МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є
"_____"_____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА (ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Програмне забезпечення системи аналізу та обліку успішності школярів

Виконавець: Макарьєв Єгор Олександрович

Керівник: Халімон Наталія Федорівна

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.
«_____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи (проєкту)
Макарьєва Єгора Олександровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної/кваліфікаційної роботи (проєкту): Програмне забезпечення системи аналізу та обліку успішності школярів
затверджена наказом ректора від «16» вересня 2022 р. №1530/СТ
2. Термін виконання роботи (проєкту): з 05 вересня 2022 року по 30 листопада 2022 року.
3. Вихідні дані до роботи (проєкту):мова програмування *Java*, структурована мова запитів *SQL*, середовище розробки *IntelliJ IDEA*, система управління базами даних *MySQL*.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) Огляд веб-додатків системи аналізу та обліку успішності школярів і засобів для їх розробки;
2) Проектування програмного забезпечення системи аналізу та обліку успішності школярів;
3) Розробка програмного забезпечення системи аналізу та обліку успішності школярів;
4) Дослідження швидкодії виконання запитів при аналізі та обліку успішності школярів.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Модель бази даних;
2) Діаграма класів;
3) Веб-сторінка аналізу успішності класів;
4) Веб-сторінка аналізу успішності учнів;
5) Веб-сторінка з інформацією про олімпіаду.

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі кваліфікаційної роботи.	16.09.22-17.09.22	
2	Вивчити спеціальну літературу і технічну документацію.	17.09.22-20.09.22	
3	Проаналізувати існуючі програмні засоби систем аналізу та обліку досягнень школярів	20.09.22-22.09.22	
4	Написати розділ 1.	22.09.22-24.09.22	
5	Проаналізувати обрану платформу розробки. Спроектувати програмний засіб системи аналізу та обліку досягнень школярів	24.09.22-01.10.22	
6	Написати розділ 2.	01.10.22-03.10.22	
7	Виконати розробку програмного засобу системи аналізу та обліку досягнень школярів	03.10.22-19.10.22	
8	Написати розділ 3.	19.10.22-21.10.22	
9	Проаналізувати отримані в ході виконання роботи дані. Навести результати досліджень.	21.10.22-23.10.22	
10	Написати розділ 4.	23.10.22-25.10.22	
11	Оформити пояснювальну записку. Підготувати графічний демонстраційний матеріал	25.10.22-29.10.22	
12	Підготувати презентацію, доклад	29.10.22-01.11.22	

Дата видачі завдання 16 вересня 2022 року

Керівник дипломного проекту Халімон Наталія Федорівна
(підпис) (П.І.Б.)

Завдання прийняв до виконання Макарьєв Єгор Олександрович
(підпис) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмне забезпечення системи аналізу та обліку успішності школярів»: 99с., 5 рис., 1 таблиця, 16 використаних джерел, 1 додаток.

Об’єкт дослідження кваліфікаційної роботи – програмне забезпечення систем контролю та обліку в сфері освіти.

Предмет дослідження дипломної роботи – проектування та розробка веб-додатку системи аналізу та обліку успішності школярів.

Мета дослідження – спроектувати та розробити програмне забезпечення системи аналізу та обліку успішності школярів.

Розроблений програмний засіб, що був створений під час виконання кваліфікаційної роботи, рекомендується використовувати для аналізу та обліку успішності школярів у містах України.

Кваліфікаційна робота присвячена актуальній тематиці розробки веб-додатків.

ЗМІСТ

ВСТУП

У сьогоденні програмне забезпечення систем аналізу та обліку успішності школярів розробляють переважно у вигляді веб-додатку через те, що доступ до таких сервісів має бути максимально швидким та простим. Можна виділити три програмних засобів систем аналізу та обліку успішності школярів: система електронних журналів і щоденників «*e-journal.iea.gov.ua*», електронні щоденники та журнали «*e-schools.info*», електронний журнал та електронний щоденник − online система для навчального процесу «*ukrschools.com.ua*». Кожен із виділених сервісів має свій унікальний набір функцій, свої переваги та недоліки. Зазвичай, вони надають наступні можливості користувачеві: створення документів тимчасового зберігання, таких як класний журнал, цифровий аналог щоденника, зручний доступ до розкладу уроків у режимі онлайн, збір освітньої статистики.

Для проектування програмного забезпечення системи аналізу та обліку успішності школярів було обрано мову програмування *Java* із наступними фреймворками: *Spring, Hibernate, Spring Boot, Spring Data, Spring MVC.* Саме даний набір інструментів частіше всього використовуються для розробки *backend* частини на мові *Java.* Розробка проводилася в середовищі *IntelliJ IDEA* з інструментом автоматичної збірки проектів *Maven*. Для тестування було обрано фреймворки *JUnit* та *Mockito*. Була обрана система управління базами даних *MySQL.*

Мова програмування *Java* було обрана через те, що це об’єктно-орієнтована мова програмування із сильною типізацією. Основний принцип даної мови програмування – можливість запуску програми на будь-якому пристрої. Це означає що написаний додаток на *Java* можна запустити на любому пристрої, де встановлене середовище виконання *Java.*

Інтегрована середа розробки *IntelliJ IDEA* була обрана через те, що вона має зручні набори інтегрованих інструменти для рефакторингу коду, що дозволяють розробникам швидко реорганізувати свій код.

Інструмент побудови та управління проектами *Maven* обрано через можливість даного інструменту автоматично завантажувати потрібні бібліотеки залежностей із репозиторію у вигляді *JAR* файлів та додавання їх до проекту.

Фреймворк *Spring* обраний через те, що він представляє собою контейнер впровадження залежностей. Він реалізує принцип інверсії управління (*IoC – Inversion of Control*). Тобто *Spring* бере на себе функцію управління класами розробника та їх залежностями.

ORM фреймворк *Hibernate* був обраний через те, що його ціль зв’язати об’єктно-орієнтоване програмування та реляційну базу даних. *Hibernate* полегшує взаємодію між розробником та базою даних у коді. Він має свою мову запитів – *HQL (Hibernate Query Language).* Її відміна від *SQL* полягає в тому, що в *HQL* запити будуються навколо назв об’єктів, змінних, тоді як в *SQL* вказується назви таблиць, колонок. Також *Hibernate* розуміє наслідування класів, а отже при виконанні запитів будуть враховуватися нащадки.

Фреймворк *Spring Boot* використовувася для полегшення розробки додатків з використанням фреймворка *Spring.* Він містить у собі вбудований локальний сервер *Tomcat*, що надає змогу тестувати веб-додаток не завантажуючи файли проекту на сторонній сервер. Також даний інструмент надає вже готові залежності для проекту, які називаються стартерами. Вони містять у собі ще декілька залежностей, які потрібні для роботи того чи іншого модуля.

Фреймворк *Spring Data* обрано для максимального спрощення взаємодії серверної частини із базою даних. Тобто, він дозволяє уникати *SQL* запитів шляхом декларування методів інтерфейсу репозиторія із зазначеними правилами.

Фреймворк для веб-розробки *Spring MVC* використовувався для реалізації патерна проектування *Model – View – Controller.* Паттерн *MVC* поділяє аспекти програми (логіку введення, бізнес-логіку та логіку *UI*), забезпечуючи вільний зв'язок між ними. Фреймворк містить у собі модуль, що називається диспетчер сервлетів. Даний компонент приймає усі запити що надходять на сервер та розподіляє їх на потрібні обробники – контролери.

Програмне забезпечення, яке розроблено, являє собою веб-додаток, що використовує архітектурний стиль *REST* для обміну інформації між сервером та клієнтом. *REST* (від англ. *Representational State Transfer* – передача репрезентативного стану) – являє собою клієнт-серверну архітектуру, в якій передача інформації між клієнтом та сервером виконується у вигляді *HTTP*-запитів та *HTTP*-відповідей. Тобто клієнт відсилає запит із певною інформацією, на що сервер відсилає назад відповідь із даними, що запросив користувач. Клієнт не має доступу до бази даних, а отже відповідальність за формування запитів до бази даних та обробка отриманих даних лежить повністю на сервері, що підвищує швидкодію програмного забезпечення.

Для реалізації програмного забезпечення системи аналізу та обліку досягнень школярів було спроектовано наступні таблиці бази даних: міста, предмети, школи, класи, вчителі, учні, уроки, оцінки за уроки, олімпіади та оцінки за олмпіади.

Для проектування *frontend* частини програмного забезпечення системи аналізу та обліку досягнень школярів було обрано мову програмування *Java* із використанням фреймворків: *Spring, Spring Boot, Spring MVC, Thymeleaf.* Розробка велася в середовищі *IntelliJ IDEA* із використанням бібліотеки *Lombok* та інструменту автоматичної зборки проекту *Maven.* Для створення графічного оформлення веб-додатку був обраний фреймворк *Bootstrap.*

*С*учасний серверний інструмент *Java* шаблонів для веб-середовища *Thymeleaf* обрано через те, що він здатен обробляти *HTML, XML, CSS, JavaScript* та навіть звичайний текст. Основна ціль *Thymeleaf* – створення зручного способу шаблонізації. *Thymeleaf* дозволяє створювати шаблони відповідно до сучасних стандартів веб-техологій.

Бібліотека *Lombok* використовувалася для скорочення коду в класах і розширення функціональності мови програмування *Java.* Вона дозволяє уникнути написання шаблонного коду, автоматично генеруючи його під час зборки проекту, що економить час розробника. Також *Lombok* робить код більш читабельним.

Фреймворк *Bootstrap* обрано через те, що він представляє собою відкритий та безкоштовний *HTML, CSS, JavaScript* фреймворк, що дозволяє швидко створювати адаптивне графічне оформлення для веб-додатків.

Було розроблено основні модулі *backend* частини проекту: модуль пошуку інформації; модуль додавання, редагування та видалення записів; модуль аналізу та обліку успішності школярів. Були розроблені наступні класи-сервіси: *CityServiceImpl* для роботи з містами*; LessonServiceImpl* для роботи з уроками*; MarkServiceImpl* для роботи з оцінками*; OlympiadMarkServiceImpl* для роботи з оцінками за олімпіади*;* для роботи з олімпіадами*; PupilServiceImpl* для роботи з учнями*; SchoolClassServiceImpl* для роботи з класами*; SchoolServiceImpl* для роботи зі школами.

Було розроблено графічний інтерфейс *frontend* частини веб-додатку. Були створені файли для сторінок аналізу: *analysis-class-results.html, analysis-class-search.html, analysis-pupil-results.html, analysis-pupil-search.html, analysis-school-results.html, analysis-school-search.html;* для роботи з містами: *cities-edit.html, cities-results.html;* для роботи з класами: *class-details.html, class-edit.html;* для роботи з оцінками: *lesson-edit.html, marks-class.html, marks-search.html;* для роботи з олімпіадами: *olympiad-details.html, olympiad-edit.html, olympiad-results.html, olympiad-search.html, olympiadmark-edit.html;* для роботи з учнями: *pupil-details.html, pupil-edit.html, pupil-results.html, pupil-search.html;* для роботи зі школами: *school-details.html, school-edit.html, school-results.html, school-search.html;* для роботи з предметами: *subject-edit.html, subject-results.html;* для роботи з вчителями: *teacher-details.html, teacher-edit.html, teacher-results.html, teacher-search.html.*

Об’єкт дослідження кваліфікаційної роботи – програмне забезпечення систем контролю та обліку в сфері освіти.

Предмет дослідження дипломної роботи – проектування та розробка веб-додатку системи аналізу та обліку успішності школярів.

Мета дослідження – спроектувати програмне забезпечення системи аналізу та обліку успішності школярів.

Дана тема кваліфікаційної роботи є актуальною в умовах тотальної інформатизації сучасного суспільства, один з найважливіших етапів якої є процес впровадження сучасних технологій у комунікативну середу. Розвиток мережевих інформаційних технологій зробило інформаційні ресурси глобальної комп'ютерної мережі потенційно доступними для кожного.

Цифровізація проникла у всі сфери життєдіяльності людини, в тому числі в систему освіти, а отже з'явилися можливості формування електронних класних журналів, учнівських щоденників, розкладу, що дозволяє учасникам освітнього процесу, перш за все, здійснювати контроль успішності учнів, тому тема кваліфікаційної роботи «Програмне забезпечення системи аналізу та обліку успішності школярів» є актуальною.

РОЗДІЛ 1
ОГЛЯД ВЕБ-ДОДАТКІВ СИСТЕМИ АНАЛІЗУ ТА ОБЛІКУ УСПІШНОСТІ ШКОЛЯРІВ І ЗАСОБІВ ДЛЯ ЇХ РОЗРОБКИ

1.1. Програмне забезпечення систем аналізу та обліку успішності школярів

У сьогоденні програмне забезпечення систем аналізу та обліку успішності школярів розробляють переважно у вигляді веб-додатку через те, що доступ до таких сервісів має бути максимально швидким та простим. Серед усіх подібних програмних засобів можна виділити наступні:

– система електронних журналів і щоденників «*e-journal.iea.gov.ua*»;

– електронні щоденники та журнали «*e-schools.info*»;

– електронний журнал та електронний щоденник - *online* система для навчального процесу «*ukrschools.com.ua*».

Державна система електронних журналів і щоденників «*e-journal.iea.gov.ua*» - це програмне забезпечення, яке доволі популярне серед українських шкіл, тому що воно було розроблено за ініціативою Міністерства освіти і науки України та є цілком безкоштовним, що робить його доступним для всіх закладів загальної середньої освіти. Даний сервіс має зручний та простий інтерфейс, але він містить у собі занадто багато налаштувань. Проте, на сайті системи є детальна інструкція для різних типів користувачів, а саме: адміністрація школи, викладачі, учні та батьки. Дане програмне забезпечення має наступні можливості:

– заклади освіти можуть створювати документи тимчасового зберігання, такі як класний журнал;

– учасники освітнього процесу мають доступ до цифрового аналога щоденника;

– зручний доступ до розкладу уроків у режимі онлайн;

– система виконує збір освітньої статистики з використанням даних учнів та педагогів, попередньо деперсоналізуючи їх, тобто кінцева статистика не міститиме в собі жодних імен, прізвищ та іншої приватної інформації;

– дані у системі захищені від несанкціонованого доступу.

Основне направлення електронних щоденників та журналів «*e-schools.info*». є електронні щоденники та журнали. Окрім цього, дане програмне забезпечення надає можливість безкоштовно створити окремий сайт для школи. Графічне оформлення системи виглядає застаріло. Даний сервіс пропонує закладам загальної середньої освіти наступні можливості:

– електронні журнали;

– електронні щоденники;

– можливість управляти гуртками у школі;

– онлайн розклад;

– файловий архів, який може містити в собі корисні учбові матеріали, накази, файли конференцій тощо;

– фотоальбоми школи;

– новини школи;

– гостьова книга для відгуків;

– зворотній зв’язок із адміністрацією школи.

Електронний журнал та електронний щоденник – *online* система для навчального процесу «*ukrschools.com.ua*» має доволі простий та зручний інтерфейс, що надає можливість користувачеві швидко зрозуміти як користуватися даною системою. Проте, даний сервіс є платним, що є основним недоліком у порівнянні з іншими програмними продуктами. Це програмне забезпечення надає наступні можливості користувачам:

– журнал оцінок;

– обмін навчальними матеріалами;

– електронні домашні завдання;

– календар, який дозволяє формувати розклад уроків та інших заходів;

– перегляд новин;

– аналітика успішності навчання учнів, ефективності навчальних курсів, отримання статистики на всіх етапах навчального процесу.

– обмін повідомленнями між вчителем та учнями, або батьками;

– рейтинг учнів у школі на основі їх успішності.

1.2. Засоби *backend* розробки веб-додатків

Backend – це розробка бізнес-логіки програмного забезпечення, таких як сайт або веб-додаток. *Backend* забезпечує взаємодію між користувачем та внутрішніми даними, які вже потім відображаються користувачеві.

На сьогоднішній день у веб-розробці найпопулярнішими засобами для розробки *backend* частини на мові програмування *Java* додатку є:

– фреймворк *Spring Boot*;

– платформа *Node.js*;

– фреймворк *Django*.

Фреймворк *Spring Boot* призначений для розробки *backend*-додатків. Він заснований на мові програмування *Java* та часто використовується для побудови мікросервісної архітектури додатку. Даний засіб полегшує створення програмних продуктів, які використовують *Spring.* Крім цього *Spring Boot* має такі переваги:

– *Spring Boot* облегшує створення та підтримку самостійних *Spring*-додатків. Даний фреймворк полегшує процес конфігурування та роботу над додатками та дозволяє їх швидко запускати за допомогою команди *java -jar;*

– *Spring Boot* має вбудований аналізатор помилок, що вкаже на проблеми ще на етапі розробки додатку;

– даний фреймворк вже містить в собі вбудований локальний сервер, такий як *Tomcat* або *Jetti*. Це означає що розробникам на *Spring Boot* не потрібно орендувати зовнішні сервери для розгортання *war(Web Archive)* файлів;

– у можливості *Spring Boot* входить автоматичне конфігурування *Spring,* що економить час розробникам;

– *Spring Boot* добре інтегрується з іншими фреймворками;

– при використанні *Spring Boot* немає потреби використання *XML*-конфігурації або засобів для генерації коду[1].

Spring Boot також має декілька недоліків:

– відсутність контролю за непотрібними залежностями. Через це, розмір розгортуваного проекту може бути занадто високим;

– занадто складний процес переведення застарілих додатків із *Spring* на *Spring Boot;*

– даний фреймворк не підходить для розробки великих проектів із монолітною архітектурою, так як масштабування та підтримка такого продукту через деякий час стане занадто дорогою та трудомісткою.

Основною причиною використання *Spring Boot* при розробці веб-додатків є наявність ефективних інструментів для масштабування та підтримки проектів із мікросервісною архітектурою, що дозволяє створювати доволі великі та гнучкі програмні продукти[2].

Node.js являє собою серверну платформу, яка є частиною набору технологій, що охоплюють всі потреби веб-розробки. Він заснований на *JavaScript* та використовує *JavaScript*-двигун *V*8, який також використовується в браузері *Google Chrome* та інших браузерах. На базі платформи *Node.js* створено багато фреймворків, включаючи такі популярні розробки як *Express*. *Node.js* має наступні переваги:

– *Node.js* дозволив розробникам створювати *backend*-проекти на базі мови програмування *JavaScript,* а отже, усі напрацювання та бібліотеки, які використовувалися у *frontend* розробці можна застосовувати для створення бізнес-логіки додатку;

– код написаний на *JavaScript* у порівнянні з іншими мовами програмування є доволі компактним та високопродуктивним, а отже він підходить до проектів, у яких важлива швидкодія коду;

– код клієнтських та серверних частин проекту легше підтримувати в узгодженому стані при подальшій розробці, так як вони написані на одній мові програмування;

– один той самий код можна використовувати як на серверній, так і на клієнтській частинах програми;

– *Node.js* має підтримку модулів, а отже розробники можуть із легкістю використовувати наробки інших програмістів у своєму коді для полегшення створення продукту;

– дана платформа та фреймворки, засновані на ній, є невибагливими до ресурсів та легко масштабуються;

– *Node.js* добре підходить для розробки додатків із мікросервісною архітектурою через систему модулів, що пропонує дана платформа;

– *Node.js* є опенсорсним проектом, тобто його код є у відкритому доступі, а отже розробник має можливість ознайомитись із поведінкою та логікою даної платформи.

Не дивлячись на великий потенціал даної платформи, вона все одно має свої недоліки:

– одним із головним недоліком *Node.js* є низька продуктивність при роботі з трудомісткими обчислювальними задачами;

– залежність від зворотнього виклику. Тобто, після виконання кожної задачі викликаються певні функції, що ускладнює програму та розуміння коду;

– *Node.js* містить у репозиторії багато модулів низької якості, які не мають належної документації чи не працюють як повинні.

Платформа *Node.js* є ефективним інструментом у *backend*-розробці. Мова програмування *JavaScrip,* у порівнянні з іншими мовами, доволі проста, проте у практиці вона не відстає від інших інструментів[1].

Фреймворк *Django* написаний на мові програмування *Python,* а отже, через популярність даної мови програмування, багато розробників користуються саме ним під час розробки *backend*-додатків. Крім цього, даний засіб має наступні переваги:

– так як *Django* написаний на мові програмування *Python,* він дозволяє створювати динамічні веб-додатки, так як мова *Python* є доволі простою;

– фреймворк підтримує патерн проектування *MVC (Model – View - Controller),* що допомагає розробникам розділяти бізнес-логіку та інтерфейс користувача;

– фреймворк обладнаний багатьма інструментами для захисту веб-додатку від різних атак, або несанкціонованого доступу. Наприклад, він надає розробнику інструменти для впровадження аутентифікації, авторизації, захист від *SQL* ін'єкцій тощо;

– проекти написані на *Django* є доволі компактними;

– *Django* надає ефективні інструменти для масштабування проекту;

– даний проект є крос-платформовим, тобто він добре працює на різних операційних системах та підтримує взаємодію із різними базами даних.

Але, *Django* також має наступні недоліки:

– *Django ORM* (*Object Relational Mapping*) є доволі застарілим та поступається *ORM* технологіям інших фреймворків;

– *Django* розвивається занадто повільно через те, що він має монолітну архітектуру та нові його версії повинні мати зворотню сумісність;

Фреймворк *Django* через те, що він написаний на мові програмування *Python,* є надійним та швидким інструментом у веб-розробці. Офіційна документація та учбові матеріали даного фреймворку є одними з найкращими серед конкурентних проектів[3].

1.3. Засоби *frontend* розробки веб-додатків

Frontend – це розробка інтерфейсу користувача, а саме зовнішнього вигляду веб-додатку та правильне відображення даних, отриманих із *backend* частини програми.

У даний момент часу серед *frontend* розробників користуються попитом наступні інструменти:

– фреймворк *Bootstrap;*

– фреймворк *ReactJS;*

– фреймворк *Vue.js*.

Фреймворк *Bootstrap* є одним із самих популярних інструментів при розробці сайтів та веб-додатків. Він включає в себе багато різних компонентів для *frontend* розробки, а саме: типографіку, веб-форми, блоки навігації, кнопки, таблиці тощо. Також *Bootstrap* має інші переваги, а саме:

– сайти, що використовують *Bootstrap* коректно відображаються у всіх сучасних браузерах;

– *Bootstrap* надає можливість розробити адаптивний сайт, який буде правильно відображатися на екранах приладів різних розмірів, буцімто персональний комп'ютер, ноутбук, планшет, смартфон тощо.

– *Bootstrap* доволі простий у розробці та у ньому легко розібратися через велику кількість уроків та інструкцій;

– код у даному фреймворку є простим та зрозумілим, а отже інший розробник з легкістю зможе розібратися у чужому коді, що значно спрощує роботу в команді;

– елементи *Bootstrap* виглядають гармонійно між собою, а отже розробник із легкістю зможе створювати сайти та сторінки в єдиному стилі.

Проте, не дивлячись на усі перераховані переваги фреймворку, він має свої недоліки:

– сайти, розроблені за допомогою *Bootstrap,* мають схоже графічне оформлення, а саме: однакову структуру, кнопки, шрифти тощо;

– *Bootstrap* є недостатньо гнучким для впровадження деяких нестандартних ідей;

– *Bootstrap* містить у собі переважно сучасні рішення, а отже сайт із використанням даного фреймворку може некоректно відображатися у старих браузерах.

Фреймворк *Bootstrap* дозволяє швидко створити сайт із стандартних заготовлених блоків. У цьому полягає його головна перевага і недолік одночасно, так як розробник може швидко створити якісний сайт, втративши його індивідуальність. Але, маючи навички у *CSS* та *Bootstrap* розробник зможе

модифікувати зовнішній вигляд заготовлених блоків та отримати сайт, дизайн якого буде унікальним[4].

Фреймворк *ReactJS* підходить для розробки великих веб-додатків, у яких дані можуть мінятися на регулярній основі. Він має наступні переваги:

– синтаксис фреймворку *ReactJS* простий, а отже розробник за короткий проміжок часу зможе освоїти дану технологію;

– високий рівень гнучкості при розробці;

– віртуальна *DOM* (*Document Object Model*), що дозволить впорядкувати документи форматів *HTML*, *XML* або *XHTML* у дерево, що більше підходить веб-браузерам при аналізі елементів веб-сторінки;

– *ReactJS* може працювати при доволі великих навантаженнях;

– зв'язування даних від більших до менших, що дозволяє уникати ситуацій коли дочірній елемент впливає на батьківський;

– додатки займають доволі мало пам'яті;

– дуже проста міграція між версіями фреймворка.

Проте, *ReactJS* має наступні недоліки:

– замало офіційної документації через те, що розробка фреймворку ведеться занадто швидко, тому розробникам доводиться вносити індивідуальні зміни до неї без систематичного підходу;

– *ReactJS* не має точного призначення;

– *ReactJS* потребує глибоких знань з приводу того, як інтегрувати інтерфейс користувача у структуру *MVC*.

ReactJS більше підійде тим розробникам, які шукають бібліотеки для розробки, а не цілий фреймворк. Проте, він надає зручні та ефективні інструменти для роботи з *frontend* частиною проекту[5].

Фреймворк *Vue.js* використовується переважно для створення адаптивних інтерфейсів користувача і складних односторінкових сайтів. Даний інструмент має такі переваги:

– доповнений *HTML,* що допоможе оптимізувати обробку деяких *HTML* блоків;

– хороша документація, що дозволить швидко навчитися користуватися фреймворком;

– *Vue.js* є доволі адаптивним, він пропонує розробникам інструменти для швидкого переходу з інших фреймворків;

– *Vue.js* має хорошу інтеграцію, тобто його можна використовувати як для розробки односторінкових сайтів, так і для створення складних та великих веб-додатків;

– *Vue.js* дозволяє створювати великі шаблони та неодноразово використовувати їх без трати великої кількості часу;

– *Vue.js* важить близько 20 КБ, при цьому зберігаючи свою швидкість та гнучкість.

