ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: **«РОЗРОБКА API ДЛЯ МОНІТОРИНГУ РОБОТИ УЧНІВ МОВОЮ JAVA»**

Виконав: студент 4 курсу, групи ПД-44

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Вихристюк О.В.

(прізвище та ініціали)

Керівник Дібрівний О.А.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Нормоконтроль

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність -121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____О.В. Негоденко

«____»__ _____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

ВИХРИСТЮКУ ОЛЕКСАНДРУ ВІКТОРОВИЧУ

1.Тема роботи: «Розробка API для моніторингу роботи учнів мовою Java»
Керівник роботи Дібрівний Олесь Андрійович, доцент кафедри, доктор філософії затверджені наказом вищого навчального закладу від — «12» березня 2021 року №65.

2. Строк подання студентом роботи 01.06.2021

3. Вхідні дані до роботи:

3.1. Середовище розробки IntelliJ IDEA 2021.1.1 (Ultimate edition)

3.2. Алгоритм дії API для електронного журналу

3.3. Інтерфейс API

3.4. Науково-технічна література, пов’язана з розробкою API

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз обов’язків студента і викладача

4.2. Аналіз та порівняння існуючих прототипів

4.3. Дослідження програмних засобів для розробки програмного забезпечення мовою Java

4.4. Розробити функціонал створеного API

6. Дата видачі завдання 19.04.2021

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково технічної літератури	19.04.21 – 22.04.21	
2	Аналіз існуючих прототипів	22.04.21 – 23.04.21	
3	Дослідження програмних засобів	23.04.21 – 01.05.21	
4	Моделювання об’єкту проектування	01.05.21 – 05.05.21	
5	Розробка функціоналу API	05.05.21 – 24.05.21	
6	Вступ, висновки, реферат	24.05.21 – 26.05.21	
7	Розробка презентації застосунку	26.05.21 – 31.05.21	
8	Попередній захист роботи	01.06.21	

Студент

Керівник роботи

РЕФЕРАТ

Текстова частина бакалаврської роботи 60с., 13 рис., 53 джерел.

Ключеві слова: IntelliJ IDEA, Java, Spring, Клієнт-серверна архітектура, API, електронний журнал, веб сайти, сайт.

Об'єкт дослідження – перенесення документообігу в електронний вигляд та автоматизація рутинних процесів.

Предмет дослідження – електронний журнал, для переходу від паперового журналу.

Мета роботи – розробка API для електронного журналу на мові програмування Java.

Методи дослідження – методи теорії інформації, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування, перевірки та верифікації програмного забезпечення.

В роботі виконано аналіз існуючих аналогів електронних журналів. Встановлено переваги та недоліки існуючих журналів. В результаті аналізу було визначено основні потреби користувачів. Проаналізовано можливості середовища розробки IntelliJ IDEA. Розроблено логіку практичних завдань та загальну концепцію представлення інформації для користувачів.

Галузь використання – журнал можуть використовувати як вчителі так і студенти які мають зв'язок з інтернетом.

ЗМІСТ

ВСТУП

В сучасному світі кожного дня кількість інформації, яку необхідно зберігати росте в геометричній прогресії , набуваючи величезних розмірів. В певний момент старі варіанти зберігання інформації, такі як паперові носії, або навіть усна форма, віджили своє і більше не могли задовольняти потреби сучасного соціуму.

Саме в цей момент і з'явилося систем автоматизації, які до сьогоднішнього дня допомагають людям у структуруванні та зберіганні інформації різного роду, взаємодії між ролями користувачів, тощо.

На сьогоднішній день кожне підприємство та установа мають свої внутрішні клієнти, які допомагають зберігати і структурувати інформацію про роботи даної установи, автоматизувати деякі процеси, тощо.

Виходячи з актуальності явища систем автоматизації, об'єктом дослідження є інформаційні системи і їх використання в різного роду структурах.

Предмет дослідження — серверна частина для інформаційної системи електронного журналу.

Мета роботи — розроблення серверної частини для інформаційної системи електронного журналу засобами мови програмування Java.

Для досягнення поставленої мети слід виконати наступні завдання:

Для виконання поставленої мети слід виконати наступні завдання:

- Провести огляд поняття серверу
- Провести аналіз клієнт-серверної
- Провести огляд поняття електронного журналу
- Обрати мову програмування
- Обрати середовище розробки
- Побудувати діаграму варіантів використання
- Побудувати діаграму класів
- Розробити сторінку входу
- Розробити сторінку реєстрації

- Розробити сторінку виставлення оцінок
- Розробити сторінку статистики успішності
- Розробити сторінку розкладу
- Розробити сторінку завдань
- Розробити сторінку профілю

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття серверу

Під час обчислень сервер - це частина апаратного чи програмного забезпечення комп'ютера (комп'ютерна програма), що забезпечує функціональність для інших програм або пристроїв, що називається «клієнтами». Ця архітектура називається модель клієнт-сервер. Сервери можуть надавати різні функціональні можливості, які часто називають "послугами", наприклад, спільний доступ до даних або ресурсів між кількома клієнтами, або виконання обчислень для клієнта. Один сервер може обслуговувати декілька клієнтів, а один клієнт може використовувати кілька серверів. Клієнтський процес може запускатися на одному пристрої або може підключатися через мережу до сервера на іншому пристрої. [1] Типовими серверами є сервери баз даних, файлові сервери, поштові сервери, сервери друку, веб-сервери, ігрові сервери та сервери додатків. [2]

Клієнт-серверні системи сьогодні найчастіше реалізуються за допомогою моделі запит-відповідь (і часто їх ідентифікують): клієнт надсилає запит серверу, який виконує певні дії та надсилає відповідь клієнту, як правило, з результатом або підтвердженням . Позначення комп'ютера як "апаратне забезпечення серверного класу" означає, що він спеціалізується на роботі серверів на ньому. Це часто означає, що він потужніший і надійніший, ніж стандартні персональні комп'ютери, але, як альтернатива, великі обчислювальні кластери можуть складатися з багатьох відносно простих, змінних серверних компонентів.

Використання слова-сервера в обчислювальній техніці походить від теорії масового обслуговування [3], де воно датується серединою 20-го століття і, зокрема, використовується в Кендалл (1953) (разом із "службою"), статті, яка представила позначення Кендалла. У попередніх роботах, таких як Erlang (1909), використовуються більш конкретні терміни, такі як "[телефонні оператори]".

Під час обчислень "сервер" датується щонайменше RFC 5 (1969) [4], одним з найдавніших документів, що описує ARPANET (попередник Інтернету), і протиставляється "користувачеві", розрізняючи два типи хостів: "сервер - хост "та" користувач-хост ". Використання "обслуговування" також відноситься до ранніх документів, таких як RFC 4, [5] протиставляючи "сервіс-хост" та "користувач-хост".

Строго кажучи, термін сервер позначає комп'ютерну програму або процес (запущену програму). За допомогою метонімії це означає пристрій, який використовується (або пристрій, призначений для) запуску однієї або кількох серверних програм. У мережі такий пристрій називається хостом. На додаток до сервера, слова serve і service (як іменник та дієслово) часто використовуються, хоча servicer і servant - ні. [A] Слово service (іменник) може позначати або абстрактну форму функціональності, наприклад Веб-сервіс. Як варіант, це може стосуватися комп’ютерної програми, яка перетворює комп’ютер на сервер, наприклад Служба Windows. Спочатку використовувався як "сервери обслуговують користувачів" (і "користувачі використовують сервери"), у значенні "підкорятися", сьогодні часто говорять, що "сервери обслуговують дані", в тому ж сенсі, що і "дають". Наприклад, веб-сервери "обслуговують веб-сторінки для користувачів" або "обслуговують їх запити".

Сервер є частиною моделі клієнт-сервер; у цій моделі сервер обслуговує дані для клієнтів. Природа спілкування між клієнтом та сервером - це запит та відповідь. Це на відміну від однорангової моделі, в якій відносини є взаємністю на вимогу. В принципі, будь-який комп'ютеризований процес, який може бути використаний або викликаний іншим процесом (особливо віддалено, зокрема для спільного використання ресурсу), є сервером, а процес або процеси, що викликаються, - клієнтом. Таким чином, будь-який комп'ютер загального призначення, підключений до мережі, може розміщувати сервери. Наприклад, якщо файли на пристрої спільно використовуються деяким процесом, цей процес є файловим сервером. Подібним чином, програмне забезпечення веб-сервера може працювати

на будь-якому здатному комп'ютері, і тому ноутбук або персональний комп'ютер може розміщувати веб-сервер.

Хоча запит-відповідь є найпоширенішим дизайном клієнт-сервер, існують і інші, такі як шаблон публікації – передплати. За шаблоном публікації-передплати клієнти реєструються на сервері pub-sub, підписуючись на певні типи повідомлень; ця первинна реєстрація може бути здійснена шляхом запиту-відповіді. Після цього сервер pub-sub пересилає відповідні повідомлення клієнтам без подальших запитів: сервер передає повідомлення клієнту, а не клієнт, який витягує повідомлення з сервера, як у запиті-відповіді. [6]

Вимоги до обладнання для серверів дуже різняться залежно від призначення сервера та його програмного забезпечення. Сервери частіше за все, потужніші та дорожчі, ніж клієнти, які до них підключаються.

Оскільки сервери, як правило, отримують доступ через мережу, багато хто працює без нагляду без монітора комп’ютера або пристрою введення, звукового обладнання та інтерфейсів USB. Багато серверів не мають графічного інтерфейсу користувача (GUI). Вони налаштовуються та управляються віддалено. Віддалене управління може здійснюватися за допомогою різних методів, включаючи консоль керування Microsoft (MMC), PowerShell, SSH та позасмугові системи управління на основі браузера, такі як iDRAC Dell або iLo HP.

Великі традиційні одиночні сервери повинні працювати без перерви протягом тривалого періоду. Доступність повинна бути дуже високою, що робить надійність та довговічність обладнання надзвичайно важливими. Критично важливі корпоративні сервери будуть дуже стійкими до несправностей і використовуватимуть спеціалізоване обладнання з низьким рівнем відмов, щоб максимізувати час безвідмовної роботи. Джерела безперебійного живлення можуть бути вбудовані для захисту від збою живлення. Сервери, як правило, включають апаратне резервування, таке як подвійні джерела живлення, дискові системи RAID та пам'ять ECC [10], а також велике тестування та перевірка пам'яті перед завантаженням. Критично важливі компоненти можуть бути замінені гарячою системою, що дозволяє технічним працівникам замінити їх на працюючому

сервері, не вимикаючи його, а також для захисту від перегріву, сервери можуть мати потужніші вентилятори або використовувати водяне охолодження. Їх часто можна буде налаштувати, включити і вимкнути або перезавантажити віддалено, використовуючи позасмугове управління, як правило, на основі IPMI. Серверні кожухи, як правило, плоскі та широкі, і призначені для монтажу в стійку або на 19-дюймових стійках, або на відкритих стійках.

Такі типи серверів часто розміщуються у спеціальних центрах обробки даних. Зазвичай вони матимуть дуже стабільне живлення та Інтернет та підвищений рівень безпеки. Шум також не викликає занепокоєння, але споживання енергії та віддача тепла можуть бути серйозною проблемою. Серверні кімнати обладнані кондиціонерами.

В Інтернеті домінуючими операційними системами серед серверів є UNIX-подібні дистрибутиви з відкритим кодом, такі як ті, що базуються на Linux та FreeBSD [15], причому Windows Server також має значну частку. Запатентовані операційні системи, такі як z / OS та macOS Server, також розгортаються, але у значно меншій кількості.

Спеціалізовані серверно-орієнтовані операційні системи традиційно мають такі функції, як:

- GUI недоступний або необов'язковий
- Можливість переналаштувати та оновити як апаратне, так і програмне забезпечення до певної міри без перезапуску
- Розширені засоби резервного копіювання, що дозволяють регулярні та часті резервні копії важливих даних в Інтернеті,
- Прозора передача даних між різними томами або пристроями
- Гнучкі та розширені можливості роботи в мережі
- Можливості автоматизації, такі як демони в UNIX та служби в Windows
- Надійна безпека системи, із вдосконаленим захистом користувачів, ресурсів, даних та пам'яті.
- Розширене виявлення та попередження про такі умови, як перегрів, несправність процесора та диска. [16]

На практиці на сьогоднішній день багато настільних та серверних операційних систем мають подібні основи коду, що відрізняються переважно конфігурацією.

1.2. Клієнт-серверна архітектура

Клієнт-серверна модель - це розподілена структура додатків, яка розділяє завдання або робочі навантаження між провайдерами ресурсу чи послуги, які називаються серверами, та запитувачами послуг, які називаються клієнтами. [1] Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому обладнанні, але і клієнт, і сервер можуть перебувати в одній системі. Хост сервера запускає одну або кілька серверних програм, які діляться своїми ресурсами з клієнтами. Клієнт, як правило, не ділиться жодним із своїх ресурсів, але він запитує вміст або послуги у сервера. Отже, клієнти ініціюють сеанси зв'язку із серверами, які очікують на вхідні запити. Прикладами комп'ютерних програм, що використовують модель клієнт-сервер, є електронна пошта, мережевий друк та Всесвітня павутина.

Характеристика "клієнт-сервер" описує взаємозв'язок взаємодіючих програм у програмі. Серверний компонент надає функцію або послугу одному або багатьом клієнтам, які ініціюють запити на такі послуги. Сервери класифікуються за послугами, які вони надають. Наприклад, веб-сервер обслуговує веб-сторінки, а файловий - комп'ютерні. Спільним ресурсом може бути будь-яке програмне забезпечення серверного комп'ютера та електронні компоненти, починаючи від програм та даних, закінчуючи процесорами та запам'ятовуючими пристроями. Спільне використання ресурсів сервера є послугою.

Чи комп'ютер є клієнтом, сервером чи обома, визначається характером програми, яка вимагає сервісних функцій. Наприклад, на одному комп'ютері може одночасно працювати веб-сервер та програмне забезпечення файлового сервера, щоб обслуговувати різні дані клієнтам, які роблять різні типи запитів. Клієнтське

програмне забезпечення може також спілкуватися із серверним програмним забезпеченням на тому самому комп'ютері. [2] Зв'язок між серверами, наприклад для синхронізації даних, іноді називають міжсерверним або міжсерверним.

