**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ТАРАСА ШЕВЧЕНКА**
**Факультет інформаційних технологій**
**Кафедра інтелектуальних технологій**

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

Система підтримки прийняття рішень визначення оптимального алгоритму пріоритетності поселення студентів

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Комп'ютерні науки»**

Освітній рівень: бакалавр

Виконала: студент 4 курсу, групи КН- 41

Іжевська Є.В.
(прізвище та ініціали)

Керівник Гнатієнко Г.М.
(прізвище та ініціали)
канд. техн. наук
(науковий ступінь, звання)

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри *інтелектуальних технологій*
Протокол № 11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ - 2022

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій
Кафедра інтелектуальних технологій
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних технологій
Іларіонов О.Є.
_____
"___"_____2021 р.

**ЗАВДАННЯ**

**НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Іжевській Єлизаветі Володимирівні
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)
Система підтримки прийняття рішень визначення оптимального алгоритму пріоритетності поселення студентів

затверджена протоколом засідання кафедри від «23» грудня 2021 р. №4

2. Термін здачі студентом закінченого проекту (роботи) 31 травня 2022 року

3. Вихідні дані до проекту (роботи)
Система підтримки прийняття рішень визначення оптимального алгоритму пріоритетності поселення студентів створюється для визначення варіантів прийняття рішень щодо справедливого заселення студентів різних курсів у гуртожитки в умовах дефіциту місць для поселення

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
Вибір та аналіз предметної області, огляд задач розподілення обмежених ресурсів, проєктування архітектури системи, розробка системи, вибір тестових задач, візуалізація результатів, тестування роботи системи

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

Огляд джерел інформації щодо поселення студентів у гуртожитки, ілюстрація ситуації прийняття рішення, архітектура системи, програмна реалізація системи, презентація Power Point.

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| 1 | Вибір та аналіз предметної області | | |
| 2 | Постановка задачі поселення студентів у гуртожитки | | |
| 2 | Огляд підходів до забезпечення справедливого поселення студентів у гуртожитки | | |
| 3 | Проєктування архітектури системи | | |
| 3 | Програмна реалізація системи | | |
| 3 | Вибір тестових задач, тестування роботи системи | | |
| | | | |

7. Дата видачі завдання 25 січня 2022 року

Керівник _____ / /

(підпис) (ПІБ)

Завдання прийняв до виконання _____ / /

(підпис) (ПІБ)

## КАЛЕНДАРНИЙ ПЛАН

| ор. № | Назва етапів випускної кваліфікаційної роботи | Термін виконання етапів випускної кваліфікаційної роботи | Примітка |
|---|---|---|---|
| . | Вибір та аналіз предметної області | 25.01-01.02 | |

| | | | |
|---|---|---|---|
| . | Постановка задачі поселення студентів у гуртожитки | 02.02-04.02 | |
| . | Огляд підходів до забезпечення справедливого поселення студентів у гуртожитки | 05.02-14.02 | |
| . | Проєктування архітектури системи | 15.02-07.03 | |
| . | Програмна реалізація системи | 08.03-25.04 | |
| . | Вибір тестових задач, тестування роботи системи | 26.04-07.05 | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент-дипломник _____ / /

(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / /

(підпис) (ПІБ)

## АНОТАЦІЯ

**Іжевська Єлизавета Володимирівна** виконала випускну кваліфікаційну роботу на тему «Система підтримки прийняття рішень визначення оптимального алгоритму пріоритетності поселення студентів» за спеціальністю 122 – «Комп’ютерні науки».

У випускній кваліфікаційній роботі проведено аналіз підходів до забезпечення справедливого поселення студентів у гуртожитки, спроектована архітектура та виконана програмна реалізація системи.

**Ключові слова:** пріоритетність, обмежені ресурси, квоти, справедливе розподілення.

## SUMMARY

The degree project: «Decision support system for determining the optimal priority algorithm for student settlement» has been completed by **Izhevska Yelyzaveta** speciality 122 – «Computer Science».

In this graduation the analysis of approaches to ensuring fair settlement of students in dormitories is carried out, the architecture is designed and the software implementation of the system is executed.

**Keywords**: limited resources, priority, quotas, equitable distribution.

# ЗМІСТ

# ВСТУП

Питання надання житлових місць для студентів є досить гострим в незалежності від навчального закладу. Адже обмеженість такого ресурсу як кількість місць для поселення спонукає розглядати питання оптимальності їх розподілу відповідно до заявок. Дана процедура має велике соціальне значення в умовах дефіциту житлового фонду, тому що у випадку невдоволення є ризик отримання негативного зворотного зв'язку, що може відповідно вплинути на рівень якості навчального закладу зі сторони здобувачів освіти. Тому основною задачею є мінімізація можливих ризиків, нечіткостей та непорозумінь при розподіленні ресурсів. Всі процеси та алгоритми розподілу ресурсів мають бути прозорими та однозначними, чітко зрозумілими для всіх, кому необхідна дана інформація. Вони не повинні допускати різних тлумачень та бути вичерпно задокументовані для запобігання непорозумінь та конфліктних ситуацій. Задля вирішення таких питань всі процеси повинні бути чітко зарегламентованими, щоб попередити можливі корупційні схеми та суб'єктивно обумовлені рішення.

Ключовою проблемою вирішення даного питання є справедливий розподіл наявних ресурсів. Прийняття рішення про справедливість визначення пріоритетності завдань, упорядкування альтернатив, послідовностей виконання функцій та виділення важливості об'єктів є класичною проблемою у багатьох галузях господарства. Обмеженість ресурсів є фундаментальною та критичною проблемою, яка постає перед нами у багатьох аспектах життя. Зазначена проблема може бути формалізована у класі задач розподілення обмежених ресурсів, яка є важливою проблемою для великих та малих угрупувань.

# РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ДО ЗАБЕЗПЕЧЕННЯ СПРАВЕДЛИВОГО ПОСЕЛЕННЯ СТУДЕНТІВ У ГУРТОЖИТКИ

## 1.1. Аналіз необхідних умов для поселення студентів

У якості демонстраційного прикладу для даної роботи доцільно буде розглянути процес поселення студентів до гуртожитків в Київському Національному університеті імені Тараса Григоровича Шевченка, хоча це і не є обов'язковою умовою, але подібна структура присутня і в інших вищих навчальних закладах.

Варто зазначити, що існують певні вказівки та закони, якими мають керуватися всі університети при розподілі житлових місць між студентами та абітурієнтами. Право на першочергове поселення у гуртожитки надається студентам й абітурієнтам (вступникам) відповідно до:

- Постанови КМУ від 05.04.1994 року №226 «Про поліпшення виховання, навчання, соціального захисту та матеріального забезпечення дітей-сиріт і дітей, позбавлених батьківського піклування»;
- Закону України «Про статус і соціальний захист громадян, які постраждали внаслідок Чорнобильської катастрофи»;
- Закону України «Про підвищення престижності шахтарської праці»;
- Розпорядження КМУ від 14.03.2001 року №92-р «Про заходи щодо підтримки становлення та розвитку студентської сім'ї» (якщо чоловік і дружина навчаються на денній формі Університету).

Пріоритетним правом на поселення в гуртожитки Університету (за наявності поданої заяви із відповідними документами, що підтверджують таке право) користуються студенти, аспіранти, слухачі, абітурієнти (вступники):

- інваліди;
- учасники бойових дій та їхні діти, у тому числі діти, які навчаються за денною формою навчання в університеті;
- з малозабезпечених сімей;

- з багатодітних сімей;
- напівсироти;
- котрі проживають (зареєстровані) у віддалених населених пунктах;
- з гірських населених пунктів;
- студенти, які мають високі результати успішності у навчанні.

Також важливим фактор при розподілі житлових місць є квотування. Адже розподіл та кількість виділених ліжко-місць для поселення, між факультетами/інститутами, здійснюється згідно з наказом ректора університету за погодженням з органами студентського самоврядування в межах виділеної квоти (розрахунок квоти проводиться за наявності вільних ліжко-місць в гуртожитках та поточної потреби в поселені, які пропорційно розподіляються між факультетами/інститутами) [1]. Це пов’язано з тим, що на різних факультетах навчається від 300 до 3500 студентів, відповідно кількість іногородніх студентів суттєво різниться. Але у відсотковому співвідношенні всі факультети отримують пропорційну кількість місць.

Квотування в широкому понятті - це встановлення державними органами відповідно до чинного законодавства, міжнародного права та різних міжнародних актів, угод обмежень щодо виробництва товарів, міжрегіональних товарних або фінансових операцій, експортно-імпортних поставок. Застосовується як засіб регулювання обсягів виробництва окремих видів товарів, а також регламентації зовнішньоекономічної діяльності. Фактично квотування забезпечує рівномірність розподілу обмеженої кількості ресурсу.

Також важливим критерієм при поселенні є вартість проживання, адже вона напряму залежить від умов проживання (кількість ліжко-місць в кімнаті, загальне облаштування кімнат та гуртожитку, якість необхідних для життя інфраструктур, відстань до фактичного місця отримання освіти). Ці критерії напряму впливають на рівень задоволення вищим навчальним закладом. Але представлені показники якості життя не будуть враховані при обчисленні

оптимальності поселення, адже вони напряму не впливають на розподіл ресурсу.

У зв’язку з ростом кількості іногородніх студентів, які виявляють бажання отримати освіту в конкретному навчальному закладі, проблема розподілу кількості ліжко-місць стає більш гострою. Для прикладу, у 2018 році із 2889 осіб, які надали документи на поселення, лише 1770 отримали місце в гуртожитку, це пов’язано з обмеженістю даного ресурсу, якого недостатньо для задоволення всіх потреб [2].

Саме тому важливо проаналізувати можливі варіанти покращення ситуації. Визначити, який із принципів розподілу ресурсу є більш оптимальним та задовольняє якомога значущу кількість критеріїв.

## 1.2. Аналіз підходів до забезпечення справедливого поселення студентів у гуртожитки

Проблема розподілу житлових місць може дещо різнитись деталями в залежності від університету і факультету, але можна виділити певні загальні підходи, які допоможуть представити в числовому вигляді стан та рівень потреби в поселенні і зможуть надати загальну картину ситуації. Також представлять різноманітні варіації порад щодо прийняття рішення про зміну квот або перерозподіл між факультетами або ж групами.

### 1.2.1. Математична модель підтримки прийняття рішень

Для моделювання задачі розподілення обмежених ресурсів слід побудувати математичну модель факультету. Для описання структури факультету та кількісних характеристик задачі ведемо такі позначення:

$Q^0$ – множина студентів факультету, які мають право на першочергове задоволення потреби у житлі;

$Q^i, i = 1,...,6,$ – множини іногородніх студентів відповідних курсів, які мають потреби у наданні житла;

$Q = \bigcup_{i=1}^{6} Q^i$ – множина студентів факультету, які претендують на надання житла, $Q^0 \subset Q$;

$\theta^{2021}$ - можливості університету щодо поселення студентів факультету у поточному році.

Оскільки студентам, які одержують місця за квотами, надання житла є обов'язковим, то кількість студентів, які мають право на першочергове задоволення потреб у житлі $q^0 = |Q^0|$ – переходить в обмеження.

Таким чином одержуємо багатовимірну модель розподілу обмеженого ресурсу:

$$A(x) = \leq \theta - q^0 \quad (1)$$

$$f_1(x) \to max,$$
$$f_2(x) \to max,$$
$$f_3(x) \to max, \quad (2)$$
$$f_4(x) \to max,$$
$$f_5(x) \to max,$$
$$f_6(x) \to max,$$

де $x = x_1, x_2 \dots x_6$ — аргументи функцій, які відповідають кількості студентів різних курсів, що потребують житла.

При розв'язанні багатокритеріальної задачі (1) – (2) слід враховувати пріоритетність поселення:

$$\rho_1 = \rho_5 > \rho_2 = \rho_3 > \rho_4 = \rho_6 \quad (3)$$

де $\rho_i, i = \overline{1,6}$ – нормовані вагові коефіцієнти функцій (2), які задовольняють умові $\sum_{i=1} \rho_i = 1$.

Отже, маємо модель $M = f\{Q,$ де $Q \ni Q^0, Q = \bigcup_{i=1}^{6} Q^i \}$.

Відповідно на основі функцій (2) та пріоритетності поселення (3) формуються списки студентів, яким буде надано житло.

Згідно даної моделі очевидним є те, що в незалежності від курсу, пільгові категорії мають найвищий пріоритет на отримання ліжко-місць. Після чого решта місць повинна надаватись студентам перших курсів відповідних освітніх рівнів. У в останню чергу місця для поселення надаються студентам інших навчальних курсів. Застосовуючи дану модель на практиці можна зіткнутись з проблемою, коли загальної кількості ліжко-місць недостатньо, щоб задовольнити всі потреби, і студентам 2 та старших курсів не буде надано житлових місць [3].

В описаному вище випадку порушуються певні поняття справедливості. Джон Ролз в своїй праці «Теорія справедливості» описував двоє ключових принципів:

1. Кожна людина повинна мати рівні права щодо найбільш великої схеми рівних основних свобод, сумісних з подібними схемами свобод для інших.
2. Соціальні та економічні нерівності мають бути влаштовані так, щоб від них можна було б розумно очікувати на переваги для всіх, і доступ до положень (positions) і посад був відкритий всім [4].

Відповідно, якщо ми не закріплюємо якийсь відсоток (квоту) за кожною множиною, то ми не надаємо очікувані переваги та обмежуємо можливості. Адже для будь-якого $Q^i$ , де $i > 1$, позитивний або негативний результат визначається в залежності від рейтингових списків, де для отримання певного місця необхідний певний ресурсний вклад зі сторони студента. Тобто, якщо буде очевидним неможливість отримання житлового місця, то це може негативно вплинути рівень задоволення системою, процесами та навчальною структурою в загальному.

### 1.2.2. Існуючі рішення

Для прикладу можна розглянути умови поселення іногородніх студентів для двох факультетів: «Інститут права» та факультет інформаційних технологій (далі ФІТ).

Критерії поселення іногородніх студентів ФІТ у гуртожитки:

1. Першочергово забезпечувати ліжко-місцями іногородніх студентів пільгових категорій всіх курсів ОР «Бакалавр», ОР «Магістр».

2. Розподіляти вільні ліжко-місця для поселення студентів 1-го курсу ОР «Бакалавр» за вступним рейтинговим балом.

3. Розподіляти вільні ліжко-місця для поселення студентів 2-го та 3-го курсів ОР «Бакалавр» за академічним балом за останній рік навчання з об’єднаного рейтингового списку.

4. Виділені ліжко-місця розподілити між категоріями студентів за такою пропорцією:

- пільгові категорії ОР «Бакалавр», ОР «Магістр» та 1-й курс ОР «Бакалавр» – 85% від наданих ліжко-місць.
- 2-й та 3-й курси ОР «Бакалавр» – 15% від наданих ліжко-місць [5].

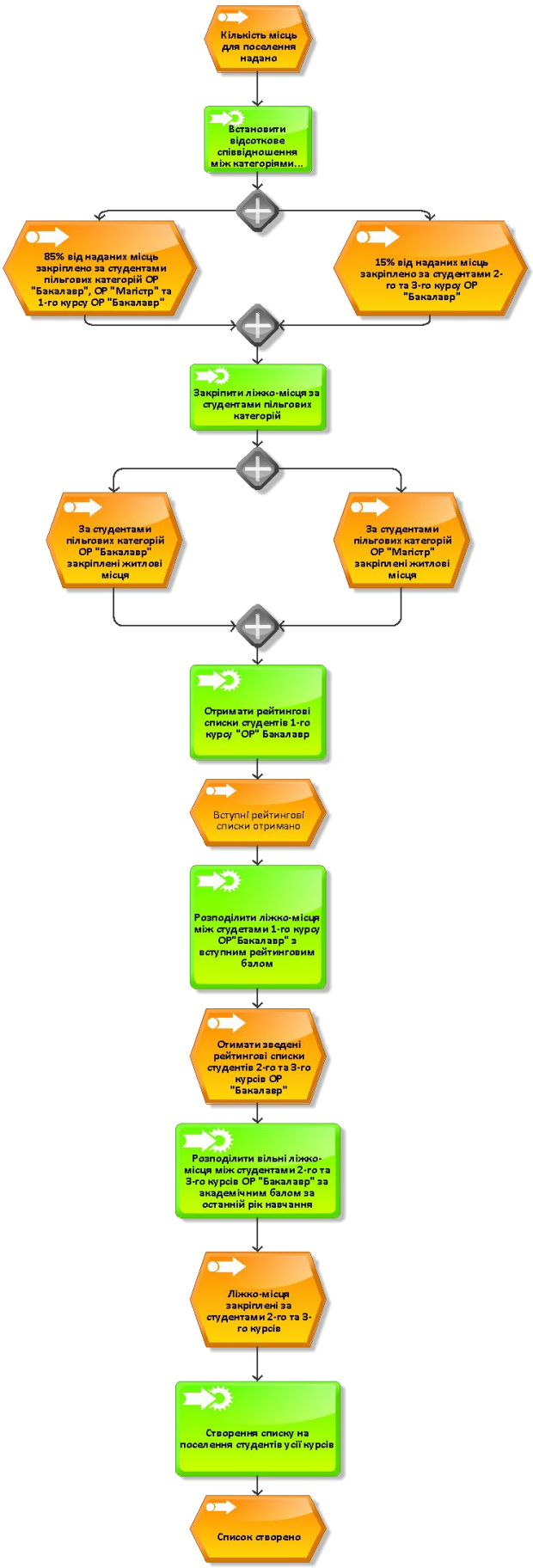
Кількість місць для поселення надано
Встановити відсоткове співвідношення між категоріями...
85% від наданих місць закріплено за студентами пільгових категорій ОР "Бакалавр", ОР "Магістр" та 1-го курсу ОР "Бакалавр"
15% від наданих місць закріплено за студентами 2-го та 3-го курсу ОР "Бакалавр"
Закріпити ліжко-місця за студентами пільгових категорій
За студентами пільгових категорій ОР "Бакалавр" закріплені житлові місця
За студентами пільгових категорій ОР "Магістр" закріплені житлові місця
Отримати рейтингові списки студентів 1-го курсу "ОР" Бакалавр
Вступні рейтингові списки отримано
Розподілити ліжко-місця між студетами 1-го курсу ОР"Бакалавр" з вступним рейтинговим балом
Отимати зведені рейтингові списки студентів 2-го та 3-го курсів ОР "Бакалавр"
Розподілити вільні ліжко-місця між студентами 2-го та 3-го курсів ОР "Бакалавр" за академічним балом за останній рік навчання
Ліжко-місця закріплені за студентами 2-го та 3-го курсів
Створення списку на поселення студентів усіх курсів
Список створено

Рисунок 1.1 – Схема алгоритму розподілу житлових місць для студентів факультету інформаційних технологій.

Критерії поселення іногородніх студентів Інституту права у гуртожитки:

1. Закріпити для поселення за студентами магістратури Інституту права 10% від загальної кількості вільних ліжко-місць, виділених Інституту права згідно з квотою на Інститут права на відповідний навчальний рік.

2. Затвердити відсоткове співвідношення для решти (90 %) вільних ліжко-місць Інституту права, згідно з квотою на Інститут права на відповідний навчальний рік, для студентів які вступили на перший курс навчання денної форми бакалаврського освітнього рівня у наступному відсотковому співвідношенні: 80% для державного замовлення та 20% для студентів, які навчаються за кошти фізичних та/або юридичних осіб.

3. Здійснювати поселення студентів денної форми навчання бакалаврського освітнього рівня та магістерського освітнього рівня в межах визначеної для них квоти до гуртожитку Інституту права наступних категорій:

- студентів, що мають пільги на поселення;
- студентів, які включені в рейтинговий список, відповідно до правил черговості [6].

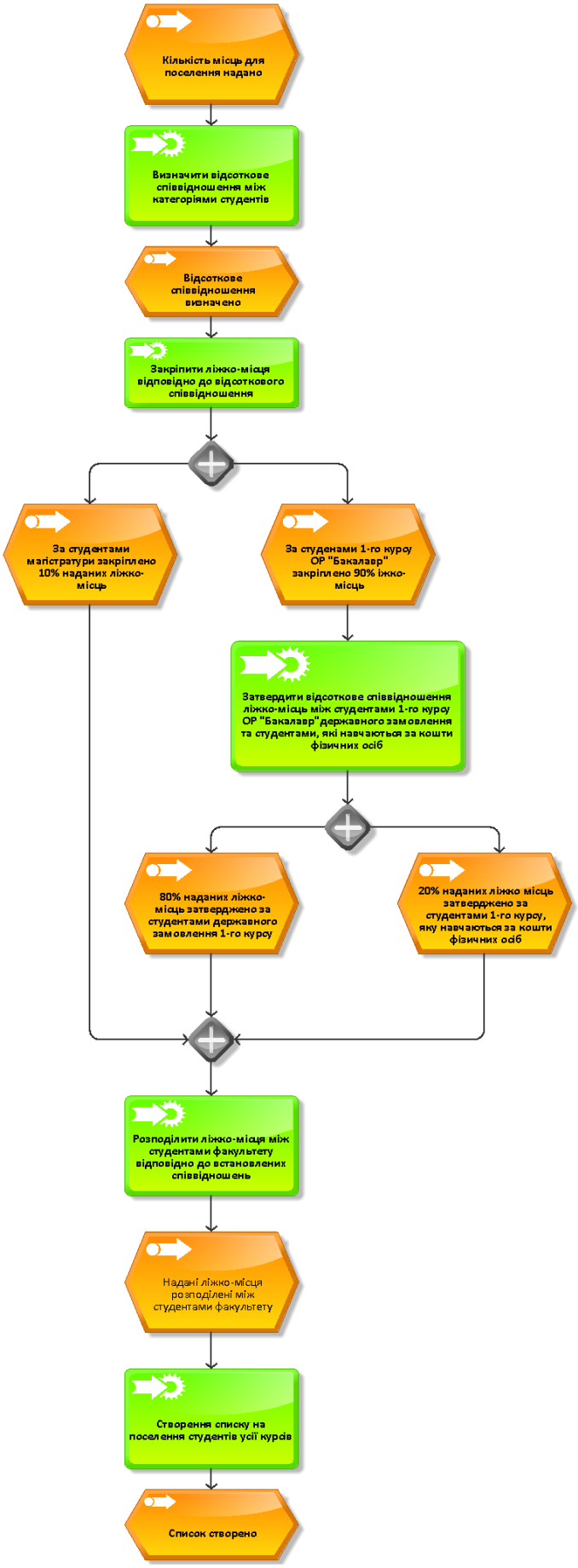
Кількість місць для поселення надано
Визначити відсоткове співвідношення між категоріями студентів
Відсоткове співвідношення визначено
Закріпити ліжко-місця відповідно до відсоткового співвідношення
За студентами магістратури закріплено 10% наданих ліжко-місць
За студенами 1-го курсу ОР "Бакалавр" закріплено 90% ліжко-місць
Затвердити відсоткове співвідношення ліжко-місць між студентами 1-го курсу ОР "Бакалавр" державного замовлення та студентами, які навчаються за кошти фізичних осіб
80% наданих ліжко-місць затверджено за студентами державного замовлення 1-го курсу
20% наданих ліжко місць затверджено за студентами 1-го курсу, яку навчаються за кошти фізичних осіб
Розподілити ліжко-місця між студентами факультету відповідно до встановлених співвідношень
Надані ліжко-місця розподілені між студентами факультету
Створення списку на поселення студентів усіх курсів
Список створено

Рисунок 1.2 – Схема алгоритму розподілу житлових місць для студентів інституту права.

