прикладом успішного впровадження інноваційних технологій у сфері управління студентським житлом. Результати дослідження не тільки сприятимуть поліпшенню конкретного процесу поселення студентів, а й відкривають шлях до подальших досліджень у сфері оптимізації процесів за допомогою еволюційних підходів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Солощенко О.М. Моделі і методи оцінювання кредитоспроможності фізичних осіб : дис. … к.т.н. : 01.05.04 / КПІ ім. Ігоря Сікорського. Київ. 2016. С. 26.
2. Васильєв О. Б., Васильєва Н. С., Кічмаренко О. Д. Методи розв’язування задач багатокритеріальної оптимізації : метод. вказ. Одеса: Одеський національний університет імені І. І. Мечникова, 2017. — 48 с.
3. Голіков М.С. Аналіз комп’ютерних моделей автоматизації поселення студентів у гуртожиток. *Комп’ютерне моделювання у наукоємних технологіях* : зб. наукових праць VIII Міжнар. наук.-техн. конф. Харків : ХНУ імені В.Н. Каразіна, 2022. – С. 32- 34.
4. Ус С.А., Коряшкіна Л.С. Моделі й методи прийняття рішень: навч. посіб. Дніпропетровськ. Нац. гірн. ун-т. – Д. : НГУ. 2014. – 300 с.
5. Струк Є., Дубленич Р., Фабрі Л., Автоматизація процесу поселення студентів у гуртожиток студмістечка. *Вісник національного університету «Львівська політехніка» Комп’ютерні науки та інформаційні технології.* 2016. Серія №843, С. 35-42.
6. Bruce Eckel. Thinking in Java. 4th ed. p. cm. Includes bibliographical references and index : книга. United States of America : Courier in Stoughton, 2006. 1057 с.
7. Корнієнко В.І., Гусєв О.Ю., Герасіна О.В. Інтелектуальне моделювання нелінійних динамічних процесів у системах керування, кібербезпеки, телекомунцікайцій : підручник. Нац. техн. ун-т «Дніпровська політехніка». Дніпро : НТУ «ДП», 2020. 536 с.
8. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми. Київ : Корнійчук, 2008. 446 с.
9. Кузьміна Н. М. Деякі методичні аспекти навчання генетичних алгоритмів. *Проблеми інформатизації навчального процесу в закладах*

загальної середньої та вищої освіти : зб. праць. Всеукр. Наук.-практ. кофн. Київ : НПУ імені М.П. Драгоманова, 2018. С. 60.

10. Миколович Ю. В. Модель розвитку простих організмів з використанням генетичних алгоритмів та глибинного навчання : маг. дис. на здобуття наук. ступеня магістра : 06.05.2019. Київ, 2019. 71 с.
11. Goldberg D.E., Deb Kalyanmoy, A Comparative Analysis of Selection Schemes Used in Genetic Algorithms : книга. Вид. 1-ше. USA, 1991. С. 69–93.
12. Whitley Darrell, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best" : proceedings of the Third International Conference on Genetic Algorithms (ICGA). USA: Morgan Kaufmann Publishers Inc., 1989. С. 116–121.
13. Holland J.H. Adaptation in natural and artificial systems : PhD thesis, The University of Michigan. UK : The MIT Press, 1975. 211 с.
14. Nataliya Boyko, Using genetic algorithms for modelling informational processes. *Computational problems of electrical engineering.* 2016. Volume 6, Number 1. С. 55-61
15. Люшенко Л.А., Хіцко Я.В. Розробка та аналіз вимог до програмного забезпечення. Курсове проєктування з дисципліни компоненти програмної інженерії : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2020. 63 с. URL: https://ela.kpi.ua/bitstream/123456789/38101/1/Rozrobka_ta_analiz_KP.pdf (дата звернення: 02.09.2023).
16. Linked.in. Social network : веб-сайт. URL : https://www.linkedin.com/pulse/importance-requirement-analysis-computer-science-muntahi-i-rabbani (дата звернення: 10.09.2023).
17. Adams K.M. "Definitions for Functional and Non-Functional Requirements". – книга. USA : Springer, 2015. С. 45–50.
18. Three-Tier Architecture Approach for Custom Applications. IT Goals Achieved with Zirous, Technology Solutions : веб-сайт. URL:

https://www.zirous.com/2022/11/15/three-tier-architecture-approach-for-custom-applications-2/ (дата звернення: 12.10.2023).

19. Spring. Spring makes Java simple : веб-сайт. URL : https://spring.io/ (дата звернення: 24.01.2023).
20. Baeldung. Learn Java : веб-сайт. URL : https://www.baeldung.com/ (дата звернення: 24.01.2023).
21. Postgresql. База даних : веб-сайт. URL : https://www.postgresql.org/ (дата звернення: 02.09.2023).
22. Apache Maven. Software project management and comprehension tool : веб-сайт. URL : https://maven.apache.org/ (дата звернення: 15.01.2023).
23. Angular. Deliver web apps with confidence : веб-сайт. URL : https://angular.io/ (дата звернення: 06.01.2023).
24. Пасічник В.В., Резніченко В.А. Організація баз даних та знань : підручник МОНУ. Київ: BHV, 2006. 384 с.
25. Visual-paradigm. Diagrams : веб-сайт. URL : https://www.visual-paradigm.com/ (дата звернення: 20.09.2023).
26. «Розробка та супровід проблемно-орієнтованих ПС» / проф. М.В. Ткачук/каф. МСТ, ХНУ ім. В.Н. Каразіна.
27. Creatly. Діаграми : веб-сайт. URL : https://app.creately.com/ (дата звернення: 02.10.2023).
28. Сучасні інформаційні технології: матеріали засідань школи-семінару. Виипуск 7. Харків: ХНУ ім. Каразіна, 2023. – 84 с.
29. Голіков М.С., Булавін Д.О. Еволюційний підхід для задачі оптимізації розселення студентів. *Комп'ютерне моделювання у наукоємних технологіях* : зб. наукових праць IX Міжнар. наук.-техн. конф. Харків : ХНУ імені В.Н. Каразіна, 2023. – С. 45- 48.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: **15 – Автоматизація та приладобудування**
Спеціальність: **151 – «Автоматизація та комп'ютерно-інтегровані технології»**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної та прикладної системотехніки
_____д.т.н., проф. Шматков С. І.

«08 » грудня 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Голіков Максим Сергійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Комп'ютерна модель автоматизації процесу поселення студентів у гуртожиток.»
керівник роботи_____Булавін Дмитро Олексійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» листопада 2023 року № 4101-5/3197

2. Строк подання студентом роботи 28 .11.2023

3. Перелік питань, які потрібно розробити)
1.Аналіз предметної області та постановка задачі.
2. Огляд мов розробки та баз даних для створення моделі.
3. Створення алгоритму розробки моделі автоматизації процесу поселення студентів у гуртожиток.
4. Програмна реалізація моделі автоматизації процесу поселення студентів у гуртожиток.
5. Тестування та дослідження ефективності роботи програмної моделі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Аналіз предметної області та постановка задачі.	08.12.2022 – 05.03.2023
2	Огляд мов розробки та баз даних для створення моделі.	05.03.2023 – 01.04.2023
3	Створення алгоритму розробки моделі автоматизації.	01.04.2023 – 29.06.2023
4	Програмна реалізація моделі автоматизації процесу поселення студентів у гуртожиток.	17.06.2023 – 24.08.2023
5	Побудова схеми бази даних	15.08.2023 – 17.09.2023
6	Оформлення пояснювальної записки.	01.09.2023 – 27.10.2023
7	Представлення кваліфікаційної роботи керівнику та рецензенту	28.10.2023 – 28.11.2023

5. Дата видачі завдання 08.12.2022

Студент М.С. Голіков _____
ініціали, прізвище підпис

Керівник роботи Д.О. Булавін _____
ініціали, прізвище підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання

на розробку програмного виробу

«Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток. 1.2. Галузь застосування: Автоматизація процесів.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 151 - «Автоматизація та комп’ютерно-інтегровані технології» 2.2. Завдання на кваліфікаційну роботу магістра затверджено наказом ректора № 4101-5/3197 від 10.11.2023р.
3.Призначення розробки	3.1. Мета розробки програмного виробу: розробка комп'ютерної моделі автоматизації процесу поселення студентів у гуртожиток. Комп’ютерна модель буде ефективно використовуватися для полегшення процесу поселення студентів та вирішення багатьох проблем, пов'язаних з організацією та проведенням цього процесу, забезпечивши більш ефективну та зручну роботу для всіх сторін. 3.2. Призначення програмного виробу: автоматизація процесу поселення студентів у гуртожиток. 3.3. Вихідні дані для розробки: автоматизована система процесу поселення студентів у гуртожитки студмістечка. У якості прототипу та антагоніста взята система описана у науковому збірнику «Праці міжнародної науково-технічної конференції

