

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова  
Навчально-науковий інститут комп'ютерних наук та управління проектами  
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

\_\_\_\_\_

к.т.н, доц. Михелєв І.Л.

" \_\_\_\_ " \_\_\_\_\_ 2022 р.

*КВАЛІФІКАЦІЙНА РОБОТА*

на здобуття ступеня вищої освіти "бакалавр"

на тему:

«Розробка підсистеми "Студент" інформаційної системи кафедри ІУСТ»

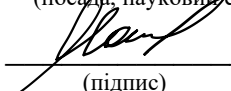
Виконав:

студент групи 4144

 Недо А.О.  
(підпис)

Керівник роботи: старший  
викладач кафедри ІУСТ НУК

(посада, науковий ступінь вчене звання)

 Маршак О.І.  
(підпис)

м. Миколаїв – 2022 р.

# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова  
Навчально-науковий інститут комп'ютерних наук та управління проектами  
Кафедра інформаційних управляючих систем та технологій  
Спеціальність 126 "Інформаційні системи та технології"  
Освітня програма "Інформаційні системи та технології"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

к.т.н, доц. НУК Книрик

Н.Р.

" \_\_\_\_ " \_\_\_\_\_ 2022 р.

## *ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ* на здобуття ступеня вищої освіти "бакалавр"

Студенту Недо Антону Олександровичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка підсистеми "Студент" інформаційної системи кафедри ІУСТ

Керівник роботи: Маршак Олена Іллівна

Затверджені наказом ректора № 269-уч від "18" травня 2022 року.

2. Термін подання роботи: 20 червня 2022 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем

4. Перелік питань, що належать до розробки (найменування розділів): Вступ, Розробка концепції підсистеми «Студент» інформаційної системи кафедри ІУСТ, розробка проектних рішень, реалізація проекту, охорона праці

5. Перелік презентаційних матеріалів: Мета, Організаційно-функціональна структура, Клієнт-серверна архітектура, Структура підсистеми, Інформаційний взаємозв'язок між компонентами підсистеми, Діаграма переходів станів, Концептуальна модель даних Core Service, Концептуальна модель даних User Service, Структура серверного програмного комплексу, Core Service API, User Service API, Веб додаток

## 6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата   |                  |
|--------|-------------------------------------------|----------------|------------------|
|        |                                           | завдання видав | завдання прийняв |
| 1      | Маршак О.І.                               | 14.02. 2022    | 21.03.2022       |
| 2      | Маршак О.І.                               | 21.03.2022     | 21.04.2022       |
| 3      | Маршак О.І.                               | 21.04.2022     | 01.06.2022       |
| 4      | Гурець Н.В.                               | 01.06.2022     | 07.06.2022       |

7. Дата видачі завдання 14 лютого 2020 р.

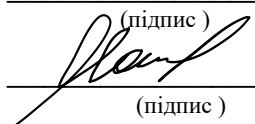
## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи)             | Строк виконання етапів роботи | Примітка |
|-------|------------------------------------------------------|-------------------------------|----------|
| 1     | Вивчення і опис предметної області                   | 21 березня                    |          |
| 2     | Розробка проектних рішень                            | 21 квітня                     |          |
| 3     | Реалізація проектних рішень                          | 1 червня                      |          |
| 4     | Вирішення питань охорони праці                       | 7 червня                      |          |
| 5     | Виконання графічних документів                       | 13 червня                     |          |
| 6     | Оформлення пояснювальної записки                     | 19 червня                     |          |
| 7     | Подання дипломної роботи (ДР) на кафедральний захист | 20 червня                     |          |
| 8     | Подання ДР рецензенту                                | 20 червня                     |          |
| 9     | Подання ДР на державний захист                       | 27 червня                     |          |
| 10    |                                                      |                               |          |

Студент

Керівник роботи

  
 \_\_\_\_\_  
 (підпис)

  
 \_\_\_\_\_  
 (підпис)

Недо А.О.  
 (ПІБ)

Маршак О.І.  
 (ПІБ)

## АНОТАЦІЯ

Тема дипломного проекту: «Розробка підсистеми "Студент" інформаційної системи кафедри ІУСТ».

Актуальність даної теми зумовлена сучасними тенденціями розвитку технологій, що потребують підвищення ефективності.

Метою створення підсистеми є спрощення способів отримання інформації студентам, а також покращення зручності використання інформації навчальної кафедри.

В ході роботи було проведено аналіз предметної області, сформульовані вимоги до автоматизованої системи, виявлення існуючих недоліків, розроблення концепції та технічного проекту автоматизованої системи, а також було реалізовано підсистему «Студент» інформаційної системи кафедри ІУСТ.

Робота виконана на 90 сторінок, містить 17 рисунків, 4 таблиці, 3 додатку, які містять інструкцію з використання.

Робота виконана українською мовою.

## ANNOTATION

Theme of the diploma project: "Development of the subsystem" Student "of the information system of the Department of IUST".

The relevance of this topic is due to current trends in technology that need to increase efficiency.

The purpose of creating a subsystem is to simplify the methods of obtaining information for students, as well as to improve the usability of information of the department.

In the course of the work the analysis of the subject area was carried out, the requirements to the automated system were formulated, the existing shortcomings were identified, the concept and technical design of the automated system were developed, and the "Student" information system subsystem of the IUST department was implemented.



The work is done on 90 pages, contains 17 figures, 4 tables, 3 appendices, which contain instructions for use.

The work is done in Ukrainian language.

## ЗМІСТ

|                                                                                            |    |
|--------------------------------------------------------------------------------------------|----|
| Вступ .....                                                                                | 8  |
| Розділ 1. Розробка концепції підсистеми «Студент» інформаційної системи кафедри ІУСТ ..... | 10 |
| 1.1. Аналіз предметної області .....                                                       | 10 |
| 1.2. Формування вимог до підсистеми «Студент» .....                                        | 15 |
| 1.3. Розробка концепції підсистеми «Студент» інформаційної системи кафедри ІУСТ. ....      | 18 |
| 1.4. Постановка задачі .....                                                               | 20 |
| Розділ 2. Розробка проектних рішень .....                                                  | 27 |
| 2.1 Загальний опис підсистеми «Студент».....                                               | 27 |
| 2.2. Рішення з інформаційного забезпечення.....                                            | 29 |
| 2.3 Рішення з технічного забезпечення .....                                                | 41 |
| 2.4 Рішення з програмного забезпечення .....                                               | 42 |
| 3 Реалізація проекту .....                                                                 | 43 |
| 3.1. Реалізація компонентів підсистеми .....                                               | 43 |
| 3.2. Екранні форми.....                                                                    | 50 |
| 3.3. Розробка робочої документації .....                                                   | 51 |
| 4 Охорона праці .....                                                                      | 52 |
| 4.1 Вступ .....                                                                            | 52 |
| 4.2. Фактори, що впливають на людину .....                                                 | 54 |
| 4.3. Заходи безпеки .....                                                                  | 57 |
| 4.4. Вимоги щодо забезпечення пожежної безпеки .....                                       | 57 |

|                                                 |    |
|-------------------------------------------------|----|
|                                                 | 7  |
| Висновки за розділом 4 .....                    | 58 |
| Висновки.....                                   | 59 |
| Список використаних джерел.....                 | 60 |
| Додаток А Технічне завдання.....                | 62 |
| Додаток Б Загальний опис системи .....          | 66 |
| Додаток В Інструкція користувача .....          | 68 |
| Додаток Г Програма і методика випробувань ..... | 70 |
| Додаток Д Фрагменти лістингу .....              | 73 |

## ВСТУП

На даний момент у багатьох університетах застарілий спосіб взаємодії та зворотного зв'язку зі студентами. У століття технологій, розвитку інформаційних систем та технологій стає все більш гострим питання про належний рівень цифрової інтеграції у навчальному процесі вищих навчальних закладів і не тільки.

Університет є однією з найскладніших структур, що поєднує в собі безліч різних сфер освітньої та наукової діяльності, що дозволяє готувати майбутніх спеціалістів. Інтеграція цифрової технології дозволить покращити процес взаємодії таких навчально-структурних одиниць, як кафедри, зі своїми студентами.

Зважаючи на недоліки комунікативних зв'язків розповсюдження повідомлень про події кафедри постає необхідність у розробці підсистеми «Студент» інформаційної системи кафедри інформаційних управляючих систем та технологій національного університету кораблебудування імені адмірала Макарова.

Підсистема дозволить налагодити взаємодію студентів із викладачами, дасть можливість отримати зворотний зв'язок від здобувачів, а також швидко повідомити про різні події. Для студентів надається широкий спектр можливостей для отримання інформації, способів її візуалізації та зберігання.

**Мета** роботи – підвищити ефективність організації освітнього процесу на кафедрі інформаційних управляючих систем та технологій через автоматизацію розкладу занять.

**Об'єктом** автоматизації є розклади подій кафедри.

**Предметом** автоматизації є інформаційні процеси, пов'язані з оптимізацією навчального розкладу кафедри.

Для досягнення цієї мети необхідно вирішити ряд завдань:

- виконати аналіз предметної області і визначити недоліки існуючих систем та рішень;
- розробити концепцію «Підсистеми Студент»;
- розробити проектні рішення;
- розробити технічне завдання;
- провести необхідні дослідження з охорони праці.

## РОЗДІЛ 1. РОЗРОБКА КОНЦЕПЦІЇ ПІДСИСТЕМИ «СТУДЕНТ» ІНФОРМАЦІЙНОЇ СИСТЕМИ КАФЕДРИ ІУСТ

### 1.1. Аналіз предметної області

#### 1.1.1. Характеристика кафедри ІУСТ

Сучасний заклад вищої освіти є досить складною соціальною та освітньою системою. Організація високоякісної підготовки майбутніх спеціалістів та високої їх конкурентоспроможності на ринку праці, їх професійної та соціальної мобільності вимагає не лише ефективної організації навчально-виховного процесу, а й цілого комплексу заходів щодо його належного забезпечення.

Кафедра інформаційних управляючих систем та технологій Національного університету кораблебудування імені адмірала Макарова здійснює підготовку фахівців за освітніми програмами трьох спеціальностей бакалаврського та магістерського рівнів (табл. 1.1):

**Таблиця 1.1**

**Освітні програми кафедри ІУСТ**

| Спеціальність                          | Освітня програма                   | Рівень   |
|----------------------------------------|------------------------------------|----------|
| 122 Комп'ютерні науки                  | Комп'ютерні науки                  | бакалавр |
|                                        | Комп'ютерні науки                  | магістр  |
| 124 Системний аналіз                   | Системний аналіз                   | бакалавр |
|                                        | Системний аналіз                   | магістр  |
| 126 Інформаційні системи та технології | Інформаційні системи та технології | бакалавр |
|                                        | Інформаційні системи та технології | магістр  |

У 2021-2022 н.р. за кафедрою закріплено 19 груп денної форми навчання. Крім того, викладачі кафедри здійснюють підготовку фахівців більшості освітніх програм НУК денної та заочної форм навчання. Заняття проводяться у трьох корпусах та у філіях університету. Розклади навчальних занять та екзаменаційних сесій формуються диспетчерською службою НУК та розміщуються в холах

навчальних корпусів, на кафедрах та на інформаційних ресурсах кафедри та університету. Інформаційними ресурсами кафедри є сайт <http://iust.nuos.edu.ua/>, сторінки кафедри у соціальних мережах Фейсбук <https://www.facebook.com/kafedra.IUST> та Інстаграм <https://www.instagram.com/kafedra.iust/>, телеграм канал <https://t.me/iustnuk>.

Також викладачі кафедри проводять консультації з своїх дисциплін. Графік консультацій розміщується на кафедрі та кафедральних ресурсах.

Велика увага приділяється олімпіадам з програмування. З 2018 р. Національний університет кораблебудування імені адмірала Макарова є базовим з проведення чвертьфінал Міжнародної студентської олімпіади з програмування АСМ-ICPC у Південному регіоні. Від університету приймають участь до 10 команд щорічно. Ще у 2020 році за ініціативою викладачів та співробітників кафедри започатковано конкурс зі спортивного програмування «Кубок К.В. Кошкіна». Тренування студентів викладачами кафедри під керівництвом координатора олімпіад Євгена Беркунського здійснюється як на спеціальному ресурсі E-Olymp так і в навчальних аудиторіях за окремим розкладом, який розміщено на особистому сайті Євгена Беркунського та на кафедрі.[1]

До процесу підготовки фахівців залучаються стейкхолдери, представники компаній приймають активну участь у заходах, які організовують на кафедрі, а саме Ярмарки вакансій, дні відкритих дверей, відкриті лекції тощо. Оголошення про дати проведення цих заходів розміщуються в холах навчальних корпусів, на кафедрах та на інформаційних ресурсах кафедри та університету.[1]

Крім того, слід зауважити що розклади є динамічними та можуть змінюватися, особливо на початку навчального семестру.

Актуальність теми дослідження також обумовлена формуванням нового тренду у розвитку системи освіти – оцифрування освіти, що змінює парадигму освітніх відносин.

Процеси оцифрування освіти отримали прискорення на тлі карантинних заходів весни 2020 р. та масового переходу учасників освітніх стосунків у формат онлайн. Ця тема вимагає особливого вивчення, оскільки є унікальною у контексті

трансформації відносин у будь-якій сфері діяльності, тим більше у контексті виходу людських відносин за межі аналогового світу офлайн у світ онлайн. [3]

### **1.1.2. Опис об'єкту автоматизації**

На даний момент студент після вступу до університету не має достатньої кількості інформації в оцифрованому та зручному для засвоєння виду. Сайт університету не завжди має достатньої кількості та якості інформації, а також актуальності, а також не має засобів зворотного зв'язку зі студентом у режимі реального часу. Зменшення часу та спрощення способу отримання інформації вплине на початкові етапи взаємодії із здобувачами освіти. Розглянемо на прикладі розкладу, які учні отримують на початку кожного нового навчального семестру, він формується та публікується у фізичному вигляді на дошці оголошень біля кафедри, або в аудиторії кафедри, та поширюється у цифровому форматі зображення, що виключає структурування сутностей розкладу, таких як предмети, викладачі, аудиторії, а також додаткова інформація, пов'язана з проведенням заняття (наприклад, посилання на громадський інформаційний ресурс викладача). Також розклад може змінюватися, доповнятися, або зовсім деякі заняття скасовуються, у звичайному випадку куратор інформує студентів за заздалегідь встановленими каналами зв'язку, таким як месенджери, номери телефонів, або іншим, що забезпечує отримання подій у реальному часі. [1]

Не існує єдиного ресурсу де було би розміщено інформацію про усі події, які відбуваються на кафедрі. Куратор та староста академічної групи ще додатково повідомляють студентам про зміни у розкладі та/або нові заходи. У такій ситуації відвідуваність занять, консультацій та заходів залежить від свідомості студента та його здібностей самоорганізації. Агрегація повідомлень за джерелами зведена у таблиці 1.2



**Агрегація повідомлень про події**

| <b>Повідомлення</b>               | <b>Джерело</b>            | <b>Розташування</b>                                                                 |
|-----------------------------------|---------------------------|-------------------------------------------------------------------------------------|
| Початок семестру                  | Графік освітнього процесу | Сайт НУК, паперовий на кафедрі                                                      |
| Кінець семестру                   | Графік освітнього процесу | Сайт НУК, паперовий на кафедрі                                                      |
| Тижневий розклад занять студентів | Розклад занять            | Сайт НУК, паперовий у холі                                                          |
| Екзаменаційна сесія               | Розклад екзаменів         | Сайт НУК, паперовий у холі                                                          |
| Розклад занять викладачів         | Розклад занять            | Сайт кафедри, паперовий на кафедрі                                                  |
| Зустрічі з куратором              | Оголошення                | Телеграм група                                                                      |
| Консультації                      | Розклад консультацій      | Індивідуальні ресурси викладачів, паперовий на кафедрі                              |
| Профорієнтаційні заходи           | Оголошення                | Інформаційні ресурси кафедри                                                        |
| Тренування з олімпіад             | Розклад тренувань         | Індивідуальні ресурси викладачів, паперовий на кафедрі                              |
| Практики                          | Графік освітнього процесу | Сайт НУК, паперовий на кафедрі                                                      |
| Конференції                       | Оголошення                | Інформаційні ресурси кафедри                                                        |
| Студентські заходи                | Оголошення                | Студентські пабліки                                                                 |
| Позанавчальні заходи              | Оголошення                | Індивідуальні ресурси викладачів,                                                   |
| Зміни розкладів                   | Оголошення                | Індивідуальні ресурси викладачів, інформаційні ресурси кафедри, студентські пабліки |

### **1.1.3. Обґрунтування необхідності створення підсистеми «Студент»**

На кафедрі використовують застаріли способи інформування студентів стосовно розкладу, змін у навчальному графіку, тощо. Використання розкладу заповненого на папері і відсутність централізованого контролю над внесенням змін, довгостроковим збереженням інформації у базах даних й інформуванням про ці зміни загострює недоліки незалежної комунікації студента із ресурсною базою. Наприклад це може призводити до несвоєчасного інформування про зміну розкладу, або складнощам щодо отримання контактної інформації. Для студентів, які тільки почали навчання важливо отримати потрібну інформацію в зручній формі з урахуванням розвитку тенденції дистанційного навчання.

Після аналізу комунікативних зав'язків було виділено такі недоліки:

- Повільне поширення інформації;
- Недостатнє використання цифрових технологій;
- Недостатня інформативність розкладу;
- Відсутність порталу для отримання зворотного зв'язку студентом;
- Витрати часу на створення паперового розкладу та його зміни;
- Витрати часу для децентралізованого інформування студенті щодо будь яких критичних змін у розкладах;
- Відсутність наочного календаря подій та запланованих заходів.

Узагальнивши ці недоліки, для їх усунення було прийнято рішення створити інформаційну підсистему «Студент» для учасників студентського життя на кафедрі під час навчання.