У випадку наявності однакових рейтингових балів при вступі пріоритет (вище місце у черзі) мають студенти, які проживають у населених пунктах більш віддалених від м. Києва. Формування рейтингових списків щодо студентів, які навчаються на першому курсі (2 семестр) та інших курсах навчання денної форми бакалаврського освітнього рівня або першому році (2 семестр) та інших роках навчання денної форми магістерського освітнього рівня здійснюється на основі середнього балу за попередній семестр (зі 100 можливих балів). У випадку наявності однакових середніх балів за попередній семестр, пріоритет (вище місце у черзі) мають студенти, які мають вищий додатковий бал за попередній семестр (із 5 можливих балів), за наявності відповідної офіційної інформації.

У якості евристики застосуємо умову, що всі гуртожитки мають ідентичні житлові характеристики (кількість кімнат, інфраструктура і тп.).

Оглянувши дані алгоритми можна побачити явні відмінності у відсотковому співвідношенні при поділі місць, що очевидно вплине на фактичні числа. Для відображення різниці були побудовані діаграми, зображені на рисунках 3 та 4.

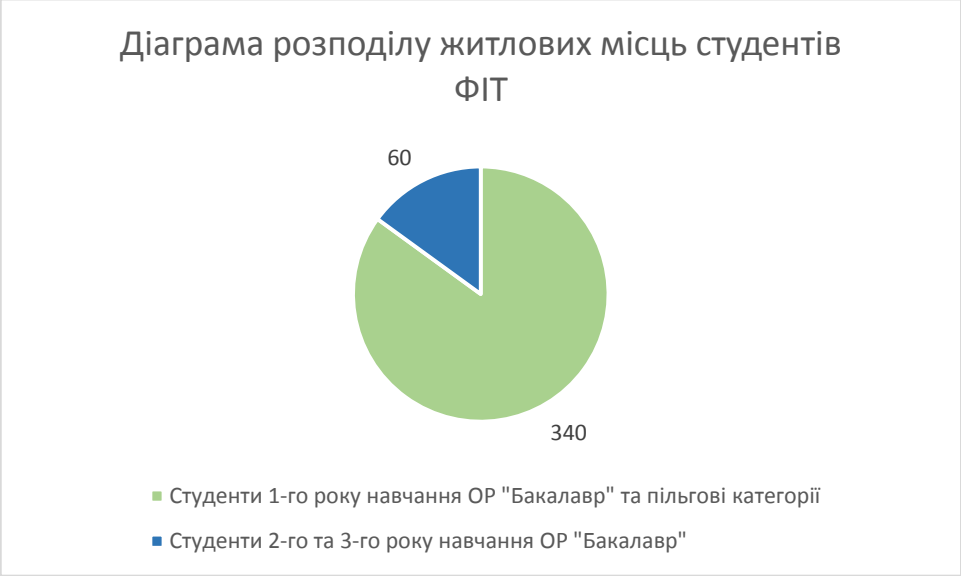


Рисунок1.3 - діаграма розподілу ліжко-місця для іногородніх студентів факультету інформаційних технологій при загальній кількості в 400 місць.

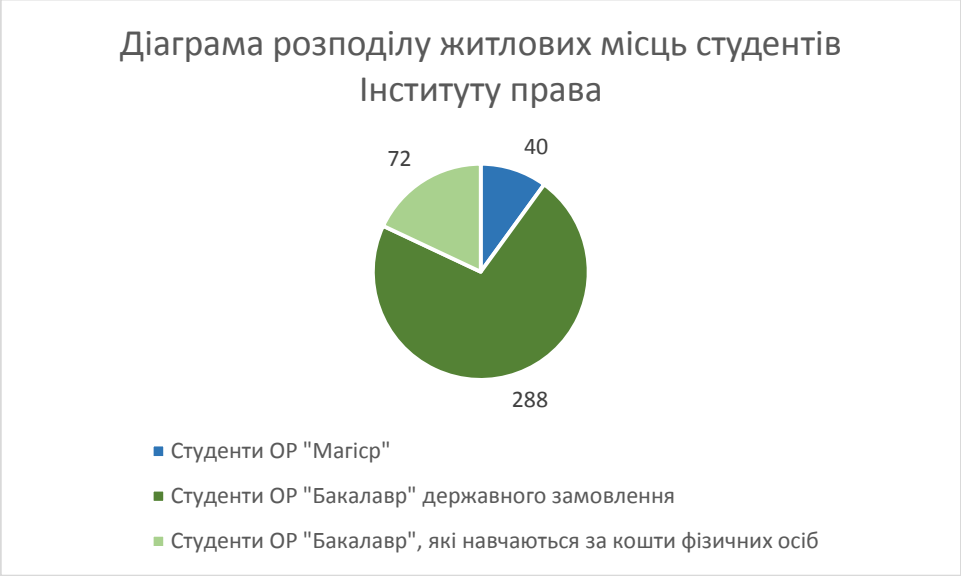


Рисунок 1.4. - Діаграма розподілу ліжко-місця для іногородніх студентів інституту права при загальній кількості в 400 місць.

При розгляді графіків краще видно різницю в фактичному розподілі місць двох факультетів, і також більш очевидним стає різниця підходів до вирішення проблеми поділу обмеженого ресурсу. Проаналізувавши дані процеси можна зазначити, що для факультету інформаційних технологій не менш важливим є поселення студентів 2-го та 3-го курсів. А в інституті права зафіксована окрема квота для студентів ОР «Магістр» та для студентів, які навчаються за кошти фізичних осіб. Варто зазначити, що реальні цифри відрізняються від представлених, адже були обрані суто для демонстрації поділу місць.

Обидва факультети проаналізували характерну для них ситуацію та прийшли до оптимального вирішення проблеми.

### 1.2.3. Рейтингова оцінка успішності

Оцінка успішності студента є важливим показником при формуванні списків на поселення. Для студентів першого року навчання ОР «Бакалавр» та ОР «Магістр» рейтинг формується на основі середнього балу за результатами ЗНО (зовнішнє незалежне оцінювання). У випадку 2-го курсу та старше може існувати декілька варіантів способів формування списків. У випадку двох факультетів, які були розглянуті вище, в першому обирався середній бал за останній рік навчання, а в другому – середній бал за останні півроку навчання. На мою думку, перший варіант надає кращий результат, адже включає в себе

більшу вибірку балів, отриманих студентом, хоча для більш точної оцінки необхідно дослідити дані підходи на реальних даних. Також важливим є те, що в залежності від спеціальності може різнитись кількість предметів, яка прямо впливає на середній бал.

З огляду на сукупність всіх вище описаних факторів, можна сказати, що проблема розподілу житлових місць потребує особливого підходу в залежності від ситуації, хоча й має спільні критерії. Для розв’язання даної ситуації доцільним є розробка системи підтримки прийняття рішень, яка дозволить спростити розрахунок всіх критеріїв та автоматизує формування списків. Також дана система надасть можливість детальніше проаналізувати можливі підходи та обрати більш оптимальний.

### 1.3. Постановка задачі розподілу житло-місць при поселенні студентів у гуртожитки

Маємо, що певну організаційну систему - університет, з відомою структурою, який має потребу в ресурсах з обмеженою кількістю – місця для поселення іногородніх студентів, які розподіляються на більш високому рівні управління. Споживачі ресурсу (факультети університету) пов’язані між собою і можуть бути запропоновані різні підходи до моделювання та розв’язання задачі. Даній організаційній системі підпорядковується декілька підсистем, які характеризуються різним статусом і, відповідно, різними права на частину ресурсів, який виділяється для певного рівня задоволення потреб організації. А саме, різна загальна кількість студентів в залежності від факультету, та відповідно кількість іногородніх студентів.

Тоді метою проекту є створення веб-застосунку для забезпечення користувача можливістю керувати даними щодо поселення студентів, їх форматування та додавання для створення списків на поселення та оцінки більш ефективних опцій.

Об’єктом дослідження роботи є структурний аналіз процесу розподілу обмеженого ресурсу (житлових місць) між студентами факультету.

Предметом дослідження цієї роботи є методи розподілу обмежених ресурсів в системі університету.

Необхідно розподілити обмежений ресурс в залежності від потреб користувачів. Врахуємо, що значення показників, які характеризують організаційну систему, є невідомими і динамічно змінними в деяких допустимих межах.

Отже, для досягнення мети необхідно вирішити такі завдання:

- змоделювати систему;
- визначити алгоритм розподілу житлових місць;
- написати програмний код, який буде генерувати результат в залежності від отриманих даних;

В результаті буде отримана система для розподілу житлових місць між студентами.

Функціональні вимоги:

- наявність інформації про алгоритм розподілу місць

Користувач повинен мати змогу побачити текстовий та графічний опис алгоритму, який формує списки студентів на поселення та додає студентів в чергу.

- додавання списку студентів

Користувач повинен мати змогу додати файл із списком студенів, на основі яких має розрахуватись список на поселення. Даний файл має містити інформацію про кожного студента щодо наявності пільг, значення рейтингового балу та чи є даний студент іногороднім. Також має міститись інформація щодо особистих даних студента (ПІБ) та рік навчання.

- додавання студента

Користувач повинен мати змогу додати студента окремо до черги на поселення. Має міститись форма для внесення всіх необхідних даних про студента (ПІБ, пільги, чи є іногороднім, рік навчання, рейтинговий бал), яку користувач повинен заповнити для додавання студента.

- додавання даних про наявні місця

Користувач повинен мати змогу додати дані щодо кількості місць в гуртожитках. Додавання інформації повинно бути можливе як за допомогою файлу так і за допомогою форми у застосунку.

- визначення розподілу місць

Користувач повинен мати змогу задати кількість місць, яку можуть отримати студенти відповідного року навчання.

- формування списків на поселення

Система має формувати список студентів на поселення на основі отриманих даних про студентів.

- формування черги на поселення

Система повинна формувати список-чергу студентів на поселення на основі решти студентів, які не потрапили в список на поселення.

- завантаження списку на поселення

Користувач повинен мати змогу завантажити файл зі сформованим списком студентів на поселення із застосунку.

- завантаження черги на поселення

Користувач повинен мати змогу завантажити файл зі сформованим списком черги студентів на поселення із застосунку.

- відображення історії дій користувача

Користувач повинен мати змогу побачити історію своїх дій в застосунку. У відповідність до кожної дії повинен бути зазначений час її виконання.

Нефункціональні вимоги:

- зовнішній інтерфейс

Даний застосунок повинен мати зовнішній інтерфейс керування системою. Цей інтерфейс повинен бути зручним в експлуатації та відповідати сучасним вимогам щодо розробки веб-застосунків.

- підтримка різних браузерів

Користувач повинен мати можливість відкрити веб-застосунок у різних браузерах. Даний функціонал дозволяє задовольнити різних користувачів, які звикли використовувати різні браузери у повсякденному житті.

Завдяки аналітичному огляду розглянутої предметної області були визначені мета та поставлене завдання випускної кваліфікаційної роботи. Після чого можна переходити до функціонального та бізнес-аналізів, а також розробки архітектури системи.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ РОЗПОДІЛУ ЖИТЛОВИХ МІСЦЬ

## 2.1. Проектування архітектури системи

### 2.1.1. IDEF0 системи розподілу житлових місць

Використаємо для подальшого моделювання методологію IDEF0 . IDEF0 застосовується для представлення функціональних меж системи, основні елементи якої: діяльність (або процес), вхідні та вихідні дані, обмеження або елементи керування, та механізми, що здійснюють активність [7]. Характерною рисою даної нотації є декомпозиція процесів. Тому, моделі IDEF0 чітко визначені, добре структуровані, легкі для розуміння, модифікації та використання [8]. Загальну схему роботи застосунку зображена на рис 2.1.

Для формування списків на поселення студентів СППР повинна отримати вхідні дані про наявні житлові місця, які були виділені факультету, інформацію про студентів, яким необхідні житлові місця. Також необхідно надати рейтингові списки студентів. Окрім цього в дані про студента повинна входити інформація, чи підпадає він під одну з пільгових категорій. Після аналізу отриманих даних СППР сформує списки на поселення, чергу студентів на поселення та історії дій користувача.

В даній системі елементом керування є стаття 40 Закону України «Про вищу освіту» та стаття 128 Житлового Кодексу України. Дані закони враховуються при визначенні пріоритетності поселення студентів, а саме урегулювання розподілу місць для студентів пільгових категорій та саме визначення пільгових категорій.

Користувач та інформаційна система підтримки прийняття рішень в представленій системі виступають механізмом.


Рисунок 2.1 – Діаграма системи в нотації IDEF0

Наступним кроком буде декомпозиція головних процесів діаграми IDEF0.

Для початку користувач надає необхідні дані про студента, інформацію про квоти та наявні житлові місця. Тобто користувач вносить дані в систему для подальшої обробки.

На основі отриманих в першому кроці даних про студента та обмеження можна переходити до аналізу цих да них та визначення пріоритетів. Цей процес виконується інформаційною системою підтримки прийняття рішень. Тобто відбуваються математичні розрахунки на основі яких формується подальший рейтинг.

Далі на основі пріоритизованих даних на наступному етапі інформаційною системою підтримка прийняття рішень формуються списки. В результаті чого на вихід отримуємо список на поселення, список-чергу на поселення та історію дій користувача.

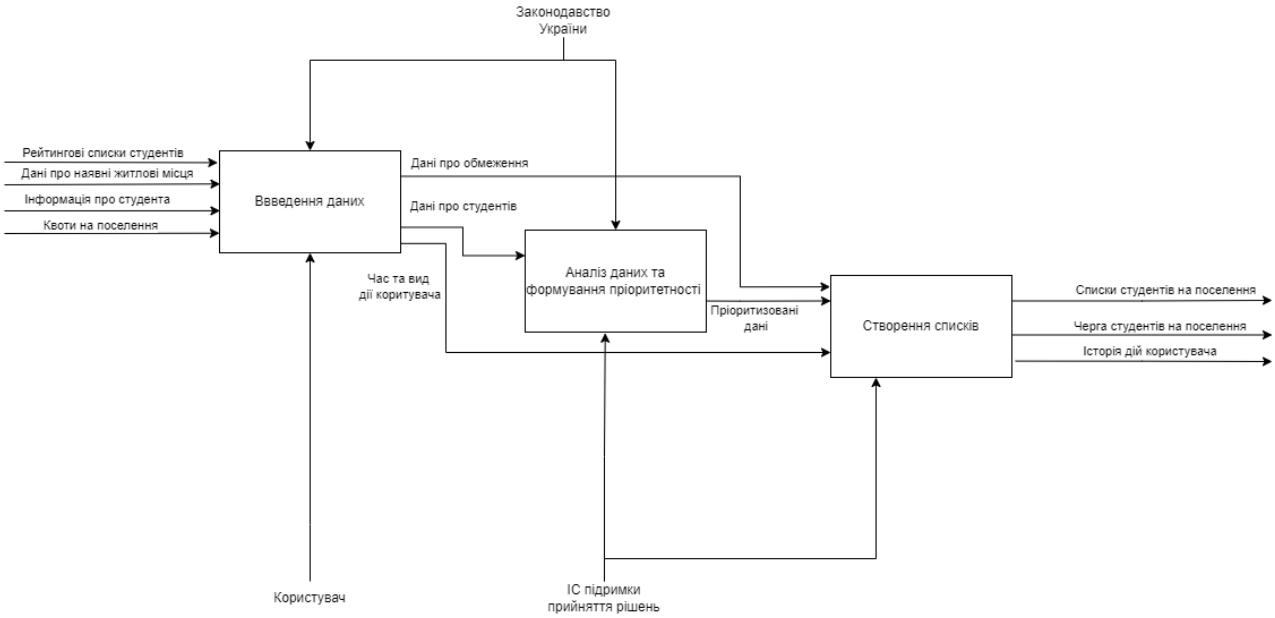


Рисунок 2.2 – Діаграма системи в нотації IDEF0. Декомпозиція процесів 1-го рівня

### 2.1.2. Функціональний аналіз

Система, що розглядається, складається з одного модуля. Для розуміння бізнес-процесів, розглянемо дерево функцій системи (рис. 2.3). Функції веб-застосунку можна умовно поділити на роботу з отриманням аналізом та передачею даних.

Функції управління даними відповідають за набір рішень, якими користувач може скористатись для внесення даних в систему, або їх зміни. Найперше це додавання студентів для розрахунку списків на поселення. Відповідно додавання списків студентів та окремо по одній особі. Наступна функція надає можливість додати студента одразу в чергу. Також у користувача є спосіб внести дані про наявне положення доступності місць для поселення.

Друга гілка дерева функцій відповідає за сферу отримання даних користувачем. В це входить візуалізація сформованих системою списків на поселення та черги, і можливість їх завантажити. Також функція отримання інформації щодо проведених операцій в системі, тобто зафіксовані дії користувача, такі як додавання студентів та формування списків. «Візуалізація алгоритму розподілу місць» це графічна та текстова інтерпретація алгоритму, який використовується при роботі системи.

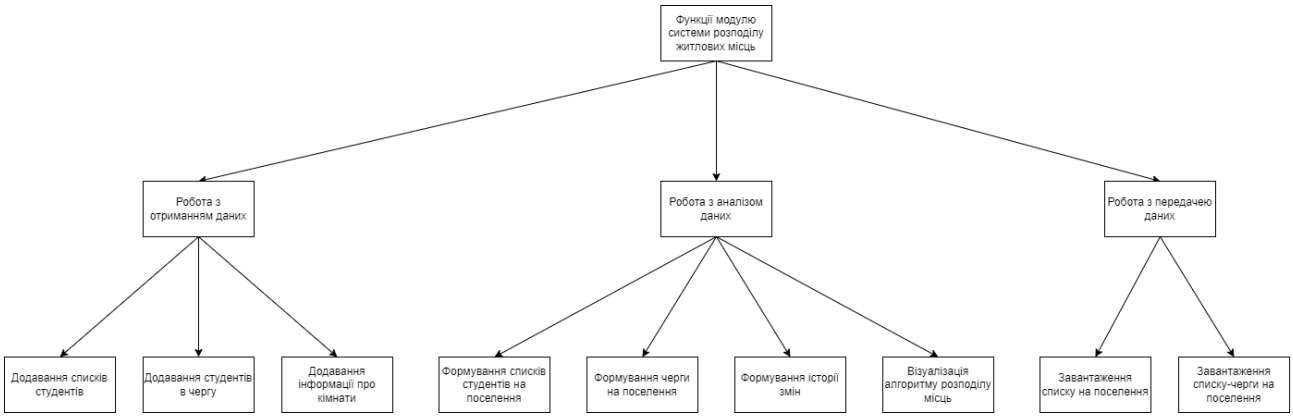


Рисунок 2.3 - Дерево функцій системи

Далі опишемо роботу користувача з функціоналом застосунку [9]. Початком роботи є відкриття додатку. Після чого користувач повинен ввести дані про студентів та інформацію про місця для поселення для подальшої ефективної роботи із застосунком. Після введення даних про місця та студенів ці дані записуються в базу даних для використання в наступних кроках. Також зберігається інформація про виконання дії користувача.

Далі користувач може ініціалізувати формування списків. Після чого користувач переадресовується на сторінку зі сформованим списком на поселення, де користувач може побачити та завантажити його. Якщо користувач хоче побачити чергу, то він може перейти на сторінку, де розташований список черги на поселення. Далі він може завантажити відображену чергу.

Іншою опцією є перехід на сторінку з історією дій користувача, де користувач може побачити інформацію про виконані їм дії та час їх виконання.

Всі розрахунки по формуванню списків та пріоритизації студентів для розподілу місць використовуються дані збережені в базі даних. Відповідно студенти додаються в чергу та на основі наявних місць визначається пріоритет для надання житло-місця. Описані процеси відображені в діаграмі Event-Driven Process Chain (рис. 2.4).

