

Fragment systemu wspomagającego funkcjonowanie studenta w uczelni

Autorzy:

Karol Pietrów

Olaf Bogusław

Mateusz Haik

Opis dziedziny problemu

Współczesne uczelnie wyższe zarządzają dużą liczbą studentów i przedmiotów. Proces zapisywania się studentów na przedmioty powinien być szybki, niezawodny i intuicyjny. Celem projektowanego modułu jest usprawnienie procesu przeglądania oferty dydaktycznej, zapisu studentów na przedmioty oraz dostępu do materiałów dydaktycznych dodawanych przez nauczycieli.

1. Koncepcja początkowa

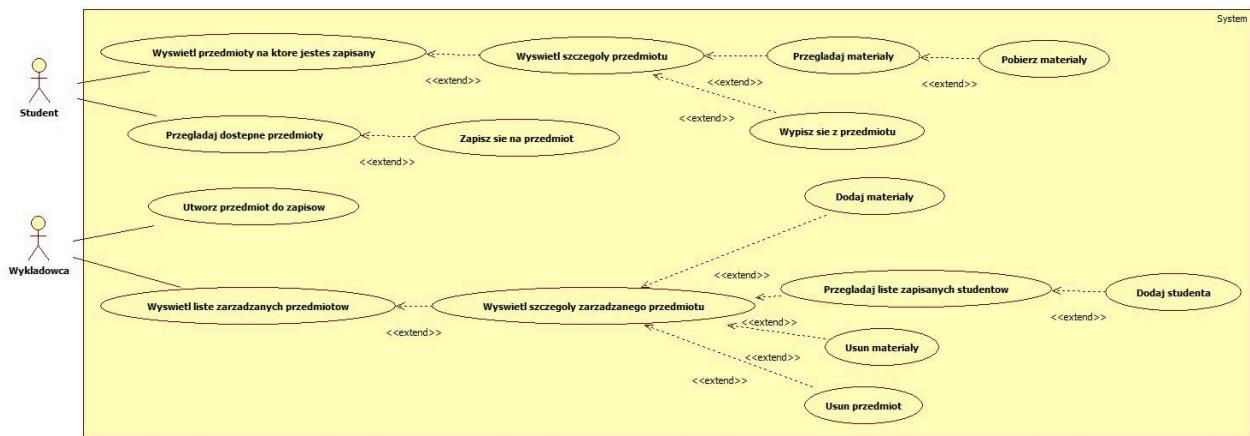
Wizją systemu wspomagającego funkcjonowanie studenta jest system koncentrujący się na procesie zapisów na przedmioty. Głównym celem modułu jest umożliwienie studentowi przeglądania oferty dydaktycznej i zapisu na wybrany przedmiot. System umożliwia wykładowcom przesyłanie plików do nauki dla studentów (materiałów), które ci mogą przeglądać i pobierać. Mogą też udostępniać linki, np. do wideokonferencji, do których mogą dołączać studenci.

2. Planowanie i analiza wymagań

2.1. Plan projektu

- Stworzenie modułu zapisów na przedmioty.
- Umożliwienie studentowi przeglądania dostępnych przedmiotów i zapisów na nie.
- Umożliwienie wykładowcy dodawania studentów, dodawania plików dla studentów
- Umożliwienie studentom wyświetlania i pobierania plików udostępnionych przez wykładowcę
- Dodanie obsługi linków
- Umożliwienie studentom wypisania się z przedmiotu
- Umożliwienie nauczycielom usuwania materiałów i przedmiotów

2.2. Analiza przypadków użycia



2.3. Aktorzy

- Student
- Wykładowca

2.4. Scenariusze

Tabela ze scenariuszami dostępna jest tutaj: [+ Przypadki użycia](#)

3. Projektowanie

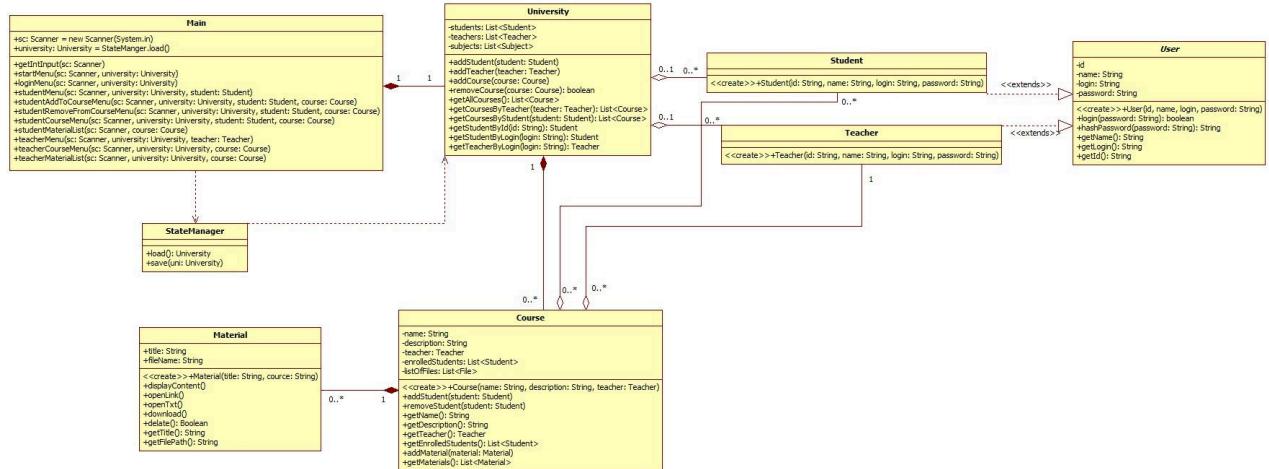
3.1. Klasa

Ustalono, iż będą potrzebne następujące klasy:

- **Main** - klasa główna, w której użytkownik prowadzi interakcję z programem. Tworzymy w niej obiekt University.
- **University** - będzie przechowywać listy studentów, nauczycieli i przedmiotów. Będzie posiadać metody dodawania studentów, nauczycieli i materiałów.
- **StateManager** - klasa umożliwiająca zapis stanu University do pliku, dzięki czemu listy studentów, nauczycieli i przedmiotów nie zostaną utracone po zamknięciu aplikacji. W klasie Main używamy metod statycznych klasy StateManager do ładowania i zapisywania danych.
- **User** - klasa abstrakcyjna. Będzie zawierać imię i nazwisko, ID, login oraz hasło, a także metodę logowania.
- **Student** oraz **Teacher** - będą rozszerzać klasę abstrakcyjną **User** (extends)
- **Course** - będzie trzymać informacje o przedmiocie (nazwa, opis, wykładowca, lista zapisanych studentów, lista materiałów). Będzie posiadać metody dodawania i usuwania studentów.

