МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 20__ р.

На правах рукопису
УДК 004.056:004.43 (079.2)

ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: Програмне забезпечення для реалізації цифрового університету

Виконавець: Кузьменко Є.Р.
Керівник: д.т.н., професор кафедри БІТ Заріцький О.В.
Нормоконтролер: д.т.н., професор кафедри БІТ Заріцький О.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Бакалавр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ С.В. Казмірчук

«__» _____ 20__ р.

ЗАВДАННЯ

на виконання дипломної роботи

здобувача вищої освіти Кузьменка Євгенія Романовича

1. Тема: *Програмне забезпечення для реалізації цифрового університету* затверджена наказом ректора від «26» квітня 2021 р. № 652/ст.
2. Термін виконання: з 10.05.2021 р. по 20.06.2021 р.
3. Вихідні дані: проаналізувати предметну область, яка включає в себе основні засоби та вимоги розробки цифрового університету; розробити структуру проекту та реалізувати цифровий університет; протестувати роботу проекту.
4. Зміст пояснювальної записки: аналіз предметної області, аналіз вимог, структура проекту, реалізація цифрового університету.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	10.05.2021	*Виконано*
2.	Аналіз літературних джерел	11.05.2021 – 13.05.2021	*Виконано*
3.	Аналіз основних засобів розробки цифрового університету	14.05.2021 – 17.05.2021	*Виконано*
4.	Аналіз вимог до проекту	18.05.2021 – 19.05.2021	*Виконано*
5.	Визначення структури проекту та реалізація цифрового університету	20.05.2021 – 27.05.2021	*Виконано*
6.	Тестування проекту	28.05.2021 – 30.05.2021	*Виконано*
7.	Створення презентації	31.05.2021 – 02.06.2021	*Виконано*
8.	Перевірка на антиплагіат	06.06.2021	*Виконано*
9.	Оформлення і друк пояснювальної записки	07.06.2021	*Виконано*
10.	Оформлення презентації	08.06.2021	*Виконано*
11.	Отримання рецензії від рецензента	10.06.2021	*Виконано*

Здобувач вищої освіти Є.Р. Кузьменко

(підпис, дата)

Керівник дипломної роботи О.В. Заріцький

(підпис, дата)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 75 сторінок, 14 джерел, 16 рисунків.

Пояснювальна записка до дипломної роботи «Програмне забезпечення для реалізації цифрового університету».

Об'єкт дослідження – програмне забезпечення для реалізації цифрового університету.

Мета дипломної роботи – створення веб-додатку для цифрового університету та певної взаємодії з ним.

Метод розробки – ООП – об’єктно-орієнтований підхід.

Технічні та програмні засоби – ПК з ОС Windows; середовище об’єктно-орієнтованого програмування IntellIJ Idea.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

БД – база даних

СУБД – система управління базами даних

MVC Controller – Model-View-Controller

ВСТУП

Актуальність. У сучасному світі та в наш час дійсно актуальне питання цифрового університету.

Зараз вся інформація знаходиться в інтернеті і потреба в такому веб-додатку дійсно є.

Будь-яка людина - будь-то студент, або ж викладач, повинна мати доступ подивитися свій розклад уроків, не виходячи з дому, що в даному проекті і буде реалізовано.

Метою дипломної роботи є покращення роботи вчителів та студентів, а конкретно – спрощення отримання розкладу предметів, як для студентів, так і для вчителів.

Об'єкт дослідження: програмне забезпечення для реалізації цифрового університету.

Предмет дослідження: веб-проект на мові програмування Java

Галузь застосування. Даний проект може буди використан у будь-якому університеті світу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз потреби цифрового університету

Потреба у цифровому університеті безумовно велика, особливо в наш час, коливсі студенти вчаться вдома та багато вчителів вчуть дистанційно.

Коли проходять заннятя онлайн, то будь-який студент повинен мати доступ дорозкладу / оцінок, в цілому своєї успішності.

Дійсно, у всіх університетах України, принаймні, у провідних, ці сайти, тобтоцифрові університеті, є.

Без цифрового університету складно уявити будь-який заклад, тому що заразусе в інтернеті та вся інформація щодо університету повинна бути також в інтернеті.

1.2. Основні засоби розробки цифрового університету

Java - об'єктно-орієнтована мова програмування, віпущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року Придбай «Sun Microsystems». В офіційній реализации Java-програми компілюються у байт- код, Який при віконанні інтерпретується віртуальною машиною для конкретної платформи.

IntelliJ IDEA - інтегроване середовище розробки програмного забезпечення длябагатьох мов програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains.

Spring Framework - це програмний каркас (фреймворк) з відкрітім кодом та контейнери з підтрімкою інверсії управління для платформи Java.

Основні особливості Spring Framework могут буті вікорістані будь-якім додатком Java, но є розширення для создания веб-додатків на платформі Java EE.

Незважаючі на це, Spring Framework НЕ нав'язує якоїсь конкретної моделі програмування, Spring Framework ставши популярним в спільноті Java як альтернатива, або даже ДОПОВНЕННЯ моделі Enterprise JavaBean (EJB).

Spring Boot - комплексний фреймворк для створення і запуску додатків з мінімальними зусиллями і настройками. Цей модуль ділиться на два стека: заснований на API сервлетів Spring MVC і реактивний Spring WebFlux.

Spring WebFlux - веб-платформа, створена, щоб по максимуму використовувати переваги сучасних багатоядерних процесорів і обробляти величезну кількість одночасних підключень.

Spring MVC побудований на API сервлетів і використовує архітектуру синхронного блокуючого вводу-виводу з моделлю «один запит на потік».

Spring Boot дозволяє вам легко створювати повноцінні, виробничого класу Spring-додатки, про які можна сказати - "просто запусти". Ми включили Spring-платформу і сторонні бібліотеки, щоб ви могли запустити з мінімум зусиллями. Більшості Spring Boot додатків потрібно зовсім маленька Spring-конфігурація

Можливості *Spring Boot* :

- Створення повноцінних Spring додатків
- Вбудований Tomcat або Jetty (не потрібно установки WAR файлів)
- Забезпечує 'початкові' POMs для спрощення вашої Maven конфігурації
- Автоматична конфігурірація Spring коли це можливо
- Забезпечує такими можливостями, як метрики, моніторинг станами і розширена конфігурація
- Абсолютно без генерації коду і без написання XML конфігурації

Spring Boot створений, щоб допомогти програмістам прискорити процес розробки. Він дозволяє позбутися від трудомісткої початкової установки і настройки середовища розгортання.

Основні переваги *Spring Boot*:

- Швидка і легка розробка додатків на основі Spring.
- Автом всіх компонентів для додатка Spring виробничого рівня.

- Готові вбудовані сервери (Tomcat, Jetty і Undertow), що забезпечують -прискорене і більш продуктивне розгортання додатків.
- HTTP end-points, що дозволяють вводити внутрішні функції програми, такі якпоказники, стан здоров'я та інші.
- Відсутність конфігурації XML.
- Величезний вибір плагінів, що полегшують роботу розробників з вбудованими базами даних і базами даних в пам'яті.
- Легкий доступ до баз даних і службам черг, таким як MySQL, Oracle, MongoDB, Redis, ActiveMQ і іншим.
- Плавне інтеграція з екосистемою Spring.
- Велике співтовариство і безліч навчальних програм, що полегшують ознайомлення.

Недоліки *Spring Boot*:

- Відсутність контролю. Spring Boot створює безліч невикористовуваних залежностей, що призводить до великому розміру файлу розгортання.
- Складний і трудомісткий процес перетворення застарілого або існуючогопроекту Spring в додатки Spring Boot.
- Не підходить для масштабних проектів. На думку багатьох розробників, незважаючи на відсутність проблем при роботі з мікросервісамі, Spring Bootне підходить для створення монолітних додатків.

Spring Security - це фреймворк, який сфокусований на забезпечення як аутентифікації, так і авторизації в Java-додатках. Як і всі Spring проекти, справжня сила Spring Security в тому, що він може бути легко доповнений за потрібне функціоналом.

Можливості *Spring Security* :

- Комплексна і розширюється підтримка як аутентифікації, так і авторизації
- Захист від атак типу фіксація сесії, клікджекінг, міжсайтовий підробказапиту і ін.

- Інтеграція з Servlet API
- Інтеграція з Spring Web MVC при необхідності

Завдяки SpringSecurity у проекті налаштована робота з сутностями – такими, кTeacher та Student.

Для цього треба додати у проект 2 класи, як я і зробив :

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
@Autowired
private TeacherService teacherService;@Autowired
private StudentService studentService;
@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
Student student = studentService.findByLogin(username);if (student != null) {
return new CustomUserDetails(student.getStudentLogin(),
student.getStudentPassword(), student.getStudentRole());
} else {
Teacher teacher = teacherService.findByLogin(username);if (teacher != null) {
return new CustomUserDetails(teacher.getTeacherLogin(),
teacher.getTeacherPassword(), teacher.getTeacherrole());
}
}
throw new UsernameNotFoundException("User '" + username + "' not
found");
}
public class CustomUserDetails implements UserDetails {

private String username;private String password;
private Collection<? extends GrantedAuthority> authorities;
public CustomUserDetails() {
```

```
super();
}
public CustomUserDetails(String username, String password, String role) {
this.username = username;
this.password = password;
List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
grantedAuthorities.add(new SimpleGrantedAuthority(role)); this.authorities =
grantedAuthorities;
}
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
return authorities;
}
@Override
public String getPassword() {
return password;
}
@Override
public String getUsername() {
return username;
}
@Override
public boolean isAccountNonExpired() {
return true;
}
@Override
public boolean isAccountNonLocked() {
return true;
}
@Override
```

```
public boolean isCredentialsNonExpired() {
return true;
}
@Override
public boolean isEnabled() {
return true;
}
}
```

Та клас SecurityConfig, де і описуються правила доступу URL :

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
@Override
protected void configure(HttpSecurity http) throws Exception {http
.authorizeRequests().antMatchers("/", "/login","/studentViews/students",
"/teacherViews/teachers", "/lessonViews/lessons",
"/groupViews/groups").permitAll().antMatchers("/teacherViews/**").hasRole("T
EACHER")
.antMatchers("/studentViews/**").hasRole("STUDENT")
.anyRequest().authenticated()
.and().formLogin()
.and()
.logout().logoutRequestMatcher(new
AntPathRequestMatcher("/logout")).logoutSuccessUrl("/");
http.exceptionHandling().accessDeniedPage("/accessDeniedPage");
}
@Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception
{
auth.authenticationProvider(authProvider());
```

```
}
@Bean
public UserDetailsService userDetailsService() {
return new CustomUserDetailsService();
}
@Bean
public DaoAuthenticationProvider authProvider() {
DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
authProvider.setUserDetailsService(userDetailsService());
authProvider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
return authProvider;
}
```

HTML і *CSS* - це два основні інструменти, які необхідні при роботі з шаблонами сайту.

