Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Паламарчук Катерина Юріївна,
студент групи АІ-233

КУРСОВА РОБОТА

Розробка серверної частини прикладного програмного забезпечення з використанням мови програмування Java та фреймворку Spring

Спеціальність:
122 Комп’ютерні науки

Освітня програма: Комп’ютерні науки

Керівник:
М.А. Годовиченко

Одеса – 2025

ЗМІСТ

Вступ……..….3
Аналіз предметної області …… …….....5
1 Розробка шару Rest-контролерів……..…….7
2 Розробка шару сервісів ……..……...…11
3 Розробка шару репрозиторіїі…….……16
4 Додавання реєстрації користувача за допомогою логіну та паролю……..…...19
5 Додавання автентифікації за допомогою логіну та паролю…….…….23
6 Додавання автентифікації за допомогою JWT–токену ……..…27
7 Реєстрація та автентифікація користувача за допомогою протоколу OAuth2 ….31
Висновки…….……...…34
Список використаних джерел…….…….35

ВСТУП

У сучасному світі онлайн-освіта відіграє дедалі важливішу роль. Завдяки розвитку цифрових технологій та зростанню попиту на дистанційне навчання виникає потреба в ефективних системах управління курсами, які дозволяють організовувати навчальний процес, стежити за прогресом студентів і підтримувати взаємодію між учасниками освітнього процесу.

Метою цієї курсової роботи є розробка веб застосунку — системи управління онлайн-курсами, що дозволяє створювати курси, реєструвати студентів, додавати навчальні модулі та відстежувати прогрес користувачів.

Завдання, що були поставлені в межах курсової роботи:

– реалізація функціоналу REST API для керування курсами, модулями, студентами, зарахуванням і прогресом;

– забезпечення реєстрації користувачів за логіном і паролем;

– реалізація автентифікації через JWT;

– інтеграція автентифікації через OAuth2;

– забезпечення зберігання даних за допомогою JPA/Hibernate.

Предметна область — система онлайн-навчання. Основні користувачі — адміністратори або викладачі (які створюють курси) та студенти (які проходять навчання). Система повинна забезпечити ефективну організацію навчального процесу.

Обрані технології:

– Java 17 — сучасна мова програмування з підтримкою ООП, зручна для створення вебзастосунків;

– Spring Boot — фреймворк для швидкої розробки веб-API та мікросервісів;

– Spring Security — для реалізації автентифікації та авторизації;

– JWT, OAuth2 — сучасні способи захисту та ідентифікації користувачів у вебзастосунках

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Характеристика системи

Система управління онлайн-курсами (варіант 15) призначена для автоматизації процесів створення та проходження навчальних курсів. Основні функції системи:

– створення курсів та додавання до них модулів;

– реєстрація та керування студентами;

– зарахування студентів на курси;

– фіксація прогресу проходження курсів.

Основні користувачі системи

Адміністратор/викладач — створює навчальні курси, додає модулі, переглядає список студентів, відстежує їхній прогрес.

Студент — реєструється у системі, переглядає доступні курси, записується на курс, проходить модулі, бачить свій прогрес.

Основні бізнес-процеси (сценарії використання)

а) Реєстрація та вхід до системи

Студент може зареєструватися за email/логіном і паролем або увійти за допомогою Google/Facebook через OAuth2.

б) Перегляд курсів

Користувач отримує список доступних курсів. Є можливість фільтрувати їх за категоріями.

в) Запис на курс

Після вибору курсу студент записується на нього. В системі створюється запис (Enrollment), який буде зберігати прогрес.

г) Проходження модулів

Студент бачить перелік модулів, що входять до курсу, та відмічає кожен як завершений.

д) Відстеження прогресу

На основі завершених модулів система обчислює відсоток проходження курсу.

Типова поведінка системи

Користувач взаємодіє із фронтендом, який відправляє HTTP-запити до REST API. Контролери обробляють запити, звертаються до сервісів, які взаємодіють із репозиторіями для зчитування або зміни даних. Всі дії, які вимагають автентифікації, перевіряються за допомогою JWT або OAuth2 токенів.

1 РОЗРОБКА ШАРУ REST-КОНТРОЛЕРІВ

1.1 Розробка контролера CourseController

REST-контролер CourseController відповідає за обробку запитів, пов’язаних із сутністю онлайн-курсу. Основними функціями є перегляд усіх курсів, перегляд курсу за ідентифікатором, створення, оновлення та видалення курсу.