Створення комунікативної системи дозволить оптимізувати способи передачі інформації студентам, а також систематизує збереження корисних даних для всіх. Підсистема має незалежну архітектуру що дозволяє її інтегрувати й використовувати на ряду с сайтом кафедри, або університету. Також підсистема має загальну ресурсну базу даних що дозволяє додатково використовувати дані у інших суміжних системах.

## **1.2. Формування вимог до підсистеми «Студент»**

### **1.2.1. Аналіз функцій підсистеми «Студент»**

Створення підсистеми призначене для автоматизації та оптимізації процесів комунікації та отримання потрібної для студенту інформації. Метою створення підсистеми є не тільки централізація інформації до студенту, але й можливість зворотного зв'язку що підвищить ефективність виховних процесів у ЗВО.

У результаті аналізу діяльності учасників взаємодій на кафедрі були сформовані певні вимоги до підсистеми «Студент», також функції и задачі які повинна виконувати система.

На основі цього, система повинна виконувати такі функції:

- Облік інформації кафедри для студентів;
- Ідентифікація користувачів;
- Формування запитів за різними критеріями;
- Повідомлення користувачів про події;
- Створення нагадувань для певних груп, або студентів за участю розкладу;
- Інтеграція з основним сайтом кафедри ЗВО.

Для використання і роботи підсистеми треба розподілити користувачів на такі категорії:

- Гість — має можливість отримання інформації, формування запитів и отримувати загальні повідомлення та новини, переглядати розклади груп).
- Авторизований користувач (Студент) — має можливість отримання персональних повідомлень, перегляду нагадувань, отримання персональних повідомлень й участь в інтерактивних заходах та зворотному зв'язку.
- Викладач — має можливість взаємодіяти зі своїми групами (створювати календарні події, повідомлення, новини, зміни в розкладі, створення нових елементів начальних процесів тощо).

- Адміністратор – має можливість додавання, редагування та видалення будь якої інформації, також обслуговування, моніторинг роботи системи, використання системних ресурсів, налаштування зовнішніх інтеграцій із системою.

Користувач має володіти навичками роботи с інтерфейсами веб браузерів та веб сайтів.

### **1.2.2. Аналіз системних вимог**

Окрім функціональних вимог підсистеми, треба врахувати вимоги до використовуюваного оточення системи в загальному вигляді:

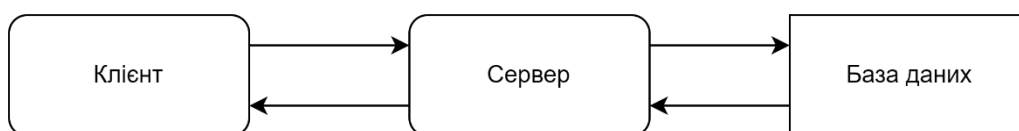
- Простий і водночас зручний графічний інтерфейс кінцевого користувача;
- Можливість обслуговування значної кількості користувачів;
- Розмежування прав користувачів;
- Надійність й безвідмовність;
- Конфіденційність даних користувачів, а також внутрішніх даних системи;
- Робота у глобальній мережі інтернету для можливості доступу з будь якого розташування користувача;
- Можливість розширеної інтеграції системи у майбутньому;
- Можливість горизонтального масштабування для збереження швидкодії;
- Можливість інтегрування месенджерів для спрощення доставки повідомлень у реальному часі;
- Мати публічний так и приватний програмний інтерфейс для комунікації із іншими системами.

### **1.2.3. Вибір логічної архітектури програмного забезпечення**

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

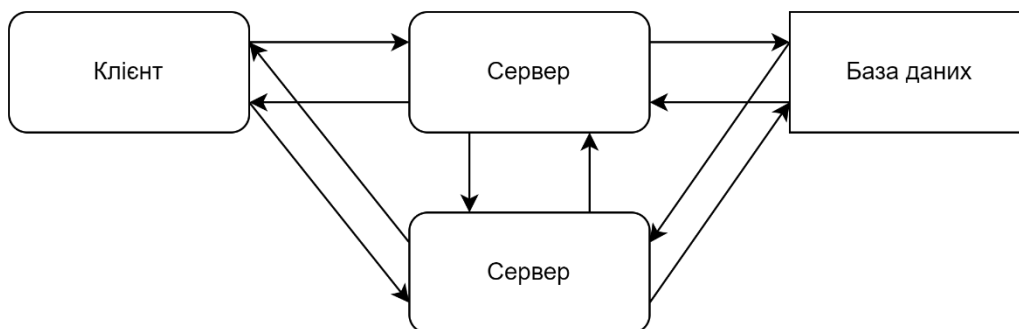
Дослідження архітектури програмного забезпечення намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодиноці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію.

Багато сучасних інформаційних систем побудовані на базі дворівневої архітектури клієнт/сервер. При цьому клієнтське програмне забезпечення відповідає за ввід та відображення даних. Самі ж дані централізовано зберігаються та оброблюються на серверах, до яких підключаються клієнти, як представлено на рисунку 1.1.



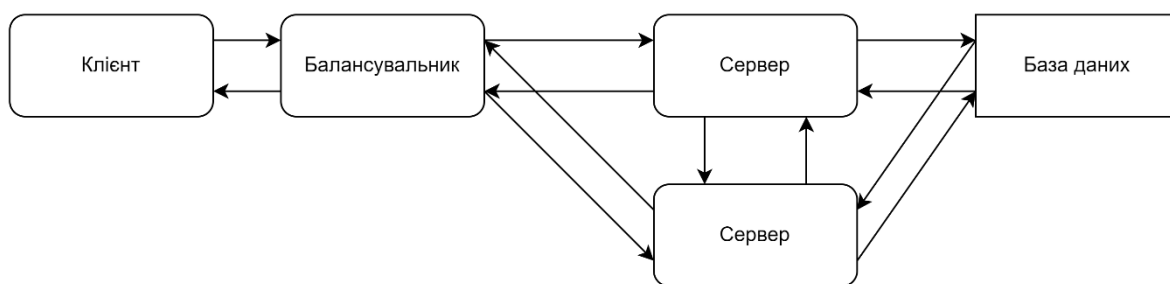
**Рисунок 1.1 - Діаграма клієнт-серверної архітектури**

Клієнт-серверна архітектура дозволяє масштабувати систему для забезпечення пропускнуєї можливості для клієнтів [5] (рис. 1.2)



**Рисунок 1.2 - Діаграма клієнт-серверної архітектури з масштабуванням кількості робочих серверів**

При використанні балансувального серверу (рис. 1.3):

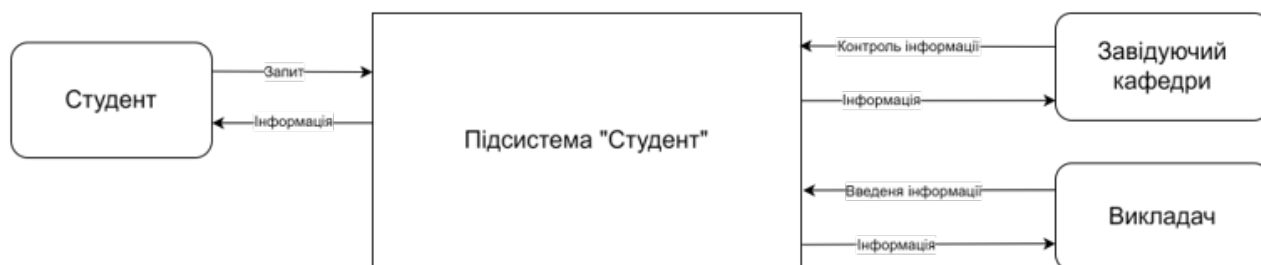


**Рисунок 1.3 - Діаграма клієнт-серверної архітектури із використанням балансувального серверу**

Для масштабування можна використати будь яку кількість серверів, як вказано на рисунках 1.2 та 1.3.

### **1.3. Розробка концепції підсистеми «Студент» інформаційної системи кафедри ІУСТ.**

Виходячи з аналізу функцій підсистеми та вимог до системи можна розробити концепцію (рис. 1.4)



**Рисунок 1.4 - Організаційно-функціональна структура**

Основними компонентами є:

- *система управління базами даних* – ядро будь-якої інформаційної системи;
- *серверна платформа і серверне програмне забезпечення* – один з основних факторів організації інформаційної системи;
- *клієнтська платформа і клієнтське програмне забезпечення*;
- *мережеве середовище*;

### 1.3.2 Техніко-економічне обґрунтування обраної концепції

Загальний кошторис витрат на створення підсистеми «Студент» за обраною концепцією наведено у таблиці 1.3.

Таблиця 1.3

#### Загальний кошторис витрат на створення підсистеми

| №                                | Найменування витрат                        | Сума, грн. |
|----------------------------------|--------------------------------------------|------------|
| 1                                | Закупівля технічних засобів                | 0,00       |
| 2                                | Витрати на програмні засоби розробки       | 0,00       |
| 3                                | Витрати на операційну систему              | 0,00       |
| 4                                | Витрати на матеріали і комплектуючі вироби | 0,00       |
| 5                                | Витрати на заробітну платню розробника     | 0,00       |
| Разом на створення системи, грн. |                                            | 0,00       |

Обраний варіант задовольняє вимоги до функціональності системи при використанні наявного програмного забезпечення.

### 1.3.3. Орієнтовний план реалізації обраного варіанта концепції системи

Орієнтований план реалізації обраного варіанта концепції створення системи зображено в таблиці 1.4:

Таблиця 1.4

#### Орієнтований план реалізації обраного варіанта концепції системи

| Етапи розробки                                | Терміни виконання       |
|-----------------------------------------------|-------------------------|
| Технічне завдання (визначення й аналіз вимог) | 14.02.2022 – 21.03.2022 |
| Ескізне проектування (розробка специфікацій)  | 21.03.2022 – 10.04.2022 |
| Технічне проектування                         | 11.04.2022 – 21.04.2022 |

|                                                           |                         |
|-----------------------------------------------------------|-------------------------|
| Робоче проектування (кодування, налагодження, тестування) | 22.04.2022 – 22.05.2022 |
| Впровадження                                              | 23.05.2022 – 1.07.2022  |

#### **1.3.4. Умови приймання системи**

Приймання створеної системи здійснюється представниками виконавця і замовника відповідно до технічного завдання. Складається двосторонній акт виконаних робіт.

### **1.4. Постановка задачі**

#### **1.4.1 Загальні відомості**

Потрібно автоматизувати процеси інформаційної взаємодії із студентами на кафедрі ІУСТ. Назва розробки «Підсистема Студент».

#### **1.4.2 Призначення і мета створення підсистеми «Студент»**

Призначенням цієї підсистеми є автоматизація взаємодій, які використовуються на кафедрі ІУСТ, створення інформаційної бази даних де буде зберігатися та оброблюватися інформація про персонал, предмети, події, розклад таких навчальних процесів як пари, сесії та іспити.

Метою створення підсистеми є спрощення способів отримання інформації студентам, а також покращення зручності використання інформації навчальної кафедри.

#### **1.4.3 Вимоги до підсистеми «Студент»**

Дати можливість використовувати підсистему студентам для отримання необхідної інформації, а саме розклад групи, або тижневий розклад предметів певного викладача на кафедрі навчання. А так само підсистема повинна мати відкритий доступ до інформації з бази даних для подальшого використання у



візуальному поданні даних, таких як веб-сайт, або сторінка в месенджерах, де складаються студенти і можливо організовується робочий процес групи з викладачами та куратором. Ця підсистема повинна мати простий і лаконічний інтерфейс управління та перегляду інформації, для спрощення подачі та використання інформації. Компоненти цієї підсистеми повинні бути реалізовані з використанням мікро сервісної архітектури, для можливості подальшого масштабування в участі в ланцюзі більших інформаційних мереж. Зазначені функції підсистеми повинні мати гнучкий інтерфейс отримання/віддачі даних, для можливості інтеграції нових функцій та підтримання актуальності тривалий час використання, з урахуванням мінімальних змін в освітній процедурі навчального закладу.

#### **1.4.4 Вимоги до функцій підсистеми «Студент»**

Підсистема «Студент» має виконувати такі функції, що наведені у списку:

Надання даних із внутрішньої системи:

- 1) отримання списку предметів;
- 2) пошук предмета;
- 3) отримання списку викладачів, пошук за різними фільтрами;
- 4) отримання списку груп;
- 5) отримання групи за ім'ям та датою надходження;
- 6) отримання списку аудиторій;
- 7) одержання розкладу групи, розкладу кафедри, розкладу викладача, розклад аудиторії;
- 8) отримання загальних повідомлень кафедри ІУСТ.

Ідентифікація відвідувачів та розмежування ролей:

- 1) Авторизація користувача;
- 2) Реєстрація користувача;
- 3) Ідентифікація ролі користувача (за номером залікової книжки та цифрового підпису отриманого на кафедрі);

- 4) Ідентифікація приналежності користувача до навчальної групи;
- 5) Перегляд повідомлень групи користувача;
- 6) Перегляд розкладу групи користувача;
- 7) Перегляд персональних повідомлень користувача.

#### Адміністрація системи:

- 1) авторизація користувача ролі адміністратора;
- 2) додавання даних викладачів;
- 3) додавання даних аудиторій;
- 4) додавання даних викладачів;
- 5) додавання даних предметів;
- 6) заповнення розкладу на основі доданих даних сутностей;
- 7) інтеграція системи із сторонніми вузлами інтерфейсів користувачів;
- 8) інтеграція на каналах месенджерів;
- 9) інтеграція у групових чатах месенджерів;
- 10) інтеграція чат-ботів у різних месенджерах.

#### **1.4.5 Вимоги до інформаційного забезпечення підсистеми «Студент»**

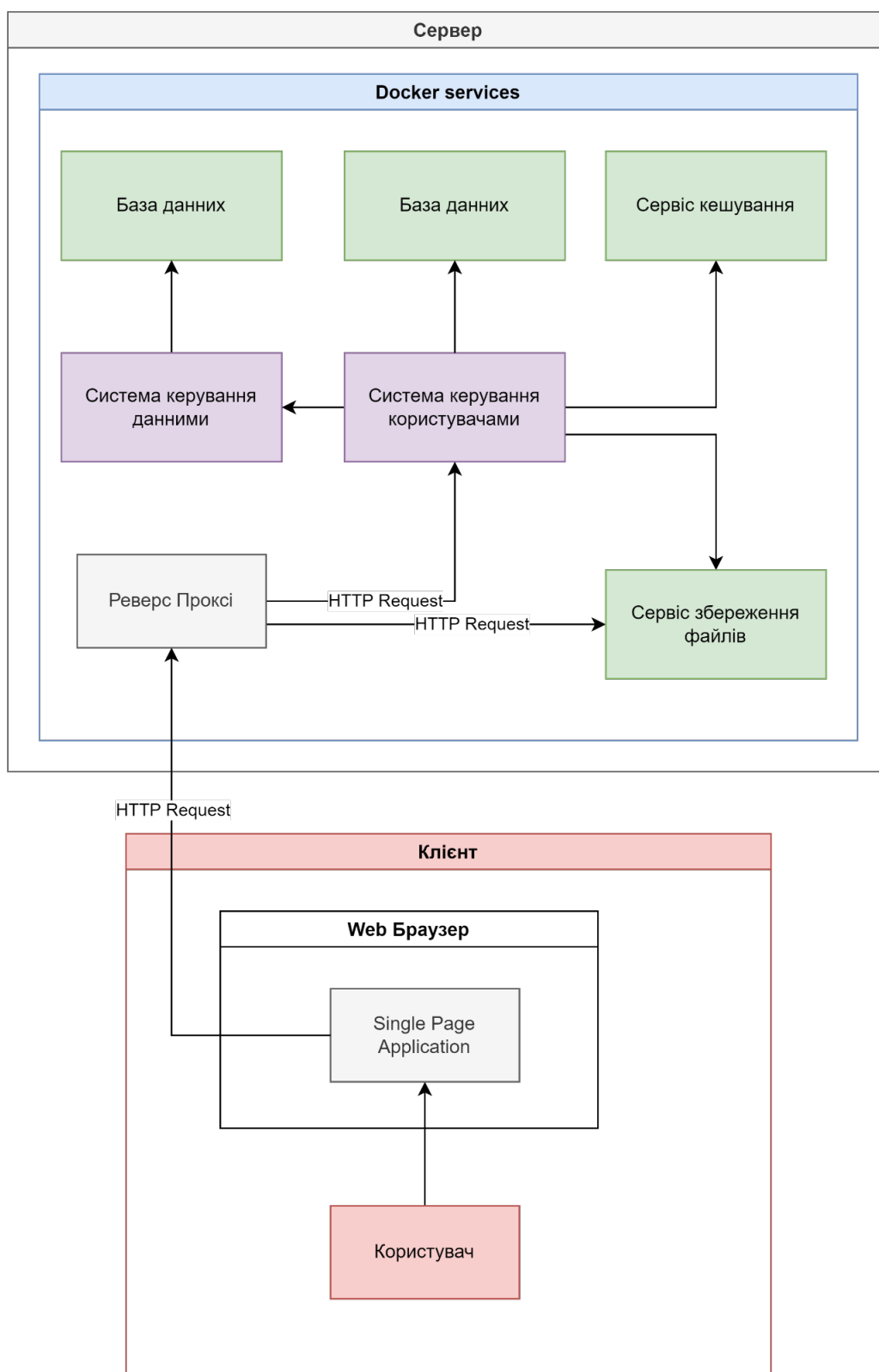
Система повинна підтримувати розподілену архітектуру для можливості горизонтального масштабування вузлів обробки інформації.

Система має бути відкритою за визначенням, прийнятим Комітетом IEEE POSIX 1003.0 (Відкритою інформаційною системою називається система, яка реалізує відкриті специфікації на інтерфейси, сервіси (послуги середовища) та підтримувані формати даних, достатні для того, щоб дати можливість належним чином розробленому прикладному програмному забезпеченню переноситися в широкому діапазоні систем з мінімальними змінами, взаємодіяти з іншими додатками. на локальних та віддалених системах, та взаємодіяти з користувачами у стилі, що полегшує перехід користувачів від системи до системи) [6].