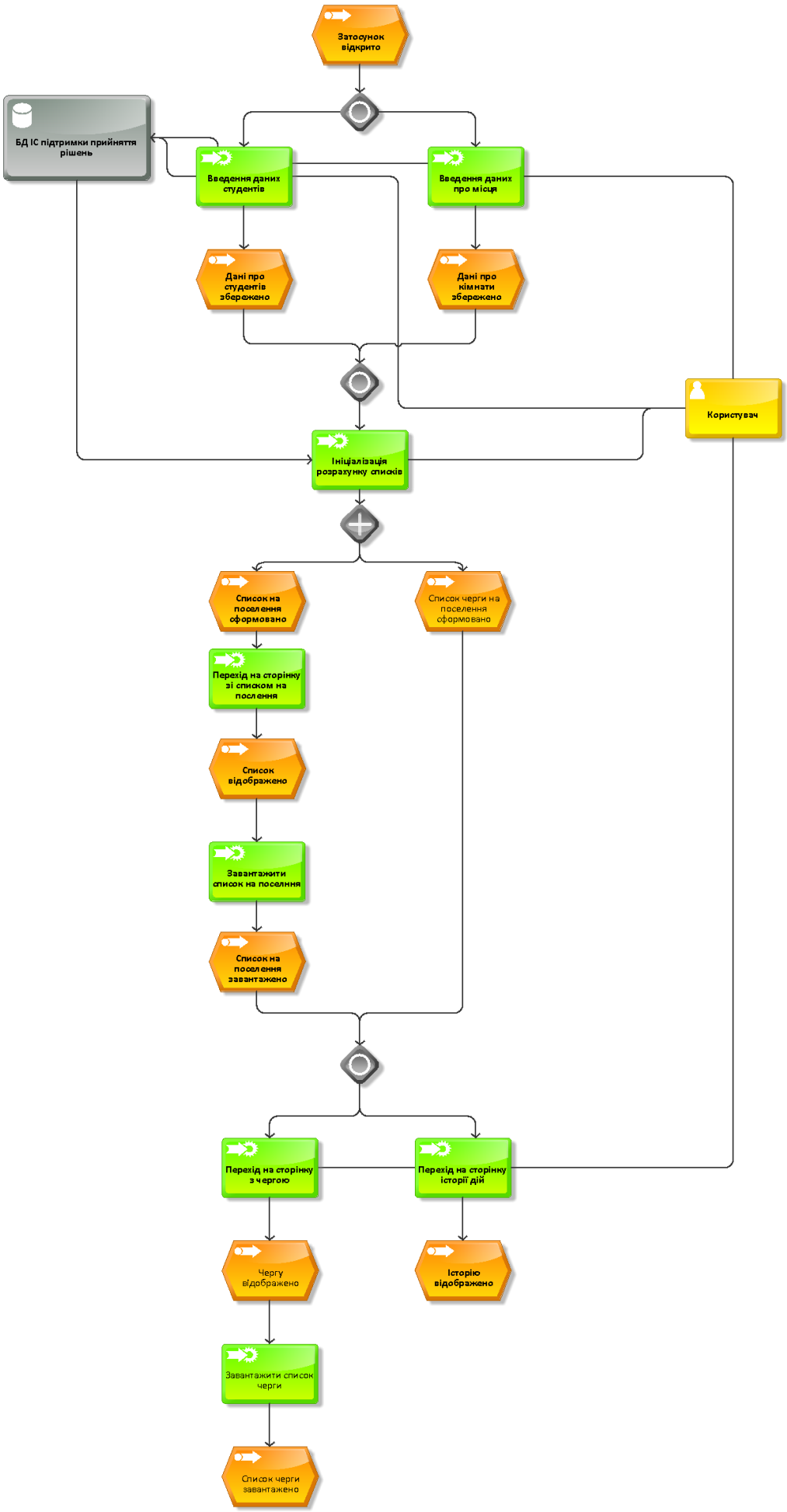
Затосунок відкрито
БД ІС підтримки прийняття рішень
Введення даних студентів
Введення даних про місця
Дані про студентів збережено
Дані про кімнати збережено
Користувач
Ініціалізація розрахунку списків
Список на поселення сформовано
Список черги на поселення сформовано
Перехід на сторінку зі списком на послення
Список відображено
Завантажити список на поселення
Список на поселення завантажено
Перехід на сторінку з чергою
Перехід на сторінку історії дій
Чергу відображено
Історію відображено
Завантажити список черги
Список черги завантажено

Рисунок 2.4. - Діаграма функціонування застосунку в нотації Event-Driven Process Chain

Опишемо процеси, які відбуваються під час використання застосунку за допомогою діаграми в нотації VAD (Value Added Chain Diagram) (рис.2.5). Ця нотація концентрується на моделюванні бізнес-процесів, які створюють цінність у вигляді послуг або продуктів. Модель процесів побудованих в нотації VAD дає загальний недеталізований погляд на процеси [10].

Найпершим процесом є «Робота із застосунком розподілу житлових місць». Цей основний процес має два підпроцеси: «Введення даних» та «Використання застосунку».

Перший з цих двох процесів має в собі ще чотири процеси, а саме: «Додавання інформації про студента», «Внесення даних про наявні місця, «Налаштування квот для відповідних років навчання» та «Додавання списку студентів». Підпроцес «Додавання інформації про студента» включає в себе чотири підпроцеси – це «Заповнення імені», «Внесення даних про пільги», «Заповнення року навчання», «Внесення рейтингового балу». Аналогічно до описного підпроцесу «Внесення даних про наявні місця» також має в собі підпроцеси: «Заповнення номерів гуртожитків», «Заповнення кількості місць в гуртожитках»

Другий процес – «Використання застосунку», аналогічно до процесу описаного в попередньому абзаці, також поділяється на наступні елементи: «Перехід на сторінку списку на поселення», який має в собі процес «Завантаження списку на поселення», «Перехід на сторінку черги» з процесом «Завантаження списку черги», «Перехід на сторінку опису алгоритму» та «Перехід на сторінку опису дій».

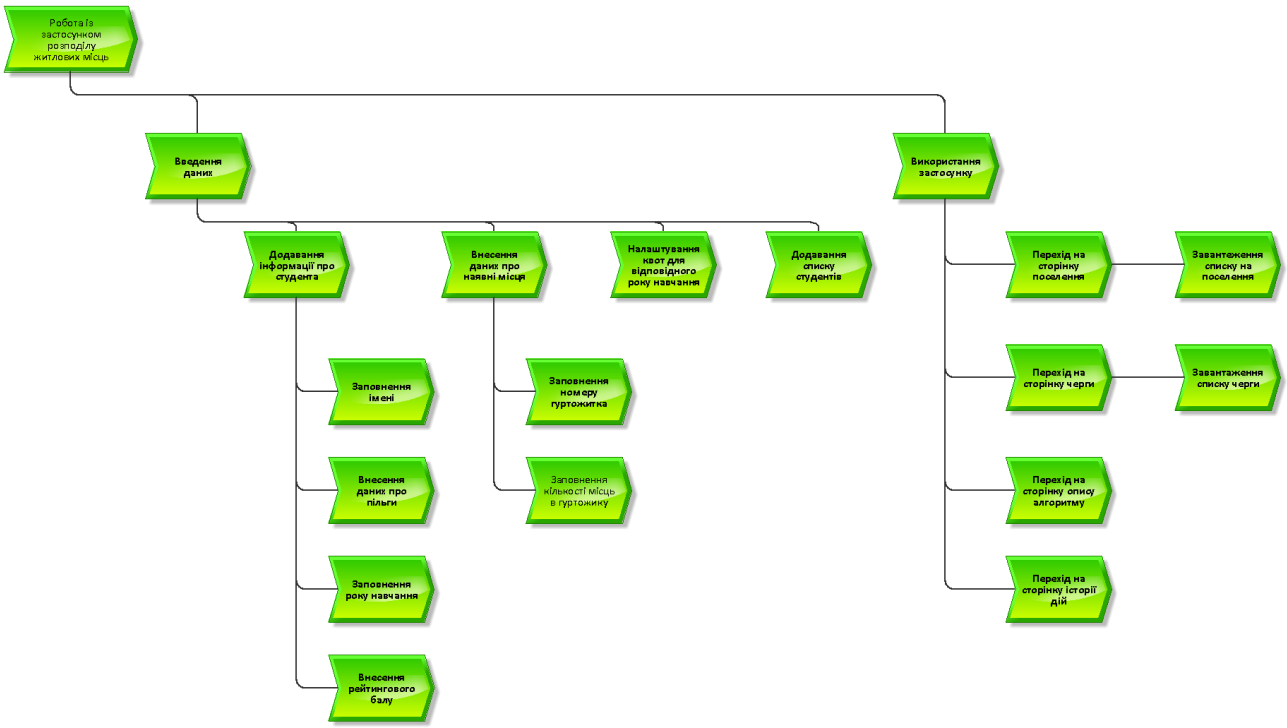


Рисунок 2.5 - Діаграма процесів в нотації VAD

### 2.1.3. Узагальнення архітектури системи підтримки прийняття рішень

Отже, наступним кроком є побудова архітектури системи (рис. 2.6), тобто організацію системи передану через її елементи та взаємодії один з одним. Представлену систему можна поділити на два модулі, які відповідають за розрахунок розподілу місць та формування історії дій користувача відповідно.

Модуль розподілу місць включає в себе опрацювання даних (дані про студентів, місці та квоти), визначення пріоритетів – ранжування студентів відповідно до наявних пільг та балів, та формування списків. Опрацьовані дані передаються для визначення пріоритетів, на основі яких формуються необхідні списки.

Другий модуль, який відповідає за формування історії дій отримує дані щодо використання користувачем застосунку. В цьому модулі фіксується системний час коли користувач виконав певну дію та її опис. Наприклад додавання списку студентів або одного студента, формування списків.

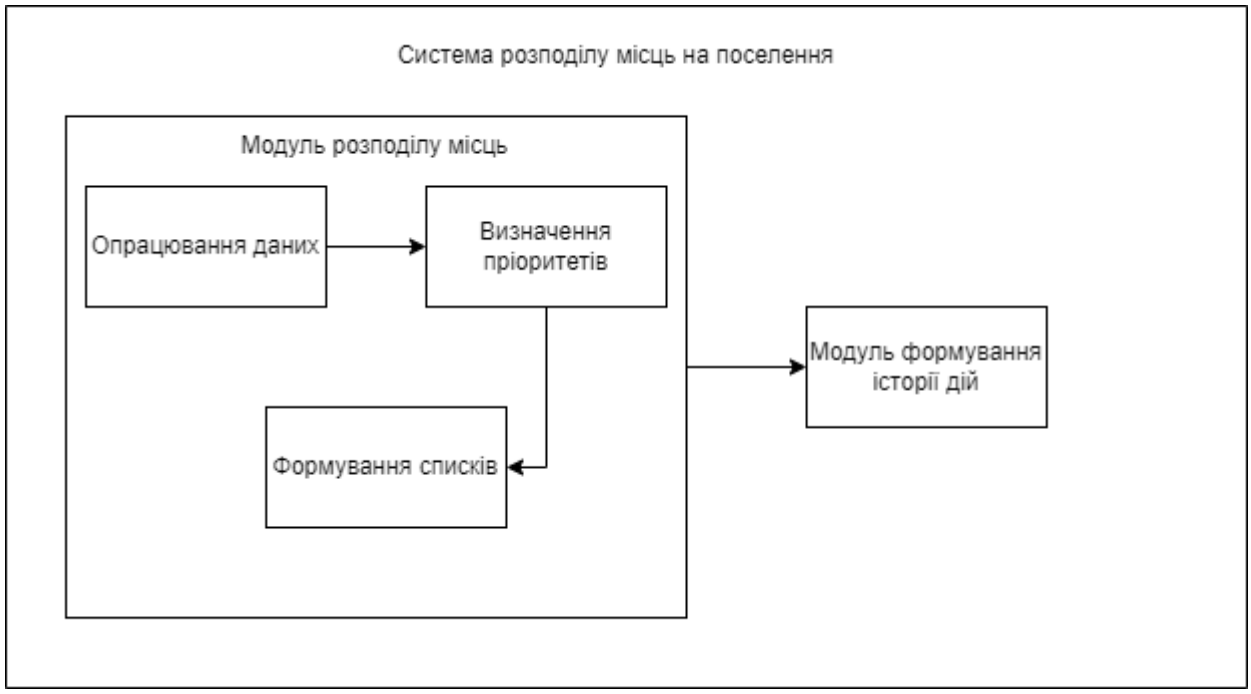


Рисунок 2.6 - Архітектура системи

## 2.2. Математичне забезпечення системі підтримки прийняття рішення, щодо поселення студентів

Перейдемо до математичного забезпечення даної системи підтримки прийняття рішень. Тобто опишемо алгоритм, який буде забезпечувати розподіл місць між усіма студентами. Що претендують на отримання місця.

Насамперед обчислюється значення пріоритету для кожного студента в черзі. Рейтинговий бал студента ділиться на 100 або 200, в залежності від року навчання, та додається до рангу кожної пільги, яку має студент, якщо такі в нього є.

$$Priority = \sum_i (mark_i + \sum_{n= i_{priv}} rank_n),$$

де $i$ – студент зі списку, $mark_i$ – рейтинговий бал студента; $n$ - пільга студента та $rank_n$ – ранг пільги.

Після обчислення пріоритетності кожного студента, отримуються дані про квоти та загальну кількість місць. Далі окремо виділяються студенти тих років навчання, для яких існують квоти. Сформовані списки студентів по кожній квоті сортуються за значенням пріоритетності від більшого до меншого.

Після чого від загальної кількості місць виділяються місця для кожного року навчання відповідно до визначеного відсотку квоти.

Отримана кількість місць надається студентам у відповідних списках, вони об’єднуються в один та повертаються як список на поселення. Всі розподілені місця видаляються, аналогічно видаляються студенти з черги, якщо вони отримали житло-місце. Якщо місць недостатньо, то студенти, що не отримали місця, залишаються в черзі.

Якщо місць наданих кожній квоті було більше, ніж претендентів, то ці місця розподіляються між рештою студентів, що не підлягають під квоту. Список цих студенті також сортується за пріоритетністю, за спаданням, та додається до загального списку на поселення.

В результаті користувач отримує список студентів, які отримали житлові місця.

## 2.3. Інформаційне забезпечення системи

### 2.3.1. Аналіз інформаційних ресурсів

Для оптимального аналізу потоку інформаційних ресурсів скористаємось діаграмою типу Data Flow Diagram (DFD). Дані діаграми відображають потік інформації для будь-якого процесу та системи. Вони використовуються для аналізу існуючих систем та моделювання нових. DFD характеризуються поділом на декілька рівнів. В даній роботі розглянута система на рівнях 0 та 1 [11].

Рівень DFD 0 також називається контекстною діаграмою. Фактично є базовим оглядом всієї системи або процесу, який аналізується або моделюється. DFD рівню 1 надають більш детальний поділ частин діаграм рівню контексту. Виділяються основні функції виконуваної системи, завдяки розбиттю високорівневих процесів контекстної діаграми на його підпроцеси.

Першим етапом є контекстна діаграма (рис. 2.7) з загальною конструкцією взаємодії між сутностями. Система підтримки прийняття рішень визначення оптимального алгоритму пріоритетності поселення студентів взаємодіє з сутністю «Користувач», який надає необхідну інформацію системі

для опрацювання і також отримує дані від системи на основі переданих. Тобто під час роботи з системою користувач надає дані про студентів, а саме списки студентів, або студента окремо з інформацією про їх ім’я та прізвище, рік навчання, наявні пільги та рейтинговий бал. Окрім цього він передає дані про номери гуртожитків з доступними місцями та саму кількість місць; та передає дані про квоти, тобто обмеження щодо розподілу місць для студентів певних років навчання. Система передає користувачу сформовані списки студентів на поселення, список черги на поселення та історію дій користувача.

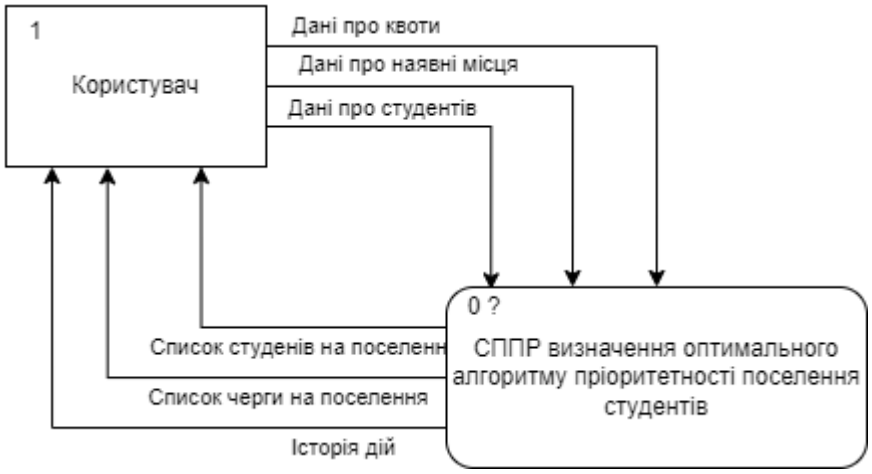


Рисунок 2.6 - Data Flow Diagram інформаційної системи 0-го рівня

Отриману діаграму можна спростити, щоб отримати більш детальний вигляд процесів системи та їх взаємодію між собою. Спершу всі дані від користувача приходять в систему та першочергово записуються до бази даних. Далі дані про студентів з бази даних передаються на наступний етап, де визначається пріоритетність студентів. Ця інформаціє передається в базу даних, де таким чином оновлюються дані студентів, а також передається на наступний етап – формування списку черги. На цьому етапі дані про студентів сортуються відповідно до визначених пріоритетів. Після чого вони передаються користувачу на наступний етап, на якому вже створюється список на поселення, для цього процесу окрім списку черги також отримуються дані з бази даних про наявні місця та визначені квоти. Отриманий список передається користувачу. Також після виконання цього етапу інформація про завершення дії записується до бази даних та передається на наступний крок – формування

історії дій. В результаті виконання процесу користувач отримує історію власних дій.

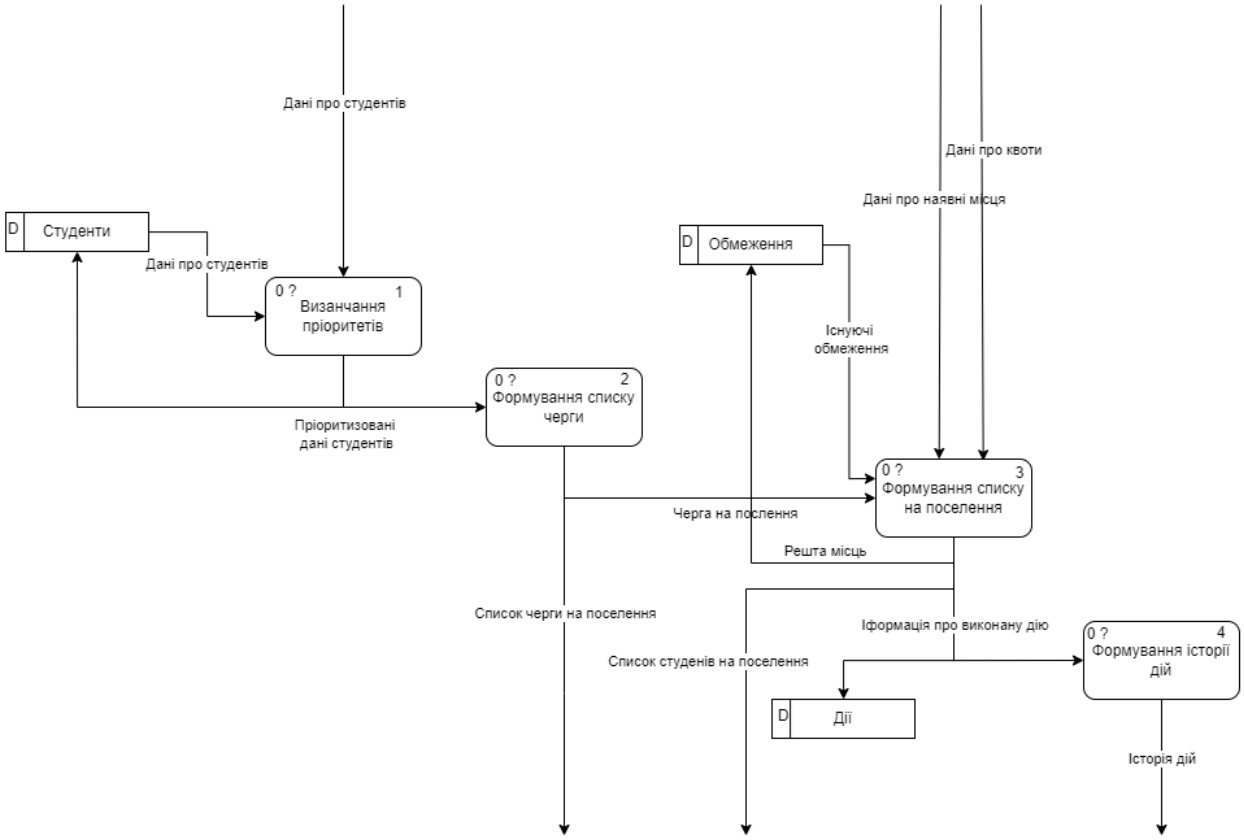


Рисунок 2.8 - Data Flow Diagram системи 1-го рівня

2.3.2. Розробка інформаційної бази системи

На основі вище описаного аналізу потоків даних буде створена база даних, в якій зберігатиметься необхідна інформація та звідки вона буде отримуватись за потребою [12].

Для початку необхідно розробити концептуальну модель бази даних (рис. 2.9) – це діаграма на якій зображуються зв’язки між об’єктами та їх характеристиками. У випадку системи, що розробляється, на діаграмі зображені наступні сутності:

- Студент;
- Гуртожиток;
- Пільга;
- Дія;
- Квота.

Зазначені вище сутності маються власні атрибути, які їх характеризують – це відповідні поля таблиць, що будуть створені. Відповідно сутність «Студент» має атрибути ПІБ, рік навчання, бал та пільга. Сутність «Гуртожиток» містить номер гуртожитка та кількість доступних місць в гуртожитку. Також є сутність «Користувач», яка описує дії користувача, тобто введення даних студента, гуртожитка, встановлення квот та виконання дій. Атрибути сутності «Пільга» - це назва та ранг. Наступна сутність, яка використовується при обчисленні списків це «Квота», вона має поля рік, факультет та відсоток, який характеризує кожен рік. «Дія» - це маніпуляції, які виконує користувач, під час використання застосунку, має в собі атрибути назва та час. Атрибут id є унікальним ідентифікатором та присутній майже у всіх сутностях. Винятками є «Дія» та «Користувач», в першому випадку унікальним ідентифікатором є атрибут час, а в другом сутність лише описує зв’язок інших сутностей між собою.

Всі зв’язки описують взаємозалежність даних між собою. Значення зв’язків наведено у наступній таблиці.