Загалом, послуга - це абстракція комп'ютерних ресурсів, і клієнт не повинен займатися тим, як працює сервер під час виконання запиту та надання відповіді. Клієнт повинен розуміти відповідь лише на основі відомого протоколу програми, тобто вмісту та форматування даних для запитуваної послуги.

Клієнти та сервери обмінюються повідомленнями за шаблоном обміну повідомленнями запит-відповідь. Клієнт надсилає запит, а сервер повертає відповідь. Цей обмін повідомленнями є прикладом міжпроцесорного спілкування. Для спілкування комп’ютери повинні мати спільну мову, і вони повинні дотримуватися правил, щоб і клієнт, і сервер знали, чого очікувати. Мова та правила спілкування визначені протоколом зв'язку. Всі протоколи працюють на рівні програми. Протокол рівня додатків визначає основні схеми діалогу. Щоб ще більше формалізувати обмін даними, сервер може реалізувати інтерфейс прикладного програмування (API). [3] API - це рівень абстракції для доступу до послуги. Обмежуючи спілкування певним форматом вмісту, це полегшує синтаксичний аналіз. Абстрагуючи доступ, це полегшує міжплатформенний обмін даними. [4]

Сервер може отримувати запити від багатьох різних клієнтів за короткий проміжок часу. Комп’ютер може виконувати обмежену кількість завдань у будь-який момент і покладається на систему планування, щоб визначити пріоритет вхідних запитів клієнтів для їх задоволення. Щоб запобігти зловживанням та максимізувати доступність, серверне програмне забезпечення може обмежити доступність для клієнтів. Атаки відмови в обслуговуванні призначені для використання зобов'язання сервера обробляти запити, перевантажуючи його надмірними частотами запитів. Шифрування слід застосовувати, якщо конфіденційна інформація повинна передаватися між клієнтом та сервером.

Ранньою формою архітектури клієнт-сервер є віддалений запис роботи, принаймні з OS / 360 (оголошено 1964 р.), Де запит повинен був виконати роботу, а відповіддю був вихідний результат.

Формулюючи модель клієнт-сервер в 1960-х і 1970-х роках, вчені-розробники ARPANET (в Стенфордському науково-дослідному інституті) використовували терміни сервер-хост (або сервісний хост) та користувач-хост (або використовуючи-хост), і вони з'являються в ранні документи RFC 5 [5] та RFC 4. [6] Це використання було продовжено в Xerox PARC у середині 1970-х.

Одним із контекстів, в якому дослідники використовували ці терміни, було проектування мови програмування комп'ютерної мережі, яка називається Мова декодування-кодування (DEL). [5] Метою цієї мови було приймати команди від одного комп'ютера (хост-користувач), який повертав би користувачеві звіти про стан, коли він кодував команди в мережевих пакетах. Інший комп'ютер, який підтримує DEL, сервер-хост, приймав пакети, декодував їх і повертав відформатовані дані до хосту користувача. Програма DEL на хості користувача отримала результати для представлення користувачеві. Це транзакція клієнт-сервер. Розробка DEL тільки починалася в 1969 році, в рік, коли Міністерство оборони США заснувало ARPANET (попередник Інтернету).

Клієнт-хост і сервер-хост мають незначно різні значення, ніж клієнт і сервер. Хост - це будь-який комп'ютер, підключений до мережі. Тоді як слова сервер та клієнт можуть позначати або комп'ютер, або комп'ютерну програму, сервер-хост та користувач-хост завжди стосуються комп'ютерів. Хост - це універсальний багатофункціональний комп'ютер; клієнти та сервери - це лише програми, що працюють на хості. У моделі клієнт-сервер сервер, швидше за все, буде присвячений завданням обслуговування.

Раннє використання слова клієнт трапляється в "Відділення даних від функцій у розподіленій файловій системі", статті 1978 року комп'ютерних вчених Xerox PARC Говарда Стерджиса, Джеймса Мітчелла та Джея Ізраїля. Автори ретельно визначають термін для читачів та пояснюють, що використовують його

для розрізнення між користувачем та мережевим вузлом користувача (клієнтом). [7] (До 1992 року слово-сервер увійшло в загальну мову). [8] [9]

Модель клієнт-сервер не диктує, що сервери-хости повинні мати більше ресурсів, ніж клієнт-хости. Швидше, це дозволяє будь-якому комп'ютеру загального призначення розширити свої можливості за допомогою спільних ресурсів інших хостів. Однак централізовані обчислення виділяють велику кількість ресурсів на невелику кількість комп'ютерів. Чим більше обчислень завантажується з клієнт-хостів на центральні комп'ютери, тим простішими можуть бути клієнт-хости. [10] Для обчислень та зберігання він значною мірою покладається на мережеві ресурси (сервери та інфраструктуру). Бездисковий вузол завантажує з мережі навіть свою операційну систему, а комп'ютерний термінал взагалі не має операційної системи; це лише інтерфейс вводу / виводу для сервера. На відміну від цього, товстий клієнт, такий як персональний комп'ютер, має багато ресурсів і не покладається на сервер для виконання основних функцій.

У міру зниження ціни та збільшення потужності мікрокомп'ютерів з 1980-х до кінця 1990-х років багато організацій перевели обчислення від централізованих серверів, таких як мейнфрейми та міні-комп'ютери, до жирних клієнтів [11]. Це дало більше, більш індивідуальне панування над комп'ютерними ресурсами, але складне управління інформаційними технологіями. [10] [12] [13] Протягом 2000-х років веб-додатки визріли настільки, щоб конкурувати з прикладним програмним забезпеченням, розробленим для конкретної мікроархітектури. Це дозрівання, більш доступне масове сховище та поява сервісно-орієнтованої архітектури були одними з факторів, що породили тенденцію хмарних обчислень у 2010-х роках.

На додаток до моделі клієнт-сервер, розподілені обчислювальні програми часто використовують архітектуру додатків однорангової мережі (P2P).

У моделі клієнт-сервер сервер часто призначений для роботи як централізована система, яка обслуговує багатьох клієнтів. Витрати на обчислювальну потужність, пам'ять та пам'ять сервера повинні бути відповідно масштабовані відповідно до очікуваного навантаження. Системи балансування

навантаження та системи відмови часто використовуються для масштабування сервера за межі однієї фізичної машини. [15] [16]

Балансування навантаження визначається як методичний та ефективний розподіл мережевого або додаткового трафіку на декількох серверах у фермі серверів. Кожен балансир навантаження знаходиться між клієнтськими пристроями та серверними серверами, отримуючи та розподіляючи вхідні запити на будь-який доступний сервер, здатний їх виконати.

У одноранговій мережі два або більше комп'ютери (однорангові мережі) об'єднують свої ресурси та спілкуються в децентралізованій системі. Піри - це рівноправні або еквіпотентні вузли в неієрархічній мережі. На відміну від клієнтів у мережі клієнт-сервер або клієнт – черга – клієнт, однорангові комунікації здійснюють безпосередню взаємодію між собою. У однорангових мережах алгоритм у протоколі однорангової комунікації врівноважує навантаження і навіть однолітки зі скромними ресурсами можуть допомогти розподілити навантаження Якщо вузол стає недоступним, його спільні ресурси залишаються доступними до тих пір, поки його пропонують інші партнери. В ідеалі, рівноправній особі не потрібно досягати високої доступності, оскільки інші надлишкові однорангові компенсують час простою будь-якого ресурсу; оскільки доступність та завантажуваність однолітків змінюються, протокол перенаправляє запити.

І клієнт-сервер, і ведучий-підлеглий розглядаються як підкатегорії розподілених однорангових систем. [17]

1.3. Електронний журнал

Електронна книга оцінок - це онлайн-запис учителя про уроки, завдання, успішність та оцінки своїх учнів. Електронна книга оцінок взаємодіє з інформаційною системою учнів, в якій зберігаються записи учнів шкільного округу, включаючи оцінки, медичні картки відвідуваності, стенограми, графіки учнів та інші дані. Деякі книги з електронними оцінками роблять оцінки, домашні завдання та графіки учнів доступними в Інтернеті для батьків та учнів.

Деякі приклади послуг, що надають підручники, - TAMO, Gradealyzer, Spiral Universe, QuickSchools.com та GPA Teacher. Все це забезпечує простий спосіб оновлення інформації про оцінки кожного студента, а також можливість швидкого та легкого перенесення цієї інформації до звітних карток, що закінчуються.

У 2010 році британське урядове агентство з питань ІКТ в освіті, BECTA, запровадило вимогу до звітних карток для всіх учнів загальноосвітньої системи, щоб їхні звіти були доступними для батьків через Інтернет.

1.4. Огляд існуючих рішень

На даний момент одним з найпопулярніших аналогів розроблюваного продукту в Україні є онлайн-сервіс e-schools.info, який представляє собою електронний журнал, доступний до підключення будь-якої школи.

Детальна інформація про проект знаходиться на сторінці «Про проект» (рис. 1.1)

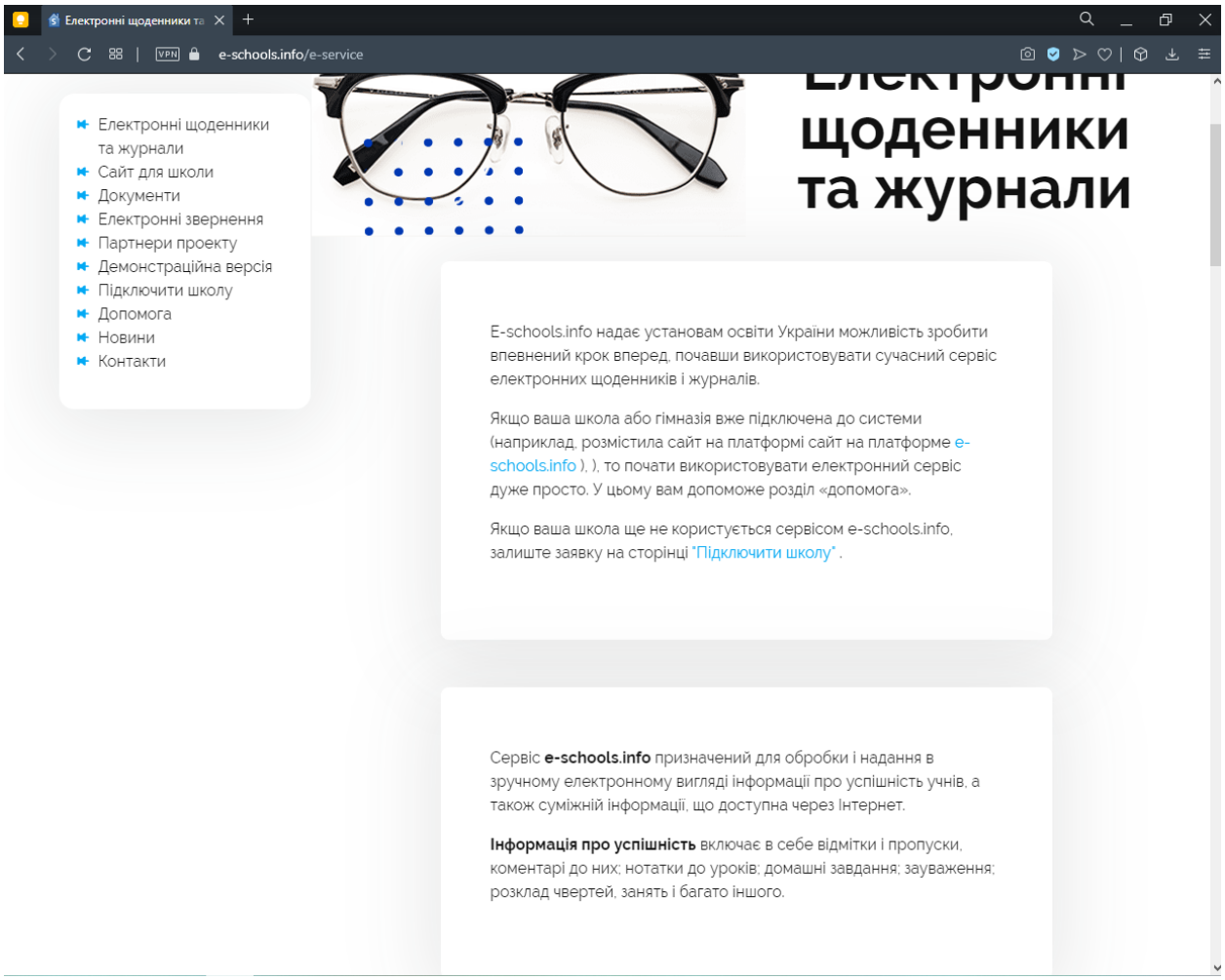


Рисунок 1.1 — Сторінка «Про проект»

Проект має демонстраційну версію, як для батьків, так і для шкіл, яка дає можливість протестувати додаток як зі сторони батьків (рис. 1.3) так і зі сторони школи (рис. 1.4). Перейти до демо-версій можна з головної сторінки проекту (рис. 1.2)