Недоліки *Vue.js:*

– даний фреймворк знаходиться на етапі розробки, а отже ще не створена спільнота розробників;

– інколи можуть траплятися проблеми з інтеграцією у великі проекти, для яких на сьогоднішній день не було знайдено рішень;

– відсутність повної документації англійською мовою.

Фреймворк *Vue.js* підійде для застарілих проектів, яким потрібен новий сучасний двигун[6].

1.4. Висновки до розділу

У даному розділі було зроблено огляд існуючого програмного забезпечення систем аналізу та обліку успішності школярів, а саме було розглянуто сайти державної системи електронних журналів і щоденників «*e-journal.iea.gov.ua*», електронні щоденники та журнали «*e-schools.info*», електронний журнал та електронний щоденник – *online* систему для навчального процесу «*ukrschools.com.ua*». Було проаналізовано основні функції сайтів, їх дизайн, основні направлення, їх переваги та недоліки.

Було розглянуто найпопулярніші засоби для розробки *backend* частини веб-додатків, а саме: фреймворк *Spring Boot,* платформам *Node.js* та фреймворк *Django.* Було проаналізовано основні напрямки використання цих інструментів, їх переваги та недоліки.

Фреймворк *Spring Boot* в основному використовується для розробки великих веб-додатків із мікросервісною архітектурою. Він також має всі переваги мови програмування *Java*, так як даний інструмент написаний цією мовою.

Платформа *Node.js* за рахунок системи модулів може використовуватися для побудови веб-додатків із мікросервісної архітектури. За рахунок мови програмування *JavaScript* код є компактним.

Фреймворк *Django* заснований на мові програмування *Python,* яка має чітку документацію та велику спільноту розробників. Також код із використанням цього інструменту є компактним та швидко виконується.

Було розглянуто популярні інструменти для *frontend* розробки веб-додатків, а саме: фреймворк *Bootstrap,* фреймворк *ReactJS,* фреймворк *Vue.js.* Було виявлено основні переваги та недоліки цих інструментів.

Фреймворк *Bootstrap* підійде для створення якісного та практичного сайту у максимально короткий проміжок часу, нехтуючи його унікальністю. Проте, якщо розробник має достатні навички, він може вирішити проблему з індивідуальністю сайту. *ReactJS* більше підійде тим розробникам, які шукають бібліотеки для розробки, а не цілий фреймворк. Фреймворк *Vue.js* підійде для застарілих проектів, яким потрібен новий сучасний двигун.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ АНАЛІЗУ ТА ОБЛІКУ УСПІШНОСТІ ШКОЛЯРІВ

2.1. Проектування функцій та основних класів *backend* частини

Для проектування програмного забезпечення системи аналізу та обліку успішності школярів було обрано мову програмування *Java* із наступними фреймворками: *Spring, Hibernate, Spring Boot, Spring Data, Spring MVC.* Саме даний набір інструментів частіше всього використовуються для розробки *backend* частини на мові *Java.* Розробка проводилася в середовищі *IntelliJ IDEA* з інструментом автоматичної збірки проектів *Maven*. Для тестування було обрано фреймворки *JUnit* та *Mockito*. Була обрана система управління базами даних *MySQL.*

Java – об'єктно-орієнтована мова програмування із сильною типізацією. Основний принцип даної мови програмування – можливість запуску програми на будь-якому пристрої. Це означає що написаний додаток на *Java* можна запустити на любому пристрої, де встановлена середа виконання *Java – JRE (Java Runtime Environment).* Даний принцип вирішується за допомогою переведення *Java* коду у байт-код у ході виконання програми. Це досягається за допомогою віртуальної машини *Java – JVM (Java Virtual Machine)*, яка вже входить до *JRE*. Однією з особливостей даної мови програмування є механізм управління пам'яттю, який називається збиральником сміття (*garbage collector*). Тобто, розробник створює об'єкти, а коли об'єкт більше не використовується, збиральник сміття автоматично видаляє його з пам'яті. Отже, на відміну від *C*-подібних мов програмування, розробник не відповідає за визволення пам'яті з-під об'єктів, за нього це робить *JRE.*

Java була розроблена компанією *Sun Microsystems* на початку 90-х років. Дана мова програмування розроблялася для програмування побутової техніки, проте

одразу після виходу першої версії *Java* її почали використовувати розробники серверного та клієнтського програмного забезпечення[7].

IntelliJ IDEA – інтегрована середа розробки для таких мов програмування як *Java, JavaScript, Python, Kotlin* тощо. Розроблена компанією *JetBrains*. Має зручні набори інтегрованих інструменти для рефакторингу коду, що дозволяють розробникам швидко реорганізувати свій код. Дизайн середи розробки орієнтований на підвищення ефективності роботи програмістів, так як *IntelliJ IDEA* здатен виконувати рутинні операції.

Maven – інструмент побудови та управління проектами на *Java*, що використовують ті чи інші фреймворки. *Maven* контролюється файлом *Project Object Model (POM)*, у який додаються залежності проекту. Даний інструмент автоматично завантажує потрібні бібліотеки залежностей із репозиторію у вигляді *JAR* файлів та додає їх до проекту[8].

Spring – фреймворк, що представляє собою контейнер впровадження залежностей. Він реалізує принцип інверсії управління (*IoC – Inversion of Control*). Тобто *Spring* бере на себе функцію управління класами розробника та їх залежностями. Це називається контекст додатку (*Application context*)[12].

Hibernate – ORM фреймворк, ціль якого зв’язати об’єктно-орієнтоване програмування та реляційну базу даних. *Hibernate* полегшує взаємодію між розробником та базою даних у коді. Він має свою мову запитів – *HQL (Hibernate Query Language).* Її відміна від *SQL* полягає в тому, що в *HQL* запити будуються навколо назв об’єктів, змінних, тоді як в *SQL* вказується назви таблиць, колонок. Також *Hibernate* розуміє наслідування класів, а отже при виконанні запитів будуть враховуватися наслідники[9].

ORM – об’єктно-реляційна модель, що описує зв’язки між програмними об’єктами та записами в базі даних.

Spring Boot – фреймворк для полегшення розробки додатків з використанням фреймворка *Spring.* Він містить у собі вбудований локальний сервер *Tomcat*, що надає змогу тестувати веб-додаток не завантажуючи файли проекту на сторонній сервер. Також даний інструмент надає вже готові залежності для проекту, які

називаються стартерами. Вони містять у собі ще декілька залежностей, потрібні для роботи того чи іншого модуля. Це робить *POM* файл проекту більш компактним[2].

Spring Data – фреймворк для максимального спрощення взаємодії серверної частини із базою даних. Тобто, він дозволяє уникати *SQL* запитів шляхом об’явлення методів інтерфейсу репозиторія із зазначеними правилами. Проте, функції даного інструменту обмежені, а отже він не здатен виконувати більш складні запити, а отже повністю уникнути використання *SQL* запитів неможливо[13].

Spring MVC – фреймворк для веб-розробки, а саме для реалізації патерна проектування *Model – View – Controller*. Паттерн *MVC* поділяє аспекти програми (логіку введення, бізнес-логіку та логіку *UI*), забезпечуючи вільний зв'язок між ними. Фреймворк містить у собі модуль, що називається диспетчер сервлетів. Даний компонент приймає усі запити що надходять на сервер та розподіляє їх на потрібні обробники – контролери.

Програмне забезпечення являє собою веб-додаток, що використовує архітектурний стиль *REST* для обміну інформації між сервером та клієнтом. *REST* (від англ. *Representational State Transfer* – передача репрезентативного стану) – являє собою клієнт-серверну архітектуру, в якій передача інформації між клієнтом та сервером виконується у вигляді *HTTP*-запитів та *HTTP*-відповідей. Тобто клієнт відсилає запит із певною інформацією, на що сервер відсилає назад відповідь із даними, що запросив користувач. Клієнт не має доступу до бази даних, а отже відповідальність за формування запитів до бази даних та обробка отриманих даних лежить повністю на сервері, що підвищує швидкодію програмного забезпечення[14].

Для реалізації програмного забезпечення системи аналізу та обліку досягнень школярів було спроектовано наступні таблиці бази даних: міста, предмети, школи, класи, вчителі, учні, уроки, оцінки за уроки, олімпіади та оцінки за олмпіади.

Таблиця міст називається *cities* та має наступні колонки: *id* – ідентифікаційний номер міста; *name* – назва міста.

Таблиця предметів називається *subjects* та має такі колонки: *id* – ідентифікаційний номер предмету; *name* – назва предмету.

Таблиця шкіл називається *schools* та має наступні колонки: *id* – ідентифікаційний номер школи; *name* – назва школи; address – адреса школи; *city_id* – зовнішній ключ, що вказує на поле *id* у таблиці міст *cities*.

Таблиця класів називається *classes* та має такі колонки: *id* – ідентифікаційний номер класу; *name* – назва класу; *school_id* – зовнішній ключ, що вказує на поле *id* у таблиці шкіл *schoools*.

Таблиця вчителів називається *teachers* та має такі колонки: *id* – ідентифікаційний номер вчителя; *firstName* – ім'я вчителя; *lastName* – прізвище вчителя; *middleName* – по-батькові; *school_id* – зовнішній ключ, що вказує на поле *id* у таблиці шкіл *schoools*; *subject_id* – зовнішній ключ, що вказує на поле *id* у таблиці предметів *subjects*.

Таблиця учнів називається *pupils* та має такі колонки: *id* – ідентифікаційний номер учня; *firstName* – ім'я учня; *lastName* – прізвище учня; *middleName* – по-батькові; *class_id* – зовнішній ключ, що вказує на поле *id* у таблиці шкіл *classes*.

Таблиця занять називається *lessons* та має такі колонки: *id* – ідентифікаційний номер заняття; *date_held* – дата проведення заняття; *class_id* – зовнішній ключ, що вказує на поле *id* у таблиці шкіл *classes*; *teacher_id* – зовнішній ключ, що вказує на поле *id* у таблиці вчителів *teachers*.

Таблиця оцінок за заняття називається *marks* та має такі колонки: *id* – ідентифікаційний номер оцінки; *mark* – оцінка за урок; *pupil_id* – зовнішній ключ, що вказує на поле *id* у таблиці учнів *pupils*; *lesson_id* – зовнішній ключ, що вказує на поле *id* у таблиці уроків *lessons*.

Таблиця олімпіад називається *olympiads* та має такі колонки: *id* – ідентифікаційний номер олімпіади; *date_held* – дата проведення олімпіади; *name* – назва олімпіади; *subject_id* – зовнішній ключ, що вказує на поле *id* у таблиці предметів *subjects*.

Таблиця оцінок за олімпіади називається *olympiad_marks* та має такі колонки: *id* – ідентифікаційний номер оцінки за олімпіаду; *mark* – оцінка за олімпіаду;

olympiad_id – зовнішній ключ, що вказує на поле *id* у таблиці олімпіад *olympiads*; *pupil_id* – зовнішній ключ, що вказує на поле *id* у таблиці учнів *pupils*; *position* – місце в рейтингу олімпіади.

Модель бази даних зображена на рис. 2.1.

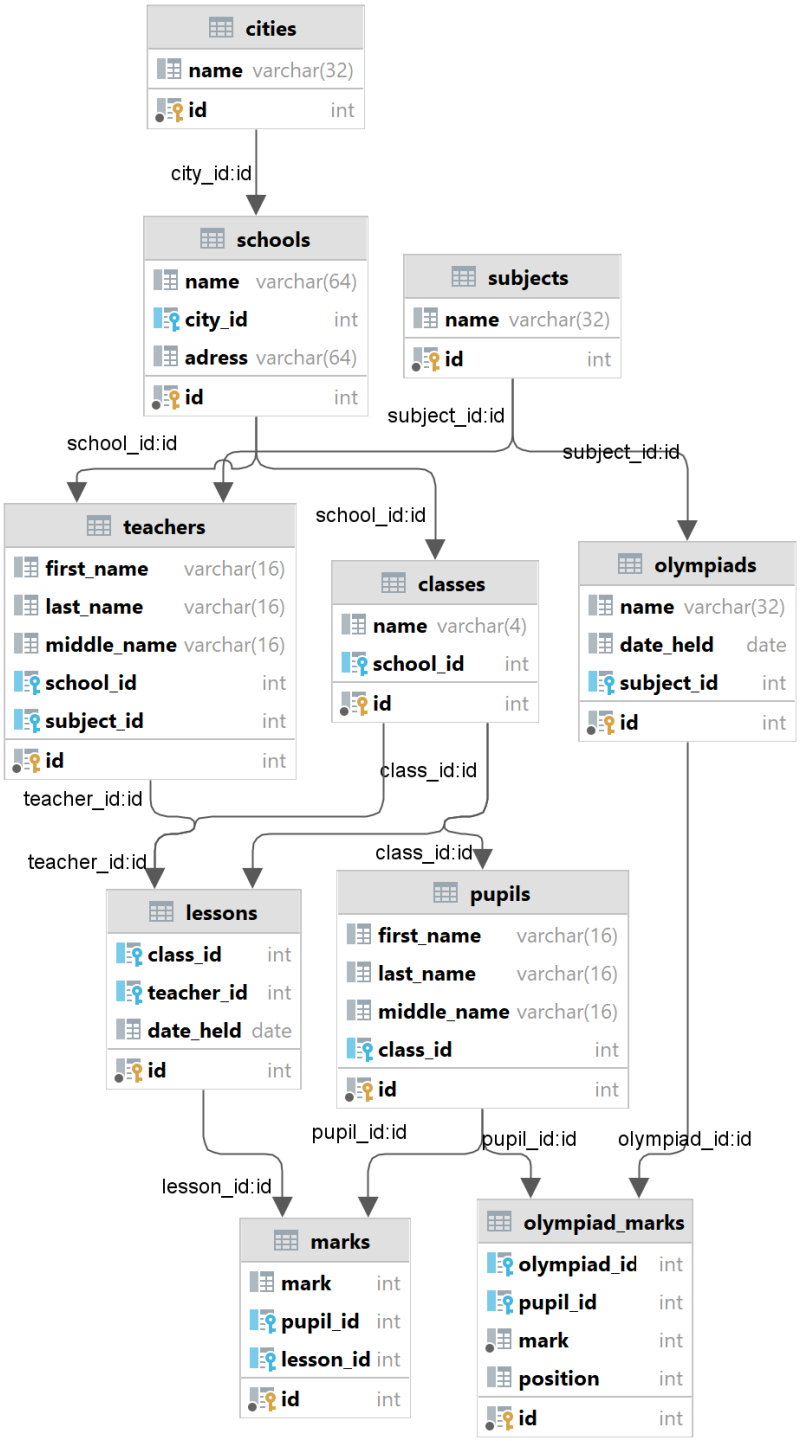

Рис. 2.1. Модель бази даних

Основними функціями *backend* частини додатку системи аналізу та обліку успішності школярів є робота з таблицями, а саме можливість їх перегляду, редагування, додавання та видалення.

Так як веб-додаток використовує архітектурний стиль *REST*, основні його функції залежні від процесу отримання класами-контролерами *HTTP* запитів від клієнта, що містять в собі певні дані. Потім йде обробка їх класами-сервісами. Далі виконується звернення до бази даних класами-репозиторіями. Отримані результати знову обробляються класами-сервісами та повертаються до класів-контролерів, які врешті-решт відправляють *HTTP* відповідь клієнту у форматі *JSON*.

HTTP запити складаються з:

– *URL* посилання до серверу;

– заголовків, що несуть додаткову інформацію. Наприклад, браузер клієнта, дату створення запиту тощо;

– тіла запиту, у якому в певному форматі можуть утримуватися інформація про якій-небудь об’єкт. Наприклад, при відправленні запит на сервер із додаванням запису до бази даних, тіло запиту може містити в собі інформацію про стан об’єкту в форматі *JSON*, або *XML;*

– метод *HTTP* запиту. Він означає який тип запиту відправляється на сервер. Найбільш вживаними є методи: *GET* – означає що клієнт хоче тільки отримати дані від серверу; *POST* – означає що клієнт відправляє певні дані на сервер з метою їх додавання в базу даних; *PUT* – означає що клієнт відправляє певні дані на сервер з метою зміни одної, або декількох записів у базі даних; *DELETE* – означає що клієнт бажає видалити певні записи із бази даних.

У свою чергу, *HTTP* відповіді мають схожу структуру. Вони містять у собі заголовки, тіло відповіді, проте, на відміну від *HTTP* запитів, у них відсутня інформація про *URL* посилання та метод. Але, у них міститься інформація про статус виконання *HTTP* запиту. Статус має числове представлення та складається з трьох цифр, перша з яких визначає категорію відповіді. Існують п’ять видів статусу: 1xx – інформативний; 2xx – запит було успішно оброблено; 3xx – статус

перенаправлення; 4xx – помилка на стороні клієнта; 5xx – помилка на стороні сервера.

Класи-контролери – класи, основне призначення яких є прийняття вхідних *HTTP* запитів та після обробки повернення клієнту *HTTP* відповіді. Інформація, що надходить разом із запитом у вигляді *JSON* за допомогою інструменту *Jackson,* автоматично переводиться у *Java* об'єкт, потім отримані дані відправляються класу-сервісу для подальшої їх обробки. Коли клас-сервіс поверне потрібні дані, клас-контролер за допомогою *Jackson* переведе *Java* об'єкт у *JSON* формат, сформує *HTTP* відповідь та відправить її назад клієнтові.

Jackson – бібліотека, що призначена для переведення *Java* об'єктів у такі формати як *JSON, XML, CSV* та навпаки. Вона за замовченням підключається разом із *Spring MVC.*

JSON (JavaScript Object Notation) – текстовий формат для представлення структурованих даних на основі синтаксису об'єктів *JavaScript.* Він представляє собою записи у вигляді «ключ-значення» та допускає вкладені об'єкти та масиви даних.

Класи-сервіси – класи, у яких знаходиться основна бізнес-логіка. Вони є зв'язуючим ланом між класами-контролерами та класами-репозиторіями. Коли клас-контролер отримує запит, він передає класу-сервісу дані, які потрібні для звернення до бази даних. Після цього, клас-сервіс відправляє класу-репозиторію отримані дані та чекає, коли він поверне потрібні дані. Клас-сервіс обробляє їх згідно бізнес-логіці та відправляє назад до класу-контролеру.

Класи-репозиторії – класи, які напряму зв'язуються з базою даних. Такі класи отримують від класів-сервісів певні дані, потрібні для взаємодії з базою даних, на їх основі вони формують *SQL* запити та відправляють їх до бази даних. Згодом, вони отримують відповідь, яку відправляють назад класам-сервісам.

Схема роботи *REST* додатку представлена на рис. 2.2.

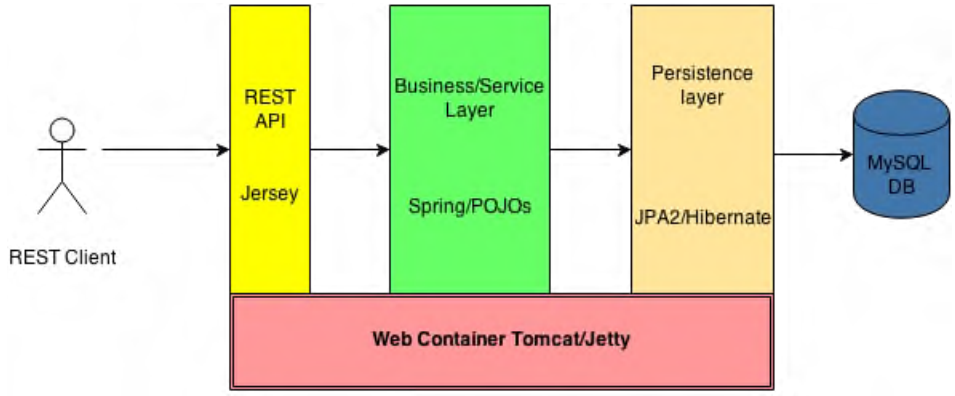

Рис 2.2. Схема роботи *REST* додатку

Так як проект використовує *ORM* фреймворки, для кожної таблиці бази даних повинен бути створений *Entity*-клас. *Entity*-клас – це *POJO* клас, який представляє таблицю бази даних у вигляді *Java* класу. Тобто, він має містити всі колонки таблиці у вигляді полів. Саме з ним ведеться робота в *Java* коді проекту.

POJO (Plain Old Java Object) – Java клас, який складається з полів, одного або декількох конструкторів та методів, що дозволяють отримувати стан об'єкту - гетерів, або змінювати його – сетерів.

Для кожної таблиці потрібні свої класи-контролери, класи-сервіси та класи-репозиторії. Так як саме за допомогою них йде робота з базою даних.

Для таблиці міст *cities* створено *Entity*-клас *City*, клас-контролер *CityRestController*, клас-сервіс *CityService* та клас-репозиторій *CityRepository.*

Клас-контролер *CityRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/cities».* Він має наступні методи: отримання міста за *id* - *getCityById*; отримання всіх міст – *getAllCities;* додавання нового міста – *addNewCity;* редагування певного міста – *updateCity;* видалення певного міста – *deleteCity.*

Клас-сервіс *CityService* містить основну бізнес-логіку що пов'язана із взаємодією з містами. В ньому містяться такі методи: пошук міста за *id* – *findById;* пошук усіх міст – *findAll*; додавання нового міста – *addNewCity;* редагування певного міста – *updateCity;* видалення певного міста – *deleteCity.*

Клас-репозиторій *CityRepository* взаємодіє з таблицею міст *cities.* Він має тільки стандартні методи для роботи з базою даних.

Для таблиці уроків *lessons* створено *Entity*-клас *Lesson,* клас-контролер *LessonRestController,* клас-сервіс *LessonService* та клас-репозиторій *LessonRepository.*

Клас-контролер *LessonRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/lessons».* Він має наступні методи: отримання уроку за *id* - *getLessonById*; отримання всіх уроків за *id* предмету та *id* класу *getAllLessonsBySubjectIdAndSchoolClassId;* отримання всіх уроків за *id* вчителя *getAllLessonsByTeacherId;* додавання нового уроку – *addNewLesson;* редагування певного уроку – *updateLesson;* видалення певного уроку – *deleteLesson.*

Клас-сервіс *LessonService* містить основну бізнес-логіку, що пов'язана із взаємодією з уроками. В ньому містяться такі методи: пошук уроку за *id* – *findById;* отримання всіх уроків за *id* предмету та *id* класу *findAllByTeacher_Subject_IdAndSchoolClass_Id*; отримання всіх уроків за *id* вчителя *findAllByTeacher_Id;* додавання нового уроку – *addNewLesson;* редагування певного уроку – *updateLesson;* видалення певного уроку – *deleteLesson.*

Клас-репозиторій *LessonRepository* взаємодіє з таблицею уроків *lessons.* Крім стандартних методів для роботи з базою даних він має такі методи: знайти всі уроки за *id* предмету та *id* класу та відсортувати за датою проведення уроку – *findAllByTeacher_Subject_IdAndSchoolClass_IdOrderByLessonDate;* знайти всі уроки за *id* вчителя та відсортувати за датою проведення уроку *findAllByTeacher_IdOrderByLessonDate.*

Для таблиці оцінок *marks* створено *Entity*-клас *Mark,* клас-контролер *MarkRestController,* клас-сервіс *MarkService* та клас-репозиторій *MarkRepository.*

Клас-контролер *MarkRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/marks».* Він має наступні методи: отримання усіх оцінок за *id* учня – *getAllMarksByPupilId;* отримання усіх оцінок за *id* предмету та *id* класу – *getAllMarksBySubjectIdAndSchoolClassId;* отримання усіх оцінок за *id* уроку – *getAllMarksByLessonId;* додавання нової оцінки – *addNewMark;* додавання декількох нових оцінок – *addNewMarks;* редагування певної оцінки – *updateMark;* видалення певної оцінки – *deleteMark.*

Клас-сервіс *MarkService* містить основну бізнес-логіку, що пов'язана із взаємодією з оцінками. В ньому містяться такі методи: отримання усіх оцінок за *id* учня – *findAllByPupil_Id;* отримання усіх оцінок за *id* предмету та *id* класу – *findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id;* отримання усіх оцінок за *id* уроку – *findAllByLesson_Id;* додавання нової оцінки – *addNewMark;* додавання декількох нових оцінок – *addNewMarks;* редагування певної оцінки – *updateMark;* видалення певної оцінки – *deleteMark.*

Клас-репозиторій *MarkRepository* взаємодіє з таблицею оцінок *marks.* Крім стандартних методів для роботи з базою даних він має такі методи: обчислити середні оцінки всіх шкіл за *id* міста – *countSchoolAverageMarkByCity_Id;* обчислити середні оцінки всіх шкіл– *countSchoolAverageMark;* обчислити середні оцінки всіх класів за *id* школи – *countSchoolClassAverageMarkBySchool_Id;* обчислити середні оцінки всіх учнів за *id* класу – *countPupilAverageMarkBySchoolClass_Id;* знайти всі оцінки за *id* предмету та *id* класу *findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id;* знайти всі оцінки за *id* предмету та *id* учня *findAllByLesson_Teacher_Subject_IdAndPupil_Id;* знайти всі оцінки за *id* предмету та *findAllByLesson_Id.*

Для таблиці оцінок за олімпіади *olympiad_marks* створено *Entity*-клас *OlympiadMark,* клас-контролер *OlympiadMarkRestController,* клас-сервіс *OlympiadMarkService* та клас-репозиторій *OlympiadMarkRepository.*