Система повинна забезпечувати захист при передачі конфіденційної інформації від перехоплення за допомогою шифрування, а також ідентифікацію

ролей користувачів під час виконання запитів до системи. Система має бути розділена на кілька багаторівневих сервісів, для спрощення взаємодії між функціональними вузлами на різних рівнях доступу до інформації, а також покращить можливості для масштабування, що збільшить потенціал обробки більшої кількості запитів.

Блок-схема інформаційного взаємозв'язку функціональних завдань автоматизованої системи наведена на рисунку 1.6.



**Рисунок 1.5 - Блок-схема інформаційного взаємозв'язку функціональних завдань автоматизованої системи**

#### **1.4.6 Вимоги до лінгвістичного забезпечення та програмного забезпечення підсистеми «Студент»**

Підсистема «Студент» розробляється мовою програмування TypeScript, яка компілюється та інтерпретується у мову JavaScript та запускається фреймворком серверного виконання – NodeJS.

До підсистеми «Студент» входить сервер на якому працюють вузли сервісів системи та клієнти (браузер, чат-боти та інше).

Вимоги до сервера для розгортання системи:

- операційна система: будь яка система, яка підтримує віртуалізацію (наприклад Linux – Debian 10+, Ubuntu 16+, Windows - Windows 7 або новіше);
- Програмне забезпечення Docker версії 16 або новіше.

Вимоги до програмного забезпечення клієнтського девайсу:

- Операційна система: будь-яка операційна система у який можна встановити будь-який сучасний веб-браузер.

#### **1.4.7 Вимоги до технічного забезпечення підсистеми «Студент»**

Вимоги до технічного забезпечення сервера для розгортання системи:

- Процесор із наявними 2 або більше потоків виконання операцій, и тактовою частотою 1.3ггц або вище. Архітектура процесора будь-яка (ARM, x86 та інші).
- Оперативна пам'ять RAM 2Гб або більше.
- Накопичувач SSD диск із вільним місцем 10Гб або більше.
- Мережеве підключення Ethernet щонайменше 10мбіт/с.

Вимоги до технічного забезпечення клієнтського девайсу:

Будь який сучасний девайс який надає змогу запускати та відображати веб сторінки із виконанням JavaScript.

#### **1.4.8 Вимоги до документації підсистеми «Студент»**

Необхідна наявність наступних документів:

- Загальний опис системи.
- Керівництво користувача (адміністратора системи).

- Керівництво адміністратора з установки програмного забезпечення на серверах.
- Програма і способи випробувань.
- Текст програми.

## РОЗДІЛ 2. РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ

### 2.1 Загальний опис підсистеми «Студент»

Мета розробки підсистеми «Студент» -- підвищення ефективності та продуктивності навчального процесу.

Функції що може виконувати студент:

- авторизуватися в системі;
- переглядати розклад;
- переглядати інформацію про викладачів;
- переглядати інформацію про предмети;
- переглядати новини;

Функції що може виконувати викладач або керівник групи:

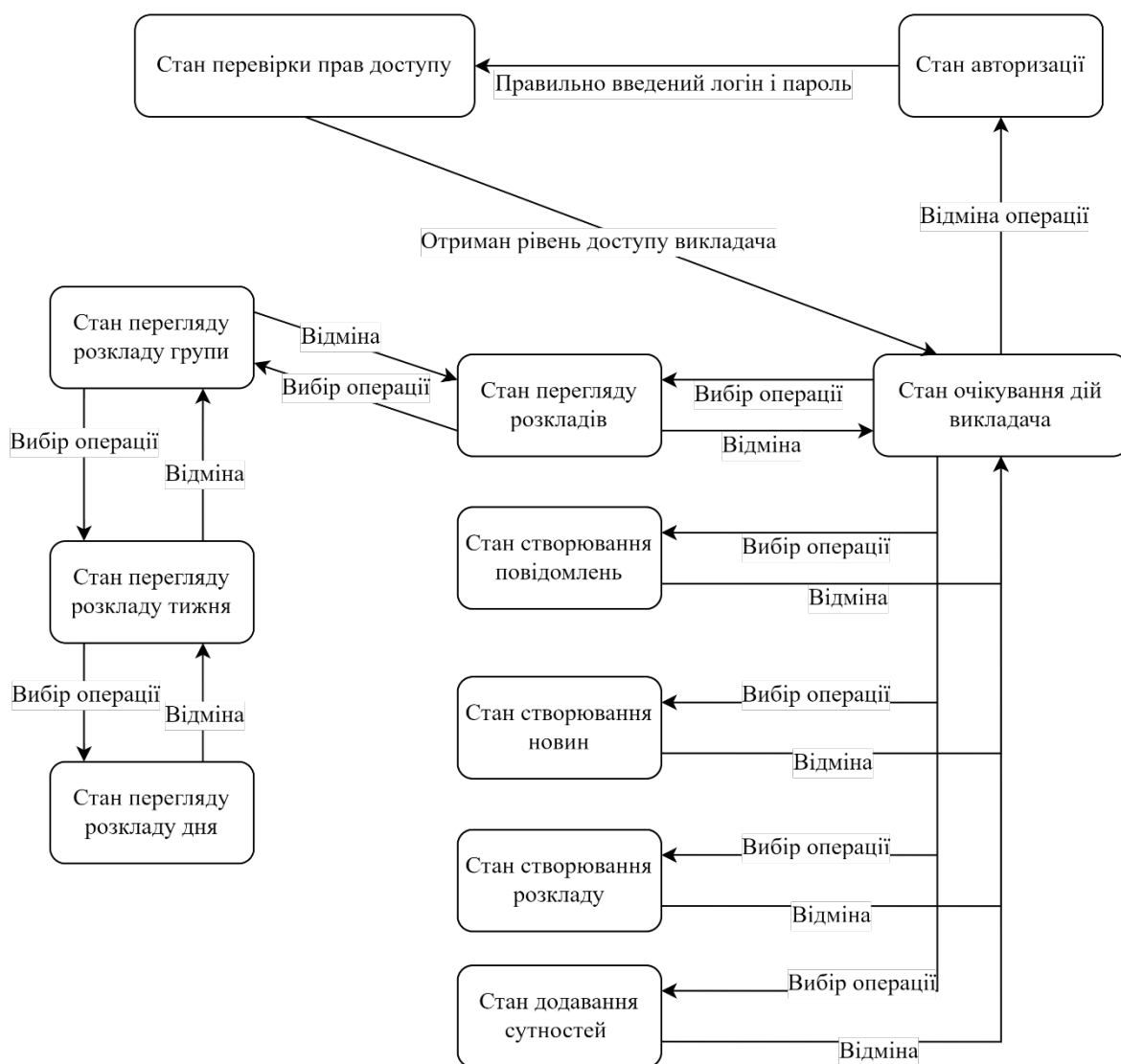
- авторизуватися в системі;
- заповнювати розклад;
- заповнювати інформацію предметів;
- створювати новини;
- створювати повідомлення для груп;
- створювати повідомлення для студентів;

Опис підсистеми «Студент»

Користувачами системи є здобувачі вищої освіти та викладачі ЗВО. Також для керування ресурсами системи потрібен адміністратор, у даному випадку роллю адміністратора може володіти будь який викладач кафедри.

При переході до системи, система дозволяє переглядати деяку інформацію у гостевому стані, або користувач має змогу авторизуватись – тобто перейти до стану авторизації.

Стани до яких може переходити система у разі якщо авторизований користувач має рівень доступу викладача представлена діаграмою переходу станів на рисунку 2.1.



**Рисунок 2.1 – Діаграма переходів станів користувача**

До стану створення розкладу система переходить якщо користувач використає операцію «Створити розклад». У цьому стані необхідно заповнити поля форми додавання сутностей розкладу. Після цього система звертається до бази даних де зберігає зміни. У разі необхідності користувач може вносити корективи не виходячи зі стану створення розкладу. Далі, після успішного збереження даних, якщо усі перевірки даних системою пройшли успішно система переходить до стану перегляду доступних розкладів. До цього стану можна також перейти безпосередньо зі стану очікування дій викладача.

Зі стану перегляду розкладів система може перейти до: стану очікування дії викладача, відміна дії, стану перегляду розкладу обраної групи, стану



перегляду розкладу групи по тижню, стану перегляду розкладу групи по тижню обраного дня, стану створення повідомлень, стану створення новин, стану створення розкладу, стану додавання сутностей.

Інформаційне забезпечення системи складається з 2 баз даних, у яких зберігається вся необхідна інформація для користувачів та ресурсів кафедри й також с кеш бази даних для забезпечення швидкодії.

Система не вносить змін в організаційну структуру кафедри й не втручається у роботу існуючих процесів взаємодії. Технічне забезпечення системи складається с одного центрального серверу та будь яких пристроїв користувачів. Для максимальної продуктивності, масштабуємості та відмовостійкості системи необхідно використовувати хмарні сервіси.

Програмне забезпечення складається з сайту та серверних додатків які забезпечують функції описаної системи.

Перевага створення системи на кафедрі полягає в тому, що інформація буде зберігатися у центральній базі даних, це дозволяє систематизувати и обробляти інформацію. Цю інформацію зможуть надалі використовувати інші інформаційні системи.

## **2.2. Рішення з інформаційного забезпечення**

Файли баз даних та статичні дані завантажені користувачами на сайті зберігаються у файловій системі серверного накопичувача. Обсяг пам'яті повинен бути не менше 1 Гб. Структура файлів буде описана в наступних підрозділах.

База даних повинна зберігатися в системі Postgres. База даних розташується на накопичувачі основного серверу. Для підвищення швидкодії бази даних при запису даних на накопичувач необхідно використовувати файлову систему XFS яка доступна лише під керуванням операційної системи Linux.

Організація збору і передачі інформації.

Введення інформації в базу даних системи повинно здійснюватися за допомогою програмного забезпечення, що розроблено за допомогою технологій веб сайтів та веб браузеру.

За замовчуванням при першому запуску системи генерується дані для входу головного адміністратора, які можна взяти або с консольного виводу серверного додатку, або з файлів конфігурації запуску додатка на сервері. Ці дані мають бути безпечно передані головним системним адміністратором після первинного налаштування програмного комплексу на серверній частині системи.

У системі використовується мікросервісна архітектура, яка поділяє систему на 2 сервіси:

- сервіс систематизації, забезпечення та первинної обробки інформації (Core Service);
- сервіс взаємодії з користувачами, який здійснює контроль доступу та забезпечує зв'язок із кінцевим клієнтом (User Service).

Мікросервісна архітектура — варіант сервіс-орієнтованої архітектури програмного забезпечення, спрямований на взаємодію наскільки це можливо невеликих, слабко пов'язаних модулів, що легко змінюються та працюють виключно з певними задачами.

У кожного зі зазначених сервісів використовується окремі бази даних кожна зі своїм набором таблиць та процедурами.

### **2.2.1 Концептуальна модель даних**

Для представлення концептуальної моделі будуть використані діаграми «сутність-зв'язок» (ERD). Із їхньою допомогою можливо визначити певні об'єкти сутності для предметної області, а також їх властивості (атрибути), відносини один до одного (зв'язки).

Для побудови концептуальної моделі даних системи, необхідно зазначити певні сутності та їх атрибути.

По-перше необхідно виділити сутності для подальшої побудови концептуальної моделі даних.

З урахуванням наявної інформації, що наведена нижче, побудуємо ER-діаграми компонентів підсистеми (рис. 2.2 – 2.3)

Можна виділити наступні сутності для сервісу забезпечення інформації (Core Service):

- особи;
- особові посади;
- локації;
- кваліфікації;
- спеціальності;
- студентські групи;
- предмети;
- типи предметів;
- розклад;
- розклад дня;
- заняття розкладу;
- заняття;

Сутності для сервісу взаємодії з користувачами (User Service):

- користувачі;
- ролі користувачів;
- присвоєні ролі користувачів;
- профілі користувачів;
- файли;
- чати;
- присвоєні учасники чатів;
- повідомлення чатів;
- підтвердження електронних пошт;
- нагадування груп;
- нагадування користувачів;

Наступний крок – ідентифікація атрибутів. Визначимо атрибути сутностей. Атрибути сутностей Core Service:

Сутність «Особи»:

- Ідентифікатор особи;
- Ім'я;
- Фамілія;
- По-батькові;
- Імейл;
- Телефон;
- Ідентифікатор позиції.

Сутність «Особові посади»:

- Ідентифікатор посади;
- Назва посади;
- Рівень ієрархічного доступу.

Сутність «Локації»:

- Ідентифікатор локації;
- Ім'я.

Сутність «Кваліфікації»:

- Ідентифікатор кваліфікації;
- Назва кваліфікації;
- Тривалість.

Сутність «Спеціальності»:

- Ідентифікатор спеціальності;
- Назва;
- Ідентифікатор факультету;
- Код спеціальності.

Сутність «Студентські групи»:

- Ідентифікатор групи;
- Назва групи;
- Ідентифікатор спеціальності;
- Ідентифікатор кваліфікації;

- Ідентифікатор особи керівника (куратора).

Сутність «Предмети»:

- Ідентифікатор предмету;
- Назва.

Сутність «Типи предметів»:

- Ідентифікатор типу предмету;
- Назва;
- Скорочена назва.

Сутність «Розклад»:

- Ідентифікатор розкладу;
- Ідентифікатор групи;
- Назва;
- Ітерація тижня (1 або 2 тиждень).

Сутність «Розклад дню»:

- Ідентифікатор розкладу дню;
- Ідентифікатор розкладу;
- Порядковий номер дню.

Сутність «Заняття»:

- Ідентифікатор заняття;
- Ідентифікатор локації;
- Ідентифікатор предмету;
- Ідентифікатор викладача (особа);
- Ідентифікатор типу предмету.

Сутність «Заняття розкладу»:

- Ідентифікатор заняття розкладу;
- Ідентифікатор дню розкладу (Розклад дню);
- Ідентифікатор наступного заняття у розкладі дню;
- Ідентифікатор попереднього заняття у розкладі дню;
- Ідентифікатор заняття.

Атрибути сутностей User Service:

Сутність «Користувачі»:

- Ідентифікатор користувача;
- Тег;
- Юзернейм;
- Імейл;
- Підтвердження імейлу;
- Телефон;
- Ім'я;
- Фамілія;
- По-батькові;
- Пароль (хеш);
- Заблокований;
- Код гугл аккаунту;
- Коли створено;
- Коли змінено;
- Ідентифікатор профілю;
- Ідентифікатор групи (код з сервісу Core).

Сутність «Ролі користувачів»:

- Ідентифікатор ролі;
- Назва ролі;
- Коли створено;
- Коли змінено;
- Рівень доступу ролі.

Сутність «Присвоєні ролі користувачів»:

- Ідентифікатор присвоєння;
- Ідентифікатор ролі;
- Коли створено;

- Коли змінено;
- Ідентифікатор користувача.

Сутність «Профілі користувачів»:

- Ідентифікатор профілю;
- День народження;
- Коли створено;
- Коли змінено;
- Ідентифікатор файлу (аватар).

Сутність «Файли»:

- Ідентифікатор файлу;
- Назва файлу;
- Ідентифікатор користувача (Власник файлу);
- Коли створено;
- Коли змінено;
- Посилання на файл.

Сутність «Чати»:

- Ідентифікатор чату;
- Тип чату; (Персональний/Спільноти)
- Коли створено;
- Коли змінено;
- Власник чату.

Сутність «Присвоєні учасники чатів»:

- Ідентифікатор присвоєння;
- Ідентифікатор чату;
- Ідентифікатор користувача;
- Коли змінено.

Сутність «Повідомлення чатів»:

- Ідентифікатор повідомлення;
- Текст;
- Коли створено;
- Коли змінено;
- Ідентифікатор чату;
- Ідентифікатор користувача (Відправник).

Сутність «Підтвердження електронних пошт»:

- Ідентифікатор підтвердження;
- Імейл;
- Код;
- Підтвердження;
- Ідентифікатор користувача;
- Коли створено;
- Коли змінено.

Сутність «Нагадування груп»:

- Ідентифікатор нагадування;
- Назва;
- Текст;
- Важливість;
- Час;
- Ідентифікатор групи;
- Ідентифікатор створювача (Користувач)
- Коли створено;
- Коли змінено.