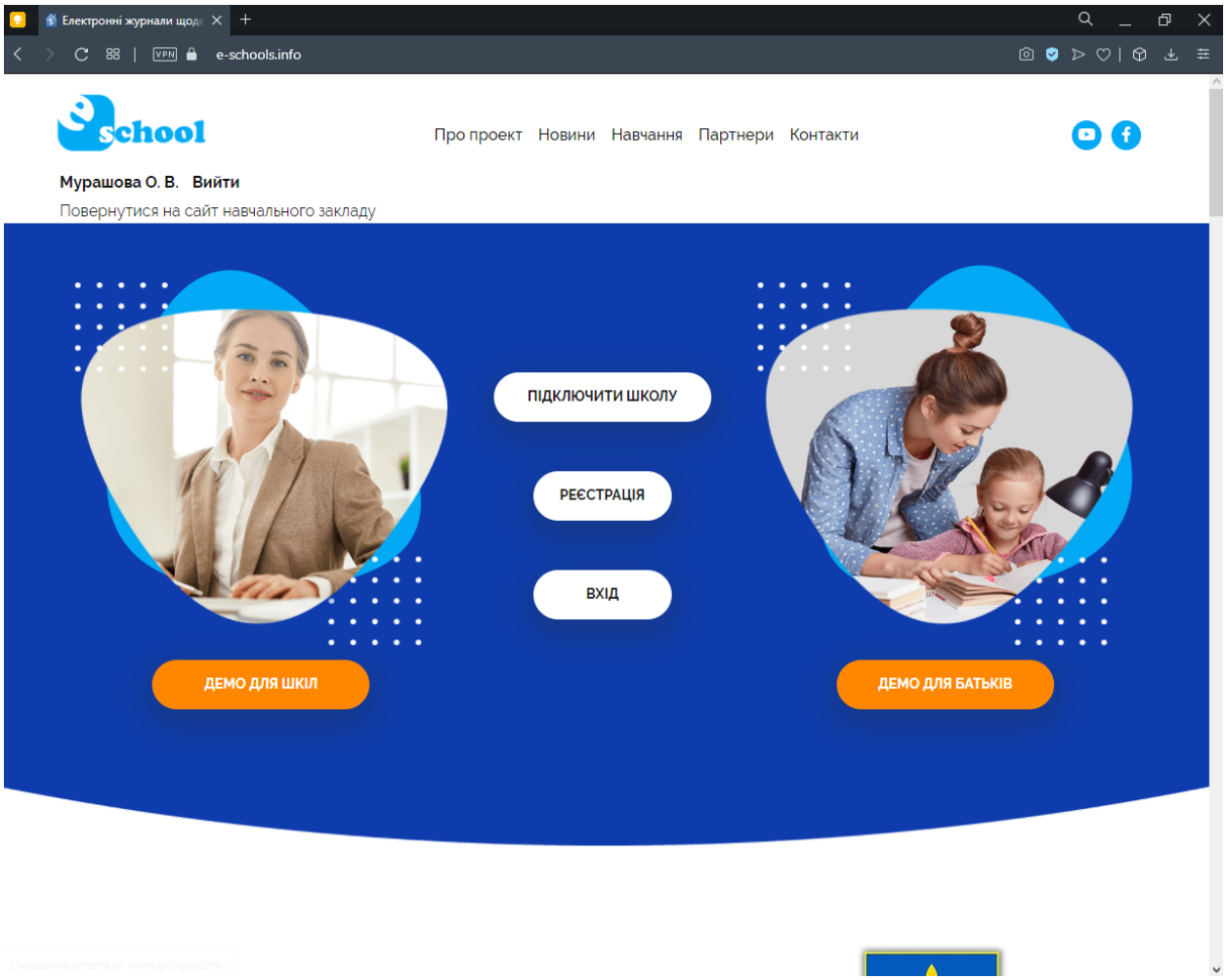


Рисунок 1.2 — Головна сторінка

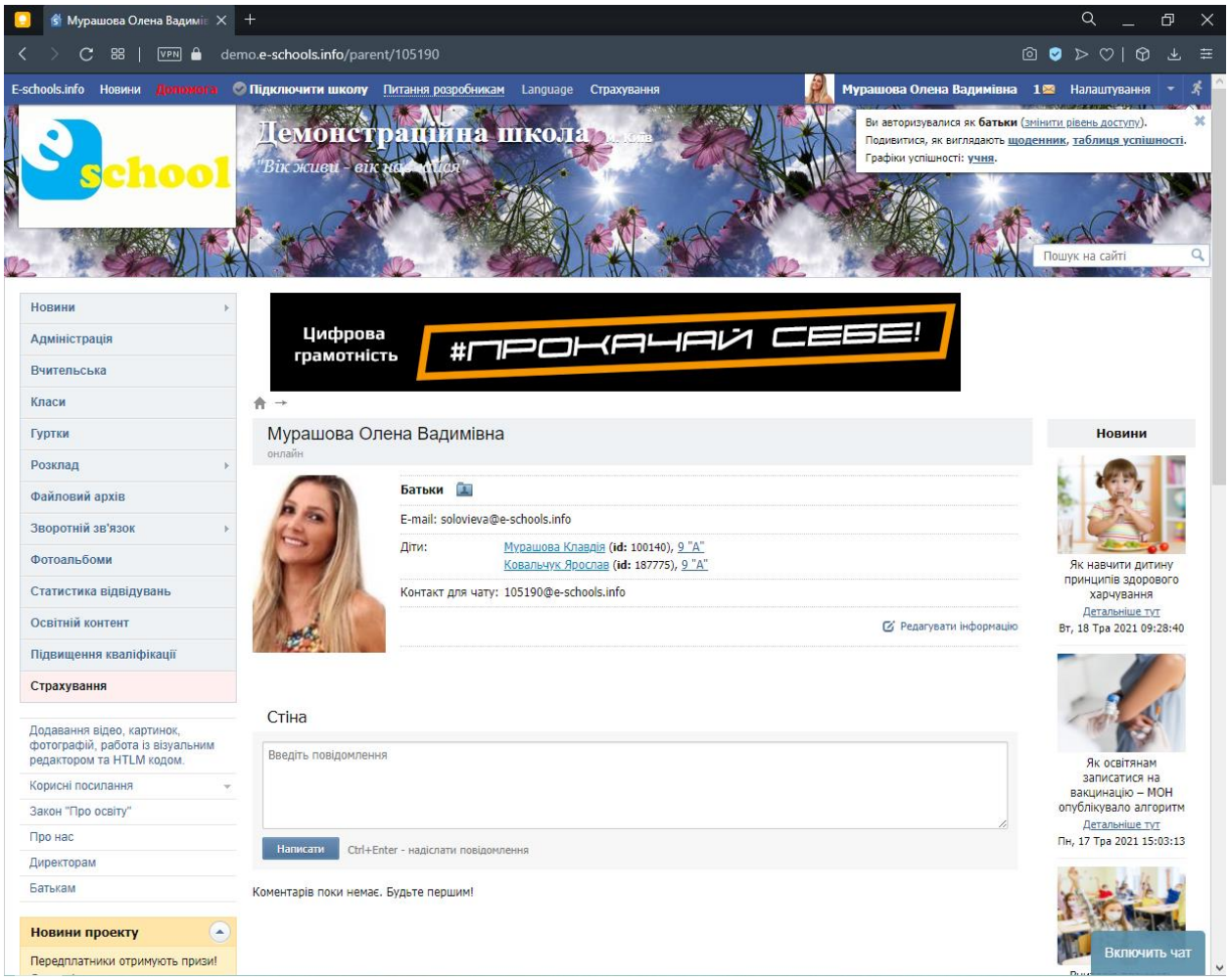


Рисунок 1.3 — Демо-сторінка для батьків

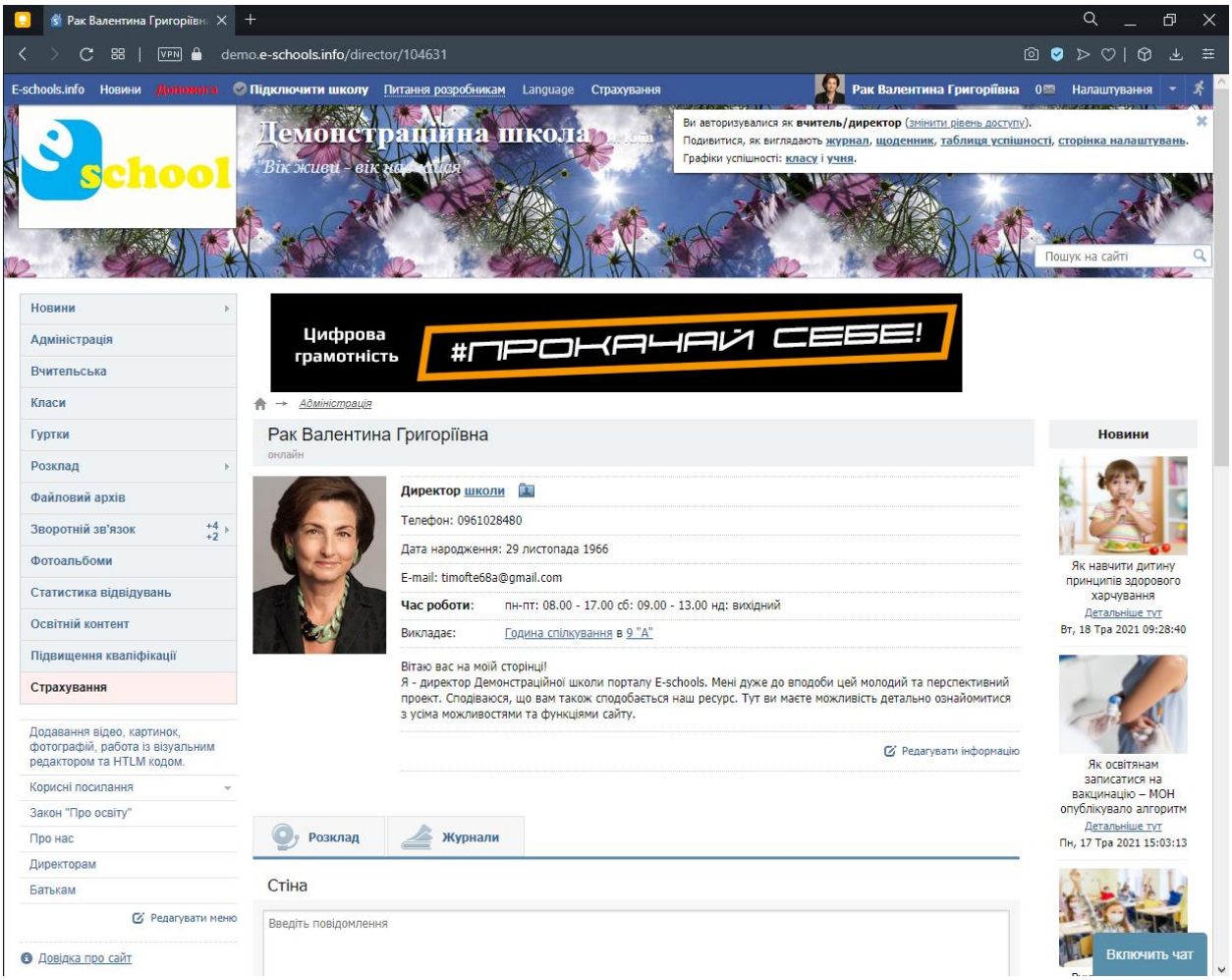


Рисунок 1.4 — Демо-сторінка для батьків

Дане рішення пропонує достойний функціонал, проте має в собі ряд недоліків, які важко не помітити:

- Велика затримка перед завантаженням сторінки
- Велика затримка при завантаженні сторінки
- Нагромадження візуальних блоків
- Інтуїтивна незрозумілість інтерфейсу
- Невдалий і застарілий дизайн

1.5. Постановка задачі

Основна задача даного проекту полягає в розробленні серверної частини (бекенду) для інформаційної системи електронного журналу.

Для повноцінного функціонування системи необхідно реалізувати наступні функції:

- Механізм реєстрації
- Механізм авторизації
- Механізм показу і зміни розкладу
- Механізм підрахунку статистки успішності учнів
- Механіку виставлення оцінок
- Алгоритм роботи з завданнями
- Реалізацію профіля користувача
- Механіку взаємодії з БД
- Механізм обробки запитів від клієнтської частини
- CRUD-механіки для БД

При реалізації описаних вище функцій задача на розробку вважається виконаною в повній мірі.

РОЗДІЛ 2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Java

Java - це об'єктно-орієнтована мова програмування, що базується на класах, і розроблена так, щоб мати якомога менше залежностей від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники програм могли писати один раз, запускати їх де завгодно (WORA) [16], що означає, що скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. [17] Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера. Синтаксис Java подібний до C та C ++, але має менше засобів низького рівня, ніж будь-який з них. Час виконання Java забезпечує динамічні можливості (такі як відображення та модифікація коду середовища виконання), які, як правило, недоступні в традиційних компільованих мовах. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, що використовується згідно з GitHub, [18] [19], особливо для веб-додатків клієнт-сервер, із 9 мільйонами розробників, про які повідомляється [20].

Спочатку Java була розроблена Джеймсом Гослінгом у Sun Microsystems (яку з тих пір придбала Oracle) і випущена в 1995 році як основний компонент Java-платформи Sun Microsystems. Оригінальні та довідкові реалізатори Java-компіляторів, віртуальних машин та бібліотек класів були спочатку випущені Sun під власні ліцензії. Станом на травень 2007 року, згідно з вимогами Процесу спільноти Java, Sun здійснила ліцензію на більшість своїх технологій Java під загальною публічною ліцензією GNU. Oracle пропонує свою власну віртуальну машину HotSpot Java, однак офіційним посиланням є OpenJDK JVM, яке є безкоштовним програмним забезпеченням з відкритим кодом і використовується більшістю розробників і є JVM за замовчуванням майже для всіх дистрибутивів Linux.

Станом на березень 2021 року останньою версією є Java 16, з Java 11, яка наразі підтримується довгостроковою підтримкою (LTS), випущена 25 вересня 2018 р. Oracle випустила останнє публічне оновлення із застарілою версією Java 8 LTS у січні 2019 року для комерційного використання, хоча в іншому випадку він все ще підтримуватиме Java 8 із загальнодоступними оновленнями для особистого користування на невизначений час. Інші постачальники почали пропонувати збірки OpenJDK 8 і 11 з низькою вартістю, які все ще отримують покращення безпеки та інші оновлення.

Oracle (та інші) настійно рекомендують видалити застарілі версії Java через серйозні ризики через невирішені проблеми безпеки. [21] Оскільки Java 9, 10, 12, 13, 14 і 15 більше не підтримуються, Oracle радить своїм користувачам негайно перейти на останню версію (на даний момент Java 16) або випуск LTS.