Клас-контролер *OlympiadMarkRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/olympiadmarks».* Він має наступні методи: отримання усіх оцінок за *id* учня – *getAllOlympiadMarksByPupilId;* отримання оцінки за *id* – *getOlympiadMarkById;* отримання усіх оцінок за *id* олімпіади – *getAllOlympiadMarksByOlympiadId;* додавання нової оцінки – *addNewOlympiadMark;* редагування певної оцінки – *updateOlympiadMark;* видалення певної оцінки – *deleteOlympiadMark.*

Клас-сервіс *OlympiadMarkService* містить основну бізнес-логіку, що пов'язана із взаємодією з оцінками за олімпіади. В ньому містяться такі методи: отримання усіх оцінок за *id* учня – *findAllByPupil_Id;* отримання оцінки за *id* – *findById;*

отримання усіх оцінок за *id* олімпіади – *findAllByOlympiad_Id;* додавання нової оцінки – *addNewOlympiadMark;* редагування певної оцінки – *updateOlympiadMark;* видалення певної оцінки – *deleteOlympiadMark.*

Клас-репозиторій Olympiad*MarkRepository* взаємодіє з таблицею оцінок *olympiad_marks.* Крім стандартних методів для роботи з базою даних він має такі методи: обчислити місця учасників олімпіади – *calculatePositions*; обчислити середні оцінки всіх шкіл за *id* міста – *countSchoolAverageOlympiadMarkByCity_Id;* обчислити середні оцінки всіх шкіл – *countSchoolAverageOlympiadMark;* обчислити середні оцінки всіх класів за *id* школи – *countSchoolClassAverageOlympiadMarkBySchool_Id;* обчислити середні оцінки всіх учнів за *id* класу – *countPupilAverageOlympiadMarkBySchoolClass_Id;* знайти всі оцінки за *id* учнів *findAllByPupil_Id;* знайти всі оцінки за *id* олімпіади *findAllByOlympiad_Id.*

Для таблиці олімпіад *olympiads* створено *Entity*-клас *Olympiad,* клас-контролер *OlympiadRestController,* клас-сервіс *OlympiadService* та клас-репозиторій *OlympiadRepository.*

Клас-контролер *OlympiadRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/olympiads».* Він має наступні методи: отримання інформації про олімпіаду за *id* - *getOlympiadById*; отримання інформації про олімпіаду за декілька *id* – *getOlympiadsByIds;* отримання інформації про всі олімпіади – *getAllOlympiads;* пошук інформації про олімпіад з одної дати по іншу – *getAllOlympiadsBetweenDates;* додавання нової олімпіади – *addNewOlympiad;* редагування певної олімпіади – *updateOlympiad;* видалення певної олімпіади – *deleteOlympiad.*

Клас-сервіс *OlympiadService* містить основну бізнес-логіку, що пов’язана із взаємодією з олімпіадами. В ньому містяться такі методи: отримання олімпіади за *id* - *findById*; отримання олімпіад за декілька *id* – *findAllById;* отримання всіх олімпіад - *findAll;* пошук олімпіад з одної дати по іншу – *findAllByDateBetween;* додавання нової олімпіади – *addNewOlympiad;* редагування певної олімпіади – *updateOlympiad;* видалення певної олімпіади – *deleteOlympiad.*

Клас-репозиторій *OlympiadRepository* взаємодіє з таблицею олімпіад *olympiads.* Крім стандартних методів для роботи з базою даних він має метод для знаходження олімпіад між датами – *findAllByDateBetween.*

Для таблиці учнів *pupils* створено *Entity*-клас *Pupil,* клас-контролер *PupilRestController,* клас-сервіс *PupilService* та клас-репозиторій *PupilRepository.*

Клас-контролер *PupilRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/pupils».* Він має наступні методи: отримання учня за *id* - *getPupilById*; отримання всіх учнів за *id* школи – *getAllPupilsBySchoolId;* отримання всіх учнів за *id* класу – *getAllPupilsBySchoolClassId;* пошук учнів за ім'ям – *getAllPupilsByName;* отримання всіх учнів за *id* олімпіади – *getAllPupilsByOlympiadId;* отримання статистики учнів за *id* класу – *getAllStatisticsPupilsBySchoolClassId;* додавання нового учня – *addNewPupil;* редагування певного учня – *updatePupil;* видалення певного учня – *deletePupil.*

Клас-сервіс *PupilService* містить основну бізнес-логіку, що пов'язана із взаємодією з учнями. В ньому містяться такі методи: отримання учня за *id* - *findById*; отримання всіх учнів за *id* школи – *findAllBySchoolClass_School_Id;* отримання всіх учнів за *id* класу – *findAllBySchoolClass_Id;* пошук учнів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase;* отримання всіх учнів за *id* олімпіади – *findAllByOlympiadId;* отримання статистики учнів за *id* класу – *findAllStatisticsBySchoolClass_Id;* додавання нового учня – *addNewPupil;* редагування певного учня – *updatePupil;* видалення певного учня – *deletePupil.*

Клас-репозиторій *PupilRepository* взаємодіє з таблицею учнів *pupils.* Крім стандартних методів для роботи з базою даних він має такі методи: знайти всіх учнів за *id* школи – *findAllBySchoolClass_School_Id;* знайти всіх учнів за *id* класу – *findAllBySchoolClass _Id;* знайти всіх учнів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase;* порахувати кількість учнів у місті – *countPupilsInSchoolsByCity_Id;* порахувати загальну кількість учнів – *countPupilsInSchools;* порахувати учнів у класах за *id* школи –

countPupilsInSchoolClassesBySchool_Id; знайти всіх учнів за *id* олімпіади – *findAllByOlympiadId.*

Для таблиці класів *classes* створено *Entity*-клас *SchoolClass,* клас-контролер *SchoolClassRestController*, клас-сервіс *SchoolClassService* та клас-репозиторій *SchoolClassRepository.*

Клас-контролер *SchoolClassRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/classes».* Він має наступні методи: отримання класу за *id* – *getSchoolClassById*; отримання всіх шкіл за *id* школи – *getAllSchoolClassesBySchoolId;* отримання статистики класів за *id* школи – *getAllStatisticsSchoolClassesBySchoolId;* додавання нового класу – *addNewSchoolClass;* редагування певного класу – *updateSchoolClass;* видалення певного класу – *deleteSchoolClass.*

Клас-сервіс *SchoolClassService* містить основну бізнес-логіку, що пов’язана із взаємодією з класами. В ньому містяться такі методи: отримання класу за *id* - *findById*; отримання всіх шкіл за *id* школи – *findAllBySchool_Id;* отримання статистики класів за *id* школи – *findAllStatisticsBySchool_Id;* додавання нового класу – *addNewSchoolClass;* редагування певного класу – *updateSchoolClass;* видалення певного класу – *deleteSchoolClass.*

Клас-репозиторій *SchoolClassRepository* взаємодіє з таблицею класів *classes.* Крім стандартних методів для роботи з базою даних він має метод для знаходження класів за *id* школи – *findAllBySchool_Id.*

Для таблиці шкіл *schools* створено *Entity*-клас *School,* клас-контролер *SchoolRestController*, клас-сервіс *SchoolService* та клас-репозиторій *SchoolRepository.*

Клас-контролер *SchoolRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/schools».* Він має наступні методи: отримання школи за *id* – *getSchoolById*; отримання всіх шкіл за *id* міста – *getAllSchoolsByCityId;* отримання статистики шкіл за *id* міста – *getAllSchoolStatisticsByCityId;* отримання статистики всіх шкіл – *getAllSchoolStatistics;* додавання нової школи – *addNewSchool;* редагування певної школи – *updateSchool;* видалення певної школи – *deleteSchool.*

Клас-сервіс *SchoolService* містить основну бізнес-логіку, що пов'язана із взаємодією з школами. В ньому містяться такі методи: отримання школи за *id* - *findById*; отримання всіх шкіл за *id* міста – *findAllByCity_Id;* отримання статистики шкіл за *id* міста – *findAllSchoolStatisticsByCity_id;* отримання статистики всіх шкіл – *findAllSchoolStatistics;* додавання нової школи – *addNewSchool;* редагування певної школи – *updateSchool;* видалення певної школи – *deleteSchool.*

Клас-репозиторій *SchoolRepository* взаємодіє з таблицею шкіл *schools.* Крім стандартних методів для роботи з базою даних він має метод для знаходження шкіл за *id* міста – *findAllByCity_Id.*

Для таблиці предметів *subjects* створено *Entity*-клас *Subject,* клас-контролер *SubjectRestController,* клас-сервіс *SubjectService* та клас-репозиторій *SubjectRepository.*

Клас-контролер *SubjectRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/subjects».* Він має наступні методи: отримання предмету за *id* - *getSubjectById*; отримання всіх предметів – *getAllSubjects;* додавання нового предмету – *addNewSubject;* редагування певного предмету – *updateSubject;* видалення певного предмету – *deleteSubject.*

Клас-сервіс *SubjectService* містить основну бізнес-логіку, що пов'язана із взаємодією з предметами. В ньому містяться такі методи: отримання предмету за *id* - *findById*; отримання всіх предметів – *findAll;* додавання нового предмету – *addNewSubject;* редагування певного предмету – *updateSubject;* видалення певного предмету – *deleteSubject.*

Клас-репозиторій *SubjectRepository* взаємодіє з таблицею предметів *subjects.* Він має тільки стандартні методи для роботи з базою даних

Для таблиці вчителів *teachers* створено *Entity*-клас *Teacher,* клас-контролер *TeacherRestController,* клас-сервіс *TeacherService* та клас-репозиторій *TeacherRepository.*

Клас-контролер *TeacherRestController* оброблює запити за основною *URL* адресою *«http://localhost:8080/api/v1/teachers».* Він має наступні методи: отримання вчителя за *id* – *getTeacherById*; отримання всіх вчителів за *id* школи –

getTeachersBySchoolId; отримання всіх вчителів за *id* класу – *getTeachersBySchoolClassId;* отримання всіх вчителів за *id* класу та *id* предмету – *getTeacherBySchoolClassIdAndSubjectId*; пошук вчителів за ім'ям – *getTeachersByName;* додавання нового вчителя – *addNewTeacher;* редагування певного вчителя – *updateTeacher;* видалення певного вчителя – *deleteTeacher.*

Клас-сервіс *TeacherService* містить основну бізнес-логіку, що пов'язана із взаємодією з вчителями. В ньому містяться такі методи: отримання вчителя за *id* - *findById*; отримання всіх вчителів за *id* школи – *findAllBySchool_Id;* отримання всіх вчителів за *id* класу – *findAllBySchoolClass_Id;* отримання всіх вчителів за *id* класу та *id* предмету – *findBySchoolClass_IdAndSubject_Id*; пошук вчителів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase;* додавання нового вчителя – *addNewTeacher;* редагування певного вчителя – *updateTeacher;* видалення певного вчителя – *deleteTeacher.*

Клас-репозиторій *TeacherRepository* взаємодіє з таблицею вчителів *teachers.* Крім стандартних методів для роботи з базою даних він має такі методи: знайти всіх вчителів за *id* школи – *findAllBySchool_Id;* знайти всіх вчителів за *id* класу – *findAllBySchoolClass_Id;* знайти всіх вчителів за *id* класу та за *id* предмету – *findBySchoolClass_IdAndSubject_Id;* знайти всіх вчителів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase.*

Діаграма класів зображена на рис. 2.3.

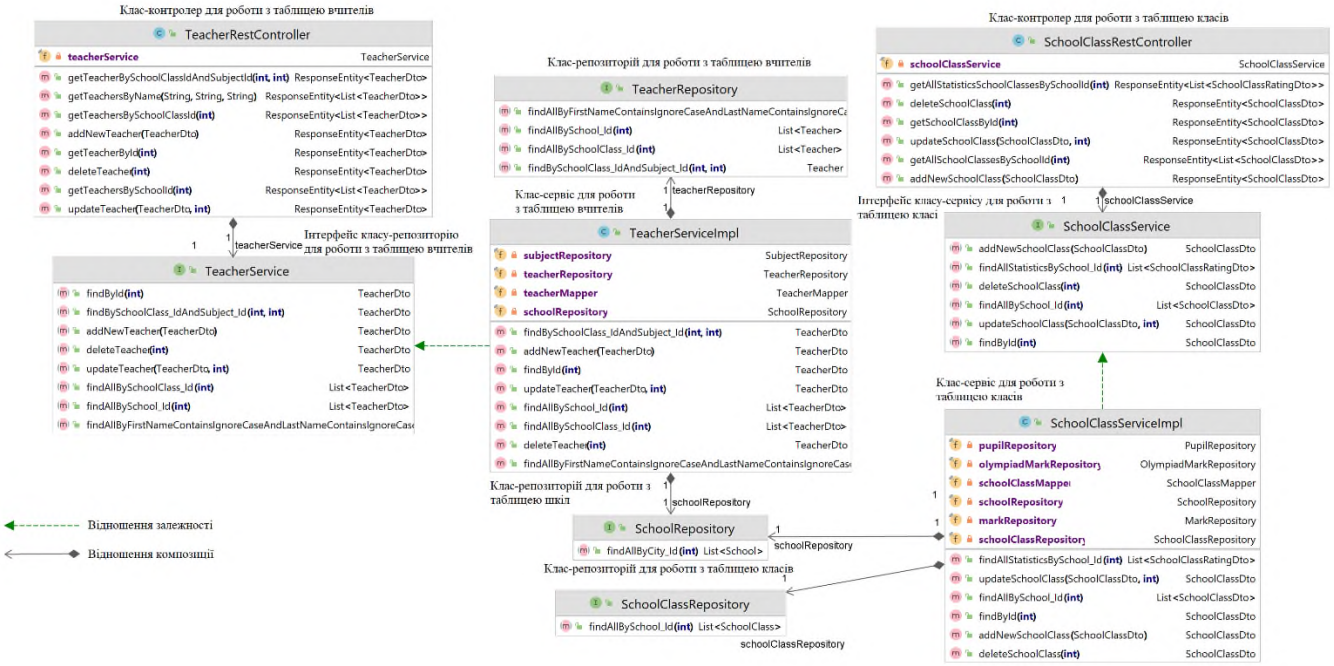

Рис 2.3. Діаграма класів

2.2. Проектування функцій та основних класів *frontend* частини

Для проектування *frontend* частини програмного забезпечення системи аналізу та обліку досягнень школярів було обрано мову програмування *Java* із використанням наступних фреймворків: *Spring, Spring Boot, Spring MVC, Thymeleaf.* Розробка велася в середовищі *IntelliJ IDEA* із використанням бібліотеки *Lombok* та інструменту автоматичної зборки проекту *Maven.* Для створення графічного оформлення веб-додатку був обраний фреймворк *Bootstrap.*

Thymeleaf – сучасний серверний інструмент *Java* шаблонів для веб-середовища. Він здатен обробляти *HTML, XML, CSS, JavaScript* та навіть звичайний текст. Основна ціль *Thymeleaf* – створення зручного та елегантного способу шаблонізації. Це досягається тим, що даний фреймворк заснований на принципу *Natural Templates*, тобто впровадження своєї логіки у шаблон не має впливу на відображення прототипу дизайну. *Thymeleaf* був створений із самого початку з дотриманням веб-стандартів, зокрема *HTML5*, що дозволяє створювати шаблони відповідні до цих стандартів[10].

Lombok – бібліотека для скорочення коду в класах і розширення функціональності мови програмування *Java*. Вона дозволяє уникнути написання шаблонного коду, автоматично генеруючи його під час зборки проекту, що економить час розробника. Також *Lombok* робить код більш читабельним.

Bootstrap – відкритий та безкоштовний *HTML, CSS, JavaScript* фреймворк, що дозволяє швидко створювати адаптивне графічне оформлення для веб-додатків. Даний інструмент представляє собою набір *CSS* та *JavaScript* файлів, які достатньо підключити до своїх сторінок. Після цього, розробнику стануть доступними усі інструменти даного фреймворку: колоночна система (або сітка *Bootstrap*), класи та компоненти. Усі ці фактори дають розробнику можливість максимально швидко та якісно зробити графічне оформлення сайту.

Сітка *Bootstrap* – це технологія фреймворку *Bootstrap*, яка умовно ділить по-горизонталі екран користувача на 12 рівних відрізків. Розробник розподіляє елементи веб-сторінки по цим відрізкам. Коли користувач відкриває дану веб-сторінку, усі елементи будуть знаходитися на свої місцях, незалежно від розміру екрана користувача. За рахунок цього і досягається адаптивність веб-сторінки[11].

Основними функціями *frontend* частини додатку системи аналізу та обліку успішності школярів є робота з базою даних через сервер. Так як *frontend* частина додатку показується користувачеві, її графічне оформлення повинно бути одночасно простим і практичним. А отже, для цього потрібно зробити зручну навігацію по сайту, підібрати палітру кольорів тощо.

Frontend частина додатку також використовує архітектурний стиль *REST*, проте, вона виступає не в ролі сервера, а в ролі клієнта. Тобто, ця частина повинна формувати *HTTP* запити та відправляти їх на сервер, а коли вона отримує відповідь від нього, правильно відображати отримані дані. Запити формуються на основі введених даних в формах на різних сторінках сайту.

Frontend частина також повинна мати класи-контролери та класи-сервіси, проте, на відміну від *backend* частини, у ній відсутні класи-репозиторії та *Entity*-класи, так як вона не має ніякого відношення до бази даних.

Для роботи з містами створено клас-контролер *CityController* та клас-сервіс *CityService.* Клас-контролер *CityController* оброблює запити за основною *URL* адресою *«http://localhost:8000/city».* Він має наступні методи: відображення сторінки зі списком міст – *resolveCityResultsView*; відображення сторінки редагування міст – *resolveCityEditView*; відображення сторінки додавання міст – *resolveCityEditAddView*; обробка додавання або редагування міст – *processEditOrAddCityThenRedirect*; обробка видалення міст – *processDeleteCity*.

Клас-сервіс *CityService* відправляє на сервер *HTTP* запити, що пов'язані з містами. В ньому містяться такі методи: пошук міста за *id* – *findById;* пошук усіх міст – *findAll*; додавання нового міста – *addNewCity;* редагування певного міста – *updateCity;* видалення певного міста – *deleteCity.*

Для роботи з уроками створено клас-контролер *LessonController* та клас-сервіс *LessonService.* Клас-контролер *LessonController* оброблює запити за основною *URL* адресою *«http://localhost:8000/lesson».* Він має наступні методи: відобразити сторінку редагування уроку – *resolveLessonEditView*; відобразити сторінку додавання уроку – *resolveLessonEditAddView*; обробка додавання або редагування уроків – *processLessonEditOrAdd*; обробка видалення уроків – *processLessonDelete*.

Клас-сервіс *LessonService* відправляє на сервер *HTTP* запити, що пов'язані з уроками. В ньому містяться такі методи: пошук уроку за *id* – *findById;* отримання всіх уроків за *id* предмету та *id* класу *findAllByTeacher_Subject_IdAndSchoolClass_Id*; отримання всіх уроків за *id* вчителя *findAllByTeacher_Id;* додавання нового уроку – *addNewLesson;* редагування певного уроку – *updateLesson;* видалення певного уроку – *deleteLesson.*

Для роботи з оцінками створено клас-контролер *MarkController* та клас-сервіс *MarkService.* Клас-контролер *MarkController* оброблює запити за основною *URL* адресою *«http://localhost:8000/marks».* Він має наступні методи: відображення сторінки пошуку оцінок - *resolveMarkSearchView*; відображення сторінки з оцінками класу за предмет– *resolveClassMarks*.

Клас-сервіс Mark*Service* відправляє на сервер *HTTP* запити, що пов'язані з оцінками. В ньому містяться такі методи: отримання усіх оцінок за *id* учня –

findAllByPupil_Id; отримання усіх оцінок за *id* предмету та *id* класу – *findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id;* отримання усіх оцінок за *id* уроку – *findAllByLesson_Id;* додавання нової оцінки – *addNewMark;* додавання декількох нових оцінок – *addNewMarks;* редагування певної оцінки – *updateMark;* видалення певної оцінки – *deleteMark.*

Для роботи з олімпіадами створено клас-контролер *OlympiadController* та клас-сервіс *OlympiadService.* Клас-контролер *OlympiadController* оброблює запити за основною *URL* адресою *«http://localhost:8000/olympiad».* Він має наступні методи: відображення сторінки пошуку олімпіад - *resolveOlympiadSearchView*; відображення сторінки з результатами пошуку олімпіад – *resolveOlympiadResultsView*; відображення сторінки з деталями олімпіади – *resolveOlympiadDetailsView*; відображення сторінки з редагуванням олімпіади – *resolveOlympiadEditView*; відображення сторінки з додаванням олімпіади – *resolveOlympiadEditAddView*; обробка додавання або редагування олімпіади – *processOlympiadEditOrAdd*; обробка видалення олімпіади – *processOlympiadDelete*.

Клас-сервіс *OlympiadService* відправляє на сервер *HTTP* запити, що пов’язані з олімпіадами. В ньому містяться такі методи: : отримання олімпіади за *id – findById*; отримання олімпіад за декілька *id – findAllById;* отримання всіх олімпіад – *findAll;* пошук олімпіад з одної дати по іншу – *findAllByDateBetween;* додавання нової олімпіади – *addNewOlympiad;* редагування певної олімпіади – *updateOlympiad;* видалення певної олімпіади – *deleteOlympiad.*

Для роботи зі школами створено клас-контролер *SchoolController* та клас-сервіс *SchoolService.* Клас-контролер *SchoolController* оброблює запити за основною *URL* адресою *«http://localhost:8000/school».* Він має наступні методи: відобразити сторінку пошуку шкіл - *resolveSchoolSearchView*; відобразити сторінку результатів пошуку шкіл – *resolveSchoolResultsView*; відображення сторінки деталей школи – *resolveSchoolDetailsView*; відображення сторінки редагування школи – *resolveSchoolEditView*; відображення сторінки додавання школи – *resolveSchoolEditAddView*; обробка додавання або редагування школи – *processSchoolEditOrAdd*; обробка видалення школи – *processSchoolDelete*;

Клас-сервіс *SchoolService* відправляє на сервер *HTTP* запити, що пов’язані зі школами. В ньому містяться такі методи: отримання школи за *id* – *findById*; отримання всіх шкіл за *id* міста – *findAllByCity_Id;* отримання статистики шкіл за *id* міста – *findAllSchoolStatisticsByCity_id;* отримання статистики всіх шкіл – *findAllSchoolStatistics;* додавання нової школи – *addNewSchool;* редагування певної школи – *updateSchool;* видалення певної школи – *deleteSchool.*

Для роботи з оцінками олімпіади створено клас-контролер *OlympiadMarkController* та клас-сервіс *OlympiadMarkService.* Клас-контролер *OlympiadMarkController* оброблює запити за основною *URL* адресою *«http://localhost:8000/olympiadmark».* Він має наступні методи: відображення сторінки редагування оцінок за олімпіади – *resolveOlympiadMarkEditView*; відображення сторінки додавання оцінок за олімпіади – *resolveOlympiadMarkEditAddView*; обробка додавання або редагування оцінки за олімпіаду – *processOlympiadMarkEditOrAdd*; обробка видалення оцінки за олімпіаду –*processOlympiadMarkDelete*;

Клас-сервіс *OlympiadMarkService* відправляє на сервер *HTTP* запити, що пов’язані з оцінками за олімпіади. В ньому містяться такі методи: отримання усіх оцінок за *id* учня – *findAllByPupil_Id;* отримання оцінки за *id* – *findById;* отримання усіх оцінок за *id* олімпіади – *findAllByOlympiad_Id;* додавання нової оцінки – *addNewOlympiadMark;* редагування певної оцінки – *updateOlympiadMark;* видалення певної оцінки – *deleteOlympiadMark.*

Для роботи з учнями створено клас-контролер *PupilController* та клас-сервіс *PupilService.* Клас-контролер *PupilController* оброблює запити за основною *URL* адресою *«http://localhost:8000/pupil».* Він має наступні методи: відображення сторінки пошуку учнів – *resolvePupilSearchView*; отримати учнів за ім’ям – *getAllPupilsContainingName*; отримати учнів за класом – *getAllPupilsByClassId;* відображення сторінки результатів пошуку учнів – *resolvePupilSearchResultsView*; відображення сторінки деталей учня – *resolvePupilDetails*; відображення сторінки редагування учня – *resolvePupilEditView*; відображення сторінки додавання учня –

resolvePupilEditAddView; обробка додавання або редагування учня – *processPupilEditOrAdd*; обробка видалення учня – *processPupilDelete*.

Клас-сервіс *PupilService* відправляє на сервер *HTTP* запити, що пов'язані з учнями. В ньому містяться такі методи: отримання учня за *id* - *findById*; отримання всіх учнів за *id* школи – *findAllBySchoolClass_School_Id;* отримання всіх учнів за *id* класу – *findAllBySchoolClass_Id;* пошук учнів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase;* отримання всіх учнів за *id* олімпіади – *findAllByOlympiadId;* отримання статистики учнів за *id* класу – *findAllStatisticsBySchoolClass_Id;* додавання нового учня – *addNewPupil;* редагування певного учня – *updatePupil;* видалення певного учня – *deletePupil.*

Для роботи з класами створено клас-контролер *SchoolClassController* та клас-сервіс *SchoolClassService.* Клас-контролер *SchoolClassController* оброблює запити за основною *URL* адресою *«http://localhost:8000/class».* Він має наступні методи: відображення сторінки з деталями – *resolveSchoolClassDetailsView*; відображення сторінки редагування класу – *resolveSchoolClassEditView*; відображення сторінки додавання класу – *resolveSchoolClassEditAddView*; обробка додавання або редагування класу – *processSchoolClassEditOrAdd*; обробка видалення класу – *processSchoolClassDelete*.