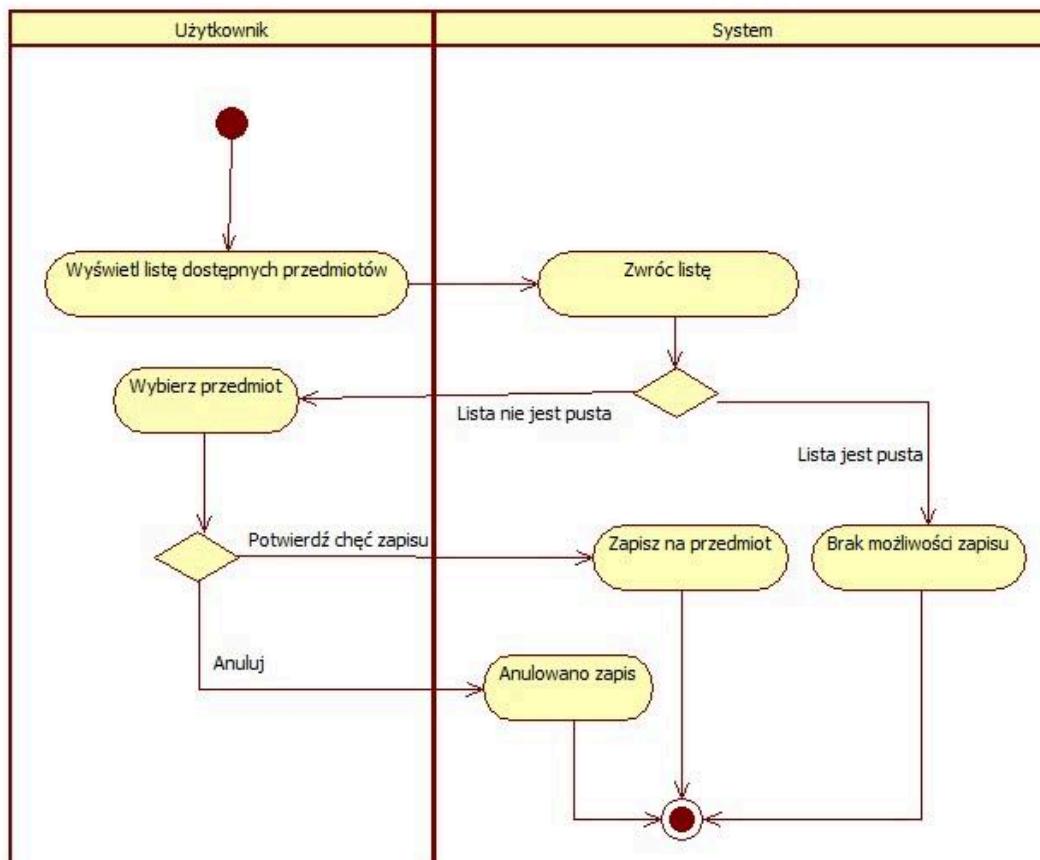
- Material** - będzie trzymać informacje o materiale (nazwa oraz ścieżka pliku lub link), a także metody do wyświetlania i pobierania materiałów przez użytkownika.