Джеймс Гослінг, Майк Шерідан та Патрік Нотон ініціювали мовний проект Java у червні 1991 р. [22] Спочатку Java була розроблена для інтерактивного телебачення, але на той час вона була надто розвиненою для галузі цифрового кабельного телебачення [23]. Спочатку мову називали Дуб на честь дуба, що стояв біля кабінету Гослінга. Пізніше проект пішов під назвою Зелений і, нарешті, був перейменований на Java, з кави Java, різновиду кави з Індонезії [24]. Гослінг розробив Java із синтаксисом стилю C / C ++, який системні та прикладні програмісти знайдуть добре. [25]

Sun Microsystems випустила першу публічну реалізацію як Java 1.0 у 1996 році. [26] Він обіцяв функцію «Написати один раз, запустити де завгодно» (WORA), забезпечуючи безкоштовний час роботи на популярних платформах. Досить безпечний і з можливістю налаштування безпеки, він дозволяв обмеження доступу до мережі та файлів. Основні веб-браузери незабаром включили можливість запуску аплетів Java на веб-сторінках, і Java швидко стала популярною. Компілятор Java 1.0 був переписаний на Java Артур ван Гоффом, щоб суворо відповідати специфікації мови Java 1.0. [27] З появою Java 2 (випущений спочатку як J2SE 1.2 у грудні 1998 - 1999), нові версії мали кілька конфігурацій, побудованих для різних типів платформ. J2EE включає технології та API для корпоративних

програм, як правило, працюють у серверних середовищах, тоді як J2ME пропонує API, оптимізовані для мобільних додатків. Настільна версія була перейменована в J2SE. У 2006 році для маркетингових цілей Sun перейменував нові версії J2 на Java EE, Java ME та Java SE відповідно.

У 1997 році Sun Microsystems звернувся до органу зі стандартів ISO / IEC JTC 1, а згодом і до Міжнародної організації екзаменаційних технологій (Ecma International), щоб офіційно оформити Java, але незабаром він відмовився від цього процесу [28] [29] [30]. Java залишається фактичним стандартом, який контролюється через процес спільноти Java. [31] Свого часу Sun зробив більшість своїх реалізацій Java доступними безкоштовно, незважаючи на статус власного програмного забезпечення. Sun приносив дохід від Java за рахунок продажу ліцензій на спеціалізовані продукти, такі як Java Enterprise System.

13 листопада 2006 року Sun випустила більшу частину своєї віртуальної машини Java (JVM) як вільне програмне забезпечення з відкритим кодом (FOSS) на умовах Загальної публічної ліцензії GNU (GPL). 8 травня 2007 року Sun завершив процес, зробивши весь основний код свого JVM доступним за умовами вільного програмного забезпечення / розповсюдження з відкритим кодом, окрім невеликої частини коду, на яку Sun не належав авторських прав. [32]

Віце-президент Sun, Річ Грін, сказав, що ідеальна роль Sun у відношенні Java - це євангеліст [33]. Після придбання корпорацією Oracle корпорації Sun Microsystems у 2009–10 рр. Oracle назвала себе розпорядником технологій Java, що невпинно прагне сприяти розвитку спільноти участі та прозорості [34]. Це не завадило Oracle незабаром подати позов проти Google за використання Java всередині Android SDK (див. Розділ Android).

2 квітня 2010 року Джеймс Гослінг звільнився з Oracle [35].

У січні 2016 року Oracle оголосив, що середовища виконання Java, засновані на JDK 9, припинять роботу плагіна браузера. [36]

Програмне забезпечення Java працює на всьому: від ноутбуків до центрів обробки даних, ігрових консолей до наукових суперкомп'ютерів. [37]

Однією з цілей дизайну Java є портативність, що означає, що програми, написані для платформи Java, повинні працювати аналогічно на будь-якій комбінації обладнання та операційної системи з адекватною підтримкою часу роботи. Це досягається компіляцією коду мови Java у проміжне подання, що називається байт-кодом Java, а не безпосередньо в машинний код, специфічний для архітектури. Інструкції байт-коду Java аналогічні машинному коду, але вони призначені для виконання віртуальною машиною (ВМ), написаною спеціально для апаратного забезпечення хоста. Кінцеві користувачі зазвичай використовують середовище виконання Java (JRE), встановлене на їх машині для окремих програм Java або у веб-браузері для аплетів Java.

Стандартні бібліотеки забезпечують загальний спосіб доступу до особливостей хоста, таких як графіка, створення потоків та створення мереж.

Використання універсального байт-коду спрощує перенесення. Однак накладні витрати на інтерпретацію байт-коду в машинних інструкціях робили інтерпретовані програми майже завжди більш повільними, ніж власні виконувані файли. Компілятори JIT, які компілюють байт-коди до машинного коду під час виконання, були введені з ранньої стадії. Сама Java не залежить від платформи і пристосована до конкретної платформи, на якій вона повинна працювати віртуальною машиною Java (JVM) для неї, яка переводить байт-код Java на машинну мову платформи. [46]

Програми, написані на Java, мають репутацію повільніших і вимагають більше пам'яті, ніж програми, написані на C ++. [47] [48] Однак швидкість виконання програм Java значно покращилася завдяки введенню вчасно зведеної компіляції в 1997/1998 для Java 1.1 [49], додавання функцій мови, що підтримують кращий аналіз коду (наприклад, внутрішні класи, клас StringBuilder, необов'язкові твердження тощо), та оптимізації у віртуальній машині Java, такі як HotSpot, яка стала стандартною JVM Sun у 2000 році. У Java 1.5 продуктивність була покращена додаванням пакета java.util.concurrent, включаючи реалізації ConcurrentMaps та інших багатоядерних колекцій, що не містять блокування, і додатково покращена за допомогою Java 1.6.

Деякі платформи пропонують пряму апаратну підтримку Java; є мікроконтролери, які можуть запускати байт-код Java в апаратному забезпеченні, а не на програмній віртуальній машині Java [50], а деякі процесори на базі ARM можуть мати апаратну підтримку для виконання байт-коду Java за допомогою опції Jazelle, хоча підтримка здебільшого відмовлена в поточних реалізаціях APM.

Java використовує автоматичний збирач сміття для управління пам’яттю в життєвому циклі об’єкта. Програміст визначає, коли створюються об'єкти, а час виконання Java відповідає за відновлення пам'яті, коли об'єкти більше не використовуються. Як тільки посилання на об'єкт не залишаються, недоступна пам'ять стає придатною для автоматичного звільнення збирачем сміття. Щось подібне до витоку пам’яті все одно може статися, якщо код програміста містить посилання на об’єкт, який більше не потрібен, як правило, коли об’єкти, які більше не потрібні, зберігаються в контейнерах, які все ще використовуються. Якщо викликаються методи для неіснуючого об'єкта, викидається нульовий виняток покажчика. [51] [52]

Однією з ідей, що стоять за моделлю автоматичного управління пам'яттю Java, є те, що програмісти можуть бути позбавлені тягаря необхідності виконувати ручне управління пам’яттю. У деяких мовах пам’ять для створення об’єктів неявно виділяється в стеку або явно виділяється та звільняється з купи. В останньому випадку відповідальність за управління пам'яттю покладається на програміста. Якщо програма не вивільняє об'єкт, відбувається витік пам'яті. Якщо програма намагається отримати доступ або звільнити пам'ять, яка вже була вивільнена, результат невизначений і важко передбачуваний, і програма, ймовірно, стане нестабільною або вийде з ладу. Це можна частково виправити за допомогою розумних покажчиків, але це додає додаткових витрат та складності. Зверніть увагу, що збирання сміття не запобігає витоку логічної пам'яті, тобто тих, де пам’ять все ще посилається, але ніколи не використовується.

Вивіз сміття може відбуватися в будь-який час. В ідеалі це відбувається, коли програма не працює. Це гарантовано спрацьовує, якщо в купі недостатньо вільної

пам’яті для виділення нового об’єкта; це може призвести до того, що програма на мить зупиниться. Явне управління пам'яттю неможливе в Java.

Java не підтримує арифметику покажчика стилю C / C ++, де адресами об'єктів можна арифметично маніпулювати (наприклад, додаванням або відніманням зміщення). Це дозволяє збиральнику сміття переміщувати об'єкти, на які посилаються, і забезпечує безпеку та захист типу.

Як і в C ++ та деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java зберігаються або безпосередньо в полях (для об'єктів), або в стеку (для методів), а не в купі, як це зазвичай стосується непримітивних даних типи (але див. аналіз втечі). Це було свідоме рішення дизайнерів Java з міркувань продуктивності.

Java містить кілька типів збирачів сміття. Починаючи з Java 9, HotSpot використовує Garbage First Garbage Collector (G1GC) за замовчуванням. [53] Однак є також кілька інших збирачів сміття, які можна використовувати для управління купою. Для більшості програм на Java достатньо G1GC. Раніше в Java 8 використовувався паралельний збирач сміття.

Вирішивши проблему управління пам’яттю, не звільняє програміста від тягаря належної обробки інших видів ресурсів, таких як підключення до мережі або бази даних, дескриптори файлів тощо, особливо за наявності винятків.

На синтаксис Java значною мірою впливають C ++ та C. На відміну від C ++, який поєднує синтаксис для структурованого, загального та об’єктно-орієнтованого програмування, Java будувалася майже виключно як об’єктно-орієнтована мова. [17] Весь код пишеться всередині класів, і кожен елемент даних є об'єктом, за винятком примітивних типів даних (тобто цілих чисел, чисел з плаваючою комою, булевих значень та символів), які не є об'єктами з міркувань продуктивності. Java повторно використовує деякі популярні аспекти C ++ (наприклад, метод printf).

На відміну від C ++, Java не підтримує перевантаження оператора [54] або множинне успадкування для класів, хоча для інтерфейсів підтримується багаторазове успадкування. [55]

Java використовує коментарі, подібні до коментарів на C ++. Існує три різних стилі коментарів: стиль одного рядка, позначений двома косими рисками (//), стиль декількох рядків, відкритий / * і закритий * /, і стиль коментування Javadoc відкритий / ** і закритий * / . Стиль коментування Javadoc дозволяє користувачеві запускати виконуваний файл Javadoc для створення документації для програми і може бути прочитаний деякими інтегрованими середовищами розробки (IDE), такими як Eclipse, щоб дозволити розробникам отримати доступ до документації в IDE.

Синтаксис

Усі вихідні файли мають бути названі за загальнодоступним класом, який вони містять, додаючи суфікс .java, наприклад, HelloWorldApp.java. Спочатку його потрібно скомпілювати в байт-код, використовуючи компілятор Java, створюючи файл із суфіксом .class (у цьому випадку HelloWorldApp.class). Тільки тоді його можна виконати або запустити. Вихідний файл Java може містити лише один загальнодоступний клас, але він може містити кілька класів із модифікатором непублічного доступу та будь-яку кількість загальнодоступних внутрішніх класів. Коли вихідний файл містить декілька класів, необхідно зробити один клас (введений ключовим словом класу) загальнодоступним (перед ним - загальнодоступне ключове слово) і назвати вихідний файл цим загальнодоступним іменем класу.

Клас, який не оголошено загальнодоступним, може зберігатися у будь-якому файлі .java. Компілятор створить файл класу для кожного класу, визначеного у вихідному файлі. Ім'я файлу класу - це ім'я класу з доданим .class. Для генерації файлів класів анонімні класи обробляються так, ніби їх ім'я є об'єднанням імені їхнього класу, що містить, $ та ціле число.

Ключове слово public означає, що метод може бути викликаний з коду в інших класах, або що клас може використовуватися класами поза ієрархією класів. Ієрархія класів пов’язана з іменем каталогу, в якому знаходиться файл .java. Це називається модифікатором рівня доступу. Інші модифікатори рівня доступу включають ключові слова private (метод, до якого можна отримати доступ лише в

тому ж класі) та захищений (що дозволяє отримати доступ до коду з того самого пакету). Якщо фрагмент коду намагається отримати доступ до приватних методів або захищених методів, JVM видасть виняток SecurityException

Ключове слово static [18] перед методом вказує на статичний метод, який пов'язаний лише з класом, а не з будь-яким конкретним екземпляром цього класу. Тільки статичні методи можна викликати без посилання на об'єкт. Статичні методи не можуть отримати доступ до будь-яких членів класу, які також не є статичними. Методи, які не позначаються статичними, є методами екземпляра та потребують роботи певного екземпляра класу.

Ключове слово void означає, що основний метод не повертає жодного значення абоненту. Якщо програма Java повинна вийти з кодом помилки, вона повинна явно викликати System.exit ().

Назва методу main не є ключовим словом у мові Java. Це просто назва методу, який запускає програма запуску Java для передачі управління програмою. Класи Java, що працюють в керованих середовищах, таких як аплети та Enterprise JavaBeans, не використовують або потребують методу main (). Програма Java може містити кілька класів, які мають основні методи, а це означає, що віртуальній машині потрібно явно сказати, з якого класу запускати.

Основний метод повинен приймати масив об'єктів String. За домовленістю, він згадується як аргументи, хоча може використовуватися будь-яка інша юридична назва ідентифікатора. Починаючи з Java 5, основний метод також може використовувати змінні аргументи у формі відкритого статичного void main (String ... args), що дозволяє викликати основний метод із довільною кількістю аргументів String. Ефект цього альтернативного оголошення семантично ідентичний (параметру args, який все ще є масивом об’єктів String), але він дозволяє альтернативний синтаксис для створення та передачі масиву.

Запуск Java запускає Java шляхом завантаження заданого класу (зазначеного в командному рядку або як атрибут у JAR) та запуску його загальнодоступного методу static void main (String []). Окремі програми повинні чітко декларувати цей метод. Параметр args String [] - це масив об’єктів String, що містить будь-які

аргументи, передані в клас. Параметри до main часто передаються за допомогою командного рядка.

Друк є частиною стандартної бібліотеки Java: Клас System визначає загальнодоступне статичне поле, яке викликається. Об'єкт out є екземпляром класу PrintStream і забезпечує безліч методів друку даних до стандартного виводу, включаючи println (String), який також додає новий рядок до переданого рядка.

2.2. Spring

Spring Framework - це програма додатків та інверсія контейнера управління для платформи Java. Основні функції фреймворку можуть використовувати будь-які програми Java, але є розширення для створення веб-додатків поверх платформи Java EE (Enterprise Edition). Хоча фреймворк не нав'язує жодної конкретної моделі програмування, він став популярним у спільноті Java як доповнення до моделі Enterprise JavaBeans (EJB). Spring Framework є відкритим кодом.