Клас-сервіс SchoolClass*Service* відправляє на сервер *HTTP* запити, що пов'язані з класами. В ньому містяться такі методи: отримання класу за *id* - *findById*; отримання всіх шкіл за *id* школи – *findAllBySchool_Id;* отримання статистики класів за *id* школи – *findAllStatisticsBySchool_Id;* додавання нового класу – *addNewSchoolClass;* редагування певного класу – *updateSchoolClass;* видалення певного класу – *deleteSchoolClass.*

Для роботи з предметами створено клас-контролер *SubjectController* та клас-сервіс *SubjectService.* Клас-контролер *SubjectController* оброблює запити за основною *URL* адресою *«http://localhost:8000/subject».* Він має наступні методи: відображення сторінки зі списком предметів – *resolveSubjectResultsView*; відображення сторінки редагування предметів – *resolveSubjectEditView*;

відображення сторінки додавання предметів – *resolveSubjectEditAddView*; обробка додавання або редагування предметів – *processEditOrAddSubjectThenRedirect*; обробка видалення предметів – *processDeleteSubject*.

Клас-сервіс *SubjectService* відправляє на сервер *HTTP* запити, що пов'язані з предметами. В ньому містяться такі методи: отримання предмету за *id* – *findById*; отримання всіх предметів – *findAll;* додавання нового предмету – *addNewSubject;* редагування певного предмету – *updateSubject;* видалення певного предмету – *deleteSubject.*

Для роботи з вчителями створено клас-контролер *TeacherController* та клас-сервіс *TeacherService.* Клас-контролер *TeacherController* оброблює запити за основною *URL* адресою *«http://localhost:8000/teacher».* Він має наступні методи: відображення сторінки з пошуком вчителів – *resolveTeacherSearchView*; пошук вчителів за ім'ям – *resolveTeacherResultsNameView*; пошук вчителів за школою – *resolveTeacherResultsSchoolView*; відобразити сторінку з деталями вчителя – *resolveTeacherDetailsView*; відображення сторінки редагування вчителя – *resolveTeacherEditView*; відображення сторінки додавання вчителя – *resolveTeacherEditAddView*; обробка додавання або редагування вчителів – *processTeacherEditOrAdd*; обробка видалення вчителів – *processTeacherDelete*.

Клас-сервіс *TeacherService* відправляє на сервер *HTTP* запити ,що пов'язані з вчителями. В ньому містяться такі методи: отримання вчителя за *id* – *findById*; отримання всіх вчителів за *id* школи – *findAllBySchool_Id;* отримання всіх вчителів за *id* класу – *findAllBySchoolClass_Id;* отримання всіх вчителів за *id* класу та *id* предмету – *findBySchoolClass_IdAndSubject_Id*; пошук вчителів за ім'ям – *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase;* додавання нового вчителя – *addNewTeacher;* редагування певного вчителя – *updateTeacher;* видалення певного вчителя – *deleteTeacher*.

Для роботи з аналітикою створено клас-контролер *AnalysisController*. Клас-контролер *AnalysisController* оброблює запити за основною *URL* адресою *«http://localhost:8000/analysis».* Він має наступні методи: відображення сторінки з пошуком шкіл – *resolveAnalysisSchoolSearchView*; відображення сторінки з

аналітикою успішності шкіл – *resolveAnalysisSchoolResultViewByCityId*; відображення сторінки з пошуком класів – *resolveAnalysisSchoolClassSearchView*; відображення сторінки з аналізом успішності класів – *resolveAnalysisSchoolClassResultViewBySchoolId*; відображення сторінки з пошуком учнів – *resolveAnalysisPupilSearchView*; відображення сторінки з аналізом успішності учнів – *resolveAnalysisPupilResultViewBySchoolId*.

2.3. Проектування функцій аналізу та обліку успішності школярів

Під час проектування програмного забезпечення було спроектовано основні функції аналізу та успішності школярів: аналіз успішності шкіл серед усіх доданих до бази даних шкіл; аналіз успішності шкіл серед шкіл певного міста; аналіз успішності класів у певній школі; аналіз успішності учнів у певному класі; аналіз успішності учнів по олімпіаді.

Аналіз успішності шкіл серед усіх доданих до бази даних шкіл було спроектовано у класах *backend* частини: *PupilRepository, MarkRepository, OlympiadMarkRepository, SchoolService*. У класі-сервісі *SchoolService* використовується метод для формування даних для аналітики – *findAllSchoolStatistics*. Він взаємодіє з класами-репозиторіями учнів, оцінок та оцінок за олімпіади, отримувати від них кількість учнів у школі, середню оцінку і середню оцінку за олімпіади та формувати на їх основі дані вже з готовою аналітикою: назву школи, місто, кількість учнів, середня оцінка учнів у школі, середня оцінка учнів за олімпіади. Потім вони відправляються на *frontend* частину веб-додатку, де отримані дані відображаються у вигляді таблиці, в якій користувач може відсортувати їх за потрібною йому колонкою, або скористатися пошуком щоб знайти потрібний йому запис.

Аналіз успішності шкіл серед шкіл певного міста було спроектовано у класах *backend* частини: *PupilRepository, MarkRepository, OlympiadMarkRepository, SchoolService*. У класі-сервісі *SchoolService* використовується метод для формування даних для аналітики – *findAllSchoolStatisticsByCity_id*. Він взаємодіє з класами-

репозиторіями учнів, оцінок та оцінок за олімпіади, отримувати від них кількість учнів, середню оцінку і середню оцінку за олімпіади та формувати на їх основі дані вже з готовою аналітикою: назву школи, місто, кількість учнів у школі, середня оцінка учнів у школі, середня оцінка учнів за олімпіади. Потім вони відправляються на *frontend* частину веб-додатку, де отримані дані відображаються у вигляді таблиці, в якій користувач може відсортувати їх за потрібною йому колонкою, або скористатися пошуком щоб знайти потрібний йому запис.

Аналіз успішності класів у певній школі було спроектовано у класах *backend* частини: *PupilRepository, MarkRepository, OlympiadMarkRepository, SchoolClassService*. У класі-сервісі *SchoolClassService* використовується метод для формування даних для аналітики – *findAllStatisticsBySchool_id*. Він взаємодіє з класами-репозиторіями учнів, оцінок та оцінок за олімпіади, отримувати від них кількість учнів у класі, середню оцінку і середню оцінку за олімпіади та формувати на їх основі дані вже з готовою аналітикою: назву школи, клас, кількість учнів, середня оцінка учнів у школі, середня оцінка учнів за олімпіади. Потім вони відправляються на *frontend* частину веб-додатку, де отримані дані відображаються у вигляді таблиці, в якій користувач може відсортувати їх за потрібною йому колонкою, або скористатися пошуком щоб знайти потрібний йому запис.

Аналіз успішності учнів у певному класі було спроектовано у класах *backend* частини: *PupilRepository, MarkRepository, OlympiadMarkRepository, PupilService*. У класі-сервісі *PupilService* використовується метод для формування даних для аналітики – *findAllStatisticsBySchoolClass_Id*. Він взаємодіє з класами-репозиторіями учнів, оцінок та оцінок за олімпіади, отримувати від них кількість учнів у класі, середню оцінку і середню оцінку за олімпіади та формувати на їх основі дані вже з готовою аналітикою: назву школи, клас, кількість учнів, середня оцінка учнів у школі, середня оцінка учнів за олімпіади. Потім вони відправляються на *frontend* частину веб-додатку, де отримані дані відображаються у вигляді таблиці, в якій користувач може відсортувати їх за потрібною йому колонкою, або скористатися пошуком щоб знайти потрібний йому запис.

Аналіз успішності учнів по олімпіаді було спроектовано у класах *backend* частини: *OlympiadMarkRepository, OlympiadMarkService*. У класі-сервісі *OlympiadMarkService* використовується метод для знаходження всіх оцінок за певну олімпіаду – *findAllByOlympiad_Id*. Він взаємодіє з класом-репозиторієм оцінок за олімпіади та отримує від них усі оцінки потрібної олімпіади, а саме: місце учня у рейтингу, повне ім’я учня, школа у якій він вчиться, його оцінка. Потім вони відправляються на *frontend* частину веб-додатку, де отримані дані відображаються у вигляді таблиці, в якій користувач може відсортувати їх за потрібною йому колонкою, або скористатися пошуком щоб знайти потрібний йому запис.

2.4. Висновки до розділу

У даному розділі було обрано мову програмування для розробки *backend* частини програми – *Java* із фреймворками: *Spring, Hibernate, Spring Boot, Spring Data, Spring MVC*. Було обрано середовище розробки *IntelliJ IDEA* з інструментом автоматичної зборки проектів *Maven*. Для тестування було обрано фреймворки *JUnit* та *Mockito*. Була обрана система управління базами даних *MySQL*.

Було спроектовано базу даних для вирішення поставленої задачі, а саме таблиці: міст, уроків, оцінок, олімпіад, оцінок за олімпіади, учні, школи, класи, предмети та вчителі. Також було спроектовано основні функції серверної частини програмного забезпечення системи аналізу та обліку успішності школярів.

Було визначено, що основними функціями *backend* частини додатку системи аналізу та обліку успішності школярів є робота з таблицями, а саме можливість їх перегляду, редагування, додавання та видалення.

Були спроектовані *Entity*-класи для *backend* частини, а саме: *City* – для міст, *Lesson* – для уроків, *Mark* – для оцінок, *Olympiad* – для олімпіад, *OlympiadMark* – для оцінок за олімпіади, *Pupil* – для учнів, *School* – для шкіл, *SchoolClass* – для класів, *Subject* – для предметів, *Teacher* – для вчителів. Були спроектовані класи-контролери: City*RestController, LessonRestController, MarkRestController, OlympiadMarkRestController, OlympiadRestController, PupilRestController,*

SchoolClassRestController, *SchoolRestController*, *SubjectRestController*, *TeacherRestController*. Були спроектовані класи-сервіси для : *CityService*, *LessonService*, *MarkService*, *OlympiadMarkService*, *OlympiadService*, *PupilService*, *SchoolClassService*, *SchoolService*, *SubjectService*, *TeacherService*. Були спроектовані класи-репозиторії: *CityRepository*, *LessonRepository*, *MarkRepository*, *OlympiadMarkRepository*, *OlympiadRepository*, *PupilRepository*, *SchoolClassRepository*, *SchoolRepository*, *SubjectRepository*, *TeacherRepository*.

Було спроектовано основні функції *frontend* частини проекту. Також були спроектовані класи-контролери для: City*Controller*, *LessonController*, *MarkController*, *OlympiadMarkController*, *OlympiadController*, *PupilController*, *SchoolClassController*, *SchoolController*, *SubjectController*, *TeacherController*, *AnalysisController*. Були спроектовані класи-сервіси: *CityService*, *LessonService*, *MarkService*, *OlympiadMarkService*, *OlympiadService*, *PupilService*, *SchoolClassService*, *SchoolService*, *SubjectService*, *TeacherService*.

Було спроектовано основні функції аналізу та успішності школярів: аналіз успішності шкіл серед усіх доданих до бази даних шкіл; аналіз успішності шкіл серед шкіл певного міста; аналіз успішності класів у певній школі; аналіз успішності учнів у певному класі; аналіз успішності учнів по олімпіаді.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ АНАЛІЗУ ТА ОБЛІКУ УСПІШНОСТІ ШКОЛЯРІВ

3.1. Склад файлів *backend* частини додатку

Розробка *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA.* Мова проекту – *Java,* інструмент для автоматизації зборки – *Maven.* Версію мови програмування *Java* було обрано одинадцяту, так як вона є найрозповсюдженою та досі підтримується компанією-розробником. Тип архівування проекту було обрано *Jar.* Щоб створити новий проект, було натиснуто на панелі зверху на кнопку «*File*», у випадаючому списку наведено курсор на надпис «*New*» та у наступному списку обрано пункт «*Project*». Після того, як відкриється вікно з налаштуванням проекту, зліва у списку обрано пункт «*Spirng Initializr*» щоб створити проект із використанням фреймворку «*Spring Boot*». Назву проекту було обрано *journal_project,* папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_project.* Назву пакету проекту – *com.makariev.journal_project.* Після натискання кнопки «*Next*» відкривається вікно для додавання залежностей до проекту та вибір версії фреймворку *Spring Boot.* Було обрано версію 2.7.5 та додано наступні залежності: *Spring Boot DevTools, Lombok, Spring Web, Spring Data JPA, MySQL Driver*. Далі був процес створення проекту та автоматичного завантаження бібліотек для залежностей, що були обрані.

Папка *journal_project* містить автоматично згенерованіні при створенні проекту папки: *.git, .idea, .mvn, src* та *target.* Також вона має автоматично створені файли: *.gitignore.txt, HELP.md, journal_project.iml, mvnw, mvnw.cmd, pom.xml.* У ході розробки програмного забезпечення системи аналізу та обліку успішності школярів були створені наступні пакети та файли у папці *src\main\java\com\makariev\journal_project*:

– пакет *config* призначений для конфігураційних файлів та містить у собі файл *CorsFilter.java,* що визначає з якими налаштуваннями клієнт може звертатися то сервера*;*

– пакет *controller* містить у собі класи-контролери. Для кожного *Entity*-класу створений свій клас-контролер*: CityRestController.java, LessonRestController.java, MarkRestController.java, OlympiadMarkRestController.java, OlympiadRestController.java, PupilRestController.java, SchoolClassRestController.java, SchoolRestController.java, SubjectRestController.java, TeacherRestController.java;*

– пакет *dto* містить у собі класи *DTO(Data Transfer Object)*. Вони потрібні для формування *HTTP* відповіді на сервері на передачі їх клієнтові. Для кожного *Entity*-класу є свій *DTO*-клас*: CityDto.java, LessonDto.java, MarkDto.java, OlympiadDto.java, OlympiadMarkDto.java, PupilDto.java, PupilRatingDto.java, SchoolClassDto.java, SchoolClassRatingDto.java, SchoolDto.java, SchoolRatingDto.java, SubjectDto.java, TeacherDto.java;*

– пакет *entity* містить у собі *Entity*-класи*: City.java, Lesson.java, Mark.java, Olympiad.java, OlympiadMark.java, Pupil.java, School.java, SchoolClass.java, Subject.java, Teacher.java;*

– пакет *exception* містить класи для виключень та їх обробки. Виключення потрібні для сповіщення про певну помилку. Якщо сервер у відповіді дає виключення, клієнт може сповістити користувача про це. У ньому містяться наступні файли*: WrongIdForEntityExceptionHandler.java, ExceptionDetails.java, WrongIdForEntityException.java;*

– пакет *mapper* містить у собі класи для перетворення *Entity*-об’єктів у *DTO*-об’єкти та навпаки. Для кожного *Entity* є свій *mapper*-клас*: CityMapper.java, LessonMapper.java, MarkMapper.java, OlympiadMapper.java, OlympiadMarkMapper.java, PupilMapper.java, SchoolClassMapper.java, SchoolMapper.java, SubjectMapper.java, TeacherMapper.java;*

– пакет *repository* містить у собі класи-репозиторії. Вони представляють собою інтерфейси з сигнатурами методів для легшої взаємодії з базою даних. Даний пакет містить файли: *CityRepository.java, LessonRepository.java, MarkRepository.java,*

OlympiadMarkRepository.java, OlympiadRepository.java, PupilRepository.java, SchoolClassRepository.java, SchoolRepository.java, SubjectRepository.java, TeacherRepository.java;

– пакет *service* містить у собі класи-сервіси. Для кожного *Entity*-класу створений свій клас-сервіс та інтерфейс до нього. Класи-сервіси мають у кінці назви *Impl,* що означає що даний клас є реалізацією інтерфейсу. Він містить файли*: CityService.java, LessonService.java, MarkService.java, OlympiadService.java, OlympiadMarkService.java, PupilService.java, SchoolClassService.java, SchoolService.java, SubjectService.java, TeacherService.java, CityServiceImpl.java, LessonServiceImpl.java, MarkServiceImpl.java, OlympiadServiceImpl.java, OlympiadMarkServiceImpl.java, PupilServiceImpl.java, SchoolClassServiceImpl.java, SchoolServiceImpl.java, SubjectServiceImpl.java, TeacherServiceImpl.java.*

Папка *.git* призначена для файлів, що пов'язані з системою контролю версій *GitHub.* Папка *.idea* містить файли пов'язані з середовищем розробки *IntelliJ IDEA* та в основному містить метаінформацію про проект. Папка *.mvn* містить у собі файли для системи автоматичної зборки проекту *Maven.* Папка *src* містить вихідний код класів, написаних користувачем. Саме в ньому зберігаються *Java* пакети.

Файл *CityRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *City.* У ньому є поле *CityService cityService* та методи: *ResponseEntity<CityDto> getCityById(@PathVariable int cityId); ResponseEntity<List<CityDto>> getAllCities(); ResponseEntity<CityDto> addNewCity(@RequestBody CityDto cityDto); ResponseEntity<CityDto> updateCity(@RequestBody CityDto cityDto, @PathVariable int cityId); ResponseEntity<CityDto> deleteCity(@PathVariable int cityId).*

Файл *LessonRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Lesson.* У ньому є поле *LessonService lessonService* та методи*: LessonDto getLessonById(@PathVariable int lessonId); ResponseEntity<List<LessonDto>> getAllLessonsBySubjectIdAndSchoolClassId(@PathVariable int subjectId, @PathVariable int schoolClassId); ResponseEntity<List<LessonDto>> getAllLessonsByTeacherId(@PathVariable int teacherId); ResponseEntity<LessonDto>*

addNewLesson(@RequestBody LessonDto lessonDto); ResponseEntity<LessonDto> updateLesson(@RequestBody LessonDto lessonDto, @PathVariable int lessonId); ResponseEntity<LessonDto> deleteLesson(@PathVariable int lessonId).

Файл *MarkRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Mark*. У ньому є поле *MarkService markService* та методи: *ResponseEntity<List<MarkDto>> getAllMarksByPupilId(@PathVariable int pupilId); ResponseEntity<List<MarkDto>> getAllMarksBySubjectIdAndSchoolClassId(@PathVariable int subjectId, @PathVariable int schoolClassId); ResponseEntity<List<MarkDto>> getAllMarksByLessonId(@PathVariable int lessonId); ResponseEntity<MarkDto> addNewMark(@RequestBody MarkDto markDto); ResponseEntity<List<MarkDto>> addNewMarks(@RequestBody List<MarkDto> markDtos, @PathVariable int lessonId); ResponseEntity<MarkDto> updateMark(@RequestBody MarkDto markDto, @PathVariable int markId); ResponseEntity<MarkDto> deleteMark(@PathVariable int markId).*

Файл *OlympiadMarkRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *OlympiadMark*. У ньому є поле *OlympiadMarkService olympiadMarkService* та методи: *ResponseEntity<OlympiadMarkDto> getOlympiadMarkById(@PathVariable int olympiadMarkId); ResponseEntity <List<OlympiadMarkDto>> getAllOlympiadMarksByPupilId(@PathVariable int pupilId); ResponseEntity <List<OlympiadMarkDto>> getAllOlympiadMarksByOlympiadId(@PathVariable int olympiadId); ResponseEntity<OlympiadMarkDto> addNewOlympiadMark(@RequestBody OlympiadMarkDto olympiadMarkDto); ResponseEntity<OlympiadMarkDto> updateOlympiadMark(@RequestBody OlympiadMarkDto olympiadMarkDto, @PathVariable int olympiadMarkId); ResponseEntity<OlympiadMarkDto> deleteOlympiadMark(@PathVariable int olympiadMarkId).*

Файл *OlympiadRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Olympiad*. У ньому є поле *OlympiadService olympiadService* та методи: *ResponseEntity<OlympiadDto> getOlympiadById(@PathVariable int*

olympiadId); ResponseEntity<List<OlympiadDto>> getOlympiadsByIds(@RequestBody List<Integer> olympiadIds); ResponseEntity<List<OlympiadDto>> getAllOlympiads(); ResponseEntity<List<OlympiadDto>> getAllOlympiadsBetweenDates(@PathVariable Date dateFrom, @PathVariable Date dateTo); ResponseEntity<OlympiadDto> addNewOlympiad(@RequestBody OlympiadDto olympiadDto); ResponseEntity<OlympiadDto> updateOlympiad(@RequestBody OlympiadDto olympiadDto, @PathVariable int olympiadId); ResponseEntity<OlympiadDto> deleteOlympiad(@PathVariable int olympiadId).

Файл *PupilRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Pupil*. У ньому є поле *PupilService pupilService* та методи: *ResponseEntity<PupilDto> getPupilById(@PathVariable int pupilId); ResponseEntity<List<PupilDto>> getAllPupilsBySchoolId(@PathVariable int schoolId); ResponseEntity<List<PupilDto>> getAllPupilsBySchoolClassId(@PathVariable int schoolClassId); ResponseEntity<List<PupilDto>> getAllPupilsByName(); ResponseEntity<List<PupilDto>> getAllPupilsByOlympiadId(@PathVariable int olympiadId); ResponseEntity<List<PupilRatingDto>> getAllStatisticsPupilsBySchoolClassId(@PathVariable int schoolClassId); ResponseEntity<PupilDto> addNewPupil(@RequestBody PupilDto pupilDto); ResponseEntity<PupilDto> updatePupil(@RequestBody PupilDto pupilDto, @PathVariable int pupilId); ResponseEntity<PupilDto> deletePupil(@PathVariable int pupilId).*

Файл *SchoolClassRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *SchoolClass*. У ньому є поле *SchoolClassService schoolClassService* та методи: *ResponseEntity<SchoolClassDto> getSchoolClassById(@PathVariable int schoolClassId); ResponseEntity<List<SchoolClassDto>> getAllSchoolClasses (@PathVariable int schoolId); ResponseEntity<List<SchoolClassRatingDto>> getAllStatisticsSchoolClasses (@PathVariable int schoolId); ResponseEntity<SchoolClassDto> addNewSchoolClass(@RequestBody SchoolClassDto schoolClassDto); ResponseEntity<SchoolClassDto> updateSchoolClass(@RequestBody SchoolClassDto*

schoolClassDto, @PathVariable int schoolClassId); ResponseEntity<SchoolClassDto> deleteSchoolClass(@PathVariable int schoolClassId).

Файл *SchoolRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *School*. У ньому є поле *SchoolService schoolService* та методи: *ResponseEntity<SchoolDto> getSchoolById(@PathVariable int schoolId); ResponseEntity<List<SchoolDto>> getAllSchoolsByCityId(@PathVariable int cityId); ResponseEntity<List<SchoolRatingDto>> getAllSchoolStatisticsByCityId(@PathVariable int cityId); ResponseEntity<List<SchoolRatingDto>> getAllSchoolStatistics(); ResponseEntity<SchoolDto> addNewSchool(@RequestBody SchoolDto schoolDto); ResponseEntity<SchoolDto> updateSchool(@RequestBody SchoolDto schoolDto, @PathVariable int schoolId); ResponseEntity<SchoolDto> deleteSchool(@PathVariable int schoolId).*

Файл *SubjectRestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Subject*. У ньому є поле *SubjectlService subjectService* та методи: *ResponseEntity<SubjectDto> getSubjectById(@PathVariable int subjectId); ResponseEntity<List<SubjectDto>> getAllSubjects(); ResponseEntity<SubjectDto> addNewSubject(@RequestBody SubjectDto subjectDto); ResponseEntity<SubjectDto> updateSubject(@RequestBody SubjectDto subjectDto, @PathVariable int subjectId); ResponseEntity<SubjectDto> deleteSubject(@PathVariable int subjectId).*

Файл Teacher*RestController.java* містить у собі реалізацію класу-контролеру для *Entity*-класу *Teacher*. У ньому є поле *SubjectlService subjectService* та методи: *ResponseEntity<TeacherDto> getTeacherById(@PathVariable int teacherId); ResponseEntity<List<TeacherDto>> getTeachersBySchoolId(@PathVariable int schoolId); ResponseEntity<List<TeacherDto>> getTeachersBySchoolClassId(@PathVariable int schoolClassId); ResponseEntity<TeacherDto> getTeacherBySchoolClassIdAndSubjectId(@PathVariable int schoolClassId, @PathVariable int subjectId); ResponseEntity<List<TeacherDto>> getTeachersByName(); ResponseEntity<TeacherDto> addNewTeacher(@RequestBody TeacherDto teacherDto); ResponseEntity<TeacherDto> updateTeacher(@RequestBody*

TeacherDto teacherDto, @PathVariable int teacherId); ResponseEntity<TeacherDto> deleteTeacher(@PathVariable int teacherId).

Файл *CityRepository.java* – клас-репозиторій для *Entity*-класу *City*. Він має тільки стандартні методи для роботи з базою даних.