HTML - стандартизований мову програмування документів. Більшість сторінокмістять опис розмітки на мові HTML. Іншими словами, HTML визначає структуру вмісту сторінки. Наприклад, саме в HTML задаються заголовки і абзаци, зображення і т.д.

CSS - мова таблиць стилів, який дозволяє прикріплювати стиль (наприклад, шрифти і колір) до структурованих документів (наприклад, до вищезазначених документів HTML). Простіше кажучи, основне призначення CSS - описувати оформлення зовнішнього вигляду контенту.

Таким чином, якщо HTML потрібен для того, щоб описати, яка саме інформаціяі в якому порядку повинна виводитися на сторінці, то CSS розширює можливість HTML і дозволяє змінювати кольори, шрифти, фон і т.д.

База даних (БД) — це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів.

БД використовують для динамічних сайтів з великими обсягами (інтернет-магазин, портал, корпоративний сайт).

База даних — це певний набір даних, які пов'язані між собою спільною ознакоюабо властивістю, та впорядковані, наприклад, за алфавітом.

Об'єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіації групування інформації — особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше.

Головною перевагою БД є швидкість внесення та використання потрібної інформації. Завдяки спеціальним алгоритмам, які використовуються для баз даних, можна легко знаходити необхідні дані всього за декілька секунд. Такожв базі даних існує певний взаємозв'язок інформації: зміна в одному рядку може спричинити зміни в інших рядках — це допомагає працювати з інформацією простіше і швидше.

Призначення. Бази даних для сайтів дають змогу зберігати інформацію, що виглядає як зв'язані між собою таблиці. Саме в БД зберігаються вся необхідна та корисна інформація для функціонування сайту (клієнтські дані, прайс-лист, список товарів).

Щоб створити запит до бази даних часто використовують Structured Query Language. SQL дає змогу додавати, редагувати та видаляти інформацію, що міститься у таблицях.

Під час програмування сайтів використовують різні системи управління БД. До основних СУБД, відносять:

- об'єктно-реляційна система управління базами даних Oracle Database;
- вільна система управління базами даних PostgreSQL;
- система керування базами даних Microsoft SQL Сервер;
- вільна система управління базами даних MySQL;

Такі системи управління відрізняються централізованою обробкою запитів,забезпечують надійність, доступність та безпеку БД.

Найбільш популярною системою управління є MySQL, вона дає зручний доступ для управління БД та підтримує велику кількість таблиць різних типів.

Системи управління базами даних

СУБД — це програмні засоби, які виступають посередником між БД та її користувачами. Завдяки сукупності мовних та програмних засобів, СУБД сприяють створенню, ведення та спільного використання БД різними користувачами.

Сучасна програма СУБД складаються з ядра, процесору мови БД, підсистеми підтримки часу виконання та сервісних програм, які надають додаткові можливості для обслуговування інформаційних систем.

SQL (англ. Structured query language - «мова структурованих запитів») - декларативний мову програмування, застосовуваний для створення, модифікації та управління даними в реляційній базі даних, керованої відповідною системою управління базами даних.

Є, перш за все, інформаційно-логічним мовою, призначеним для опису, зміни і вилучення даних, що зберігаються в реляційних базах даних. SQL вважається мовою програмування, в загальному випадку (без ряду сучасних розширень) неє Тьюринг-повним, але разом з тим стандарт мови специфікацією SQL / PSM передбачає можливість його процедурних розширень.

Спочатку SQL був основним способом роботи користувача з базою даних і дозволяв виконувати наступний набір операцій:

- створення в базі даних нової таблиці;
- додавання в таблицю нових записів;
- зміна записів;
- видалення записів;
- вибірка записів з однієї або декількох таблиць (відповідно до заданогоумовою);
- зміна структур таблиць.

Згодом SQL ускладнився - збагатився новими конструкціями, забезпечив можливість опису та управління новими збереженими об'єктами (наприклад, індекси, уявлення, тригери і процедури) - і став набувати рис, властиві мовам програмування.

При всіх своїх змінах SQL залишається найпоширенішим лінгвістичним засобом для взаємодії прикладного програмного забезпечення з базами даних. У той же час сучасні СУБД, а також інформаційні системи, що використовують СУБД, надають користувачеві розвинені засоби візуального побудови запитів.

Мова структурованих запитів (SQL), як ми всі знаємо, є мовою бази даних, за допомогою якої ми можемо виконувати певні операції з існуючою базою даних,а також ми можемо використовувати цю мову для створення бази даних. SQL використовує певні команди, такі як "Створити", "Відпустити", "Вставити" тощо для виконання необхідних завдань.

Ці команди SQL в основному поділяються на чотири категорії:

- DDL - мова визначення даних
- DQl - мова запитів даних
- DML - мова керування даними
- DCL - мова контролю даних

DDL (Мова визначення даних): DDL або Мова визначення даних фактично складається з команд SQL, які можна використовувати для визначення схеми бази даних. Він просто має справу з описами схеми бази даних і використовується для створення та модифікації структури об'єктів бази даних у базі даних.

Приклади команд DDL:

CREATE - використовується для створення бази даних або її об’єктів (наприклад, таблиці, індексу, функції, подань, процедури зберігання татригерів).

DROP - використовується для видалення об’єктів з бази даних.ALTER - використовується для зміни структури бази даних.

TRUNCATE - використовується для видалення всіх записів із таблиці, включаючи всі пробіли, виділені для записів.

КОМЕНТАР - використовується для додавання коментарів до словника даних.RENAME - використовується для перейменування об'єкта, що існує в базі даних.

DQL (мова запитів даних):

Оператори DQL використовуються для виконання запитів щодо даних в об'єктах схеми. Призначення команди DQL - отримати деяке відношення схеми на основі переданого їй запиту.

Приклад DQL:

SELECT - використовується для отримання даних з бази даних.

DML (мова маніпуляції даними): Команди SQL, які займаються маніпулюванням даними, наявними в базі даних, належать DML або мовою маніпуляції даними, і це включає більшість операторів SQL.

Приклади DML:

INSERT - використовується для вставки даних у таблицю.

UPDATE - використовується для оновлення наявних даних у таблиці. DELETE - використовується для видалення записів з таблиці бази даних.

DCL (мова контролю даних): DCL включає такі команди, як GRANT та REVOKE, які в основному стосуються прав, дозволів та інших елементів керування системою баз даних.

Приклади команд DCL:

GRANT - надає права користувача на доступ до бази даних.

REVOKE - відкликати права доступу користувача, надані за допомогою команди GRANT.

TCL (Мова контролю транзакцій): Команди TCL мають справу з транзакцією вбазі даних.

Приклади команд TCL:

COMMIT - здійснює транзакцію.

ROLLBACK - відкат транзакції у разі виникнення помилки. SAVEPOINT – встановлює точку збереження в межах транзакції. ВСТАНОВИТИ ОПЕРАЦІЮ - вкажіть характеристики транзакції.

PostgreSQL - це популярна вільна об'єктно-реляційна система управління базами даних. PostgreSQL базується на мові SQL і підтримує численні можливості.

Переваги PostgreSQL :

- підтримка БД необмеженого розміру;
- потужні і надійні механізми транзакцій і реплікації;
- розширювана система вбудованих мов програмування і підтримка - завантаження С-сумісних модулів;
- спадкування;
- легка розширюваність.

Поточні обмеження:

- Немає обмежень на максимальний розмір бази даних
- Немає обмежень на кількість записів в таблиці
- Немає обмежень на кількість індексів в таблиці
- Максимальний розмір таблиці - 32 Тбайт
- Максимальний розмір запису - 1,6 Тбайт
- Максимальний розмір поля - 1 Гбайт
- Максимум полів в запісі250-1600 (в залежності від типів полів) Особливості *PostgrelSQL* :

Функції в PostgreSQL є блоками коду, виконуваними на сервері, а не на клієнтаБД. Хоча вони можуть писатися на чистому SQL, реалізація додаткової логіки,наприклад, умовних переходів і циклів, виходить за рамки власне SQL і вимагає використання деяких мовних розширень. Функції можуть писатися з використанням різних мов програмування. PostgreSQL допускає використання функцій, які повертають набір записів, який далі можна використовувати так само, як і результат виконання звичайного запиту. Функції можуть виконуватися як з правами їх творця, так і з правами поточного користувача.

Іноді функції ототожнюються з збереженими процедурами, проте між цими поняттями є різниця.

Тригери в PostgreSQL визначаються як функції, що ініціюються DML-

операціями. Наприклад, операція INSERT може запускати тригер, перевіряючий додану запис на відповідності певним умовам. При написанні функцій для тригерів можуть використовуватися різні мови програмування. Тригери асоціюються з таблицями. Множинні тригери виконуються в алфавітному порядку.

Механізм правил в PostgreSQL є механізм створення користувацьких обробників не тільки DML-операцій, а й операції вибірки. Основна відмінність від механізму тригерів полягає в тому, що правила спрацьовують на етапі розбору запиту, до вибору оптимального плану виконання і самого процесу виконання. Правила дозволяють перевизначати поведінку системи при виконанні SQL-операції до таблиці.

Індекси в PostgreSQL наступних типів: B-дерево, хеш, R-дерево, GiST, GIN. При необхідності можна створювати нові типи індексів, хоча це далеко не тривіальний процес.

Багатоверсійність підтримується в PostgreSQL - можлива одновременнуя модифікація БД кількома користувачами за допомогою механізму Multiversion Concurrency Control (MVCC). Завдяки цьому дотримуються вимоги ACID, і практично відпадає потреба в блокування читання.

Розширення PostgreSQL для власних потреб можливо практично в будь-якому аспекті. Є можливість додавати власні перетворення типів, типи даних, домени (призначені для користувача типи з самого початку накладеними обмеженнями), функції (включаючи агрегатні), індекси, оператори (включаючи перевизначення вже існуючих) і процедурні мови.

Спадкування в PostgreSQL реалізовано на рівні таблиць. Таблиці можуть успадковувати характеристики і набори полів від інших таблиць (батьківських). При цьому дані, додані в породжену таблицю, автоматично будуть брати участь (якщо це не зазначено окремо) в запитах до батьківської таблиці.

У своєму проекті я використовую PostgreSQL у якості БД та можу привести

приклади створення таблиць.