Для прикладу, метод для перегляду одного курсу реалізований наступним чином:

```
    @GetMapping("/{id}")
    public                    ResponseEntity<OnlineCourse>
getCourseById(@PathVariable Long id) {
                    Optional<OnlineCourse>    course    =
courseService.getCourseById(id);
                                                    return
course.map(ResponseEntity::ok).orElseGet(()              ->
ResponseEntity.notFound().build());
    }
```

Цей метод обробляє HTTP GET-запит на шлях /api/courses/{id} та повертає об'єкт курсу або повідомлення про помилку 404, якщо курс не знайдено.

1.2 Розробка контролера StudentController

Контролер StudentController обробляє запити, пов'язані з операціями над студентами: створення, перегляд, оновлення, видалення.

Метод для створення студента виглядає так:

```
@PostMapping
public Student createStudent(@RequestBody Student student) {
    return studentService.createStudent(student);
}
```

Цей метод приймає об’єкт Student у тілі запиту та зберігає його в базі даних.

1.3 Розробка контролера EnrollmentController

Контролер EnrollmentController реалізує логіку запису студента на курс та отримання записів. Зокрема, реалізовано методи для отримання всіх записів студента або курсу, а також можливість відписатися від курсу.

```
@PostMapping
public Enrollment enrollStudentToCourse(@RequestParam Long studentId, @RequestParam Long courseId) {
                                                    return enrollmentService.enrollStudentToCourse(studentId, courseId);
}
```

Цей метод використовує параметри studentId та courseId, щоб створити запис про зарахування студента на курс.

1.4 Розробка контролера ModuleController

Контролер ModuleController дозволяє працювати з модулями курсу: створення, редагування, перегляд. Контролер підтримує вкладену маршрутизацію — модулі прив’язані до конкретного курсу.

```
    @GetMapping
    public List<Module> getModulesByCourseId(@PathVariable Long courseId) {
        return moduleService.getModulesByCourseId(courseId);
    }
```

Цей метод повертає список модулів для зазначеного курсу.

1.5 Розробка контролера ProgressController

Контролер ProgressController відповідає за облік прогресу студента в курсі. Наприклад, метод updateModuleProgress дозволяє оновити статус проходження модуля.

```
    @PostMapping
    public Progress updateModuleProgress(
            @RequestParam Long enrollmentId,
            @RequestParam Long moduleId,
            @RequestParam Boolean completed) {
        return progressService.updateModuleProgress(enrollmentId, moduleId, completed);
    }
```

Цей метод приймає ідентифікатори зарахування, модуля та статус завершення (true/false), після чого зберігає зміни в базі даних.

2 РОЗРОБКА ШАРУ СЕРВІСІВ

Шар сервісів у проєкті реалізує бізнес-логіку, яка виконується між рівнем контролерів та репозиторіями. Кожна сутність має відповідний сервісний клас, який інкапсулює обробку даних, перевірки та взаємодію з базою даних.

2.1 Розробка сервісу CourseService

Сервіс CourseService забезпечує повний CRUD-функціонал для роботи з курсами. Він взаємодіє з репозиторієм CourseRepository та реалізує логіку створення, оновлення, видалення та отримання курсів.

Приклад методу оновлення курсу:

```
public OnlineCourse updateCourse(Long id, OnlineCourse
courseDetails) {
    OnlineCourse course =
courseRepository.findById(id).orElseThrow();
    course.setTitle(courseDetails.getTitle());
    course.setDescription(courseDetails.getDescription());
    course.setCategory(courseDetails.getCategory());
    return courseRepository.save(course);
}
```

Цей метод дозволяє змінити інформацію про курс, і зберігає оновлений об’єкт у базі даних.

2.2 Розробка сервісу StudentService

Сервіс StudentService відповідає за створення, оновлення та пошук студентів у системі. Він реалізує логіку перевірки та перетворення даних перед передачею до репозиторію.

Метод створення нового студента:

```
public Student createStudent(Student student) {
    return studentRepository.save(student);
}
```

2.3 Розробка сервісу EnrollmentService

Сервіс EnrollmentService обробляє логіку запису студентів на курси, включаючи перевірку наявності попереднього зарахування.
Приклад методу зарахування студента на курс:

```
public Enrollment enrollStudentToCourse(Long studentId,
Long courseId) {
    if
(enrollmentRepository.existsByStudentIdAndCourseId(student
Id, courseId)) {
        throw new RuntimeException("Student is already
enrolled in this course");
    }

    Student student =
studentRepository.findById(studentId).orElseThrow();
    OnlineCourse course =
courseRepository.findById(courseId).orElseThrow();
```

```
    Enrollment enrollment = new Enrollment();
    enrollment.setStudent(student);
    enrollment.setCourse(course);
    enrollment.setEnrolledAt(LocalDate.now());

    return enrollmentRepository.save(enrollment);
}
```

Цей метод перевіряє, чи вже студент зарахований на вказаний курс, і лише тоді створює новий запис.