3.2. Diagram klas

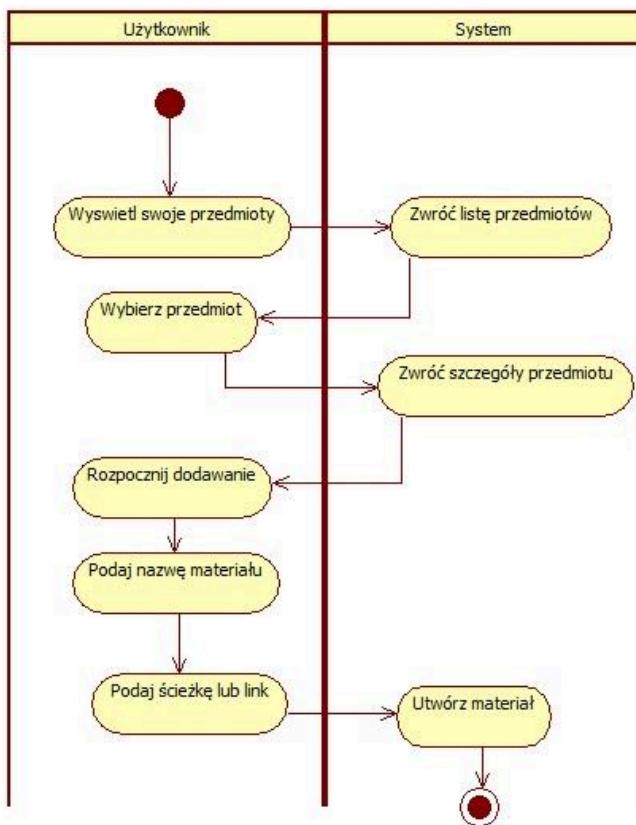


3.3. Diagramy aktywności

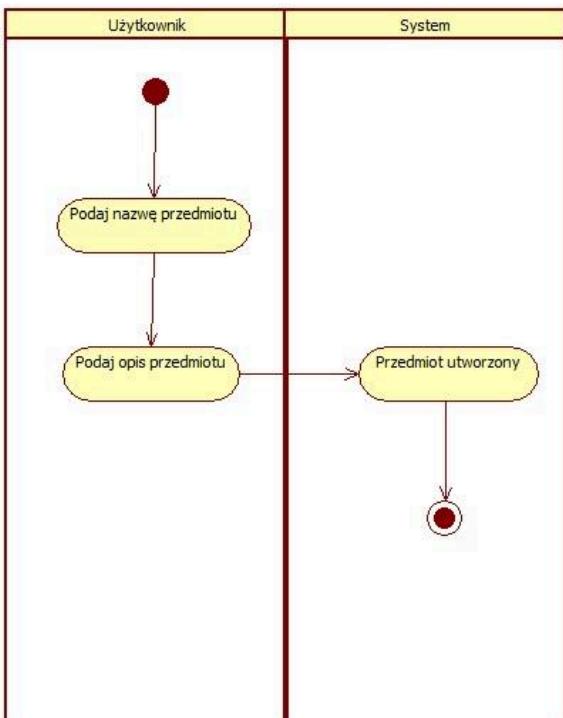
Zapis studenta na przedmiot



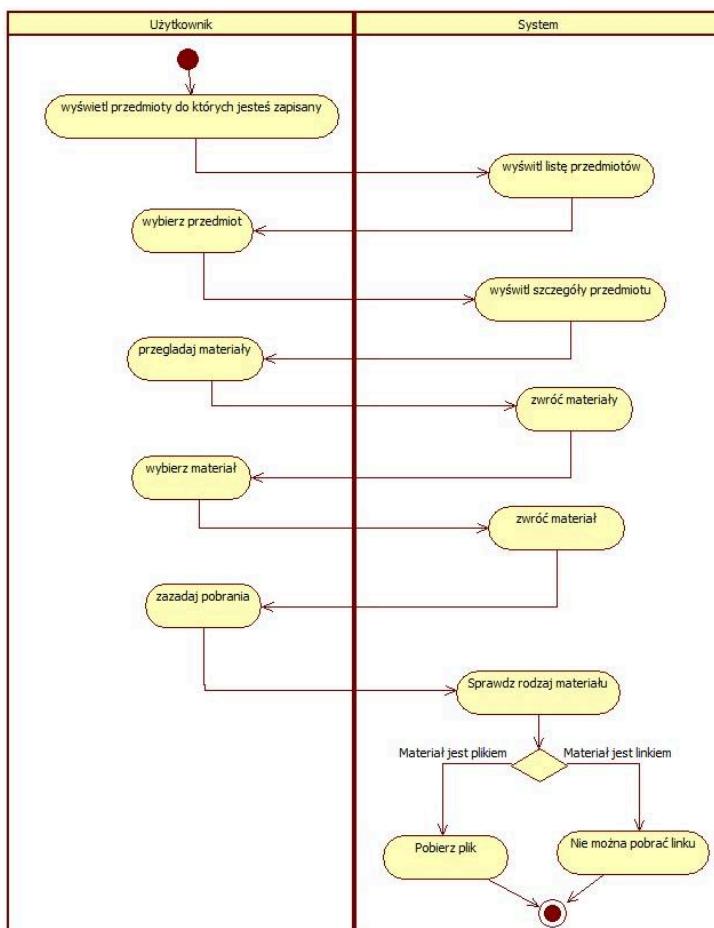
Dodawanie materiału



Utworzenie przedmiotu



Pobierz materiały



3.4. Diagramy sekwencji

Diagram sekwencji dodawania studenta do kursu

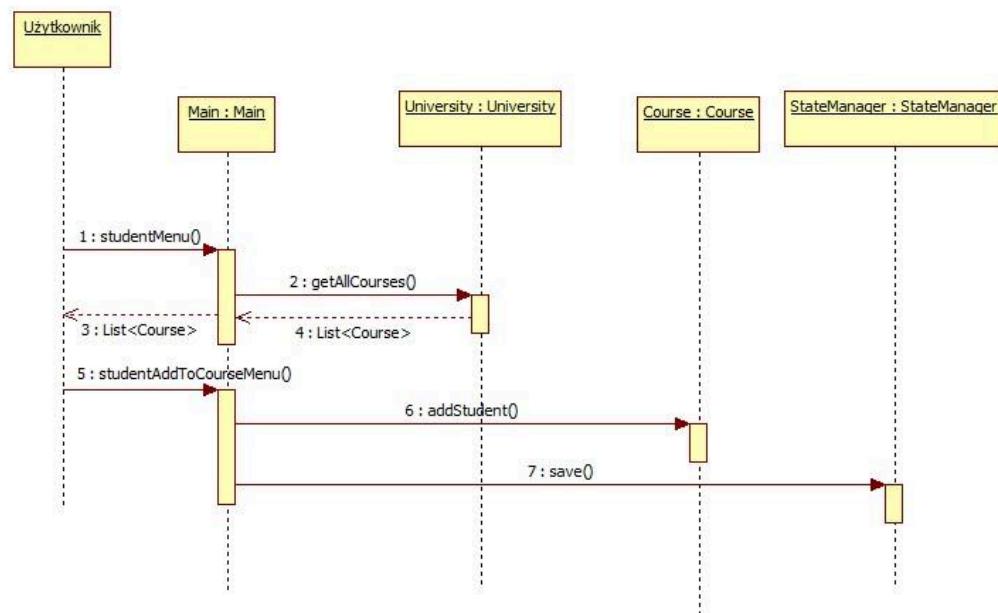
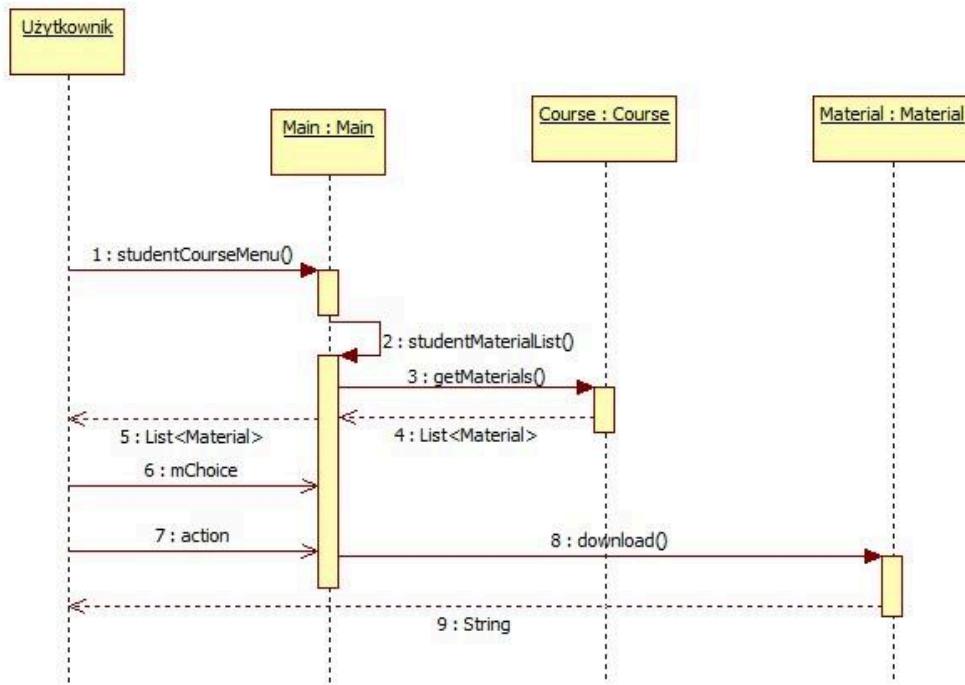
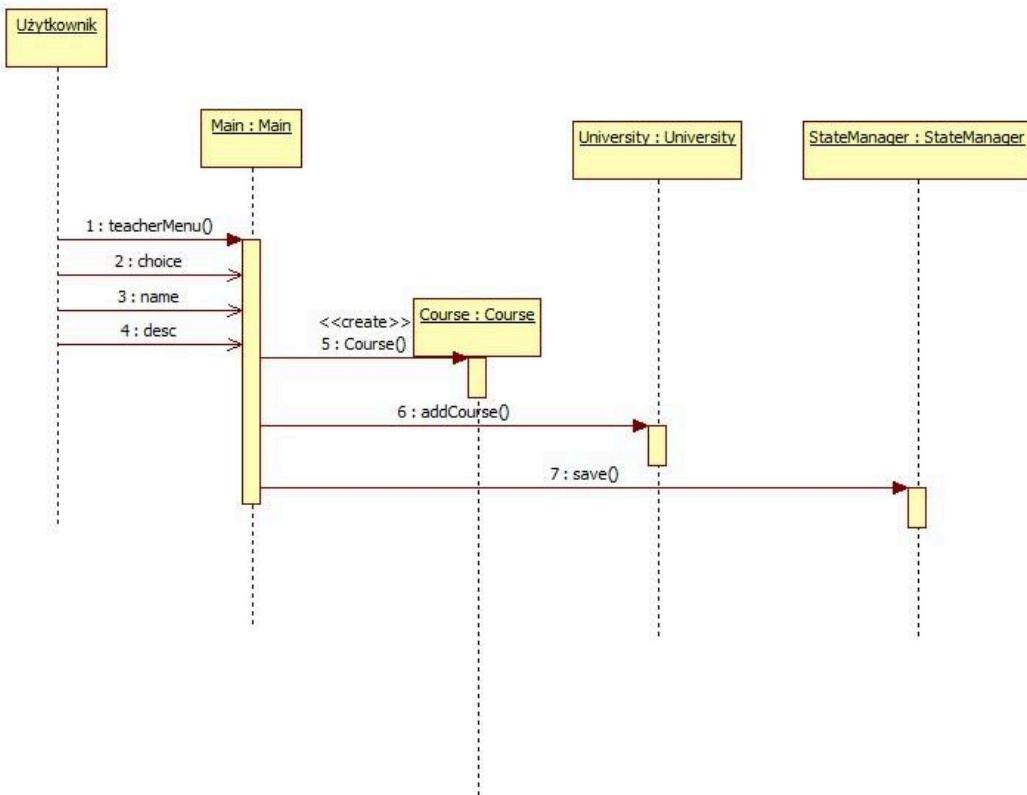