Перша версія була написана Родом Джонсоном, який випустив фреймворк з публікацією його книги Expert One-on-One J2EE Design and Development у жовтні 2002 р. Фреймворк був вперше випущений за ліцензією Apache 2.0 у червні 2003 р. випуск, 1.0, вийшов у березні 2004 р. [2] Структура Spring 1.2.6 отримала нагороду за продуктивність Jolt та нагороду за інновації JAX у 2006 році. [3] [4] Весна 2.0 була випущена в жовтні 2006 р., Весна 2.5 у листопаді 2007 р., Весна 3.0 у грудні 2009 р., Весна 3.1 у грудні 2011 р., І весна 3.2.5 у листопаді 2013 р. [5] Spring Framework 4.0 вийшов у грудні 2013 року. [6] Помітні вдосконалення у Spring 4.0 включали підтримку Java SE (Standard Edition) 8, Groovy 2, деякі аспекти Java EE 7 та WebSocket.

Spring Framework 4.2.0 був випущений 31 липня 2015 року і був негайно оновлений до версії 4.2.1, яка вийшла 01 вересня 2015 року. [7] Він "сумісний з Java 6, 7 і 8, з акцентом на основні вдосконалення та сучасні веб-можливості". [8]

Spring Framework 4.3 вийшов 10 червня 2016 року і підтримуватиметься до 2020 року [9]. Це "буде остаточним поколінням в рамках загальних системних вимог Spring 4 (Java 6+, Servlet 2.5+), [...]". [10]

Очікується, що Spring 5 буде побудований на реакторному ядрі, сумісному з реактивними потоками. [11]

Spring Framework включає кілька модулів, що надають цілий ряд послуг:

- Spring Core Container: це базовий модуль Spring, який забезпечує пружинні контейнери (BeanFactory та ApplicationContext). [12]
- Аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні проблеми.
- Аутентифікація та авторизація: конфігуровані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів та практик за допомогою підпроекту Spring Security (раніше Acegi Security System for Spring).
- Домовленість щодо конфігурації: у модулі Spring Roo пропонується рішення для швидкої розробки додатків для корпоративних програм на основі Spring
- Доступ до даних: робота з реляційними системами управління базами даних на платформі Java з використанням Java Database Connectivity (JDBC) та об’єктно-реляційних інструментів відображення та з базами даних NoSQL
- Інверсія контрольного контейнера: конфігурація компонентів програми та управління життєвим циклом об'єктів Java, здійснюється головним чином за допомогою введення залежностей
- Обмін повідомленнями: конфігураційна реєстрація об'єктів прослуховування повідомлень для прозорого споживання повідомлень з черг повідомлень за допомогою служби повідомлень Java (JMS), вдосконалення відправки повідомлень через стандартні API JMS
- Модель-вигляд-контролер: фреймворк на основі HTTP та сервлетів, що забезпечує гачки для розширення та налаштування для веб-додатків та веб-сервісів RESTful (репрезентативний стан).

- Рамка віддаленого доступу: конфігуративний виклик віддалених процедур (RPC) у стилі маршалювання об’єктів Java через мережі, що підтримують виклик віддаленого методу Java (RMI), CORBA (Common Object Request Broker Architecture) та протоколи на основі HTTP, включаючи веб-послуги (SOAP (Simple Object Access Протокол))
- Управління транзакціями: об’єднує кілька API управління транзакціями та координує транзакції для об’єктів Java
- Віддалене управління: конфігураційне опромінення та управління об'єктами Java для локальної або віддаленої конфігурації за допомогою розширень Java Management Extensions (JMX)
- Тестування: підтримка класів для написання модульних тестів та інтеграційних тестів

Центральним для Spring Framework є його інверсія управління (IoC) контейнер, який забезпечує послідовний спосіб налаштування та управління об'єктами Java за допомогою відображення. Контейнер відповідає за управління життєвими циклами об'єктів конкретних об'єктів: створення цих об'єктів, виклик методів їх ініціалізації та налаштування цих об'єктів шляхом їх з'єднання.

Об'єкти, створені контейнером, також називаються керованими об'єктами або компонентами. Контейнер можна налаштувати, завантаживши файли XML (Розширювана мова розмітки) або виявивши певні анотації Java у класах конфігурації. Ці джерела даних містять визначення зерна, які надають інформацію, необхідну для створення зерен.

Об'єкти можна отримати або за допомогою пошуку залежностей, або за допомогою ін'єкції залежностей [13]. Пошук залежностей - це шаблон, коли абонент запитує у об’єкта контейнера об’єкт із певним іменем або певним типом. Введення залежності - це шаблон, коли контейнер передає об'єкти за іменами іншим об'єктам за допомогою конструкторів, властивостей або заводських методів.

У багатьох випадках не потрібно використовувати контейнер, використовуючи інші частини Spring Framework, хоча його використання, швидше

за все, спростить налаштування та налаштування програми. Контейнер Spring забезпечує послідовний механізм конфігурації програм і інтегрується майже з усіма середовищами Java, від невеликих програм до великих корпоративних програм.

Контейнер можна перетворити на частково сумісний контейнер EJB (Enterprise JavaBeans) 3.0 за допомогою проекту Pitchfork. Деякі [хто?] Критикують Spring Framework за невідповідність стандартам. [14] Однак SpringSource не вважає відповідність EJB 3 головною метою і стверджує, що Spring Framework і контейнер дозволяють створювати більш потужні моделі програмування. [15] Програміст безпосередньо не створює об'єкт, а описує, як його слід створювати, визначаючи його у файлі конфігурації Spring. Подібним чином послуги та компоненти не викликаються безпосередньо; натомість файл конфігурації Spring визначає, які служби та компоненти потрібно викликати. Цей IoC призначений для полегшення обслуговування та тестування.

Структура доступу до даних Spring вирішує загальні труднощі, з якими стикаються розробники при роботі з базами даних у додатках. Забезпечується підтримка всіх популярних платформ доступу до даних на Java: JDBC, iBatis / MyBatis, Hibernate, Java Data Objects (JDO, припинено з 5.x), Java Persistence API (JPA), Oracle TopLink, Apache OJB та Apache Cayenne, серед інших.

Для всіх цих підтримуваних фреймворків Spring надає ці функції

- Управління ресурсами - автоматичне отримання та звільнення ресурсів бази даних
- Обробка винятків - переведення виключення, пов’язаного з доступом до даних, до ієрархії доступу до даних Spring
- Участь у транзакціях - прозора участь у поточних операціях
- Розгортання ресурсу - отримання об’єктів бази даних із обгортки пулу підключень
- Абстракція для обробки двійкового великого об'єкта (BLOB) та великого об'єкта символів (CLOB)

Усі ці функції стають доступними при використанні класів шаблонів, наданих Spring для кожного підтримуваного фреймворку. Критики відзначають, що ці класи шаблонів є нав'язливими і не надають переваг перед використанням (наприклад, Hibernate API безпосередньо. [18] У відповідь розробники Spring дали можливість використовувати API Hibernate та JPA безпосередньо. Однак це вимагає прозорого управління транзакціями, оскільки код програми більше не бере на себе відповідальність за отримання та закриття ресурсів бази даних і не підтримує переклад винятків.

Разом з управління транзакціями Spring, його фреймворк доступу до даних пропонує гнучку абстракцію для роботи з фреймворками доступу до даних. Spring Framework не пропонує загального API доступу до даних; натомість, повна потужність підтримуваних API залишається незмінною. Spring Framework - це єдина платформа, доступна в Java, яка пропонує середовища керованого доступу до даних поза сервером додатків або контейнером. [19]

2.3. Intellij IDEA

IntelliJ IDEA - це інтегроване середовище розробки (IDE), написане на Java для розробки комп'ютерного програмного забезпечення. Він розроблений JetBrains (раніше відомий як IntelliJ) і доступний як ліцензована спільнота Apache 2 [6] та у власній комерційній версії. Обидва вони можуть бути використані для комерційного розвитку. [7]

Перша версія IntelliJ IDEA була випущена в січні 2001 року і була однією з перших доступних IDE Java з розширеними можливостями навігації коду та можливостями рефакторингу коду. [8] [9]

У звіті InfoWorld за 2010 рік IntelliJ отримав найвищий бал центру тестування з чотирьох найкращих інструментів програмування Java: Eclipse, IntelliJ IDEA, NetBeans та JDeveloper. [10]

У грудні 2014 року Google оголосив версію 1.0 Android Studio, IDE з відкритим вихідним кодом для програм для Android, засновану на виданні спільноти з відкритим кодом IntelliJ IDEA. [11] Інші середовища розробки, засновані на рамках IntelliJ, включають AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm та MPS. [12]

IDE надає певні функції [15], такі як заповнення коду шляхом аналізу контексту, навігація кодом, яка дозволяє перейти до класу або оголошення в коді безпосередньо, рефакторинг коду, налагодження коду [16], встановлення посилань та можливості виправлення невідповідностей за допомогою пропозицій.

IDE забезпечує [15] інтеграцію з інструментами складання / упаковки, такими як grunt, bower, gradle та SBT. Він підтримує системи контролю версій, такі як Git, Mercurial, Perforce та SVN. Бази даних, такі як Microsoft SQL Server, Oracle, PostgreSQL, SQLite і MySQL, можна отримати безпосередньо з IDE у версії Ultimate через вбудовану версію DataGrip.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови Java
- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме IntelliJ IDEA 2021.1.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Діаграма варіантів використання

Найпростіша діаграма використання - це представлення взаємодії між користувачем та системою, яка показує взаємозв'язок між користувачем та різними видами використання, що беруть участь у користувачі. Діаграми випадків використання можуть ідентифікувати різні типи користувачів системи та різні випадки використання, і часто супроводжуються іншими типами діаграм. Використовуйте кола або еліпси для позначення використання.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів може допомогти забезпечити огляд системи більш високого рівня. Я вже говорив, що "план використання - це принцип вашої системи".

Завдяки спрощеному характеру, план використання може бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ та надати зацікавленим сторонам ідеї щодо розвитку системи. Сіау та Лі провели дослідження, щоб визначити, чи взагалі існувала дійсна ситуація для схем використання або вони були непотрібними. Було виявлено, що діаграми випадків використання передають намір системи більш спрощеним чином зацікавленим сторонам і що вони "інтерпретуються більш повно, ніж діаграми класів".

Мета використання діаграм - показати динамічні аспекти системи. Інші схеми та документи можуть бути використані для забезпечення повного функціонального та технічного представлення системи. Вони забезпечують спрощене графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямоугольник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації про полезну інформацію,
- актор (англ. актор) - стилізований людський персонаж, обозначаючий набір ролей користувача (розуміється в широкому змісті: людина, внешняя сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що обозначає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, ілюстріруючою повіддю. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії студента в системі.