Сутність «Нагадування користувачів»:

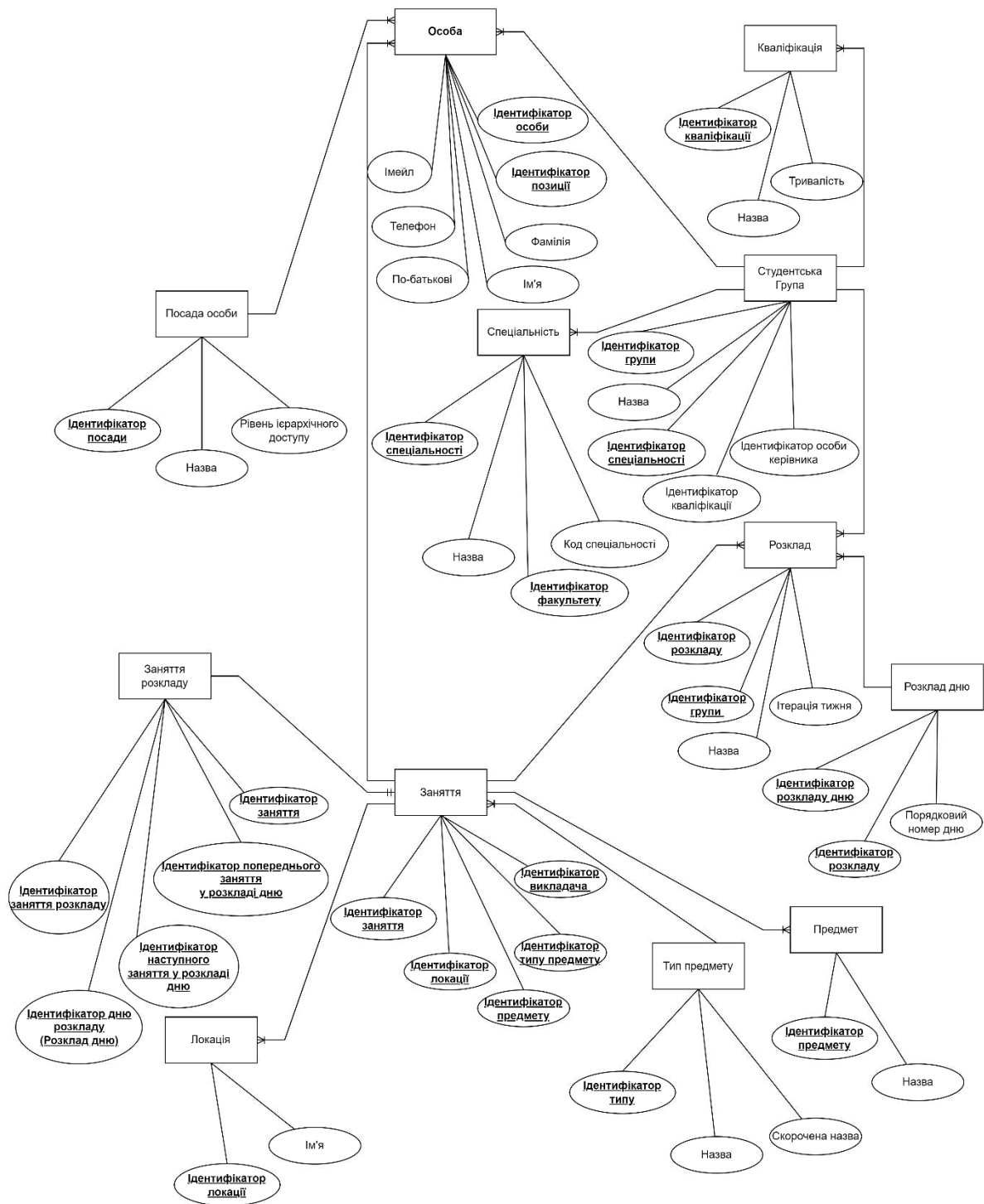
- Ідентифікатор нагадування;
- Назва;



- Текст;
- Важливість;
- Час;
- Ідентифікатор користувача;
- Ідентифікатор створювача (Користувач);
- Коли створено;
- Коли змінено.

### **2.2.2 Розробка логічної моделі бази даних**

Виходячи з аналізу концептуальної моделі даних системи, розробимо логічну структуру бази даних. Логічна структура бази даних системи представлена на рисунку 2.3.



**Рисунок 2.2 - Концептуальна модель даних Core Service**

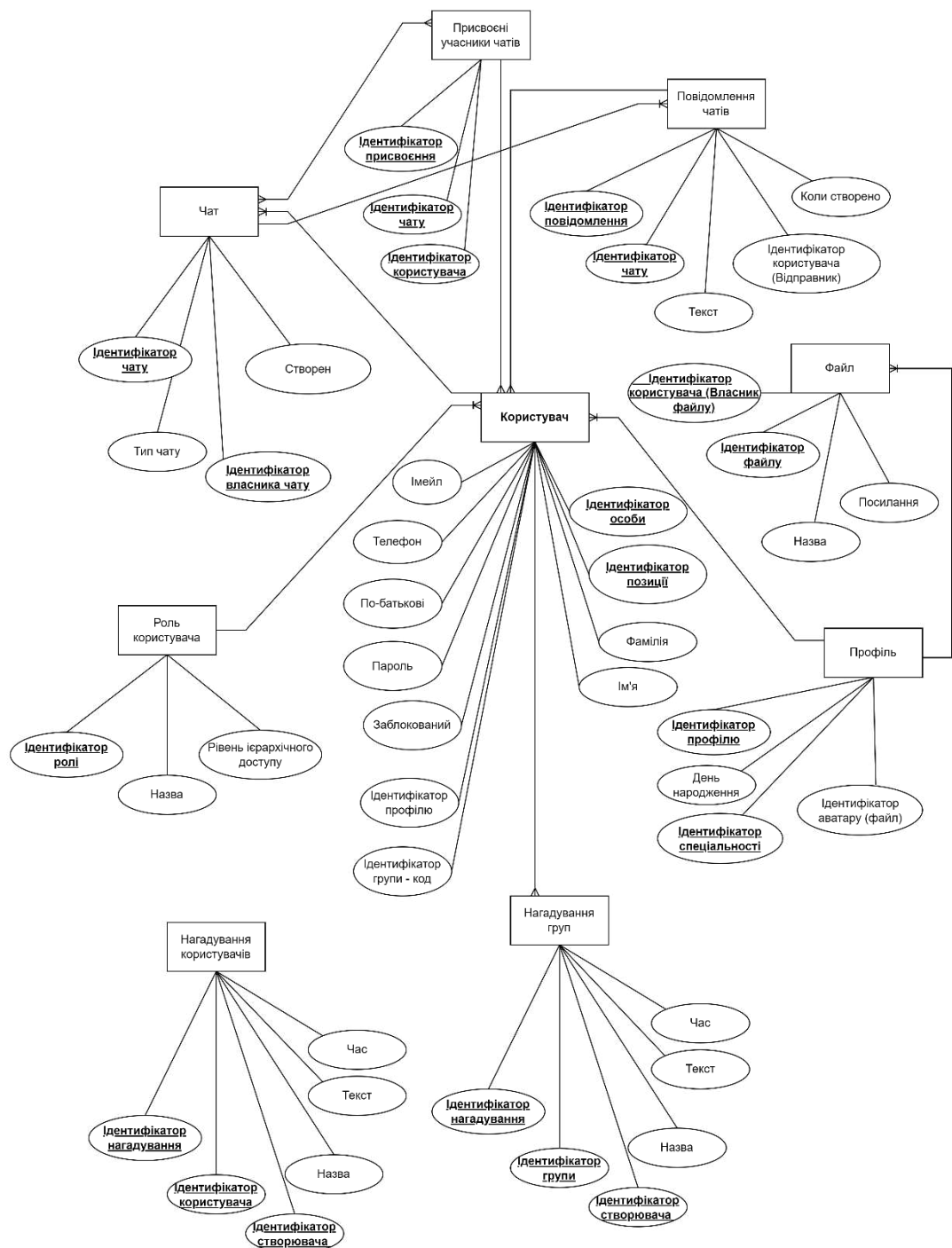


Рисунок 2.3 - Концептуальна модель даних User Service



Рисунок 2.4 - Логічна модель даних User Service

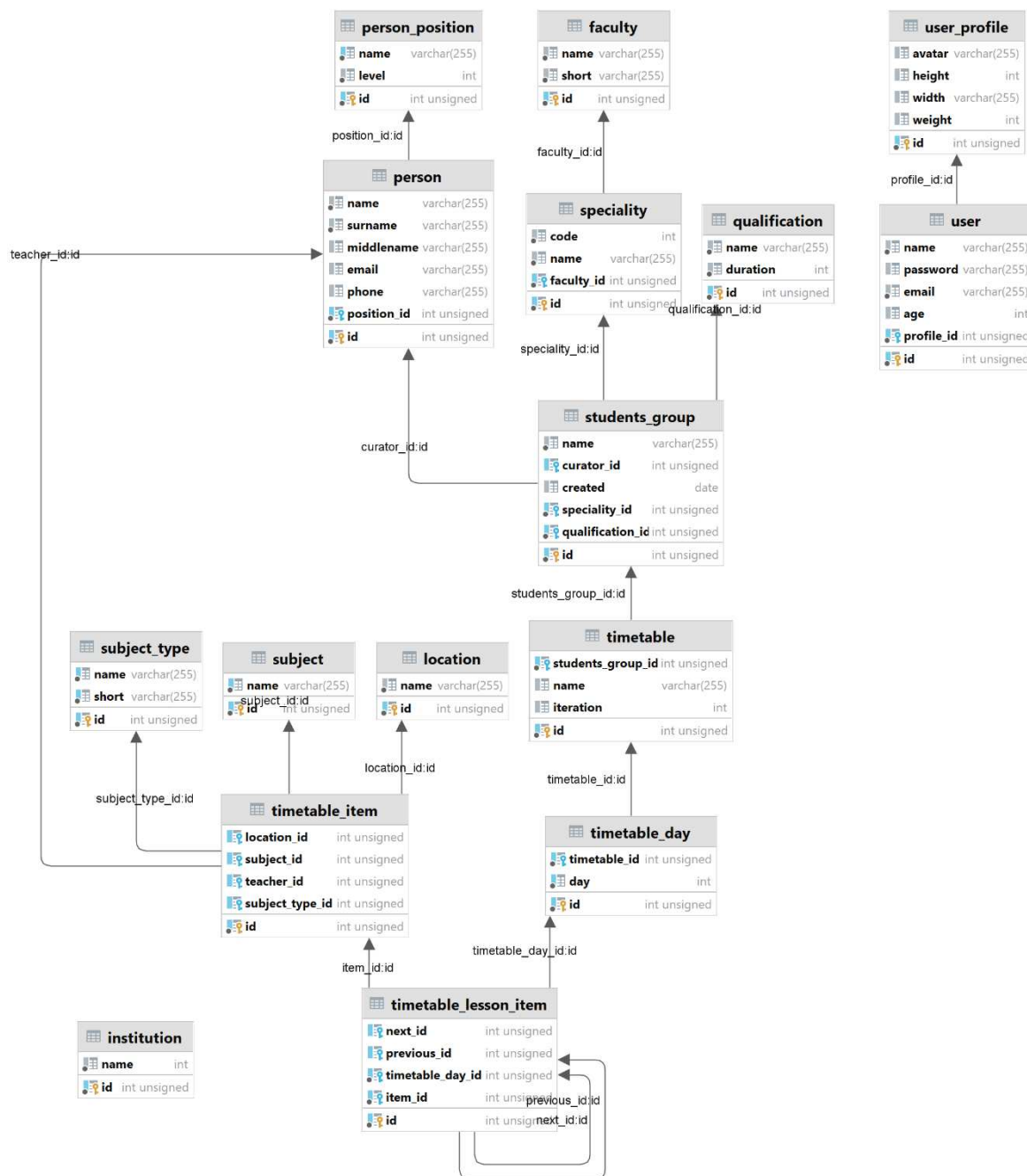


Рисунок 2.5 – Логічна модель даних Core Service

## 2.3 Ршення з технічного забезпечення

Підсистема «Студент» призначена для роботи у глобальній мережі для спільної комунікації учасників навчального процесу.

Для передачі даних на транспортному рівні використовується протокол TCP/IP, на рівні представлення використовується XML та JSON, на прикладному рівні - HTTP із використанням шифрування SSL (сеанс TLS).

Комплекс програмного забезпечення підсистеми «Студент» передбачає використання одного (або більше) серверів та будь-якої кількості клієнтів (залежно від ресурсів масштабованої системи на серверах).

## **2.4 Рішення з програмного забезпечення**

Сервер для програмного комплексу з такими параметрами:

- Процесор Xeon E5-2640v4 (2 потоки);
- Оперативна пам'ять 2 гб;
- SSD накопичувач 50 гб;
- ОС Linux Debian 11;
- Docker
- Postgres

Ці показники задовольняють вимогам (п. 1.4.6) до програмного забезпечення

## 3 РЕАЛІЗАЦІЯ ПРОЕКТУ

### 3.1. Реалізація компонентів підсистеми

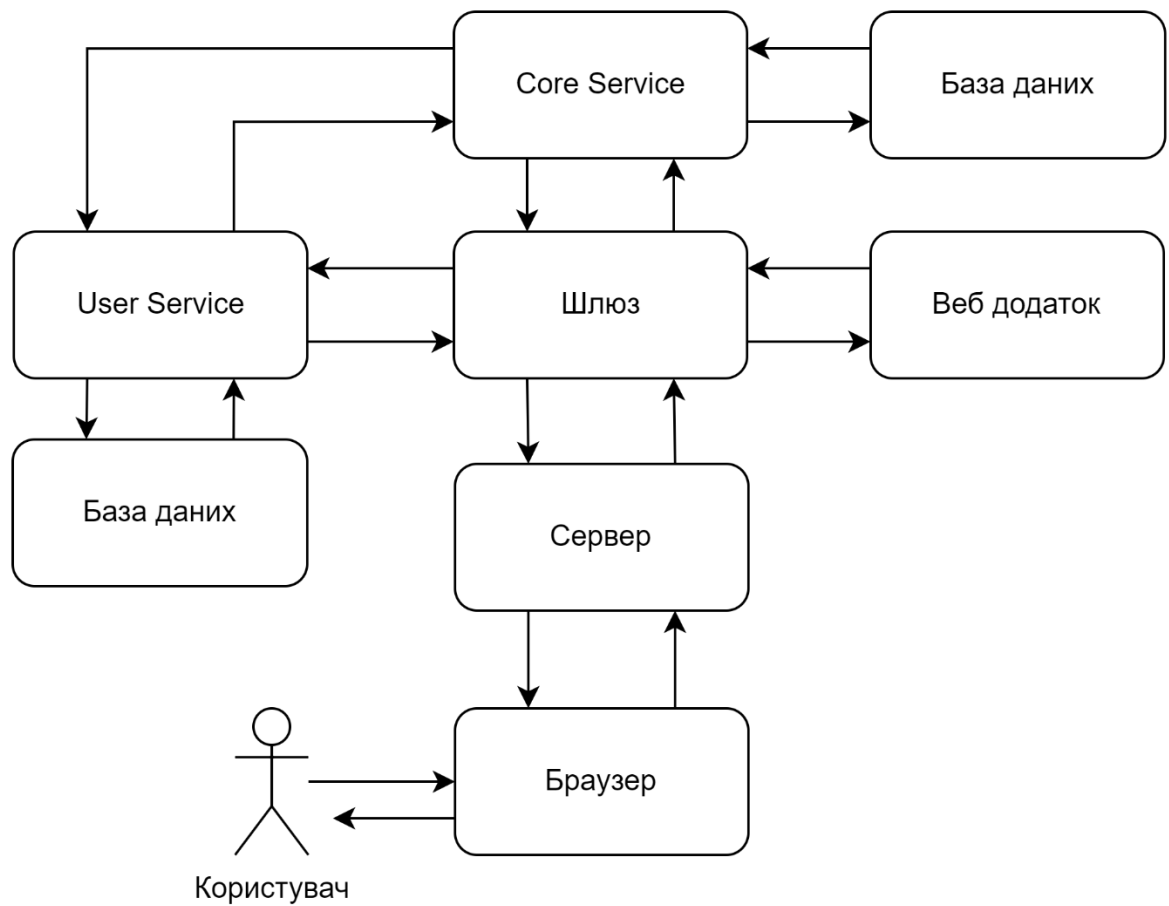
Підсистема розподілена на низку сервісів які забезпечують заявлені функції повного програмного комплексу.

Core Service – сервіс який забезпечує систему інформаційними функціями контролю, такими як зчитування інформації, редагування, видалення, створення запитів, заплановані дії з агрегацією бази даних. Цей сервіс має знаходитись у приватній мережі, в якій мають знаходитись сервіси які потребують взаємодії із Core Service, сервіс не має контролю доступу на будь якому з рівнів введення або зчитування даних. Інформаційний інтерфейс забезпечує прикладний протокол – HTTP. Сервіс має виділену реляційну базу даних в якій виконується збереження даних.

User Service – сервіс який забезпечує систему функціями взаємодії з кінцевими клієнтами. Має знаходитись в мережі із Core Service, й також у мережі шлюзу, який забезпечує вихід до глобальної мережі. Цей сервіс має функції контролю доступу до ресурсів які забезпечує Core Service, також виконує роль зв'язування клієнтських даних із даними зазначеними у Core Service. Всі запити кінцевих користувачів які використовують функції системи мають проходити через цей сервіс користувача, це забезпечує контроль доступу та контроль введення інформації, сервіс має виділену базу даних збереження клієнтських даних, яка забезпечує найвищий рівень конфіденційності.

Окрім цих сервісів система має шлюз який виконує маршрутизацію клієнтських запитів до сервісів програмного комплексу серверу. Також цей шлюз запобігає зовнішнього втручання у процеси роботи системи, також забезпечує ефективну маршрутизацію запитів у системі.

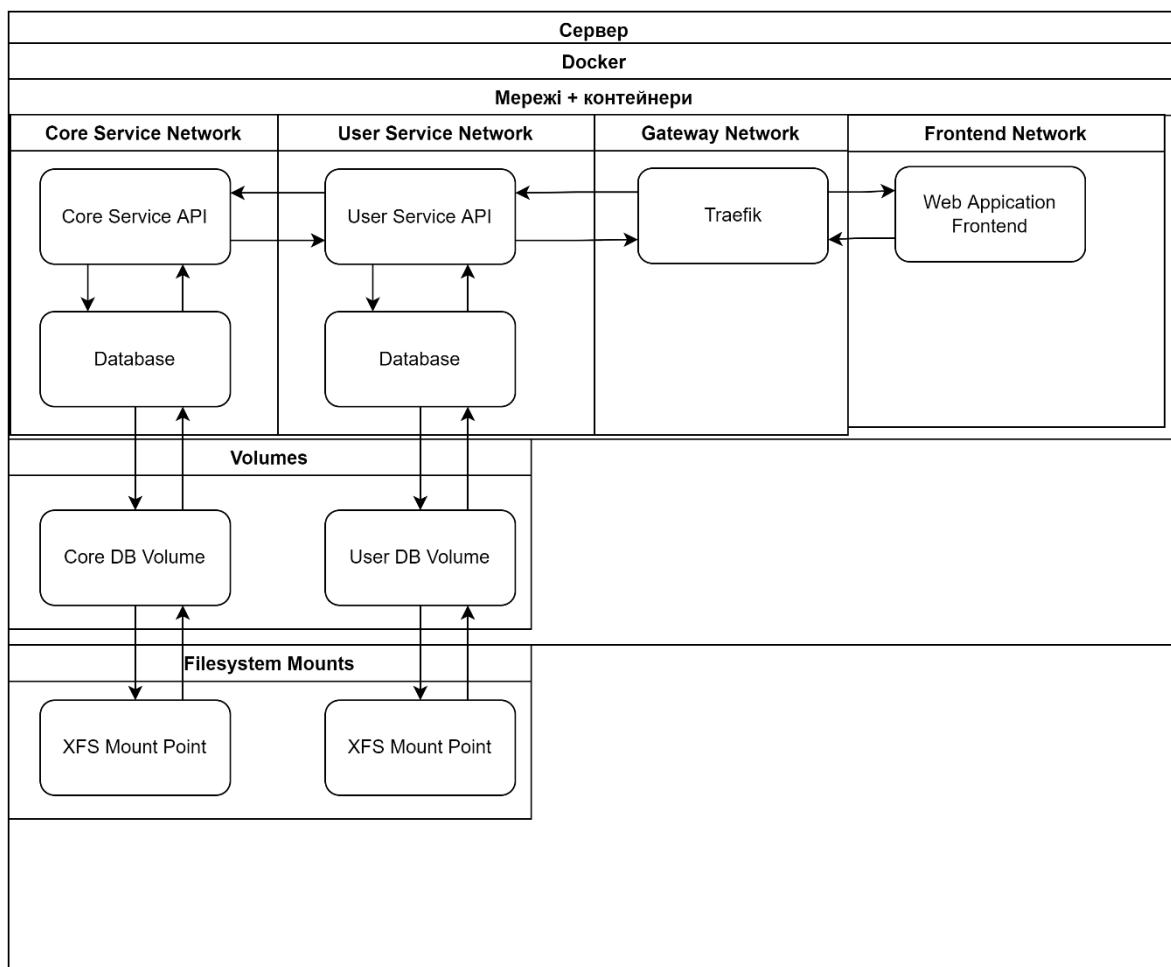
На рисунку 3.1 запропоновано організаційну діаграму архітектури реалізації.