Файл *LessonRepository.java* – клас-репозиторій для *Entity*-класу *Lesson*. У ньому описані наступні методи: *List<Lesson> findAllByTeacher_Subject_IdAndSchoolClass_IdOrderByLessonDate(int subjectId, int schoolClassId); List<Lesson> findAllByTeacher_IdOrderByLessonDate(int teacherId).*

Файл *MarkRepository.java* – клас-репозиторій для *Entity*-класу *Mark*. У ньому описані наступні методи: *List<Mark> findAllByPupil_Id(int pupilId); List<Integer> findAverageMarkByPupil_Id(int pupilId); List<Integer> findAverageMarkByPupil_IdAndSubject_Id(int pupilId, int subjectId); List<AverageMarkSchool> countSchoolAverageMarkByCity_Id(int cityId); List<AverageMarkSchool> countSchoolAverageMark(); List<AverageMarkSchoolClass> countSchoolClassAverageMarkBySchool_Id(int schoolId); List<AverageMarkPupil> countPupilAverageMarkBySchoolClass_Id(int schoolClassId); List<Mark> findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id(int subjectId, int schoolClassId); List<Mark> findAllByLesson_Teacher_Subject_IdAndPupil_Id(int subjectId, int pupilId); List<Mark> findAllByLesson_Id(int lessonId).*

Файл *OlympiadMarkRepository.java* – клас-репозиторій для *Entity*-класу *OlympiadMark*. У ньому описані наступні методи: *void calculatePositions(@Param("olympiadId") int olympiadId); List<AverageOlympiadMarkSchool> countSchoolAverageOlympiadMarkByCity_Id(int cityId); List<AverageOlympiadMarkSchool> countSchoolAverageOlympiadMark(); List<AverageOlympiadMarkSchoolClass> countSchoolClassAverageOlympiadMarkBy (int schoolId); List<AverageOlympiadMarkPupil> countPupilAverageOlympiadMarkBy (int schoolClassId); List<OlympiadMark> findAllByPupil_Id(int pupilId); List<OlympiadMark> findAllByOlympiad_Id(int olympiadId).*

Файл *OlympiadRepository.java* – клас-репозиторій для *Entity*-класу *Olympiad.* У ньому міститься опис методу *List<Olympiad> findAllByDateBetween(Date from, Date to).*

Файл *PupilRepository.java* – клас-репозиторій для *Entity*-класу *Pupil.* У ньому міститься опис методів: *List<Pupil> findAllBySchoolClass_School_Id(int schoolId); List<Pupil> findAllBySchoolClass_Id(int schoolClassId); List<Pupil> findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase(String firstName, String lastName, String middleName); List<PupilCountSchool> countPupilsInSchoolsByCity_Id(int cityId); List<PupilCountSchool> countPupilsInSchools();List<PupilCountSchoolClass> countPupilsInSchoolClassesBySchool_Id(int schoolId); List<Pupil> findAllByOlympiadId(int olympiadId).*

Файл *SchoolClassRepository.java* – клас-репозиторій для *Entity*-класу *SchoolClass.* У ньому міститься опис методу *List<SchoolClass> findAllBySchool_Id(int schoolId).*

Файл *SchoolRepository.java* – клас-репозиторій для *Entity*-класу *School.* У ньому міститься опис методу *List<School> findAllByCity_Id(int cityId).*

Файл *SubjectRepository.java* – клас-репозиторій для *Entity*-класу *Subjecct.* Він має тільки стандартні методи для роботи з базою даних.

Файл *TeacherRepository.java* – клас-репозиторій для *Entity*-класу *Teacher.* У ньому міститься опис методів: *List<Teacher> findAllBySchool_Id(int schoolId); List<Teacher> findAllBySchoolClass_Id(int schoolClassId); Teacher findBySchoolClass_IdAndSubject_Id(int schoolClassId, int subjectId); List<Teacher> findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase(String firstName, String lastName, String middleName);*

Файл *CityService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *City – CityServiceImpl.java.* У ньому описані наступні методи: *CityDto findById(int cityId); List<CityDto> findAll();CityDto addNewCity(CityDto cityDto); CityDto updateCity(CityDto cityDto, int cityId); CityDto deleteCity(int cityId).*

Файл *LessonService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Lesson* – *LessonServiceImpl.java*. У ньому описані наступні методи: *List<LessonDto> findAllByTeacher_Subject_IdAndSchoolClass_Id(int subjectId, int schoolClassId); LessonDto findById(int lessonId); List<LessonDto> findAllByTeacher_Subject_IdAndSchoolClass_IdAndSetAdditionalInfo(int subjectId, int schoolClassId); List<LessonDto> findAllByTeacher_Id(int teacherId); LessonDto addNewLesson(LessonDto lessonDto); LessonDto updateLesson(LessonDto lessonDto, int lessonId); LessonDto deleteLesson(int lessonId).*

Файл *MarkService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Mark* – *MarkServiceImpl.java*. У ньому описані наступні методи: *List<MarkDto> findAllByPupil_Id(int pupilId); List<MarkDto> findAllByLesson_Teacher_ Subject_IdAndPupil_Id(int subjectId, int pupilId); List<MarkDto> findAllByLesson_Id(int lessonId); List<MarkDto> findAllByLesson_Teacher_Subject_IdAndPupil_ SchoolClass_Id(int subjectId, int schoolClassId); MarkDto addNewMark(MarkDto markDto); List<MarkDto> addNewMarks(List<MarkDto> markDtos, int lessonId); MarkDto updateMark(MarkDto markDto, int markId); MarkDto deleteMark(int markId).*

Файл *OlympiadMarkService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *OlympiadMark* – *OlympiadMarkServiceImpl.java*. У ньому описані наступні методи: *OlympiadMarkDto findById(int olympiadMarkId); List<OlympiadMarkDto> findAllByPupil_Id(int pupilId); List<OlympiadMarkDto> findAllByOlympiad_Id(int olympiadId); OlympiadMarkDto addNewOlympiadMark(OlympiadMarkDto olympiadMarkDto); OlympiadMarkDto updateOlympiadMark(OlympiadMarkDto olympiadMarkDto, int olympiadMarkId); OlympiadMarkDto deleteOlympiadMark(int olympiadMarkId).*

Файл *OlympiadService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Olympiad* – *OlympiadServiceImpl.java*. У ньому описані наступні методи: *OlympiadDto findById(int olympiadId); List<OlympiadDto> findAllById(List<Integer> olympiadIds); List<OlympiadDto> findAll();List<OlympiadDto> findAllByDateBetween(Date from, Date to); OlympiadDto*

addNew Olympiad(OlympiadDto olympiadDto); OlympiadDto update Olympiad(OlympiadDto olympiadDto, int olympiadId); OlympiadDto deleteOlympiad(int olympiadId).

Файл *PupilService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Pupil – PupilServiceImpl.java.* У ньому описані наступні методи: *PupilDto findById(int pupilId); List<PupilDto> findAllBySchoolClass_School_Id(int schoolId); List<PupilDto> findAllBySchoolClass_Id(int schoolClassId); List<PupilDto> findAllByFirstName ContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCa se(String firstName, String lastName, String middleName); List<PupilDto> findAllByOlympiadId(int olympiadId); List<PupilDto> findAllByOlympiadId(int olympiadId); List<PupilRatingDto> findAllStatisticsBySchoolClass_Id(int schoolClassId); PupilDto addNewPupil(PupilDto pupilDto); PupilDto updatePupil(PupilDto pupilDto, int pupilId); PupilDto deletePupil(int pupilId).*

Файл *SchoolClassService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *SchoolClass – SchoolClassServiceImpl.java.* У ньому описані наступні методи: *SchoolClassDto findById(int schoolClassId); List<SchoolClassDto> findAllBySchool_Id(int schoolId); List<SchoolClassRatingDto> findAllStatisticsBy School_Id(int schoolId); SchoolClassDto addNewSchoolClass(SchoolClassDto schoolClassDto); SchoolClassDto updateSchoolClass(SchoolClassDto schoolClassDto, int schoolClassId); SchoolClassDto deleteSchoolClass(int schoolClassId).*

Файл *SchoolService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *School – SchoolServiceImpl.java.* У ньому описані наступні методи: *List<SchoolDto> findAllByCity_Id(int cityId); SchoolDto findById(int schoolId); List<SchoolRatingDto> findAllSchoolStatisticsByCity_id(int cityId); List<SchoolRatingDto> findAllSchoolStatistics();SchoolDto addNewSchool(SchoolDto schoolDto); SchoolDto updateSchool(SchoolDto schoolDto, int schoolId); SchoolDto deleteSchool(int schoolId).*

Файл *SubjectService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Subject – SubjectServiceImpl.java.* У ньому описані наступні методи: *SubjectDto findById(int subjectId); SubjectDto addNewSubject(SubjectDto subject); List<SubjectDto> findAll();SubjectDto updateSubject(SubjectDto subjectDto, int subjectId); SubjectDto deleteSubject(int subjectId).*

Файл *TeacherService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Teacher – TeacherServiceImpl.java.* У ньому описані наступні методи: *TeacherDto findById(int teacherId); List<TeacherDto> findAllBySchool_Id(int schoolId); List<TeacherDto> findAllBySchoolClass_Id(int schoolClassId); TeacherDto findBySchool Class_IdAndSubject_Id(int schoolClassId, int subjectId); List<TeacherDto> findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase(String firstName, String lastName, String middleName); TeacherDto addNewTeacher(TeacherDto teacherDto); TeacherDto updateTeacher(TeacherDto teacherDto, int teacherId); TeacherDto deleteTeacher(int teacherId).*

3.2. Склад файлів *frontend* частини додатку

Розробка *frontend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA.* Назву проекту було обрано *journal_client,* папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_client.* Мова проекту – *Java,* інструмент для автоматизації зборки – *Maven.* Назву пакету проекту обрано – *com.makariev.journal_client.* Версію мови програмування *Java* було обрано одинадцяту. Тип архівування проекту було обрано *Jar.* Після натискання кнопки «*Next*» відкривається вікно для додавання залежностей до проекту та вибір версії фреймворку *Spring Boot.* Було обрано версію 2.7.5 та додано наступні залежності: *Spring Boot DevTools, Lombok, Spring Web, Thymeleaf.* Далі був процес створення проекту та автоматичного завантаження бібліотек для залежностей, що були обрані.

Папка *journal_client* містить автоматично створені при створенні проекту папки: *.git, .idea, .mvn, src* та *target.* Також вона має автоматично створені файли: *.gitignore.txt, HELP.md, journal_project.iml, mvnw, mvnw.cmd, pom.xml.* У ході розробки клієнтської частини програмного забезпечення системи аналізу та обліку успішності школярів були створені наступні пакети та файли у папці *src\main\java\com\makariev\journal_client*:

– пакет *config* призначений для конфігураційних файлів*: CorsFilter.java* – визначає налаштування клієнта для можливості звертатися то сервера*; WebConfig.java* – зв'язує *URL* посилання із сторінками сайту*;*

– пакет *controller* містить у собі класи-контролери. Для *frontend* частини вони потрібні для прийому *URL* запитів до сайту та виведення потрібної сторінки користувачеві на екран. У ньому містяться файли*: AnalysisController.java, CityController.java, LessonController.java, MarkController.java, OlympiadMarkController.java, OlympiadController.java, PupilController.java, SchoolClassController.java, SchoolController.java, SubjectController.java, TeacherController.java; ОПИС*

– пакет *dto* містить файли *DTO*-об'єктів*: CityDto.java, LessonDto.java, MarkDto.java, OlympiadDto.java, OlympiadMarkDto.java, PupilDto.java, PupilRatingDto.java, SchoolClassDto.java, SchoolClassRatingDto.java, SchoolDto.java, SchoolRatingDto.java, SubjectDto.java, TeacherDto.java, LessonPupilsMarksDto.java, PupilLessonMarkDto.java, MarkSubjectDto.java;*

– пакет *service* містить у собі класи-сервіси*: CityService.java, LessonService.java, MarkService.java, OlympiadService.java, OlympiadMarkService.java, PupilService.java, SchoolClassService.java, SchoolService.java, SubjectService.java, TeacherService.java, CityServiceImpl.java, LessonServiceImpl.java, MarkServiceImpl.java, OlympiadServiceImpl.java, OlympiadMarkServiceImpl.java, PupilServiceImpl.java, SchoolClassServiceImpl.java, SchoolServiceImpl.java, SubjectServiceImpl.java, TeacherServiceImpl.java;*

– пакет *util* містить у собі допоміжні класи. Наприклад, є файли, що містять в собі назви сторінок сайту та їх *URL* посилання. У ньому містяться файли*:*

AnalysisMappings.java, LessonMappings.java, MarkMappings.java, OlympiadMappings.java, OlympiadMarkMappings.java, PupilMappings.java, SchoolMappings.java, SchoolClassMappings.java, SubjectMappings.java, TeacherMappings.java, UrlArgumentsHelper.java, ViewNames.java.

Файл *AnalysisController.java* містить у собі реалізацію класу-контролеру *AnalysisController* та призначений для обробки запитів, пов'язаних з аналізом. У ньому описані наступні методи: *String resolveAnalysisSchoolSearchView(Model model); String resolveAnalysisSchoolResultViewByCityId(Model model, @RequestParam(value = "cityId", required = false) Integer cityId); String resolveAnalysisSchoolClassSearchView(Model model); String resolveAnalysisSchoolClassResultViewBySchoolId(Model model, @RequestParam("schoolId") int schoolId); String resolveAnalysisPupilSearchView(Model model); String resolveAnalysisPupilResultViewBySchoolId(Model model, @RequestParam("classId") int schoolClassId).*

Файл *CityController.java* містить у собі реалізацію класу-контролеру *CityController* та призначений для обробки запитів, пов'язаних з пошуком міст. У ньому описані наступні методи: *String resolveCityResultsView(Model model); String resolveCityEditView(Model model, @RequestParam("cityId") int cityId); String resolveCityEditAddView(Model model); String processEditOrAddCity ThenRedirect(@ModelAttribute("city") CityDto cityDto); String processDeleteCity(@RequestParam("cityId") int cityId).*

Файл *HomeController.java* містить у собі реалізацію класу-контролеру *HomeController* та призначений для обробки запитів, пов'язаних з головною сторінкою. У ньому є опис методу *String resolvePupilView(Model model).*

Файл *LessonController.java* містить у собі реалізацію класу-контролеру *LessonController* та призначений для обробки запитів, пов'язаних з уроками. У ньому описані наступні методи: *String resolveLessonEditView(Model model, @RequestParam("lessonId") int lessonId); String resolveLessonEditAddView(Model model, @RequestParam("classId") int classId, @RequestParam("teacherId") int*

teacherId); String processLessonEditOrAdd(Model model, @ModelAttribute LessonPupilsMarksDto lessonPupilsMarksDto); String processLessonDelete(int lessonId).

Файл *MarkController.java* містить у собі реалізацію класу-контролеру *MarkController* та призначений для обробки запитів, пов’язаних з оцінками. У ньому описані наступні методи: *String resolveMarkSearchView(Model model); String resolveClassMarks(Model model, @RequestParam("subjectId") int subjectId, @RequestParam("classId") int schoolClassId).*

Файл *OlympiadMarkController.java* містить у собі реалізацію класу-контролеру *OlympiadMarkController* та призначений для обробки запитів, пов’язаних з оцінками за олімпіади. У ньому описані наступні методи: *String resolveOlympiadMarkEditView(Model model, @RequestParam("olympiadMarkId") int olympiadMarkId); String resolveOlympiadMarkEditAddView(Model model, @RequestParam("olympiadId") int olympiadId); String processOlympiadMarkEditOrAdd(@ModelAttribute("olympiadMark") OlympiadMarkDto olympiadMarkDto); String processOlympiadMarkDelete(int olympiadMarkId).*

Файл *OlympiadController.java* містить у собі реалізацію класу-контролеру *OlympiadController* та призначений для обробки запитів, пов’язаних з пошуком та редагування олімпіад. У ньому описані наступні методи: *String resolveOlympiadSearchView(Model model); String resolveOlympiadResultsView(Model model, @RequestParam("dateFrom") Date dateFrom, @RequestParam("dateTo") Date dateTo); String resolveOlympiadDetailsView(Model model, @RequestParam("olympiadId") int olympiadId); String resolveOlympiadEditView(Model model, @RequestParam ("olympiadId") int olympiadId); String resolveOlympiadEditAddView(Model model); String processOlympiadEditOrAdd(@ModelAttribute("olympiad") OlympiadDto olympiadDto); String processOlympiadDelete(@RequestParam("olympiadId") int olympiadId).*

Файл *PupilController.java* містить у собі реалізацію класу-контролеру *PupilController* та призначений для обробки запитів, пов’язаних з учнями. У ньому описані наступні методи: *String resolvePupilSearchView(Model model); ModelAndView getAllPupilsContainingName(@ModelAttribute("pupil") PupilDto pupilDto);*

ModelAndView getAllPupilsByClassId(@ModelAttribute("class") SchoolClassDto schoolClassDto); String resolvePupilSearchResultsView(Model model); String resolvePupilDetails(Model model, @RequestParam("pupilId") int pupilId); String resolvePupilEditView(Model model, @RequestParam("pupilId") int pupilId); String resolvePupilEditAddView(Model model); String processPupilEditOrAdd(@ModelAttribute("pupil") PupilDto pupilDto); String processPupilDelete(@RequestParam("pupilId") int pupilId).

Файл *SchoolClassController.java* містить у собі реалізацію класу-контролеру *SchoolClassController* та призначений для обробки запитів, пов’язаних з класами. У ньому описані наступні методи: *String resolveSchoolClassDetailsView(Model model, @RequestParam("classId") int classId); String resolveSchoolClassEditView(Model model, @RequestParam("classId") int classId); String resolveSchoolClassEditAddView(Model model); String processSchoolClassEditOrAdd(@ModelAttribute("class") SchoolClassDto schoolClassDto); String processSchoolClassDelete(@RequestParam("classId") int classId).*

Файл *SchoolController.java* містить у собі реалізацію класу-контролеру *SchoolController* та призначений для обробки запитів, пов’язаних зі школами. У ньому описані наступні методи: *String resolveSchoolSearchView(Model model); String resolveSchoolResultsView(Model model, @RequestParam("cityId") int cityId); String resolveSchoolDetailsView(Model model, @RequestParam("schoolId") int schoolId); String resolveSchoolEditView(Model model, @RequestParam("schoolId") int schoolId); String resolveSchoolEditAddView(Model model); String processSchoolEditOrAdd(@ModelAttribute("school") SchoolDto schoolDto); String processSchoolDelete(@RequestParam("schoolId") int schoolId).*

Файл *SubjectController.java* містить у собі реалізацію класу-контролеру *SubjectController* та призначений для обробки запитів, пов’язаних з предметами. У ньому описані наступні методи: *String resolveSubjectResultsView(Model model); String resolveSubjectEditView(Model model, @RequestParam("subjectId") int subjectId); String resolveSubjectEditAddView(Model model); String*

processEditOrAddSubjectThenRedirect(@ModelAttribute("subject") SubjectDto subjectDto); String processDeleteSubject(@RequestParam("subjectId") int subjectId).

Файл *TeacherController.java* містить у собі реалізацію класу-контролеру *TeacherController* та призначений для обробки запитів, пов'язаних з вчителями. У ньому описані наступні методи: *String resolveTeacherSearchView(Model model); String resolveTeacherResultsSchoolView(Model model, @ModelAttribute("urlArgs") UrlArgumentsHelper urlArgumentsHelper); String resolveTeacherResultsNameView (Model model, @ModelAttribute("teacher") TeacherDto teacherDto); String resolveTeacherDetailsView(Model model, @RequestParam("teacherId") int teacherId); String resolveTeacherEditView(Model model, @RequestParam("teacherId") int teacherId); String resolveTeacherEditAddView (Model model); String processTeacherEdit OrAdd(@ModelAttribute("teacher") TeacherDto teacherDto); String processTeacher Delete(@RequestParam("teacherId") int teacherId).*

Пакет *service* містить у собі класи-сервіси. Для кожного *Entity*-класу створений свій клас-сервіс та інтерфейс до нього. Класи-сервіси мають у кінці назви *Impl*, що означає що даний клас є реалізацією інтерфейсу.

Файл *CityService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *City – CityServiceImpl.java.* У ньому описані наступні методи: *CityDto findById(int cityId); List<CityDto> findAll();CityDto addNewCity(CityDto cityDto); CityDto updateCity(CityDto cityDto, int cityId); CityDto deleteCity(int cityId).*

Файл *LessonService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Lesson – LessonServiceImpl.java.* У ньому описані наступні методи: *List<LessonDto> findAllByTeacher_Subject_IdAndSchoolClass_Id(int subjectId, int schoolClassId); LessonDto findById(int lessonId); List<LessonDto> findAllByTeacher_Subject_IdAndSchoolClass_IdAndSetAdditionalInfo(int subjectId, int schoolClassId); List<LessonDto> findAllByTeacher_Id(int teacherId); LessonDto addNewLesson(LessonDto lessonDto); LessonDto updateLesson(LessonDto lessonDto, int lessonId); LessonDto deleteLesson(int lessonId).*

Файл *MarkService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Mark – MarkServiceImpl.java.* У ньому описані наступні методи: *List<MarkDto> findAllByPupil_Id(int pupilId); List<MarkDto> findAllByLesson_Teacher_ Subject_IdAndPupil_Id(int subjectId, int pupilId); List<MarkDto> findAllByLesson_Id(int lessonId); List<MarkDto> findAllByLesson_Teacher_Subject_IdAndPupil_ SchoolClass_Id(int subjectId, int schoolClassId); MarkDto addNewMark(MarkDto markDto); List<MarkDto> addNewMarks(List<MarkDto> markDtos, int lessonId); MarkDto updateMark(MarkDto markDto, int markId); MarkDto deleteMark(int markId).*

Файл *OlympiadMarkService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *OlympiadMark – OlympiadMarkServiceImpl.java.* У ньому описані наступні методи: *OlympiadMarkDto findById(int olympiadMarkId); List<OlympiadMarkDto> findAllByPupil_Id(int pupilId); List<OlympiadMarkDto> findAllByOlympiad_Id(int olympiadId); OlympiadMarkDto addNewOlympiadMark(OlympiadMarkDto olympiadMarkDto); OlympiadMarkDto updateOlympiadMark(OlympiadMarkDto olympiadMarkDto, int olympiadMarkId); OlympiadMarkDto deleteOlympiadMark(int olympiadMarkId).*

Файл *OlympiadService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Olympiad – OlympiadServiceImpl.java.* У ньому описані наступні методи: *OlympiadDto findById(int olympiadId); List<OlympiadDto> findAllById(List<Integer> olympiadIds); List<OlympiadDto> findAll();List<OlympiadDto> findAllByDateBetween(Date from, Date to); OlympiadDto addNew Olympiad(OlympiadDto olympiadDto); OlympiadDto update Olympiad(OlympiadDto olympiadDto, int olympiadId); OlympiadDto deleteOlympiad(int olympiadId).*

Файл *PupilService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Pupil – PupilServiceImpl.java.* У ньому описані наступні методи: *PupilDto findById(int pupilId); List<PupilDto> findAllBySchoolClass_School_Id(int schoolId); List<PupilDto> findAllBySchoolClass_Id(int schoolClassId); List<PupilDto> findAllByFirstName*

ContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase(String firstName, String lastName, String middleName); List<PupilDto> findAllByOlympiadId(int olympiadId); List<PupilDto> findAllByOlympiadId(int olympiadId); List<PupilRatingDto> findAllStatisticsBySchoolClass_Id(int schoolClassId); PupilDto addNewPupil(PupilDto pupilDto); PupilDto updatePupil(PupilDto pupilDto, int pupilId); PupilDto deletePupil(int pupilId).

Файл *SchoolClassService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *SchoolClass – SchoolClassServiceImpl.java.* У ньому описані наступні методи: *SchoolClassDto findById(int schoolClassId); List<SchoolClassDto> findAllBySchool_Id(int schoolId); List<SchoolClassRatingDto> findAllStatisticsBy School_Id(int schoolId); SchoolClassDto addNewSchoolClass(SchoolClassDto schoolClassDto); SchoolClassDto updateSchoolClass(SchoolClassDto schoolClassDto, int schoolClassId); SchoolClassDto deleteSchoolClass(int schoolClassId).*

Файл *SchoolService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *School – SchoolServiceImpl.java.* У ньому описані наступні методи: *List<SchoolDto> findAllByCity_Id(int cityId); SchoolDto findById(int schoolId); List<SchoolRatingDto> findAllSchoolStatisticsByCity_id(int cityId); List<SchoolRatingDto> findAllSchoolStatistics();SchoolDto addNewSchool(SchoolDto schoolDto); SchoolDto updateSchool(SchoolDto schoolDto, int schoolId); SchoolDto deleteSchool(int schoolId).*

Файл *SubjectService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Subject – SubjectServiceImpl.java.* У ньому описані наступні методи: *SubjectDto findById(int subjectId); SubjectDto addNewSubject(SubjectDto subject); List<SubjectDto> findAll();SubjectDto updateSubject(SubjectDto subjectDto, int subjectId); SubjectDto deleteSubject(int subjectId).*

Файл *TeacherService.java* містить елементи для створення інтерфейсу класу-сервіса для *Entity*-класу *Teacher – TeacherServiceImpl.java.* У ньому описані наступні методи: *TeacherDto findById(int teacherId); List<TeacherDto> findAllBySchool_Id(int schoolId); List<TeacherDto> findAllBySchoolClass_Id(int schoolClassId); TeacherDto*

findBySchool Class_IdAndSubject_Id(int schoolClassId, int subjectId); List<TeacherDto> findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase(String firstName, String lastName, String middleName); TeacherDto addNewTeacher(TeacherDto teacherDto); TeacherDto updateTeacher(TeacherDto teacherDto, int teacherId); TeacherDto deleteTeacher(int teacherId).

Також у ході розробки програмного засобу були створені папки *src\main\resources*: *static* та *templates*. Папка *static* містить у собі допоміжні файли, що потрібні для коректного відображення сайту. Наприклад, файли стилів, потрібні зображення, іконки тощо.

Папка *templates* містить у собі файли сторінок сайту для сторінок аналізу: *analysis-class-results.html, analysis-class-search.html, analysis-pupil-results.html, analysis-pupil-search.html, analysis-school-results.html, analysis-school-search.html;* для роботи з містами: *cities-edit.html, cities-results.html;* для роботи з класами: *class-details.html, class-edit.html;* для роботи з оцінками: *lesson-edit.html, marks-class.html, marks-search.html;* для роботи з олімпіадами: *olympiad-details.html, olympiad-edit.html, olympiad-results.html, olympiad-search.html, olympiadmark-edit.html;* для роботи з учнями: *pupil-details.html, pupil-edit.html, pupil-results.html, pupil-search.html;* для роботи зі школами: *school-details.html, school-edit.html, school-results.html, school-search.html;* для роботи з предметами: *subject-edit.html, subject-results.html;* для роботи з вчителями: *teacher-details.html, teacher-edit.html, teacher-results.html, teacher-search.html.*

3.3. Розробка основних модулів *backend* частини додатку

Основна бізнес-логіка *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів зосереджена в класах-сервісах, а саме в класах, що містять їх реалізацію: *CityServiceImpl, LessonServiceImpl, MarkServiceImpl, OlympiadMarkServiceImpl, OlympiadServiceImpl, PupilServiceImpl, SchoolClassServiceImpl, SchoolServiceImpl*.