У проекті є 4 таблиці з сутностями – Groups, Lessons, Students, Teachers. postgres – це логін адміна для роботи з базої даних (так кажучи, під капотом)

Код для створення таблиці Groups :

```
create table groups
(
group_id serial not nullconstraint groups_pkey primary key,
group_name varchar(100)
);
alter table groups owner to postgres;
```

Код для створення таблиці Lessons :

```
create table lessons(
lesson_id     serial not nullconstraint lessons_pkey primary key,
lesson_name  varchar(100),class_number integer,
date   timestamp,teacher_id    integer
constraint fk_lessons_teachers
references teachers, group_id  integer constraint fk_lessons_groups
references groups
);
alter table lessons
owner to postgres;
```

Код для створення таблиці Students :

```
create table students(
student_id   serial not nullconstraint students_pkey
primary key,
student_login varchar(100) not null,
student_password varchar(100) not null, student_role
varchar(100) not null, student_name varchar(100),
student_surname varchar(100),
```

```
        student_age integer, entry_year integer,
graduate_year integer, faculty_name
        varchar(100),group_id integer constraint fk_students_groups
        references groups,
        enabled boolean default true not null
        );
        alter table students
        owner to postgres;
```

Код для створення таблиці Teachers :

```
        create table teachers(
        teacher_id serial not nullconstraint teachers_pkey
        primary key,
        teacher_login varchar(100) not null,
teacher_password varchar(100) not null, teacher_role
        varchar(100) not null, teacher_name varchar(100),
        position varchar(100),
        enabled boolean default true not null
        );
        alter table teachers
        owner to postgres;
```

1.3. Автентіфікація

Автентіфіка́ція (з грец. Αυθεντικός; реальний або Істинний) - процедура встановлення належності користувачеві информации в системе пред'явленняним ідентіфікатора.

З позіцій інформаційної безпеки автентифікація є частина процедури Надання доступу для роботи в інформаційній сістемі, наступна після

ідентіфікації и передує авторізації.

1.4. Двухфакторна аутентифікація

Двухфакторна аутентифікація - це додатковий рівень захисту облікового запису. Крім введення пароля, потрібно також ввести одноразовий код, який приходить на пошту або телефон, або відбиток пальця. Цим ви підтверджуєте свою особистість. Коли ви активуєте цю опцію, крім пароля хакеру потрібно також ввести код, щоб зайти в ваш аккаунт. Ви також отримаєте повідомлення, якщо хтось спробує отримати доступ до вашої учетку.

Одноразовий код діє лише пару хвилин або годин, після чого він самознищується. Таким чином, завдяки двофакторної аутентифікації ваші онлайн-акаунти стають невразливими для кіберзлочинців.

Двухфакторная аутентифікація по SMS. Секретний одноразовий пароль приходить на мобільний користувача в SMS-повідомленні.

Переваги:

Так як приходить SMS, кожен може скористатися цією опцією - незалежно віднаявності інтернету.

Недоліки:

Сигнал мережі - головний фактор, щоб отримати SMS з кодом.

Якщо ви втратили SIM-карту або телефон, ви не зможете пройти аутентифікацію.

Двухфакторная аутентифікація по телефону. Користувачі отримують код перевірки по дзвінку після того, як вони правильно ввели пароль і ім'я користувача.

Переваги:

Так як це дзвінок, кожен може скористатися цією опцією - незалежно від наявності інтернету.

Недоліки:

Сигнал мережі - головний фактор, щоб вам додзвонилися. Якщо ви в роумінгу, це буде дорого коштувати.

Якщо ви втратили SIM-карту або телефон, ви не зможете пройти аутентифікацію.

Двухфакторная аутентифікація поштою. Користувачі отримують код перевірки або унікальне посилання на пошту післятого, як вони правильно ввели пароль і ім'я користувача.

Переваги:

Доступно на комп'ютерах і телефонах.

Недоліки:

На відміну від SMS або дзвінка, для отримання коду потрібен інтернет. Лист може потрапити в спам або не дійти через проблеми з сервером.

1.5. Висновки до першого розділу

У першому розділі дипломного проекту були описані технології, які використовувалися у розробці даного проекту. Також було порушена тема аутентифікації та двухфакторної аутентифікації.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ

2.1. Функціональні вимоги до проекту

Функціональні вимоги - це набір сервісів, які повинна виконувати система. Цифровий університет повинен задовольняти вимогам:

- юзер (будь-то студент чи вчитель) повинен авторизуватися в системі задля перегляду свого розкладу в STUDENT PERSONAL AREA чи TEACHER PERSONAL AREA відповідно до ролі;
- якщо дані (логін та пароль) невірні, то юзер це побачить, тому що на екрані з’явиться надпис “Bad credentials”, тобто невірні дані, і юзер зможе знову ввести свої дані;
- якщо юзер зайшов, як вчитель та хоче потрапити на сторінку STUDENT PERSONAL AREA, він туди не потрапить, а лише з’явиться надпис із картинкою “Access Denied”;
- аналогічно до п.3, якщо студент захоче потрапити на вкладку TEACHER PERSONAL AREA;
- студент може побачити в Student Personal Area свій розклад пар;
- вчитель також може побачити в Teacher Personal Area свій розклад пар;
- є можливість потрапити на сторінку Groups будь-якому, навіть не авторизованому юзеру;
- є можливість потрапити на сторінку Teachers будь-якому, навіть не авторизованому юзеру;
- є можливість потрапити на сторінку Students будь-якому, навіть не авторизованому юзеру;
- є можливість потрапити на сторінку Lessons будь-якому, навіть не авторизованому юзеру;

- юзер має можливість вийти з систему, тобто LOGOUT, після виходу із систему він потрапить на Welcome Page.

2.2. Нефункціональні вимоги до проекту

Нефункціональні вимоги можуть включати в себе різні обмеження: обмеження на процес розробки системи, тимчасові обмеження тощо.

Вимоги до дизайну. В першу чергу, коли ми торкаємося вимог до дизайну веб-додатку, а саме про їх розробку, то слід пам’ятати, що вони входять до процесу проектування сайту. Вимоги до дизайну мають давати можливість керувати сайтом будь-якому користувачеві, навіть без знання веб-програмування. Звісно, як будуть складені вимоги до дизайну і, як вони будуть реалізовані, така і ефективність буде сайту. Адже дизайн — друга за важливістю складова ефективного сайту. Від нього залежить, наскільки приваблива сторінка, а це врешті-решт додатково привертає потенційних користувачів та клієнтів.

Програмні вимоги до інтерфейсу. Дані вимоги повинні бути реалізовані на рівні програмних компонентів, з яких складається весь веб-сервіс для моніторингу авіарейсів. Мова йде про підтримку платформ, базу даних та інших програмних структур. Всі вони детальніше описані далі;

1) системою управління базами даних повинна бути реляційна PostgreSQL;
2) мова запитів – SQL;
3) кросплатформеність – підтримка всіх браузерів.

2.3. Вимоги до бази даних

Основними вимогами до бази даних є:

1) цілісність бази даних
2) багаторазове використання даних
3) швидкий пошук та отримання інформації по запитам користувачів
4) простота оновлення даних
5) зменшення надмірності даних
6) захист даних від несанкціонованого копіювання

2.4. Висновки до другого розділу

У другому розділі були порушені функціональні вимоги, нефункціональні вимоги та вимоги до бази даних

РОЗДІЛ 3. СТРУКТУРА ПРОЕКТУ

3.1. Model-View-Controller

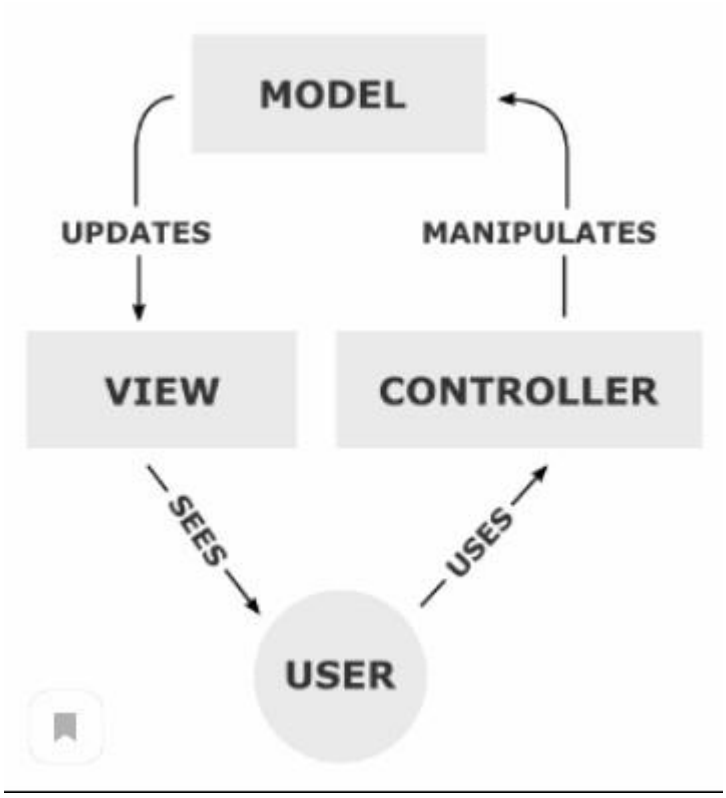


Рис. 3.1. Model-View-Controller

Проект працює на основі технології MVC (Model-View-Controller).

Модель (Model) надає дані та реагує на команди контролера, змінюючи свійстан.

Представлення (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.

Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель пронеобхідність змін.

3.2. Розкриття поняття MVC

Model (Модель)

Компонент моделі зберігає дані і пов'язану з ними логіку. Він являє собою дані, які передаються між компонентами контролера або будь-який інший пов'язаної бізнес-логікою. Наприклад, об'єкт контролера буде витягувати інформацію про клієнта з бази даних. Він маніпулює даними і відправляє їх назад в базу даних або використовує її для візуалізації тих же даних.

Він реагує на запит від уявлень, а також реагує на інструкції контролера по оновленню самого себе. Це також найнижчий рівень патерну, який відповідає за збереження даних.

View (Вид)

Уявлення - це та частина програми, яка представляє уявлення даних.

Уявлення створюються на основі даних, зібраних з даних моделі. Подання запитує у моделі інформацію таким чином, щоб вона обурювалася поданням виведення користувачеві.

Подання також представляє дані з чатів, діаграм і таблиць. Наприклад, будь- яке представлення клієнта буде включати в себе всі компоненти користувальницького інтерфейсу, такі як текстові поля, списки, що випадають і т. Д.

Controller (Контролер)

Контролер-це та частина програми, яка обробляє взаємодія з користувачем. Контролер інтерпретує вводи миші і клавіатури від користувача, інформуючи модель і уявлення про зміну в міру необхідності.