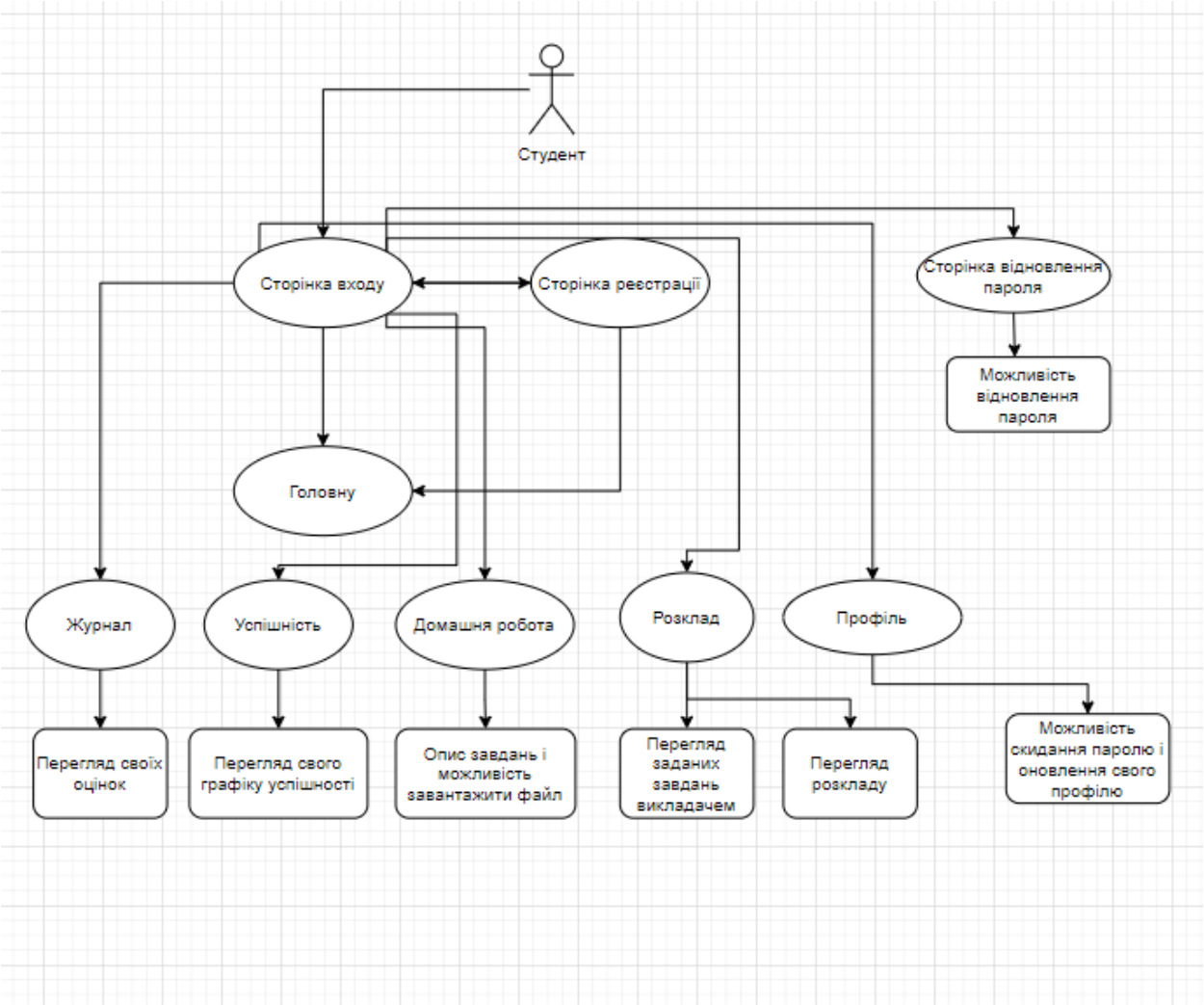
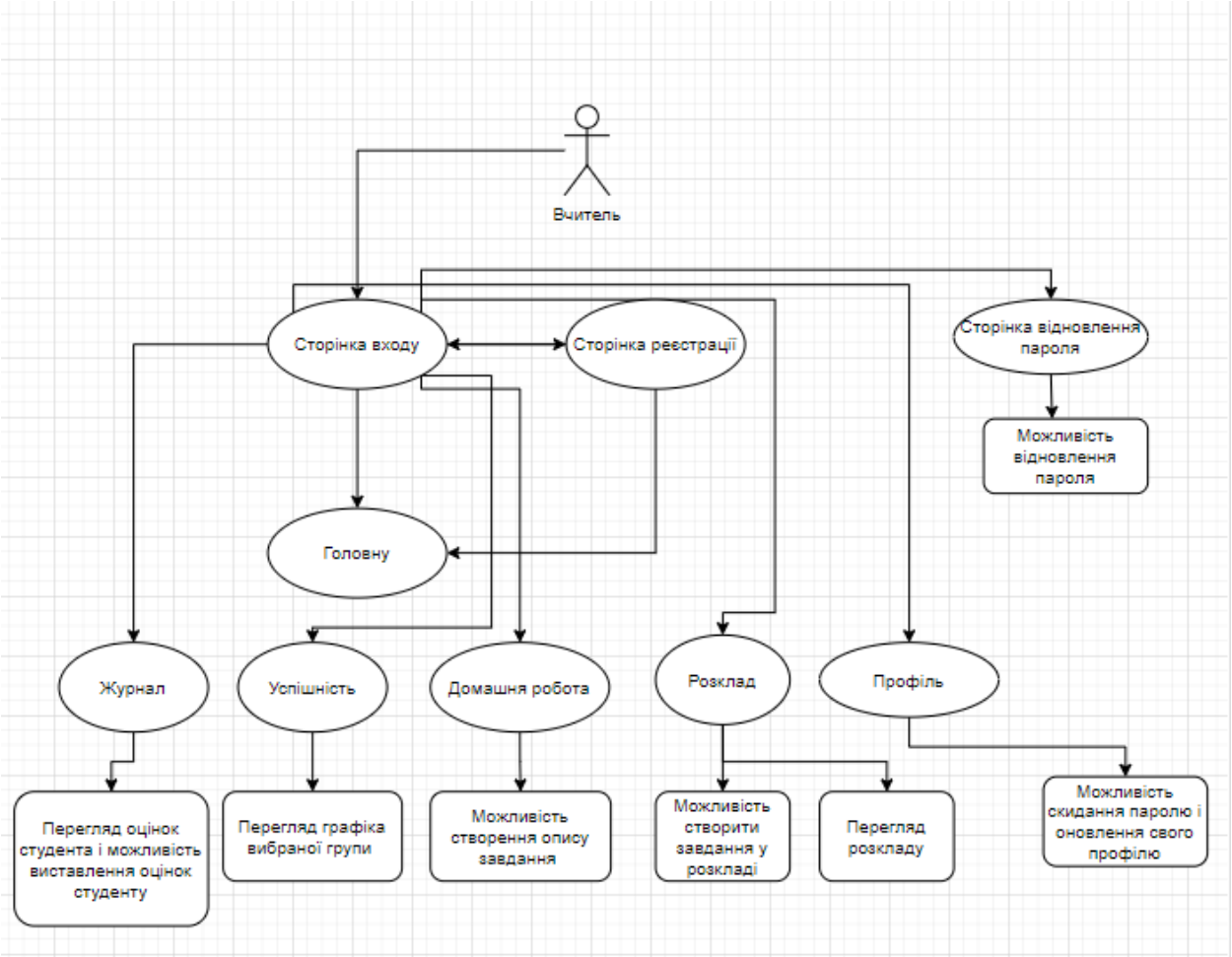


Рисунок 3.1 — Діаграма варіантів використання для студента

На рисунку 3.2 зображено діаграму варіантів використання, яка описує можливі дії вчителя в системі.



Риунок 3.2 — Діаграма варіантів використання для вчителя

3.2. Діаграма класів

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи ряд класів ідентифікується та згруповується у схему класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проекту часто поділяються на ряд підкласів.

Залежність - це семантичний зв’язок між залежними та незалежними елементами моделі. Він існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела). Ця асоціація є односпрямованою. Залежність відображається у вигляді штрихової лінії з відкритою стрілкою, яка вказує від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою стану або машиною стану UML.

Асоціація представляє родину посилань. Двійкова асоціація (з двома кінцями) зазвичай представляється у вигляді рядка. Асоціація може пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінці асоціації можна прикрасити іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Існує чотири різні типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш поширеними.

Наприклад, клас польоту асоціюється з класом літака двонаправлено. Асоціація представляє статичне відношення, яке ділиться між об'єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частково цілі або часткові стосунки. Як показано на зображенні, професор "має" клас для викладання. Як тип асоціації, агрегація може бути названа та мати ті самі прикраси, що і асоціація. Однак агрегація не може включати більше двох класів; це має бути бінарна асоціація. Крім того, навряд чи існує різниця між агрегаціями та асоціаціями під час реалізації, і діаграма може взагалі пропустити відносини агрегування. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але вміщені класи не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера все ще існує, коли контейнер знищений.

В UML він графічно представлений у вигляді порожнистої форми ромба на вміщуючому класі одним рядком, що зв’язує його із вміщеним класом. Сукупність - це семантично розширений об’єкт, який у багатьох операціях трактується як одиниця, хоча фізично він складається з декількох менших об’єктів.

Приклад: Бібліотека та студенти. Тут студент може існувати без бібліотеки, зв'язок між студентом і бібліотекою є агрегацією.

Це вказує на те, що один із двох пов’язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу. Зразкове дерево узагальнень цієї форми зустрічається в біологічній класифікації: людина - це підклас маймуни, який є підкласом ссавців тощо. Зв’язок найлегше зрозуміти за допомогою фрази „A - це B” (людина - це ссавець, ссавець - тварина).

Графічне представлення UML узагальнення - це форма порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також відомі як спадщина або відносини "є".

Суперклас (базовий клас) у відносинах узагальнення також відомий як "батьківський", суперклас, базовий клас або базовий тип.

Підтип у відносинах спеціалізації також відомий як "дочірній", підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Зверніть увагу, що ці стосунки нічим не схожі на біологічні стосунки батьків та дітей: використання цих термінів надзвичайно поширене, але може ввести в оману.

A - це тип B

Наприклад, "дуб - це тип дерева", "автомобіль - це тип транспортного засобу"

Узагальнення може бути показано лише на діаграмах класів та на діаграмах використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, в яких один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, яку вказує інший елемент моделі (постачальник).

Графічне представлення UML реалізації - це порожниста форма трикутника на кінці інтерфейсу штрихової лінії (або дерева рядків), яка з'єднує її з одним або кількома реалізаторами. Проста головка стрілки використовується на кінці інтерфейсу штрихової лінії, що з'єднує її з користувачами. У діаграмах компонентів використовується графічна умова «м'яч і сокет» (реалізатори виставляють кульку або льодяник, тоді як користувачі показують сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднує елемент клієнта з елементом постачальника. Зв'язок реалізації між класами / компонентами та інтерфейсами показує, що клас / компонент реалізує операції, пропоновані інтерфейсом.

Залежність - це слабша форма зв'язку, яка вказує на те, що один клас залежить від іншого, оскільки він використовує його в певний момент часу. Один клас залежить від іншого, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між двома класами дуже слабкі. Вони взагалі не реалізовані зі змінними-членами. Швидше

вони можуть бути реалізовані як аргументи функції-члена. Ці відносини зазвичай описуються як "А має В" (у матері-кота є кошенята, у кошенят - мати-кішка).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення. Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

Діаграма класів, яка повністю відображає внутрішню будову системи, зображена на рисунку 3.3.

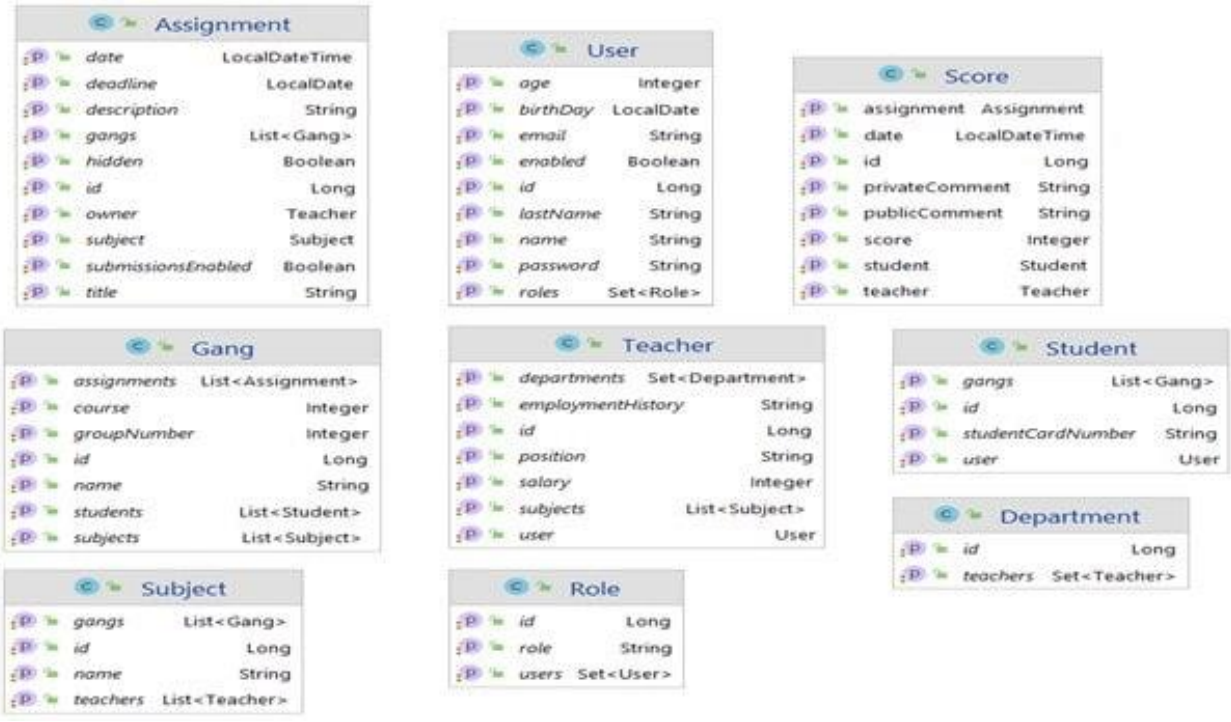


Рисунок 3.3 — Діаграма класів програмного продукту

3.3. Сторінка входу

Сторінка входу представляє собою форму авторизації, яка допомагає відсіяти незареєстрованих користувачів, тобто, підвищити безпечність конфіденційної інформації.

В загальному понятті, авторизація - це функція визначення прав / привілеїв доступу до ресурсів, яка пов’язана із загальною інформаційною та комп’ютерною безпекою, а також з управлінням доступом, зокрема. Більш формально "авторизувати" - це визначати політику доступу. Наприклад, кадровий персонал, як правило, уповноважений отримувати доступ до записів працівників, і ця політика часто оформляється як правила контролю доступу в комп'ютерній системі. Під час роботи система використовує правила контролю доступу, щоб вирішити, чи будуть запити на доступ від (аутентифікованих) споживачів схвалені (надані) чи відхилені (відхилені). Ресурси включають окремі файли або дані елемента, комп'ютерні програми, комп'ютерні пристрої та функціональні можливості, що надаються комп'ютерними програмами. Прикладами споживачів є користувачі комп'ютерів, комп'ютерне програмне забезпечення та інше обладнання на комп'ютері.

Реалізація авторизації користувача в системі виконана з використанням таких технологій як JWT – Json Web Tocken та Spring Security. Метод очікує один параметр типу LoginRequest на вхід який включає в собі два поля username та password та повертає після успішного виконання об’єкт типу AuthenticationResponse який включає в себе:

- authenticationToken – токен з допомогою якого ми будемо перевіряти чи має користувач доступ до сайту, також має свій термін придатності
- refreshToken – цей токен використовується у випадку коли в authenticationToken закінчується термін придатності і нам потрібно його замінити на новий.
- expiresAt – поле яке зберігає саме термін придатності authenticationToken.
- username – ім’я користувача для подальшого відображення на UI

В тілі методу спершу викликається метод authenticate який виконує аунтетифікацію користувача та визначає його ролі в системі, після свого виконання повертає об’єкт типу Authentication – це об’єк який я представленням авторизованого користувача в Spring. Також обов’язковим порядком отриманий об’єкт зберігається в контекст програмного додатку, для того що б ми могли перевірити чи авторизований користувач та отримати інформацію про нього в коді там де це буде потрібно. Наступним кроком йде генерація токена з допомогою екземпляру обєкту Authentication. Після цих кроків створюється об’єкт типу AuthenticationResponse який ми будемо повертати, створення цього об’єкту відбувається з використанням Builder патерну програмування який використовується для винесення конструювання об’єкта за межі цього класу, та надати це вкладеним класам які прийнято називати Builder. Код методу який відображає логіку авторизації зображений на рисунку 3.4.

```
public AuthenticationResponse login(LoginRequest loginRequest) {
    Authentication authenticate = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
            loginRequest.getUsername(), loginRequest.getPassword()));
    SecurityContextHolder.getContext().setAuthentication(authenticate);
    String token = jwtProvider.generateToken(authenticate);

    return AuthenticationResponse.builder()
            .authenticationToken(token)
            .refreshToken(refreshTokenService.generateRefreshToken().getToken())
            .expiresAt(Instant.now().plusMillis(jwtProvider.getJwtExpirationInMills()))
            .username(loginRequest.getUsername())
            .build();
}
```

Рисунок 3.4 — Код методу входу

3.4. Сторінка реєстрації

Сторінка реєстрації представляє собою форму, яка несе в собі функціонал створення нового облікового запису шляхом додавання його в систему.