*Таблиця 2.1 Опис зв’язків між сутностями*

**Зв’язки між сутностями**

| № | Сутності, що утворюють зв’язок | Тип зв’язку | Пояснення |
|---|---|---|---|
| 1 | Користувач - Квота | Один до багатьох | Один користувач встановлює декілька квот. Певна квота може бути встановлена одним користувачем. |
| 2 | Користувач – Дія | Один до багатьох | Один користувач виконує багато дій. Одну дію виконує тільки один користувач. |
| 3 | Користувач – Студент | Один до багатьох | Користувач може ввести інформацію про багатьох студентів. Багато студентів можуть бути описані одним користувачем. |
| 4 | Гуртожиток – Студент | Один до багатьох | В одному гуртожитку може мешкати багато студентів. Але студент може жити тільки в одному гуртожитку. |

| | | | |
|---|---|---|---|
| 5 | Студент – Пільга | Багато до багатьох | Студент може мати декілька пільг. Одну й ту саму пільгу може мати багато студентів. |

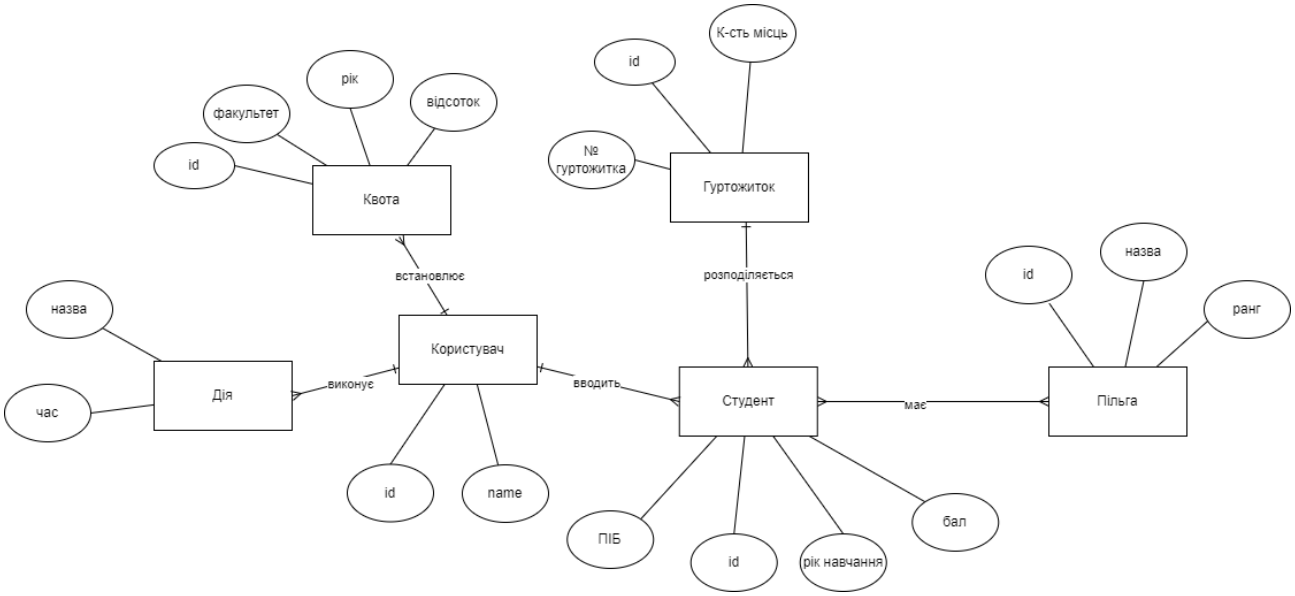


Рисунок 2.9 – Концептуальна модель бази даних

Далі представлена логічна модель бази даних (рис. 2.10). Логічна модель бази даних виділяє основні об’єкти бази даних та як і концептуальна модель визначає зв’язки між цими об’єктами, окрім цього визначаються типи даних окремих об’єктів. Також вказуються первинні ключі, тобто в даному випадку поля «id», та поле часу для дії користувача («Action») та визначаються зовнішні ключі, які описують з’язки між таблицями.

Для спрощення існуючих зв’язків багато до багатьох створюються додаткові проміжні таблиці. Відповідно необхідно розділити зв’язок між таблицями «Student» та «Privilege», реалізуємо це допомогою допоміжної таблиці «StudentPrivilege». Зв’язок між проміжною таблицею та основною є один до багатьох.

Останнім етапом є створення фізичної моделі бази даних (рис. 2.11), яка розробляється на основі даталогічної моделі. Дана діаграма відображає детальний опис даних, які зберігаються в базі даних. На цій діаграмі можна побачити тип даних, який до них застосовується та певні обмеження, для певних з них. Встановлені обмеження описані в таблиці 2.2.

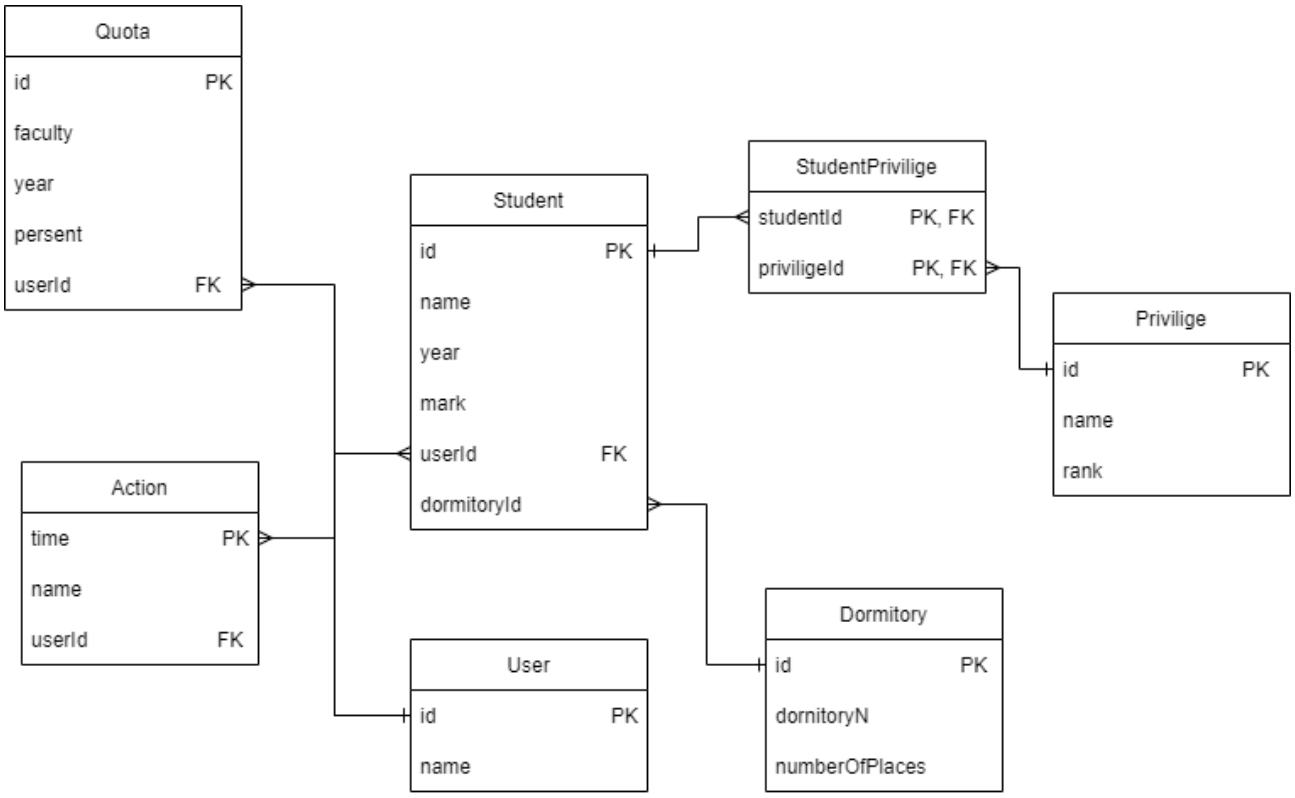


Рисунок 2.10 – Логічна модель бази даних

*Таблиця 2.2 Опис атрибутів таблиць*

**Склад та характеристика атрибутів таблиць**

| № п/п | Назва елемента даних | Тип елемента даних | Обов'язкове значення | Обмеження | Ключ |
|---|---|---|---|---|---|
| **Таблиця User «Користувач»** | | | | | |
| 1 | name | Символьний | ні | 20 | |
| 2 | id | Ціле число | так | | ПК |
| **Таблиця Action «Дія»** | | | | | |
| 1 | id | Ціле число | так | | ПК |
| 2 | name | Текст | так | | |
| 3 | userid | Ціле число | так | | ЗК |
| **Таблиця Quota «Квота»** | | | | | |
| 1 | id | Ціле число | так | | ПК |
| 2 | faculty | Символьний | так | 100 | |
| 3 | year | Ціле число | так | | |
| 4 | persent | Дійсне число | так | | |
| 4 | userId | Ціле число | так | | ЗК |

| **Таблиця Student «Студент»** |
|---|

| | | | | | |
|---|---|---|---|---|---|
| 1 | id | Ціле число | так | | ПК |
| 2 | name | Символьний | так | 50 | |
| 3 | year | Ціле число | так | | |
| 4 | mark | Дійсне число | так | | |
| 5 | userId | Ціле число | так | | ЗК |
| 6 | dormitoryId | Ціле число | так | | ЗК |
| **Таблиця Privilege «Пільга»** | | | | | |
| 1 | id | Ціле число | так | | ПК |
| 2 | name | Символьний | так | 50 | |
| 3 | rank | Ціле число | так | | |
| **Таблиця Room «Гуртожиток»** | | | | | |
| 1 | id | Ціле число | так | | ПК |
| 2 | dormitoryN | Ціле число | так | | |
| | numberOfRooms | Ціле число | так | | |
| **Таблиця StudentPrivilege** | | | | | |
| 1 | studentId | Ціле число | так | | ПК, ЗК |
| 2 | privilegeId | Ціле число | так | | ПК, ЗК |

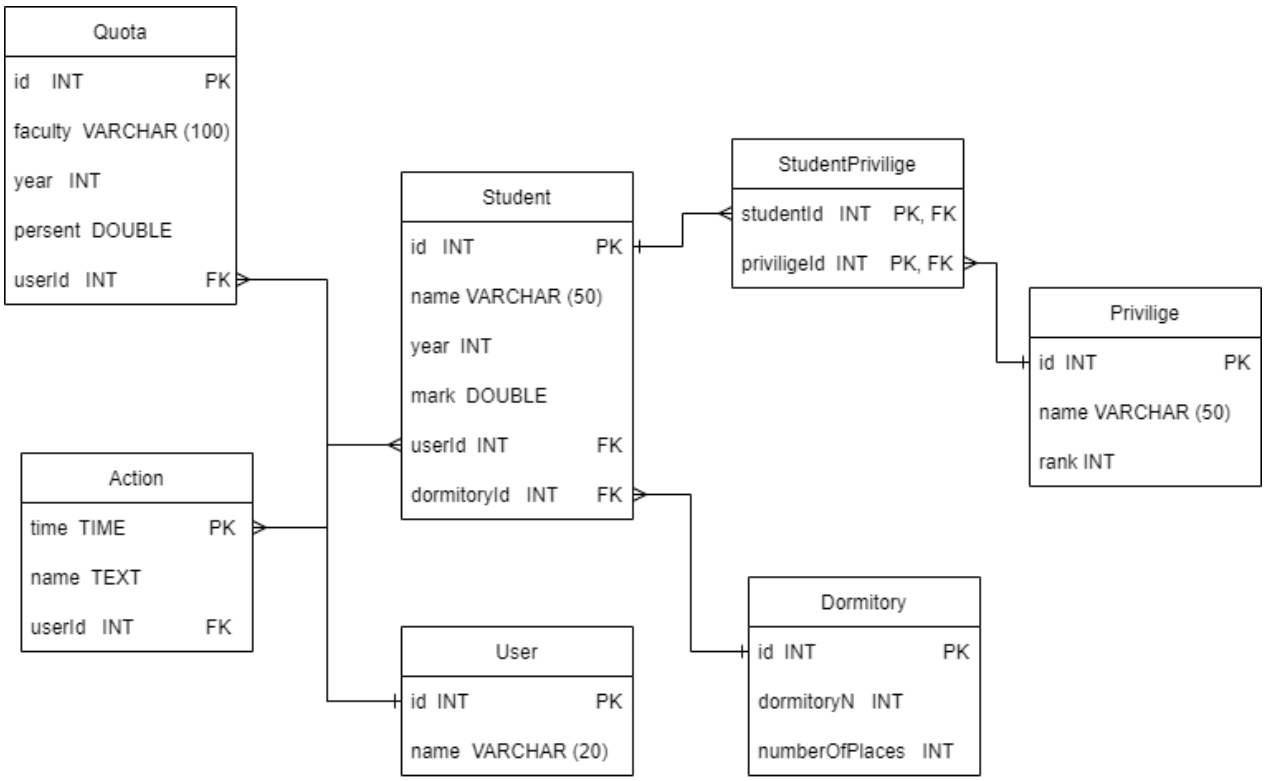


Рисунок 2.10 – Фізична модель бази даних

Після розробки архітектури системи застосунку та його інформаційної бази, можна переходити до фактичної реалізації застосунку.

# РОЗДІЛ 3. ТЕХНОЛОГІЧНІ ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ РОЗПОДІЛУ ЖИТЛОВИХ МІСЦЬ

## 3.1. Опис обраних програмних засобів

На початковому етапі програмування застосунку були обрані певні ключові програмні засоби, за допомогою яких реалізований додаток.

MongoDB – система керування базами даних, яка відноситься до документо-орієнтованих. Тобто дані зберігаються у форматі певних документів., зокрема для MongoDB, ці документи мають JSON-подібну структуру. MogoDB це збалансований вибір між швидкими, але обмеженими системами формату ключ-значенні, та реляційними базами даних [13].

До переваг MongoDB можна віднести зручний формат зберігання, гнучку мову запитів, автоматичне створення індексів, журнал змін.

Приклад документу:

```
{
   "name" : "John",
   "address" : {
     "street" : "Stepana Bandery 9",
     "city" : "Lviv",
     "state" : "Lvivska oblast"
   }
 }
```

Приклад запиту з використанням mongoose:

```
db.users.find({"address.state" : "Lvivska oblast"})
db.meal.insert({"vegetable" : ["carrot", "potato",
"tomato"]})
db.meal.delete({"vegetable" : "tomato"})
```

ExpressJS на базі NodeJS - це простий та гнучкий фреймворк розробки веб-застосунків на базі NodeJS. Він надає велику кількість функцій, що полегшують процес розробки веб-сервісів. ExpressJS використовує для роботи локально обгортку у вигляді NodeJS, фреймворка, який дозволяє запускати код JavaScript на сервері, а не лише у браузері [14].

NodeJS має багато відмінностей від звичайних мов, серед яких є:

- однопотокова асинхронна модель для виконання запитів
- неблокуючий ввід та вивід
- система модулів CommonJS
- движок JS V8 від Google

Завдяки подібні1 структурі NodeJS є найкращим рішенням для роботи над невеликими застосунками, в яких іде постійне спілкування клієнта та сервера. Завдяки одному неблокуючому потоку клієнт може зробити відразу декілька запитів і не чекати розблокування потоку для продовження роботи, а розробник не повинен займатися розбиванням задач на потоки.

React – це JavaScript-бібліотека з відкритим кодом для розробки користувацьких інтерфейсів. Одними з переваг фреймворку React є віртуальний DOM – це віртуальна копія реального DOM-дерева, завдяки ній є можливість маніпулювати та змінювати навіть найменші компоненти і лише потім порівнювати результат з реальним DOM та вносити зміни; та підтримка великої кількості сторонніх модулів (Redux, React Dev Tools. React Router і т.д.) [15].

## 3.2. Опис структури інтерфейсу веб-застосунку

Для розробки веб-застосунку необхідно розуміти основні елементи, які мають бути розташовані на сайті. Важливим є відображення всього необхідного для користувача функціоналу. Задля цього для початку необхідно розробити макет застосунку (рис. 3.1 – 3.4). Інтерфейс складається з наступних модулів:

- назва (присутня на кожній сторінці)
- меню (присутнє на кожній сторінці)
- сторінка для внесення даних
    - форма для додавання інформації про студентів
    - форма для додавання інформації про місця
    - форма для додавання інформації про квоти
    - кнопка розрахунку списків

- сторінка зі списком на поселення та зі списком черги (однакова структура)
    - список
    - кнопки (завантаження, видалення зі списку)
- сторінка з історією
    - історія дій користувача
- сторінка з алгоритмом
    - опис алгоритму