Контролер відправляє команди моделі для поновлення її стану (наприклад, збереження певного документа). Контролер також посилає команди пов'язаному з ним поданням для зміни уявлення уявлення (наприклад, прокручування певного документа).

3.3. Висновки до третього розділу

У третьому розділі була розкрита тема MVC, а саме – Model-View-Controller.Це те, як в цілому працює наш проект та за що відповідає Модель, Вид на Контролер.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ЦИФРОВОГО УНІВЕРСИТЕТУ

4.1. Опис використаних технологій

Проект розроблен на мові програмування Java за допомогою середовища розробки INTELIJ IDEA.

У процесі розробки був використан фреймворк Spring.

Базу даних було використано PostgreSQL, де знаходяться дані користувачів.

Сторінки, які бачить користувач у браузері – це html-сторінки, до яких здійснюється перехід за допомогою контролерів.

У репозиторіях здійснюються запити до бази даних, а сервси реалізовані так, що в їх методах викликаються методи репозиторіїв.

4.2. Запуск проекту

Запуск веб-серверу стартує з класу DyplomProjectApplication.java, за допомогою анотації @SpringBootApplication та точки входу, метода “publicstatic void main(String[] args)”.

```
@SpringBootApplication
public class DyplomProjectApplication {
public static void main(String[] args) {
SpringApplication.run(DyplomProjectApplication.class, args);
}
}
```

Сервер відкриває порт 8069, попередньо запущений сервер БД PostgreSQL на порту 5432.

Усі значення конфігурації зберігаються у файлі application.properties. База

даних має назву testdb, дані для входу у БД також є в файлі нижче.

```
server.port=8069
spring.datasource.url=jdbc:postgresql://localhost:5432/testdb
spring.datasource.username=postgres spring.datasource.password=postgres
```

4.3. Репозиторії проекту

Для проведення запитів до БД реалізовано інтерфейси GroupRepository, LessonRepository, StudentRepository, TeacherRepository, кожен з яких успадковує інтерфейс CrudRepository, в якому за допомогою фреймворку Hibernate прописані CRUD операції (Create, Read, Update, Delete).

Класи с репозиторіями знаходяться в пакеті com.dyplom.repository.

GroupRepository :

```
public interface GroupRepository extends CrudRepository<Group, Long> {
@Query("SELECT groupId FROM Group WHERE groupName = ?1")
Integer getGroupIdByName(String groupName);
@Query("FROM Group WHERE groupId =?1") Group
getGroupById(Integer groupId);
@Modifying
@Query("UPDATE Group SET groupName = ?1 WHERE groupId = ?2")
void changeGroupName(String groupName, Integer groupId);
@Query("SELECT COUNT(G) FROM Group G")
Integer getCountOfGroups();
}
```

LessonRepository :

```
public interface LessonRepository extends CrudRepository<Lesson, Long> {
@Query(" SELECT NEW LessonDTO (L.lessonId, L.lessonName,
L.classNumber, L.date, T.teacherName, " +
```

```
        "G.groupName) FROM Lesson L LEFT JOIN Teacher T ONL.teacherId =
T.teacherId LEFT JOIN Group G ON " +
        "L.groupId=G.groupId WHERE L.lessonId = ?1") LessonDTO
getLessonDtoById(int lessonId);
        @Query(" SELECT NEW LessonDTO (L.lessonId, L.lessonName,
L.classNumber, L.date, T.teacherName, " +
        "G.groupName) FROM Lesson L LEFT JOIN Teacher T ONL.teacherId =
T.teacherId LEFT JOIN Group G ON " +
        "L.groupId=G.groupId ORDER BY L.lessonId") List<LessonDTO>
getAllLessonDtos();
        @Modifying
        @Query("DELETE FROM Lesson L WHERE L.groupId =?1") void
deleteAllDataWithSpecifiedGroup(Integer groupId);
        @Query("FROM Lesson WHERE lessonId =?1") Lesson
getLessonById(Integer lessonId);
        @Modifying
        @Query("UPDATE Lesson SET lessonName = ?1,classNumber = ?2, date =
        ?3, teacherId = ?4, groupId =?5 WHERE lessonId = ?6")
        void editLesson(String lessonName, Integer classNumber, Date date, Integer
teacherId, Integer groupId, Integer lessonId);
        @Query("SELECT COUNT(L) FROM Lesson L")
        Integer getCountOfLessons();
        @Query(" SELECT NEW LessonDTO (L.lessonId, L.lessonName,
L.classNumber, L.date, T.teacherName, " +
        "G.groupName) FROM Lesson L LEFT JOIN Teacher T ONL.teacherId =
T.teacherId LEFT JOIN Group G ON " +
        "L.groupId=G.groupId WHERE L.groupId=?1 ORDER BYL.lessonId")
        List<LessonDTO> getScheduleForStudent(Integer groupId);
        @Query(" SELECT NEW LessonDTO (L.lessonId, L.lessonName,
L.classNumber, L.date, T.teacherName, " +
```

```
"G.groupName) FROM Lesson L LEFT JOIN Teacher T ONL.teacherId =
T.teacherId LEFT JOIN Group G ON " +
"L.groupId=G.groupId WHERE L.teacherId=?1 ORDER BYL.lessonId")
List<LessonDTO> getScheduleForTeacher(Integer teacherId);
}
```

StudentRepository :

```
public interface StudentRepository extends CrudRepository<Student, Long> {
@Query("SELECT NEW StudentDTO (S.studentId, S.studentName,
S.studentSurname, S.studentAge, S.entryYear," +
"S.graduateYear, S.facultyName, G.groupName) FROM Student S LEFT
JOIN Group G ON S.groupId=G.groupId ORDER BY S.studentId")
List<StudentDTO> getAllStudentDtos();
@Modifying
@Query("UPDATE Student SET studentName = ?1,studentSurname = ?2,"
+
"studentAge = ?3, entryYear = ?4, graduateYear = ?5, facultyName = ?6,
groupId = ?7 WHERE studentId = ?8")
void editStudent(String studentName, String studentSurname, Integer
studentAge,Integer entryYear,
Integer graduateYear, String facultyName, Integer groupId, Integer
studentId);
@Query("SELECT NEW StudentDTO (S.studentId, S.studentName,
S.studentSurname, S.studentAge, S.entryYear," +
"S.graduateYear, S.facultyName, G.groupName) FROM Student S LEFT
JOIN Group G ON S.groupId=G.groupId WHERE S.studentId=?1")
StudentDTO getStudentDtoById(Integer studentId);
@Query("FROM Student WHERE studentId =?1") Student
getStudentById(Integer studentId);
@Modifying
@Query("DELETE FROM Student S WHERE S.groupId =?1") void
```

```
deleteAllDataWithSpecifiedStudent(Integer groupId);
    @Query("SELECT COUNT(S) FROM Student S") Integer getCountOfStudents();
    @Query("FROM Student WHERE studentLogin=?1") Student findByLogin(String username);
    @Query("SELECT s.groupId FROM Student s WHERE s.studentLogin=?1")Integer getGroupIdByStudentLogin(String studentLogin);
    }
```

TeacherRepository :

```
public interface TeacherRepository extends CrudRepository<Teacher, Long> {
    @Query("FROM Teacher WHERE teacherId = ?1") Teacher getTeacherById(Integer teacherId);
    @Modifying
    @Query("UPDATE Teacher SET teacherName = ?1, position = ?2 WHERE teacherId = ?3")
    void editTeacher(String teacherName, String position, Integer teacherId);
    @Query("SELECT teacherId FROM Teacher WHERE teacherName = ?1")Integer getTeacherIdByName(String name);
    @Query("SELECT COUNT(T) FROM Teacher T") Integer getCountOfTeachers();
    @Query("FROM Teacher WHERE teacherLogin=?1") Teacher findBylogin(String username);
    @Query("SELECT teacherId FROM Teacher WHERE teacherLogin= ?1")Integer getTeacherIdByTeacherLogin(String teacherLogin);
    }
```

4.4. Бізнес-логіка проекту

Бізнес-логіка проекту знаходиться в пакеті com.dyplom.service :

GroupService :

```
@Service @Transactional
public class GroupService {
@Autowired GroupRepository repo;
@Autowired LessonRepository lessonRepo;
@Autowired
StudentRepository studentRepo;
public Integer getGroupIdByName(String groupName) {
return repo.getGroupIdByName(groupName);
}
public List<Group> getAll() {
return (List<Group>) repo.findAll();
}
public Group getGroupById(Integer groupId) {
return repo.getGroupById(groupId);
}
public void addGroup(Group group) {
repo.save(group);
}
public void changeGroupName(Group group) {
repo.changeGroupName(group.getGroupName(), group.getGroupId());
}
public void removeGroup(Group group) {
lessonRepo.deleteAllDataWithSpecifiedGroup(group.getGroupId());
studentRepo.deleteAllDataWithSpecifiedStudent(group.getGroupId());
repo.delete(group);
```

```
}
public Integer getCountOfGroups() {
return repo.getCountOfGroups();
}
}
```

LessonService :

```
@Service @Transactional
public class LessonService {
@Autowired LessonRepository repo;
public void removeLesson(Lesson lesson) {
repo.delete(lesson);
}
public LessonDTO getLessonDtoById(Integer lessonId) {
return repo.getLessonDtoById(lessonId);
}
public List<LessonDTO> getAllLessonDtos() {
return repo.getAllLessonDtos();
}
public void addLesson(Lesson lesson) {
repo.save(lesson);
}
public Lesson getLessonById(int lessonId) {
return repo.getLessonById(lessonId);
}
public void editLesson(Lesson lesson) {
repo.editLesson(lesson.getLessonName(),
lesson.getClassNumber(), lesson.getDate(), lesson.getTeacherId(),
lesson.getGroupId(),
lesson.getLessonId());
}
```

```
public Integer getCountOfLessons() {
return repo.getCountOfLessons();
}
public List<LessonDTO> getScheduleForStudent(Integer groupId) {
return repo.getScheduleForStudent(groupId);
}
public List<LessonDTO> getScheduleForTeacher(Integer teacherId) {
return repo.getScheduleForTeacher(teacherId);
}
}
```

StudentService :

```
@Service @Transactional
public class StudentService {
@Autowired StudentRepository repo;
public List<StudentDTO> getAllStudentDtos() {
return repo.getAllStudentDtos();
}
public void addStudent(Student student) {
repo.save(student);
}
public void editStudent(Student student) {
repo.editStudent(student.getStudentName(), student.getStudentSurname(),
student.getStudentAge(),
student.getEntryYear(), student.getGraduateYear(),student.getFacultyName(),
student.getGroupId(), student.getStudentId());
}
public StudentDTO getStudentDtoById(Integer studentId) {
return repo.getStudentDtoById(studentId);
}
public void removeStudent(Student student) {
```

```
repo.delete(student);
}
public Student getStudentById(Integer studentId) {
return repo.getStudentById(studentId);
}
public Integer getCountOfStudents() {
return repo.getCountOfStudents();
}
public Student findByLogin(String username) {
return repo.findByLogin(username);
}
public Integer getGroupIdByStudentLogin(String studentLogin) {
return repo.getGroupIdByStudentLogin(studentLogin);
}
}
```