Diagram sekwencji pobierania materiału przez studenta



Dodanie sekwencji tworzenia przedmiotu przez wykładowcę



4. Implementacja

Projekt został napisany w języku Java. Repozytorium z kodem dostępny jest pod adresem <https://github.com/KarolPietrow/Java-UniversitySystem>

Klasa abstrakcyjna User.java

Obsługuje tworzenie i logowanie użytkowników.

```
5 @↓ public abstract class User { 2 inheritors  ↳ Karol Pietrów
6     private String id;
7     private String name;
8     private String login;
9     private String password;
10
11     public User(String id, String name, String login, String password) {
12         this.id = id;
13         this.name = name;
14         this.login = login;
15         this.password = hashPassword(password);
16     }
17
18     @Override  ↳ Karol Pietrów
19     public boolean equals(Object o) {
20         if (this == o) return true;
21         if (!(o instanceof User other)) return false;
22         return Objects.equals(this.id, other.id);
23     }
24
25     @Override  ↳ Karol Pietrów
26     public int hashCode() { return Objects.hash(id); }
27
28     boolean login(String password) {  ↳ Karol Pietrów
29         if (Objects.equals(this.password, hashPassword(password))) {
30             return true;
31         } else {
32             System.out.println("Niepoprawne dane logowania");
33             return false;
34         }
35     }
36
37 }
38
39
private String hashPassword(String password) { 2 usages  ↳ Karol Pietrów
try {
    MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");
    byte[] hashed = md.digest(password.getBytes());
    StringBuilder sb = new StringBuilder();
    for (byte b : hashed) {
        sb.append(String.format("%02x", b));
    }
    return sb.toString();
} catch (NoSuchAlgorithmException e) {
    throw new RuntimeException("SHA-256 not supported", e);
}
}

public String getName() { return name; }

public String getLogin() { return login; }

public String getId() { return id; }
```

Klasy Student.java i Teacher.java - klasy rozszerzające User

Pozwalają na oddzielne konta dla studentów i wykładowców.

```
public class Student extends User { 29 usages & Karol Pietrów
    public Student(String id, String name, String login, String password) {
        super(id, name, login, password);
    }
}

public class Teacher extends User { 23 usages & Karol Pietrów
    Teacher(String id, String name, String login, String password) {
        super(id, name, login, password);
    }
}
```

Klasa Course.java

Przechowuje informacje o przedmiocie, listę zapisanych studentów oraz listę materiałów.

```
public class Course { 32 usages & Karol Pietrów
    private String name; 2 usages
    private String description; 2 usages
    private Teacher teacher; 2 usages
    private List<Student> enrolledStudents = new ArrayList<>(); 4 usages
    private List<Material> materials = new ArrayList<>(); 2 usages

    public Course(String name, String description, Teacher teacher) {
        this.name = name;
        this.description = description;
        this.teacher = teacher;
    }

    public void addStudent(Student student) { 8 usages & Karol Pietrów
        if (!enrolledStudents.contains(student)) {
            enrolledStudents.add(student);
        }
    }

    public void removeStudent(Student student) { 4 usages & Karol Pietrów
        enrolledStudents.remove(student);
    }

    public String getName() { 8 usages & Karol Pietrów
        return name;
    }

    public String getDescription() { 5 usages & Karol Pietrów
        return description;
    }
```

```

public Teacher getTeacher() { 5 usages  & Karol
    return teacher;
}

public List<Student> getEnrolledStudents() {
    return enrolledStudents;
}

public void addMaterial(Material material) {
    materials.add(material);
}

public List<Material> getMaterials() { 7 usag
    return materials;
}

```

Klasa Material.java

Przechowuje informacje o materiale - nazwę, oraz ścieżkę pliku (lub adres URL). Posiada metody pozwalające [m.in.](#) na pobranie pliku od nauczyciela, wyświetlenie pliku TXT, otwarcie linka, i pobranie pliku przez studenta.

```

public class Material { 16 usages  & Karol Pietrów
    private String title;  4 usages
    private String filePath;  19 usages

    public Material(String title, String source) throws IOException { 5 usages  & Karol Pietrów
        this.title = title;
        source = source.trim();
        if ((source.startsWith("''") && source.endsWith("'')) ||
            (source.startsWith("\\"") && source.endsWith("\\"")))) {
            source = source.substring(1, source.length() - 1).trim();
        }
        if (source.startsWith("http://") || source.startsWith("https://")) { // Link
            this.filePath = source;
        } else { // Plik
            Path materialsDir = Path.of( first: "materials");
            if (!Files.exists(materialsDir)) {
                Files.createDirectories(materialsDir);
            }
            Path src = Path.of(source);
            if (!Files.exists(src)) {
                throw new IOException("Plik źródłowy nie istnieje: " + src.toAbsolutePath());
            }
            Path dest = materialsDir.resolve(src.getFileName());
            Files.copy(src, dest, StandardCopyOption.REPLACE_EXISTING);
            this.filePath = dest.toAbsolutePath().toString();
            System.out.println("Skopiowano plik do: " + this.filePath);
        }
    }

    public String getTitle() { return title; } 3 usages  & Karol Pietrów
    public String getFilePath() { return filePath; } 11 usages  & Karol Pietrów

```

```
    @Override & Karol Pietrów
    public String toString() {
        return title + " (" + filePath + ")";
    }

    public void displayContent() { 5 usages & Karol Pietrów
        if (filePath.startsWith("http://") || filePath.startsWith("https://")) {
            openLink();
        } else if (filePath.toLowerCase().endsWith(".txt")) {
            openTxt();
        } else {
            System.out.println("Podgląd dostępny tylko dla plików .txt oraz linków");
        }
    }

    private void openLink() { 1 usage & Karol Pietrów
        try {
            Desktop.getDesktop().browse(new URI(filePath));
            System.out.println("Otwieranie linku w przeglądarce: " + filePath);
        } catch (IOException | URISyntaxException e) {
            System.out.println("Błąd otwierania linku: " + e.getMessage());
        }
    }
}
```

```
public void openTxt() { 1 usage & Karol Pietrów
    File file = new File(filePath);
    if (!file.exists()) {
        System.out.println("Plik nie istnieje: " + filePath);
        return;
    }
    System.out.println("----- ZAWARTOŚĆ " + title + " -----");
    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.out.println("Błąd odczytu pliku: " + e.getMessage());
    }
    System.out.println("----- KONIEC PODGLĄDU -----");
}
```

```
public void download(String targetDir) { 5 usages & Karol Pietrów
    if (filePath.startsWith("http://") || filePath.startsWith("https://")) {
        System.out.println("Linków nie można pobrać");
        return;
    } else {
        String dir = targetDir.trim();
        if ((dir.startsWith("/") && dir.endsWith("/")) ||
            (dir.startsWith("\\")) && dir.endsWith("\\"))) {
            dir = dir.substring(1, dir.length() - 1).trim();
        }

        Path src = Path.of(filePath);
        if (!Files.exists(src)) {
            System.out.println("Plik źródłowy nie istnieje: " + filePath);
            return;
        }
        Path destDir = Path.of(dir);
        try {
            Files.createDirectories(destDir);
        } catch (IOException e) {
            System.out.println("Nie udało się utworzyć katalogu docelowego: " + destDir.toAbsolutePath());
            return;
        }

        Path dest = destDir.resolve(src.getFileName());
        try {
            Files.copy(src, dest, StandardCopyOption.REPLACE_EXISTING);
            System.out.println("Pobrano plik do: " + dest.toAbsolutePath());
        } catch (IOException e) {
            System.out.println("Błąd pobierania pliku: " + e.getMessage());
        }
    }
}
```

```

public boolean delete() { 5 usages  & Karol Pietrów
    if (filePath.startsWith("http://") || filePath.startsWith("https://")) {
        return true;
    } else {
        File f = new File(filePath);
        boolean fileDeleted = true;
        if (f.exists()) {
            fileDeleted = f.delete();
        }
        return fileDeleted;
    }
}