2.4 Розробка сервісу ModuleService

Сервіс ModuleService реалізує логіку роботи з модулями курсу: створення, редагування, перегляд. Кожен модуль пов’язаний із певним курсом.

Приклад методу створення модуля:

```
public Module createModule(Long courseId, Module module) {
    OnlineCourse course =
courseRepository.findById(courseId).orElseThrow();
    module.setCourse(course);
    return moduleRepository.save(module);
}
```

2.5 Розробка сервісу ProgressService

Сервіс ProgressService дозволяє фіксувати прогрес студента у модулях курсу та обчислювати відсоток завершення.

Метод updateModuleProgress дозволяє створити або оновити запис про проходження модуля:

```
public Progress updateModuleProgress(Long enrollmentId,
Long moduleId, Boolean completed) {
    Enrollment enrollment =
enrollmentRepository.findById(enrollmentId).orElseThrow();
    Module module =
moduleRepository.findById(moduleId).orElseThrow();

    Progress progress =
progressRepository.findByEnrollmentIdAndModuleId(enrollmen
tId, moduleId)
            .orElseGet(() -> {
                Progress newProgress = new Progress();
                newProgress.setEnrollment(enrollment);
                newProgress.setModule(module);
                return newProgress;
            });

    progress.setCompleted(completed);
    return progressRepository.save(progress);
}
```

А метод getCourseProgressPercentage() обчислює, скільки модулів студент завершив:

```
public double getCourseProgressPercentage(Long
enrollmentId) {
```

```
    List<Progress> progresses =
progressRepository.findByEnrollmentId(enrollmentId);
    if (progresses.isEmpty()) return 0;

    long completed =
progresses.stream().filter(Progress::getCompleted).count()
;
    return (double) completed / progresses.size() * 100;
}
```

3 РОЗРОБКА ШАРУ РЕПОЗИТОРІЇВ

Шар репозиторіїв у додатку реалізовано за допомогою Spring Data JPA. Кожен інтерфейс репозиторію розширює JpaRepository, що забезпечує базову підтримку CRUD-операцій без необхідності писати SQL-запити вручну. У разі необхідності використовуються іменовані методи (findBy..., existsBy...) для генерації специфічних запитів.

3.1 Розробка CourseRepository

Інтерфейс CourseRepository відповідає за доступ до таблиці курсів. Крім стандартних методів, реалізовано запит для отримання курсів за категорією:

```
@Repository
public interface CourseRepository extends
JpaRepository<OnlineCourse, Long> {
    List<OnlineCourse> findByCategory(String category);
}
```

Цей метод дозволяє знайти всі курси, що належать до певної категорії.

3.2 Розробка StudentRepository

StudentRepository — це простий репозиторій без додаткових методів, оскільки всі необхідні операції забезпечує базовий функціонал JpaRepository.

```
@Repository
```

```
public interface StudentRepository extends
JpaRepository<Student, Long> {

}
```

3.3 Розробка EnrollmentRepository

EnrollmentRepository використовується для обробки зарахувань студентів на курси. Реалізовано кілька методів пошуку:

```
@Repository

public interface EnrollmentRepository extends
JpaRepository<Enrollment, Long> {

    List<Enrollment> findByStudentId(Long studentId);

    List<Enrollment> findByCourseId(Long courseId);

    boolean existsByStudentIdAndCourseId(Long studentId,
Long courseId);

}
```

Ці методи дозволяють: отримати всі записи за студентом або курсом, перевірити, чи студент вже записаний на певний курс.

3.4 Розробка ModuleRepository

```
    ModuleRepository відповідає за доступ до модулів
курсів:

@Repository

public interface ModuleRepository extends
JpaRepository<Module, Long> {
```

```
    List<Module> findByCourseId(Long courseId);
}
```

Цей метод дозволяє отримати список усіх модулів, що належать певному курсу.