	«Комп'ютерне моделювання у наукоємних технологіях (КМНТ-2022)», (21-25 листопада 2022 р., м. Харків, Україна). – Харків: ХНУ, 2022. // Голіков М.С. Аналіз комп’ютених моделей автоматизації процесу поселення студентів у гуртожиток.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: можливість розселяти студентів по кімнатах за критеріями. 4.2. Вимоги до надійності: забезпечити користувачам безпеку за допомогою аутентифікації під час входу на свою сторінку. 4.3. Вимоги до умов експлуатації: встановлення мови програмування Java, фреймворку Angular та бази даних PostgreSql, а також усіх інших необхідних бібліотек. 4.4. Вимоги до складу і параметрів технічних засобів: комп’ютер або ноутбук із 4 ГБ оперативної пам’яті та процесором не нижче Intel Core i3 9-го покоління. 4.5. Вимоги до інформаційної та програмної сумісності: ОС Linux або Windows 8/10, Java 8, npm 16.16.0 4.6. Вимоги до маркування та упаковки: жорстка упаковка з пластмаси, маркування на українській та англійській мовах. 4.7. Вимоги до транспортування і зберігання: транспортування в упаковці будь-яким способом, температура транспортування/зберігання +5-+20 С. 4.8. Спеціальні вимоги: відсутні.

<table>
<tr><td>5. Вимоги до програмної документації.</td><td colspan="2">Програмною документацією до виробу «Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток» вважати:
1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).
3) Опис програмного виробу (представити в розділах 2, 3 пояснювальної записки до кваліфікаційної роботи).
4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>6. Техніко економічні показники</td><td colspan="2">Орієнтовна оцінка термінів розробки комп’ютерної моделі складає 5 місяців. Витрати грошових коштів відсутні.</td></tr>
<tr><td rowspan="2">7. Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>08.12.2022 - 05.03.2023
05.03.2023 - 01.04.2023
01.04.2023 - 29.06.2023
17.06.2023 - 24.08.2023</td><td>1. Аналіз предметної області та постановка задачі.
2. Огляд мов розробки та баз даних для створення моделі.
3. Створення алгоритму розробки моделі автоматизації.
4. Програмна реалізація моделі автоматизації процесу поселення студентів у гуртожиток.</td></tr>
</table>

	15.08.2023 - 17.09.2023 23.08.2022 - 27.09.2022 28.10.2023 - 28.11.2023	5. Побудова схеми бази даних 6. Програмна реалізація моделі автоматизації процесу поселення студентів у гуртожиток. 7. Розробка рекомендацій щодо застосування комп'ютерної моделі автоматизації поселення студентів у гуртожиток.
8. Порядок контролю і приймання	В даному розділі повинні бути вказані загальні вимоги до приймання розробленого програмного виробу наприклад: 1) Перевірка ходу розробки програмного виробу. Керівнику робіт виконувати 1 раз в 3 тижні. 2) Випробування програмного виробу відповідно до Програми і методики випробувань провести на базі комп'ютерного класу або приватного приміщення. 3) Захист розробленого програмного виробу провести на засіданні атестаційної комісії. 4) Пояснювальну записку надати на паперових носіях в одному примірнику, в електронному вигляді.	

Виконавець:
студент групи КУ-61
Голіков М.С.
___Golikov_

Замовник:
к.т.н., доцент
Булавін Д.О.
___ ___

Додаток В

Програма і методика випробувань програмного виробу

«Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток»

1. Об'єкт випробувань

1. Назва програмного виробу: «Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток».
2. Галузь застосування: Автоматизація процесів.
3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації комп’ютерної моделі заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до дипломної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри або приватного приміщення в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

– працювати на основних операційних системах: Windows, Linux, MacOS;

– вимоги до надійності;

– передбачити захист від некоректних дій користувача;

– сумісність з іншими програмними продуктами;

– бути легко розширюваною;

– елементи програми повинні бути ізольовані одне від одного для зменшення їх впливу на роботи програми під час редагування програмного коду;

– вимоги до складу і параметрів технічних засобів;

– вимоги до маркування та упаковки (не висуваються);

– вимоги до транспортування і зберігання (не висуваються).

– Спеціальні вимоги (не пред’являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Комп’ютерна модель автоматизації процесу поселення студентів у гуртожиток» вважати:

– справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);

– програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);

– рекомендацій щодо застосування створеної програмної стандартизації у проектах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи);

– текст програми(представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Для проведення випробувань необхідний проект для розробки програмної моделі за допомогою мови програмування Java, з використання бази даних PostgreSql, фреймворків Angular та Spring Boot.

6.2. Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);

- випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

3. Перевірку комплектності складу технічних і програмних засобів.

4. Методику проведення перевірок на 1 етапі випробувань.

5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. Перевірку відповідності технічних характеристик програми вимогам технічного завдання.

2. Перевірку ступеня виконання функціональних вимог до програми.

3. Методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

1. Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux та MacOS.

2. Для роботи необхідна мова програмування Java, версії не нижче 8.

3. Порядок проведення випробувань:

3.1. Спочатку потрібно запустити IntelijIDEA, і запустити код на Angular та Spring Boot натиснувши кнопку запуску.

3.2. Після запуску програми необхідно пройти етап авторизації у браузері за конкретним доменом.

3.3. Після чого необхідно щоб адміністратор запустив процес поселення на тестових даних.

3.4. Після запуску у консолі з явиться результат поселення.

Для проведення випробувань пропонується провести тест 1, тест 2 та тест 3.

Тест 1

1. Перевірка виконання авторизації користувача.

2. Натискання на кнопку відправлення даних для авторизації на сервер.

3. Отримання результатів у вигляді повідомлення про роль, під якою зайшов користувач, що свідчить про успішну авторизацію.

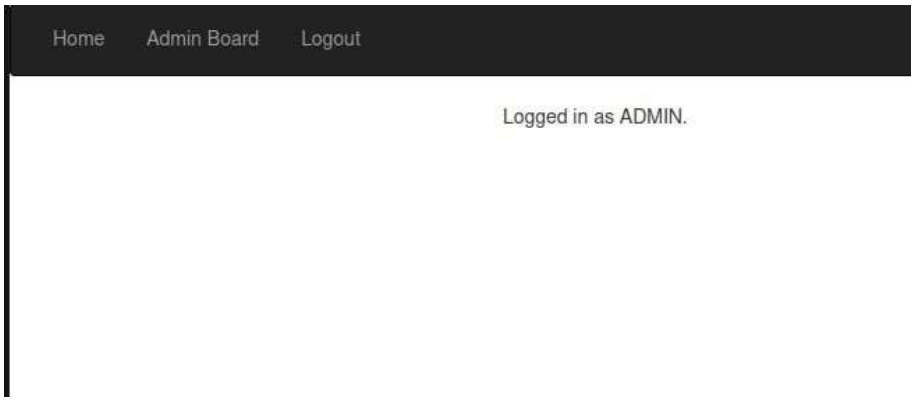

Рисунок В.1 – Тест 1

Тест 2

1. Перевірка виконання запиту та отримання відповіді у вигляді файлу JSON формату, даних про усіх студентів.

2. Запит робиться зі сторони сервера до бази даних.

3. Отримання результату виконання в консолі.