**Рисунок 3.1** - Організаційна діаграма архітектури реалізації

Детально при розгортанні програмного комплексу із використанням оркестратора контейнерів Docker на основі образів сервісів та допоміжних вузлів системи можна зазначити на рисунку 3.2.





**Рисунок 3.2** – Структура програмного комплексу на сервері.

Core Service Network – ізольована мережа сервісу Core.

User Service Network -- ізольована мережа сервісу User.

Gateway Network -- ізольована мережа шлюзу.

Frontend Network – ізольована мережа веб додатку.

Database – база даних

Volumes – розділи для збереження даних баз

XFS Mount Point – точка монтування вискооефективної файлової системи

XFS операційної системи Linux.

Вхідною точкою серверною архітектури є контейнер-сервіс Traefik який забезпечує подальшу маршрутизацію трафіку.

Docker – це програмна платформа для швидкої розробки, тестування та розгортання програм. Docker упаковує програмне забезпечення у

стандартизовані блоки, які називаються контейнерами. Кожен контейнер включає все необхідне для застосування: бібліотеки, системні інструменти, код і середовище виконання. Завдяки Docker можна швидко розгортати та масштабувати програми у будь-якому середовищі та зберігати впевненість у тому, що код працюватиме.

Traefik - це зворотний проксі-сервер з відкритим вихідним кодом, що забезпечує просту роботу з мікросервісами та/або просто контейнерами з вашими програмами.

Сервіси реалізовані мовою програмування JavaScript при участі серверного фреймворку під назвою NodeJS, який забезпечує роботу коду на сервері та має асинхронні ІО операції що надає високою ефективності із найменшим споживанням ресурсів.

При написанні проекту також була використана надбудована мова над JS – TypeScript що забезпечило контроль типізації під час програмування.

Для створення серверу запитів HTTP було використано фреймворк NestJS, який надає можливість використання ін'єкцій залежностей та передбачує використання модульної архітектури.

Базами даних для кожного сервісу була обрана Postgres SQL, що забезпечить високу продуктивність із обмеженим ресурсом серверу.

Взаємодія між сервісами створена на основі приватних мереж та HTTP запитів.

Веб додаток створено за допомогою HTML, CSS, JS та фреймворку VueJS.

За допомогою візуалізації запитів Swagger (Open API) на рисунках 3.3 – 3.7 можна побачити доступні запити до сервісів.

| Students groups |                       | ^ |
|-----------------|-----------------------|---|
| POST            | /students-groups      | ▼ |
| GET             | /students-groups      | ▼ |
| GET             | /students-groups/{id} | ▼ |
| PATCH           | /students-groups/{id} | ▼ |
| DELETE          | /students-groups/{id} | ▼ |
| Qualifications  |                       | ^ |
| POST            | /qualifications       | ▼ |
| GET             | /qualifications       | ▼ |
| GET             | /qualifications/{id}  | ▼ |
| PATCH           | /qualifications/{id}  | ▼ |
| DELETE          | /qualifications/{id}  | ▼ |
| Persons         |                       | ^ |
| POST            | /persons              | ▼ |
| GET             | /persons              | ▼ |
| GET             | /persons/{id}         | ▼ |
| PATCH           | /persons/{id}         | ▼ |
| DELETE          | /persons/{id}         | ▼ |

Рисунок 3.3 – Запити Core Service

| Timetable |                                          | ^ |
|-----------|------------------------------------------|---|
| POST      | /timetable                               | ▼ |
| GET       | /timetable                               | ▼ |
| GET       | /timetable/{id}                          | ▼ |
| PATCH     | /timetable/{id}                          | ▼ |
| DELETE    | /timetable/{id}                          | ▼ |
| POST      | /timetable/{id}/day                      | ▼ |
| DELETE    | /timetable/{id}/{day_number}             | ▼ |
| POST      | /timetable/{id}/{day_number}/lesson      | ▼ |
| PATCH     | /timetable/{id}/{day_id}/{lesson_id}     | ▼ |
| DELETE    | /timetable/{id}/{day_number}/{lesson_id} | ▼ |

Рисунок 3.4 – Запити Core Service

| Request URL                                    |                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>http://127.0.0.1:3000/timetable/1</code> |                                                                                                                                                                                                                                                                                                                                                                                |
| Server response                                |                                                                                                                                                                                                                                                                                                                                                                                |
| Code                                           | Details                                                                                                                                                                                                                                                                                                                                                                        |
| 200                                            | <p>Response body</p> <pre>{   "id": 1,   "name": "Неделя под чертой",   "iteration": 0,   "studentsGroup": {     "id": 1,     "name": "4144",     "curator": 1,     "speciality": 1,     "qualification": 1,     "created": "2018-10-01"   },   "days": [     {       "day": 0,       "lessons": [         {           "id": 4,           "item": {             "id": 6,</pre> |

Рисунок 3.5 – Запити Core Service, перегляд розкладу

| Account |                          | ^   |
|---------|--------------------------|-----|
| PUT     | /account/profile         | ✓ 🔒 |
| GET     | /account/profile-options | ✓ 🔒 |
| POST    | /account/avatar          | ✓ 🔒 |
| PUT     | /account/password        | ✓ 🔒 |
| Auth    |                          | ^   |
| POST    | /auth/login              | ✓   |
| POST    | /auth/register           | ✓   |
| GET     | /auth/me                 | ✓ 🔒 |
| GET     | /auth/me/group           | ✓ 🔒 |
| GET     | /auth/forward            | ✓ 🔒 |
| POST    | /auth/confirm-email      | ✓   |
| OAuth   |                          | ^   |
| GET     | /oauth/google            | ✓   |
| GET     | /oauth/google/redirect   | ✓   |
| Chats   |                          | ^   |
| GET     | /chats                   | ✓ 🔒 |
| POST    | /chats                   | ✓ 🔒 |
| GET     | /chats/{id}              | ✓ 🔒 |

Рисунок 3.6 – Запити User Service

GET /auth/me

**Parameters**

No parameters

**Responses**

| Code    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| default | <p>Media type</p> <p><b>application/json</b> ▾</p> <p>Controls Accept header.</p> <p>Example Value   Schema</p> <pre>{   "id": 0,   "tag": "string",   "username": "string",   "email": "string",   "emailConfirmed": true,   "phone": "string",   "name": "string",   "surname": "string",   "patronymic": "string",   "profile": {     "id": 0,     "birthday": "2022-06-26T23:38:16.240Z",     "phone": "string",     "avatar": {       "id": 0,       "name": "string",       "url": "string",       "createdAt": "2022-06-26T23:38:16.240Z",       "updatedAt": "2022-06-26T23:38:16.240Z"     },     "createdAt": "2022-06-26T23:38:16.240Z",     "updatedAt": "2022-06-26T23:38:16.240Z"   },   "roles": [     "string"   ] }</pre> |

Рисунок 3.7 – Запити User Service, перегляд інформації поточного авторизованого користувача

3.2. Екранні форми

Екранні форми приклади з інтерфейсу веб додатку для роботи с системою наведені на 3.10 – 3.11.

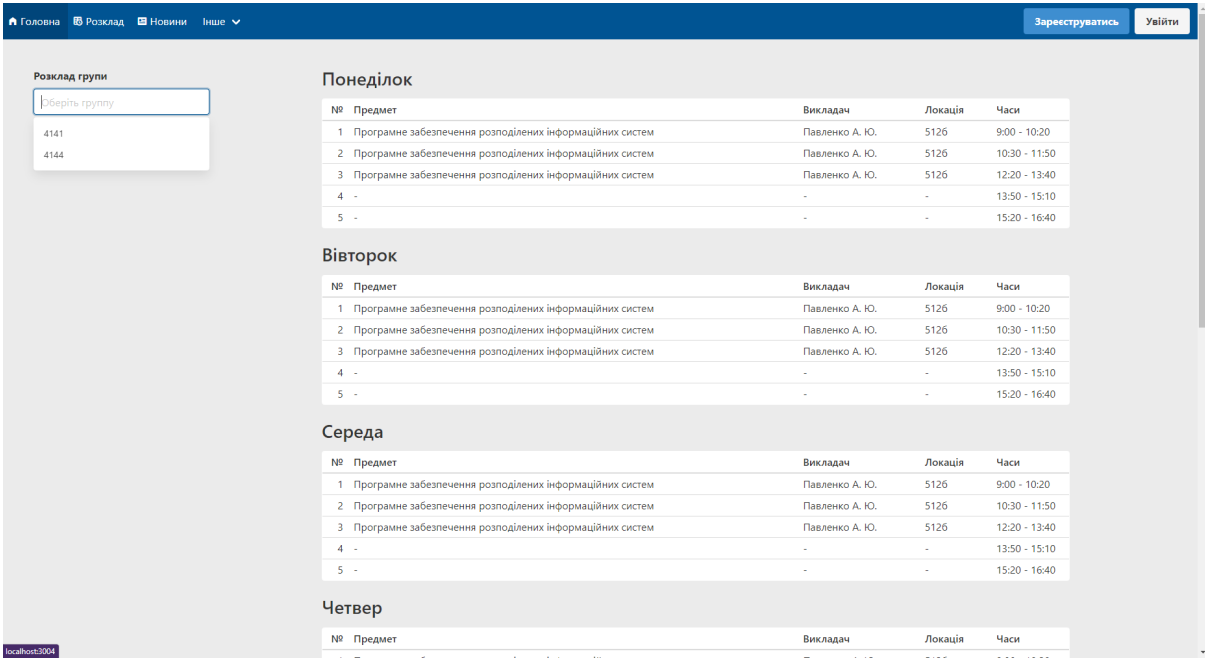


Рисунок 3.6 – Приклад сторінки розкладу

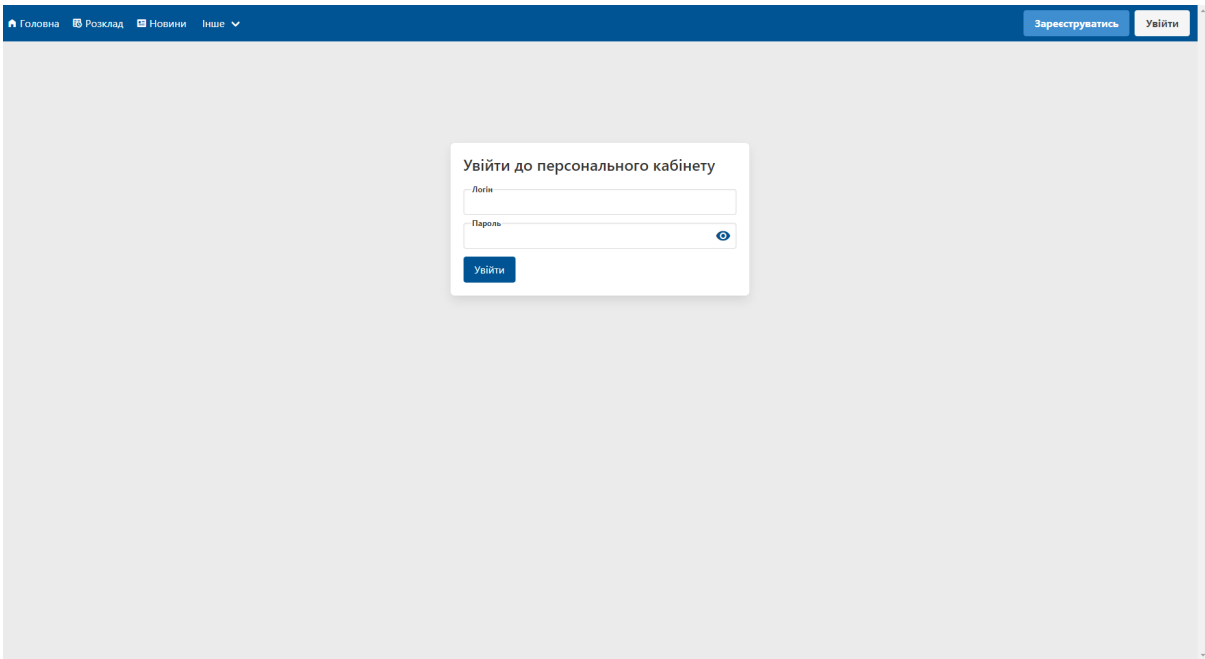


Рисунок 3.7 – Приклад сторінки авторизації

### 3.3. Розробка робочої документації

При постановці задачі (дивиться п. 1.4) визначені вимоги до документації. Відповідно до цих вимог були розроблені наступні документи:

- Технічне завдання (Додаток А).
- Загальний опис системи (Додаток Б).
- Інструкція користувача (Додаток В).
- Програма і методика випробувань (Додаток Г).
- Текст програми (Додаток Д).

Перелік найменувань розроблюваних документів і їх комплектність на ПС "Студент" визначені в технічному завданні (додаток А).

У першому розділі дипломної роботи визначені вимоги до ПС, опис діючої системи та її недоліків, обґрунтування необхідності вдосконалення діючої ІС, функції та задачі створюваної ПС, розробка концепції ПС.

У другому розділі йдеться про розробку проектних рішень щодо ПС та її частин. В цьому розділі визначені рішення з інформаційного, технічного та програмного забезпечень. Загальний опис системи приведений в додатку Б.

Інструкція користувача розробленої ПС приведена в додатку В. В роботі було проведено випробування ПС "Студент", методика та результати приведені в додатку Г.

У додатку Д наведено фрагменти лістингу.

## 4 ОХОРОНА ПРАЦІ

### 4.1 Вступ

Охорона праці - це система охорони життя і здоров'я працівників у процесі трудової діяльності, що включає такі заходи, як закони, соціальна економіка, організація і технології, гігієна та санітарія, лікувально-профілактичні та реабілітаційні заходи.

Умови:

Питання охорони праці регулюються Законом України про працю. Визначення понять і термінів. Охорона праці - це правові, соціально-економічні, організаційні технологічні, санітарно-гігієнічні та лікувально-профілактичні заходи та засоби, спрямовані на збереження життя, здоров'я та працездатності людини в процесі трудової діяльності.

Роботодавець - власник підприємства, установи, організації або його ліцензіари незалежно від форми власності, виду діяльності, керівництва, а також особи, які використовують найману працю.

Працівник - особа, яка працює на підприємстві, в організації, в установі та виконує обов'язки або функції відповідно до трудового договору (контракту).

Дія цього Закону поширюється на всіх юридичних та фізичних осіб, які використовують працівників відповідно до законодавства, та поширюється на всіх працівників.

Законодавство про охорону праці. Законодавство про охорону праці включає цей Закон, Кодекс законів про працю України, Закон України «Про загальнообов'язкове державне соціальне страхування від нещасних випадків на виробництві та профзахворювання з інвалідністю» та нормативно-правові акти, ухвалені відповідно до цих законів.

Якщо міжнародним договором, що є обов'язковим до виконання, ратифікованим Верховною Радою України, передбачені інші норми, ніж ті, що



передбачені законодавством України про охорону праці, застосовуються норми цього міжнародного договору.

Національна політика в галузі охорони праці. Національна політика в галузі охорони праці встановлюється відповідно до Конституції України, Верховної Ради України та спрямована на створення належних і безпечних умов праці та запобігання нещасним випадкам. Національна політика в галузі охорони праці ґрунтується на таких засадах:

- Пріоритет надається життю і здоров'ю працівників, а роботодавці несуть відповідальність за створення належних, безпечних і здорових умов праці;

- підвищення рівня промислової безпеки шляхом забезпечення постійного технічного контролю виробництва, технології та стану продукції, а також сприяння підприємствам у створенні безпечних і нешкідливих умов праці;

- комплексно вирішувати питання охорони праці на основі загальнодержавних, галузевих та регіональних програм з урахуванням інших напрямів економічної та соціальної політики, науково-технічних досягнень та охорони навколишнього середовища;

- соціальний захист працівників з повним відшкодуванням шкоди особам, які постраждали внаслідок нещасних випадків на виробництві та професійних захворювань;

- розробити єдині вимоги охорони праці для всіх суб'єктів господарювання та комерційних структур незалежно від форм власності та видів діяльності;

- здатність адаптувати трудовий процес до працівника з урахуванням стану здоров'я та психологічного стану працівника;

- використання економічного підходу до управління охороною праці, участі держави у фінансуванні заходів з охорони праці, залучення добровільних внесків та інших доходів на цілі, що не суперечать законодавству;

- інформування населення, навчання з охорони праці, професійне навчання та перепідготовка працівників;

- Забезпечити координацію між органами державної влади, установами, організаціями, громадськими об'єднаннями з питань охорони здоров'я, гігієни та

безпеки, а також співпрацю між роботодавцями та працівниками (їх представниками), між усіма соціальними групами у прийнятті рішень з охорони праці та консультаціях на місцевому та державному рівні. рівнів;

- На основі міжнародного співробітництва використовувати досвід Всесвітньої організації праці для покращення умов праці та підвищення безпеки праці.