В загальному розумінні користувач - це особа, яка користується комп’ютером або послугою мережі. Користувачам комп'ютерних систем та програмних

продуктів, як правило, не вистачає технічного досвіду, необхідного для повного розуміння їх роботи. [1] Потужні користувачі використовують розширені функції програм, хоча вони не обов'язково здатні до комп'ютерного програмування та системного адміністрування. [2] [3]

Користувач часто має обліковий запис і ідентифікується системою за допомогою імені користувача (або імені користувача). Інші терміни імені користувача включають ім'я для входу, ім'я екрана (або ім'я екрана), ім'я облікового запису, псевдонім (або псевдонім) та дескриптор, що походить від ідентичного терміна радіостанції для громадян.

Деякі програмні продукти надають послуги іншим системам і не мають прямих кінцевих користувачів.

Реалізація реєстрації в програмному застосунку використовує Spring Security, Spring Data JPA(Java Persistence API), JavaMailSender. Метод буде виконуватись в транзакції, тому що він зберігає дані користувача в базу даних та відправляє електронний лист з підтвердженням, тому нам потрібно впевнитись що метод виконається повністю. Першим кроком в методі нам потрібно створити новий пустий об'єкт типу User та з допомогою set та get методів перенести інформацію з об'єкта типу RegisterRequest який ми очікуємо як параметр в наш метод, ми переносимо такі поля як: username ТА email. Окремо потрібно розглядати поле password, тому що з точки зору безпеки заборонено зберігати пароль користувача в базі даних в його звичайному вигляді, тому потрібно обов'язково хешувати методом хешування SHA-2 – на сьогоднішній час це найбезпечнішій метод хешування. Отже, хешування відбувається за допомогою виклику методу encode в об'єкту passwordEncoder. Наступним кроком заповнюємо поля які зашились, а саме: created – теперішнім часовим значенням в момент виклику методу та enabled – поставимо в false, тому що ми очікуємо що при реєстрації обліковий запис користувача буде вимкнений і його потрібно активувати підтвердивши свою електронну адресу. Наступним кроком потрібно записати отриманні дані в базу даних, для цього викликається метод save в об'єкту userRepository для генерування insert запиту в табличку Users. Далі створюється унікальний код для створення

унікального URL при переході на який користувач активує свій обліковий запис. Наступним кроком створюється та відбравляється тіло електронного листа який власне включає URL для активації. Код методу який відображає логіку авторизації зображений на рисунку 3.5.

```
@Transactional
public void signup(RegisterRequest registerRequest) {
    User user = new User();
    user.setUsername(registerRequest.getUsername());
    user.setEmail(registerRequest.getEmail());
    user.setPassword(passwordEncoder.encode(registerRequest.getPassword()));
    user.setCreated(Instant.now());
    user.setEnabled(false);
    userRepository.save(user);

    String token = generateVerificationToken(user);
    String body = "Thank you for signing up, please click on the below url to activate " +
            "your account : " + "http://localhost:8080/api/auth/accountVerification/" + token;
    mailService.sendMail(new NotificationEmail( subject: "Please Activate your Account", user.getEmail(), body));
}
```

Рисунок 3.5 — Код методу реєстрації

3.5. Виставлення оцінок

Для того що б зберегти оцінку нам обов’язково потрібно знати викладача який її ставить, завдання за яке ставиться оцінка, студент якому ця оцінка викладається і звісно сама оцінка. Частина інформації яка нам потрібна вже є збережена базі, тому нам потрібно просто знати унікальний ідентифікатор за яким ми зможемо діставати інформації. Тому в методі який зберігає оцінку в базу даних спершу витягуються об’єкти типів Teacher, Assignment, Student. В випадку якщо одного з цих об’єктів немає в базі даних ми будемо закінчувати виконання методу та відправляти на UI повідомлення що ми не можемо знайти об’єкт за наданим ідентифікатором. Після того ж як всі об’єкти успішно знайшлись ми будемо створювати об’єкт типу Score для того що б перенести в нього необхідну інформацію перед збереженням в базу даних. Також варто згадати що об’єкт типу Score надає нам підтримку публічного коментаря – який після виставлення оцінки буде бачити студент, та приватного коментаря який буде бачити тільки викладач

тому він зможе зберігати свої замітки. Після всіх маніпуляцій об’єкт який ми отримували ми записуємо в базу даних викликом методу save в об’єкту scoreRepository. Код методу який відображає логіку авторизації зображений на рисунку 3.6.

```
@Override
public Score saveScore(ScoreSaveRequest scoreSaveRequest) {
    Score score = new Score();

    Teacher teacher = teacherRepository.findById(scoreSaveRequest.getTeacherId())
            .orElseThrow(() -> new RuntimeException("not found Teacher by this id - " + scoreSaveRequest.getTeacherId()));

    Assignment assignment = assignmentRepository.findById(scoreSaveRequest.getAssignmentId())
            .orElseThrow(() -> new RuntimeException("not found Teacher by this id - " + scoreSaveRequest.getAssignmentId()));

    Student student = studentRepository.findById(scoreSaveRequest.getStudentId())
            .orElseThrow(() -> new RuntimeException("not found Teacher by this id - " + scoreSaveRequest.getStudentId()));

    score.setScore(scoreSaveRequest.getScore());
    score.setDate(LocalDateTime.now());
    score.setPublicComment(scoreSaveRequest.getPublicComment());
    score.setPrivateComment(scoreSaveRequest.getPrivateComment());
    score.setTeacher(teacher);
    score.setStudent(student);
    score.setAssignment(assignment);

    return scoreRepository.save(score);
}
```

Рис. 3.6 — Код методу збереження оцінок

3.6. Завдання

Сторінка завдань представляє собою табличне представлення списку завдань, розбите по категоріям, таким як день здачі, предмет, тощо.

Відображення завдань студенту реалізовано надзвичайно зручно, тому що метод який відповідає за повернень завдань студенту повертає об’єкт типу AssignmentSummaryDto який зображений на рисунку 3.7. Кожне поле це список завдань по певним критеріям, розглянемо детальніше кожен з них:

- today - зберігає завдання які потрібно зробити сьогодні.
- tomorrow – зберігає завдання в яких дедлайн до завтра, виключаючи завдання які потрібно зробити сьогодні.

- overdue – зберігає інформацію про завдання які студент не встиг здати
- longTerm – зберігає завдання на виконання яких в студента ще дуже багато часу починаючи від завтра, закінчуючи кінцем навчального семестру.

```
public class AssignmentSummaryDto {
    List<AssignmentDto> today;
    List<AssignmentDto> tomorrow;
    List<AssignmentDto> overdue;
    List<AssignmentDto> longTerm;
}
```

Рис. 3.7 — Клас AssignmentSummaryDto

Метод який відповідає за заповнення цих списків очікує як параметр тільки ідентифікатор студента, для того що б віддати по ньому фільтровані завдання. Половина тіла методу займають виклики методів з шару репозиторію, які займаються витягненням інформації з бази даних. Всі SQL запити написані на JPQL – це мові запитів яка дає можливість писати запити використовуючи ООП підхід в запитах та фігурувати об’єктами а не записами, також JPQL під час виконання компілюється в потрібний діалект, тобто написавши раз на JPQL ви можете використовувати запити з будь-якою базою даних, наприклад: MySQL, MS SQL, Oracle, PostgreSQL, та інші.

Отже, спершу ми дістаємо та фільтруємо списки з бази даних, далі використовуючи Stream API змінюємо тип отриманих результатів з Assignment в AssignmentDto шляхом перенесення даних з одного об’єкту в інший. Перенесення в DTO (Data Transfer Object) об’єкт потрібно для коректного відображення об’єкту на UI також для уникнення вкладених об’єктів коли це не потрібно. Також після отримання списків DTO-об’єктів вони встановлюються в об’єкт який буде повертатись як результат цього методу. Код методу який відображає логіку авторизації зображений на рисунку 3.8.

```java
@Override
public AssignmentSummaryDto getAssignmentSummary(Long studentId) {
    AssignmentSummaryDto assignmentSummaryDto = new AssignmentSummaryDto();

    List<Assignment> today = assignmentRepository
            .findAllByDateRangeAndStudentId(LocalDate.now(), studentId);
    List<Assignment> tomorrow = assignmentRepository
            .findAllByDateRangeAndStudentId(LocalDate.now().plusDays(1), studentId);
    List<Assignment> longTerm = assignmentRepository
            .findAllByStudentIdAAndDeadlineBefore(LocalDate.now().plusDays(1), studentId);
    List<Assignment> overdue = assignmentRepository
            .findAllByStudentIdAAndDeadlineAfter(LocalDate.now(), studentId);

    assignmentSummaryDto.setTomorrow(tomorrow.stream().map(assignmentMapper::toDto).collect(Collectors.toList()));
    assignmentSummaryDto.setToday(today.stream().map(a -> mapper.map(a, AssignmentDto.class)).collect(Collectors.toList()));
    assignmentSummaryDto.setOverdue(overdue.stream().map(assignmentMapper::toDto).collect(Collectors.toList()));
    assignmentSummaryDto.setLongTerm(longTerm.stream().map(assignmentMapper::toDto).collect(Collectors.toList()));
    return assignmentSummaryDto;
}
```

Рис. 3.8 — Код методу отримання завдань для студента

3.7. Профіль

Обліковий запис користувача в основному побудований на класі User, а також на класах Teacher та Student які мають свої унікальні поля. В конструюванні класу User використовувались такі технології як:

- Lombok – для зменшення рутинного та згенерованого коду, а саме get та set методи, конструктори, метод toString та метод hashCode.
- Java Persistence API – використовується для представлення класу в вигляді таблиці в базі даних, а також для побудування сполучень між класами.
- Hibernate Validator – використовується для валідації вхідних параметрів.