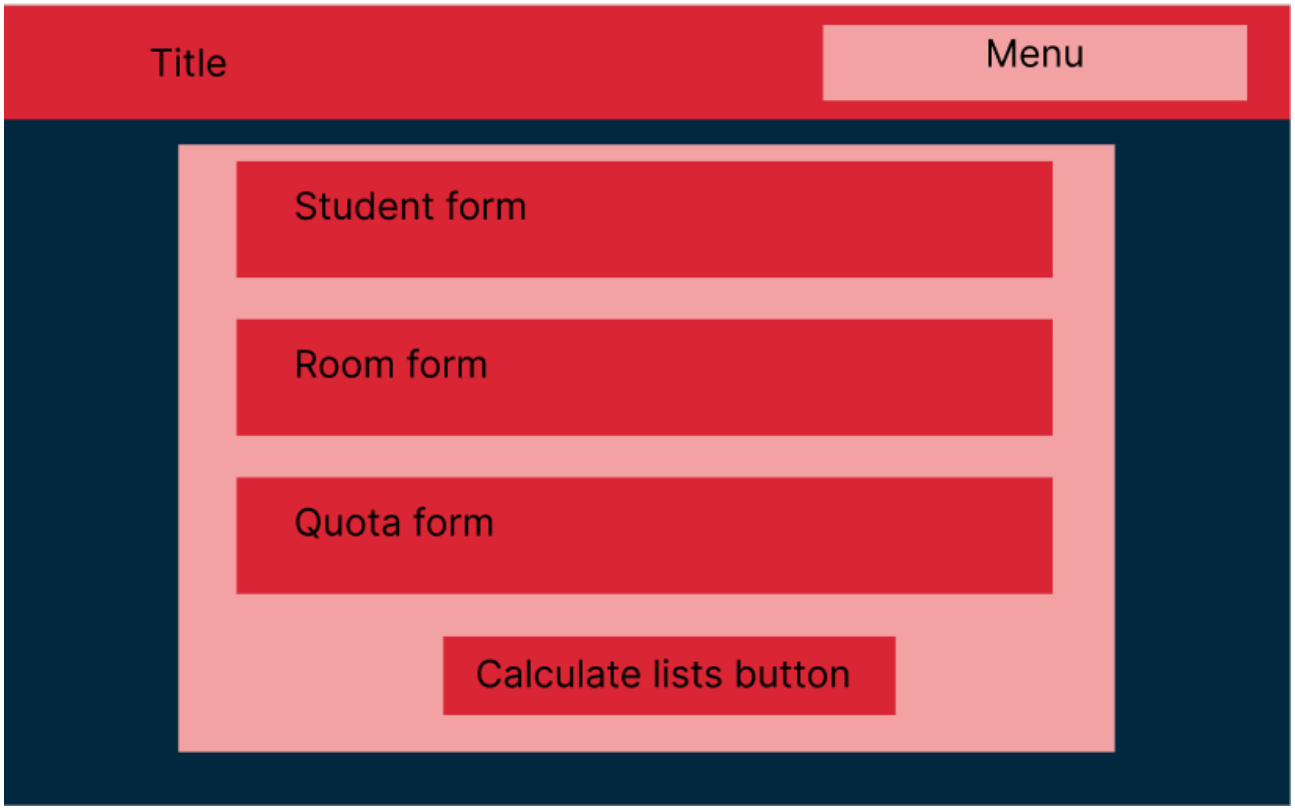


Рисунок 3.1 – Структура сторінки для додавання даних

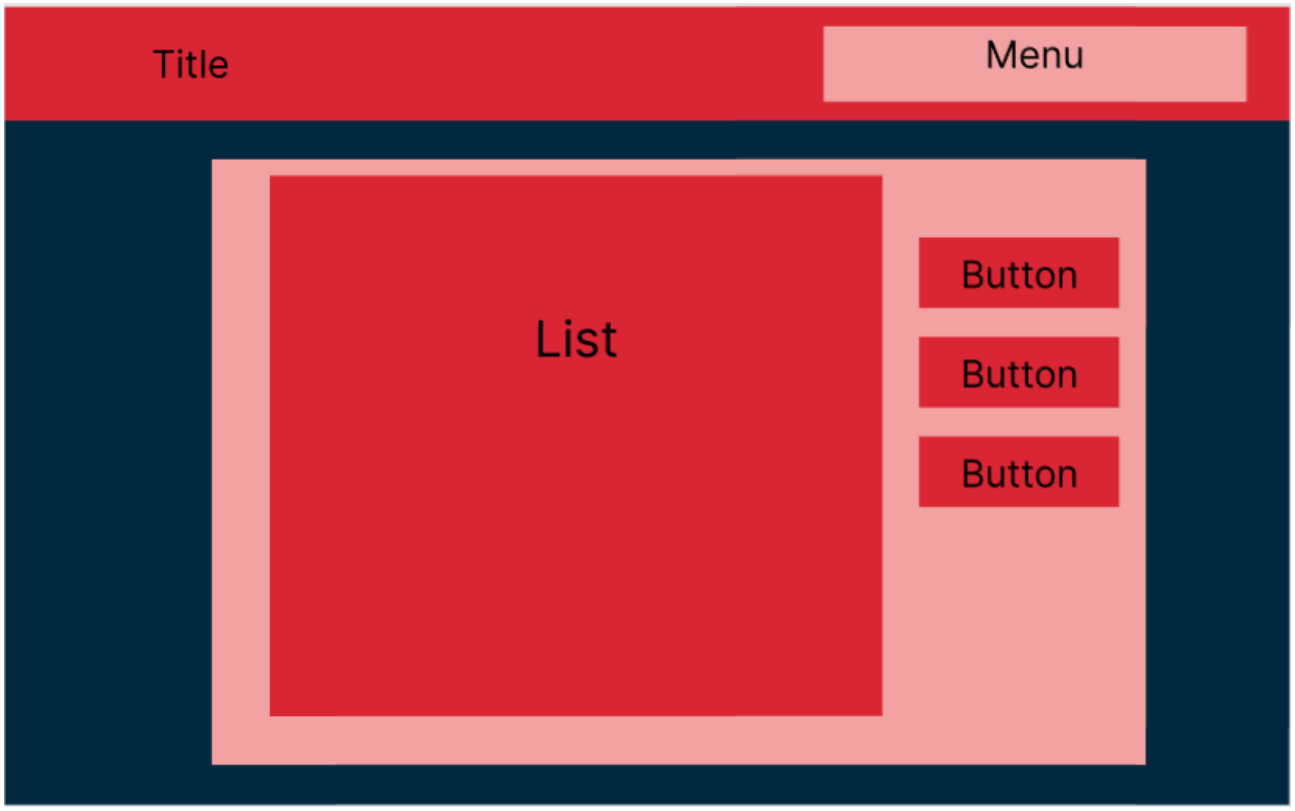


Рисунок 3.2 – Структура сторінок зі списком на поселення та черги

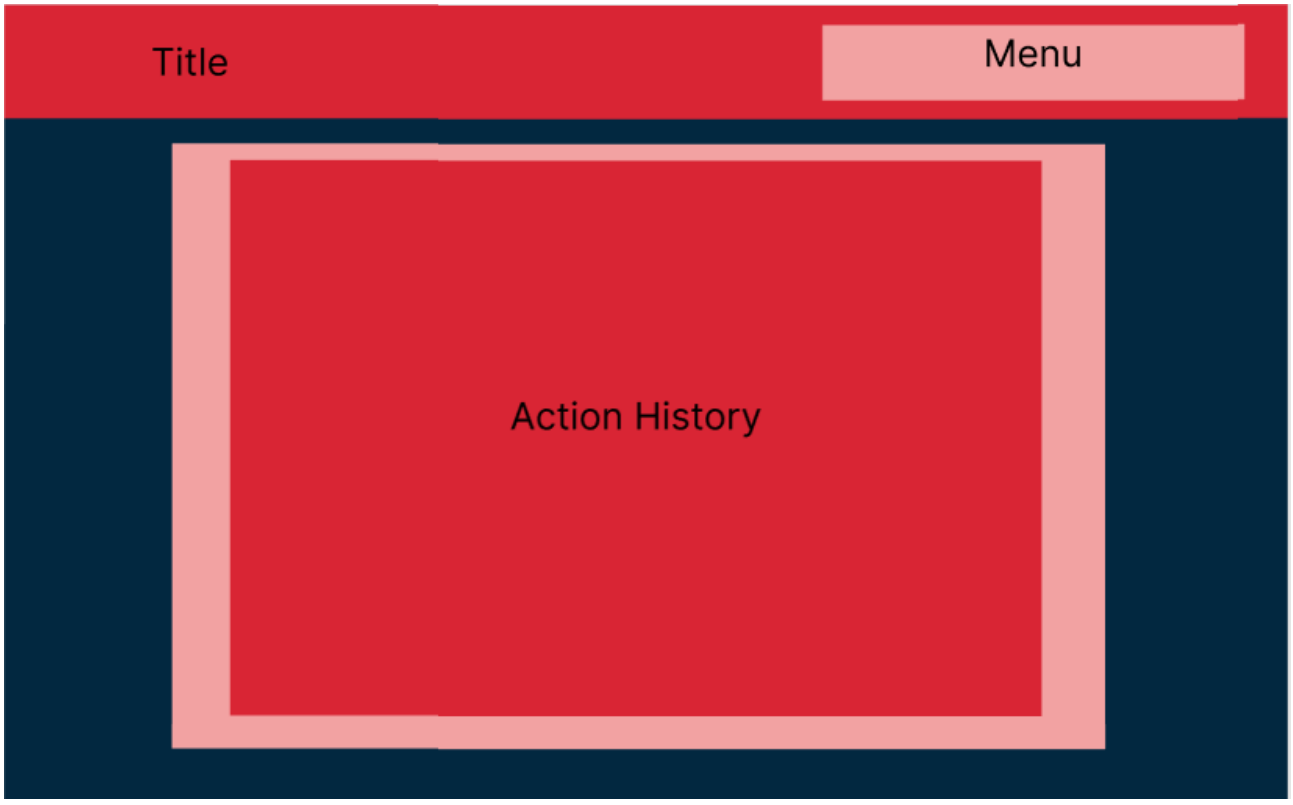


Рисунок 3.3 – Структура сторінки з історією дій

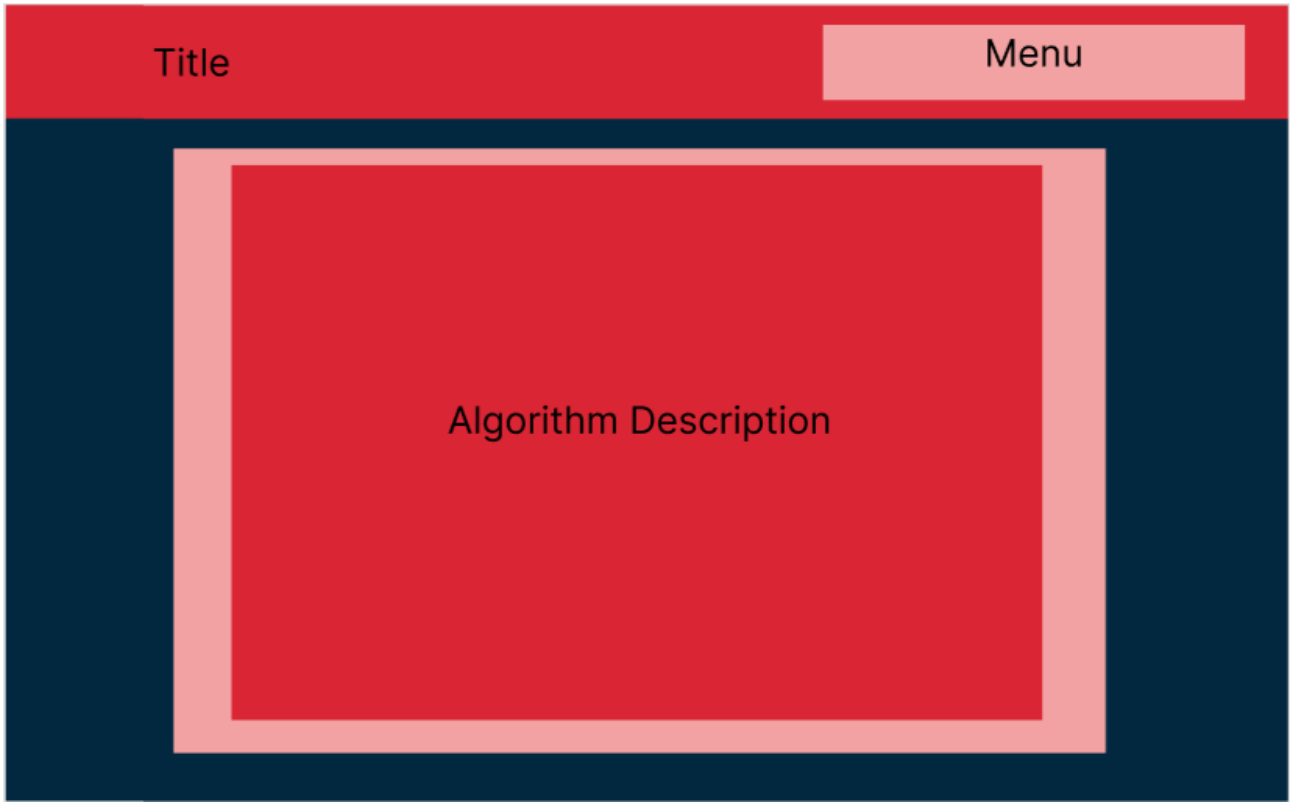


Рисунок 3.4 – Структура сторінки з описом алгоритму

## 3.3. Опис стуктури програмного забезпечення застосунку

### 3.3.1. Опис структури бекенд частини застосунку

Далі оглянемо сервісну частину веб-застосунку та її складові. Бекенд-частина застосунку має певну систему файлів та директорій, які взаємодіють між собою (рис.3.5).

Express.js працює за патерном MVC, який розшифровується як Model-View-Controller [16]. Спрощену схему роботи цього шаблону в реалізації Express.js зображено на рисунку 3.6.

У випадку застосунку, що розробляється, частина «Представлення» реалізована у фронтенд-частині веб-сервісу, тому на сервері необхідно реалізувати лише контролер, моделі та маршрутизацію.

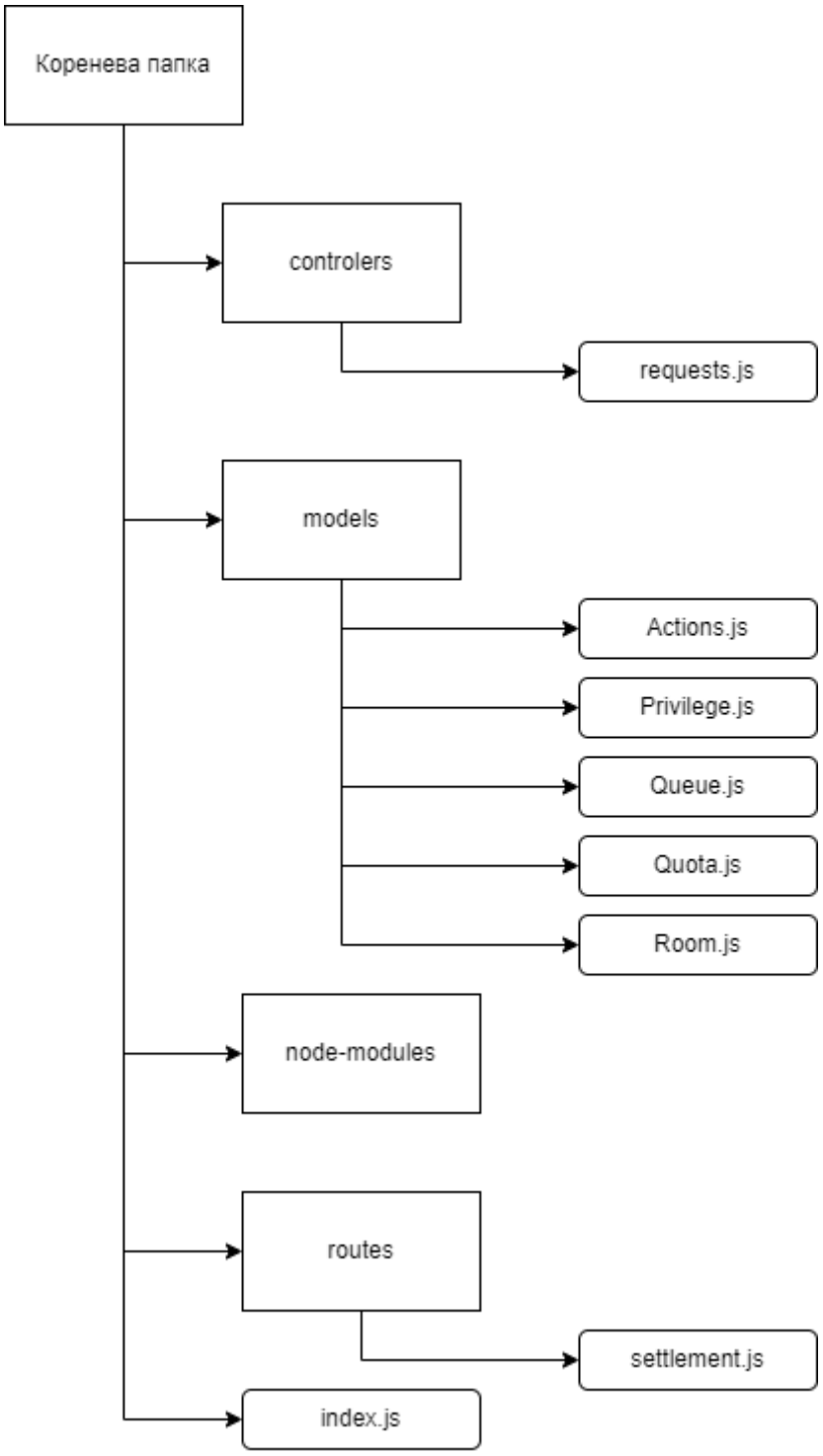


Рисунок 3.5 - Схема файлової структури бекенд-частини застосунку

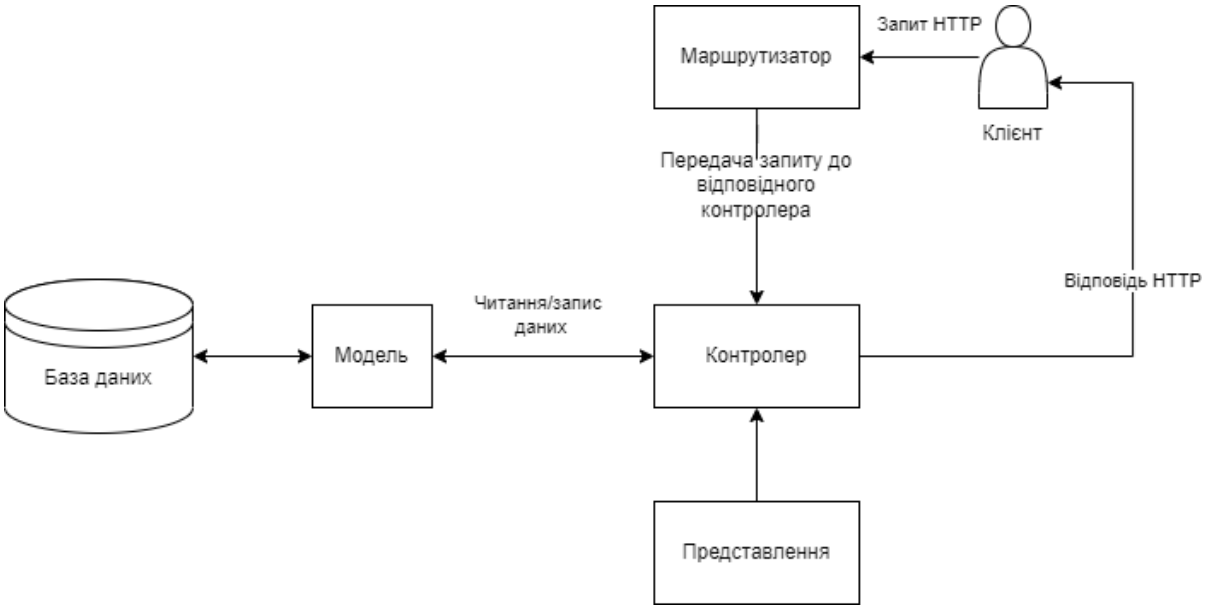


Рисунок 3.6 - Спрощена схема роботи шаблону MVC

Далі опишемо відповідні дерикторії файлової системи бекенд-частини застосунку.

*Таблиця 3.1 Опис дерикторій бекенд-застосунку*

| Назва | Опис |
|---|---|
| Коренева папка | Папка, в якій знаходяться всі файли веб-застосунку |
| node_modules | Папка, що містить всі необхідні для роботи веб-застосунку модулі. Це включаю в себе як і чисту бібліотеку React, так і будь-які сторонні модулі, встановлені розробником |
| controlers | Павпка, що містить реалізації всіх запитів до бази даних із застосуванням моделей. |
| models | Папка, що містить всі моделі для роботи застомунку, а саме: Action.js, Privilege.js, Queue.js, Quota.js, Room.js. Вони відповідно |

| | |
|---|---|
| | описують дії, пільги, чергу, квоти та місця. |
| routes | Папка, в якій всі кінцеві точки підключення до бази даних для взаємодії з конкретними структурами |
| index.js | Файл в якому реалізоване підключення до бази даних та основні бібліотеки, що використовуються |

3.3.2. Опис структури фронтенд-частини веб-застосунку

Аналогічно до структури, описаної в попередньому пункті, фронтенд-частина застосунку також має певні папки та компоненти, що необхідні для функціонування сайту. На схемі на рис. 3.7 представлено внутрішню файлову структуру програми

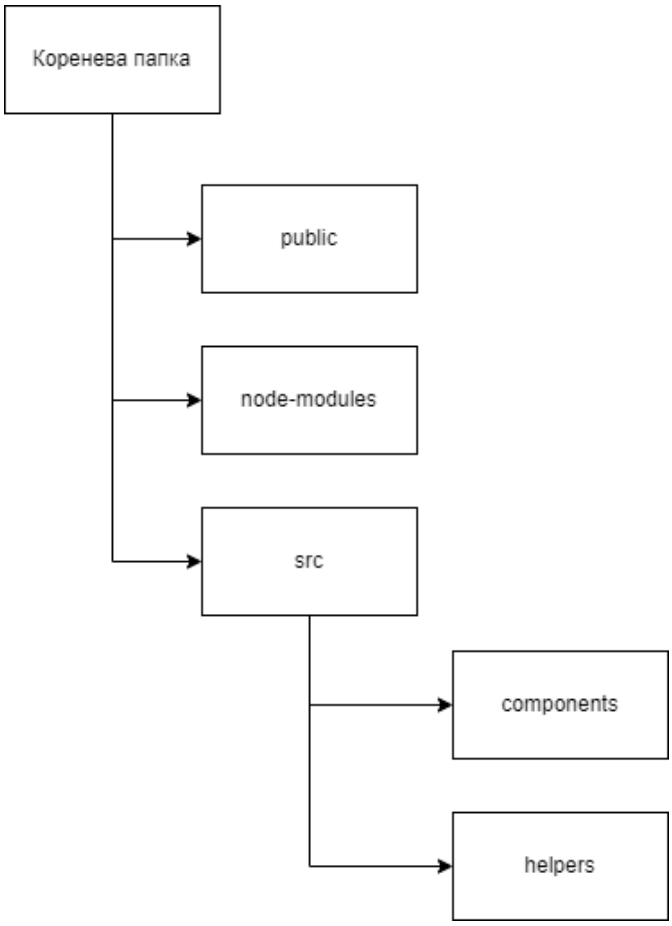


Рисунок 3.7 - Схема файлової структури фронтенд-частини застосунку

*Таблиця 3.2 Опис дерикторій фронтенд-застосунку*

| Назва | Опис |
|---|---|
| Коренева папка | Папка, в якій знаходяться всі файли веб-застосунку |
| node_modules | Папка, що містить всі необхідні для роботи веб-застосунку модулі. Це включаю в себе як і чисту бібліотеку React, так і будь-які сторонні модулі, встановлені розробником |
| public | Папка, що містить кінцеву сторінку з розміткою HTML, іконки, які будуть використовуватися на вкладці сайту, логотип, а також певні конфігурації файлу |
| src | Папка, в якій знаходиться весь основний код застосунку |
| components | Папка, в якій зберігаються код всіх компонентів веб-застосунку, та також файли стилів |
| helpers | Папка, в які містяться файли-хелпери, які призначені для роботи всередині інших компонентів |

3.3.3. Сторонні модулі, використані у веб-застосунку

Розглянемо модулі, що були використані для полегшення та пришвидшення розробки веб-застосунку, вони допомагають створити необхідний функціонал. Слід зазначити, що всі модулі пристосовані до роботи в браузерах на базі Chromium. Відповідні модулі описані у наступній талиці.

*Таблиця 3.3 – Сторонні модулі, використані у веб-застосунку, їх опис та призначення*

| Назва модуля | Опис модуля |
|---|---|
| react-loader-spinner | Бібліотека, яка надає для використання готові рішення для утворення лоадерів, які необхідні для застосунку в той час, як він отримує дані з сервера |
| swr | Модуль, який додає до функціоналу React новий хук, який полегшує роботу з сервером |
| react-bootstrap | Бібліотека стилів, що полегшує розмітку застосунку та об’єм опису стилів |
| xlsx | Модуль, який забезпечує взаємодію з файлами з розширенням xlxs та їх конвертацію в формат json |

## 3.4. Опис інструктивних матеріалів користувача

На даному етапі проект запускається локально на робочій машині користувача. Для початку необхідно запустити бекенд-частину за допомогою команди npm run dev та схожим чином запустити фронтенд-частину команжою npm start. Після чого застосунок відкриється на новій сторінці в браузері по замовчуванню.

Далі оглянемо інтерфейс застосунку. Першої сторінкою, яку бачить користувач після відкриття застосунку є сторінка алгоритму (рис. 3.8). На ній коротко описано призначення застосунку та порядок використання сайту. Також коротко описано алгоритм формування списків та розрахунку пріоритетів.

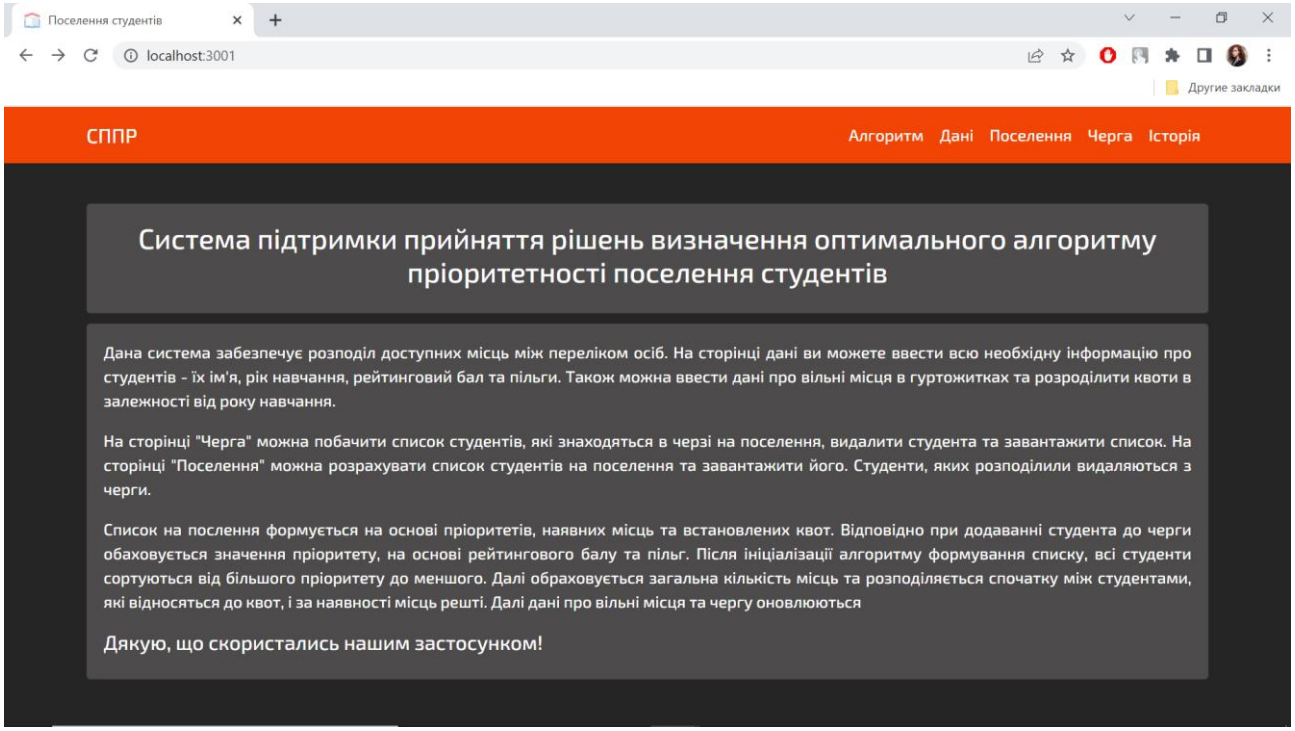


Рисунок 3.8 – Головна сторінка застосунку

На верхній панелі користувач може перейти на одну з п’яти сторінок: «Дані», «Поселення», «Черга» та «Історія». Спочатку розглянемо сторінку «Дані» (рис.9). На ній користувач може внести всю необхідну інформацію для розрахунків та перейти до формування списків на поселення.