#### **4.2. Фактори, що впливають на людину**

Тривала робота за комп'ютером може мати серйозний вплив на здоров'я людини. Нині комп'ютери все ширше використовуються у всіх сферах життя, тому все більше людей змушені цілими днями сидіти перед моніторами комп'ютерів.

Розглянемо основні аспекти тривалої роботи за комп'ютером:

1. Люди, які тривалий час працюють за комп'ютером, повинні підтримувати відносно нерухому позу, що негативно позначається на хребті та системному кровообігу (застій крові). Зокрема, інтенсивний застій крові проявляється на рівні органів малого тазу та кінцівок. При тривалих порушеннях кровообігу руйнуються поживні речовини тканини, пошкоджуються стінки судин, що в свою чергу призводить до їх необоротного розширення. Таке розширення судин спостерігається, наприклад, при геморої;

2. Зчитування інформації з монітора може призвести до втоми очей. В основному це пов'язано з тим, що відстань від тексту до ока залишається незмінною при читанні з монітора, тому очні м'язи, які регулюють акомодацию, знаходяться в постійній напрузі. З часом це може призвести до порушення акомодации ока, що може призвести до погіршення зору;

3. Довге введення тексту на клавіатурі призведе до перевантаження м'язів зап'ястя та передпліччя;

4. Монітори, оснащені електронними гарматами, є джерелами сильних електромагнітних полів. Прискорення постійного «бомбардування» електронів по тілу людини призводить до різних розладів нервової системи та очей;

5. Робота за комп'ютером передбачає обробку великого обсягу інформації та безперервну увагу, тому тривала робота за комп'ютером часто призводить до розумової втоми та дефіциту уваги;

6. Люди, які працюють за комп'ютером, часто змушені приймати рішення, які визначають їх продуктивність. Інколи важко передбачити наслідки кроку (особливо в контексті хронічної втоми). Тому тривала робота за комп'ютером часто є причиною хронічного стресу. Зауважте, що потреба мати справу з великою кількістю різномірної (і переважно непотрібної) інформації також може призвести до стресу;

7. З'являється все більше повідомлень про комп'ютерну залежність. Насправді, тривала робота за комп'ютером, робота в Інтернеті та комп'ютерні ігри можуть призвести до подібних психічних розладів;

8. Робота за комп'ютером нерідко поглинає всю увагу працюючої людини і тому, такі люди часто нехтують нормальним харчуванням і працюють впроголодь весь день. Неправильне харчування призводить не тільки до порушень роботи органів травного тракту, а й до виникнення мінеральної та вітамінної недостатності. Відомо, що нестача вітамінів і мінералів негативно позначається на процесі обміну речовин в організмі, що призводить до зниження інтелектуальних здібностей людини. Зниження ефективності роботи що в свою чергу, викликає необхідність знаходитися ще більше часу за комп'ютером. Таким чином, утворюється своєрідний «порочне коло», в якому тривала робота за комп'ютером є пусковим моментом визначаючим всі наступні порушення.

Захворювання прямої кишки. Серед захворювань прямої кишки найчастіше зустрічається геморої. Висока захворюваність на ці захворювання серед людей, які тривалий час проводять за комп'ютером, пов'язана не з шкідливим впливом останнього на організм людини, а через те, що оператор комп'ютера тривалий час перебуває в сидячому положенні. Геморої - це

варикозне розширення вен нижньої частини прямої кишки. Основною причиною такого розширення є застій крові в цих органах при малорухомому веденні. При цьому варикозне розширення вен виступає в просвіт прямої кишки і навіть відвисає від заднього проходу. У деяких випадках можливий розвиток тромбозу або інфікування гемороїдальних вен. При цьому спостерігається інтенсивний біль і кровотеча з отвору заднього проходу.

Захворювання зап'ястя. Тривала робота за комп'ютером може призвести до серйозних нервово-м'язових розладів. Особливо чутливими частинами тіла є пальці, зап'ястя та передпліччя. При роботі за комп'ютером руки виконують переважно механічну роботу, і важлива не величина фізичної активності (зазвичай невелика), а тривалість роботи. Як ми всі знаємо, подушечка пальця є найбільш чутливою частиною людського тіла. На цьому рівні зосереджена велика кількість чутливих нервових закінчень (отже, пальці виконують функцію дотику). Під час багатогодинної роботи за комп'ютером (на клавіатурі) постійно збуджуються нервові закінчення пальців. З часом це призводить до виснаження нервових шляхів, які з'єднують пальці з корою головного мозку. В результаті порушується координація рухів пальців і судоми в кисті та передпліччя. Британські дослідники назвали хворобу RSI (Repetitive Strain Injury) або хронічне захворювання зап'ястя. Захворювання кістково-м'язової системи. Часто тривале використання комп'ютера може призвести до порушення постави або викривлення хребта. Найбільш сприйнятливі до цього захворювання діти з викривленням хребта, переданим за типом сколіозу, тобто викривлення хребта (бічні). У дорослих між міжхребцевими дисками можуть виникнути грижі, які викликають біль у нервовому корінці та ішіас.

Основною причиною розвитку захворювань хребта є неправильне положення на робочому місці. Часто працівник адаптується і через деякий час перестає відчувати, що положення сидючи неправильне, тому хвороба продовжує розвиватися.

Захворювання нервової системи. Робота за комп'ютером – це суто інтелектуальна праця. Так основна частина навантаження припадає на нервову систему, головний мозок.

#### **4.3. Заходи безпеки**

Перед входом на робоче місце працівники повинні отримати вступні та попередні інструкції з техніки безпеки для демонстрації прикладів безпечної та розумної роботи. Потім періодичне навчання проводиться не рідше ніж кожні 6 місяців, можливо, групою співробітників за тією ж спеціальністю чисельністю не більше 20 осіб. При зміні правил охорони праці, порушенні інструкцій з техніки безпеки, зміні характеру роботи працівників вони повідомляються нерегулярно.

У місцях, де часто використовується електрообладнання, «Інструкції з безпеки» мають бути вивішені там, де люди можуть дістатися, а також вони повинні визначати поведінку людей у разі аварії, пожежі або ураження електричним струмом. Керівник структурного підрозділу відповідає за організацію правильної та безпечної роботи комп'ютерної техніки та периферійних пристроїв, а також їх ефективне використання, контроль за дотриманням персоналом вимог цієї інструкції з техніки безпеки.

#### **4.4. Вимоги щодо забезпечення пожежної безпеки**

На робочих місцях заборонено використовувати легкозаймисті матеріали.

Заборонені місця:

- а) розпалити багаття;
- б) Якщо в приміщенні пахне газом, увімкніть електрообладнання;
- в) куріння;
- г) сушіння чогось на обігрівачі;
- е) Закрийте вентиляційні отвори електрообладнання.

Джерелами займання є:

- а) іскри під час електростатичного розряду;
- б) іскри від електрообладнання;
- в) іскри від удару та тертя;
- г) відкритий вогонь.

У разі виникнення пожежі або пожежі персонал повинен негайно вжити необхідних заходів щодо її ліквідації та повідомити пожежну частину.

Місця електрообладнання повинні бути обладнані вогнегасником типу ОУ-2 або ОУБ-3.

#### **Висновки за розділом 4**

Психічне здоров'я та соціальна свобода сучасної людини все більше залежать від її здібностей у сфері інформаційних технологій, що стає необхідним чинником визнання в очах інших, успішного функціонування в різних сферах та досягнення ідеального суспільства. . статус. У цьому розділі розглядаються питання охорони праці при використанні ЕОМ, пожежної безпеки, законодавчої бази у сфері охорони праці.

## ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було розроблено підсистему «Студент» інформаційної системи кафедри інформаційних управляючих систем Національного університету кораблебудування імені адмірала Макарова. Для розробки підсистеми було вирішено наступні задачі:

- виконано аналіз предметної області і визначено недоліки існуючих систем та рішень;
- розроблено концепцію «Підсистеми Студент»;
- розроблено проектні рішення;
- розроблено технічне завдання;
- проведено необхідні дослідження з охорони праці.

Для аналізу сучасних інформаційних технологій, що використовуються для розробки інформаційних систем, був проведений огляд літератури і Internet-джерел. У результаті дослідження були описані необхідні технології для розробки інформаційної системи.

При формуванні вимог підсистеми була вивчена діюча неавтоматизована інформаційна система кафедр університетів і її недоліки, обґрунтована необхідність автоматизації діючої системи, визначені цілі, критерії, обмеження, функції та задачі розробленої інформаційної підсистеми.

Для розробки загальносистемних рішень була побудована функціональна модель системи за допомогою діаграми переходів станів.

У процесі розробки рішень з інформаційного забезпечення були розроблені концептуальна, логічна і фізична модель даних для бази даних системи.

Було реалізоване необхідне програмне забезпечення системи засобами мов програмування JavaScript.

Розроблена необхідна документація до системи, включає наступні документи: технічне завдання, інструкцію користувача, текст програми.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Матеріали сторінки: <http://iust.nuos.edu.ua/>
2. Матеріали сторінки: <https://nuos.edu.ua/pro-universitet/struktura/>
3. Матеріали сторінки: <https://nuos.edu.ua/pro-universitet/zagalna-dokumentaciya/>
4. Ажищев В.Ф., Кошкін К.В., Морозова А.С. «Методичні вказівки до виконання дипломного проектування». – Міністерство освіти та науки України, НУК ім. адмірала Макарова, 2014. – 3.
5. Матеріали сторінки:  
[https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0\\_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)
6. Матеріали сторінки:  
[https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0\\_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)
7. ISO/IEC/IEEE 12207:2017 Systems and software engineering – Software life cycle processes. URL: <https://www.iso.org/ru/standard/63712.html>
8. Quality of Open Source Software: how many eyes are enough? URL: <https://blogs.worldbank.org/opendata/quality-open-source-software-how-many-eyes-are-enough>
9. Положення про випускну кваліфікаційну роботу у Національному університеті кораблебудування імені адмірала Макарова. URL: <https://nuos.edu.ua/wp-content/uploads/2020/10/Polozhennya-pro-vipusknu-kvalifikacijnu-robotu-NUK2020.pdf>



10. Методичні вказівки до виконання кваліфікаційної роботи за спеціальністю 126 – "Інформаційні системи та технології" галузі знань 12 – "Інформаційні технології" першого (бакалаврського) рівня вищої освіти / І. Л. Михелєв, А. Ю. Гайда, О. В. Гайдаєнко, Н. Р. Книрик, Є. Ю. Беркунський, О. І. Маршак, Г. С. Морозова, – Миколаїв : НУК, 2021, – 12 с.

11. ГОСТ 12.0.003-74 Небезпечні та шкідливі виробничі фактори. Класифікація. ГОСТ (Міждержавний стандарт). URL: [http://online.budstandart.com/ua/catalog/doc-page.html?id\\_doc=48127](http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=48127)

## **ДОДАТОК А**

### **ТЕХНІЧНЕ ЗАВДАННЯ**

#### **Загальні відомості**

Потрібно автоматизувати процеси інформаційної взаємодії із студентами на кафедрі ІУСТ. Назва розробки «Підсистема Студент».

#### **Призначення і мета створення підсистеми «Студент»**

Призначенням цієї підсистеми є автоматизація взаємодій, які використовуються на кафедрі ІУСТ, створення інформаційної бази даних де буде зберігатися та оброблюватися інформація про персонал, предмети, події, розклад таких навчальних процесів як пари, сесії та іспити.

Метою створення підсистеми є спрощення способів отримання інформації студентам, а також покращення зручності використання інформації навчальної кафедри.

#### **Вимоги до підсистеми**

Дати можливість використовувати підсистему студентам для отримання необхідної інформації, а саме розклад групи, або тижневий розклад предметів певного викладача на кафедрі навчання. А так само підсистема повинна мати відкритий доступ до інформації з бази даних для подальшого використання у візуальному поданні даних, таких як веб-сайт, або сторінка в месенджерах, де складаються студенти і можливо організовується робочий процес групи з викладачами та куратором. Ця підсистема повинна мати простий і лаконічний інтерфейс управління та перегляду інформації, для спрощення подачі та використання інформації. Компоненти цієї підсистеми повинні бути реалізовані з використанням мікро сервісної архітектури, для можливості подальшого масштабування в участі в ланцюзі більших інформаційних мереж. Зазначені функції підсистеми повинні мати гнучкий інтерфейс отримання/віддачі даних, для можливості інтеграції нових функцій та підтримання актуальності тривалий час використання, з урахуванням мінімальних змін в освітній процедурі навчального закладу.

### **Вимоги до функцій АРМ**

Підсистема «Студент» має виконувати такі функції, що наведені у списку:

Надання даних із внутрішньої системи:

- 1) отримання списку предметів;
- 2) пошук предмета;
- 3) отримання списку викладачів, пошук за різними фільтрами;
- 4) отримання списку груп;
- 5) отримання групи за ім'ям та датою надходження;
- 6) отримання списку аудиторій;
- 7) одержання розкладу групи, розкладу кафедри, розкладу викладача, розклад аудиторії;
- 8) отримання загальних повідомлень кафедри ІУСТ.

Ідентифікація відвідувачів та розмежування ролей:

- 1) Авторизація користувача;
- 2) Реєстрація користувача;
- 3) Ідентифікація ролі користувача (за номером залікової книжки та цифрового підпису отриманого на кафедрі);
- 4) Ідентифікація приналежності користувача до навчальної групи;
- 5) Перегляд повідомлень групи користувача;
- 6) Перегляд розкладу групи користувача;
- 7) Перегляд персональних повідомлень користувача.

Адміністрація системи:

- 1) авторизація користувача ролі адміністратора;
- 2) додавання даних викладачів;
- 3) додавання даних аудиторій;
- 4) додавання даних викладачів;
- 5) додавання даних предметів;

- 6) заповнення розкладу на основі доданих даних сутностей;
- 7) інтеграція системи із сторонніми вузлами інтерфейсів користувачів;
- 8) інтеграція на каналах месенджерів;
- 9) інтеграція у групових чатах месенджерів;
- 10) інтеграція чат-ботів у різних месенджерах.

### **Вимоги до інформаційного забезпечення підсистеми «Студент»**

Система повинна підтримувати розподілену архітектуру для можливості горизонтального масштабування вузлів обробки інформації.

Система має бути відкритою за визначенням, прийнятим Комітетом IEEE POSIX 1003.0 (Відкритою інформаційною системою називається система, яка реалізує відкриті специфікації на інтерфейси, сервіси (послуги середовища) та підтримувані формати даних, достатні для того, щоб дати можливість належним чином розробленому прикладному програмному забезпеченню переноситися в широкому діапазоні систем з мінімальними змінами, взаємодіяти з іншими додатками. на локальних та віддалених системах, та взаємодіяти з користувачами у стилі, що полегшує перехід користувачів від системи до системи) [6].

Система повинна забезпечувати захист при передачі конфіденційної інформації від перехоплення за допомогою шифрування, а також ідентифікацію ролей користувачів під час виконання запитів до системи. Система має бути розділена на кілька багаторівневих сервісів, для спрощення взаємодії між функціональними вузлами на різних рівнях доступу до інформації, а також покращить можливості для масштабування, що збільшить потенціал обробки більшої кількості запитів.

### **Вимоги до лінгвістичного забезпечення та програмного забезпечення підсистеми «Студент»**

Підсистема «Студент» розробляється мовою програмування TypeScript, яка компілюється та інтерпретується у мову JavaScript та запускається фреймворком серверного виконання – NodeJS.

До підсистеми «Студент» входить сервер на якому працюють вузли сервісів системи та клієнти (браузер, чат-боти та інше).

Вимоги до сервера для розгортання системи:

- операційна система: будь яка система, яка підтримує віртуалізацію (наприклад Linux – Debian 10+, Ubuntu 16+, Windows - Windows 7 або новіше);
- Програмне забезпечення Docker версії 16 або новіше.

Вимоги до програмного забезпечення клієнтського девайсу:

- Операційна система: будь-яка операційна система у який можна встановити будь-який сучасний веб-браузер.

### **Вимоги до технічного забезпечення підсистеми «Студент»**

Вимоги до технічного забезпечення сервера для розгортання системи:

- Процесор із наявними 2 або більше потоків виконання операцій, и тактовою частотою 1.3ггц або вище. Архітектура процесора будь-яка (ARM, x86 та інші).
- Оперативна пам'ять RAM 2Гб або більше.
- Накопичувач SSD диск із вільним місцем 10Гб або більше.
- Мережеве підключення Ethernet щонайменше 10мбіт/с.

Вимоги до технічного забезпечення клієнтського девайсу:

Будь який сучасний девайс який надає змогу запускати та відображати веб сторінки із виконанням JavaScript.

## ДОДАТОК Б

### ЗАГАЛЬНИЙ ОПИС СИСТЕМИ

Мета розробки підсистеми «Студент» -- підвищення ефективності та продуктивності навчального процесу.

Функції що може виконувати студент:

- авторизуватися в системі;
- переглядати розклад;
- переглядати інформацію про викладачів;
- переглядати інформацію про предмети;
- переглядати новини;

Функції що може виконувати викладач або керівник групи:

- авторизуватися в системі;
- заповнювати розклад;
- заповнювати інформацію предметів;
- створювати новини;
- створювати повідомлення для груп;
- створювати повідомлення для студентів;

Опис підсистеми «Студент»

Користувачами системи є здобувачі вищої освіти та викладачі ЗВО. Також для керування ресурсами системи потрібен адміністратор, у даному випадку роллю адміністратора може володіти будь який викладач кафедри.

При переході до системи, система дозволяє переглядати деяку інформацію у гостевому стані, або користувач має змогу авторизуватись – тобто перейти до стану авторизації.