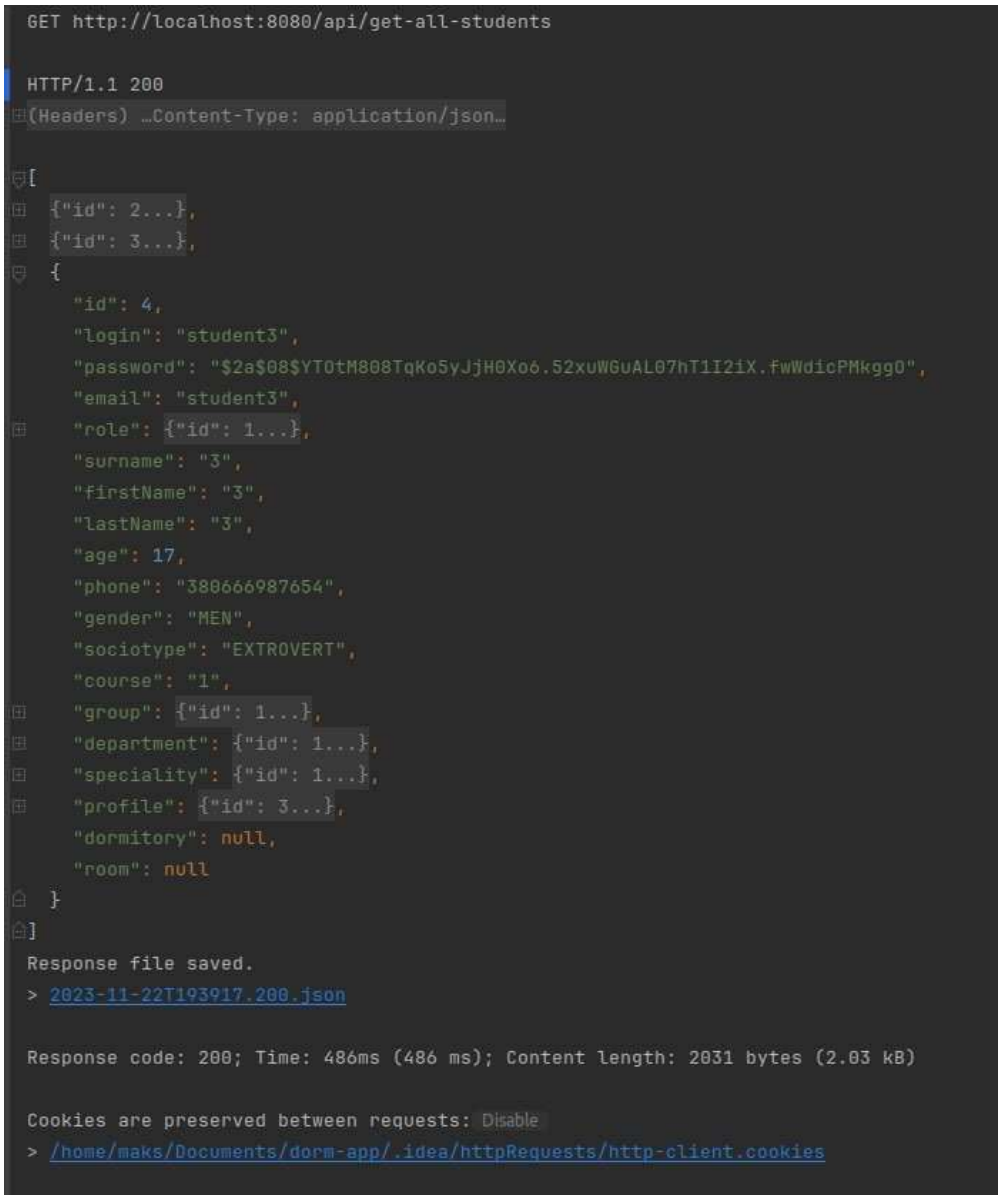

Рисунок В.2 – Тест 2

Тест 3

1. Перевірка виконання генетичного алгоритму для поселення студентів у гуртожиток.

2. Запуск алгоритму на основі тестових даних.

3. Отримання результатів виконання алгоритму в консоль, у вигляді кімнат та поселених студентів.

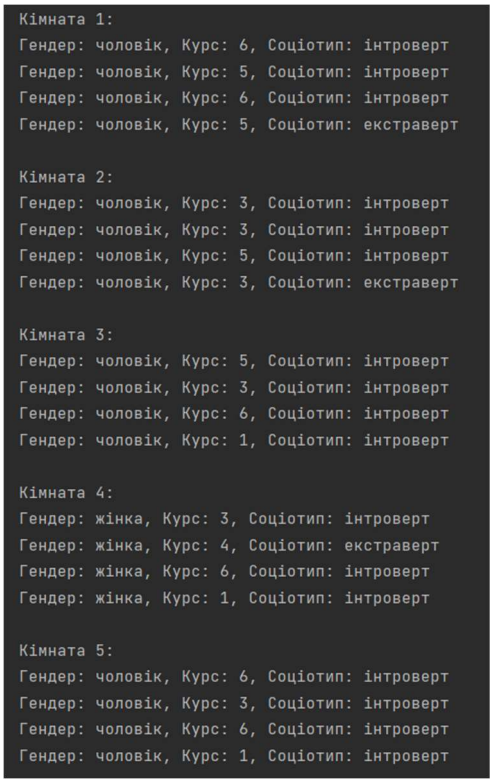

Рисунок В.3 – Тест 3

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: випробування пройшло успішно, оскільки кожен з тестів показав очікуванні результати.

Виконавець:

студент групи КУ-61, Голіков М. С.