```

Klasa University.java

Przechowuje listy studentów, nauczycieli i przedmiotów, i posiada metody pozwalające na ich dodawanie. Umożliwia wyszukiwanie kursów na podstawie nauczyciela zarządzającego, oraz na znajdowanie studentów i nauczycieli na podstawie loginu lub ID.

```

public class University { 20 usages  & Karol Pietrów
    private List<Student> students = new ArrayList<>();  4 usages
    private List<Teacher> teachers = new ArrayList<>();  3 usages
    private List<Course> courses = new ArrayList<>();  6 usages

    public void addStudent(Student student) { 3 usages  & Karol Pietrów
        if (!students.contains(student)) {
            students.add(student);
        }
    }

    public void addTeacher(Teacher teacher) { 3 usages  & Karol Pietrów
        if (!teachers.contains(teacher)) {
            teachers.add(teacher);
        }
    }

    public void addCourse(Course course) { 4 usages  & Karol Pietrów
        if (!courses.contains(course)) {
            courses.add(course);
        }
    }

    public boolean removeCourse(Course course) { 2 usages  & Karol Pietrów
        for (Student s : new ArrayList<>(course.getEnrolledStudents())) {
            course.removeStudent(s);
        }
        return courses.remove(course);
    }

    public List<Course> getAllCourses() { return courses; }
}

```

```

public List<Course> getCoursesByTeacher(Teacher teacher) { 2 usages & Karol Pietrów
    return courses.stream()
        .filter( Course c -> c.getTeacher().equals(teacher))
        .toList();
}

public List<Course> getCoursesByStudent(Student student) { 4 usages & Karol Pietrów
    return courses.stream()
        .filter( Course c -> c.getEnrolledStudents().contains(student))
        .toList();
}

public Student getStudentById(String id) { 2 usages & Karol Pietrów
    for (Student student : students) {
        if (Objects.equals(student.getId(), id)) {
            return student;
        }
    }
    return null;
}

public Student getStudentByLogin(String login) { 5 usages & Karol Pietrów
    for (Student student : students) {
        if (Objects.equals(student.getLogin(), login)) {
            return student;
        }
    }
    return null;
}

```

```

public Teacher getTeacherByLogin(String login) { 4 usages
    for (Teacher teacher : teachers) {
        if (Objects.equals(teacher.getLogin(), login)) {
            return teacher;
        }
    }
    return null;
}

```

Klasa StateManager.java

Zapisuje stan obiektu University do pliku JSON, aby dane mogły zostać załadowane przy ponownym uruchomieniu programu.

```
public class StateManager { 14 usages  à Karol Pietrów
    private static final String DATA_FILE = "data.json"; 2 usages
    private static final Gson gson = new GsonBuilder() 2 usages
        .setPrettyPrinting()
        .create();

    public static University load() { 2 usages  à Karol Pietrów
        File f = new File(DATA_FILE);
        if (!f.exists()) {
            return new University();
        }
        try (Reader reader = new FileReader(f)) {
            return gson.fromJson(reader, University.class);
        } catch (IOException e) {
            System.err.println("Błąd podczas ładowania danych: " + e.getMessage());
            return new University();
        }
    }

    public static void save(University uni) { 12 usages  à Karol Pietrów
        try (Writer writer = new FileWriter(DATA_FILE)) {
            gson.toJson(uni, writer);
        } catch (IOException e) {
            System.err.println("Błąd podczas zapisu danych: " + e.getMessage());
        }
    }
}
```

Przykładowy plik data.json



```
{
    "students": [
        {
            "id": "144501",
            "name": "Andrzej Kowalski",
            "login": "144501@edu.pl",
            "password": "65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5"
        },
        {
            "id": "144502",
            "name": "Aleksander Wójcik",
            "login": "144502@edu.pl",
            "password": "42ad28944380f770cf17432c3494c07c32f680173b42c3562888f096e738ef7a"
        },
        {
            "id": "144503",
            "name": "Marta Zielińska",
            "login": "144503@edu.pl",
            "password": "42ad28944380f770cf17432c3494c07c32f680173b42c3562888f096e738ef7a"
        }
    ]
}
```

```

"teachers": [
    {
        "id": "15",
        "name": "Jan Nowak",
        "login": "jan.nowak@edu.pl",
        "password": "a15f8ae07675bfb96e084bfb4f52fb2c22091061aae86e0eb76a55f4e52dd74e"
    },
    {
        "id": "16",
        "name": "Stanisław Wiśniewski",
        "login": "stanislaw.wisniewski@edu.pl",
        "password": "b712e1cb406fb80c3a6ef21e78d25c2d719a1b90e24c58b4254293077ac002f"
    },
    {
        "id": "17",
        "name": "Ewelina Kowalczyk",
        "login": "ewelina.kowalczyk@edu.pl",
        "password": "a15f8ae07675bfb96e084bfb4f52fb2c22091061aae86e0eb76a55f4e52dd74e"
    }
],
"courses": [
    {
        "name": "Podstawy programowania 2024",
        "description": "Nauka podstaw programowania w języku C++",
        "teacher": {
            "id": "15",
            "name": "Jan Nowak",
            "login": "jan.nowak@edu.pl",
            "password": "a15f8ae07675bfb96e084bfb4f52fb2c22091061aae86e0eb76a55f4e52dd74e"
        },
        "enrolledStudents": [],
        "materials": []
    }
]

```

Klasa Main.java

Załadowanie obiektu University, ewentualne utworzenie nowych kont użytkowników, uruchomienie ekranu głównego:

```

public class Main {    ↳ Karol Pietrów *
    public static void main(String[] args) {    ↳ Karol Pietrów *
        Scanner sc = new Scanner(System.in);
        University university = StateManager.load();

        //    ↳ Utworzenie kont
        Teacher teacher1 = new Teacher("17", "Ewelina Kowalczyk");
        Teacher teacher2 = new Teacher("16", "Stanisław Wiśniewski");
        university.addTeacher(teacher1);
        university.addTeacher(teacher2);

        //    ↳ Dodawanie kursów
        university.addStudent(new Student("144503", "Martin"));
        university.addCourse(new Course("Programowanie obiektowe"));
        university.addCourse(new Course("ASyKo25", "Architektura"));

        StateManager.save(university);

        while (true) {
            startMenu(sc, university);
        }
    }
}

```

Funkcja obsługująca bezpieczne wczytanie wartości liczbowej od użytkownika:

```
private static int getIntInput(Scanner sc) { 17 usages  ♡ Karol Pietrów
    int input;
    while (true) {
        try {
            input = Integer.parseInt(sc.nextLine());
            return input;
        } catch (NumberFormatException e) {
            System.out.println("Nieprawidłowy format. Proszę wprowadzić liczbę.");
        }
    }
}
```