Файли баз даних та статичні дані завантажені користувачами на сайті зберігаються у файловій системі серверного накопичувача. Обсяг пам'яті повинен бути не менше 1 Гб. Структура файлів буде описана в наступних підрозділах.

База даних повинна зберігатися в системі Postgres. База даних розташується на накопичувачі основного серверу. Для підвищення швидкодії бази даних при запису даних на накопичувач необхідно використовувати файлову систему XFS яка доступна лише під керуванням операційної системи Linux.

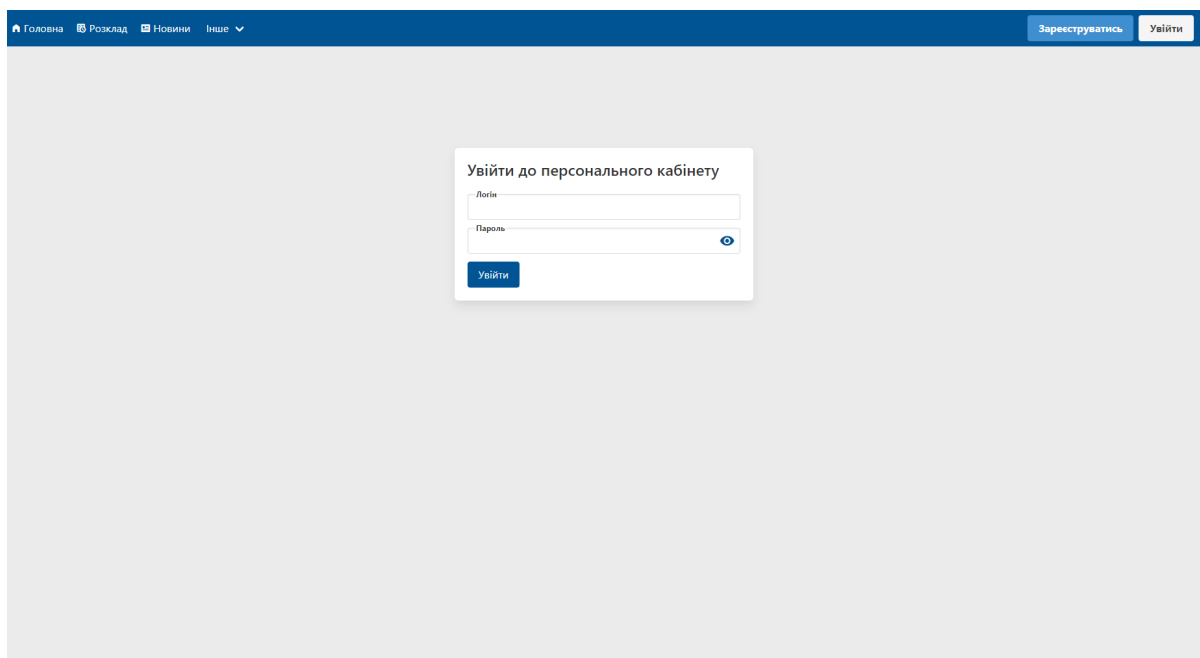
Організація збору і передачі інформації.

Введення інформації в базу даних системи повинно здійснюватися за допомогою програмного забезпечення, що розроблено за допомогою технологій веб сайтів та веб браузеру.

## ДОДАТОК В

### ІНСТРУКЦІЯ КОРИСТУВАЧА

#### Початок роботи



#### Рисунок В.1 – Окно авторизації

Робота починається з авторизації користувача в системі. Для цього у вікні, що з'являється після запуску програми, потрібно обрати ім'я користувача, ввести пароль та натиснути кнопку "Увійти". (на рис. В.1)

#### Відображення розкладу

На рис. В.2 – Перегляд розкладу



Головна

Розклад

Новини

Інше

Зареєструватись

Увійти

Розклад групи

Вибір групи

4141

4144

Понеділок

| № | Предмет                                                  | Викладач       | Локація | Часи          |
|---|----------------------------------------------------------|----------------|---------|---------------|
| 1 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 9:00 - 10:20  |
| 2 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 10:30 - 11:50 |
| 3 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 12:20 - 13:40 |
| 4 | -                                                        | -              | -       | 13:50 - 15:10 |
| 5 | -                                                        | -              | -       | 15:20 - 16:40 |

Вівторок

| № | Предмет                                                  | Викладач       | Локація | Часи          |
|---|----------------------------------------------------------|----------------|---------|---------------|
| 1 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 9:00 - 10:20  |
| 2 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 10:30 - 11:50 |
| 3 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 12:20 - 13:40 |
| 4 | -                                                        | -              | -       | 13:50 - 15:10 |
| 5 | -                                                        | -              | -       | 15:20 - 16:40 |

Середа

| № | Предмет                                                  | Викладач       | Локація | Часи          |
|---|----------------------------------------------------------|----------------|---------|---------------|
| 1 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 9:00 - 10:20  |
| 2 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 10:30 - 11:50 |
| 3 | Програмне забезпечення розподілених інформаційних систем | Павленко А. Ю. | 5126    | 12:20 - 13:40 |
| 4 | -                                                        | -              | -       | 13:50 - 15:10 |
| 5 | -                                                        | -              | -       | 15:20 - 16:40 |

Четвер

| № | Предмет | Викладач | Локація | Часи |
|---|---------|----------|---------|------|
|---|---------|----------|---------|------|

localhost:3004

## **ДОДАТОК Г**

### **ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ**

#### **Об'єкт випробувань**

Об'єктом випробувань є підсистема «Студент», яке призначене для автоматизації розкладу кафедри ІУСТ.

#### **Цілі випробувань**

Цілі випробувань: перевірка придатності підсистеми до використання по призначенню, її відповідність заданим вимогам і документації; перевірка працездатності і правильності роботи системи.

#### **Вимоги до системи**

Повний список вимог та функцій до системи приведений у пункті 1.4 пояснювальної записки.

#### **Вимоги до програмної документації**

У комплект програмної документації повинні входити наступні документи:

- Технічне завдання.
- Загальний опис системи.
- Інструкція користувача.
- Програма і методика випробувань.
- Текст програми.

#### **Засоби і порядок випробувань**

Вимоги до складу і параметрів технічних і програмних засобів вказані в пункті 1.4 пояснювальної записки. Для правильної установки і настройки програмного забезпечення системи необхідно скористатися інструкцією з формування і ведення бази даних.

Перед початком (не пізніше 3 днів) випробувань уточнюється кінцевий склад групи, що буде проводити випробування. Кожний член групи отримує

необхідну документацію до системи (технічне завдання, загальний опис системи, керівництво користувача, програма і методика випробувань).

### **Порядок проведення випробувань:**

- Виконуються контрольні тести в довільному порядку. Результати заносяться у відповідний журнал.
- Робиться аналіз отриманих результатів і встановлюється відповідність програми заданим вимогам і системним документам.

При виявленні помилок у роботі системи складається їхній перелік і обговорюється термін їхнього виправлення розробником. Після цього замовник проводить повторне тестування в повному обсязі (можливо використання нових чи додаткових тестів).

### **Методика випробувань**

Процес тестування являє собою експлуатацію системи в контрольованих умовах і вивчення отриманих результатів. При цьому перевіряється робота системи з нормальними і помилковими даними і подіями. Тестування розподілених систем сильно ускладнено великою кількістю взаємозв'язків. Найкраще для тестування розподілених систем застосовувати метод «знизу-нагору». Спочатку розробник повинен перевірити кожен компонент (елемент системи) окремо; потім – протестувати взаємодію компонента з іншими компонентами системи на автономному комп'ютері; і тільки потім – перевірити роботу всієї системи в розподіленому середовищі. Усі ці види тестування виконуються розробником системи без участі замовника. Розробник не зобов'язаний надавати тести і протоколи результатів тестування кому-небудь.

При прийманні системи проводяться випробування системи в розподіленому середовищі. Замовник і/або його представники самостійно проводять тестування усіх функцій системи методом «чорного ящика» у присутності розробника.

## Програма випробувань

Тести складаються замовником і/або його представниками. При цьому перевіряється функціонування системи з нормальними і помилковими даними і подіями.

Наприклад:

- Аутентифікація користувача з правильними даними.
- Аутентифікація користувача з неправильними даними.

Під неправильними чи помилковими даними в контексті даної системи потрібно розуміти:

- Логін і/або пароль користувача, яких немає у файлах аутентифікації (для перших двох тестів).
- Неіснуюча дата (для всіх інших тестів).

Програмне забезпечення розроблено таким чином, що користувач не може ввести які-небудь інші неправильні дані в систему через наданий йому інтерфейс. Таким чином, будь-які дані, що не є неправильними, потрібно вважати правильними.

- Для визначення переліку вхідних даних для кожного тесту можна скористатися формами для введення відповідних даних. У формах зазначені необхідні поля.
- Вихідні дані чи результат при правильних даних повинні відповідати виконанню функції, для тестування якої створений тест; при неправильних даних – відмовлення в доступі до системи (тест 2) чи повідомлення про помилку.

## ДОДАТОК Д

### ФРАГМЕНТИ ЛІСТИНГУ

main.ts - запуск

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app/app.module';
import { ValidationPipe } from '@nestjs/common';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
    }),
  );
  // app.useGlobalInterceptors(new ClassSerializerInterceptor(app.get(Reflector)));
  app.enableCors({
    origin: '*',
  });

  const options = new DocumentBuilder()
    .setTitle('NUOS Core API')
    .setDescription('API Doc')
    .setVersion('1.0')
    .addBearerAuth()
    .build();
  const document = SwaggerModule.createDocument(app, options);
  SwaggerModule.setup("", app, document);

  await app.listen(process.env.PORT || 3000, '0.0.0.0');
  console.log(`Application is running on: ${await app.getUrl()}`);
}
```

```
bootstrap();
```

timetable.service.ts – сервіс керування розкладом

```
import { Injectable } from '@nestjs/common';
import { CreateTimetableDto } from './dto/create-timetable.dto';
import { UpdateTimetableDto } from './dto/update-timetable.dto';
import { InjectRepository } from '@mikro-orm/nestjs';
import { EntityRepository } from '@mikro-orm/core';
import { TimetableLessonItem } from './entities/timetable-lesson-item.entity';
import { Timetable } from './entities/timetable.entity';
import { TimetableDay } from './entities/timetable-day.entity';
import { TimetableItem } from './entities/timetable-item.entity';
import { StudentsGroupsService } from '../students-groups/students-groups.service';
import { CreateTimetableDayDto } from './dto/create-timetable-day.dto';
import { CreateTimetableDayLessonDto } from './dto/create-timetable-day-lesson.dto';
import { SubjectsService } from '../subjects/subjects.service';
import { PersonsService } from '../persons/persons.service';
import { LocationsService } from '../locations/locations.service';
import { UpdateTimetableDayLessonDto } from './dto/update-timetable-day-lesson.dto';
```

```
@Injectable()
```

```
export class TimetableService {
```

```
  constructor(
```

```
    @InjectRepository(Timetable)
```

```
    private readonly timetableEntityRepository: EntityRepository<Timetable>,
```

```
    @InjectRepository(TimetableDay)
```

```
    private readonly timetableDayEntityRepository: EntityRepository<TimetableDay>,
```

```
    @InjectRepository(TimetableItem)
```

```
    private readonly timetableItemEntityRepository: EntityRepository<TimetableItem>,
```

```
    @InjectRepository(TimetableLessonItem)
```

```
    private readonly timetableLessonItemEntityRepository:
```

```
    EntityRepository<TimetableLessonItem>,
```

```

    private readonly studentsGroupsService: StudentsGroupsService,
    private readonly subjectsService: SubjectsService,
    private readonly personsService: PersonsService,
    private readonly locationsService: LocationsService,
  ) {}

```

```

  async create(createTimetableDto: CreateTimetableDto) {
    const timetable = this.timetableEntityRepository.create({
      ...createTimetableDto,
      studentsGroup: await this.studentsGroupsService.findOne(
        createTimetableDto.studentsGroup,
      ),
    });
    await this.timetableEntityRepository.persistAndFlush(timetable);

    return timetable;
  }

```

```

  async findAll() {
    return this.timetableEntityRepository.findAll();
  }

```

```

  findOne(id: number) {
    return this.timetableEntityRepository.findOneOrFail(
      {
        id,
      },
      {
        populate: [
          'studentsGroup',
          'days',
          'days.lessons',
          'days.lessons.item',
        ],
      },
    );
  }

```

```

    ],
  },
);
}

```

```

update(id: number, updateTimetableDto: UpdateTimetableDto) {
  return this.timetableEntityRepository.nativeUpdate(
    { id },
    updateTimetableDto,
  );
}

```

```

remove(id: number) {
  return this.timetableEntityRepository.nativeDelete({ id });
}

```

```

async createTimetableDay(id: number, dto: CreateTimetableDayDto) {
  const day = this.timetableDayEntityRepository.create({
    timetable: await this.timetableEntityRepository.findOneOrFail({ id }),
    day: dto.day,
  });
}

```

```

  await this.timetableDayEntityRepository.persistAndFlush(day);
  return day;
}

```

```

removeTimetableDay(timetable: number, day: number) {
  return this.timetableDayEntityRepository.nativeDelete({ timetable, day });
}

```

```

async createTimetableDayLesson(
  id: number,
  dayNumber: number,

```



```

lessonDto: CreateTimetableDayLessonDto,
prevDayId: number,
nextDayId: number,
) {
  const timetableItem = this.timetableItemEntityRepository.create({
    location: lessonDto.location
      ? await this.locationsService.findOne(lessonDto.location)
      : null,
    subject: lessonDto.subject
      ? await this.subjectsService.findOne(lessonDto.subject)
      : null,
    teacher: lessonDto.teacher
      ? await this.personsService.findOne(lessonDto.teacher)
      : null,
    subjectType: lessonDto.subjectType
      ? await this.subjectsService.findTypeOne(lessonDto.subjectType)
      : null,
  });
  const timetableDay = await this.timetableDayEntityRepository.findOneOrFail({
    timetable: id,
    day: dayNumber,
  });
  await this.timetableItemEntityRepository.persistAndFlush(timetableItem);
  const previous =
    prevDayId !== 0
      ? await this.timetableLessonItemEntityRepository.findOne({
          id: prevDayId,
          timetableDay,
        })
      : await this.timetableLessonItemEntityRepository.findOne({
          next: null,
          timetableDay,
        });
}

```

```

const next =
  nextDayId !== 0
    ? await this.timetableLessonItemEntityRepository.findOne({
        id: nextDayId,
        timetableDay,
      })
    : null;
const timetableLessonItem = this.timetableLessonItemEntityRepository.create(
  {
    item: timetableItem,
    timetableDay,
    previous,
    next,
  },
);
if (previous) previous.next = timetableLessonItem;
if (next) next.previous = timetableLessonItem;
await this.timetableLessonItemEntityRepository.persistAndFlush([
  timetableLessonItem,
  ...(previous ? [previous] : []),
  ...(next ? [next] : []),
]);
return timetableLessonItem;
}

async updateTimetableDayLesson(
  id: number,
  dayNumber: number,
  lessonId: number,
  lessonDto: UpdateTimetableDayLessonDto,
  prevDayId: number,
  nextDayId: number,
) {

```

```

const lessonItem =
  await this.timetableLessonItemEntityRepository.findOneOrFail({
    id: lessonId,
  });
if (prevDayId !== 0) {
  lessonItem.previous =
    await this.timetableLessonItemEntityRepository.findOneOrFail({
      id: prevDayId,
    });
}
if (nextDayId !== 0) {
  lessonItem.next =
    await this.timetableLessonItemEntityRepository.findOneOrFail({
      id: nextDayId,
    });
}
await this.timetableLessonItemEntityRepository.persistAndFlush(lessonItem);
}

deleteTimetableDayLesson(id: number) {
  return this.timetableLessonItemEntityRepository.nativeDelete({ id });
}
}

```

timetable.controller.ts – контролер http запитів розкладу

```

import {
  Body,
  Controller,
  DefaultValuePipe,
  Delete,
  Get,
  Param,

```

```

    ParseIntPipe,
    Patch,
    Post,
    Query,
    UseFilters,
} from '@nestjs/common';
import { TimetableService } from './timetable.service';
import { CreateTimetableDto } from './dto/create-timetable.dto';
import { UpdateTimetableDto } from './dto/update-timetable.dto';
import { ApiQuery, ApiTags } from '@nestjs/swagger';
import { CreateTimetableDayDto } from './dto/create-timetable-day.dto';
import { DbUniqueExceptionFilter } from '../db-unique-exception.filter';
import { CreateTimetableDayLessonDto } from './dto/create-timetable-day-lesson.dto';
import { UpdateTimetableDayLessonDto } from './dto/update-timetable-day-lesson.dto';

@ApiTags('Timetable')
@Controller('timetable')
export class TimetableController {
    constructor(private readonly timetableService: TimetableService) {}

    @Post()
    create(@Body() createTimetableDto: CreateTimetableDto) {
        return this.timetableService.create(createTimetableDto);
    }

    @Get()
    findAll() {
        return this.timetableService.findAll();
    }

    @Get('/:id')
    findOne(@Param('id', new ParseIntPipe()) id: number) {
        return this.timetableService.findOne(id);
    }

```

```
}
```

```
@UseFilters(DbUniqueExceptionFilter)
@Post('/:id/day')
createTimetableDay(
    @Param('id', new ParseIntPipe()) id: number,
    @Body() createTimetableDayDto: CreateTimetableDayDto,
) {
    return this.timetableService.createTimetableDay(id, createTimetableDayDto);
}
```

```
@Delete('/:id/:day_number')
deleteTimetableDay(
    @Param('id', new ParseIntPipe()) id: number,
    @Param('day_number', new ParseIntPipe()) dayNumber: number,
) {
    return this.timetableService.removeTimetableDay(id, dayNumber);
}
```