Клас-сервіс *CityServiceImpl* переймає поведінку інтерфейсу *CityService* та містить реалізацію усіх його методів. Він працює з класом-репозиторієм *CityRepository* та *mapper*-класом *CityMapper* та має наступні методи: *findById, findAll, addNewCity, updateCity, deleteCity.*

Клас-сервіс *LessonServiceImpl* переймає поведінку інтерфейсу *LessonService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *LessonRepository, SchoolClassRepository, TeacherRepository* та *mapper*-класом *LessonMapper* та має наступні методи: *findById, findAllByTeacher_Subject_Id AndSchoolClass_Id, findAllByTeacher_Id, addNewLesson, updateLesson, deleteLesson.*

Клас-сервіс *MarkServiceImpl* переймає поведінку інтерфейсу *MarkService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *MarkRepository, PupilRepository, LessonRepository* та *mapper*-класом *MarkMapper* та має наступні методи: *findAllByPupil_Id, findAllByLesson_Teacher_Subject_IdAndPupil _SchoolClass_Id, findAllByLesson_Teacher_Subject_IdAndPupil_Id, findAllByLesson_Id, addNewMark, addNewMarks, updateMark, deleteMark.*

Клас-сервіс *OlympiadMarkServiceImpl* переймає поведінку інтерфейсу *OlympiadMarkService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *OlympiadMarkRepository, PupilRepository, OlympiadRepository* та *mapper*-класом *OlympiadMarkMapper* та має наступні методи: *findById, findAllByPupil_Id, findAllByOlympiad_Id, addNewOlympiadMark, updateOlympiadMark, deleteOlympiadMark.*

Клас-сервіс *OlympiadServiceImpl* переймає поведінку інтерфейсу *OlympiadService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *OlympiadRepository, SubjectRepository* та *mapper*-класом *OlympiadMapper* та має наступні методи: *findById, findAll, findAllByDateBetween, addNewOlympiad, updateOlympiad, deleteOlympiad.*

Клас-сервіс *PupilServiceImpl* переймає поведінку інтерфейсу *PupilService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *PupilRepository, MarkRepository, SchoolClassRepository, OlympiadMarkRepository* та *mapper*-класом *PupilMapper* та має наступні методи: *findById,*

findAllBySchoolClass_School_Id, findAllBySchoolClass_Id, findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnoreCaseAndMiddleNameContainsIgnoreCase, findAllByOlympiadId, findAllStatisticsBySchoolClass_Id, addNewPupil, updatePupil, deletePupil.

Клас-сервіс *SchoolClassServiceImpl* переймає поведінку інтерфейсу *SchoolClassService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *PupilRepository, MarkRepository, SchoolClassRepository, OlympiadMarkRepository, SchoolRepository* та *mapper*-класом *SchoolClassMapper* та має наступні методи: *findById, findAllBySchool_Id, findAllStatisticsBySchool_Id, addNewSchoolClass, updateSchoolClass, deleteSchoolClass.*

Клас-сервіс *SchoolServiceImpl* переймає поведінку інтерфейсу *SchoolService* та містить реалізацію усіх його методів. Він працює з класами-репозиторіями *PupilRepository, MarkRepository, CityRepository, OlympiadMarkRepository, SchoolRepository* та *mapper*-класом *SchoolMapper* та має наступні методи: *findById, findAllByCity_Id, findAllSchoolStatisticsByCity_id, addSchoolClass, updateSchool, deleteSchool.*

3.3.1. Модуль пошуку інформації

При розробці модуля пошуку інформації було створено клас-сервіс *CityServiceImpl* із методами: *findById, findAll.*

Метод *findById* класу *CityServiceImpl* призначений для пошуку міста за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *CityRepository,* отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація, що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAll* класу *CityServiceImpl* призначений для пошуку всіх міст. Він не приймає жодних аргументів. Даний метод звертається до класу-репозиторію *CityRepository,* отримує від нього список *entity*-об’єктів, який переводить у список *DTO*-об’єктів за допомогою *mapper*-класу та повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *LessonServiceImpl* із методами: *findById, findAllByTeacher_Subject_IdAndSchool Class_Id, findAllByTeacher_Id.*

Метод *findById* класу *LessonServiceImpl* призначений для пошуку уроку за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *LessonRepository,* отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAllByTeacher_Subject_IdAndSchoolClass_Id* класу *LessonServiceImpl* призначений для пошуку уроків за *id* предмету та *id* класу і приймає їх за аргументи. Він звертається до класу-репозиторію *LessonRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByTeacher_Id* класу *LessonServiceImpl* призначений для пошуку уроків за *id* і приймає його за аргумент. Він звертається до класу-репозиторію *LessonRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *MarkServiceImpl* із методами: : *findAllByPupil_Id, findAllByLesson_Teacher_Subject _IdAndPupil_SchoolClass_Id, findAllByLesson_Teacher_Subject_IdAndPupil_Id, findAllByLesson_Id.*

Метод *findAllByPupil_Id* класу *MarkServiceImpl* призначений для пошуку оцінок за *id* учня і приймає його за аргумент. Він звертається до класу-репозиторію *MarkRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id* класу *MarkServiceImpl* призначений для пошуку оцінок за *id* предмета та *id* класу і приймає їх за аргументи. Він звертається до класу-репозиторію *MarkRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByLesson_Teacher_Subject_IdAndPupil_Id* класу *MarkServiceImpl* призначений для пошуку оцінок за *id* предмета та *id* учня і приймає їх за аргументи. Він звертається до класу-репозиторію *MarkRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByLesson_Id* класу *MarkServiceImpl* призначений для пошуку оцінок за *id* уроку і приймає його за аргумент. Він звертається до класу-репозиторію *MarkRepository,* отримує від нього список *entity*-об’єктів які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *OlympiadMarkServiceImpl* із методами: *findById, findAllByPupil_Id.*

Метод *findById* класу *OlympiadMarkServiceImpl* призначений для пошуку оцінки за олпімпіаду за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *OlympiadMarkRepository,* отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAllByPupil_Id* класу *OlympiadMarkServiceImpl* призначений для пошуку оцінок за олімпіаду за *id* учня і приймає його за аргумент. Він звертається до класу-репозиторію *OlympiadMarkRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *OlympiadServiceImpl* із методами: *findById, findAll, findAllByDateBetween.*

Метод *findById* класу *OlympiadServiceImpl* призначений для пошуку олімпіади за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *OlympiadRepository,* отримує від нього *entity*-об’єкт який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAll* класу *OlympiadServiceImpl* призначений для пошуку всіх олімпіад. Він не приймає жодних аргументів. Даний метод звертається до класу-репозиторію *OlympiadRepository,* отримує від нього список *entity*-об’єктів, який переводить у список *DTO*-об’єктів за допомогою *mapper*-класу та повертає його.

Метод *findAllByDateBetween* класу *OlympiadServiceImpl* призначений для пошуку олімпіад за двома датами і приймає їх за аргументи. Він звертається до класу-репозиторію *OlympiadRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *PupilServiceImpl* із методами: *findById, findAllBySchoolClass_School_Id, findAllBySchoolClass_Id, findAllByFirstNameContainsIgnoreCaseAndLastNameContains IgnoreCaseAndMiddleNameContainsIgnoreCase, findAllByOlympiadId.*

Метод *findById* класу *PupilServiceImpl* призначений для пошуку учня за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *PupilRepository,* отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAllBySchoolClass_School_Id* класу *PupilServiceImpl* призначений для пошуку учнів за *id* школою і приймає його за аргумент. Він звертається до класу-репозиторію *PupilRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllBySchoolClass_Id* класу *PupilServiceImpl* призначений для пошуку учнів за *id* класу і приймає його за аргумент. Він звертається до класу-репозиторію *PupilRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByFirstNameContainsIgnoreCaseAndLastNameContainsIgnore CaseAndMiddleNameContainsIgnoreCase* класу *PupilServiceImpl* призначений для пошуку учнів за ім’ям, прізвищем та по-батькові і приймає їх за аргументи. Він

звертається до класу-репозиторію *PupilRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

Метод *findAllByOlympiadId* класу *PupilServiceImpl* призначений для пошуку учнів за *id* олімпіади і приймає його за аргумент. Він звертається до класу-репозиторію *PupilRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *SchoolClassServiceImpl* із методами: *findById, findAllBySchool_Id.*

Метод *findById* класу *SchoolClassServiceImpl* призначений для пошуку класу за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *SchoolClassRepository,* отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAllBySchool_Id* класу *SchoolClassServiceImpl* призначений для пошуку класів за *id* школи і приймає його за аргумент. Він звертається до класу-репозиторію *SchoolClassRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля пошуку інформації було створено клас-сервіс *SchoolServiceImpl* із методами: *findById, findAllByCity_Id.*

Метод *findById* класу *SchoolServiceImpl* призначений для пошуку школи за *id* та приймає його як аргумент. Він звертається до класу-репозиторію *SchoolRepository* , отримує від нього *entity*-об’єкт, який шукається, та переводить його у *DTO*-об’єкт за допомогою *mapper*-класу. Якщо такого знайдено не було, повертається виключна ситуація що повідомляє про те, що об’єкт із таким *id* знайдено не було.

Метод *findAllByCity_Id* класу *SchoolServiceImpl* призначений для пошуку класів за *id* міста і приймає його за аргумент. Він звертається до класу-репозиторію

SchoolRepository, отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

3.3.2. Модуль додавання, зміни та видалення даних

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *CityServiceImpl* із методами: *addNewCity, updateCity, deleteCity.*

Метод *addNewCity* класу *CityServiceImpl* призначений для додавання нового міста. Він приймає *DTO*-об’єкт, який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *CityRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updateCity* класу *CityServiceImpl* призначений для зміни існуючого міста. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* міста. Метод звертається до класу-репозиторію *CityRepository,*щоб знайти об’єкт, який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *CityRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteCity* класу *CityServiceImpl* призначений для видалення існуючого міста. Він приймає *id* міста. Метод звертається до класу-репозиторію *CityRepository* щоб знайти об’єкт який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *CityRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *LessonServiceImpl* із методами: *addNewLesson, updateLesson, deleteLesson.*

Метод *addNewLesson* класу *LessonServiceImpl* призначений для додавання нового уроку. Він приймає *DTO*-об’єкт який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *LessonRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updateLesson* класу *LessonServiceImpl* призначений для зміни існуючого уроку. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* міста. Метод звертається до класу-репозиторію *LessonRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *LessonRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteLesson* класу *LessonServiceImpl* призначений для видалення існуючого уроку. Він приймає *id* міста. Метод звертається до класу-репозиторію *LessonRepository* щоб знайти об’єкт, який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *LessonRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *MarkServiceImpl* із методами: *addNewMark, addNewMarks, updateMark, deleteMark.*

Метод *addNewMark* класу *MarkServiceImpl* призначений для додавання нової оцінки. Він приймає *DTO*-об’єкт який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *MarkRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *addNewMarks* класу *MarkServiceImpl* призначений для додавання декількох нових оцінок. Він приймає список *DTO*-об’єктів, які потрібно додати, та *id* уроку. Метод спочатку звертається до класу-репозиторію *LessonRepository* та знаходить потрібний урок. Потім він звертається до класу-репозиторію *PupilRepository* та знаходить усіх учнів, яким потрібно присвоїти оцінки. Метод перетворює прийняті об’єкти в *entity*-об’єкти, змінює їх поля на потрібні та звертається до класу-репозиторію *MarkRepository* щоб додати їх до бази даних. Повертається список новостворених *entity*-об’єкт.

Метод *updateMark* класу *MarkServiceImpl* призначений для зміни існуючої оцінки. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* оцінки.

Метод звертається до класу-репозиторію *MarkRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *MarkRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteMark* класу *MarkServiceImpl* призначений для видалення існуючої оцінки. Він приймає *id* оцінки. Метод звертається до класу-репозиторію *MarkRepository* щоб знайти об’єкт який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *MarkRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *OlympiadMarkServiceImpl* із методами: *addNewOlympiadMark, updateOlympiadMark, deleteOlympiadMark.*

Метод *addNewOlympiadMark* класу *OlympiadMarkServiceImpl* призначений для додавання нової оцінки за олімпіаду. Він приймає *DTO*-об’єкт, який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *OlympiadMarkRepository* щоб додати його до бази даних. Також він викликає метод для виклику збереженої процедури у базі даних для перерахунку місця для кожного учасника олімпіади. Повертається новостворений *entity*-об’єкт.

Метод *updateOlympiadMark* класу *OlympiadMarkServiceImpl* призначений для зміни існуючої оцінки за олімпіаду. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* оцінки. Метод звертається до класу-репозиторію *OlympiadMarkRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *LessonRepository* щоб внести зміни до потрібного об’єкту в базі даних. Також він викликає метод для виклику збереженої процедури у базі даних для перерахунку місця для кожного учасника олімпіади. Повертає вже змінений об’єкт.

Метод *deleteOlympiadMark* класу *OlympiadMarkServiceImpl* призначений для видалення існуючої оцінки за олімпіаду. Він приймає *id* оцінки. Метод звертається до класу-репозиторію *OlympiadMarkRepository* щоб знайти об’єкт який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *OlympiadMarkRepository* щоб видалити знайдений об’єкт в базі даних. Також він викликає метод для виклику збереженої процедури у базі даних для перерахунку місця для кожного учасника олімпіади. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *OlympiadServiceImpl* із методами: *addNewOlympiad, updateOlympiad, deleteOlympiad.*

Метод *addNewOlympiad* класу *OlympiadServiceImpl* призначений для додавання нової олімпіади. Він приймає *DTO*-об’єкт, який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *OlympiadRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updateOlympiad* класу *OlympiadServiceImpl* призначений для зміни існуючої олімпіади. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* міста. Метод звертається до класу-репозиторію *LessonRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *OlympiadRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteOlympiad* класу *OlympiadServiceImpl* призначений для видалення існуючої олімпіади. Він приймає *id* міста. Метод звертається до класу-репозиторію *OlympiadRepository* щоб знайти об’єкт, який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *OlympiadRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *OlympiadServiceImpl* із методами: *addNewPupil, updatePupil, deletePupil.*

Метод *addNewPupil* класу *PupilServiceImpl* призначений для додавання нового учня. Він приймає *DTO*-об’єкт який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *PupilRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updatePupil* класу *PupilServiceImpl* призначений для зміни існуючого учня. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* учня. Метод звертається до класу-репозиторію *PupilRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *PupilRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deletePupil* класу *PupilServiceImpl* призначений для видалення існуючого учня. Він приймає *id* учня. Метод звертається до класу-репозиторію *PupilRepository* щоб знайти об’єкт, який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *PupilRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *SchoolClassServiceImpl* із методами: *addNewSchoolClass, updateSchoolClass, deleteSchoolClass.*

Метод *addNewSchoolClass* класу *SchoolClassServiceImpl* призначений для додавання нового класу. Він приймає *DTO*-об’єкт який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *SchoolClassRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updateSchoolClass* класу *SchoolClassServiceImpl* призначений для зміни існуючого класу. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* класу. Метод звертається до класу-репозиторію *SchoolClassRepository* щоб знайти об’єкт який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та

звертається до класу-репозиторію *SchoolClassRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteSchoolClass* класу *SchoolClassServiceImpl* призначений для видалення існуючого класу. Він приймає *id* класу. Метод звертається до класу-репозиторію *SchoolClassRepository* щоб знайти об’єкт, який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *SchoolClassRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

При розробці модуля додавання, зміни та видалення даних було створено клас-сервіс *SchoolServiceImpl* із методами: *addSchoolClass, updateSchool, deleteSchool.*

Метод *addSchoolClass* класу *SchoolServiceImpl* призначений для додавання нової школи. Він приймає *DTO*-об’єкт, який потрібно додати. Метод перетворює даний об’єкт в *entity*-об’єкт та звертається до класу-репозиторію *SchoolRepository* щоб додати його до бази даних. Повертається новостворений *entity*-об’єкт.

Метод *updateSchool* класу *SchoolServiceImpl* призначений для зміни існуючої школи. Він приймає *DTO*-об’єкт із даними, які потрібно змінити, та *id* класу. Метод звертається до класу-репозиторію *SchoolRepository* щоб знайти об’єкт, який потрібно змінити. Якщо такого не існує, повертається виключна ситуація. Поля знайденого об’єкту змінюються на поля отриманого *DTO*-об’єкту та звертається до класу-репозиторію *SchoolRepository* щоб внести зміни до потрібного об’єкту в базі даних. Повертає вже змінений об’єкт.

Метод *deleteSchool* класу *SchoolServiceImpl* призначений для видалення існуючої школи. Він приймає *id* школи. Метод звертається до класу-репозиторію *SchoolRepository* щоб знайти об’єкт який потрібно видалити. Якщо такого не існує, повертається виключна ситуація. Йде звернення до класу-репозиторію *SchoolRepository* щоб видалити знайдений об’єкт в базі даних. Повертає видалений об’єкт.

3.3.3. Модуль аналізу та обліку успішності школярів

При розробці модуля аналізу та обліку успішності школярів було створено клас-сервіс *SchoolServiceImpl* із методами: *findAllSchoolStatisticsByCity_id, findAllSchoolStatistics.*

Метод *findAllSchoolStatisticsByCity_id* класу *SchoolServiceImpl* призначений для пошуку статистики шкіл за *id* міста і приймає його за аргумент. Він звертається до різних класів-репозиторіїв і отримує від них інформацію потрібну для формування статистики. Потім у циклі кожній школі присвоюється статистика та формується *DTO*-об’єкт, який згодом повертається.

Метод *findAllSchoolStatistics* класу *SchoolServiceImpl* призначений для пошуку статистики всіх шкіл що знаходяться в базі даних і не приймає жодного аргументу. Він звертається до різних класів-репозиторіїв і отримує від них інформацію потрібну для формування статистики. Потім у циклі кожній школі присвоюється статистика та формується *DTO*-об’єкт, який згодом повертається.

При розробці модуля аналізу та обліку успішності школярів було створено клас-сервіс *SchoolClassServiceImpl* із методом *findAllStatisticsBySchool.* Метод *findAllStatisticsBySchool_Id* класу *SchoolClassServiceImpl* призначений для пошуку статистики класів за *id* школи і приймає його за аргумент. Він звертається до різних класів-репозиторіїв і отримує від них інформацію потрібну для формування статистики. Потім у циклі кожному класу присвоюється статистика та формується *DTO*-об’єкт, який згодом повертається.

При розробці модуля аналізу та обліку успішності школярів було створено клас-сервіс *OlympiadMarkServiceImpl* із методом *findAllByOlympiad_Id.* Метод *findAllByOlympiad_Id* класу *OlympiadMarkServiceImpl* призначений для пошуку оцінок за олімпіаду та формування статистичних даних за *id* олімпіади і приймає його за аргумент. Він звертається до класу-репозиторію *OlympiadMarkRepository,* отримує від нього список *entity*-об’єктів, які шукаються, та переводить їх у список *DTO*-об’єктів за допомогою *mapper*-класу і повертає його.

При розробці модуля аналізу та обліку успішності школярів було створено клас-сервіс *PupilServiceImpl* із методом *findAllStatisticsBySchoolClass_Id*. Метод *findAllStatisticsBySchoolClass_Id* класу *PupilServiceImpl* призначений для пошуку статистики учнів за *id* класу і приймає його за аргумент. Він звертається до різних класів-репозиторіїв і отримує від них інформацію потрібну для формування статистики. Потім у циклі кожному учню присвоюється статистика та формується *DTO*-об’єкт, який згодом повертається.

3.4. Розробка графічного інтерфейсу

Розробка графічного інтерфейсу програмного забезпечення системи аналізу та обліку успішності школярів було виконано з використанням технологій *HTML, Thymeleaf* та *Bootstrap.* Були використані стилі з бібліотеки *Bootstrap,* а також основні елементи сайту як навігаційне меню, шапка та низ сайту. Загальний вигляд сайту на головній веб-сторінці зображено на рис. 3.1.

Для веб-сторінок аналізу було створено шість файлів: *analysis-class-search.html, analysis-class-results.html, analysis-pupil-search.html, analysis-pupil-results.html, analysis-school-search.html, analysis-school-results.html.*

Файли *analysis-class-search.html* та *analysis-class-results.html* відповідають за графічний інтерфейс веб-сторінок аналізу успішності класів у школі. Перший містить у собі веб-сторінку пошуку, другий – результати пошуку. Вигляд даних веб-сторінок зображено на рис. 3.2 та рис. 3.3 відповідно.

Файли *analysis-pupil-search.html* та *analysis-pupil-results.html* відповідають за графічний інтерфейс веб-сторінок аналізу успішності учнів у класі. Перший містить у собі веб-сторінку пошуку, другий – результати пошуку. Вигляд веб-сторінки результатів пошуку представлено на рис. 3.4.

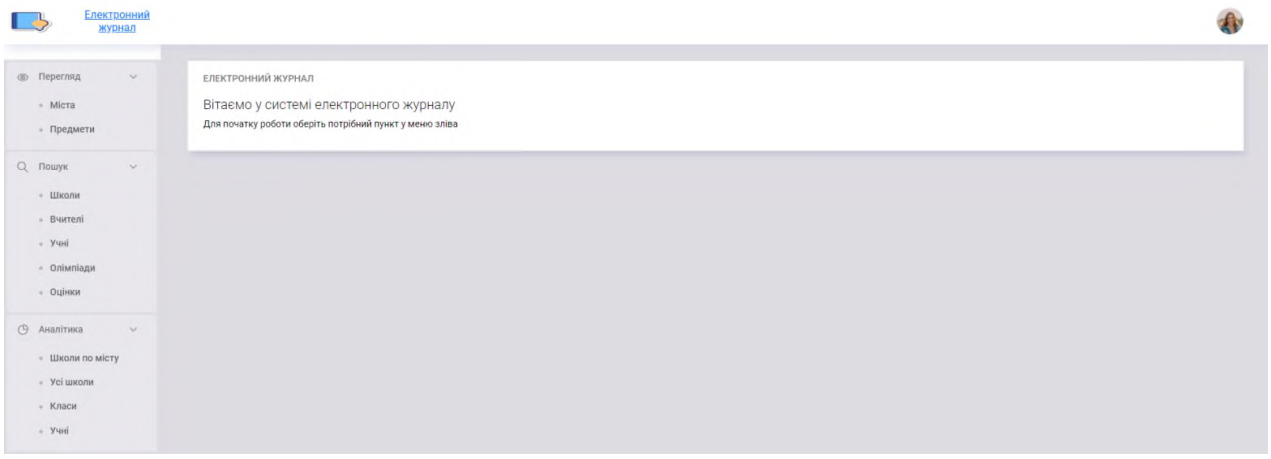

Рис 3.1. Головна веб-сторінка сайту

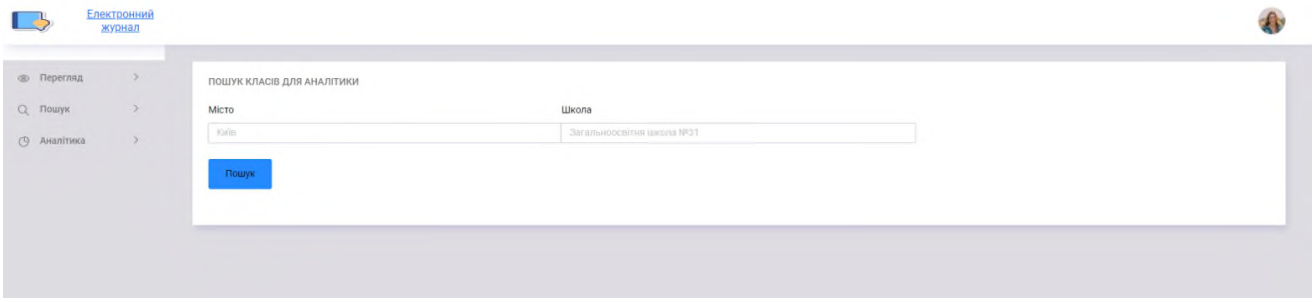

Рис 3.2. Веб-сторінка пошуку класів для аналізу

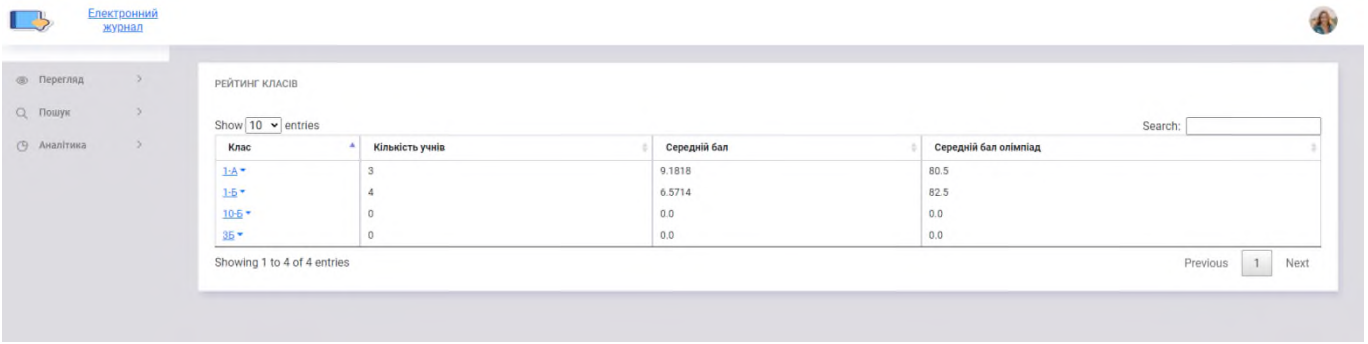

Рис 3.3. Веб-сторінка аналізу успішності класів

Рис 3.4. Веб-сторінка аналізу успішності учнів

Файли *analysis-school-search.html* та *analysis-school-results.html* відповідають за графічний інтерфейс веб-сторінок аналізу успішності шкіл. Перший містить у собі веб-сторінку пошуку, другий – результати пошуку. Вигляд веб-сторінки результатів пошуку представлено на рис. 3.5.