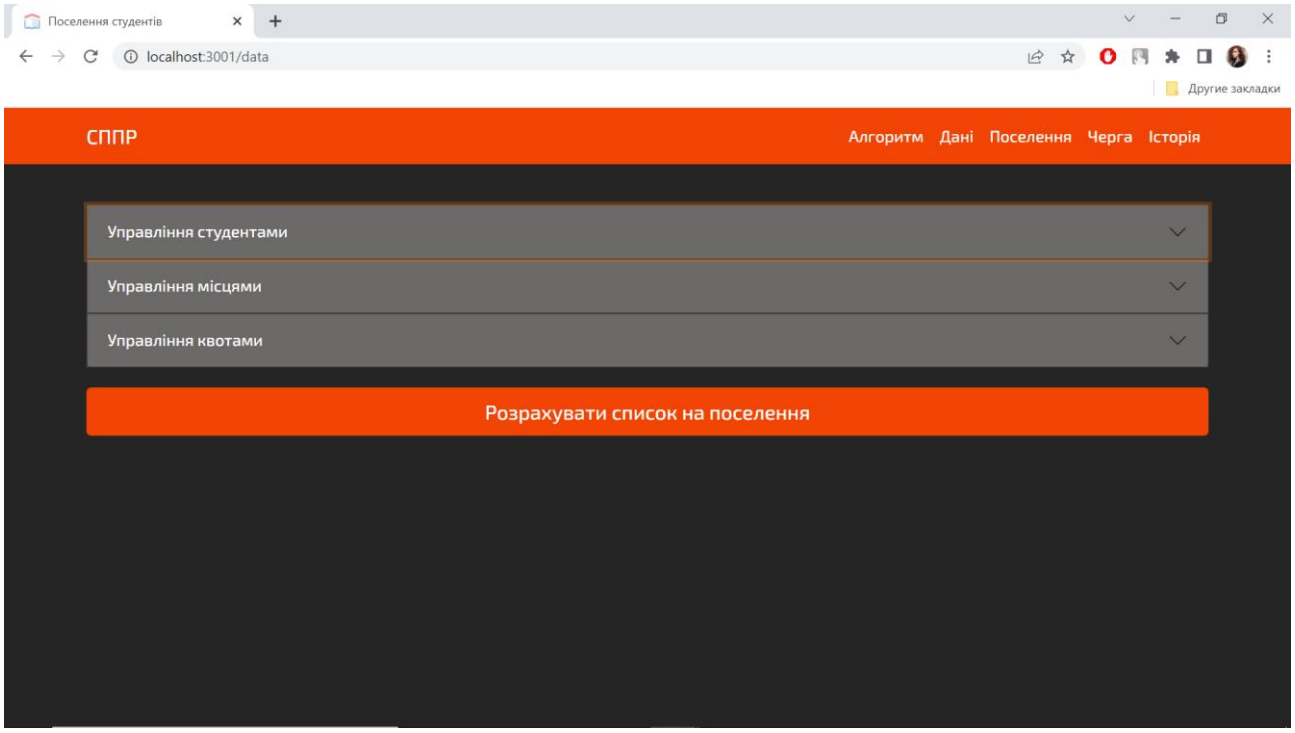


Рисунок 3.9 – Сторінка «Дані».

На даній сторінці є три блоки, поділені по функціоналу. Блок «Управління студентами», «Управління місцями» та «Управління квотами». Автоматично відкритий перший блок, де користувач може додати студента (рис. 3.10) , або додати список студентів (рис. 3.11).


Рисунок 3.10 – Сторінка «Дані». Форма додавання студента


Рисунок 3.11 – Сторінка «Дані». Вкладка додавання файлу зі списком студентів

У наступному блоці «Управління місцями» також є дві вкладки. Перша - «Наявні місця» (рис. 3.12) та друга – «Додати місця» (рис. 3.13). Відповідно на першій користувач може побачити місця, які вже доступні, а на другій вкладці він може додати гуртожиток з місцями. Для цього йому необхідно зазначити номер гуртожитку в першому полі та кількість місць в наступному. Також за необхідності користувач може додати місця без вказання номеру гуртожитка.

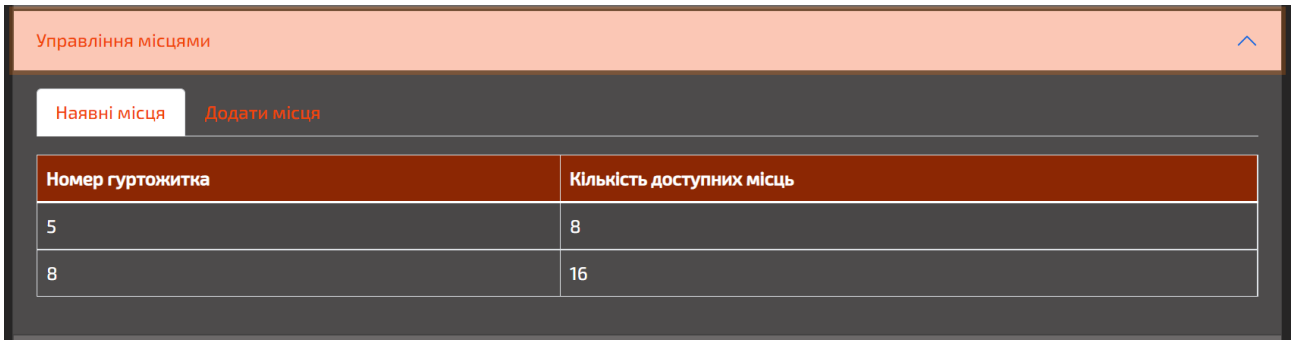


Рисунок 3.12 – Сторінка «Дані». Вкладка з наявним списком гуртожитків та місць.

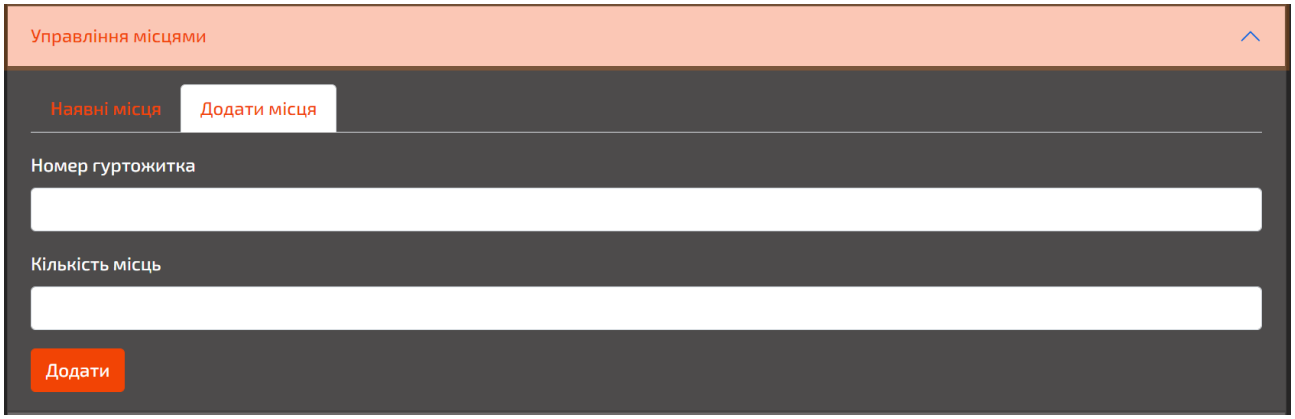


Рисунок 3.13 – Сторінка «Дані». Вкладка з формою додавання місць.

Аналогічно до попереднього блоку в блоці «Управління квотами» є вкладки зі списком наявних місць (рис. 3.14), де також у користувача є можливість видалити квоту, та з формою для додавання нових квот (рис. 3.15).

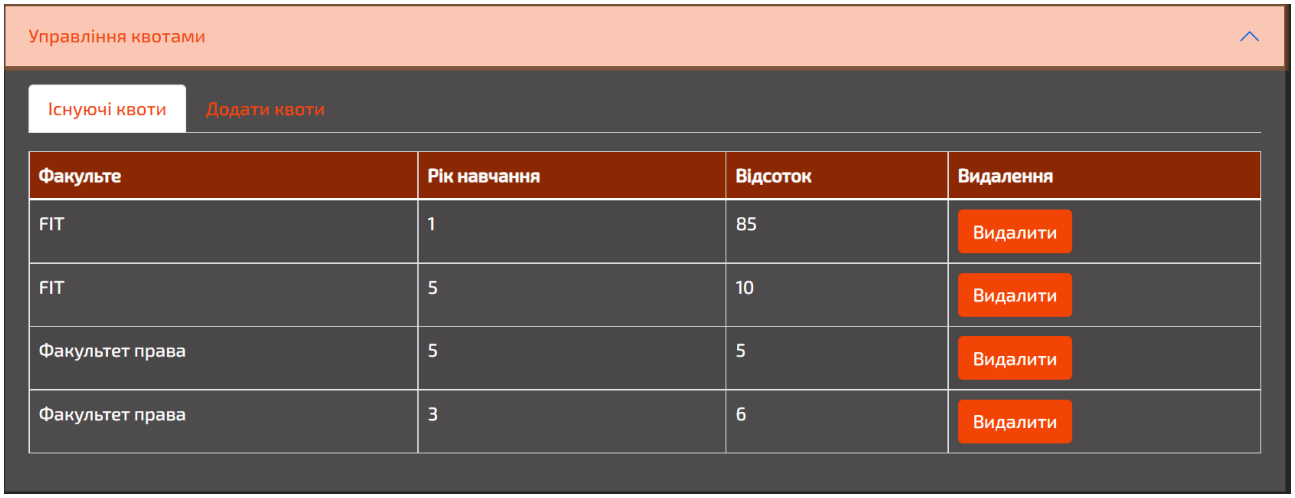


Рисунок 3.14 – Сторінка «Дані». Вкладка зі списком квот.

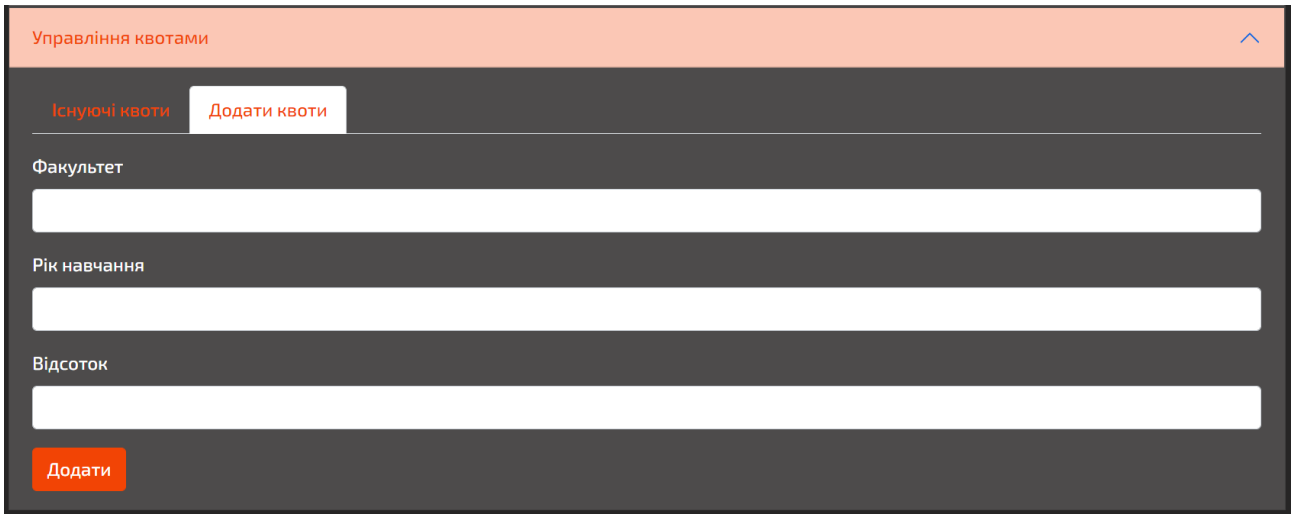


Рисунок 3.15 – Сторінка «Дані». Вкладка з формою для додавання квот.

Після введення всіх даних користувач може натиснути на кнопку «Розрахувати список на поселення», після чого його переадресує на сторінку «Поселення». Або він може натиснути на одне з посилань на верхній панелі.

Далі розглянемо сторінку «Черга» (рис. 3.16), на ній користувач може побачити список студентів в черзі. ПІБ студента, його рік навчання, рейтинговий бал та пільги за наявності. Також поряд з кожним студентом є кнопка «Видалити», що видаляє студента з бази даних. Користувач може завантажити список черги. Список завантажується у форматі xlsx (рис.3. 17).

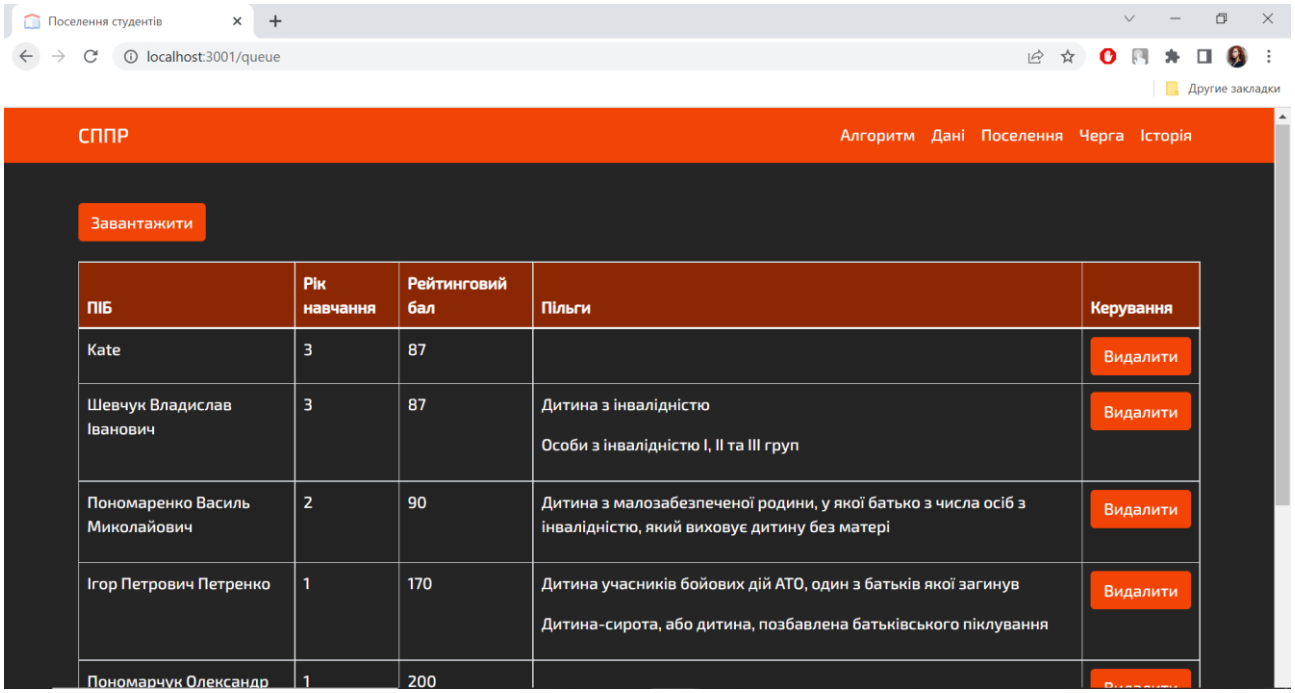


Рисунок 3.16 – Сторінка «Черга»

| | ПІБ | Рік навчання | Оцінка | Пільги |
|---|---|---|---|---|
| 2 | Kate | 3 | 87 | |
| 3 | Шевчук Владислав Іванович | 3 | 87 | Дитина з інвалідністю,Особи з інвалідністю I, II та III груп |
| 4 | Пономаренко Василь Миколайович | 2 | 90 | Дитина з малозабезпеченої родини, у якої батько з числа осіб з інвалідністю, який виховує дитину без матері |
| 5 | Ігор Петрович Петренко | 1 | 170 | Дитина учасників бойових дій АТО, один з батьків якої загинув,Дитина-сирота, або дитина, позбавлена батьківського |
| 6 | Пономарчук Олександр Іванович | 1 | 200 | |
| 7 | Ольга Федорович Кравчук | 1 | 187 | Чорнобильці (перша, друга та третя категорії) |
| 8 | Шевченко Тимофій Олексійович | 1 | 191 | |
| 9 | Брованенко Юлія Тарасович | 6 | 89 | |
| 10 | Роман Іванович Микитюк | 5 | 89 | |
| 11 | Василенко Любов Андрійович | 5 | 66 | Внутрішньо переміщені особи |

Рисунок 3.17 – Завантажений файл

Далі розглянемо сторінку «Поселення» (рис. 3.18). На ній користувачу потрібно вказати факультет, відповідно до якого будуть обиратись квоти. Заповнення поля факультету є обов’язковим Далі відбувається обчислення та формування списку. Після чого він відображається на сторінці. Аналогічно до списку черги його можна завантажити.

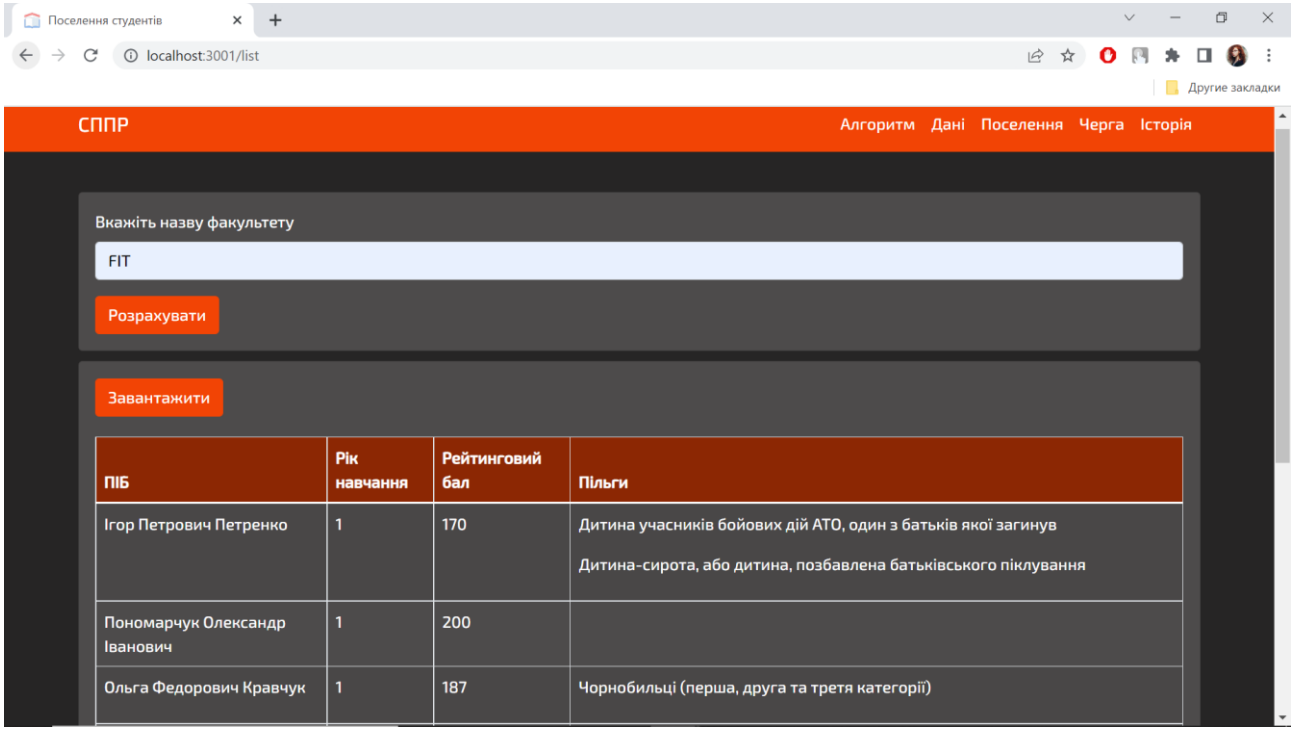


Рисунок 3.18 – Сторінка «Поселення»

Останньою розглянемо сторінку «Історія», де користувач може побачити історію своїх дій та час їх виконання (рис.3.19). Дії відображаються у порядку від найпершої до останньої. Час кожної дії передається в форматі дд/мм/рррр @ гг:хх:сс.

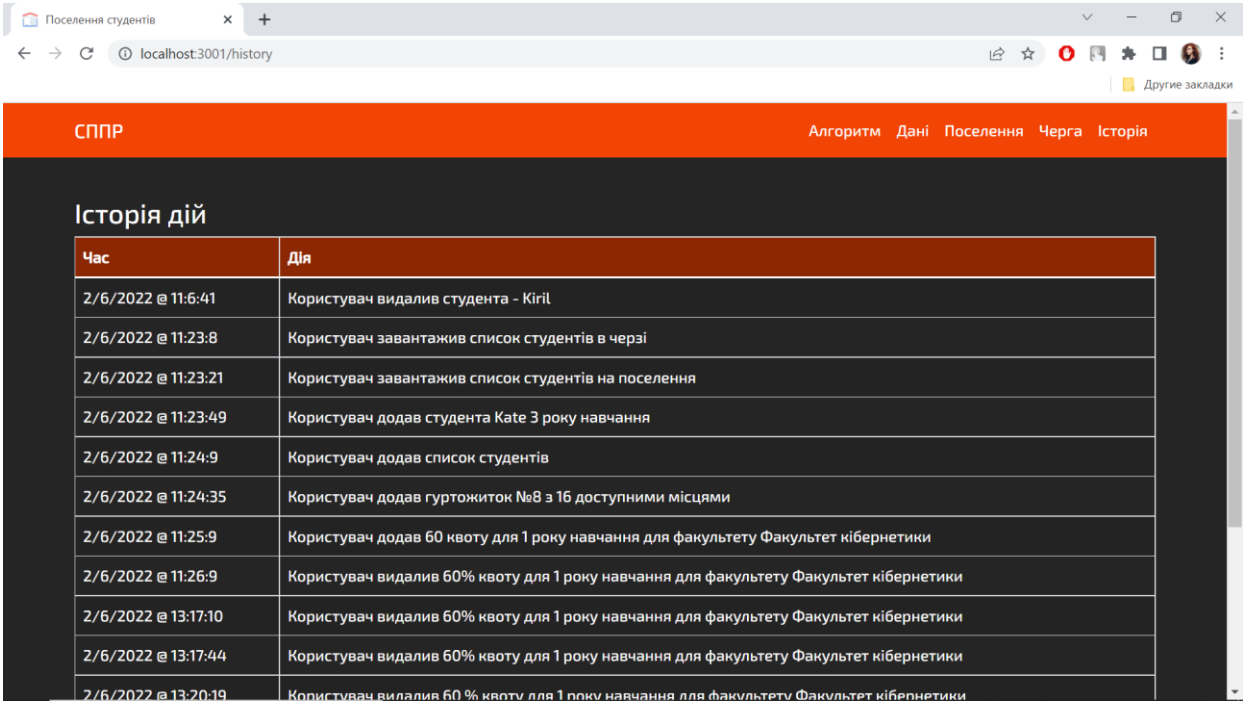


Рисунок 3.18 – Сторінка «Історія»

## 3.5. Тестування застосунку

На завершальному етапі розглянемо процес тестування СППР визначення оптимального алгоритму пріоритетності поселення студентів . Задля цього напишемо тести у вигляді тест-кейсів.