Додаток Г

Лістинг програмного коду

Генетичний алгоритм:

```
package com.company.springboot;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

class RoomAssignment {
    private List<List<Student>> rooms;
    private double fitness; // Значення пристосованості

    public RoomAssignment(List<List<Student>> rooms) {
        this.rooms = rooms;
    }

    public RoomAssignment(List<List<Student>> rooms, double fitness) {
        this.rooms = rooms;
        this.fitness = fitness;
    }

    public List<List<Student>> getRooms() {
        return rooms;
    }

    public double getFitness() {
        return fitness;
    }

    public void setRooms(List<List<Student>> rooms) {
        this.rooms = rooms;
    }

    public void setFitness(double fitness) {
        this.fitness = fitness;
    }

    static void calculateFitness(RoomAssignment assignment) {
        List<List<Student>> rooms = assignment.getRooms();
        double fitness = 0.0;

        // Критерій 1: жінки до жінок, чоловіки до чоловіків
        int genderMismatch = 0;
        for (List<Student> room : rooms) {
            boolean isFemaleRoom = false; //  позначає, що це кімната для
жінок
            boolean isMaleRoom = false;   //  позначає, що це кімната для
чоловіків
            for (Student student : room) {
                if (student.getGender() == 1) { // Якщо студент - жінка
                    isFemaleRoom = true;
                } else { // У іншому випадку (чоловік)
                    isMaleRoom = true;
                }
                // Якщо і жінки та чоловіки присутні у кімнаті, це не
відповідає критерію
                if (isFemaleRoom && isMaleRoom) {
                    genderMismatch++;
```

```
                    break; // Виходимо з циклу, так як вже є невідповідність
                }
            }
        }
        //якщо хоч 1 не той що треба, то genderMissmatch++;

        // Критерій 2: Баланс між інтровертами та екстравертами
        int sociotypeMismatch = 0;
        int totalStudents = 0;

        for (List<Student> room : rooms) {
            int introvertCount = 0;
            int extravertCount = 0;

            for (Student student : room) {
                if (student.getSociotype() == 0) { // 0 - интроверт, 1 - экстраверт
                    introvertCount++;
                } else {
                    extravertCount++;
                }
            }

            totalStudents += room.size();

            int sociotypeDifference = Math.abs(introvertCount - extravertCount);

            if (sociotypeDifference == 0) {
                // Збільшуємо sociotypeMismatch, якщо різниця дорівнює 0 (половина інтровертів, половина екстровертів)
                sociotypeMismatch++;
            }
        }

        // Якщо у половині кімнат співвідношення інтровертів та екстравертів рівна, то збільшуємо sociotypeMismatch
        if (sociotypeMismatch == totalStudents / 2) {
            sociotypeMismatch++;
        }
        //якщо 3 экстраверти, то ок (0.75 коф +-), якщо 2 екстарверти то непідходить, і тоді sociotypeMismatch++


        // Критерій 3: Розміщення студентів певних курсів у одну кімнату
        int courseMismatch = 0;
        Map<Integer, Integer> courseCounts = new HashMap<>();
        int totalCourses = 0;
        int totalStudent = 0;

        // Спочатку обчислюємо суму всіх курсів та кулькість студентів
        for (List<Student> room : rooms) {
            for (Student student : room) {
                int course = student.getCourse();
                courseCounts.put(course, courseCounts.getOrDefault(course, 0) + 1);

                totalCourses += course;
                totalStudent++;
            }
        }

        // Обчислюємо середню різницю між курсами
        int averageCourseDifference = totalCourses / totalStudent;
```

```
        // Перерозрахуємо courseMismatch
        for (List<Student> room : rooms) {
            for (Student student : room) {
                int course = student.getCourse();
                int courseCount = courseCounts.get(course);
                int courseDifference = Math.abs(course -
averageCourseDifference);

                // Якщо різниця більше 1, то збільшуємо courseMismatch
                if (courseDifference > 1) {
                    courseMismatch++;
                }
            }
        }

        // якщо 1 і 5 то courseMismatch++, якщо 3 4 5 то (то ок), якщо 1 2 5
6 (courseMismatch++),
        // (середня різниця між курсами, якщо більше 1 то courseMismatch++,
по іншому ок)
        //ідея селити людей приближених курсів, а не максимально однакових
(середня різниця між курсами)

        // Обчислюємо загальну пристосованість
        fitness = (genderMismatch * 0.8 + sociotypeMismatch * 0.1 +
courseMismatch * 0.1);

        //System.out.printf("%f%n",fitness);
        // Встановлюємо пристосованість для призначення
        assignment.setFitness(fitness);
    }

    @Override
    public String toString() {
        return "RoomAssignment{" +
                "rooms=" + rooms +
                ", fitness=" + fitness +
                '}';
    }
}
```

```
package com.company.springboot;


import org.checkerframework.checker.units.qual.A;

import java.util.*;
import java.util.stream.Collectors;

public class RoomAssignmentGA {
    private static final int POPULATION_SIZE = 100;
    private static double MUTATION_RATE = 0.1;
    private static final int MAX_GENERATIONS = 10000;

    private static double CROSSOVER_RATE = 0.8; // Початкова вірогідність
кроссовера
    private static double MUTATION_RATE_MIN = 0.01; // Початкова вірогідність
мутації
    private static double MUTATION_RATE_MAX = 0.5; // Початкова вірогідність
мутації
```

```
    private static double CROSSOVER_RATE_MIN = 0.6; // Початкова вірогідність
кросовера
    private static double CROSSOVER_RATE_MAX = 0.9; // Початкова вірогідність
кросовера
    private static double ADAPTATION_THRESHOLD = 0.01; // Максимальне
значення адаптації

    private static List<Student> students; // Список студентів

    private static List<Integer> numRooms = new ArrayList<>();

    public static void main(String[] args) {
        // Ініціалізуєм список студентів (тут потрібні реальні дані)
        students = initializeStudents();

        // Створюємо початкову популяцію
        List<RoomAssignment> population = initializePopulation();

        for (int generation = 1; generation <= MAX_GENERATIONS; generation++)
{
            // Відбір и кросовер
            population = evolvePopulation(population);

            // Знаходимо найкраще рішення у теперішнбої популяції
            RoomAssignment bestAssignment = Collections.min(population,
Comparator.comparing(RoomAssignment::getFitness));
            System.out.println("Покоління " + generation + ": Краща
пристосованість = ");
            System.out.printf("%f%n", bestAssignment.getFitness());

        }

        // Виводимо підсумкове рішення
        RoomAssignment bestAssignment = Collections.min(population,
Comparator.comparing(RoomAssignment::getFitness));
        printRoomAssignment(bestAssignment.getRooms());

        System.out.println("++++++++++++++++++++++++++++++++++");


        // Виконуємо коррекцію кімнат


        //printRoomAssignment(bestAssignment);

        List<List<Student>> rooms =  adjustRooms(bestAssignment.getRooms());
        for (int i = 0; i < rooms.size(); i++) {
            //System.out.printf( "%s", assignment.getFitness());
            System.out.println("Кімната " + (numRooms.get(i)) + ":");
            List<Student> room = rooms.get(i);
            for (Student student : room) {
                System.out.println("Гендер: " + (student.getGender() == 0 ?
"чоловік" : "жінка") +
                        ", Курс: " + student.getCourse() +
                        ", Соціотип: " + (student.getSociotype() == 0 ?
"інтроверт" : "екстраверт"));
            }
            System.out.println();
        }

    }

    // Метод для обміну студентами між кімнатами
```

```
    private static void swapStudents(List<Student> room1, List<Student>
room2, int index1, int index2) {
        Student student1 = room1.get(index1);
        Student student2 = room2.get(index2);
        room1.set(index1, student2);
        room2.set(index2, student1);
    }

    private static List<List<Student>> adjustRooms(List<List<Student>> rooms)
{
        List<List<Student>> roomsCopy = new ArrayList<>(rooms);

        for (int i = 0; i < rooms.size(); i++) {
            List<Student> room = rooms.get(i);
            List<Student> maleStudents = new ArrayList<>();
            List<Student> femaleStudents = new ArrayList<>();

            for (Student student : room) {
                if (student.getGender() == 0) {
                    maleStudents.add(student);
                } else {
                    femaleStudents.add(student);
                }
            }

            // Якщо у кімнаті студенти одного гендеру то нічого не робимо
            if (maleStudents.size() == 0 || femaleStudents.size() == 0) {
                roomsCopy.remove(room);
                continue; // Пропускаємо кімнати з однаковим гендером
            }

            numRooms.add(i+1);

        }

        for (int i = 0; i < roomsCopy.size(); i++) {

            if(rooms.size() == 0){
                System.out.println("ROOMS HAS 0 ROOM, ALL STUDENTS IS
CORRECT");
                break;
            }

            if(rooms.size() == 1){
                System.out.println("ROOMS HAS 1 ROOM");
                break;
            }

            int countOfDifferentGender = 0;
            Student firstStudent = null;

            int genderToMatch = 1;

            List<Student> studentsWithDifferentGender =
roomsCopy.get(i).stream()
                    .filter(student -> student.getGender() != genderToMatch)
                    .collect(Collectors.toList());

            int countOfStudentsWithDifferentGender =
studentsWithDifferentGender.size();

            System.out.println("Кількість студентів з іншим гендером: " +
countOfStudentsWithDifferentGender);
```

```
            for (int j = 0; j < roomsCopy.get(i).size(); j++) { // цикл по
студентам у кімнаті
                if (roomsCopy.get(i).get(j).getGender() == 1) { // якщо у
кімнаті студент жіночого гендеру
                    // Знаходимо інший гендерний студент в іншій кімнаті
                    for (int k = 0; k < roomsCopy.size(); k++) {
                        if (k != i) { // не перевіряємо поточну кімнату
                            for (int m = 0; m < roomsCopy.get(k).size(); m++)
{
                                if (roomsCopy.get(k).get(m).getGender() == 0)
{ // знашли студента чоловічого гендеру
                                    // Обмін студентами між кімнатами
                                    swapStudents(roomsCopy.get(i),
roomsCopy.get(k), j, m);
                                    break; // виходимо з циклу, оскільки ми
вже знайшли підходящого студента
                                }
                            }
                        }
                    }
                }
            }


        }

        return roomsCopy;

    }



    private static List<Student> initializeStudents() {

        List<Student> students = new ArrayList<>();
        Random random = new Random();

        for (int i = 0; i < 111; i++) {
            int gender = random.nextInt(2);
            int course = random.nextInt(6) + 1;
            int sociotype = random.nextInt(2);
            students.add(new Student(gender, course, sociotype));
        }

        return students;
    }

    static int numRoom = 28;
    static int numPlaces = 4;

    private static List<RoomAssignment> initializePopulation() {
        List<RoomAssignment> population = new ArrayList<>();

        for (int i = 0; i < POPULATION_SIZE; i++) {
            List<Student> studentsCopy = new ArrayList<>(students);
            List<List<Student>> rooms = new ArrayList<>();

            for (int j = 0; j < numRoom; j++) {          //кількість кімнат
                List<Student> room = new ArrayList<>();
                for (int k = 0; k < numPlaces; k++) {        //кількість місць
в кімнаті
                    if (!studentsCopy.isEmpty()) {
```

```
                        int randomIndex = (int) (Math.random() *
studentsCopy.size());
                        room.add(studentsCopy.get(randomIndex));
                        studentsCopy.remove(randomIndex);
                    }
                }
                rooms.add(room);
            }
            RoomAssignment assignment = new RoomAssignment(rooms);
            RoomAssignment.calculateFitness(assignment);
            population.add(assignment);
            //population.add(new RoomAssignment(rooms));
        }

        return population;
    }


    private static List<RoomAssignment> evolvePopulation(List<RoomAssignment>
population) {

        List<RoomAssignment> newPopulation = new ArrayList<>();
        int populationSize = population.size();

        population.sort(Comparator.comparing(RoomAssignment::getFitness));

        List<RoomAssignment> copyPopulation = new ArrayList<>(population);
        if (Math.random() < ADAPTATION_THRESHOLD) {
            double averageFitness = copyPopulation.stream()
                    .mapToDouble(RoomAssignment::getFitness)
                    .average()
                    .orElse(0.0);

            MUTATION_RATE = Math.max(MUTATION_RATE_MIN, MUTATION_RATE * (1 +
(averageFitness - 1)));
            MUTATION_RATE = Math.min(MUTATION_RATE_MAX, MUTATION_RATE);
            CROSSOVER_RATE = Math.min(CROSSOVER_RATE_MAX, CROSSOVER_RATE * (1
+ (averageFitness - 1)));
            CROSSOVER_RATE = Math.max(CROSSOVER_RATE_MIN, CROSSOVER_RATE);
        }

        int numParents = populationSize / 2;
        List<RoomAssignment> parents = population.subList(0, numParents);

        for (int i = 0; i < numParents; i += 2) {

            RoomAssignment parent1 = parents.get(i);
            if(numParents-i==1){
                newPopulation.add(parent1);
                break;
            }
            RoomAssignment parent2 = parents.get(i + 1);

            if (Math.random() < CROSSOVER_RATE) {
                List<List<Student>> child1Rooms =
crossover(parent1.getRooms(), parent2.getRooms());
                List<List<Student>> child2Rooms =
crossover(parent2.getRooms(), parent1.getRooms());


                if (Math.random() < MUTATION_RATE) {
                    child1Rooms = mutate(child1Rooms);
                }
```

```
                if (Math.random() < MUTATION_RATE) {
                    child2Rooms = mutate(child2Rooms);
                }

                RoomAssignment child1Assignment = new
RoomAssignment(child1Rooms);
                RoomAssignment child2Assignment = new
RoomAssignment(child2Rooms);

                RoomAssignment.calculateFitness(child1Assignment);
                RoomAssignment.calculateFitness(child2Assignment);


                newPopulation.add(child1Assignment);
                newPopulation.add(child2Assignment);
            }
        }

        newPopulation.addAll(parents);

        return newPopulation;
    }

    private static List<List<Student>> crossover(List<List<Student>>
parent1Rooms, List<List<Student>> parent2Rooms) {
        int numRooms = parent1Rooms.size();
        List<List<Student>> childRooms = new ArrayList<>(numRooms);

        Random random = new Random();
        int crossoverPoint = random.nextInt(numRooms);

        for (int i = 0; i < crossoverPoint; i++) {
            childRooms.add(new ArrayList<>(parent1Rooms.get(i)));
        }

        for (int i = crossoverPoint; i < numRooms; i++) {
            childRooms.add(new ArrayList<>(parent2Rooms.get(i)));
        }

        return childRooms;
    }

    private static List<List<Student>> mutate(List<List<Student>> rooms) {

        int roomIndex1 = (int) (Math.random() * rooms.size());
        int roomIndex2 = (int) (Math.random() * rooms.size());

        while (roomIndex1 == roomIndex2) {
            roomIndex2 = (int) (Math.random() * rooms.size());
        }

        List<Student> room1 = rooms.get(roomIndex1);
        List<Student> room2 = rooms.get(roomIndex2);

        if (room1.size() == 0 || room2.size() == 0) {
            return rooms;
        }

        int studentIndex1 = (int) (Math.random() * room1.size());
        int studentIndex2 = (int) (Math.random() * room2.size());

        Student student1 = room1.get(studentIndex1);
        Student student2 = room2.get(studentIndex2);
```

```
        room1.set(studentIndex1, student2);
        room2.set(studentIndex2, student1);

        return rooms;
    }

    private static void printRoomAssignment(List<List<Student>> rooms) {
        for (int i = 0; i < rooms.size(); i++) {
            System.out.println("Кімната " + (i + 1) + ":");
            List<Student> room = rooms.get(i);
            for (Student student : room) {
                System.out.println("Гендер: " + (student.getGender() == 0 ?
"чоловік" : "жінка") +
                        ", Курс: " + student.getCourse() +
                        ", Соціотип: " + (student.getSociotype() == 0 ?
"інтроверт" : "екстраверт"));
            }
            System.out.println();
        }
    }
}
package com.company.springboot;

class Student {
    private int gender; // 0 - men, 1 - women
    private int course; // number course
    private int sociotype; // соціотип (наприклад, 0 - інтроверт, 1 -
екстраверт)

    public Student(int gender, int course, int sociotype) {
        this.gender = gender;
        this.course = course;
        this.sociotype = sociotype;
    }

    public int getGender() {
        return gender;
    }

    public int getCourse() {
        return course;
    }

    public int getSociotype() {
        return sociotype;
    }
}
```