TeacherService :

```
@Service @Transactional
public class TeacherService {
@Autowired TeacherRepository repo;
public List<Teacher> getAll() {
return (List<Teacher>) repo.findAll();
}
public void addTeacher(Teacher teacher) {
repo.save(teacher);
}
public void removeTeacher(Teacher teacher) {
repo.delete(teacher);
}
public Teacher getTeacherById(Integer teacherId) {
return    repo.getTeacherById(teacherId);
```

```
}
public void editTeacher(Teacher teacher) {
repo.editTeacher(teacher.getTeacherName(), teacher.getPosition(),
teacher.getTeacherId());
}
public Integer getTeacherIdByName(String name) {
return repo.getTeacherIdByName(name);
}
public Integer getCountOfTeachers() {
return repo.getCountOfTeachers();
}
public Teacher findByLogin(String username) {
return repo.findBylogin(username);
}
public Integer getTeacherIdByTeacherLogin(String teacherLogin) {
return   repo.getTeacherIdByTeacherLogin(teacherLogin);
}
}
```

4.5. Сутності

Сутності в програмі знаходяться в пакеті com.dyplom.model. Саме вони є основними POJO об’єктами для створення таблиць в БД : Group.java, Lesson.java, Student.java, Teacher.java

Group :

```
@Entity @ToString@Setter @Getter
@NoArgsConstructor@AllArgsConstructor
@Table(name = "groups", schema = "public")public class Group {
@Id
```

@Column(name = "group_id") @GeneratedValue(strategy = GenerationType.*IDENTITY*)**private int groupId**;

@NotNull

@Column(name = "group_name")**private** String **groupName**;

}

Lesson:

@Entity @ToString@Setter @Getter

@NoArgsConstructor@AllArgsConstructor

@Table(name = "lessons", schema = "public")**public class** Lesson {

@Id

@Column(name = "lesson_id") @GeneratedValue(strategy = GenerationType.*IDENTITY*)**private int lessonId**;

@Column(name = "lesson_name") **private** String **lessonName**; @Column(name = "class_number")

private int classNumber; @Column(name = "date") **private** Date **date**; @Column(name = "teacher_id") **private int teacherId**; @Column(name = "group_id") **private int groupId**;

public Lesson(String lessonName, Integer classNumber, Timestamp date, IntegerteacherId, Integer groupId) {

this.**lessonName** = lessonName; **this**.**classNumber** = classNumber;**this**.**date** = date;

this.**teacherId** = teacherId;

this.**groupId** = groupId;

}

}

Student :

@Entity @ToString@Setter @Getter

@NoArgsConstructor@AllArgsConstructor

@Table(name = "students", schema = "public")**public class** Student {

@Id

@Column(name = "student_id") @GeneratedValue(strategy = GenerationType.*IDENTITY*)**private int studentId**;

@Column(name = "student_name")**private** String **studentName**;

@Column(name = "student_login")**private** String **studentLogin**;

@Column(name = "student_password")**private** String **studentPassword**;

@Column(name = "student_role")**private** String **studentRole**;

@Column(name = "student_surname")

private String **studentSurname**; @Column(name = "student_age") **private int studentAge**; @Column(name = "entry_year") **private int entryYear**; @Column(name = "graduate_year")**private int graduateYear**; @Column(name = "faculty_name") **private** String **facultyName**; @Column(name = "group_id") **private int groupId**;

}

Teacher :

@Entity @ToString@Setter @Getter

@NoArgsConstructor@AllArgsConstructor

@Table(name = "teachers", schema = "public")**public class** Teacher {

@Id

@Column(name = "teacher_id") @GeneratedValue(strategy = GenerationType.*IDENTITY*)**private int teacherId**;

@Column(name = "teacher_name")@NotNull

private String **teacherName**;

@Column(name = "teacher_login")**private** String **teacherLogin**;

@Column(name = "teacher_password")**private** String **teacherPassword**;

@Column(name = "teacher_role")**private** String **teacherrole**;

@Column(name = "position")@NotNull

private String **position**;

4.6. DTO (Data Transfer Object)

Для представлення об’єктів використана технологія DTO. Всі класи знаходяться в пакеті com.dyplom.dto

LessonDTO :

```
@Entity @ToString@Setter @Getter
@NoArgsConstructor @AllArgsConstructor public class LessonDTO {
@Id
private int id; private String name;
private int classNumber;
@DateTimeFormat(pattern = "yyyy-dd-MM HH:mm:ss")private Date date;
private String teacherName;private String groupName;
}
```

LessonDTOForm :

```
@ToString@Setter @Getter
@NoArgsConstructor @AllArgsConstructor
public class LessonDTOForm {
private int id;@NotNull
@Size(min = 2, max = 45, message = "Lesson nameshould be not less than 2 and not more than 45!")
private String name;@NotNull
@Min(value = 1, message = "Class number can not benull!")
@Max(value = 999, message = "Class number can not bemore than 999!")
private int classNumber;@NotNull
@Size(min = 3, max = 75, message = "Teacher nameshould be more than 3 and less than 75!")
private String teacherName;@NotNull
@Size(min = 4, max = 4, message = "Group name shouldconsist of 4 symbols!")
```

```
private String groupName;@NotNull
@Min(value = 2020, message = "Lesson year can not beless than 2020!")
@Max(value = 2030, message = "Lesson year can not bemore than 2030!")
private int year;@NotNull
@Min(value = 1, message = "Month can not be less than
1!")
@Max(value = 12, message = "Month can not be more
than 12!")
private int month;@NotNull
@Min(value = 1, message = "Day can not be less than
1!")
@Max(value = 31, message = "Day can not be more than
31!")
private int day;@NotNull
@Min(value = 1, message = "Hour can not be less than
1!")
@Max(value = 24, message = "Hour can not be more than
24!")
private int hour;@NotNull
@Min(value = 1, message = "Minute can not be lessthan 1!")
@Max(value = 60, message = "Minute can not be morethan 60!")
private int minute;
}
```

StudentDTO :

```
@Entity @ToString@Setter @Getter
@NoArgsConstructor @AllArgsConstructor public class StudentDTO {
@Id
private int studentId;@NotNull
@Size(min = 2, max = 30, message = "Name should consist of 2 to 30
symbols!")
```

```
    private String studentName;@NotNull
    @Size(min = 2, max = 30, message = "Surname should consist of 2 to 30 symbols!")
    private String studentSurname;@NotNull
    @Min(value = 10, message = "Student age should be more than 10!")private int studentAge;
    @NotNull
    @Min(value = 1900, message = "Entry year should be more than 1900!") @Max(value = 2021, message = "Entry year should be less than 2021!") private int entryYear;
    @NotNull
    @Min(value = 2020, message = "Graduate year should be not less than 2020!")private int graduateYear;
    @NotNull
    @Size(min = 3, message = "Faculty name should consist of minimum 3 symbols!")
    private String facultyName;@NotNull
    @Size(min = 4, max = 4,message = "Group name should consist of 4 symbols!")
    private String groupName;
    }
```

4.7. Controllers

Контролериdля прийому GET та POSTзапитів лежать в пакетіcom.dyplom.controller :

WebController :

```
@Controller
public class WebController {
```

```
@RequestMapping(path = "/accessDeniedPage", method = RequestMethod.GET)
public String home() {
return "accessDeniedPage";
}
@RequestMapping(path = "/", method = RequestMethod.GET)
public String welcome() {
return "welcomePage";
}
}
```

GroupController :

```
@Controller @RequestMapping("/groupViews")public class GroupController {
private static final String GROUP_MODEL = "group";@Autowired
GroupService groupService;
@Autowired
StudentService studentService;
@GetMapping("groups")
public String groupsPage(Model model) { List<Group> groupList = groupService.getAll(); model.addAttribute("groups", groupList); return "/groupViews/groupsPage";
}
@PostMapping("successGroupEdition")
public String editGroup(@ModelAttribute("group") Group group) {
Group updatedGroup = groupService.getGroupById(group.getGroupId()); updatedGroup.setGroupName(group.getGroupName()); groupService.changeGroupName(updatedGroup);
return "/groupViews/successGroupEdition";
}
```

```
@GetMapping("editGroup")
public String editGroup(Model model, @RequestParam(value = "groupId") Integer groupId) {
Group group = groupService.getGroupById(groupId); model.addAttribute(GROUP_MODEL, group); return "/groupViews/editGroup";
}
@PostMapping("successGroupAddition")
public String addGroup(@ModelAttribute("group") @Validated Group group, Errors errors, Model model) {
if (errors.hasErrors()) { model.addAttribute(GROUP_MODEL, group);return "/groupViews/addGroup";
}
groupService.addGroup(group);
return "/groupViews/successGroupAddition";
}
@GetMapping("addGroup")
public String addStudent(Model model) { model.addAttribute(GROUP_MODEL, new Group()); return "/groupViews/addGroup";
}
@GetMapping("groupDescription")
public String readGroup(Model model, @RequestParam(value = "groupId") Integer groupId) {
model.addAttribute(GROUP_MODEL, groupService.getGroupById(groupId)); return "/groupViews/groupDescription";
}
@GetMapping("removeGroup")
public String removeStudent(Model model, @RequestParam(value = "groupId")Integer groupId) {
```

```
groupService.removeGroup(groupService.getGroupById(groupId)); return "/groupViews/removeGroup";
}
```