Для роботи з містами було створено два файли: *cities-edit.html* – веб-сторінка редагування міста; *cities-results.html* – веб-сторінка з усіма містами.

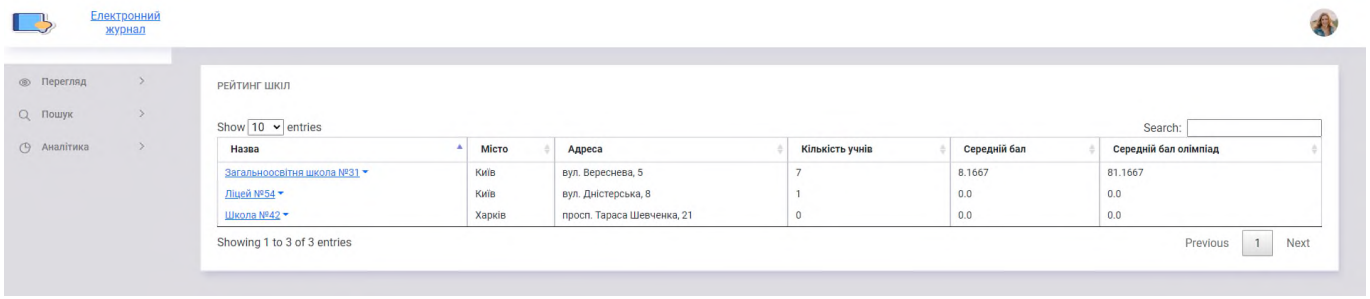

Рис 3.5. Веб-сторінка аналізу успішності шкіл

Для роботи з класами було створено два файли: *class-details.html* – веб-сторінка з детальною інформацією про клас, представлено на рис. 3.6; *class-edit.html* – веб-сторінка редагування класу.

Для роботи з оцінками було створено три файли: *lesson-edit.html* – веб-сторінка редагування уроку та оцінок до нього, представлено на рис. 3.7; *marks-class.html* – веб-сторінка з оцінками класу за предмет, зображено на рис. 3.8; *marks-search.html* – веб-сторінка пошуку оцінок.

Для роботи з олімпіадами було створено чотири файли: *olympiad-edit.html* – сторінка редагування олімпіади; *olympiad-details.html* – веб-сторінка з інформацією про олімпіаду, зображено на рис. 3.9; *olympiad-results.html* – веб-сторінка результатів пошуку олімпіад; *olympiad-search.html* – веб-сторінка пошуку олімпіад.

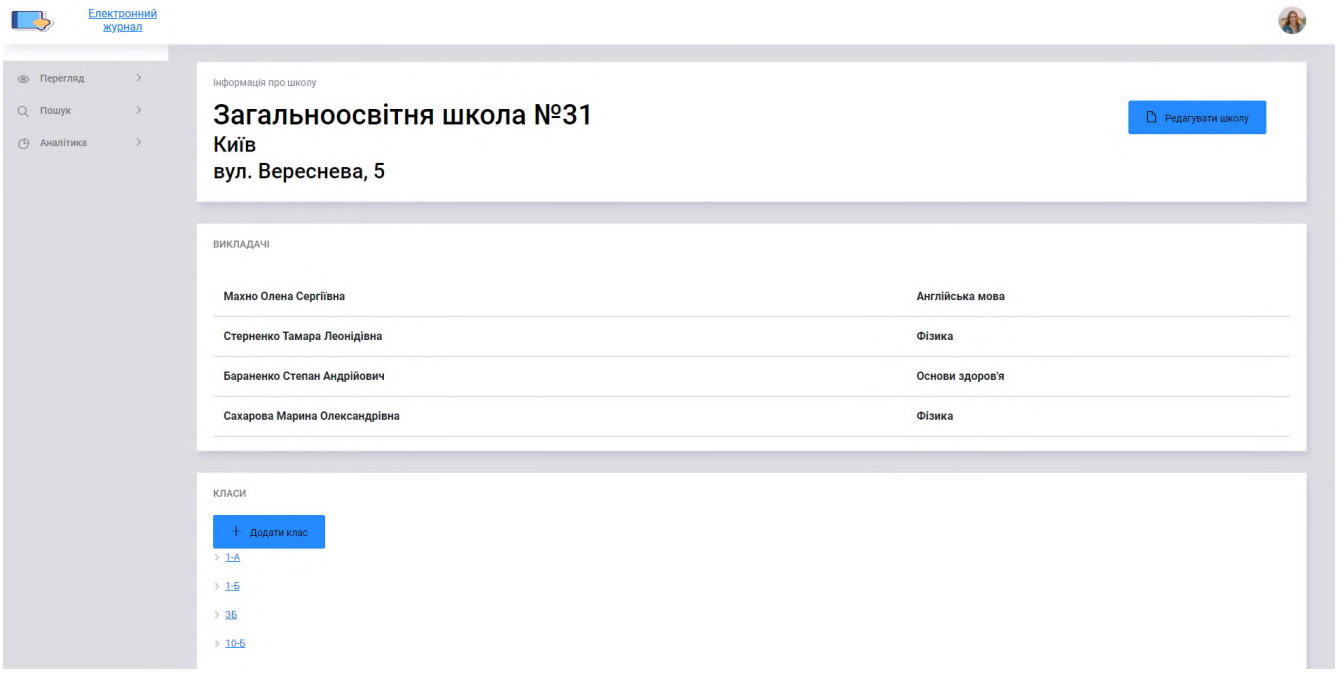

Рис 3.6. Веб-сторінка з детальною інформацією про клас

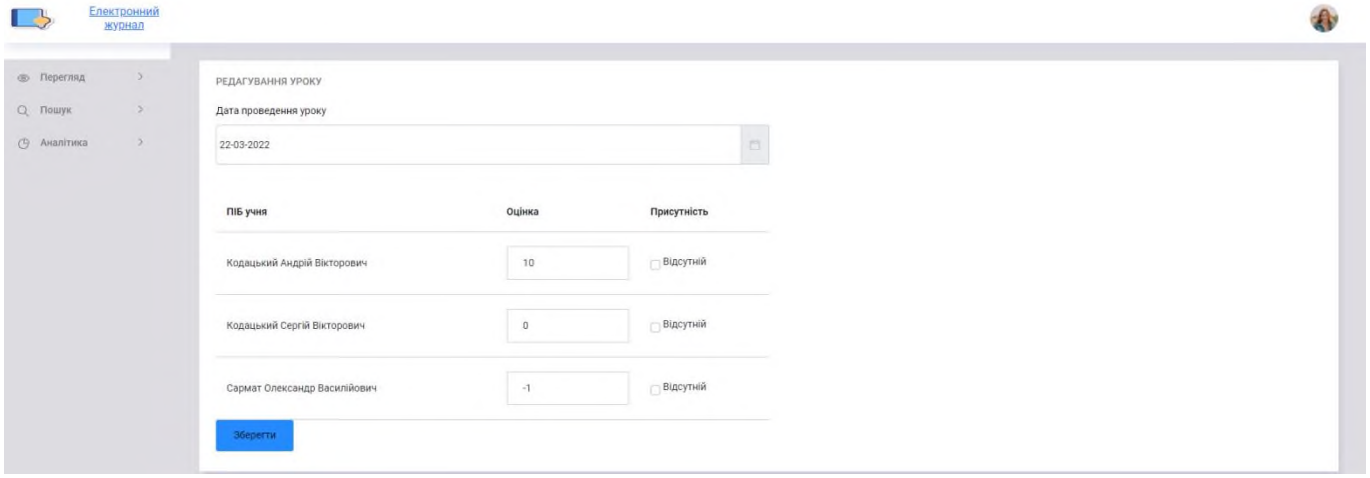

Рис 3.7. Веб-сторінка редагування уроку та оцінок до нього

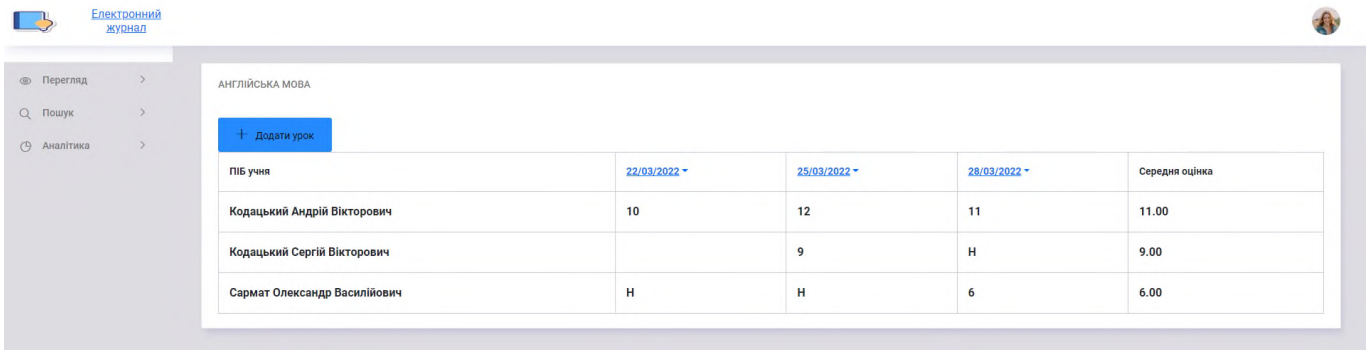

Рис 3.8. Веб-сторінка з оцінками класу за предмет

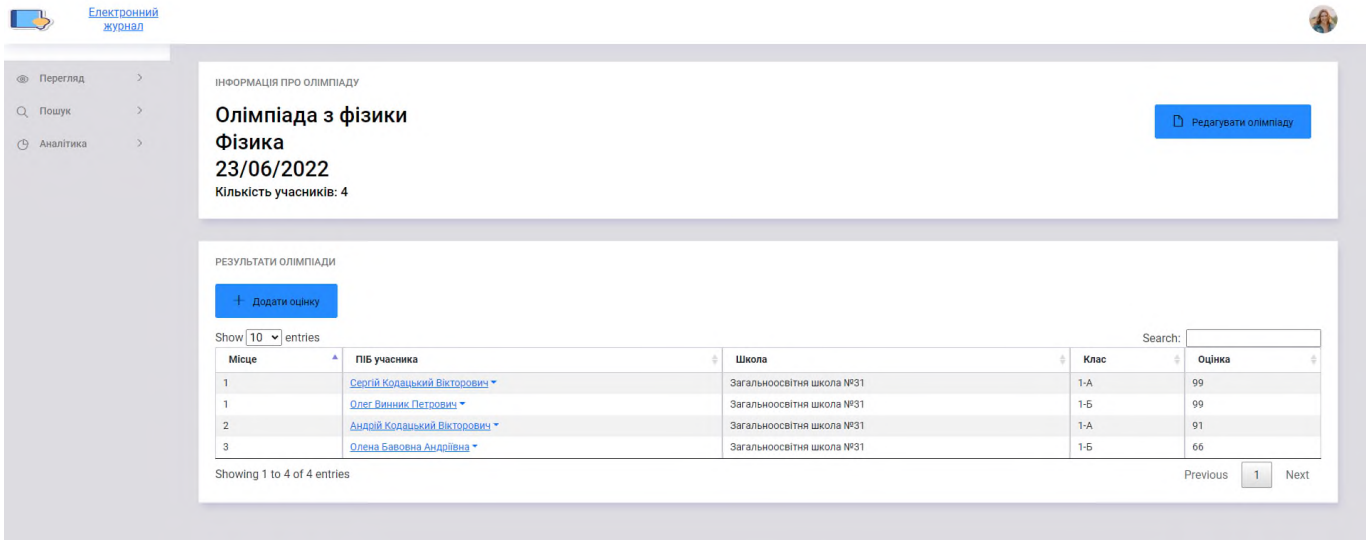

Рис 3.9. Веб-сторінка з інформацією про олімпіаду

Для роботи з учнями було створено чотири файли: *pupil-edit.html* – веб-сторінка редагування учня; *pupil-details.html* – веб-сторінка з інформацією про учня, зображено на рис. 3.10; *pupil-results.html* – веб-сторінка результатів пошуку учнів; *pupil-search.html* – веб-сторінка пошуку учнів.

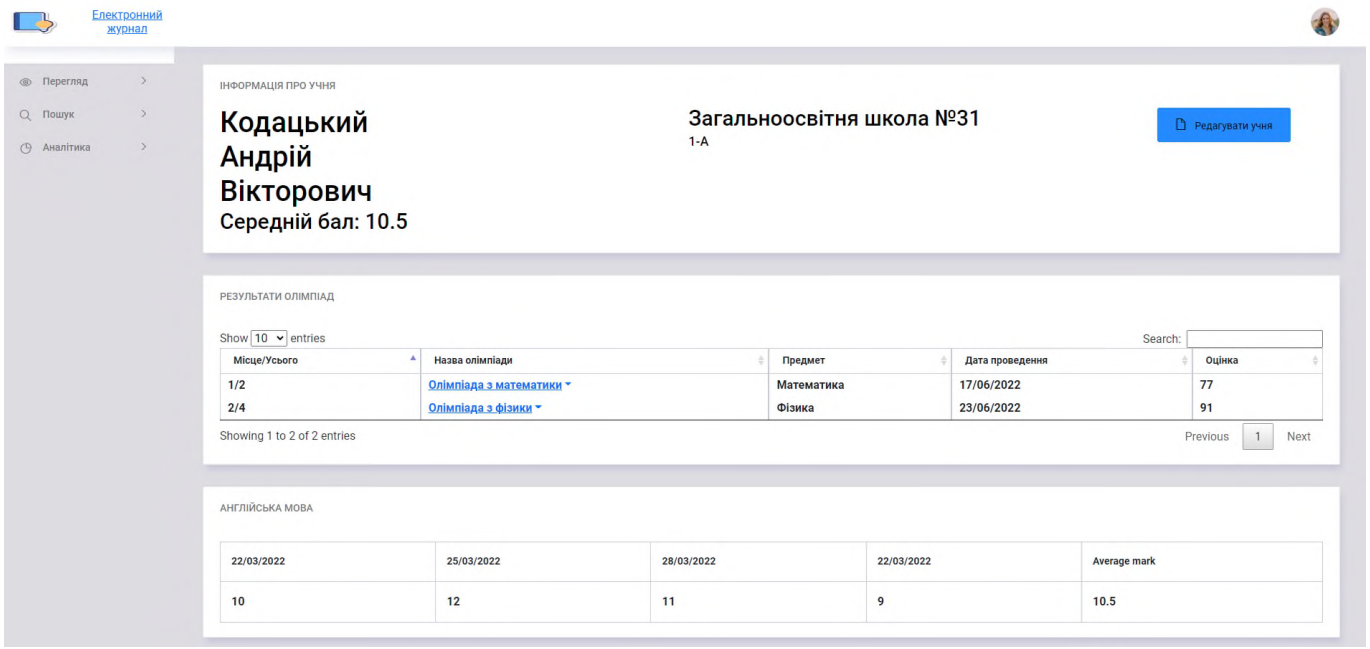

Рис 3.10. Веб-сторінка з інформацією про учня

Для роботи зі школами було створено чотири файли: *school-edit.html* – сторінка редагування школи; *school-details.html* – веб-сторінка з інформацією про

школу; *school-results.html* – веб-сторінка результатів пошуку шкіл*; school-search.html* – веб-сторінка пошуку шкіл.

Для роботи з вчителями було створено чотири файли: *teacher-edit.html* – веб-сторінка редагування вчителів; *teacher-details.html* – веб-сторінка з інформацією про вчителя; *teacher-results.html* – веб-сторінка результатів пошуку вчителів*; teacher-search.html* – веб-сторінка пошуку вчителів.

Для роботи з предметами було створено два файли: *subject-edit.html* – веб-сторінка редагування предмету; *subject-results.html* – веб-сторінка зі всіма предметами.

3.5. Висновки до розділу

У даному розділі було розглянуто склад файлів як для *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів, так і *frontend* частини. Було детально розглянуто файли та папки, які були створені автоматично та вручну. Було описано пакети, які є в проектах, та файли класів із перерахованими методами в них.

Розробка *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA.* Назву проекту було обрано *journal_project,* папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_project.* Назву пакету проекту – *com.makariev.journal_project.* Тип архівування проекту було обрано *Jar.* Було обрано версію *Spring Boot* 2.7.5 та додано наступні залежності: *Spring Boot DevTools, Lombok, Spring Web, Spring Data JPA, MySQL Driver*.

Розробка *frontend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA.* Назву проекту було обрано *journal_client,* папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_client.* Назву пакету проекту – *com.makariev.journal_client.* Тип архівування проекту було обрано *Jar.* Було обрано

версію *Spring Boot* 2.7.5 та додано наступні залежності: *Spring Boot DevTools, Lombok, Spring Web, Thymeleaf.*

Також у цьому розділі було розроблено основні модулі *backend* частини проекту: модуль пошуку інформації; модуль додавання, редагування та видалення записів; модуль аналізу та обліку успішності школярів. Були розроблені наступні класи-сервіси: *CityServiceImpl* для роботи з містами та має наступні методи: *findById, findAll, addNewCity, updateCity, deleteCity; LessonServiceImpl* для роботи з уроками та має наступні методи: *findById, findAllByTeacher_Subject_Id AndSchoolClass_Id, findAllByTeacher_Id, addNewLesson, updateLesson, deleteLesson; MarkServiceImpl* для роботи з оцінками та має наступні методи: *findAllByPupil_Id, findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id, findAllByLesson_Teache r_Subject_IdAndPupil_Id, findAllByLesson_Id, addNewMark, addNewMarks, updateMark, deleteMark; OlympiadMarkServiceImpl* для роботи з оцінками за олімпіади та має наступні методи: *findById, findAllByPupil_Id, findAllByOlympiad_Id, addNewOlympiadMark, updateOlympiadMark, deleteOlympiadMark;* для роботи з олімпіадами та має наступні методи: *findById, findAll, findAllByDateBetween, addNewOlympiad, updateOlympiad, deleteOlympiad; PupilServiceImpl* для роботи з учнями та має наступні методи: *findById, findAllBySchoolClass_School_Id, findAllBySchoolClass_Id, findAllByFirstNameContainsIgnoreCaseAndLastNameContains IgnoreCaseAndMiddleNameContainsIgnoreCase, findAllByOlympiadId, findAllStatistics BySchoolClass_Id, addNewPupil, updatePupil, deletePupil; SchoolClassServiceImpl* для роботи з класами та має наступні методи: *findById, findAllBySchool_Id, findAllStatisticsBySchool_Id, addNewSchoolClass, updateSchoolClass, deleteSchoolClass; SchoolServiceImpl* для роботи зі школами та має наступні методи: *findById, findAllByCity_Id, findAllSchoolStatisticsByCity_id, addSchoolClass, updateSchool, deleteSchool.*

У даному розділі також було розроблено графічний інтерфейс *frontend* частини веб-додатку. Були створені файли для веб-сторінок аналізу: *analysis-class-results.html, analysis-class-search.html, analysis-pupil-results.html, analysis-pupil-search.html, analysis-school-results.html, analysis-school-search.html;* для роботи з

містами: *cities-edit.html, cities-results.html;* для роботи з класами: *class-details.html, class-edit.html;* для роботи з оцінками: *lesson-edit.html, marks-class.html, marks-search.html;* для роботи з олімпіадами: *olympiad-details.html, olympiad-edit.html, olympiad-results.html, olympiad-search.html, olympiadmark-edit.html;* для роботи з учнями: *pupil-details.html, pupil-edit.html, pupil-results.html, pupil-search.html;* для роботи зі школами: *school-details.html, school-edit.html, school-results.html, school-search.html;* для роботи з предметами: *subject-edit.html, subject-results.html;* для роботи з вчителями: *teacher-details.html, teacher-edit.html, teacher-results.html, teacher-search.html.*

РОЗДІЛ 4
ДОСЛІДЖЕННЯ ШВИДКОДІЇ ВИКОНАННЯ ЗАПИТІВ ПРИ АНАЛІЗІ ТА ОБЛІКУ УСПІШНОСТІ ШКОЛЯРІВ

Мова програмування *Java* використовується для розробки великих веб-додатків, що працюють з базами даних, тому вона має багато фреймворків та інструментів для роботи з ними. Програмне забезпечення системи аналізу та обліку успішності школярів використовує фреймворк *Spring Data*. Даний засіб надає додатковий рівень абстракції поверх *ORM* специфікації *JPA(Java Persistence API).* За замовчуванням він використовує фреймворк *Hibernate,* у якості *ORM* провайдера, для здійснення запитів до бази даних.

Специфікація *JPA* – технологія, яка дозволяє зв'язати таблиці бази даних із *Java* класами. Сама по собі вона не є інструментом або фреймворком, а скоріше набором концепцій, які можуть бути реалізованим іншим інструментом. Дана специфікація була розроблена на основі стандартного іструменту *Java* для роботи за базами даних – *JDBC(Java DataBase Connectivity).*

Hibernate – один із самих популярних *ORM* фреймворків для роботи з базою даних, який реалізує специфікацію *JPA.* Тобто, без фреймворку *Hibernate* від специфікації *JPA* немає сенсу, так як можна назвати *JPA* інтерфейсом, а *Hibernate* як реалізацію цього інтерфейсу.

JDBC – стандартний інструмент *Java* для роботи з базами даних. Основним його недоліком є те, що код, який виходить у кінці, є дуже об'ємним та складно читаємими. Це стосується всіх операцій з базами даних: пошук, додавання, редагування та видалення. Саме для виправлення цього недоліку й була створена специфікація *JPA.*

Тобто, можна сказати, що *Spring Data* не працює з базою даних, а даний фреймворк використовує багато шарів із інших інструментів, що потенційно може сказатися на швидкодії взаємодії з базою даних та роботи додатку вцілому. Принцип роботи фреймворку *Spring Data* представлено на рис. 4.1.

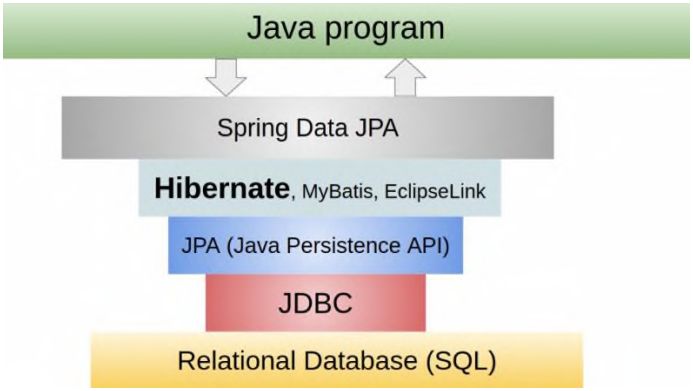

Рис 4.1. Принцип роботи фреймворку *Spring Data*

Для порівняння швидкодії виконання запитів було використано таблицю оцінок *marks*. Вона має наступні колонки: *id* – ідентифікаційний номер оцінки; *mark* – оцінка за урок; *pupil_id* – зовнішній ключ, що вказує на поле *id* у таблиці учнів *pupils*; *lesson_id* – зовнішній ключ, що вказує на поле *id* у таблиці уроків *lessons*. Для проведення вимірів швидкодії було використано *JMH(Java Microbenchmark Harness).*

JMH – набор бібліотек для тестування швидкодії окремих невеликих функцій. У даному випадку було проведено виміри швидкодії додавання запису до таблиці бази даних за допомогою інструменту *JDBC* та *Spring Data.* Для реалізації цього було зроблено виміри на основі десяти ітерацій для кожного засобу та пораховано середній час за який виконується одна ітерація. Час однієї ітерації було виміряно у наносекундах для отримання більш точних даних. Було задано запит «*INSERT INTO marks(id, mark, pupil_id, lesson_id)*». Результати тестування були виведені у консоль та мали наступний вигляд:

 # Measurement: 10 iterations, 10 s each

 # Benchmark mode: Average time, time/op

 # Benchmark: demo.benchmark.SpringDataJpaBenchmarking.springDataJpaQuery

Iteration 1: 46131,695 ns/op

Iteration 2: 46466,689 ns/op

Iteration 3: 46123,271 ns/op

Iteration 4: 46758,502 ns/op

Iteration 5: 46482,808 ns/op

Iteration 6: 46371,979 ns/op

Iteration 7: 46622,712 ns/op

Iteration 8: 46520,822 ns/op

Iteration 9: 46601,090 ns/op

Iteration 10: 46587,256 ns/op

Result ".benchmark.SpringDataJpaBenchmarking.springDataJpaQuery":

46466,682 ±(99.9%) 312,457 ns/op [Average]

(min, avg, max) = (46123,271, 46466,682, 46758,502)

 # Measurement: 10 iterations, 10 s each

 # Benchmark mode: Average time, time/op

 # Benchmark: demo.benchmark.JdbcBenchmarking.jdbcQuery

Iteration 1: 9072,540 ns/op

Iteration 2: 9102,913 ns/op

Iteration 3: 9106,280 ns/op

Iteration 4: 9114,262 ns/op

Iteration 5: 9132,792 ns/op

Iteration 6: 9046,303 ns/op

Iteration 7: 9100,860 ns/op

Iteration 8: 9080,942 ns/op

Iteration 9: 9033,891 ns/op

Iteration 10: 9062,018 ns/op

Result "demo.benchmark.JdbcBenchmarking. JdbcQuerydemo ":

9085,280 ±(99.9%) 47,741 ns/op [Average]

(min, avg, max) = (9033,891, 9085,280, 9132,792)

В таблиці 4.1 наведено дані про швидкість виконання запитів до таблиць з використанням різних інструментів.