3.5 Розробка ProgressRepository

ProgressRepository працює з об'єктами, що зберігають прогрес студентів у модулях.

```
@Repository
public interface ProgressRepository extends
JpaRepository<Progress, Long> {
    List<Progress> findByEnrollmentId(Long enrollmentId);
    List<Progress> findByModuleId(Long moduleId);
    Optional<Progress> findByEnrollmentIdAndModuleId(Long
enrollmentId, Long moduleId);
}
```

Важливим є метод findByEnrollmentIdAndModuleId, який дає змогу знайти унікальний запис про прогрес за конкретним модулем для конкретного зарахування. Якщо запису ще немає, він створюється у ProgressService.

4 ДОДАВАННЯ РЕЄСТРАЦІЇ КОРИСТУВАЧА ЗА ДОПОМОГОЮ ЛОГІНУ ТА ПАРОЛЮ

4.1 Реалізація моделі користувача

Для зберігання облікових записів користувачів використовується сутність User, яка реалізує інтерфейс UserDetails:

```
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role;
}
```

Користувач має унікальні ім’я користувача (логін), email та зашифрований пароль. Також вказується роль користувача (наприклад, STUDENT чи ADMIN).

4.2 DTO для реєстрації

Для прийому даних від клієнта використовується DTO-клас SignUpRequest:

```
@Data
public class SignUpRequest {
    @NotBlank
    @Size(min = 4, max = 40)
    private String username;

    @NotBlank
    @Size(min = 6, max = 20)
    private String password;

    @NotBlank
    @Email
    private String email;
}
```

4.3 Реалізація реєстрації користувача в AuthController

Контролер AuthController обробляє запит на реєстрацію:

```
@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody
SignUpRequest signUpRequest) {
    if
(userService.existsByUsername(signUpRequest.getUsername())
) {
        return ResponseEntity.badRequest().body(new
ApiResponse(false, "Username taken"));
    }
```

```
    if
(userService.existsByEmail(signUpRequest.getEmail())) {
        return ResponseEntity.badRequest().body(new
ApiResponse(false, "Email use!"));
    }

    userService.registerStudent(signUpRequest);
    return ResponseEntity.ok(new ApiResponse(true,
"Successfully"));
}
```

Перед реєстрацією перевіряється унікальність логіну та email. Якщо все коректно, викликається сервіс UserService.

4.4 Реєстрація користувача в UserService

Логіка створення облікового запису та зв’язаної сутності Student виконується у UserService:

```
public User registerStudent(SignUpRequest signUpRequest) {
    User user = new User();
    user.setUsername(signUpRequest.getUsername());
    user.setEmail(signUpRequest.getEmail());

user.setPassword(passwordEncoder.encode(signUpRequest.getP
assword()));
    user.setRole(Role.ROLE_STUDENT);

    User savedUser = userRepository.save(user);

    Student student = new Student();
    student.setName(user.getUsername());
```

```
    student.setEmail(user.getEmail());
    student.setUser(savedUser);
    studentRepository.save(student);

    return savedUser;
}
```

Пароль шифрується за допомогою BCryptPasswordEncoder. Після створення користувача, автоматично створюється об’єкт Student, пов’язаний з цим користувачем.

5 ДОДАВАННЯ АВТЕНТИФІКАЦІЇ ЗА ДОПОМОГОЮ ЛОГІНУ ТА ПАРОЛЮ

Для забезпечення автентифікації за логіном та паролем у Spring Boot проєкті реалізовано механізм із використанням AuthenticationManager, JWT-токенів і кастомного сервісу користувачів.

5.1 DTO для входу в систему

Запити на вхід надходять через DTO-клас LoginRequest, який містить логін (або email) та пароль:

```
public class LoginRequest {
    @NotBlank
    private String usernameOrEmail;

    @NotBlank
    private String password;
}
```

5.2 Обробка автентифікації у AuthController

Метод authenticateUser() контролера AuthController обробляє POST-запити на /api/auth/signin:

```
@PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid
@RequestBody LoginRequest loginRequest) {
    Authentication authentication =
authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(
            loginRequest.getUsernameOrEmail(),
            loginRequest.getPassword()
```

```
        )
    );


SecurityContextHolder.getContext().setAuthentication(authe
ntication);
    String jwt =
tokenProvider.generateToken(authentication);

    return ResponseEntity.ok(new
JwtAuthenticationResponse(jwt));
}
```

Якщо введені дані коректні, генерується JWT-токен, який клієнт зберігає на своєму боці.