```
@ApiQuery({ name: 'prev_day_id', required: false })
@ApiQuery({ name: 'next_day_id', required: false })
@Post('/:id/:day_number/lesson')
createTimetableDayLesson(
    @Param('id', new ParseIntPipe()) id: number,
    @Param('day_number', new ParseIntPipe()) dayNumber: number,
    @Query('prev_day_id', new DefaultValuePipe(0), new ParseIntPipe())
    prevDayId: number,
    @Query('next_day_id', new DefaultValuePipe(0), new ParseIntPipe())
    nextDayId: number,
    @Body() lessonDto: CreateTimetableDayLessonDto,
) {
    return this.timetableService.createTimetableDayLesson(
        id,
```

```

    dayNumber,
    lessonDto,
    prevDayId,
    nextDayId,
  );
}

```

```

@ApiQuery({ name: 'prev_day_id', required: false })
@ApiQuery({ name: 'next_day_id', required: false })
@Patch('/:id/:day_id/:lesson_id')
updateTimetableDayLesson(
  @Param('id', new ParseIntPipe()) id: number,
  @Param('day_number', new ParseIntPipe()) dayNumber: number,
  @Param('lesson_id', new ParseIntPipe()) lessonId: number,
  @Query('prev_day_id', new DefaultValuePipe(0), new ParseIntPipe())
  prevDayId: number,
  @Query('next_day_id', new DefaultValuePipe(0), new ParseIntPipe())
  nextDayId: number,
  @Body() lessonDto: UpdateTimetableDayLessonDto,
) {
  return this.timetableService.updateTimetableDayLesson(
    id,
    dayNumber,
    lessonId,
    lessonDto,
    prevDayId,
    nextDayId,
  );
}

```

```

@Delete('/:id/:day_number/:lesson_id')
deleteTimetableDayLesson(
  @Param('id', new ParseIntPipe()) id: number,

```

```

    @Param('day_number', new ParseIntPipe()) dayNumber: number,
    @Param('lesson_id', new ParseIntPipe()) lessonId: number,
  ) {
    return this.timetableService.deleteTimetableDayLesson(lessonId);
  }

  @Patch('/:id')
  update(
    @Param('id', new ParseIntPipe()) id: number,
    @Body() updateTimetableDto: UpdateTimetableDto,
  ) {
    return this.timetableService.update(id, updateTimetableDto);
  }

  @Delete('/:id')
  remove(@Param('id', new ParseIntPipe()) id: number) {
    return this.timetableService.remove(id);
  }
}

```

person.entity.ts – сутність особи

```

import { Entity, ManyToOne, PrimaryKey, Property } from '@mikro-orm/core';
import { ApiProperty } from '@nestjs/swagger';
import { PersonPosition } from '../persons-positions/entities/person-position.entity';

@Entity()
export class Person {
  @ApiProperty()
  @PrimaryKey()
  id!: number;

  @ApiProperty()
  @Property()

```

```
name!: string;
```

```
@ApiProperty()
```

```
@Property()
```

```
surname!: string;
```

```
@ApiProperty()
```

```
@Property({ nullable: true })
```

```
middlename!: string;
```

```
@ApiProperty()
```

```
@Property({ nullable: true })
```

```
email!: string;
```

```
@ApiProperty()
```

```
@Property({ nullable: true })
```

```
phone!: string;
```

```
@ApiProperty({
```

```
  type: PersonPosition,
```

```
})
```

```
@ManyToOne(() => PersonPosition)
```

```
position: PersonPosition;
```

```
}
```

timetable-lesson-item.entity.ts – сутність частини розкладу

```
import { Entity, ManyToOne, PrimaryKey } from '@mikro-orm/core';
```

```
import { ApiProperty } from '@nestjs/swagger';
```

```
import { TimetableDay } from './timetable-day.entity';
```

```
import { TimetableItem } from './timetable-item.entity';
```

```
@Entity()
```

```
export class TimetableLessonItem {
```



```
@ApiProperty()
```

```
@PrimaryKey()
```

```
id!: number;
```

```
@ApiProperty()
```

```
@ManyToOne(() => TimetableItem)
```

```
item: TimetableItem;
```

```
@ApiProperty()
```

```
@ManyToOne(() => TimetableLessonItem, { nullable: true })
```

```
next: TimetableLessonItem;
```

```
@ApiProperty()
```

```
@ManyToOne(() => TimetableLessonItem, { nullable: true })
```

```
previous?: TimetableLessonItem;
```

```
@ApiProperty()
```

```
@ManyToOne(() => TimetableDay)
```

```
timetableDay: TimetableDay;
```

```
}
```

users.service.ts – сервіс керування користувачами

```
import { Injectable, OnApplicationBootstrap } from '@nestjs/common';
```

```
import { UpdateUserDto } from './dto/update-user.dto';
```

```
import { InjectRepository } from '@nestjs/typeorm';
```

```
import { User } from './entities/user.entity';
```

```
import { Repository } from 'typeorm';
```

```
import { UserProfile } from './entities/user-profile.entity';
```

```
import qrcode from 'qrcode';
```

```
import { join } from 'path';
```

```
import { access } from 'fs/promises';
```

```
import { ConfigService } from '@nestjs/config';
```

```
import { FileEntity } from '../files/entities/file.entity';
```

```

import { FileService } from '../files/files.service';
import { AuthRegisterDto } from '../auth/dto/auth-register.dto';
import { nanoid } from 'nanoid';
import * as fs from 'fs';
import * as bcrypt from 'bcrypt';
import { UserRole } from '../entities/user-role.entity';

```

```
@Injectable()
```

```

export class UsersService implements OnApplicationBootstrap {
  constructor(
    @InjectRepository(User)
    private usersRepository: Repository<User>,
    @InjectRepository(UserProfile)
    private userProfileRepository: Repository<UserProfile>,
    @InjectRepository(UserRole)
    private userRoleRepository: Repository<UserRole>,
    private configService: ConfigService,
    private fileService: FileService,
  ) {}

```

```

  async create(dto: AuthRegisterDto): Promise<User> {
    const user = await this.usersRepository.create({
      name: dto.name,
      username: dto.username,
      email: dto.email,
      password: dto.password,
    });
    if (user) {
      user.profile = await this.userProfileRepository.create({});
      await this.userProfileRepository.save(user.profile);

      await this.usersRepository.save(user);
    }
  }

```

```

    return user;
  } else {
    throw new Error('nouser');
  }
}

async generateUserQRCode(user: User, force = false) {
  const url = new URL(this.configService.get<string>('FRONTEND_BASE_URL'));
  url.pathname = '/tips/checkout';
  url.searchParams.append('user_id', String(user.id));
  const filePath = join(
    process.cwd(),
    'files/qr-codes',
    String(user.id).concat('.png'),
  );
  if (!force) {
    try {
      await access(filePath);
      return null;
    } catch {}
  }
  await qrcode.toFile(filePath, url.toString());
}

async existsUserByPhone(phone: string) {
  const user = await this.usersRepository.findOne({
    where: {
      phone,
    },
  });
  return !!user;
}

```

```
findAll() {
    return this.usersRepository.find({ loadEagerRelations: false });
}
```

```
async findById(id: number) {
    return this.usersRepository.findOneOrFail({ id });
}
```

```
async findByPhone(phone: string) {
    return this.usersRepository.findOne({ phone });
}
```

```
async findByEmail(email: string) {
    return this.usersRepository.findOne({ email });
}
```

```
async findByUsername(username: string) {
    return this.usersRepository.findOne({ username });
}
```

```
update(id: number, updateUserDto: UpdateUserDto) {
    return `This action updates a #${id} user`;
}
```

```
async remove(id: number) {
    const user = await this.usersRepository.findOne({
        where: {
            id,
        },
    });
    return await this.usersRepository.remove([user]);
}
```

```

async setUserAvatar(id: number, file: FileEntity): Promise<UserProfile> {
  const user = await this.usersRepository.findOne({
    where: {
      id,
    },
    relations: ['profile'],
    select: ['id', 'profile'],
  });
  const oldLogoFile = user.profile.avatar;
  user.profile.avatar = file;
  const updated = await this.userProfileRepository.save(user.profile);

  if (oldLogoFile) {
    // console.log(oldLogoFile);
    try {
      await this.filesService.removeFileById(oldLogoFile.id);
    } catch {}
  }

  return updated;
}

async onApplicationBootstrap() {
  const user = await this.findByUsername('admin');
  const envPassword = this.configService.get('ADMIN_PASSWORD');
  const adminPassFile = './admin-password.txt';
  if (!user) {
    const password = envPassword ?? nanoid(16);
    const adminUser = await this.create({
      name: 'Admin',
      username: 'admin',
      email: 'admin@email.com',
      password: await bcrypt.hash(password, 10),
    });
  }
}

```

```

});
const userRole = this.userRoleRepository.create({
  user: adminUser,
  role: 'admin',
});
await this.userRoleRepository.save(userRole);
if (!envPassword) fs.writeFileSync(adminPassFile, password);
console.log(
  adminUser.email +
    ':' +
    (envPassword ?? fs.readFileSync(adminPassFile).toString()),
);
} else {
console.log(
  user.email +
    ':' +
    (envPassword ?? fs.readFileSync(adminPassFile).toString()),
);
}
}
}

```

user.entity.ts – сутність користувача

```

import {
  Column,
  CreateDateColumn,
  Entity,
  Index,
  JoinColumn,
  OneToMany,
  OneToOne,
  PrimaryGeneratedColumn,
  UpdateDateColumn,

```

```

} from 'typeorm';
import { Exclude, Expose, Transform } from 'class-transformer';
import { ApiProperty } from '@nestjs/swagger';
import { UserProfile } from './user-profile.entity';
import { nanoid } from 'nanoid';
import { UserRole } from './user-role.entity';

```

```

@Entity()
export class User {
  @ApiProperty()
  @PrimaryGeneratedColumn()
  id: number;

  @ApiProperty()
  @Column({
    default: () => `${nanoid(12)}`,
  })
  tag!: string;

```

```

/* Identity */
@ApiProperty()
@Column({ nullable: true, unique: true })
username: string;
@Expose({ groups: ['admin'] })
@ApiProperty()
@Column({ nullable: true, unique: true })
email: string;

```

```

@Expose({ groups: ['admin'] })
@ApiProperty()
@Column({ default: false })
emailConfirmed: boolean;

```

```

@Expose({ groups: ['admin'] })
@ApiProperty()
@Column({ nullable: true, unique: true })
phone: string;

/* Name Surname Patronymic */
@ApiProperty()
@Column({ nullable: true })
name: string;
@ApiProperty()
@Column({ nullable: true })
surname: string;
@ApiProperty()
@Column({ nullable: true })
patronymic: string;

/* Profile */

@ApiProperty()
@OneToOne(() => UserProfile, { eager: true, onDelete: 'CASCADE' }) // specify inverse
side as a second parameter
@JoinColumn()
profile: UserProfile;

@ApiProperty()
@OneToMany(() => UserRole, (r) => r.user, { eager: true })
@Transform(({ value }) => value.map((r) => r.role))
roles: UserRole[];
// @ApiProperty()
// @OneToMany(() => Company, (company) => company.owner)
// @JoinColumn()
// companies: Company[];

```



```

/* Auth fields */
@ApiProperty()
@Column({ nullable: true })
@Exclude()
password: string;

@ApiProperty()
@Column({ nullable: true })
@Exclude()
studentGroupId: number;

@ApiProperty()
@Expose({ groups: ['admin'] })
@Column({ default: false })
disabled: boolean;

@Expose({ groups: ['admin'] })
@ApiProperty()
get hasPassword(): boolean {
    return this.password !== null;
}

@Expose({ groups: ['admin'] })
@ApiProperty()
@Column({ nullable: true })
google_id: string;
/*
    Meta - Facebook
*/
@Expose({ groups: ['admin'] })
@ApiProperty()
@Column({ nullable: true })
meta_id: string;

```

```

/* Timestamps */
@Expose({ groups: ['admin'] })
@ApiProperty()
@Index()
@CreateDateColumn({ type: 'timestamp' })
createdAt: Date;
@Expose({ groups: ['admin'] })
@ApiProperty()
@Index()
@UpdateDateColumn({ type: 'timestamp' })
updatedAt: Date;
}

user-profile.entity – сутність профілю
import {
Column,
CreateDateColumn,
Entity,
Index,
JoinColumn,
JoinTable,
ManyToMany,
OneToOne,
PrimaryGeneratedColumn,
UpdateDateColumn,
} from 'typeorm';
import { ApiProperty } from '@nestjs/swagger';
import { Exclude, Transform } from 'class-transformer';
import { formatPublicFileUrl } from '../files/utills';
import { FileEntity } from '../files/entities/file.entity';
import { User } from './user.entity';

@Entity()

```

```

export class UserProfile {
    @Exclude()
    @ApiProperty()
    @PrimaryGeneratedColumn()
    id: number;

    @OneToOne(() => User, (user) => user.profile)
    // @JoinColumn()
    user: User;

    @ApiProperty()
    @Column({ type: 'date', nullable: true })
    birthday: Date;

    @ApiProperty()
    @Column({ nullable: true })
    phone: string;

    @ApiProperty()
    @Transform(({ value }) => formatPublicFileUrl({ value: value?.url }))
    @OneToOne(() => FileEntity, {
        eager: true,
        onDelete: 'CASCADE',
    })
    @JoinColumn()
    avatar: FileEntity;
    /* Timestamps */
    @Exclude()
    @ApiProperty()
    @Index()
    @CreateDateColumn({ type: 'timestamp' })
    createdAt: Date;
    @Exclude()

```

```

@ApiProperty()
@Index()
@UpdateDateColumn({ type: 'timestamp' })
updatedAt: Date;
}

```

core.service.ts – сервіс запитів User Service до Core Service

```

import { Injectable, OnApplicationBootstrap } from '@nestjs/common';
import { HttpService } from '@nestjs/axios';
import { lastValueFrom, map, Observable } from 'rxjs';
import { AxiosResponse } from 'axios';

export function getAxiosData(t: Observable<AxiosResponse>) {
  return lastValueFrom(t.pipe(map((x) => x.data)));
}

```

```

@Injectable()
export class CoreService {
  constructor(private api: HttpService) {}

  getGroups(): Promise<any> {
    return getAxiosData(this.api.get('/students-groups'));
  }

  getPersons(): Promise<any> {
    return getAxiosData(this.api.get('/persons'));
  }

  getPersonsPositions(): Promise<any> {
    return getAxiosData(this.api.get('/persons-positions'));
  }

  getSubjects(): Promise<any> {

```

```

    return getAxiosData(this.api.get('/subjects'));
  }

  getLocations(): Promise<any> {
    return getAxiosData(this.api.get('/locations'));
  }

  getGroupById(studentGroupId: number) {
    return getAxiosData(this.api.get('/students-groups/' + studentGroupId));
  }
}

```

Dockerfile

FROM node:14

WORKDIR /src/app

COPY package\*.json ./

COPY yarn.lock ./

RUN yarn install --silent

COPY . .

RUN yarn run build

EXPOSE 3000

ENTRYPOINT ["node", "dist/main.js"]

chats.gateway.ts – чат сокети

```
import {
```

```
  OnGatewayConnection,
```

```
  OnGatewayDisconnect,
```

```

    WebSocketGateway,
    WebSocketServer,
  } from '@nestjs/websockets';
import { ChatsService } from './chats.service';
import { Server, Socket } from 'socket.io';
import { AuthService } from '../auth/auth.service';
import { UsersService } from '../users/users.service';
import { ChatMessage } from './entities/chat-message.entity';
import { instanceToPlain } from 'class-transformer';
import { User } from '../users/entities/user.entity';
import { CACHE_MANAGER, Inject, Logger } from '@nestjs/common';
import { Cache } from 'cache-manager';

export class SocketMetaData {
  user: User;
}

@WebSocketGateway({
  allowEIO3: true,
  cors: {
    origin: '*',
  },
})
export class ChatsGateway implements OnGatewayConnection, OnGatewayDisconnect {
  @WebSocketServer()
  server: Server;

  private readonly logger = new Logger(ChatsGateway.name);

  constructor(
    private readonly chatsService: ChatsService,
    private readonly authService: AuthService,
    private readonly usersService: UsersService,
  ) {}
}

```

```

@Inject(CACHE_MANAGER) private cacheManager: Cache,
) {}

```

```

async handleConnection(client: Socket, ...args: any[]) {
  const token = client.handshake.auth?.token;
  if (!token) {
    client.disconnect();
    return;
  }
  const payload = await this.authService.verify(token);
  const user = await this.userService.findById(payload.id);
  if (!user) {
    client.disconnect();
    return;
  }

  this.logger.log(`connected user ${user.id}`, 'handle-connection');
  client.data['user'] = user;
  client.join(`userspace:${user.id}`);
  this.cacheManager.set(`user-online:${user.id}`, true, { ttl: 0 });
}

```

```

handleDisconnect(client: Socket): any {
  if (!client.data?.user) return;
  if (this.server.of(`userspace:${client.data.user.id}`).sockets.size === 0) {
    this.cacheManager.set(`user-online:${client.data.user.id}`, false, {
      ttl: 0,
    });
  }
  this.logger.log(
    `disconnected user ${client.data.user.id}`,
    'handle-connection',
  );
}

```

```
}
```

```
async sendChatMessage(message: ChatMessage) {
  const members = await this.chatsService.getChatMembers(message.chat.id);
  for (const id of members) {
    this.server.to(`userspace:${id}`).emit(
      'ChatMessage',
      instanceToPlain({
        id: message.id,
        chat: {
          id: message.chat.id,
          updatedAt: message.chat.updatedAt,
        },
        sender: message.sender,
        text: message.text,
        createdAt: message.createdAt,
        updatedAt: message.updatedAt,
      }),
    );
  }
}
```

```
async sendUserBalanceChange(userId: number, value: number) {
  this.server.to(`userspace:${userId}`).emit('BalanceChange', { value });
}
```

```
roomChatId(id: number) {
  return `chat:${id}`;
}
```

```
async joinUsersSocketsToChatRoom(userIds: number[], chatIdRoom: number) {
  const sockets = await this.server.fetchSockets();
  sockets
```



```
.filter((s) => userIds.includes(s.data?.user.id))  
.forEach((s) => {  
  s.join(this.roomChatId(chatIdRoom));  
});  
}  
}
```