Backend (фрагмент):

```
package com.company.holikov.backend.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authen
ticationManagerBuilder;
```

```
import
org.springframework.security.config.annotation.method.configuration.EnableGlo
balMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityC
onfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
        prePostEnabled = true,
        securedEnabled = true
       )
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception
{
        return super.authenticationManagerBean();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(8);
    }

    private final UserDetailsService userDetailsService;

    @Autowired
    public WebSecurityConfig(@Qualifier("userDetailsServiceImpl")
UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Override
    public void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {

authenticationManagerBuilder.authenticationProvider(daoAuthenticationProvider
());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .cors().and().csrf().disable()
                .authorizeRequests()
                .antMatchers("/auth/*").permitAll()
                .antMatchers("/api/*").permitAll()
                .anyRequest().authenticated();

    }
```

```
    @Bean
    protected DaoAuthenticationProvider daoAuthenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new
DaoAuthenticationProvider();
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());

        daoAuthenticationProvider.setUserDetailsService(userDetailsService);
        return daoAuthenticationProvider;
    }
}
package com.company.holikov.backend.controller;

import com.company.holikov.backend.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "http://localhost:4200")
public class AdminController {

    @Autowired
    private StudentRepository studentRepository;


    @GetMapping(value = "/get-all-students", produces =
MediaType.APPLICATION_JSON_VALUE)
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<?> getAllStudents(){

        System.out.println("getAllStudents +_+_+_++_+_+_+_+_+");
        System.out.println(studentRepository.findAll());

        return ResponseEntity.ok(studentRepository.findAll());
    }
}
package com.company.holikov.backend.controller;

import com.company.holikov.backend.model.Role;
import com.company.holikov.backend.model.student.Profile;
import com.company.holikov.backend.model.student.Student;
import com.company.holikov.backend.model.university.Department;
import com.company.holikov.backend.model.university.Group;
import com.company.holikov.backend.model.university.Speciality;
import com.company.holikov.backend.pojo.AuthResponse;
import com.company.holikov.backend.pojo.LoginRequest;
import com.company.holikov.backend.pojo.MessageResponse;
import com.company.holikov.backend.pojo.SignupUserRequest;
import com.company.holikov.backend.repository.*;
import com.company.holikov.backend.service.ProfileService;
import com.company.holikov.backend.service.StudentService;
import com.company.holikov.backend.service.UserService;
import com.company.holikov.backend.service.impl.user.UserDetailsImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
```

```
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/auth")
@CrossOrigin(origins = "http://localhost:4200")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private StudentRepository studentRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RoleRepository roleRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private UserService userService;

    @Autowired
    private ProfileService profileService;

    @Autowired
    private GroupRepository groupRepository;

    @Autowired
    private DepartmentRepository departmentRepository;

    @Autowired
    private SpecialityRepository specialityRepository;

    @Autowired
    private StudentService studentService;

    @Autowired
    private UserDetailsService userDetailsService;

    @PostMapping(value = "/signin", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> authenticate(@RequestBody LoginRequest
loginRequest) {

        UserDetails userDetails =
userDetailsService.loadUserByUsername(loginRequest.getUsername());
        UsernamePasswordAuthenticationToken authenticationToken =
```

```
                new UsernamePasswordAuthenticationToken(userDetails,
loginRequest.getPassword(), userDetails.getAuthorities());

        authenticationManager.authenticate(authenticationToken);
        if (authenticationToken.isAuthenticated()) {

SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        }
        UserDetailsImpl userDetails1 = (UserDetailsImpl)
authenticationToken.getPrincipal();
        List<String> roles = userDetails1.getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .collect(Collectors.toList());

        System.out.println(authenticationToken.getPrincipal());

        System.out.println(roles);


System.out.println(SecurityContextHolder.getContext().getAuthentication().get
Authorities());

        return ResponseEntity.ok(new AuthResponse(
                userDetails1.getId(),
                userDetails1.getUsername(),
                userDetails1.getEmail(),
                roles));
    }

    @PostMapping("/signup-student")
    public ResponseEntity<?> registerUser(@RequestBody SignupUserRequest
signupUserRequest) {

        System.out.println(signupUserRequest);
        if (userRepository.existsByLogin(signupUserRequest.getLogin())) {
            System.out.println("true");
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Login is exist"));
        }

        if (userRepository.existsByEmail(signupUserRequest.getEmail())) {
            System.out.println("true");
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Email is exist"));
        }

        Role role =
roleRepository.findByName(signupUserRequest.getRole().toUpperCase());


        Profile profile = new Profile(
                false,
                false,
                false
        );

        profileService.create(profile);

        Group group =
groupRepository.findByNameAndNumber(signupUserRequest.getGroupName().toUpperC
```

```
ase(), signupUserRequest.getGroupNumber());

        if (group==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the group"));
        }

        System.out.println(group);
        Department department =
departmentRepository.findByName(signupUserRequest.getDepartmentName().toUpper
Case());


        if (department==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the
department"));
        }

        Speciality speciality =
specialityRepository.findByName(signupUserRequest.getSpecialityName().toUpper
Case());

        if (speciality==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the
speciality"));
        }

        Student student = new Student(
                signupUserRequest.getSurname(),
                signupUserRequest.getFirstName(),
                signupUserRequest.getLastName(),
                signupUserRequest.getAge(),
                signupUserRequest.getPhone(),
                signupUserRequest.getGender(),
                signupUserRequest.getSociotype(),
                signupUserRequest.getCourse(),
                group,
                department,
                speciality,
                profile,
                null,
                null,
                signupUserRequest.getLogin(),
                passwordEncoder.encode(signupUserRequest.getPassword()),
                signupUserRequest.getEmail(),
                role
        );

        studentService.create(student);


        return ResponseEntity.ok(new MessageResponse("User CREATED"));
    }
}
package com.company.holikov.backend.controller;

import com.company.holikov.backend.model.Role;
import com.company.holikov.backend.model.student.Profile;
```

```
import com.company.holikov.backend.model.student.Student;
import com.company.holikov.backend.model.university.Department;
import com.company.holikov.backend.model.university.Group;
import com.company.holikov.backend.model.university.Speciality;
import com.company.holikov.backend.pojo.AuthResponse;
import com.company.holikov.backend.pojo.LoginRequest;
import com.company.holikov.backend.pojo.MessageResponse;
import com.company.holikov.backend.pojo.SignupUserRequest;
import com.company.holikov.backend.repository.*;
import com.company.holikov.backend.service.ProfileService;
import com.company.holikov.backend.service.StudentService;
import com.company.holikov.backend.service.UserService;
import com.company.holikov.backend.service.impl.user.UserDetailsImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/auth")
@CrossOrigin(origins = "http://localhost:4200")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private StudentRepository studentRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RoleRepository roleRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private UserService userService;

    @Autowired
    private ProfileService profileService;

    @Autowired
    private GroupRepository groupRepository;

    @Autowired
    private DepartmentRepository departmentRepository;
```

```
    @Autowired
    private SpecialityRepository specialityRepository;

    @Autowired
    private StudentService studentService;

    @Autowired
    private UserDetailsService userDetailsService;

    @PostMapping(value = "/signin", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> authenticate(@RequestBody LoginRequest
loginRequest) {

        UserDetails userDetails =
userDetailsService.loadUserByUsername(loginRequest.getUsername());
        UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(userDetails,
loginRequest.getPassword(), userDetails.getAuthorities());

        authenticationManager.authenticate(authenticationToken);
        if (authenticationToken.isAuthenticated()) {

SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        }
        UserDetailsImpl userDetails1 = (UserDetailsImpl)
authenticationToken.getPrincipal();
        List<String> roles = userDetails1.getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .collect(Collectors.toList());

        System.out.println(authenticationToken.getPrincipal());

        System.out.println(roles);


System.out.println(SecurityContextHolder.getContext().getAuthentication().get
Authorities());

        return ResponseEntity.ok(new AuthResponse(
                userDetails1.getId(),
                userDetails1.getUsername(),
                userDetails1.getEmail(),
                roles));
    }

    @PostMapping("/signup-student")
    public ResponseEntity<?> registerUser(@RequestBody SignupUserRequest
signupUserRequest) {

        System.out.println(signupUserRequest);
        if (userRepository.existsByLogin(signupUserRequest.getLogin())) {
            System.out.println("true");
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Login is exist"));
        }

        if (userRepository.existsByEmail(signupUserRequest.getEmail())) {
            System.out.println("true");
            return ResponseEntity
                    .badRequest()
```

```
                    .body(new MessageResponse("Error: Email is exist"));
        }

        Role role =
roleRepository.findByName(signupUserRequest.getRole().toUpperCase());


        Profile profile = new Profile(
                false,
                false,
                false
        );

        profileService.create(profile);

        Group group =
groupRepository.findByNameAndNumber(signupUserRequest.getGroupName().toUpperC
ase(), signupUserRequest.getGroupNumber());

        if (group==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the group"));
        }

        System.out.println(group);
        Department department =
departmentRepository.findByName(signupUserRequest.getDepartmentName().toUpper
Case());


        if (department==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the
department"));
        }

        Speciality speciality =
specialityRepository.findByName(signupUserRequest.getSpecialityName().toUpper
Case());

        if (speciality==null){
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: not find the
speciality"));
        }

        Student student = new Student(
                signupUserRequest.getSurname(),
                signupUserRequest.getFirstName(),
                signupUserRequest.getLastName(),
                signupUserRequest.getAge(),
                signupUserRequest.getPhone(),
                signupUserRequest.getGender(),
                signupUserRequest.getSociotype(),