LessonController :

```
@Controller @RequestMapping("/lessonViews") public class LessonController {
private static final String LESSON_MODEL = "lesson";@Autowired
LessonService lessonService;
@Autowired
TeacherService teacherService;
@Autowired
GroupService groupService;
@GetMapping("editLesson")
public String editLesson(Model model, @RequestParam(value = "lessonId") Integer lessonId) {
LessonDTO lessonDTO = lessonService.getLessonDtoById(lessonId); model.addAttribute(LESSON_MODEL, lessonDTO);
return "/lessonViews/editLesson";
}
@GetMapping("lessons")
public String lessonsPage(Model model) {
List<LessonDTO> lessonDTOList = lessonService.getAllLessonDtos(); model.addAttribute("lessons", lessonDTOList);
return "/lessonViews/lessonsPage";
}
@PostMapping("successLessonEdition")
public String editLessonName(@ModelAttribute("lesson") LessonDTO lessonDTO) {
Lesson lesson = new Lesson(lessonDTO.getId(), lessonDTO.getName(), lessonDTO.getClassNumber(),
```

```
        lessonDTO.getDate(),
        teacherService.getTeacherIdByName(lessonDTO.getTeacherName()),
groupService.getGroupIdByName(lessonDTO.getGroupName()));
        lessonService.editLesson(lesson);
        return "/lessonViews/successLessonEdition";
        }
        @PostMapping("successLessonAddition")
        public String addLesson(@ModelAttribute("lesson") @Validated
LessonDTOForm lessonDTOForm, Errors errors, Model model) {
        if (errors.hasErrors()) { model.addAttribute(LESSON_MODEL,
lessonDTOForm); return "/lessonViews/addLesson";
        }
        Timestamp lessonDate = new Timestamp(lessonDTOForm.getYear() - 1900,
lessonDTOForm.getMonth() + 1,
        lessonDTOForm.getDay(), lessonDTOForm.getHour(),
lessonDTOForm.getMinute(), 00, 00);
        Lesson lesson = new Lesson(lessonDTOForm.getName(),
lessonDTOForm.getClassNumber(), lessonDate,
teacherService.getTeacherIdByName(lessonDTOForm.getTeacherName()),
        groupService.getGroupIdByName(lessonDTOForm.getGroupName()));
lessonService.addLesson(lesson);
        return "/lessonViews/successLessonAddition";
        }
        @GetMapping(value = "lessonDescription")
        public String readStudent(Model model, @RequestParam(value = "lessonId")
Integer lessonId) {
        LessonDTO lessonDTO = lessonService.getLessonDtoById(lessonId);
model.addAttribute(LESSON_MODEL, lessonDTO);
        return "/lessonViews/lessonDescription";
        }
```

```
@GetMapping("removeLesson")
public String removeLesson(Model model, @RequestParam(value = "lessonId")int lessonId) {
lessonService.removeLesson(lessonService.getLessonById(lessonId)); return "/lessonViews/removeLesson";
}
@GetMapping("addLesson")
public String addStudent(Model model) { model.addAttribute(LESSON_MODEL, new LessonDTOForm()); return "/lessonViews/addLesson";
}
}
```

StudentController :

```
@Controller @RequestMapping("/studentViews") public class StudentController {
private static final String STUDENT_MODEL = "student";@Autowired
StudentService studentService;@Autowired
TeacherService teacherService;@Autowired
GroupService groupService;@Autowired
LessonService lessonService;
@GetMapping("students")
public String showStudents(Model model) {
List<StudentDTO> students = studentService.getAllStudentDtos(); model.addAttribute("students", students);
return "/studentViews/studentsPage";
}
@GetMapping("schedule")
public String schedule(Model model) {Authentication loggedInUser =
SecurityContextHolder.getContext().getAuthentication(); String studentLogin =
```

```
loggedInUser.getName();
        Integer groupId = studentService.getGroupIdByStudentLogin(studentLogin);
List<LessonDTO> lessons = lessonService.getScheduleForStudent(groupId);
model.addAttribute("lessons", lessons);
        return "/studentViews/studentSchedule";
        }

        @GetMapping("studentPersonalArea")
        public String studentPersonalArea(Model model) {
        return    "/studentViews/studentPersonalArea";
        }

        @GetMapping("editStudent")
        public String editStudent(Model model, @RequestParam(value = "studentId")
Integer studentId) {
        StudentDTO studentDTO = studentService.getStudentDtoById(studentId);
model.addAttribute(STUDENT_MODEL, studentDTO);
        return "/studentViews/editStudent";
        }
        @GetMapping("addStudent")
        public String addStudent(Model model) {
model.addAttribute(STUDENT_MODEL, new StudentDTO()); return
"/studentViews/addStudent";
        }
        @GetMapping(value = "studentDescription")
        public String readStudent(Model model, @RequestParam(value =
"studentId")Integer studentId) {
        StudentDTO studentDTO = studentService.getStudentDtoById(studentId);
model.addAttribute("studentDto", studentDTO);
        return "/studentViews/studentDescription";
```

```
    }

    @GetMapping("removeStudent")
    public String removeStudent(Model model, @RequestParam(value =
    "studentId") Integer studentId) {
studentService.removeStudent(studentService.getStudentById(studentId)); return
"/studentViews/removeStudent";
    }
    }
```

TeachertController :

```
    @Controller @RequestMapping("/teacherViews") public class
TeacherController {
    private static final String TEACHER_MODEL = "teacher";@Autowired
    LessonService lessonService;
    @Autowired
    TeacherService teacherService;
    @GetMapping("schedule")
    public String schedule(Model model) {Authentication loggedInUser =
    SecurityContextHolder.getContext().getAuthentication(); String teacherLogin =
loggedInUser.getName(); Integer teacherId =
    teacherService.getTeacherIdByTeacherLogin(teacherLogin);
    List<LessonDTO> lessons = lessonService.getScheduleForTeacher(teacherId);
    model.addAttribute("lessons", lessons);
    return "/teacherViews/teacherSchedule";
    }
    @GetMapping("teacherPersonalArea")
    public String teacherPersonalArea(Model model) {
    return    "/teacherViews/teacherPersonalArea";
    }
```

```
@GetMapping("addTeacher")
public String addTeacher(Model model) {
model.addAttribute(TEACHER_MODEL, new Teacher()); return
"teacherViews/addTeacher";
}
@GetMapping("teachers")
public String showTeachers(Model model) { List<Teacher> teacherDTOs =
teacherService.getAll();model.addAttribute("teachers", teacherDTOs);
return "/teacherViews/teachersPage";
}
@GetMapping("removeTeacher")
public String removeTeacher(Model model, @RequestParam(value =
"teacherId") Integer teacherId) {
teacherService.removeTeacher(teacherService.getTeacherById(teacherId)); return
"/teacherViews/removeTeacher";
}
@GetMapping("editTeacher")
public String editTeacher(Model model, @RequestParam(value =
"teacherId")Integer teacherId) {
Teacher teacher = teacherService.getTeacherById(teacherId);
model.addAttribute(TEACHER_MODEL, teacher);
return "/teacherViews/editTeacher";
}

@GetMapping(value = "teacherDescription")
public String readTeacher(Model model, @RequestParam(value =
"teacherId")Integer teacherId) {
Teacher teacher = teacherService.getTeacherById(teacherId);
model.addAttribute(TEACHER_MODEL, teacher);
return "/teacherViews/teacherDescription";
```

```
    }
    @PostMapping("successTeacherAddition")
    public String addTeacher(@ModelAttribute("teacher") @Validated Teacher teacher, Errors errors, Model model) {
    if (errors.hasErrors()) { model.addAttribute(TEACHER_MODEL, teacher); return "/teacherViews/addTeacher";
    }
    teacherService.addTeacher(teacher);
    return "/teacherViews/successTeacherAddition";
    }
    @PostMapping("successTeacherEdition")
    public String editTeacherName(@ModelAttribute("teacher") Teacher teacher) {
    teacherService.editTeacher(teacher);
    return "/teacherViews/successTeacherEdition";
    }
    }
```

4.8. HTML-сторінки

HTML сторінки знаходяться у папці templates:

GroupPage :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>UNIVERSITY</title>
```

```
<style>
body {
background-color: #c7b39b;
}
a {
font-size: 45pt;
}
.container {
font-size: 30pt;
}
</style>
</head>
<body>

<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/studentViews/students" target="_blank"><big>Students</big></a>
<a class="click-me" href="/groupViews/groups" target="_blank"><big>Groups</big></a>
<a class="click-me" href="/teacherViews/teachers" target="_blank"><big>Teachers</big></a>
<a class="click-me" href="/lessonViews/lessons" target="_blank"><big>Lessons</big></a>
</div>
<div class="container">
<h2 align="center">GROUPS</h2>
<table class="table table-dark" border="1" width="100%" cellpadding="5">
```

```
<thead>
<tr>
<th> ID</th>
<th> Group</th>
</tr>
</thead>
<tbody>
<tr th:if="${groups.empty}">
<td colspan="2"> No groups Available</td>
</tr>
<tr th:each="group : ${groups}">
<td><span th:text="${group.groupId}"></span></a></td>
<td><span th:text="${group.groupName}"></span></td>
</tr>
</tbody>
</table>
</div>
</body>
</html>
```

LessonPage :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>UNIVERSITY</title>
<style>
body {
background-color: #c7b39b;
```

```
}
a {
font-size: 45pt;
}
.container {
font-size: 30pt;
}
</style>
</head>
<body>

<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/studentViews/students" target="_blank"><big>Students</big></a>
<a class="click-me" href="/groupViews/groups" target="_blank"><big>Groups</big></a>
<a class="click-me" href="/teacherViews/teachers" target="_blank"><big>Teachers</big></a>
<a class="click-me" href="/lessonViews/lessons" target="_blank"><big>Lessons</big></a>
</div>
<div class="container">
<h2 align="center">LESSONS</h2>
<table class="table table-dark" border="1" width="100%" cellpadding="5">
<thead>
<tr>
<th> ID</th>
```

```
<th> NAME</th>
<th> CLASS NUMBER</th>
<th> DATE</th>
<th> TEACHER</th>
<th> GROUP</th>
</tr>
</thead>
<tbody>
<tr th:if="${lessons.empty}">
<td colspan="2"> No lessons Available</td>
</tr>
<tr th:each="lesson : ${lessons}">
<td><span th:text="${lesson.id}"></span></a></td>
<td><span th:text="${lesson.name}"></span></td>
<td><span th:text="${lesson.classNumber}"></span></td>
<td><span th:text="${lesson.date}"></span></td>
<td><span th:text="${lesson.teacherName}"></span></td>
<td><span th:text="${lesson.groupName}"></span></td>
</tr>
</tbody>
</table>
</div>
</body>
</html>
```

StudentPage :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
```

```
1">
        <title>UNIVERSITY</title>
        <style>
        body {
        background-color: #c7b39b;
        }
        a {
        font-size: 45pt;
        }
        .container {
        font-size: 30pt;
        }
        </style>
        </head>
        <body>
        <div align="center">
        <a class="click-me" href="/" target="_blank"><big>Welcome
Page</big></a>
        <a class="click-me" href="/studentViews/students"
target="_blank"><big>Students</big></a>
        <a class="click-me" href="/groupViews/groups"
target="_blank"><big>Groups</big></a>
        <a class="click-me" href="/teacherViews/teachers"
target="_blank"><big>Teachers</big></a>
        <a class="click-me" href="/lessonViews/lessons"
target="_blank"><big>Lessons</big></a>
        </div>
        <div class="container">
        <h2 align="center">STUDENTS</h2>
        <table class="table table-dark" border="1" width="100%"
```