*Таблиця 3.4 – Опис тест-кейсів*

| № | Пріоритет | Блок | Кроки | Очікувані результати |
|---|---|---|---|---|
| 1 | критичний | Ініціалізація | Користувач може зайти на сайт<br>1. Запустити застосунок на локальному сервері | 1. Застосунок відкрився на сторінці «Алгоритм» |
| 2 | надвисокий | Поселення | Користувач може отримати список на поселення<br>Передумови:<br>1. Студенти є в черзі<br>2. Наявні вільні місця<br>3. Налаштовані квоти | 1. Сторінка «Поселення» відкрита<br>2. Форма заповнена |

| | | | | |
|---|---|---|---|---|
| | | | Кроки:<br>1. Перейти на сторінку «Поселення»<br>2. Внести назву факультету<br>3. Натиснути на кнопку «Обчислити»<br>4. Оглянути екран | 3. Кнопка переходить в стан «Обчислення»<br>4. Кнопка повертається до попереднього стану.<br>Список на поселення відображено |
| 3 | надвисокий | Управління даними | Користувач може додати квоти<br>1. Перейти на сторінку «Дані»<br>2. Відкрити блок «Управління квотами»<br>3. Натиснути на вкладку «Додати квоти»<br>4. Заповнити форму<br>5. Натиснути на кнопку «Додати»<br>6. Натиснути на вкладку «Існуючі квоти»<br>7. Оглянути вміст вкладки | 1. Сторінка «Дані» відкрита<br>2. Вкладка «Існуючі квоти» відображена<br>3. Вкладка «Додати квоти» відображена<br>4. Форма заповнена<br>5. Напис на кнопці змінюється |

| | | | | |
|---|---|---|---|---|
| | | | | на «Додавання». Та повертається в попередній стан<br>6. Вкладка «Існуючі квоти» відкрита<br>7. Додана квота є в таблиці |
| 4 | високий | Управління даними | Користувач може додати вільні місця<br>1. Перейти на сторінку «Дані»<br>2. Відкрити блок «Управління місцями»<br>3. Натиснути на вкладку «Додати місця»<br>4. Заповнити форму<br>5. Натиснути кнопку «Додати»<br>6. Натиснути на вкладку «Наявні місця»<br>7. Оглянути вміст вкладки | 1. Сторінка «Дані» відкрита<br>2. Вкладка «Наявні місця» відображена<br>3. Вкладка «Додати місця» відображена<br>4. Форма заповнена<br>5. Напис на кнопці змінюється |

| | | | | |
|---|---|---|---|---|
| | | | | на «Додавання». Та повертається в попередній стан<br>6. Вкладка «Наявні місця» відкрита<br>7. Додані місця є в таблиці |
| 5 | високий | Управління даними | Користувач може додати нового студента до черги<br>1. Перейти на сторінку «Дані»<br>2. Заповнити форму студента в блоці «Управління студентами»<br>3. Натиснути кнопку «Додати»<br>4. Перейти на сторінку «Черга»<br>5. Оглянути список черги | 1. Сторінка відкрита<br>2. Форма заповнена<br>3. Після натискання напис на кнопці – «Додавання». Повернення в початковий стан<br>4. Сторінка «Черга» відкрита.<br>5. Додани |

| | | | | |
|---|---|---|---|---|
| | | | | й студент присутній |
| 6 | високий | Завантаження | Користувач може завантажити список на поселення<br>Передумови:<br>1. Список на поселення сформовано<br>Кроки:<br>1. Натиснути на кнопку «Завантажити» | 1. Список завантажено |
| 7 | середній | Черга | Користувач може переглянути список черги<br>1. Перейти на сторінку «Черга»<br>2. Оглянути екран | 1. Сторінка відкрита<br>2. Таблиця списку черги присутня |
| 8 | середній | Історія | Користувач може побачити історію своїх дій<br>Передумови:<br>1. Виконати операцію додавання/видалення/обчислення<br>Кроки:<br>1. Перейти на сторінку «Історія»<br>2. Переглянути вміст | 1. Сторінка «Історія» відрита<br>2. Список історії дій відображено |

В результаті програмної розробки застосунку та проведення тестування отримуємо застосунок, який дозволяє розподілити місця в гуртожитках між здобувачами освіти.

# ВИСНОВКИ

Незважаючи на перешкоди стаціонарне навчання в університетах не втратить своєї популярності, а це значить необхідність надання житла здобувачам освіти, які цього потребують. Враховуючи це розроблений застосунок буде актуальним та доцільним для використання.

У ході виконання дипломної роботи було проведено аналіз існуючих рішень щодо розподілу обмежених ресурсів, таких як житлові місця. В результаті аналізу були сформовані вимоги до застосунку для забезпечення зручної взаємодії користувача з ним. Була сформована математична модель факультету, яка було використано в розробці застосунку.

Для реалізації даного веб-застосунку були спроектовані база даних, дерево функцій, структура додатку та його архітектура.

У результаті роботи було створено веб-застосунок на базі інструментів MongoDB, NodeJS та React. Функціонал застосунку було протестовано за тест-кейсами описаними в даній роботі. Всі тести пройшли успішно, що підтверджує працездатність застосунку.

В перспективах даний додаток можна покращити додавши особистий кабінет користувача для персоніфікації збережених даних, оптимізувати роботу застосунку на різних платформах та уніфікувати структури для більшої кількості користувачів. Це забезпечить більшу популярність системи та підтвердить її необхідність.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний веб-сайт студмістечка КНУ [Електронний ресурс]: Режим доступу - https://studmisto.knu.ua/dormitories

2. Офіційний веб-сайт Київського Національного університету імені Тараса Шевченка [Електронний ресурс] - Режим доступу: http://www.univ.kiev.ua/pdfs/zvit/zvit-rektora-2021.pdf

3. Hnatiienko H., Izhevska Ye. Mathematical Model for Determination of Prioritizing Services When Allocating Limited Resources // Information Technology and Implementation (Satellite): Conference Proceedings, December 02, 2021, Kyiv, Ukraine / Taras Shevchenko National University of Kyiv and [etc]; Vitaliy Snytyuk (Editor). – Kyiv: Publisher Individual entrepreneur Picha Y.V., 2021. Pp. 44-46.

4. Дж. Ролз. Теорія справедливості. 1995. ISBN: 5-7615-0365-4 - 536 с.

5. Офіційний веб-сайт факультету Інформаційних технологій КНУ [Електронний ресурс] - Режим доступу: http://fit.univ.kiev.ua/wp-content/uploads/2020/08/%D0%9D%D0%B0%D0%BA%D0%B0%D0%B7.pdf

6. Офіційний веб-сайт Інституту права [Електронний ресурс] - Режим доступу: https://law.knu.ua/ua/

7. The Use of IDEF0 for the Design and Specification of Methodologies [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/2447898_The_Use_of_IDEF0_for_the_Design_and_Specification_of_Methodologies

8. Automation of strategy using IDEF0 — A proof of concept [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S2214716015000111

9. James Urquhart. The data flow value chain. 2016 [Електронний ресурс] - Режим доступу: https://medium.com/digital-anatomy/the-data-flow-value-chain-45b0dd3083e8

10. Моделювання бізнес процесів. [Електронний ресурс] - Режим доступу: https://koptelov.info/publikatsii/modelirovanie-biznes-protsessov/

11. Data Flow Diagram. [Електроний ресурс] – Режим доступу: https://www.smartdraw.com/data-flow-diagram/

12. Основні поняття реляційних БД: нормалізація, зв’язок та ключі [Електронний ресурс] // Bondarenko – Режим доступу до ресурсу: https://bondarenko.dn.ua/osnovni-ponyattya-relyatsijnih-bd-normalizatsiya-zv-yazok-ta-klyuchi/

13. Mongoose API Docs [Електронний ресурс] // MongooseJS – Режим доступу: https://mongoosejs.com/docs/api.html (дата звернення: 02.03.2022)

14. Ітан Браун. Веб-розробка з використанням Node та Express: Підручник / Ітан Браун – Веб-розробка з використанням Node та Express. Повноцінне використання стеку JavaScript. 2-га редакція, 2019 – 124 с.

15. React JS: Advantages and Disadvantages? [Електронний ресурс] // Wojciech Baranowski – Режим доступу до ресурсу: https://massivepixel.io/blog/react-advantages-disadvantages/

16. MVC Pattern [Електронний ресурс] // Tutorialspoint – Режим доступу до ресурсу: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

# ДОДАТОК А

Лістинг Front-end частини застосунку

## index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from "react-router-dom";

import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

## App.js

```
//Packages
import React from 'react';
import {Routes, Route} from "react-router-dom";
//Components
import Menu from "./components/Menu/Menu";
import Queue from "./components/Queue/Queue";
import InputData from "./components/InputData/InputData";
import StudentList from "./components/StudentList/StudentList";
import Algorithm from "./components/Algorithm/Algorithm";
import History from "./components/History";
//Styles
import './App.scss';
import 'bootstrap/dist/css/bootstrap.min.css';


const App = () => {
    return (
        <div className='App'>
            <Menu/>
            <Routes>
                <Route path='/' element={<Algorithm/>}/>
                <Route path='/data' element={<InputData/>}/>
```

```
            <Route path='/list' element={<StudentList/>}/>
            <Route path='/queue' element={<Queue/>}/>
            <Route path='/history' element={<History/>}/>
          </Routes>
        </div>
    )
}

export default App;
```

## mapHelper.js

```
export const actionSave = (text) => {
    let currentDate = new Date();
    let datetime = currentDate.getDate() + "/"
        + (currentDate.getMonth() + 1) + "/"
        + currentDate.getFullYear() + " @ "
        + currentDate.getHours() + ":"
        + currentDate.getMinutes() + ":"
        + currentDate.getSeconds();
    let action = {"name": text, "time": datetime}
    fetch('http://localhost:3000/action/add', {
        method: 'POST',
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify(action)
    }).then()
}
```

## useFetch.js

```
import {useState, useEffect} from "react";


const useFetch = (url) => {
    const [data, setData] = useState(null)
    const [isPending, setIsPending] = useState(true)

    useEffect(() => {
        fetch(url)
            .then(res => {
                return res.json()
            }
            )
            .then(data => {
                setData(data)
```

```
            setIsPending(false)
        })
    }, [url])
    return {data, isPending}
}

export default useFetch
```

## Menu.js

```
import React, {useState} from 'react'
import './Menu.scss'
import {Nav, Navbar, Container} from "react-bootstrap";


const Menu = () => {

    const [active, setActive] = useState('default');
    return (
        <Navbar>
            <Container>
                <Navbar.Brand href="/">СППР</Navbar.Brand>
                <Nav className="justify-content-end"
                    activeKey={active}
                    onSelect={(selectedKey) => setActive(selectedKey)}
                >
                    <Nav.Item>
                        <Nav.Link href="/" className={'linkHover'}>Алгоритм</Nav.Link>
                    </Nav.Item>
                    <Nav.Item>
                        <Nav.Link href="/data" className={'linkHover'}>Дані</Nav.Link>
                    </Nav.Item>
                    <Nav.Item>
                        <Nav.Link href="/list" className={'linkHover'}>Поселення</Nav.Link>
                    </Nav.Item>
                    <Nav.Item>
                        <Nav.Link href="/queue" className={'linkHover'}>Черга</Nav.Link>
                    </Nav.Item>
                    <Nav.Item>
                        <Nav.Link href="/history" className={'linkHover'}>Історія</Nav.Link>
                    </Nav.Item>
                </Nav>
            </Container>
```

```
        </Navbar>
    )
}

export default Menu
```

## Queue.js

```
import React, {useEffect, useState} from "react";
import './Queue.sass'
import {Rings} from 'react-loader-spinner'
import {Button, Table} from "react-bootstrap";
import useFetch from "../../heplers/useFetch";
import FileSaver, {fileType, fileFormat} from 'file-saver'
import * as XLSX from "xlsx";
import {actionSave} from "../../heplers/mapHelper";



const Queue = () => {
    async function deleteStudent(id) {
        await fetch(`http://localhost:3000/queue/delete/${id}`, {
            method: "DELETE",
        });
    }

    const {data, isPending} = useFetch('http://localhost:3000/queue')
    const [tableData, setTableData] = useState(data)



    const refresh = () => {
        window.location.reload();
    }

    const handleDownload = () => {
        const studentArr = []
        data.forEach(student => {
            studentArr.push(student.student[0])
        })
        studentArr.forEach(student => {
            student.privilege = student.privilege.toString()
            delete student.priority
            delete student._id
        })
```

```
    let header = ["ПІБ", "Рік навчання", "Оцінка", "Пільги"];
    const ws = XLSX.utils.book_new();
    XLSX.utils.sheet_add_aoa(ws, [header]);
    XLSX.utils.sheet_add_json(ws, studentArr, {origin: 'A2', skipHeader: true});
    const wb = {Sheets: {'data': ws}, SheetNames: ['data']};
    const excelBuffer = XLSX.write(wb, {bookType: fileType, type: 'array', cellStyles: true});
    const file = new Blob([excelBuffer], {type: fileFormat});
    FileSaver.saveAs(file, "Queue.xlsx");
}

return (
    <div className='container col-sm6'>
        <Button variant="primary" onClick={(e) => {
            handleDownload(e)
            actionSave('Користувач завантажив список студентів в черзі')
        }}>
            Завантажити
        </Button>
        <div className="resultTable"></div>
        {data && <Table responsive striped bordered hover>
            <thead>
            <tr>
                <th>ПІБ</th>
                <th>Рік навчання</th>
                <th>Рейтинговий бал</th>
                <th>Пільги</th>
                <th>Керування</th>
            </tr>
            </thead>
            <tbody>{data.map(el => (
                <tr key={el._id}>
                    <td>{el.student[0].name}</td>
                    <td>{el.student[0].year}</td>
                    <td>{el.student[0].mark}</td>
                    <td>{el.student[0].privilege.map(priv => (
                        <p key={priv}>{priv}</p>
                    ))}</td>
                    <td><Button onClick={
                        () => {
                            deleteStudent(el._id).then(refresh)
                            actionSave(`Користувач видалив студента  - ${el.student[0].name}`)
```

```
                    }
                }>Видалити</Button></td>
            </tr>
        ))}</tbody>
      </Table>}
      {isPending && <Rings/>}
    </div>
  )
}

export default Queue
```

## StudentList.js

```
import React, {useState} from "react";
import {Button, Card, Form, Table} from "react-bootstrap";
import * as XLSX from 'xlsx'
import FileSaver, {fileType,fileFormat} from 'file-saver'
import './StudentList.scss'
import {actionSave} from "../../heplers/mapHelper";


async function getData(url) {
  return await fetch(url)
    .then(res => res.json())
    .then(responseJson => {
      return responseJson
    })
}

const StudentList = () => {
  const [faculty, setFaculty] = useState(null)
  const [isPending, setIsPending] = useState(false)
  const [isDisplayed, setIsDisplayed] = useState(false)
  const [finalData, setFinalData] = useState([])

  const handleSubmit = async (e) => {
    e.preventDefault()
    setIsPending(true)

    const students = await getData('http://localhost:3000/queue')
    let rooms = await getData('http://localhost:3000/rooms')
    let allPlaces = 0
```

```
let tackedNPlaces = 0
const places = []
const settledStudents = []
let isAllPlacesBooked = false
rooms.map((el) => {
   allPlaces += el.nPlaces
})
const quotas = await getData(`http://localhost:3000/quotas/${faculty}`)
for (let i = 0; i < quotas.length; i++) {
   places.push(Math.floor(quotas[i].percent / 100 * allPlaces))
   if (i === quotas.length - 1) {
      let booked = places.reduce((el1, el2) => el1 + el2)
      places.push(allPlaces - booked)
   }
   students.sort((a, b) => (a.priority > b.priority) ? 1 : -1)
   console.log("students", students)
   let currentStudents = students.filter((student) => student.student[0].year === quotas[i].year)
   currentStudents.sort((a, b) => (a.priority > b.priority) ? 1 : -1)
   console.log("current students" ,currentStudents)
   currentStudents.forEach(currentStudent => {
      if (allPlaces > 0) {
         allPlaces -= 1
         students.splice(students.indexOf(currentStudent), 1)
         fetch(`http://localhost:3000/queue/delete/${currentStudent._id}`, {
                  method: "DELETE",
               })
         tackedNPlaces += 1
         settledStudents.push(currentStudent)
      } else {
         isAllPlacesBooked = true
      }
   })
   !isAllPlacesBooked && students.forEach(student => {
      if (allPlaces > 0) {
         allPlaces -= 1
         students.splice(students.indexOf(student), 1)
         fetch(`http://localhost:3000/queue/delete/${student._id}`, {
                  method: "DELETE",
               })
         tackedNPlaces += 1
         settledStudents.push(student)
```

```
            } else {
                isAllPlacesBooked = true
            }
        })
        console.log("first", tackedNPlaces)
        for(let k = 0; k<rooms.length && tackedNPlaces>0; k++) {
            if (tackedNPlaces < rooms[k].nPlaces) {
                let newPlaces = rooms[k].nPlaces - tackedNPlaces
                rooms[k].nPlaces = newPlaces
                await fetch(`http://localhost:3000/room/update/${rooms[k]._id}`, {
                    method: 'PUT',
                    headers: {"Content-Type": "application/json"},
                    body: JSON.stringify({"nPlaces": newPlaces})
                })
                tackedNPlaces = 0
            } else {
                await fetch(`http://localhost:3000/room/delete/${rooms[k]._id}`, {
                    method: "DELETE",
                })
                tackedNPlaces -= rooms[k].nPlaces
                rooms[k].nPlaces = 0
                console.log(rooms[k].dormitoryN," ",  tackedNPlaces)
            }
        }

    }
    setFinalData(settledStudents)
    setIsPending(false)
    setIsDisplayed(true)
}

const handleDownload = () => {
    const studentArr = []
    finalData.forEach(student => {
        studentArr.push(student.student[0])
    })
    studentArr.forEach(student => {
        student.privilege = student.privilege.toString()
        delete student.priority
        delete student._id
    })
```

```
    let header = ["ПІБ", "Рік навчання", "Оцінка", "Пільги"];
    const ws = XLSX.utils.book_new();
    XLSX.utils.sheet_add_aoa(ws, [header]);
    XLSX.utils.sheet_add_json(ws, studentArr, { origin: 'A2', skipHeader: true });
    const wb = { Sheets: { 'data': ws }, SheetNames: ['data'] };
    const excelBuffer = XLSX.write(wb, { bookType: fileType, type: 'array', cellStyles:true });
    const file = new Blob([excelBuffer], { type: fileFormat });
    FileSaver.saveAs(file, "Students.xlsx");
}


return (
    <div className={'container'}>
        <Card body>
            <Form onSubmit={handleSubmit}>
                <Form.Group className="mb-3" controlId="addRooms">
                    <Form.Label>Вкажіть назву факультету</Form.Label>
                    <Form.Control type="text"
                                  placeholder=""
                                  value={faculty}
                                  onChange={(e) => setFaculty(e.target.value)}
                                  required={true}
                    />
                </Form.Group>
                {!isPending && <Button variant="primary" type="submit">
                    Розрахувати
                </Button>}
                {isPending && <Button variant="primary" type="submit">
                    Обчислення...
                </Button>}
            </Form>
        </Card>
        {isDisplayed && <Card body>
            <Button variant="primary" onClick={(e)=>{
                handleDownload(e)
                actionSave('Користувач завантажив список студентів на поселення')
            }}>
                Завантажити
            </Button>
            <div className="resultTable"></div>
            {finalData && <Table responsive striped bordered hover>
```

```
                <thead>
                <tr>
                  <th>ПІБ</th>
                  <th>Рік навчання</th>
                  <th>Рейтинговий бал</th>
                  <th>Пільги</th>
                </tr>
                </thead>
                <tbody>{finalData.map(el => (
                  <tr key={el._id}>
                    <td>{el.student[0].name}</td>
                    <td>{el.student[0].year}</td>
                    <td>{el.student[0].mark}</td>
                    <td>{el.student[0].privilege.map(priv => (
                      <p key={priv}>{priv}</p>
                    ))}</td>
                  </tr>
                ))}</tbody>
              </Table>}
            </Card>}
          </div>
    )
}

export default StudentList
```