5.3 Реалізація CustomUserDetailsService

Клас CustomUserDetailsService завантажує користувача з бази даних за ім’ям користувача або ID:

```
@Override
public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new
UsernameNotFoundException("User not found with username: "
+ username));
    return UserPrincipal.create(user);
}
```

Клас UserPrincipal реалізує інтерфейс UserDetails і слугує адаптером між сутністю User та Spring Security.

5.4 Конфігурація безпеки SecurityConfig

Конфігурація Spring Security налаштована для використання токенів замість сесій:

```
@Bean
public SecurityFilterChain
securityFilterChain(HttpSecurity http,
JwtAuthenticationFilter jwtAuthenticationFilter) throws
Exception {
    http.csrf(csrf -> csrf.disable())
        .exceptionHandling(exception ->
exception.authenticationEntryPoint(unauthorizedHandler))
        .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATEL
ESS))
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**").permitAll()
            .anyRequest().authenticated()
        );

    http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

5.5 Автентифікація користувача в базі

Користувачі зберігаються у базі даних, а їхні паролі шифруються. Для шифрування використовується PasswordEncoder:

```
@Bean
public PasswordEncoder passwordEncoder() {
```

```
    return new BCryptPasswordEncoder();
}
```

Під час автентифікації Spring автоматично розшифровує пароль та порівнює його з введеним користувачем.

6 ДОДАВАННЯ АВТЕНТИФІКАЦІЇ ЗА ДОПОМОГОЮ JWT-ТОКЕНУ

JWT (JSON Web Token) використовується для безпечної передачі інформації між клієнтом і сервером після успішної автентифікації. У цьому проєкті JWT-токени формуються після входу користувача і використовуються для доступу до захищених ресурсів.

6.1 Генерація JWT-токену

Генерація токену відбувається у класі JwtTokenProvider, який створює токен із вказаним терміном дії та підписує його:

```
public String generateToken(Authentication authentication)
{
    UserPrincipal userPrincipal = (UserPrincipal)
authentication.getPrincipal();

    Date now = new Date();
    Date expiryDate = new Date(now.getTime() +
jwtExpirationInMs);

    return Jwts.builder()
        .setSubject(Long.toString(userPrincipal.getId()))
        .setIssuedAt(now)
        .setExpiration(expiryDate)
        .signWith(key)
        .compact();
}
```

Метод generateToken() викликається після автентифікації у AuthController.

6.2 Збереження токену на клієнті

Сервер повертає JWT у відповіді на вхід. На клієнті токен зберігається в localStorage, sessionStorage або у вигляді cookie.

6.3 Валідація токену

На кожен запит до захищеного ресурсу клієнт надсилає токен у заголовку Authorization. Валідація токена відбувається у класі JwtTokenProvider:

```
public boolean validateToken(String authToken) {
    try {

Jwts.parserBuilder().setSigningKey(key).build().parseClaim
sJws(authToken);
        return true;
    } catch (JwtException ex) {
        return false;
    }
}
```

Якщо токен невалідний або термін його дії сплив, запит буде відхилено.

6.4 Фільтрація запитів з токеном

Фільтр JwtAuthenticationFilter перехоплює кожен запит і перевіряє, чи містить він JWT:

```
protected void doFilterInternal(HttpServletRequest
request, HttpServletResponse response, FilterChain
filterChain) throws IOException, ServletException {
    String jwt = getJwtFromRequest(request);
```

```
    if (StringUtils.hasText(jwt) &&
tokenProvider.validateToken(jwt)) {
        Long userId = tokenProvider.getUserIdFromJWT(jwt);
        UserDetails userDetails =
customUserDetailsService.loadUserById(userId);
        UsernamePasswordAuthenticationToken authentication
= new UsernamePasswordAuthenticationToken(
            userDetails, null,
userDetails.getAuthorities()
        );
        authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));


SecurityContextHolder.getContext().setAuthentication(authe
ntication);
    }

    filterChain.doFilter(request, response);
}
```

Метод getJwtFromRequest() отримує токен із заголовку запиту:

```
private String getJwtFromRequest(HttpServletRequest
request) {
    String bearerToken =
request.getHeader("Authorization");
    if (StringUtils.hasText(bearerToken) &&
bearerToken.startsWith("Bearer ")) {
        return bearerToken.substring(7);
```

```
    }
    return null;
}
```

6.5 Додавання фільтра до SecurityConfig

У конфігурації безпеки токен-фільтр реєструється перед стандартним фільтром автентифікації:

```
http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
```

7 РЕЄСТРАЦІЯ ТА АВТЕНТИФІКАЦІЯ КОРИСТУВАЧА ЗА ДОПОМОГОЮ ПРОТОКОЛУ OAuth2

Для зручної реєстрації та входу через сторонні сервіси (наприклад, Google або Facebook) у проєкті реалізовано підтримку протоколу OAuth2. Це дозволяє користувачам автентифікуватися без введення пароля безпосередньо на сайті, використовуючи існуючі акаунти.

7.1 Основні компоненти реалізації

Підтримка OAuth2 базується на таких класах:
– CustomOAuth2UserService — кастомний сервіс для обробки даних користувача;
– OAuth2AuthenticationSuccessHandler — обробка успішної автентифікації;
– TokenProvider — генерація токена після успішного входу;
– CookieUtils — збереження токена у cookie;
– OAuth2Config — конфігурація клієнтів OAuth2;
– SecurityConfig — загальна конфігурація безпеки.

7.2 Конфігурація клієнтів OAuth2

Клас OAuth2Config використовує дані з application.yml/application.properties для налаштування Google/Facebook-клієнтів:

```
@Configuration
@EnableConfigurationProperties(OAuth2ClientProperties.clas
s)
public class OAuth2Config {
    private ClientRegistration getRegistration(String
client) {
        if ("google".equals(client)) {
```

```
            return
CommonOAuth2Provider.GOOGLE.getBuilder(client)
                .clientId(...)
                .clientSecret(...)
                .scope("email", "profile")
                .build();
        }
           }
}
```

7.3 Отримання користувача через OAuth2

Клас CustomOAuth2UserService перехоплює дані користувача, що повертаються від Google/Facebook:

```
@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest)
throws OAuth2AuthenticationException {
    OAuth2User oAuth2User = super.loadUser(userRequest);
    return new CustomOAuth2User(oAuth2User);
}
```

Клас CustomOAuth2User обгортає отриманого користувача та надає доступ до атрибутів, зокрема email.

7.4 Обробка успішної автентифікації

Після успішної автентифікації клас OAuth2AuthenticationSuccessHandler генерує токен і додає його у cookie:

```
@Override
public void onAuthenticationSuccess(HttpServletRequest
request, HttpServletResponse response, Authentication
authentication) throws IOException {
```

```
    String token =
tokenProvider.createToken(authentication);
    CookieUtils.addCookie(response, "token", token, 1800);
    getRedirectStrategy().sendRedirect(request, response,
"/");
}
```

7.5 Конфігурація безпеки для OAuth2

У SecurityConfig налаштовано маршрути та поведінку під час авторизації:

```
.oauth2Login(oauth2 -> oauth2
    .authorizationEndpoint(authorization ->
authorization.baseUri("/oauth2/authorize"))
    .redirectionEndpoint(redirection ->
redirection.baseUri("/oauth2/callback/*"))
    .userInfoEndpoint(userInfo ->
userInfo.userService(customOAuth2UserService))
    .successHandler(oAuth2AuthenticationSuccessHandler)
);
```

7.6 Отримання авторизованого користувача

Для отримання поточного авторизованого користувача з OAuth2 реалізовано окремий контролер:

```
@RestController
@RequestMapping("/api/oauth2")
public class OAuth2Controller {
    @GetMapping("/user")
    public Principal user(Principal principal) {
        return principal;
    }
}
```

ВИСНОВКИ

У результаті виконання курсової роботи було спроєктовано та реалізовано вебзастосунок — систему управління онлайн-курсами з підтримкою базової логіки навчання, автентифікації, реєстрації та захисту даних.

Досягнуто таких результатів:

– реалізовано архітектуру проєкту з чітким розділенням відповідальностей між контролерами, сервісами та репозиторіями;

– додано REST-контролери для роботи з курсами, модулями, студентами та прогресом;

– реалізовано реєстрацію та автентифікацію користувачів з використанням JWT та OAuth2;

Під час роботи над проєктом було здобуто практичний досвід у сфері розробки RESTful сервісів, інтеграції Spring Security, об'єктно-реляційного моделювання, управління користувачами та реалізації сучасних підходів до захисту даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spring Security Reference Documentation. Spring.io. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 25.04.2025).

2. JWT.io. JSON Web Tokens Introduction. URL: https://jwt.io/introduction (дата звернення: 27.05.2025).

3. Spring Boot. Getting Started. Spring.io. URL: https://spring.io/guides/gs/spring-boot/ (дата звернення: 24.04.2025).