                signupUserRequest.getCourse(),
                group,
                department,
                speciality,
                profile,
                null,
```

```
                null,
                signupUserRequest.getLogin(),
                passwordEncoder.encode(signupUserRequest.getPassword()),
                signupUserRequest.getEmail(),
                role
        );

        studentService.create(student);


        return ResponseEntity.ok(new MessageResponse("User CREATED"));
    }
}
```

```
package com.company.holikov.backend.model.dormitory;

import javax.persistence.*;
import javax.validation.constraints.NotBlank;

@Entity(name = "dormitory")
@Table(name = "dormitory")
public class Dormitory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "number", nullable = false)
    @NotBlank
    private int number;

    @Column(name = "address", nullable = false)
    @NotBlank
    private String address;

    @Column(name = "num_rooms", nullable = false)
    @NotBlank
    private int numRooms;

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "warden")
    private Warden warden;

    public Dormitory(Long id, int number, String address, int numRooms,
Warden warden) {
        this.id = id;
        this.number = number;
        this.address = address;
        this.numRooms = numRooms;
        this.warden = warden;
    }

    public Dormitory() {

    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getNumRooms() {
        return numRooms;
    }

    public void setNumRooms(int numRooms) {
        this.numRooms = numRooms;
    }

    public Warden getWarden() {
        return warden;
    }

    public void setWarden(Warden warden) {
        this.warden = warden;
    }
}
create table if not exists warden
(
    id    integer generated always as identity primary key,
    name  varchar(50) not null,
    phone varchar(15) not null
);

create table if not exists dormitory
(
    id        integer generated always as identity primary key,
    number    integer unique not null,
    address   varchar(50)    not null,
    num_rooms integer unique not null,
    warden    integer,

    constraint dormitory_warden_fk foreign key (warden) references warden
(id)

);

create table if not exists room
(
    id         integer generated always as identity primary key,
    num_seats  integer not null,
    free_p     integer not null,
    occupied_p integer not null,
    dormitory  integer,

    constraint room_dormitory_fk foreign key (dormitory) references dormitory
(id)
```

```
);

create table if not exists profile
(
    id     integer generated always as identity primary key,
    eco    bool not null,
    deputy bool not null,
    nurse  bool not null
);

create table if not exists department
(
    id        integer generated always as identity primary key,
    full_name varchar(50) not null,
    name      varchar(50) not null
);

create table if not exists groups
(
    id         integer generated always as identity primary key,
    name       varchar(50) not null,
    number     integer     not null,
    department integer,

    constraint groups_department_fk foreign key (department) references
department (id)

);

create table if not exists speciality
(
    id         integer generated always as identity primary key,
    name       varchar(50) not null,
    number     integer     not null,
    department integer,

    constraint groups_department_fk foreign key (department) references
department (id)
);

create table if not exists role
(
    id   integer generated always as identity primary key,
    name varchar(50) not null
);

create table if not exists users
(
    id       integer generated always as identity primary key,
    login    varchar(50) unique not null,
    password varchar(100)        not null,
    email    varchar(50) unique not null,
    role integer,

    constraint users_id_fk foreign key (role) references role (id)
);

create table if not exists role_emp
(
    id   integer generated always as identity primary key,
    name VARCHAR NOT NULL
);
```

```
create table if not exists student
(
    id         bigserial primary key,
    surname    varchar(50) not null,
    first_name varchar(50) not null,
    last_name  varchar(50) not null,
    age        integer     not null,
    phone      varchar(50) not null,
    gender     varchar(50) not null,
    sociotype  varchar(50) not null,
    course     varchar(50) not null,
    groups     integer,
    department integer,
    speciality integer,
    profile    integer,
    dormitory  integer,
    room       integer,

    constraint student_groups_fk foreign key (groups) references groups (id),
    constraint student_department_fk foreign key (department) references
department (id),
    constraint student_speciality_fk foreign key (speciality) references
speciality (id),
    constraint student_profile_fk foreign key (profile) references profile
(id),
    constraint student_dormitory_fk foreign key (dormitory) references
dormitory (id),
    constraint student_room_fk foreign key (room) references room (id),
    constraint student_id_fk foreign key (id) references users (id)
);

create table if not exists employee
(
    id         bigserial primary key,
    surname    varchar(50) not null,
    first_name varchar(50) not null,
    last_name  varchar(50) not null,
    age        integer     not null,
    phone      varchar(15) not null,
    role_emp integer,

    constraint employee_id_fk foreign key (id) references users (id),
    constraint employee_role_emp_fk foreign key (role_emp) references
role_emp (id)
);

create table if not exists administrator
(
    id   bigserial primary key,
    name varchar(50) unique not null,

    constraint administrator_id_fk foreign key (id) references users (id)
);

create or replace function f(num_room integer)
    returns setof dormitory
as
'
    declare
        counter integer = 0;
    begin
        while counter < num_room
            LOOP
```

```
                counter = counter + 1;
                insert
                into room (num_seats, free_p, occupied_p)
                values (4, 4, 0)
                ON CONFLICT DO NOTHING;
            END LOOP;
    end;
' language plpgsql;

CREATE OR REPLACE FUNCTION insert_data_after_table_creation()
    RETURNS void AS
'
    BEGIN
        IF EXISTS (SELECT *
                   FROM administrator
                   WHERE id = 1) THEN
        else
            insert into warden (name, phone)
            values (''GALINA'', ''380951501940'')
            ON CONFLICT DO NOTHING;

            insert into dormitory (number, address, num_rooms, warden)
            values (6, ''Valtera 14'', 525, 1)
            ON CONFLICT DO NOTHING;

            perform f((select num_rooms
                       from dormitory
                       where id = 1));

            insert into profile (eco, deputy, nurse)
            values (false, false, false)
            ON CONFLICT DO NOTHING;
            insert into profile (eco, deputy, nurse)
            values (false, false, false)
            ON CONFLICT DO NOTHING;
            insert
            into profile (eco, deputy, nurse)
            values (false, false, false)
            ON CONFLICT DO NOTHING;

            insert into department (full_name, name)
            values (''comp sinse depart'', ''CSD'')
            ON CONFLICT DO NOTHING;

            insert into groups (name, number, department)
            values (''KY'', 51, 1)
            ON CONFLICT DO NOTHING;

            insert into speciality (name, number, department)
            values (''AUTOMATIZACION'', ''151'', 1)
            ON CONFLICT DO NOTHING;

            insert into role (name)
            values (''STUDENT'')
            ON CONFLICT DO NOTHING;
            insert into role (name)
            values (''EMPLOYEE'')
            ON CONFLICT DO NOTHING;
            insert
            into role (name)
            values (''ADMIN'')
            ON CONFLICT DO NOTHING;
```

```
            insert into role_emp (name)
            values (''NURSE'')
            ON CONFLICT DO NOTHING;
            insert into role_emp (name)
            values (''ECONOMIST'')
            ON CONFLICT DO NOTHING;
            insert into role_emp (name)
            values (''DEPUTY'')
            ON CONFLICT DO NOTHING;

            insert into users (login, password, email, role)
            values (''admin'',
''$2a$08$VViG71CeXz0AM1YepOHbnefTNhpYStIII5mVrAuOqZwDYysCc8FBe'',
''admin@gmail.com'',
                    3)
            ON CONFLICT DO NOTHING;
            insert into users (login, password, email, role)
            values (''student1'',
''$2a$08$VIk81xXPLcAP93k5y2viVOjnKfXPQuOYRIG5o.0cJjcN8T35W4VM6'',
''student1'',
                    1)
            ON CONFLICT DO NOTHING;
            insert into users (login, password, email, role)
            values (''student2'',
''$2a$08$n5bvqAmtsKzYZdZpTFA1ReAKbnxU8GJhwbHelGIf7LUlFjUmq7ZSa'',
''student2'',
                    1)
            ON CONFLICT DO NOTHING;
            insert into users (login, password, email, role)
            values (''student3'',
''$2a$08$YTOtM808TqKo5yJjH0Xo6.52xuWGuAL07hT1I2iX.fwWdicPMkggO'',
''student3'',
                    1)
            ON CONFLICT DO NOTHING;
            insert into users (login, password, email, role)
            values (''nurse'',
''$2a$08$DrgSIgvFFkjIAKnLfWncjeI0ca.eGajYjAmMmEbXOetFHsCdcTB0m'', ''nurse'',
                    2)
            ON CONFLICT DO NOTHING;

            insert into users (login, password, email, role)
            values (''economist'',
''$2a$08$S0RtXGXjP/CoRsqz0G4yX.FOoU6caiIn5TEwJtPmhJLX2FlhGZgzO'',
''economist'',
                    2)
            ON CONFLICT DO NOTHING;

            insert into users (login, password, email, role)
            values (''deputy'',
''$2a$08$HhyhOuQWuKyLpMYWbz2Ixe4/lkAAJl/HfpZdILcP.Xjbzmicqa3ve'', ''deputy'',
                    2)
            ON CONFLICT DO NOTHING;

            insert into administrator (id, name)
            values (1, ''admin'')
            ON CONFLICT DO NOTHING;

            insert into student (id, surname, first_name, last_name, age,
phone, gender, sociotype, course,
                                 groups, department, speciality, profile,
dormitory, room)
            values (2, ''Golikov'', ''Max'', ''Sergiyovich'', 21,
''380660789987'', ''MEN'', ''INTROVERT'', ''1'', 1, 1,
```

```
                    1, 1, null, null)
            ON CONFLICT DO NOTHING;
            insert into student (id, surname, first_name, last_name, age,
phone, gender, sociotype, course,
                                 groups, department, speciality, profile,
dormitory, room)
            values (3, ''2'', ''2'', ''2'', 21, ''380660123456'', ''WOMEN'',
''INTROVERT'', ''1'', 1, 1, 1, 2, null,
                    null)
            ON CONFLICT DO NOTHING;
            insert into student (id, surname, first_name, last_name, age,
phone, gender, sociotype, course,
                                 groups, department, speciality, profile,
dormitory, room)
            values (4, ''3'', ''3'', ''3'', 17, ''380666987654'', ''MEN'',
''EXTROVERT'', ''1'', 1, 1, 1, 3, null, null)
            ON CONFLICT DO NOTHING;

            insert into employee (id, surname, first_name, last_name, age,
phone, role_emp)
            values (5, ''nurse'', ''nurse'', ''nurse'', 45, ''380111987654'',
1)
            ON CONFLICT DO NOTHING;

            insert into employee (id, surname, first_name, last_name, age,
phone, role_emp)
            values (6, ''economist'', ''economist'', ''economist'', 45,
''3801119876534'', 2)
            ON CONFLICT DO NOTHING;

            insert into employee (id, surname, first_name, last_name, age,
phone, role_emp)
            values (7, ''deputy'', ''deputy'', ''deputy'', 45,
''380111987954'', 3)
            ON CONFLICT DO NOTHING;

        end if;
    END;
' LANGUAGE plpgsql;

select insert_data_after_table_creation();

select *
from users
```

```
spring.jpa.database=POSTGRESQL
spring.datasource.url=jdbc:postgresql://localhost:5432/settling
spring.datasource.username=postgres
spring.datasource.password=123
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.show-sql=true
#spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDiale
ct
spring.jpa.hibernate.ddl-auto=update
spring.sql.init.data-locations= classpath:/import.sql
spring.sql.init.mode=always

#spring.jpa.defer-datasource-initialization=true
#spring.sql.init.mode=always
```