## OneStudentUpload.js

```
import React, {useState} from "react";
import '../InputData.scss'
import {Accordion, Button, Form} from "react-bootstrap";
import {Rings} from "react-loader-spinner";
import {actionSave} from "../../../heplers/mapHelper";



const OneStudentUpload = () => {
    const [name, setName] = useState('')
    const [year, setYear] = useState(null)
    const [mark, setMark] = useState(null)
    const [privilege, setPrivilege] = useState([])
    const [isPending, setIsPending] = useState(false)

    const handlePrivileges = (e) => {
```

```
    let newPrivileges = [...privilege];
    if (e.target.checked) {
      newPrivileges = [...privilege, e.target.value];
    } else {
      newPrivileges.splice(privilege.indexOf(e.target.value), 1);
    }
    setPrivilege(newPrivileges);
  }

  const handleSubmit = (e) => {
    e.preventDefault()
    const student = [{name, year, mark, privilege}]

    setIsPending(true)
    fetch('http://localhost:3000/queue/add', {
      method: 'POST',
      headers: {"Content-Type": "application/json"},
      body: JSON.stringify(student)
    }).then(() => {
      console.log("post request")
      setIsPending(false)
    })
  }
  return (
    <Form onSubmit={handleSubmit}>
      <Form.Group className="mb-3" controlId="formAddStudent">
        <Form.Label>ПІБ студента</Form.Label>
        <Form.Control
          type="text"
          placeholder="Петренко Петро Петрович"
          required
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </Form.Group>

      <Form.Group className="mb-3" controlId="formAddStudent">
        <Form.Label>Рік навчання</Form.Label>
        <Form.Control
          type="number"
          placeholder="1"
```

```
                required
                value={year}
                onChange={(e) => setYear(e.target.value)}
              />
            </Form.Group>

            <Form.Group className="mb-3" controlId="formAddStudent">
              <Form.Label>Рейтинговий бал</Form.Label>
              <Form.Control
                type="number"
                placeholder="100"
                required
                value={mark}
                onChange={(e) => setMark(e.target.value)}
              />
            </Form.Group>

            <Form.Group className="mb-3" controlId="formBasicCheckbox">
              <Accordion>
                <Accordion.Item eventKey="0">
                  <Accordion.Header> <Form.Label>Оберіть пільги, які має студент
                    (необов'язково)</Form.Label></Accordion.Header>
                  <Accordion.Body>
                    <Form.Check type="checkbox" value={"Дитина з інвалідністю"}
                          onChange={handlePrivileges} label="Дитина з інвалідністю"/>
                    <Form.Check type="checkbox"
                          value={"Дитина-сирота, або дитина, позбавлена батьківського піклування"}
                          onChange={handlePrivileges}
                          label="Дитина-сирота, або дитина, позбавлена батьківського піклування"/>
                    <Form.Check type="checkbox" onChange={handlePrivileges}
                          value={"Дитина з багатодітної сім'ї"} label="Дитина з багатодітної сім'ї"/>
                    <Form.Check type="checkbox" onChange={handlePrivileges}
                          value={"Дитина загиблого військовослужбовця"}
                          label="Дитина загиблого військовослужбовця"/>
                    <Form.Check type="checkbox" onChange={handlePrivileges}
                          value={"Особи з інвалідністю I, II та III груп"}
                          label="Особи з інвалідністю I, II та III груп"/>
                    <Form.Check type="checkbox" onChange={handlePrivileges}
                          value={"Дитина з малозабезпеченої родини, у якої обидва батьки є особами з
інвалідністю"}
```

```
                label="Дитина з малозабезпеченої родини, у якої обидва батьки є особами з інвалідністю"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина з малозабезпеченої родини, у якої один з батьків особа з інвалідністю, а інший помер"}
                label="Дитина з малозабезпеченої родини, у якої один з батьків особа з інвалідністю, а інший помер"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина з малозабезпеченої родини, у якої мати з числа осіб з інвалідністю і виховує дитину без батька"}
                label="Дитина з малозабезпеченої родини, у якої мати з числа осіб з інвалідністю і виховує дитину без батька"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина з малозабезпеченої родини, у якої батько з числа осіб з інвалідністю, який виховує дитину без матері"}
                label="Дитина з малозабезпеченої родини, у якої батько з числа осіб з інвалідністю, який виховує дитину без матері"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина учасників бойових дій АТО, один з батьків якої загинув"}
                label="Дитина учасників бойових дій АТО, один з батьків якої загинув"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина учасників бойових дій АТО"}
                label="Дитина учасників бойових дій АТО"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина учасників бойових дій, які загинули або зникли безвісти на території інших держав"}
                label="Дитина учасників бойових дій, які загинули або зникли безвісти на території інших держав"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Внутрішньо переміщені особи"} label="Внутрішньо переміщені особи"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Чорнобильці (перша, друга та третя категорії)"}
                label="Чорнобильці (перша, друга та третя категорії)"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина шахтаря (стаж 15 років, або батьки-шахтарі загинули чи отримали інвалідність)"}
                label="Дитина шахтаря (стаж 15 років, або батьки-шахтарі загинули чи отримали інвалідність)"/>
            <Form.Check type="checkbox" onChange={handlePrivileges}
                value={"Дитина-призер міжнародних олімпіад"}
```

```
                        label="Дитина-призер міжнародних олімпіад"/>
                    <Form.Check type="checkbox" onChange={handlePrivileges} value={"Напівсирота"}
                        label="Напівсирота"/>
                </Accordion.Body>
            </Accordion.Item>
        </Accordion>
    </Form.Group>
    {!isPending && <Button variant="primary" type="submit" onClick={() =>
        actionSave(`Користувач додав студента ${name} ${year} року навчання`)}>
        Додати студента
    </Button>}
    {isPending && <Button variant="primary" type="submit">
        Додавання
    </Button>}
</Form>

)
}

export default OneStudentUpload
```

## StudentUploader.js

```
import React, {useState} from 'react';
import {Button, Form} from "react-bootstrap";
import * as XLSX from 'xlsx'
import {actionSave} from "../../../heplers/mapHelper";

const StudentUploader = () => {
    const [isSelected, setIsSelected] = useState(false);
    const [studentData, setStudentData] = useState([])
    let data = []

    const changeHandler = async (e) => {
        console.log(e)
        e.preventDefault();
        if (e.target.files) {
            const reader = new FileReader();
            reader.onload = (e) => {
                const result = e.target.result;
                const workbook = XLSX.read(result, {type: "array"});
                const sheetName = workbook.SheetNames[0];
                const worksheet = workbook.Sheets[sheetName];
```

```
            const json = XLSX.utils.sheet_to_json(worksheet)
            console.log("first json", json)
            data = json
        };
        await reader.readAsArrayBuffer(e.target.files[0]);
    }
    setIsSelected(true);
}

const mapJsonData = (students) => {
    return students.map((el) => {
        if (el.privilege) {
            el.privilege = el.privilege.split(", ")
        } else {
            el.privilege = []
        }
    })
}

const handleSubmission = () => {
    mapJsonData(data)
    console.log(data)
    fetch('http://localhost:3000/queue/add', {
        method: 'POST',
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify(data)
    }).then(() => {
        console.log("post request")
    })
};

return (
    <div>
        <Form.Group controlId="formFileLg" className="mb-3">
            <Form.Control type="file" size="lg" onChange={changeHandler}/>
        </Form.Group>
        {isSelected ? (
            <p>Файл обрано</p>
        ) : (
            <p>Оберіть файл</p>)}
        <div>
```

```
            <Button onClick={(e) => {
                handleSubmission(e)
                actionSave('Користувач додав список студентів')
            }}>Додати</Button>
        </div>
    </div>
  )

}

export default StudentUploader
```

## Rooms.js

```
import React, {useState} from "react";
import {Accordion, Tab, Tabs} from "react-bootstrap";
import AvailableRooms from "./AvailableRooms";
import AddRooms from "./AddRooms";
import '.././InputData.scss'


const Rooms = () => {
  const [key, setKey] = useState('availableRooms');
  return (
    <Accordion.Item eventKey="1">
      <Accordion.Header>Управління місцями</Accordion.Header>
      <Accordion.Body>
        <Tabs
          id="controlled-tab-example"
          activeKey={key}
          onSelect={(k) => setKey(k)}
          className="mb-3"
        >
          <Tab eventKey="availableRooms" title="Наявні місця">
            <AvailableRooms/>
          </Tab>
          <Tab eventKey="addRooms" title="Додати місця">
            <AddRooms/>
          </Tab>
        </Tabs>
      </Accordion.Body>
    </Accordion.Item>
  )
```

```
}

export default Rooms
```

## AddRooms.js

```
import React, {useState} from "react";
import {Button, Form, FormGroup} from "react-bootstrap";
import {actionSave} from "../../../heplers/mapHelper";


const AddRooms = () => {
    const [dormitoryN, setDormitoryN] = useState(null)
    const [nPlaces, setNPlaces] = useState(null)
    const [isPending, setIsPending] = useState(false)

    const handleSubmit = (e) => {
        e.preventDefault()
        const data = {dormitoryN, nPlaces}
        console.log(data)
        setIsPending(true)
        fetch('http://localhost:3000/room/add', {
            method: 'POST',
            headers: {"Content-Type": "application/json"},
            body: JSON.stringify(data)
        }).then(() => {
            console.log("post request")
            setIsPending(false)
            window.location.reload();
        })
    }

    return (
        <Form onSubmit={handleSubmit}>
            <Form.Group className="mb-3" controlId="addRooms">
                <Form.Label>Номер гуртожитка</Form.Label>
                <Form.Control type="number"
                              placeholder=""
                              value={dormitoryN}
                              onChange={(e) => setDormitoryN(e.target.value)}/>
            </Form.Group>
            <FormGroup className="mb-3" controlId="addPlaces">
```

```
                <Form.Label>Кількість місць</Form.Label>
                <Form.Control type="number"
                        placeholder=""
                        value={nPlaces}
                        onChange={(e) => setNPlaces(e.target.value)}/>
            </FormGroup>
            {!isPending && <Button variant="primary" type="submit" onClick={() =>
              actionSave(`Користувач додав гуртожиток №${dormitoryN} з ${nPlaces} доступними місцями`)}>
              Додати
            </Button>}
            {isPending && <Button variant="primary" type="submit">
              Додавання...
            </Button>}
          </Form>
        )
      }

      export default AddRooms
```

## AvailableRooms.js

```
import React, {useEffect, useState} from "react";
import {Rings} from 'react-loader-spinner'
import {Table} from "react-bootstrap";
import useFetch from "../../../heplers/useFetch";


const AvailableRooms = () => {
  const {data, isPending} = useFetch('http://localhost:3000/rooms')
  return (
    <div>
      {data && <Table responsive striped bordered hover>
        <thead>
        <tr>
          <th>Номер гуртожитка</th>
          <th>Кількість доступних місць</th>
        </tr>
        </thead>
        <tbody>{data.map(el => (
          <tr key={el._id}>
            <td>{el.dormitoryN}</td>
            <td>{el.nPlaces}</td>
```

```
                </tr>
            ))}</tbody>
        </Table>}
        {isPending && <Rings/>}
      </div>
    )
}

export default AvailableRooms
```

## Quotas.js

```
import React, {useState} from "react";
import {Accordion, Tab, Tabs} from "react-bootstrap";
import ExistedQuotas from "./ExistedQuotas";
import AddQuotas from "./AddQuotas";
import '../../InputData.scss'

const Quotas = () => {
    const [key, setKey] = useState('existingQuotas');
    return (
        <Accordion.Item eventKey="2">
            <Accordion.Header>Управління квотами</Accordion.Header>
            <Accordion.Body>
                <Tabs
                    id="controlled-tab-example"
                    activeKey={key}
                    onSelect={(k) => setKey(k)}
                    className="mb-3"
                >
                    <Tab eventKey="existingQuotas" title="Існуючі квоти" activeKey={key}>
                        <ExistedQuotas/>
                    </Tab>
                    <Tab eventKey="addQuotas" title="Додати квоти" activeKey={key}>
                        <AddQuotas/>
                    </Tab>
                </Tabs>
            </Accordion.Body>
        </Accordion.Item>
    )
}

export default Quotas
```

## AddQuotas.js

```
import React, {useState} from "react";
import {Button, Form, FormGroup} from "react-bootstrap";
import {Rings} from "react-loader-spinner";
import {actionSave} from "../../../heplers/mapHelper";

const AddQuotas = () => {
    const [faculty, setFaculty] = useState(null)
    const [year, setYear] = useState(null)
    const [percent, setPercent] = useState(null)
    const [isPending, setIsPending] = useState(false)

    const handleSubmit = (e) => {
        e.preventDefault()
        const data = {faculty, year, percent}
        console.log(JSON.stringify(data))
        setIsPending(true)
        fetch('http://localhost:3000/quota/add', {
            method: 'POST',
            headers: {"Content-Type": "application/json"},
            body: JSON.stringify(data)
        }).then(() => {
            console.log("post request")
            setIsPending(false)
            window.location.reload();
        })
    }

    return (
        <Form onSubmit={handleSubmit}>
            <Form.Group className="mb-3" controlId="setFaculty">
                <Form.Label>Факультет</Form.Label>
                <Form.Control type="text"
                              placeholder=""
                              value={faculty}
                              onChange={(e) => setFaculty(e.target.value)}/>
            </Form.Group>
            <FormGroup className="mb-3" controlId="setYear">
                <Form.Label>Рік навчання</Form.Label>
                <Form.Control type="number"
                              placeholder=""
```

```
                        value={year}
                        onChange={(e) => setYear(e.target.value)}/>
                </FormGroup>
                <FormGroup className="mb-3" controlId="setPercent">
                  <Form.Label>Відсоток</Form.Label>
                  <Form.Control type="number"
                        placeholder=""
                        value={percent}
                        onChange={(e) => setPercent(e.target.value)}/>
                </FormGroup>
                {!isPending && <Button variant="primary" type="submit" onClick={() =>
                  actionSave(`Користувач додав ${percent}% квоту для ${year} року навчання для факультету ${faculty}`)}>
                  Додати
                </Button>}
                {isPending && <Button variant="primary" type="submit">
                  Додавання
                </Button>}
              </Form>
            )
          }

          export default AddQuotas
```

## ExistedQuotas.js

```
import React from "react";
import {Rings} from 'react-loader-spinner'
import {Button, Table} from "react-bootstrap";
import useFetch from "../../../heplers/useFetch";
import {actionSave} from "../../../heplers/mapHelper";


const ExistedQuotas = () => {
  const {data, isPending} = useFetch('http://localhost:3000/quotas')
  const refresh = () => {
    window.location.reload();
  }
  async function deleteQuota(id) {
    await fetch(`http://localhost:3000/quota/delete/${id}`, {
      method: "DELETE",
```

```
    });
  }
  return (
    <div>
      {data && <Table responsive striped bordered hover on>
        <thead>
        <tr>
          <th>Факульте</th>
          <th>Рік навчання</th>
          <th>Відсоток</th>
          <th>Видалення</th>
        </tr>
        </thead>
        <tbody>{data.map(el => (
          <tr key={el._id}>
            <td>{el.faculty}</td>
            <td>{el.year}</td>
            <td>{el.percent}</td>
            <td><Button onClick={
              (e) => {
                deleteQuota(el._id).then(refresh)
                actionSave(`Користувач видалив ${el.percent} % квоту для ${el.year} року навчання для факультету ${el.faculty}`)
              }
            }>Видалити</Button>
            </td>
          </tr>
        ))}</tbody>
      </Table>}
      {isPending && <Rings/>}
    </div>
  )
}

export default ExistedQuotas
```

## ДОДАТОК Б

Лістинг Back-end частини

### index.js

```
import fetch from 'node-fetch'
import express from 'express'
import mongoose from 'mongoose'
import router from './routes/settlement.js'
import path from "path";
import cors from "cors";

const PORT = process.env.PORT || 3000
const app = express()
const __dirname = path.resolve()

app.use(cors())
app.use(express.urlencoded({extended: true}))
app.use(express.json())
app.use(router)
app.set('view engine', 'ejs')

async function start(){
   try {
      await
mongoose.connect('mongodb+srv://Danza:NsRyN63XD7RsQTWG@cluster0.4gag7ju.mongodb.net/diplom', {
         useNewUrlParser: true
      })
      app.listen(PORT, () => {
         console.log('Service has been started')
      })
   } catch (e){
      console.log(e)
   }
}
start()
```

### settlement.js

```
import { Router } from 'express'
import path from 'path'
import {fileURLToPath} from 'url'
import * as request from "../controlers/requests.js";
```

```
const router = new Router()
const __filename = fileURLToPath(import.meta.url)
const __dirname = path.resolve(__filename)

router.get('/', (req, res) =>{
   res.render('index', {title: 'Main Page', active: 'index'})
})

router.get('/queue', request.getAllStudentsInQueue)

router.get('/rooms', request.getAllRooms)

router.get('/quotas', request.getAllQuotas)

router.get('/actions', request.getAllActions)

router.get('/privilege', request.getAllPrivileges)

router.get('/privilege/:id', request.getPrivilegeById)

router.get('/quota/:id', request.getQuotaById)

router.get('/quotas/:faculty', request.getAllQuotasByFaculty)

router.post('/quota/add', request.addQuota)

router.post('/room/add', request.addRoom)

router.post('/action/add', request.addAction)

router.post('/queue/add', request.addStudentToQueue)

router.put('/room/update/:id', request.updateRoom)

router.put('quota/update/:id', request.updateQuota)

router.delete('/room/delete/:id', request.deleteRoom)

router.delete('/quota/delete/:id', request.deleteQuota)

router.delete('/queue/delete/:id', request.deleteStudentFromQueue)
```

```
router.delete('/action/delete/:id', request.deleteAction)


export default router
```

## requests.js

```
import Room from '../models/Room.js'
import Quota from '../models/Quota.js'
import Queue from "../models/Queue.js";
import Action from "../models/Action.js";
import Privilege from "../models/Privilege.js";

export const getAllStudentsInQueue = async (req, res) => {
    try {
        await Queue.find().exec((err, students) => {
            return res.json(students)
        })
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}

export const getAllRooms = async (req, res) => {
    await Room.find().exec((err, rooms) => {
        return res.json(rooms)
    })
}

export const getAllQuotas = async (req, res) => {
    try {
        await Quota.find().exec((err, quotas) => {
            return res.json(quotas)
        })
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}

export const getAllActions = async (req, res) => {
    try {
        await Action.find().exec((err, actions) => {
```

```
            return res.json(actions)
        })
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}

export const getAllPrivileges = async (req, res) => {
    try {
        await Privilege.find().exec((err, privileges) => {
            return res.json(privileges)
        })
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}

export const getPrivilegeById = async (req, res) => {
    try {
        return res.json(await Privilege.findById(req.params.id))
    } catch (error) {
        res.status(500).json({message: error.message})
    }
}

export const getQuotaById = async (req, res) => {
    try {
        return res.json(await Quota.findById(req.params.id))
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}

export const getAllQuotasByFaculty = async (req, res) => {
    try {
        return res.json(await Quota.find().where({faculty: req.params.faculty}))
    } catch (err) {
        return res.status(500).json({message: err.message})
    }
}
```

```
export const addQuota = async (req, res) => {
    let quota = new Quota({
        faculty: req.body.faculty,
        year: req.body.year,
        percent: req.body.percent
    })
    try {
        quota = await quota.save()
        res.send(quota)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const addRoom = async (req, res) => {
    let room = new Room({
        dormitoryN: req.body.dormitoryN,
        nPlaces: req.body.nPlaces
    })
    try {
        room = await room.save()
        res.send(room)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const addAction = async (req, res) => {
    let action = new Action({
        name: req.body.name,
        time: req.body.time
    })
    try {
        action = await action.save()
        res.send(action)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const addStudentToQueue = async (req, res) => {
```

```
    let priorityValue = 0;
    try {
        for (const student of req.body) {
            priorityValue = 0
            if (student.year === 1 || student.year === 5) {
                priorityValue += 1
            }
            if (student.year === 1) {
                priorityValue += student.mark / 200
            } else {
                priorityValue += student.mark / 100
            }
            let newPriorityValue = await calculatePrivilege(student.privilege, priorityValue)
            await Queue.create({
                    student: [{
                        name: student.name,
                        year: student.year,
                        mark: student.mark,
                        privilege: student.privilege,
                        priority: newPriorityValue
                    }]
                }
            )
        }
        await res.sendStatus(200)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

async function calculatePrivilege(privileges, priorityValue) {
    await Privilege.find({}, (err, data) => {
        data.map((priv) => {
            if (privileges.includes(priv.name)) {
                priorityValue += priv.rank
            }
        })
    }).clone()
    return priorityValue
}
```

```
export const updateRoom = async (req, res) => {
    try {
        let room = await Room.findById(req.params.id)
        if (req.body.dormitoryN) {
            room.dormitoryN = req.body.dormitoryN
        }
        if (req.body.nPlaces) {
            room.nPlaces = req.body.nPlaces
        }
        await room.save()
        res.json(room)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const updateQuota = async (req, res) => {
    try {
        let quota = await Quota.findById(req.params.id)
        if (req.body.percent) {
            quota.percent = req.body.percent
        }
        if (req.body.name) {
            quota.name = req.body.name
        }
        if (req.body.year) {
            quota.year = req.body.year
        }
        await quota.save()
        res.json(quota)
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const deleteRoom = async (req, res) => {
    try {
        await Room.findByIdAndDelete(req.params.id)
        res.send('Room has been deleted..')
    } catch (error) {
        res.status(400).json({message: error.message})
```

```
    }
}

export const deleteQuota = async (req, res) => {
    try {
        await Quota.findByIdAndDelete(req.params.id)
        res.send('Quota has been deleted..')
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}

export const deleteStudentFromQueue = async (req, res) => {
    try {
        await Queue.findByIdAndDelete(req.params.id)
        res.send('Student has been deleted from the queue..')
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}



export const deleteAction = async (req, res) => {
    try {
        await Action.findByIdAndDelete(req.params.id)
        res.send('Action has been deleted from the queue..')
    } catch (error) {
        res.status(400).json({message: error.message})
    }
}
```

## Action.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
    name: {
        type: String
    },
    time: {
        type: String
    }
```

```
})

export default mongoose.model('Action', schema)
```

## Privilege.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
   name: {
      type: String
   },
   rank: {
      type: Number
   }

})

export default mongoose.model('Privileges', schema)
```

## Queue.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
   student: [{
      name: {
         type: String,
         required: true
      },
      year: {
         type: Number,
         required: true,
         max: 7,
         min: 1
      },
      mark: {
         type: Number,
         required: true,
         max: 200,
         min: 30
      },
      privilege: [{
         type: String
```

```
        }],
        priority: {
            type: Number
        }
    }]
})

export default mongoose.model('Queue', schema)
```

## Quota.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
    faculty: {
        type: String,
        required: true
    },
    year: {
        type: Number
    },
    percent: {
        type: Number
    }
})

export default mongoose.model('Quota', schema)
```

## Room.js

```
import mongoose from 'mongoose'

const schema = new mongoose.Schema({
    dormitoryN: {
        type: Number
    },
    nPlaces: {
        type: Number,
        required: true
    }
})

export default mongoose.model('Room', schema)
```