```
cellpadding="5">
        <thead>
        <tr>
        <th> ID</th>
        <th> Name</th>
        <th> Surname</th>
        <th> Age</th>
        <th> Entry Year</th>
        <th> Graduate Year</th>
        <th> Faculty</th>
        <th> Group</th>
        </tr>
        </thead>
        <tbody>
        <tr th:if="${students.empty}">
        <td colspan="2"> No students Available</td>
        </tr>
        <tr th:each="student : ${students}">
        <td><span th:text="${student.studentId}"></span></a></td>
        <td><span th:text="${student.studentName}"></span></td>
        <td><span th:text="${student.studentSurname}"></span></td>
        <td><span th:text="${student.studentAge}"></span></td>
        <td><span th:text="${student.entryYear}"></span></td>
        <td><span th:text="${student.graduateYear}"></span></td>
        <td><span th:text="${student.facultyName}"></span></td>
        <td><span th:text="${student.groupName}"></span></td>
        </tr>
        </tbody>
        </table>
        </div>
```

```
</body>
</html>
```

StudentSchedule :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>UNIVERSITY</title>
<style>
body {
background-color: #c7b39b;
}
a {
font-size: 45pt;
}
.container {
font-size: 30pt;
}
</style>
</head>
<body>
<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome
Page</big></a>
<a class="click-me" href="/studentViews/students"
target="_blank"><big>Students</big></a>
```

```
<a class="click-me" href="/groupViews/groups" target="_blank"><big>Groups</big></a>
<a class="click-me" href="/teacherViews/teachers" target="_blank"><big>Teachers</big></a>
<a class="click-me" href="/lessonViews/lessons" target="_blank"><big>Lessons</big></a>
</div>
<div class="container">
<h2 align="center">LESSONS</h2>
<table class="table table-dark" border="1" width="100%" cellpadding="5">
<thead>
<tr>
<th> ID</th>
<th> NAME</th>
<th> CLASS NUMBER</th>
<th> DATE</th>
<th> TEACHER</th>
<th> GROUP</th>
</tr>
</thead>
<tbody>
<tr th:if="${lessons.empty}">
<td colspan="2"> No lessons Available</td>
</tr>
<tr th:each="lesson : ${lessons}">
<td><span th:text="${lesson.id}"></span></a></td>
<td><span th:text="${lesson.name}"></span></td>
<td><span th:text="${lesson.classNumber}"></span></td>
<td><span th:text="${lesson.date}"></span></td>
```

```
<td><span th:text="${lesson.teacherName}"></span></td>
<td><span th:text="${lesson.groupName}"></span></td>
</tr>
</tbody>
</table>
</div>
</body>
</html>
```

StudentPersonalArea :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>UNIVERSITY</title>
<style>
body {
background-color: #c7b39b;
}
a {
font-size: 45pt;
}
.center {
display: block; margin-left: auto; margin-right: auto;width: 50%;
}
button {
height: 100px;
width: 150px;
}
```

```
</style>
</head>
<body>
<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/studentViews/schedule" target="_blank"><big>My schedule</big></a>
</div>
<div align="center">
<form class="form-inline my-2 my-lg-0" form-method="post" th:action="@{/logout}">
<button class="btn btn-outline-danger my-2 my-sm-0 btn-sm" type="submit"><big>Logout</big></button>
</form>
</div>
<br>
<img class="center" src="https://pk.nau.edu.ua/wp-content/uploads/2019/07/nau_21.jpg">
</body>
</html>
```

TeacherPage :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>UNIVERSITY</title>
<style>
```

```
body {
background-color: #c7b39b;
}
a {
font-size: 45pt;
}
.container {
font-size: 30pt;
}
</style>
</head>
<body>
<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/studentViews/students" target="_blank"><big>Students</big></a>
<a class="click-me" href="/groupViews/groups" target="_blank"><big>Groups</big></a>
<a class="click-me" href="/teacherViews/teachers" target="_blank"><big>Teachers</big></a>
<a class="click-me" href="/lessonViews/lessons" target="_blank"><big>Lessons</big></a>
</div>
<div class="container">
<h2 align="center">TEACHERS</h2>
<table class="table table-dark" border="1" width="100%" cellpadding="5">
<thead>
<tr>
```

```
<th> ID</th>
<th> Name</th>
<th> Position</th>
</tr>
</thead>
<tbody>
<tr th:if="${teachers.empty}">
<td colspan="2"> No teachers Available</td>
</tr>
<tr th:each="teacher : ${teachers}">
<td><span th:text="${teacher.teacherId}"></span></a></td>
<td><span th:text="${teacher.teacherName}"></span></td>
<td><span th:text="${teacher.position}"></span></td>
</tr>
</tbody>
</table>
</div>
</body>
</html>
```

TeacherSchedule :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>UNIVERSITY</title>
<style>
body {
background-color: #c7b39b;
```

```
}
a {
font-size: 45pt;
}
.container {
font-size: 30pt;
}
</style>
</head>
<body>
<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/studentViews/students"
target="_blank"><big>Students</big></a>
<a class="click-me" href="/groupViews/groups" target="_blank"><big>Groups</big></a>
<a class="click-me" href="/teacherViews/teachers" target="_blank"><big>Teachers</big></a>
<a class="click-me" href="/lessonViews/lessons" target="_blank"><big>Lessons</big></a>
</div>
<div class="container">
<h2 align="center">LESSONS</h2>
<table class="table table-dark" border="1" width="100%" cellpadding="5">
<thead>
<tr>
<th> ID</th>
<th> NAME</th>
```

```
<th> CLASS NUMBER</th>
<th> DATE</th>
<th> TEACHER</th>
<th> GROUP</th>
</tr>
</thead>
<tbody>
<tr th:if="${lessons.empty}">
<td colspan="2"> No lessons Available</td>
</tr>
<tr th:each="lesson : ${lessons}">
<td><span th:text="${lesson.id}"></span></a></td>
<td><span th:text="${lesson.name}"></span></td>
<td><span th:text="${lesson.classNumber}"></span></td>
<td><span th:text="${lesson.date}"></span></td>
<td><span th:text="${lesson.teacherName}"></span></td>
<td><span th:text="${lesson.groupName}"></span></td>
</tr>
</tbody>
</table>
</div>
</body>
</html>
```

TeacherPersonalArea :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
```

```
<title>UNIVERSITY</title>
<style>
body {
background-color: #c7b39b;
}
a {
font-size: 45pt;
}
.center {
display: block; margin-left: auto; margin-right: auto;width: 50%;
}
button {
height: 100px;
width: 150px;
}
</style>
</head>
<body>
<div align="center">
<a class="click-me" href="/" target="_blank"><big>Welcome Page</big></a>
<a class="click-me" href="/teacherViews/schedule" target="_blank"><big>My schedule</big></a>
</div>
<div align="center">
<form class="form-inline my-2 my-lg-0" form-method="post" th:action="@{/logout}">
<button class="btn btn-outline-danger my-2 my-sm-0 btn-sm" type="submit"><big>Logout</big></button>
</form>
```

```
</div>
<br>
<img class="center"
src="https://fastly.4sqi.net/img/general/600x600/115924791_hlMfbYwWTu3QJJ L7Yotf-UEvplQzeuD_TZHIcfu5j30.jpg">
</body>
</html>
```

Access denied page :

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>University</title>
</head>
<body>
<h1>Access is Denied!</h1>
<img class="center"
src="https://statusycitaty.ru/images/stories/Image/Prikolnie/pechalnye- statusy.jpg"
width="50%">
</body>
</html>
```

Welcome page :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>UNIVERSITY</title>
<style>
body {
```

```
        background-color: #c7b39b;
        }
        a {
        font-size: 45pt;
        }
        .center {
        display: block; margin-left: auto; margin-right: auto;width: 50%;
        }
        .personalArea {
        }
        </style>
        </head>
        <body>
        <div align="center">
        <a class="click-me" href="/studentViews/students"
target="_blank"><big>Students</big></a>
        <a class="click-me" href="/groupViews/groups"
target="_blank"><big>Groups</big></a>
        <a class="click-me" href="/teacherViews/teachers"
target="_blank"><big>Teachers</big></a>
        <a class="click-me" href="/lessonViews/lessons"
target="_blank"><big>Lessons</big></a>
        </div>
        <div align="center" class="personalArea">
        <a class="click-me" href="/studentViews/studentPersonalArea"
target="_blank"><big>STUDENT PERSONAL AREA</big></a>
        <br>
        <a class="click-me" href="/teacherViews/teacherPersonalArea"
target="_blank"><big>TEACHER PERSONAL AREA</big></a>
        </div>
```

```
<img class="center" src="https://diplom24.com.ua/wp-content/uploads/2018/03/nau-fly.jpg">
</body>
</html>
```

4.9. Висновки до четвертого розділу

В четвертому розділі була реалізована бізнес-логіка проекту, репозиторії проекту, сутності, контролери та були описані використані технології.

РОЗДІЛ 5. Результати тестування проекту

5.1. Тестування

При запуску проекту ми потрапляємо на головну сторінку. При запуску проекту ми потрапляємо на головну сторінку

Рис. 5.1. Welcome Page

Потрапити на вкладки Student, Groups, Teachers and Lessons можна без авторизації будь-якому користувачі, навіть не авторизованому, тому що там нема прив’язки до моделі об’єкта, там тільки список користувачів у БД.

Нижче наглядно можна це побачити.

Вкладка Students

Welcome PageStudentsGroupsTeachersLessons

STUDENTS

ID	Name	Surname	Age	Entry Year	Graduate Year	Faculty	Group
1	Alexandr	Platonchik	37	1999	2004	journalism	STEN
2	Yevhenii	Haverz	19	2015	2019	psychology	STEN
3	Roman	Petrov	45	1998	2002	astronomy	TLEK
4	Maria	Grosu	18	2020	2024	cybersecurity	FLOK
5	Dmytry	Fleks	28	1991	1995	journalism	LOBS
6	Svetlana	Bolshunova	31	1994	1998	jurisprudence	BAOS
7	Antonina	Ivanchenko	59	1986	1990	art	KOLB
8	Arkady	Panchenko	22	2017	2021	psychology	STEN
9	Petr	Scvortsov	20	2020	2024	art	AOLB
10	Vasyly	Andreev	45	1984	1988	philosophy	AOLB

Рис. 5.2. Students

Вкладка Groups

Welcome PageStudentsGroupsTeachersLessons

GROUPS

ID	Group
1	BIKS
2	STEN
3	TLEK
4	FLOK
5	BAOS
6	LOBS
7	KOLB
8	AOLB
9	BLOK
10	STOL

Рис. 5.3. Groups

Вкладка Teachers

Welcome PageStudentsGroupsTeachersLessons

TEACHERS

ID	Name	Position
1	Lyudmula Vysotskaya	assistant
2	Diana Pavlova	assistant
3	Arina Lomovaya	main teacher

Рис. 5.4. Teachers

Вкладка Lessons

Welcome PageStudentsGroupsTeachersLessons

LESSONS

ID	NAME	CLASS NUMBER	DATE	TEACHER	GROUP
1	Math	225	2021-06-22 16:00:00.0	Diana Pavlova	STEN
2	English	315	2021-06-21 13:15:00.0	Diana Pavlova	TLEK
3	Programming	405	2021-06-22 14:00:00.0	Diana Pavlova	TLEK
4	World history	259	2021-06-24 08:45:00.0	Diana Pavlova	TLEK
5	Spain	364	2021-06-23 16:15:00.0	Diana Pavlova	TLEK

Рис. 5.5. Lessons

А вже для того, щоб потрапити на вкладки STUDENT PERSONAL AREA and TEACHER PERSONAL AREA треба бути авторизованим.