Frontend (фрагмент):

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';

import {RegisterComponent} from './register/register.component';
import {LoginComponent} from './login/login.component';
import {HomeComponent} from "./home/home.component";


const routes: Routes = [

  {
    path: 'auth/signin',
    component: LoginComponent
  },
  {
    path: 'auth/signup-student',
    component: RegisterComponent
  },
  {
    path: 'home',
    component: HomeComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import {FormsModule} from "@angular/forms";
import { RegisterComponent } from './register/register.component';
import { RouterModule } from '@angular/router';
import {HttpClientModule} from "@angular/common/http";
import {AppRoutingModule} from "./app-routing.module";
import { HomeComponent } from './home/home.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    RegisterComponent,
    HomeComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

import {Component, OnInit} from '@angular/core';
```

```
import {TokenStorageService} from './service/token-storage.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {

  roles!: string[];
  authority!: string;

  constructor(private tokenStorage: TokenStorageService) {
  }

  ngOnInit() {
    this.roles = this.tokenStorage.getAuthorities();

    console.log(this.roles.every(toString) + ' )()()');
    if (this.roles == null) {
      console.log('it is null roles');
    }

    this.roles.every(role => {
      if (role === 'ADMIN') {
        this.authority = 'ADMIN';
        return false;
      }
      this.authority = 'USER';
      return true;
    });