Ekran główny:

```
static void startMenu(Scanner sc, University university) { 1 usage  ♡ Karol Pietrów
    System.out.println("----- SYSTEM ZARZĄDZANIA STUDIAMI -----");
    System.out.println("1. Zaloguj się");
    System.out.println("2. Wyjdź");

    int choice = Integer.parseInt(sc.nextLine());
    int choice = getIntInput(sc);

    switch (choice) {
        case 1 -> loginMenu(sc, university);
        case 2 -> {
            System.out.println("Dziękujemy za skorzystanie z systemu. Do zobaczenia!");
            System.exit(status: 0);
        }
        default -> {
            System.out.println("Nieprawidłowy wybór");
            return;
        }
    }
}
```

Ekrany logowania:

```
static void loginMenu(Scanner sc, University university) { 1 usage  & Karol
    System.out.println("----- ZALOGUJ SIĘ -----");
    System.out.println("Podaj login:");
    String login = sc.nextLine();
    Student student = university.getStudentByLogin(login);
    Teacher teacher = university.getTeacherByLogin(login);
    boolean isAuthenticated;

    if (student != null) {
        System.out.println("Podaj hasło:");
        String password = sc.nextLine();
        isAuthenticated = student.login(password);
        if (isAuthenticated) {
            System.out.println("Zalogowano pomyślnie.");
            while (isAuthenticated) {
                isAuthenticated = studentMenu(sc, university, student);
            }
        } else {
            System.out.println("Nieprawidłowe hasło.");
        }
    } else if (teacher != null) {
        System.out.println("Podaj hasło: ");
        String password = sc.nextLine();
        isAuthenticated = teacher.login(password);
        if (isAuthenticated) {
            System.out.println("Zalogowano pomyślnie.");
            while (isAuthenticated) {
                isAuthenticated = teacherMenu(sc, university, teacher);
            }
        } else {
            System.out.println("Nieprawidłowe hasło.");
        }
    } else {
        System.out.println("Nieprawidłowy login.");
    }
}
```

Menu studenta:

```
static boolean studentMenu(Scanner sc, University university, Student student) { 1 usage & Karol Pietrów
    System.out.println("----- MENU STUDENTA -----");
    System.out.println("1. Przedmioty, na które jesteś zapisany");
    System.out.println("2. Przedmioty, na które możesz się zapisać");
    System.out.println("3. Wyloguj się");

    int choice = getIntInput(sc);
    switch (choice) {
        case 1 -> {
            List<Course> courses = university.getCoursesByStudent(student);
            if (!courses.isEmpty()) {
                System.out.println("TWOJE PRZEDMIOTY: ");
                for (int i = 0; i < courses.size(); i++) {
                    Course c = courses.get(i);
                    System.out.println(i+1 + ". " + c.getTeacher().getName() + " - " + c.getName() + " - " + c.getDescription());
                }
                System.out.println("-----");
                System.out.println("Wpisz numer przedmiotu, aby wyświetlić szczegóły, lub 0, aby wyjść.");
                int choice2 = getIntInput(sc);
                if (choice2 == 0) {
                    return true;
                } else if (choice2 <= courses.size()) {
                    studentCourseMenu(sc, university, student, courses.get(choice2-1));
                }
            } else {
                System.out.println("Nie jesteś zapisany na żaden przedmiot.");
            }
            return true;
        }

        case 2 -> {
            List<Course> all = university.getAllCourses();
            List<Course> notEnrolled = new ArrayList<>();
            for (Course c : all) {
                if (!university.getCoursesByStudent(student).contains(c)) {
                    notEnrolled.add(c);
                }
            }
            if (!notEnrolled.isEmpty()) {
                System.out.println("----- DOSTĘPNE PRZEDMIOTY -----");
                for (int i = 0; i < notEnrolled.size(); i++) {
                    Course c = notEnrolled.get(i);
                    System.out.println(i+1 + ". " + c.getTeacher().getName() + " - " + c.getName() + " - " + c.getDescription());
                }
                System.out.println("-----");
                System.out.println("Wpisz numer przedmiotu, na który chcesz się zapisać, lub 0, aby wyjść.");
                int choice2 = getIntInput(sc);
                if (choice2 == 0) {
                    return true;
                } else if (choice2 <= notEnrolled.size()) {
                    studentAddToCourseMenu(sc, university, student, notEnrolled.get(choice2-1));
                }
            } else {
                System.out.println("Brak dostępnych przedmiotów do zapisania.");
            }
            return true;
        }

        case 3 -> {
            System.out.println("Wylogowano.");
            return false;
        }

        default -> {
            System.out.println("Nieprawidłowy wybór");
            return true;
        }
    }
}
```

Menu dodawania studenta do kursu

```
static void studentAddToCourseMenu(Scanner sc, University university, Student student, Course course) {  
    System.out.println("----- UWAGA -----");  
    System.out.println("Czy na pewno chcesz zapisać się na przedmiot " + course.getName() + "?");  
    System.out.println("Y - tak");  
    System.out.println("N - nie");  
    String choice = sc.nextLine().trim();  
    if (choice.equalsIgnoreCase("Y")) {  
        course.addStudent(student);  
        StateManager.save(university);  
    }  
}
```

Menu usuwania studenta z kursu

```
static void studentRemoveFromCourseMenu(Scanner sc, University university, Student student, Course course) { 1 usage 2 Karol Pietrów  
    System.out.println("----- UWAGA -----");  
    System.out.println("Czy na pewno chcesz wypisać się z przedmiotu " + course.getName() + "? Nie będziesz miał dostępu do materiałów do czasu ponownego zapisu.");  
    System.out.println("Y - tak");  
    System.out.println("N - nie");  
    String choice = sc.nextLine().trim();  
    if (choice.equalsIgnoreCase("Y")) {  
        course.removeStudent(student);  
        StateManager.save(university);  
    }  
    System.out.println("Wypisano z przedmiotu.");  
}
```

Menu kursów (studenta)

```
static void studentCourseMenu(Scanner sc, University university, Student student, Course course) { 1 usag  
    System.out.println("----- SZCZEGÓŁY PRZEDMIOTU -----");  
    System.out.println("Nazwa: " + course.getName());  
    System.out.println("Opis: " + course.getDescription());  
    System.out.println("Wykładowca: " + course.getTeacher().getName());  
    System.out.println("Liczba zapisanych studentów: " + course.getEnrolledStudents().toArray().length);  
    System.out.println("Lista dodanych materiałów: " + course.getMaterials().toArray().length);  
    System.out.println("-----");  
    System.out.println("1. Wyświetl listę materiałów");  
    System.out.println("2. Wypisz się z przedmiotu");  
    System.out.println("3. Powrót");  
  
    int choice = getIntInput(sc);  
    switch (choice) {  
        case 1 -> {  
            studentMaterialList(sc, course);  
        }  
        case 2 -> {  
            studentRemoveFromCourseMenu(sc, university, student, course);  
            StateManager.save(university);  
        }  
        case 3 -> {  
            break;  
        }  
        default -> {  
            break;  
        }  
    }  
}
```