Розглянемо вкладку Student Personal Area :

Рис. 5.6. Перехід на Student Personal Area

Після нажаття на “Student Personal Area” користувач потрапляє на сторінкуавторизації :

Рис. 5.7. Сторінка авторизації

Якщо користувач введе невірний логін чи пароль, то в доступі до сторінки будевідмовлено та буде показана помилка “BAD CREDENTIALS” :

Рис. 5.8. Невірні дані при авторизації

Якщо користувач зайде під логіном і паролем Teacher’а, то він також не потрапить на цю сторінку, тому що на ній доступ мають лише студенти. Спроба потрапити вчителю:

Рис. 5.9. Відмова у доступі Користувачу відмовлено в доступі.

Якщо ми все же таки введемо вірні дані студента, то потрапимо на сторінку:

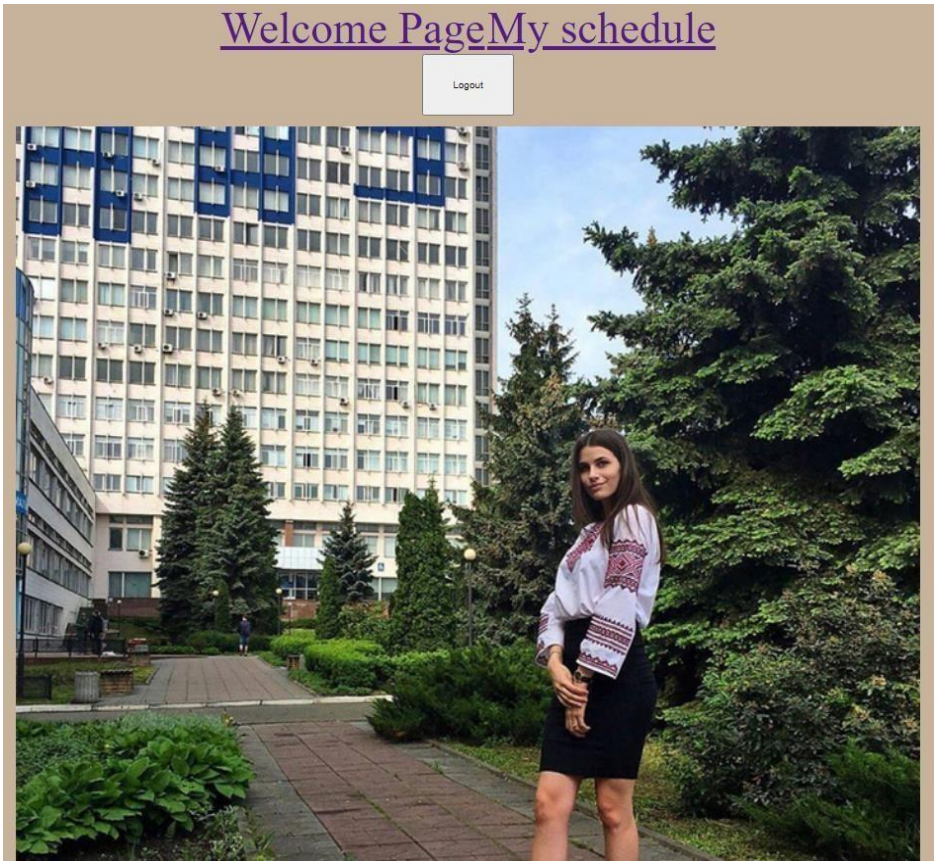


Рис. 5.10. Персональний кабінет студента

Тут також ми можемо:

1) Повернутись на Welcome Page

Рис. 5.11. Повернення на Welcome Page

2) Побачити розклад конкретно авторизованого студента

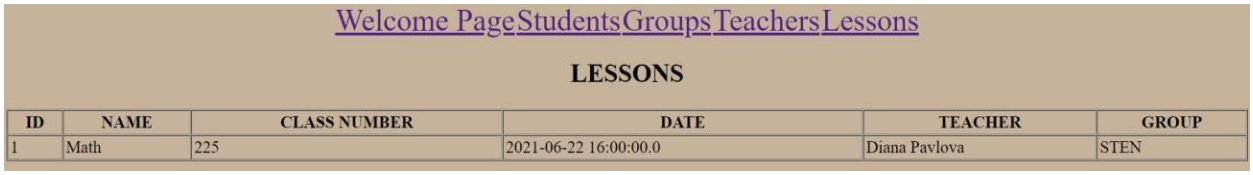


Рис. 5.12. Розклад студента

3)вийти з аккаунта, нажавши button “Logout”

І після виходу з аккаунту ми потрапимо на нашу Welcome Page

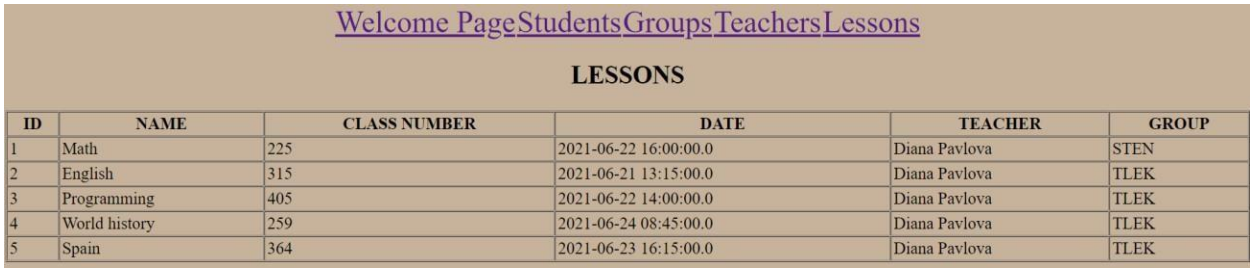
Аналогічна система і для Teacher Personal Area :Авторизовуємося

Рис. 5.13. Авторизація вчителя

Потрапляємо на сторінку Teacher’a :

Рис. 5.14. Персональний кабінет вчителя

Клацаємо на My Schedule та бачимо розклад конкретно авторизованого вчителя



ID	NAME	CLASS NUMBER	DATE	TEACHER	GROUP
1	Math	225	2021-06-22 16:00:00.0	Diana Pavlova	STEN
2	English	315	2021-06-21 13:15:00.0	Diana Pavlova	TLEK
3	Programming	405	2021-06-22 14:00:00.0	Diana Pavlova	TLEK
4	World history	259	2021-06-24 08:45:00.0	Diana Pavlova	TLEK
5	Spain	364	2021-06-23 16:15:00.0	Diana Pavlova	TLEK

Рис. 5.15. Розклад вчителя

Ну і при бажанні вийти з аккаунту, це можна зробити аналогічно, нажавшиbutton Logout.

5.2. Висновки до п’ятого розділу

У п’ятому розділі була показана детальна реалізація проекту та демонстрація його роботи.

ВИСНОВКИ

В ході даної курсової роботи був розроблен проект на мові програмування Java в середі розробки INTELIJ IDEA.

Використан фреймворк Spring, для зберігання даних використав базу данихPostgrelSQL.

За допогомогою MVC-controller’s можна потрапляти на html – сторінки увеб-браузері.

Дійсно, у цифровому університеті в наш час є велика потреба, оскільки навчання проводиться дистанційно, та як, якщо не через інтернет десь дивитися розклад, вчителів и т.д.

Так само і для вчителів, за допомогою цифрового університета вчитель може побачити розклад не тільки всіх пар, але і окремо для себе, що є дуже зручним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java - Вікіпедія [Електронний ресурс] https://uk.wikipedia.org/wiki/Java

2. Intelij Idea - Вікіпедія [Електронний ресурс] https://uk.wikipedia.org/wiki/IntelliJ_IDEA

3. Spring Framework – Вікіпедія [Електронний ресурс] https://uk.wikipedia.org/wiki/Spring_Framework

4. Обзор модулей Spring для Java [Електроний ресурс] https://tproger.ru/articles/spring-modules-overview/#:~:text=Spring%20Boot%20—%20комплексный%20фреймворк,обрабатывать%20огромное%20количество%20одновременных%20подключений

5. Spring Boot | Spring по-русски! [Електроний ресурс]https://spring-projects.ru/projects/spring-boot/

6. Кофе-брейк #75. Преимущества и недостатки Spring Boot. [Електронний ресурс] https://javarush.ru/groups/posts/3380-kofe-breyk-75-preimujshestva-i-nedostatki-ispoljhzovanija-spring-boot-funkcii-dlja-strok-v-java

7. Spring Security | Spring по-русски! [Електронний ресурс]http://spring-projects.ru/projects/spring-security/

8. Что такое HTML и CSS? [Електронний ресурс] https://help.megagroup.ru/chto-takoye-html-i-css#:~:text=CSS%20-%20язык%20таблиц%20стилей%2C,описывать%20оформление%20внешнего%20вида%20контента

9. Що таке база даних? | Кафедра АПЕПС ТЕФ КПІ – програмна інженерія[Електронний ресурс] https://apeps.kpi.ua/shco-take-basa-danykh

10. SQL – Википедия [Електронний ресурс] https://ru.wikipedia.org/wiki/SQL

11. PostgreSQL – популярная свободная объектно-реляционная система управления базами данных [Електронний ресурс]

https://web-creator.ru/articles/postgresql

12. SQL | DDL, DQL , DML , DCL AND TCL commands – GeeksforGeeks [Електронний ресурс]

https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

13. Что такое Двухфакторная аутентификация пользователя | Блог SSL.com.ua[Електронний ресурс]

https://ssl.com.ua/blog/what-is-2fa/

14. Model-View-Controller – Википедия [Електронний ресурс] https://ru.wikipedia.org/wiki/Model-View-Controller