  }

  logout() {
    this.tokenStorage.signOut();
    window.location.reload();
  }
}
```

```
<nav class="navbar navbar-inverse">
  <div class="container-fluid">
    <!--<div class="navbar-header">
      <a class="navbar-brand" href="#">go to start page</a>
    </div>-->
    <ul class="nav navbar-nav" routerLinkActive="active">
      <li class="nav-item"><a class="nav-link"
routerLink="home">Home</a></li>

      <li *ngIf="authority === 'ADMIN'" class="nav-item">
        <a class="nav-link" routerLink="/api/admin">Admin Board</a>
      </li>

      <li *ngIf="authority === 'ADMIN' || authority==='USER'">
        <a class="nav-link" (click)="logout()">Logout</a>
      </li>

      <li *ngIf="!authority" class="nav-item">
        <a class="nav-link" routerLink="auth/signin">Login</a>
      </li>
    </ul>

  </div>
```

```
</nav>
<div class="container">
  <router-outlet></router-outlet>
</div>

import {Component, OnInit} from '@angular/core';
import {AuthLoginInfo} from "../service/temp/login-info";
import {AuthService} from "../service/auth.service";
import {TokenStorageService} from '../service/token-storage.service';



@Component({
  selector: 'app-a',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit{
  form: any = {};
  isLoggedIn = false;
  isLoginFailed = false;
  errorMessage = '';
  roles: string[] = [];
  private loginInfo!: AuthLoginInfo;

  constructor(private authService: AuthService, private tokenStorage:
TokenStorageService) {}


  ngOnInit(): void {
    if(this.tokenStorage.getAuthorities() === null){
      this.isLoggedIn = false;
    } else {
      this.isLoggedIn = true;
    }

    this.roles = this.tokenStorage.getAuthorities();
  }

  onSubmit() {
    this.loginInfo = new AuthLoginInfo(
      this.form.username,
      this.form.password);

    this.authService.attemptAuth(this.loginInfo).subscribe(
      data => {
        this.tokenStorage.saveUsername(data.username);
        this.tokenStorage.saveAuthorities(data.roles);

        this.isLoginFailed = false;
        this.isLoggedIn = true;
        this.roles = this.tokenStorage.getAuthorities();
        this.reloadPage();

      },
      error => {
        this.isLoginFailed = true;
      }
    );
  }

  reloadPage() {
    window.location.reload();
```

```
  }
}

<div *ngIf="isLoggedIn; else loggedOut">
  Logged in as {{roles}}.
</div>

<ng-template #loggedOut>
  <div class="row col-sm-6" style="max-width:350px;">
    <form name="form" (ngSubmit)="f.form.valid && onSubmit()" #f="ngForm"
novalidate>
      <div class="form-group">
        <label for="username">Login</label>
        <input id="username" type="text" class="form-control" name="username"
[(ngModel)]="form.username" #username="ngModel"
               required />
        <div *ngIf="f.submitted && username.invalid">
          <div *ngIf="username.errors?.['required']">Username is
required</div>
        </div>
      </div>
      <div class="form-group">
        <label for="password">Password</label>
        <input id="password" type="password" class="form-control"
name="password" [(ngModel)]="form.password" #password="ngModel"
               required minlength="4" />
        <div *ngIf="f.submitted && password.invalid">
          <div *ngIf="password.errors?.['required']">Password is
required</div>
          <div *ngIf="password.errors?.['minlength']">Password must be at
least 4 characters</div>
        </div>
      </div>
      <div class="form-group">
        <button class="btn btn-primary">Login</button>
        <div *ngIf="f.submitted && isLoginFailed" class="alert alert-danger">
          Login failed: {{errorMessage}}
        </div>
      </div>
    </form>
    <hr />
    <p>Don't have an account?</p>
    <a href="/auth/signup" class="btn btn-success">Sign Up</a>
  </div>
</ng-template>

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FrontendApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
<app-root></app-root>
</body>
</html>
```

Додаток Д

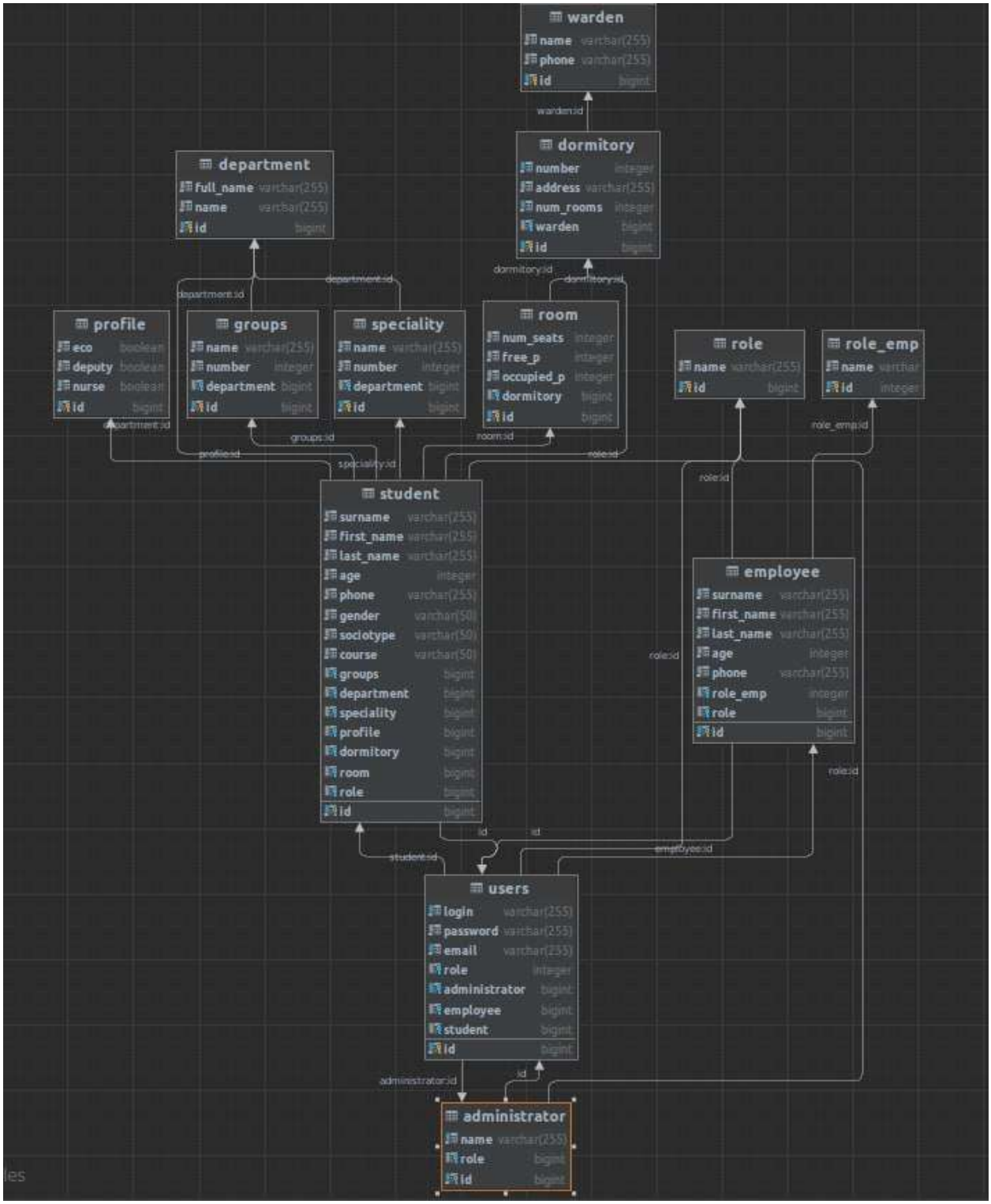

Рисунок Д.1 – Повна схема бази даних