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    @Id @GeneratedValue
    private Long id;
    @Email
    private String email;
    @Pattern(regexp = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\\S+$).{8,}$")
    private String password;
    private Boolean enabled;
    private LocalDate birthDay;
    @Min(value = 14)
    private Integer age;
    @NotNull
    @NotBlank
    private String name;
    @NotNull
    @NotBlank
    private String lastName;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "user_role",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles;
}
```

Рис. 3.9 — Код класу User

ВИСНОВКИ

Метою даної роботи стало розроблення серверної системи для інформаційної системи електронного журналу засобами мови програмування Java.

Для досягнення поставленої мети в процесі виконання роботи наступні завдання:

- Було проаналізовано загальне розуміння поняття «сервер», виявлено його основні структурні частини, особливості та призначення.
- Проведено повний аналіз засобів розробки поняття клієнт-серверної архітектури, виявлено її особливості.
- Проведено огляд поняття електронного журналу, виявлено його суть та історичні витоки даного явища.
- В якості мови програмування було обрано Java, як мову яка найбільш повною мірою підходить під поставлені задачі. Проведено огляд мови програмування Java.
- В якості середовища розробки обрано Itellij IDEA, інтегроване середовище розробки компанії JetBrains.
- Було побудовано діаграму класів
- Було розроблено сторінку входу
- Було розроблено сторінку реєстрації
- Було розроблено сторінку виставлення оцінок
- Було розроблено сторінку статистики успішності
- Було розроблено сторінку розкладу
- Було розроблено сторінку завдань
- Було розроблено сторінку профілю

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті було отримано повноцінну серверну частину для інформаційної системи електронного журналу, яка готова до використання в реальних умовах у зв’язці з UI.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Windows Server Administration Fundamentals. Microsoft Official Academic Course. 111 River Street, Hoboken, NJ 07030: John Wiley & Sons. 2011. pp. 2–3. ISBN 978-0-470-90182-3.
2. Comer, Douglas E.; Stevens, David L (1993). Vol III: Client-Server Programming and Applications. Internetworking with TCP/IP. Department of Computer Sciences, Purdue University, West Lafayette, IN 479: Prentice Hall. pp. 11d. ISBN 978-0-13-474222-9.
3. Richard A. Henle, Boris W. Kuvshinoff, C. M. Kuvshinoff (1992). Desktop computers: in perspective. Oxford University Press. p. 417. ISBN 9780195070316. Server is a fairly recent computer networking term derived from queuing theory.
4. Rulifson, Jeff (June 1969). DEL. IETF. doi:10.17487/RFC0005. RFC 5. Retrieved 30 November 2013.
5. Shapiro, Elmer B. (March 1969). Network Timetable. IETF. doi:10.17487/RFC0004. RFC 4. Retrieved 30 November 2013.
6. Using the HTTP Publish-Subscribe Server, Oracle
7. IT Explained. "Server - Definition and Details". www.paessler.com.
8. IT Explained. "DNS Server Not Responding". www.dnsservernotresponding.org.
9. "Web Servers". IT Business Edge. Retrieved July 31, 2013.
10. Li, Huang, Shen, Chu (2010). ""A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility". Usenix Annual Tech Conference 2010" (PDF). Retrieved 2017-01-30.
11. "Google uncloaks once-secret server". CNET. CBS Interactive. Retrieved 2017-01-30.
12. "Mobile Server, Power to go, EUROCOM Panther 5SE". Archived from the original on 2013-03-17.
13. "Mobile Server Notebook".
14. "Server-caliber Computer Doubles as a Mobile Workstation".

15. "Usage statistics and market share of Linux for websites". Retrieved 18 Jan 2013.
16. "Server Oriented Operating System". Retrieved 2010-05-25.
17. Markoff, John (31 Jul 2011). "Data Centers Using Less Power Than Forecast, Report Says". NY Times. Retrieved 18 Jan 2013.
18. "SMART 2020: Enabling the low carbon economy in the information age" (PDF). The Climate Group. 6 Oct 2008. Archived from the original (PDF) on 22 November 2010. Retrieved 18 Jan 2013.
19. Binstock, Andrew (May 20, 2015). "Java's 20 Years of Innovation". Forbes. Archived from the original on March 14, 2016. Retrieved March 18, 2016.
20. Barbara Liskov with John Guttag (2000). Program Development in Java - Abstraction, Specification, and Object-Oriented Design. USA, Addison Wesley. ISBN 9780201657685.
21. Chaudhary, Harry H. (July 28, 2014). "Cracking The Java Programming Interview :: 2000+ Java Interview Que/Ans". Retrieved May 29, 2016.
22. Java 5.0 added several new language features (the enhanced for loop, autoboxing, varargs and annotations), after they were introduced in the similar (and competing) C# language. [1] Archived March 19, 2011, at the Wayback Machine [2] Archived January 7, 2006, at the Wayback Machine
23. Gosling, James; McGilton, Henry (May 1996). "The Java Language Environment". Archived from the original on May 6, 2014. Retrieved May 6, 2014.
24. Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad. "The Java Language Specification, 2nd Edition". Archived from the original on August 5, 2011. Retrieved February 8, 2008.
25. "The A-Z of Programming Languages: Modula-3". Computerworld.com.au. Archived from the original on January 5, 2009. Retrieved June 9, 2010.
26. Niklaus Wirth stated on a number of public occasions, e.g. in a lecture at the Polytechnic Museum, Moscow in September 2005 (several independent first-hand accounts in Russian exist, e.g. one with an audio recording: Filippova, Elena (September 22, 2005). "Niklaus Wirth's lecture at the Polytechnic Museum in Moscow".), that the Sun Java design team licensed the Oberon compiler sources a

27. number of years prior to the release of Java and examined it: a (relative) compactness, type safety, garbage collection, no multiple inheritance for classes – all these key overall design features are shared by Java and Oberon.
28. Patrick Naughton cites Objective-C as a strong influence on the design of the Java programming language, stating that notable direct derivatives include Java interfaces (derived from Objective-C's protocol) and primitive wrapper classes. [3] Archived July 13, 2011, at the Wayback Machine
29. TechMetrix Research (1999). "History of Java" (PDF). Java Application Servers Report. Archived from the original (PDF) on December 29, 2010. The project went ahead under the name green and the language was based on an old model of UCSD Pascal, which makes it possible to generate interpretive code.
30. "A Conversation with James Gosling – ACM Queue". Queue.acm.org. August 31, 2004. Archived from the original on July 16, 2015. Retrieved June 9, 2010.
31. In the summer of 1996, Sun was designing the precursor to what is now the event model of the AWT and the JavaBeans component architecture. Borland contributed greatly to this process. We looked very carefully at Delphi Object Pascal and built a working prototype of bound method references in order to understand their interaction with the Java programming language and its APIs.White Paper About Microsoft's Delegates
32. "Chapel spec (Acknowledgements)" (PDF). Cray Inc. October 1, 2015. Archived (PDF) from the original on February 5, 2016. Retrieved January 14, 2016.
33. "Gambas Documentation Introduction". Gambas Website. Archived from the original on October 9, 2017. Retrieved October 9, 2017.
34. "Facebook Q&A: Hack brings static typing to PHP world". InfoWorld. March 26, 2014. Archived from the original on February 13, 2015. Retrieved January 11, 2015.
35. "Write once, run anywhere?". Computer Weekly. May 2, 2002. Retrieved July 27, 2009.
36. "1.2 Design Goals of the Java™ Programming Language". Oracle. January 1, 1999. Archived from the original on January 23, 2013. Retrieved January 14, 2013.

37. McMillan, Robert (August 1, 2013). "Is Java Losing Its Mojo?". wired.com. Archived from the original on February 15, 2017. Retrieved March 8, 2017. Java is on the wane, at least according to one outfit that keeps on eye on the ever-changing world of computer programming languages. For more than a decade, it has dominated the TIOBE Programming Community Index, and is back on top – a snapshot of software developer enthusiasm that looks at things like internet search results to measure how much buzz different languages have. But lately, Java has been slipping.
38. Chan, Rosalie (January 22, 2019). "The 10 most popular programming languages, according to the 'Facebook for programmers'". Business Insider. Archived from the original on June 29, 2019. Retrieved June 29, 2019.
39. "JavaOne 2013 Review: Java Takes on the Internet of Things". www.oracle.com. Archived from the original on April 19, 2016. Retrieved June 19, 2016. Alt URL
40. "Why should I uninstall older versions of Java from my system?". Oracle. Retrieved September 9, 2016.
41. Byous, Jon (c. 1998). "Java technology: The early years". Sun Developer Network. Sun Microsystems. Archived from the original on April 20, 2005. Retrieved April 22, 2005.
42. Object-oriented programming "The History of Java Technology". Sun Developer Network. c. 1995. Archived from the original on February 10, 2010. Retrieved April 30, 2010.
43. Murphy, Kieron (October 4, 1996). "So why did they decide to call it Java?". JavaWorld. Retrieved 2020-07-13.
44. Kabutz, Heinz; Once Upon an Oak Archived April 13, 2007, at the Wayback Machine. Artima. Retrieved April 29, 2007.
45. "JAVASOFT SHIPS JAVA 1.0". Archived from the original on March 10, 2007. Retrieved May 13, 2018.
46. Object-oriented Programming with Java: Essentials and Applications. Tata McGraw-Hill Education. p. 34.

47. "JSG – Java Study Group". open-std.org. Archived from the original on August 25, 2006. Retrieved August 2, 2006.
48. "Why Java™ Was – Not – Standardized Twice" (PDF). Archived (PDF) from the original on January 13, 2014. Retrieved June 3, 2018.
49. "What is ECMA—and why Microsoft cares". Archived from the original on May 6, 2014. Retrieved May 6, 2014.
50. "Java Community Process website". Jcp.org. May 24, 2010. Archived from the original on August 8, 2006. Retrieved June 9, 2010.
51. "JAVAONE: Sun – The bulk of Java is open sourced". GrnLight.net. Archived from the original on May 27, 2014. Retrieved May 26, 2014.
52. "Sun's Evolving Role as Java Evangelist". O'Reilly Media. Archived from the original on September 15, 2010. Retrieved August 2, 2009.
53. "Oracle and Java". oracle.com. Oracle Corporation. Archived from the original on January 31, 2010. Retrieved August 23, 2010. Oracle has been a leading and substantive supporter of Java since its emergence in 1995 and takes on the new role as steward of Java technology with a relentless commitment to fostering a community of participation and transparency.

ДОДАТКИ

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка API для моніторингу роботи учнів мовою Java

Виконав студент 4 курсу
групи ПД-44
Вихристюк Олександр Вікторович
Керівник роботи
Дібрівний Олесь Андрійович

Київ – 2020

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - розробка API для електронного журналу на мові програмування Java.
- **Об'єкт дослідження –** перенесення документообігу в електронний вигляд та автоматизація рутинних процесів.
- **Предмет дослідження -** електронний журнал, для переходу від паперового журналу.

АНАЛОГИ

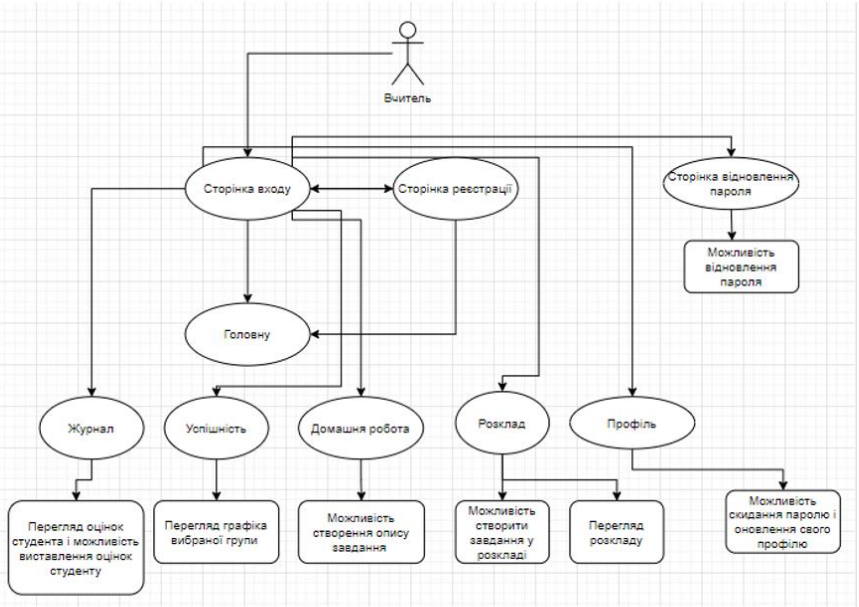
Переваги:
- Більш зручна система для шкіл
- Більше не рутинних можливостей
- Зрозуміла документація

Недоліки:
- Незручність інтерфейсу
- Важкість в освоєнні
- Помилки в перекладі на російську мову
- Безліч посилань на інші не захищені сайти

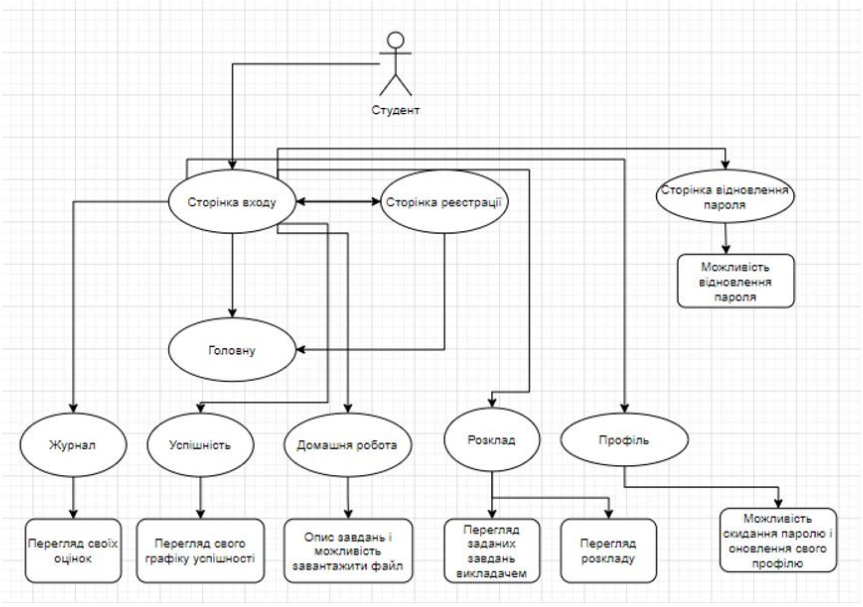
ТЕХНІЧНІ ЗАВДАННЯ

- *1 Механізм реєстрації*
- *2 Механізм авторизації*
- *3 Механізм показу та зміни розкладу*
- *4 Механізм підрахунку статистики успішності учнів*
- *5 Механіка виставлення оцінок*
- *6 Алгоритм роботи з завданнями*
- *7 Реалізація профіля користувача*
- *8 Механіка взаємодії з БД*
- *9 Механізм обробки запитів від клієнтської частини*

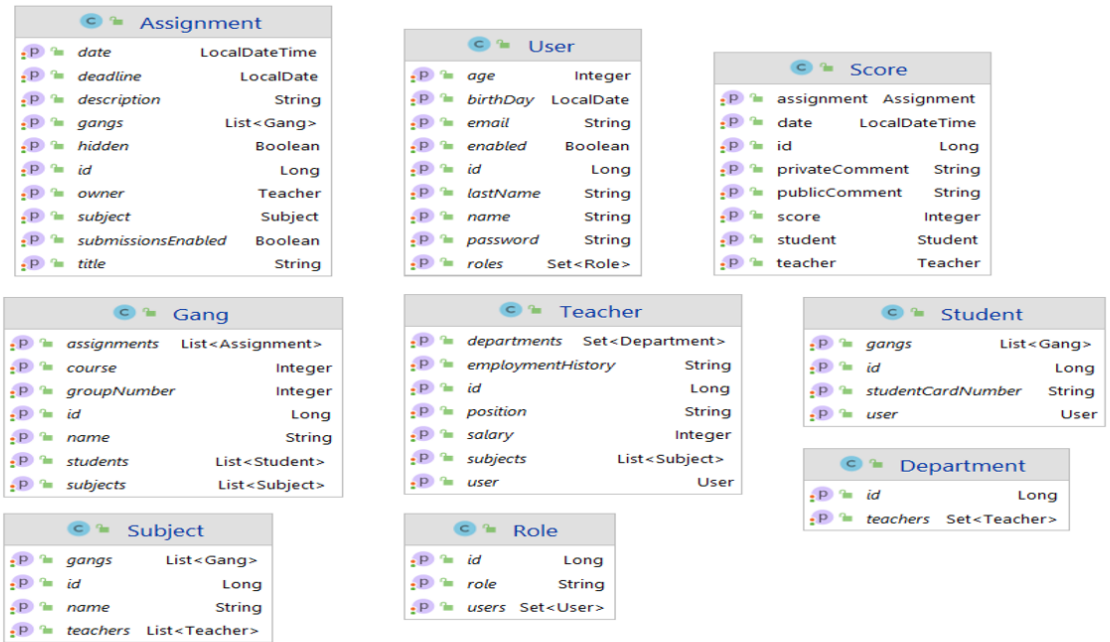
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



Діаграма варіантів використання, яка описує можливі дії викладача



Діаграма варіантів використання, яка описує можливі дії студента



Діаграма класів

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Науково-технічна конференція «Застосування програмного забезпечення в ІКТ», РОЗРОБКА API ДЛЯ ВІДСТЕЖЕННЯ УСПІШНОСТІ УЧНІВ В НАВЧАЛЬНИХ ЗАКЛАДАХ НА МОВІ JAVA
- XII Науково-технічна конференція студентів та молодих вчених «Сучасні інфокомунікаційні технології», ВИБІР ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ВЕЛИКИХ ТА НАДІЙНИХ СТИСТЕМ»

ВИСНОВКИ

1. Було проаналізовано загальне розуміння поняття «сервер», виявлено його основні структурні частини, особливості та призначення.
2. Проведено повний аналіз засобів розробки поняття клієнт-серверної архітектури, виявлено її особливості.
3. Проведено огляд поняття електронного журналу
4. Була вибрана мова програмування Java, та середовище розробки Intellij IDEA
5. Були розроблені діаграми класів та діаграми використання користувачами
5. Було розроблено базовий функціонал
6. Протестовано розроблений функціонал

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті було отримано повноцінну серверну частину для інформаційної системи електронного журналу, яка готова до використання в реальних умовах у зв’язці з UI.

ДЯКУЮ ЗА УВАГУ!