Табл. 4.1.

Час виконання запитів до таблиць з використанням різних інструментів.

Номер ітерації	*Spring Data*, наносекунди	*JDBC*, наносекунди
1	46131,695	9072,540
2	46466,689	9102,913
3	46123,271	9106,280
4	46758,502	9114,262
5	46482,808	9132,792
6	46371,979	9046,303
7	46622,712	9100,860
8	46520,822	9080,942
9	46601,090	9033,891
10	46587,256	9062,018
Середній час	46466,682	9085,280

Середній час виконання однієї ітерації з використанням *JDBC* – 9085,280 наносекунд, коли результат *Spring Data* – 46466,628 наносекунд. Із цього можна зробити висновок, що *JDBC* швидше виконує запити, порівнянно з *Spring Data*, у 5,11 разів. Але, отриманні дані можуть різнитися в залежності від апаратної частини ком'ютера або сервера. На них можуть впливати такі параметри: тактова частота процесору, розмір кешів процесору, швидкість зчитування та запису жорсткого диску, тактова частота оперативної пам'яті. А отже, різниця між швидкодією *JDBC* та *Spring Data* може бути меншою, якщо комп'ютер або сервер має більш сучасні та потужні компоненти апаратного забезпечення.

Проте, не зважаючи на отримані результати, *Spring Data* є більш сучасним та популярним засобом взаємодії з базою даних, ніж *JDBC,* так як технологія *JPA* зменшує час на написання коду розробниками.

4.1. Висновки до розділу

У даному розділі було розглянуто інструменти для роботи з базами даних у мові програмування *Java*. Було детально досліджено роботу фреймворку *Spring Data* та слоїв, які він використовує для роботи з базами даних: *ORM* фреймворк *Hibernate*, специфікація *JPA*, стандартний інструмент *Java – JDBC*.

Також у цьому розділі було порівняно швидкодію виконання запитів до бази даних із використанням фреймворку *Spring Data* та стандартного інструменту *Java – JDBC*. Для порівняння швидкодії виконання запитів було використано таблицю оцінок *marks*. Вона має наступні колонки: *id* – ідентифікаційний номер оцінки; *mark* – оцінка за урок; *pupil_id* – зовнішній ключ, що вказує на поле *id* у таблиці учнів *pupils*; *lesson_id* – зовнішній ключ, що вказує на поле *id* у таблиці уроків *lessons*. Для проведення вимірів швидкодії було використано *JMH(Java Microbenchmark Harness).*

Було зроблено виміри на основі десяти ітерацій для кожного засобу та пораховано середній час за який виконується одна ітерація. Час однієї ітерації було виміряно у наносекундах для отримання більш точних даних

У результаті порівняння швидкодіїї двох засобів взаємодії з базами даних було встановлено, що середній час виконання однієї ітерації з використанням *JDBC* – 9085,280 наносекунд, а результат *Spring Data* – 46466,628 наносекунд. Тобто *JDBC* виявився швидшим за *Spring Data* у 5,11 разів.

Проте, не зважаючи на отримані результати, *Spring Data* є більш сучасним та популярним засобом взаємодії з базою даних, ніж *JDBC*, так як технологія *JPA* зменшує час на написання коду розробниками.

ВИСНОВКИ

Кваліфікаційна робота присвячена тематиці системі аналізу та обліку успішності школярів. У ході виконання кваліфікаційної роботи було досліджено існуюче програмне забезпечення систем аналізу та обліку успішності школярів, засоби *backend* та *frontend* розробки. Було спроектовано функції та основні класи *backend* та *frontend* частин додатку, а також функції аналізу та обліку успішності школярів. Було розроблено основно модулі *backend* частини додатку та графічний інтерфейс *frontend* частини додатку. Також було виконано дослідження швидкодії виконання запитів при аналізі та обліку успішності школярів.

У першому розділі було зроблено огляд існуючого програмного забезпечення систем аналізу та обліку успішності школярів, а саме було розглянуто сайти державної системи електронних журналів і щоденників «e-*journal.iea.gov.ua*», електронні щоденники та журнали «*e-schools.info*», електронний журнал та електронний щоденник – *online* систему для навчального процесу «*ukrschools.com.ua*». Було проаналізовано основні функції сайтів, їх дизайн, основні направлення, їх переваги та недоліки.

Було розглянуто найпопулярніші засоби для розробки *backend* частини веб-додатків, а саме: фреймворк *Spring Boot,* платформам *Node.js* та фреймворк *Django.* Було проаналізовано основні напрямки цих інструментів, їх переваги та недоліки.

Фреймворк *Spring Boot* в основному використовується для розробки великих веб-додатків із мікросервісною архітектурою. Він також має всі переваги мови програмування *Java*, так як даний інструмент написаний цією мовою.

Платформа *Node.js* за рахунок системи модулів може використовуватися для побудови веб-додатків із мікросервісної архітектури. За рахунок мови програмування *JavaScript* код є компактним. Також спільнота *Node.js* є одною із самих масштабних, а отже можна розраховувати на готові рішення сторонніх розробників та на їх допомогу.

Фреймворк *Django* є доволі простим у навчанні через те, що він заснований на мові програмування *Python*, яка є доволі простою, та чіткою документацією. Також код із використанням цього інструменту є компактним та швидко виконується.

Було розглянуто популярні інструменти для *frontend* розробки веб-додатків, а саме: фреймворк *Bootstrap,* фреймворк *ReactJS,* фреймворк *Vue.js.* Було виявлено основні переваги та недоліки цих інструментів.

Фреймворк *Bootstrap* підійде для створення якісного та практичного сайту у максимально короткий проміжок часу, нехтуючи його унікальністю. Проте, якщо розробник має достатні навички, він може вирішити проблему з індивідуальністю сайту. *ReactJS* більше підійде тим розробникам, які шукають бібліотеки для розробки, а не цілий фреймворк. Фреймворк *Vue.js* підійде для застарілих проектів, яким потрібен новий сучасний двигун.

У другому розділі було обрано мову програмування для розробки *backend* частини програми – *Java* із фреймворками: *Spring, Hibernate, Spring Boot, Spring Data, Spring MVC.* Було обрано середовище розробки *IntelliJ IDEA* з інструментом автоматичної зборки проектів *Maven*. Для тестування було обрано фреймворки *JUnit* та *Mockito*. Була обрана система управління базами даних *MySQL.*

Було спроектовано базу даних для вирішення поставленої задачі, а саме таблиці: міст, уроків, оцінок, олімпіад, оцінок за олімпіади, учні, школи, класи, предмети та вчителі. Також було спроектовано основні функції серверної частини програмного забезпечення системи аналізу та обліку успішності школярів.

Було визначено, що основними функціями *backend* частини додатку системи аналізу та обліку успішності школярів є робота з таблицями, а саме можливість їх перегляду, редагування, додавання та видалення.

Були спроектовані *Entity*-класи для *backend* частини, а саме: *City* – для міст, *Lesson* – для уроків, *Mark* – для оцінок, *Olympiad* – для олімпіад, *OlympiadMark* – для оцінок за олімпіади, *Pupil* – для учнів, *School* – для шкіл, *SchoolClass* – для класів, *Subject* – для предметів, *Teacher* – для вчителів. Були спроектовані класи-контролери: City*RestController, LessonRestController, MarkRestController, OlympiadMarkRestController, OlympiadRestController, PupilRestController,*

SchoolClassRestController, *SchoolRestController*, *SubjectRestController*, *TeacherRestController*. Були спроектовані класи-сервіси: *CityService*, *LessonService*, *MarkService*, *OlympiadMarkService*, *OlympiadService*, *PupilService*, *SchoolClassService*, *SchoolService*, *SubjectService*, *TeacherService*. Були спроектовані класи-репозиторії: *CityRepository*, *LessonRepository*, *MarkRepository*, *OlympiadMarkRepository*, *OlympiadRepository*, *PupilRepository*, *SchoolClassRepository*, *SchoolRepository*, *SubjectRepository*, *TeacherRepository*.

Було спроектовано основні функції *frontend* частини проекту. Також були спроектовані класи-контролери: City*Controller*, *LessonController*, *MarkController*, *OlympiadMarkController*, *OlympiadController*, *PupilController*, *SchoolClassController*, *SchoolController*, *SubjectController*, *TeacherController*, *AnalysisController*. Були спроектовані класи-сервіси: *CityService*, *LessonService*, *MarkService*, *OlympiadMarkService*, *OlympiadService*, *PupilService*, *SchoolClassService*, *SchoolService*, *SubjectService*, *TeacherService*.

Було спроектовано основні функції аналізу та успішності школярів: аналіз успішності шкіл серед усіх доданих до бази даних шкіл; аналіз успішності шкіл серед шкіл певного міста; аналіз успішності класів у певній школі; аналіз успішності учнів у певному класі; аналіз успішності учнів по олімпіаді.

У третьому розділі було розглянуто склад файлів як для *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів, так і *frontend* частини. Було детально розглянуто файли та папки, які були створені автоматично та вручну. Було описано пакети, які є в проектах, та файли класів із перерахованими методами в них.

Розробка *backend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA*. Назву проекту було обрано *journal_project*, папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_project*. Назву пакету проекту – *com.makariev.journal_project*. Тип архівування проекту було обрано *Jar*. Було обрано версію *Spring Boot* 2.7.5 та додано наступні залежності: *Spring Boot DevTools*, *Lombok*, *Spring Web*, *Spring Data JPA*, *MySQL Driver*.

Розробка *frontend* частини програмного забезпечення системи аналізу та обліку успішності школярів велася в середовищі розробки *IntelliJ IDEA.* Назву проекту було обрано *journal_client,* папку проекту створено за адресою *D:\Users\Yegor\IdeaProjects\journal_client.* Назву пакету проекту – *com.makariev.journal_client.* Тип архівування проекту було обрано *Jar.* Було обрано версію *Spring Boot* 2.7.5 та додано наступні залежності: *Spring Boot DevTools, Lombok, Spring Web, Thymeleaf.*

Також у третьому розділі було розроблено основні модулі *backend* частини проекту: модуль пошуку інформації; модуль додавання, редагування та видалення записів; модуль аналізу та обліку успішності школярів. Були розроблені наступні класи-сервіси: *CityServiceImpl* для роботи з містами та має наступні методи: *findById, findAll, addNewCity, updateCity, deleteCity; LessonServiceImpl* для роботи з уроками та має наступні методи: *findById, findAllByTeacher_Subject_Id AndSchoolClass_Id, findAllByTeacher_Id, addNewLesson, updateLesson, deleteLesson; MarkServiceImpl* для роботи з оцінками та має наступні методи: *findAllByPupil_Id, findAllByLesson_Teacher_Subject_IdAndPupil_SchoolClass_Id, findAllByLesson_Teacher_Subject_IdAndPupil_Id, findAllByLesson_Id, addNewMark, addNewMarks, updateMark, deleteMark; OlympiadMarkServiceImpl* для роботи з оцінками за олімпіади та має наступні методи: *findById, findAllByPupil_Id, findAllByOlympiad_Id, addNewOlympiadMark, updateOlympiadMark, deleteOlympiadMark;* для роботи з олімпіадами та має наступні методи: *findById, findAll, findAllByDateBetween, addNewOlympiad, updateOlympiad, deleteOlympiad; PupilServiceImpl* для роботи з учнями та має наступні методи: *findById, findAllBySchoolClass_School_Id, findAllBySchoolClass_Id, findAllByFirstNameContainsIgnoreCaseAndLastNameContains IgnoreCaseAndMiddleNameContainsIgnoreCase, findAllByOlympiadId, findAllStatistics BySchoolClass_Id, addNewPupil, updatePupil, deletePupil; SchoolClassServiceImpl* для роботи з класами та має наступні методи: *findById, findAllBySchool_Id, findAllStatisticsBySchool_Id, addNewSchoolClass, updateSchoolClass, deleteSchoolClass; SchoolServiceImpl* для роботи зі школами та має наступні методи:

findById, findAllByCity_Id, findAllSchoolStatisticsByCity_id, addSchoolClass, updateSchool, deleteSchool.

У третьому розділі також було розроблено графічний інтерфейс *frontend* частини веб-додатку. Були створені файли для сторінок аналізу: *analysis-class-results.html, analysis-class-search.html, analysis-pupil-results.html, analysis-pupil-search.html, analysis-school-results.html, analysis-school-search.html;* для роботи з містами: *cities-edit.html, cities-results.html;* для роботи з класами: *class-details.html, class-edit.html;* для роботи з оцінками: *lesson-edit.html, marks-class.html, marks-search.html;* для роботи з олімпіадами: *olympiad-details.html, olympiad-edit.html, olympiad-results.html, olympiad-search.html, olympiadmark-edit.html;* для роботи з учнями: *pupil-details.html, pupil-edit.html, pupil-results.html, pupil-search.html;* для роботи зі школами: *school-details.html, school-edit.html, school-results.html, school-search.html;* для роботи з предметами: *subject-edit.html, subject-results.html;* для роботи з вчителями: *teacher-details.html, teacher-edit.html, teacher-results.html, teacher-search.html.*

У четвертому розділі було розглянуто інструменти для роботи з базами даних у мові програмування *Java*. Було досліджено роботу фреймворку *Spring Data* та слоїв, які він використовує для роботи з базами даних: *ORM* фреймворк *Hibernate*, специфікація *JPA*, стандартний інструмент *Java – JDBC*.

Також у четвертому розділі було порівняно швидкодію виконання запитів до бази даних із використанням фреймворку *Spring Data* та стандартного інструменту *Java – JDBC*. Для порівняння швидкодії виконання запитів було використано таблицю оцінок *marks*. Вона має наступні колонки: *id* – ідентифікаційний номер оцінки; *mark* – оцінка за урок; *pupil_id* – зовнішній ключ, що вказує на поле *id* у таблиці учнів *pupils*; *lesson_id* – зовнішній ключ, що вказує на поле *id* у таблиці уроків *lessons*. Для проведення вимірів швидкодії було використано *JMH(Java Microbenchmark Harness).*

Було зроблено виміри на основі десяти ітерацій для кожного засобу та пораховано середній час за який виконується одна ітерація. Час однієї ітерації було виміряно у наносекундах для отримання більш точних даних

У результаті порівняння швидкодіїї двох засобів взаємодії з базами даних було встановлено, що середній час виконання однієї ітерації з використанням *JDBC* – 9085,280 наносекунд, а результат *Spring Data* – 46466,628 наносекунд. Тобто *JDBC* виявився швидшим за *Spring Data* у 5,11 разів.

Проте, не зважаючи на отримані результати, *Spring Data* є більш сучасним та популярним засобом взаємодії з базою даних, ніж *JDBC,* так як технологія *JPA* зменшує час на написання коду розробниками.

У звіті кваліфікаційної роботи були описані основні етапи проектування та розробки програмного забезпечення системи аналізу та обліку успішності школярів.

Дипломний проект було виконано з дотриманням діючих стандартів та положень [15], [16].

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Выбираем лучший бэкенд-фреймворк 2021 года [електронний ресурс] – Режим доступу: *https://habr.com/ru/company/ruvds/blog/519478/* (дата звернення 15.10.2022)
2. М. Хеклер. *Spring Boot* по-быстрому. Создаем облачные приложения на Java и Kotlin. – Санкт-Петербург: Питер Пресс, 2022. – 352 с.
3. Плюсы и минусы *Django* [електронний ресурс] – Режим доступу: *https://habr.com/ru/post/473042/* (дата звернення 15.10.2022)
4. Плюсы и минусы *Bootstrap* [електронний ресурс] – Режим доступу: *https://timeweb.com/ru/community/articles/plyusy-i-minusy-bootstrap-1* (дата звернення 15.10.2022)
5. Плюсы и минусы *React*: виртуальная *DOM*, синтаксис *JSX* и другие аргументы для спора [електронний ресурс] – Режим доступу: *https://nuancesprog.ru/p/14500/* (дата звернення 16.10.2022)
6. *React vs Angular vs Vue.js — What to choose in 2021?* [електронний ресурс] – Режим доступу: *https://medium.com/techmagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d* (дата звернення 16.10.2022)
7. К. Сьєрра, Б. Бейтс. *Head First. Java*. 2-ге, оновлене видання. – Харків: Фабула, 2022. – 720 с.
8. Що таке *Maven* - Підручник *Maven* для початківців [електронний ресурс] – Режим доступу: *https://uk.myservername.com/what-is-maven-maven-tutorial* (дата звернення 17.10.2022)
9. Б. Кристиан, К. Гэвин, Г. Гэри. Java Persistence API и Hibernate. – Москва: ДМК Пресс, 2018. – 632 с.
10. Учебник *Thymeleaf*: Глава 1. Знакомство [електронний ресурс] – Режим доступу: *https://habr.com/ru/post/350864/* (дата звернення 17.10.2022)
11. Что такое Bootstrap и зачем он нужен? [електронний ресурс] – Режим доступу: *https://itchief.ru/bootstrap/introduction* (дата звернення 17.10.2022)

12. К.Уоллс. *Spring* в действии. – Москва: ДМК Пресс, 2015. – 754 с.

13. К. Бастани. *Java* в облаке. *Spring Boot, Spring Cloud, Cloud Foundry*. – Санкт-Петербург: Питер Пресс, 2019. – 624 с.

14. Архитектура REST [електронний ресурс] – Режим доступу: *https://habr.com/ru/post/38730/* (дата звернення 17.10.2022)

15. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86с.

16. Слободян О. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: Видавництво НАУ, 2017. – 63с.

ДОДАТОК А
ФРАГМЕНТ ВИХІДНОГО КОДУ

//Код файлу *SchoolClassRestController.java*

```
@RestController
@RequestMapping("/api/v1/classes")
@RequiredArgsConstructor
public class SchoolClassRestController {
    private final SchoolClassService schoolClassService;
    @GetMapping("/{schoolClassId}")
    @ResponseBody
    public ResponseEntity<SchoolClassDto> getSchoolClassById(@PathVariable int
schoolClassId) {
        return new ResponseEntity<>(schoolClassService.findById(schoolClassId),
HttpStatus.OK);
    }
    @GetMapping("/school/{schoolId}")
    @ResponseBody
    public ResponseEntity<List<SchoolClassDto>>
getAllSchoolClassesBySchoolId(@PathVariable int schoolId) {
        return new ResponseEntity<>(schoolClassService.findAllBySchool_Id(schoolId),
HttpStatus.OK);
    }
    @GetMapping("/statistics/school/{schoolId}")
    @ResponseBody
    public ResponseEntity<List<SchoolClassRatingDto>>
getAllStatisticsSchoolClassesBySchoolId(@PathVariable int schoolId) {
```

```
        return new
ResponseEntity<>(schoolClassService.findAllStatisticsBySchool_Id(schoolId),
HttpStatus.OK);
    }
    @PostMapping
    @ResponseBody
    public ResponseEntity<SchoolClassDto> addNewSchoolClass(@RequestBody
SchoolClassDto schoolClassDto) {
        return new
ResponseEntity<>(schoolClassService.addNewSchoolClass(schoolClassDto),
HttpStatus.CREATED);
    }
    @PutMapping("/{schoolClassId}")
    @ResponseBody
    public ResponseEntity<SchoolClassDto> updateSchoolClass(@RequestBody
SchoolClassDto schoolClassDto, @PathVariable int schoolClassId) {
        return new
ResponseEntity<>(schoolClassService.updateSchoolClass(schoolClassDto,
schoolClassId), HttpStatus.OK);
    }
    @DeleteMapping("/{schoolClassId}")
    @ResponseBody
    public ResponseEntity<SchoolClassDto> deleteSchoolClass(@PathVariable int
schoolClassId) {
        return new
ResponseEntity<>(schoolClassService.deleteSchoolClass(schoolClassId),
HttpStatus.OK);
    }
}
```

//Код файлу *SchoolClassServiceImpl.java*

```
@Service
@RequiredArgsConstructor
public class SchoolClassServiceImpl implements SchoolClassService {
    private final SchoolClassRepository schoolClassRepository;
    private final SchoolClassMapper schoolClassMapper;
    private final SchoolRepository schoolRepository;
    private final MarkRepository markRepository;
    private final OlympiadMarkRepository olympiadMarkRepository;
    private final PupilRepository pupilRepository;
    @Transactional
    @Override
    public SchoolClassDto findById(int schoolClassId) {
        return
schoolClassMapper.schoolClassEntityToModel(schoolClassRepository.findById(school
ClassId)
                .orElseThrow(() -> new WrongIdForEntityException(schoolClassId,
"Class")));
    }
    @Transactional
    @Override
    public List<SchoolClassDto> findAllBySchool_Id(int schoolId) {
        return
schoolClassMapper.schoolClassEntitiesToModels(schoolClassRepository.findAllByScho
ol_Id(schoolId));
    }
    @Transactional
    @Override
    public List<SchoolClassRatingDto> findAllStatisticsBySchool_Id(int schoolId) {
        List<SchoolClassRatingDto> schoolClassRatingDtoList = new ArrayList<>();
```

```
List<SchoolClassDto> schoolClassDtos =
schoolClassMapper.schoolClassEntitiesToModels(schoolClassRepository.findAllBySchoo
l_Id(schoolId));
List<AverageMarkSchoolClass> averageMarkSchoolClassList =
markRepository.countSchoolClassAverageMarkBySchool_Id(schoolId);
List<PupilCountSchoolClass> pupilCountSchoolClassList =
pupilRepository.countPupilsInSchoolClassesBySchool_Id(schoolId);
List<AverageOlympiadMarkSchoolClass> averageOlympiadMarkSchoolClassList
=
olympiadMarkRepository.countSchoolClassAverageOlympiadMarkBySchool_Id(schoolI
d);
for (SchoolClassDto schoolClassDto : schoolClassDtos) {
    SchoolClassRatingDto schoolRatingDto = new SchoolClassRatingDto();
    schoolRatingDto.setSchoolClassDto(schoolClassDto);
    schoolClassRatingDtoList.add(schoolRatingDto);
}
for (SchoolClassRatingDto schoolClassRatingDto : schoolClassRatingDtoList) {
    schoolClassRatingDto.setPupilCount(pupilCountSchoolClassList.stream()
            .filter(f -> f.getSchoolClassId() ==
schoolClassRatingDto.getSchoolClassDto().getId())
            .findFirst()
            .orElse(new
PupilCountSchoolClass(schoolClassRatingDto.getSchoolClassDto().getId(), 0))
            .getPupilCount());
    schoolClassRatingDto.setAverageMark(averageMarkSchoolClassList.stream()
            .filter(f -> f.getSchoolClassId() ==
schoolClassRatingDto.getSchoolClassDto().getId())
            .findFirst()
            .orElse(new
AverageMarkSchoolClass(schoolClassRatingDto.getSchoolClassDto().getId(), 0.0))
```

```
                .getAverageMark());
schoolClassRatingDto.setAverageOlympiadMark(averageOlympiadMarkSchoolClassLis
t.stream()
                .filter(f -> f.getSchoolClassId() ==
schoolClassRatingDto.getSchoolClassDto().getId())
                .findFirst()
                .orElse(new
AverageOlympiadMarkSchoolClass(schoolClassRatingDto.getSchoolClassDto().getId(),
0.0))
                .getAverageOlympiadMark());
        }
        return schoolClassRatingDtoList;
    }
    @Transactional
    @Override
    public SchoolClassDto addNewSchoolClass(SchoolClassDto schoolClassDto) {
        School school = schoolRepository.findById(schoolClassDto.getSchoolId())
                .orElseThrow(() -> new
WrongIdForEntityException(schoolClassDto.getSchoolId(), "Class"));
        SchoolClass schoolClass =
schoolClassMapper.schoolClassModelToEntity(schoolClassDto);
        schoolClass.setSchool(school);
        return
schoolClassMapper.schoolClassEntityToModel(schoolClassRepository.save(schoolClas
s));
    }
    @Transactional
    @Override
    public SchoolClassDto updateSchoolClass(SchoolClassDto schoolClassDto, int
schoolClassId) {
```

```
        SchoolClass schoolClass = schoolClassRepository.findById(schoolClassId)
                .orElseThrow(() -> new WrongIdForEntityException(schoolClassId,
"Class"));
        School school = schoolRepository.findById(schoolClassDto.getSchoolId())
                .orElseThrow(() -> new
WrongIdForEntityException(schoolClassDto.getSchoolId(), "Class"));
        schoolClass.setName(schoolClassDto.getName());
        schoolClass.setSchool(school);
        return
schoolClassMapper.schoolClassEntityToModel(schoolClassRepository.save(schoolClas
s));
    }
    @Transactional
    @Override
    public SchoolClassDto deleteSchoolClass(int schoolClassId) {
        SchoolClass schoolClass = schoolClassRepository.findById(schoolClassId)
                .orElseThrow(() -> new WrongIdForEntityException(schoolClassId,
"Class"));
        schoolClassRepository.delete(schoolClass);
        return schoolClassMapper.schoolClassEntityToModel(schoolClass);
    }
}
```

//Код файлу *SchoolClassRepository.java*

```
public interface SchoolClassRepository extends JpaRepository<SchoolClass, Integer> {
    List<SchoolClass> findAllBySchool_Id(int schoolId);
}
```