Menu materiałów kursu (studenta)

```
static void studentMaterialList(Scanner sc, Course course) { 1 usage ☺ Karol Pietrów
    System.out.println("Materiały:");
    List<Material> mats = course.getMaterials();
    if (mats.isEmpty()) {
        System.out.println("Brak materiałów.");
    } else {
        for (int i = 0; i < mats.size(); i++) {
            System.out.printf("%d. %s%n", i+1, mats.get(i));
        }
    }
    System.out.print("Wybierz numer materiału, by go wyświetlić/pobrać, lub 0, by wrócić: ");
    int mChoice = getIntInput(sc);
    if (mChoice > 0 && mChoice <= mats.size()) {
        Material m = mats.get(mChoice - 1);
        if (m.getFilePath().startsWith("http://") || m.getFilePath().startsWith("https://")) {
            System.out.println("1. Otwórz link");
            System.out.println("0. Powrót");
            int action = getIntInput(sc);
            if (action == 1) {
                m.displayContent();
            }
        } else if (m.getFilePath().toLowerCase().endsWith(".txt")) {
            System.out.println("1. Podgląd");
            System.out.println("2. Pobierz plik");
            System.out.println("0. Powrót");
            int action = getIntInput(sc);
            switch (action) {
                case 1 -> m.displayContent();
                case 2 -> {
                    System.out.print("Podaj ścieżkę katalogu docelowego: ");
                    String dir = sc.nextLine().trim();
                    m.download(dir);
                }
                default -> {
                }
            }
        }
    }
}
```

```
} else {
    System.out.println("1. Pobierz plik");
    System.out.println("0. Powrót");
    int action = getIntInput(sc);
    if (action == 1) {
        System.out.print("Podaj ścieżkę katalogu docelowego: ");
        String dir = sc.nextLine().trim();
        m.download(dir);
    }
}
```

Menu nauczyciela

```
static boolean teacherMenu(Scanner sc, University university, Teacher teacher) { 1 usage & Karol Pietrów
    System.out.println("1. Utwórz nowy przedmiot");
    System.out.println("2. Wyświetl swoje przedmioty");
    System.out.println("3. Wyloguj się");

    int choice = getIntInput(sc);
    switch (choice) {
        case 1 -> {
            System.out.println("Podaj nazwę przedmiotu:");
            String name = sc.nextLine();
            System.out.println("Podaj krótki opis przedmiotu:");
            String desc = sc.nextLine();
            university.addCourse(new Course(name, desc, teacher));
            System.out.println("Przedmiot utworzony.");
            StateManager.save(university);
            return true;
        }
        case 2 -> {
            List<Course> courses = university.getCoursesByTeacher(teacher);
            if (!courses.isEmpty()) {
                System.out.println("TWOJE PRZEDMIOTY: ");
                for (int i = 0; i < courses.size(); i++) {
                    Course c = courses.get(i);
                    System.out.println(i+1 + ". " + c.getName() + " - " + c.getDescription());
                }
                System.out.println("-----");
                System.out.println("Wpisz numer przedmiotu, aby wyświetlić szczegóły, lub 0, aby wyjść.");
                int choice2 = getIntInput(sc);
                if (choice2 == 0) {
                    return true;
                } else if (choice2 <= courses.size()) {
                    teacherCourseMenu(sc, university, courses.get(choice2-1));
                }
            } else {
                System.out.println("Nie zarządzasz żadnymi przedmiotami.");
            }
            return true;
        }
        case 3 -> {
            System.out.println("Wylogowano.");
            return false;
        }
        default -> {
            System.out.println("Nieprawidłowy wybór");
            return true;
        }
    }
}
```

Menu kursów (nauczyciela):

```
static void teacherCourseMenu(Scanner sc, University university, Course course) { 1 usage  ↳ Karol Pietrów
    System.out.println("----- SZCZEGÓŁY PRZEDMIOTU -----");
    System.out.println("Nazwa: " + course.getName());
    System.out.println("Opis: " + course.getDescription());
    System.out.println("Wykładowca: " + course.getTeacher().getName());
    System.out.println("Liczba zapisanych studentów: " + course.getEnrolledStudents().toArray().length);
    System.out.println("Lista dodanych materiałów: " + course.getMaterials().toArray().length);
    System.out.println("-----");
    System.out.println("1. Wyświetl liste zapisanych studentów");
    System.out.println("2. Dodaj studenta");
    System.out.println("3. Wyświetl liste materiałów");
    System.out.println("4. Dodaj materiał");
    System.out.println("5. Usuń przedmiot");
    System.out.println("0. Powrót");

    int choice = getIntInput(sc);
    switch (choice) {
        case 1 -> {
            List<Student> students = course.getEnrolledStudents();
            if (!students.isEmpty()) {
                for (int i = 0; i < students.toArray().length; i++) {
                    System.out.println(i + 1 + ". " + students.get(i).getName());
                }
            } else {
                System.out.println("Brak studentów zapisanych na przedmiot");
            }
        }
        case 2 -> {
            System.out.println("Podaj ID studenta, którego chcesz dodać, lub 0 aby anulować:");
            int choice2 = getIntInput(sc);
            if (choice2 != 0) {
                Student student = university.getStudentById(String.valueOf(choice2));
                if (student != null) {
                    course.addStudent(student);
                    System.out.println(student.getName() + " został dodany do przedmiotu");
                    StateManager.save(university);
                }
            }
        }
    }
}
```

```

        } else {
            System.out.println("Niepoprawne ID studenta.");
        }
    }
}

case 3 -> {
    teacherMaterialList(sc, university, course);
}

case 4 -> {
    System.out.print("Podaj tytuł materiału: ");
    String title = sc.nextLine().trim();
    System.out.print("Podaj pełną ścieżkę do pliku, lub link (https://):");
    String source = sc.nextLine().trim();
    try {
        Material m = new Material(title, source);
        course.addMaterial(m);
        System.out.println("Materiał dodany: " + m.getFilePath());
        StateManager.save(university);
    } catch (IOException e) {
        System.out.println("Nie udało się dodać materiału: " + e.getMessage());
    }
    break;
}

case 5 -> {
    System.out.println("----- UWAGA !!! -----");
    System.out.print("Czy na pewno chcesz USUNĄĆ cały przedmiot \""
        + course.getName() + "\"? Tej czynności NIE MOŻNA cofnąć! (Y/N): ");
    String conf = sc.nextLine().trim();
    if (conf.equalsIgnoreCase(anotherString: "Y")) {
        boolean removed = university.removeCourse(course);
        if (removed) {
            System.out.println("Przedmiot usunięty pomyślnie.");
            StateManager.save(university);
        } else {
            System.out.println("Błąd: nie udało się usunąć przedmiotu.");
        }
        return;
    } else {
        System.out.println("Anulowano usuwanie przedmiotu.");
    }
}

case 0 -> {
    return;
}

default -> {
    System.out.println("Nieprawidłowy wybór.");
}
}
}

```

Menu materiałów kursu (nauczyciela):

```
static void teacherMaterialList(Scanner sc, University university, Course course) { 1 usage  ↗ Karol Pietrów
    System.out.println("Materiały:");
    List<Material> mats = course.getMaterials();
    if (mats.isEmpty()) {
        System.out.println("Brak materiałów.");
    } else {
        for (int i = 0; i < mats.size(); i++) {
            System.out.printf("%d. %s%n", i+1, mats.get(i));
        }
        System.out.print("Wybierz numer materiału, by go wyświetlić/pobrać, lub 0, by wrócić: ");
        int mChoice = getIntInput(sc);
        if (mChoice > 0 && mChoice <= mats.size()) {
            Material m = mats.get(mChoice - 1);
            if (m.getFilePath().startsWith("http://") || m.getFilePath().startsWith("https://")) {
                System.out.println("1. Otwórz link");
                System.out.println("2. Usuń linka");
                System.out.println("0. Powrót");
                int action = getIntInput(sc);
                switch (action) {
                    case 1 -> m.displayContent();
                    case 2 -> {
                        System.out.print("Czy na pewno chcesz usunąć link \"" + m.getTitle() + "? Tej czynności NIE MOŻNA cofnąć! (Y/N):");
                        String conf = sc.nextLine().trim();
                        if (conf.equalsIgnoreCase("Y")) {
                            if (m.delete()) {
                                course.getMaterials().remove(m);
                                System.out.println("Link usunięty pomyślnie.");
                                StateManager.save(university);
                            } else {
                                System.out.println("Nie udało się usunąć linka: " + m.getFilePath());
                            }
                        } else {
                            System.out.println("Anulowano usuwanie.");
                        }
                    }
                }
            } else {
                default -> {
                    System.out.println("Wybierz numer materiału, by go wyświetlić/pobrać, lub 0, by wrócić: ");
                }
            }
        }
    }
}
```

```
        }
    }

} else if (m.getFilePath().toLowerCase().endsWith(".txt")) {
    System.out.println("1. Podgląd");
    System.out.println("2. Pobierz plik");
    System.out.println("3. Usuń plik");
    System.out.println("0. Powrót");
    int action = getIntInput(sc);
    switch (action) {
        case 1 -> m.displayContent();
        case 2 -> {
            System.out.print("Podaj katalog docelowy (pełna ścieżka): ");
            String dir = sc.nextLine().trim();
            m.download(dir);
        }
        case 3 -> {
            System.out.print("Czy na pewno chcesz usunąć materiał \""
                + m.getTitle() + "\"? Tej czynności NIE MOŻNA cofnąć! (Y/N):");
            String conf = sc.nextLine().trim();
            if (conf.equalsIgnoreCase("Y")) {
                if (m.delete()) {
                    course.getMaterials().remove(m);
                    System.out.println("Materiał usunięty pomyślnie.");
                    StateManager.save(university);
                } else {
                    System.out.println("Nie udało się usunąć pliku: " + m.getFilePath());
                }
            } else {
                System.out.println("Anulowano usuwanie.");
            }
        }
        default -> {
        }
    }
}

} else {
```

5. Testowanie

Zostały napisane testy w JUnit w celu weryfikacji poprawnego działania programu.

CourseTest.java

```
1 import org.junit.jupiter.api.BeforeEach;
2 import org.junit.jupiter.api.Test;
3
4 import static org.junit.jupiter.api.Assertions.*;
5
6 class CourseTest { new *
7     private Teacher teacher; 2 usages
8     private Student s1; 6 usages
9     private Course course; 8 usages
10
11    @BeforeEach new *
12    void init() {
13        teacher = new Teacher( id: "5", name: "Marcin Kwiatkowski", login: "marcin", password: "x");
14        course = new Course( name: "Matematyka", description: "Matematyka dyskertna 2025", teacher);
15        s1 = new Student( id: "11", name: "Ewa Mazurek", login: "ewa", password: "e");
16    }
17
18    @Test new *
19    void testAddAndRemoveStudent() {
20        assertTrue(course.getEnrolledStudents().isEmpty());
21
22        course.addStudent(s1);
23        assertTrue(course.getEnrolledStudents().contains(s1));
24
25        course.addStudent(s1);
26        assertEquals( expected: 1, course.getEnrolledStudents().size());
27
28        course.removeStudent(s1);
29        assertFalse(course.getEnrolledStudents().contains(s1));
30    }
31 }
```

MaterialTest.java

```
10 class MaterialTest { new *
11     @TempDir Path tempDir; no usages
12
13
14     @Test new *
15     void testLinkMaterial() throws IOException {
16         Material m = new Material( title: "Zoom", source: "https://zoom.us/j/123");
17         assertTrue(m.getFilePath().startsWith("https://"));
18         assertDoesNotThrow(m::delete);
19     }
20
21
22     @Test new *
23     void testTxtMaterialCopyAndDisplay(@TempDir Path temp) throws Exception {
24         Path src = temp.resolve( other: "test.txt");
25         Files.writeString(src, csq: "Hello\nWorld");
26         Material m = new Material( title: "Test", src.toString());
27         Path dest = Path.of(System.getProperty("user.dir"), ...more: "materials", "test.txt");
28         assertTrue(Files.exists(dest));
29         assertDoesNotThrow(m::displayContent);
30         assertTrue(m.delete());
31         assertFalse(Files.exists(dest));
32     }
33
34     @Test new *
35     void testDownload(@TempDir Path temp) throws IOException {
36         Path matDir = Path.of(System.getProperty("user.dir"), ...more: "materials");
37         Files.createDirectories(matDir);
38         Path src = matDir.resolve( other: "dl.txt");
39         Files.writeString(src, csq: "Data");
40         Material m = new Material( title: "DL", src.toString());
41         m.download(temp.toString());
42         Path downloaded = temp.resolve( other: "dl.txt");
43         assertTrue(Files.exists(downloaded));
44         assertEquals( expected: "Data", Files.readString(downloaded));
45     }
46 }
```

StateManagerTest.java

```
10 >> class StateManagerTest { new *
11     private static final String TEST_FILE = "data.json";  2 usages
12
13     @BeforeEach new *
14     void cleanUp() throws Exception {
15         Files.deleteIfExists(Path.of(TEST_FILE));
16     }
17
18     @Test new *
19     void testSaveAndLoad() {
20         University uni = new University();
21         Student s = new Student(id: "9", name: "Ivy", login: "ivy", password: "i");
22         uni.addStudent(s);
23         StateManager.save(uni);
24
25         assertTrue(new File(TEST_FILE).exists());
26         University loaded = StateManager.load();
27         assertNotNull(loaded.getStudentByLogin("ivy"));
28         assertEquals(expected: "Ivy", loaded.getStudentByLogin("ivy").getName());
29     }
30 }
```

StudentTest.java

```
public class StudentTest { new *
    private Student student;  8 usages
    private Teacher teacher;  2 usages
    private Course course;  8 usages

    @BeforeEach new *
    void setUp() {
        student = new Student(id: "1", name: "Alicja", login: "alicja", password: "pass");
        teacher = new Teacher(id: "2", name: "Marcin", login: "marcin", password: "secret");
        course = new Course(name: "Matematyka dyskretna", description: "Matematyka dyskretna 2025", teacher);
    }

    @Test new *
    void testLoginSuccess() {
        assertTrue(student.login(password: "pass"));
    }

    @Test new *
    void testLoginFailure() {
        assertFalse(student.login(password: "wrong"));
    }

    @Test new *
    void testEnrollAndRemoveCourse() {
        assertTrue(course.getEnrolledStudents().isEmpty());

        course.addStudent(student);
        assertTrue(course.getEnrolledStudents().contains(student));

        course.addStudent(student);
        assertEquals(expected: 1, course.getEnrolledStudents().size());

        course.removeStudent(student);
        assertFalse(course.getEnrolledStudents().contains(student));
    }
```

TeacherTest.java

```
6  class TeacherTest { new *
7      private Teacher teacher; 3 usages
8
9      @BeforeEach new *
10     void setUp() {
11         teacher = new Teacher(id: "10", name: "Marcin Kwiatkowski", login: "marcin", password: "pwd");
12     }
13
14     @Test new *
15     void testLoginSuccess() {
16         assertTrue(teacher.login(password: "pwd"));
17     }
18
19     @Test new *
20     void testLoginFailure() {
21         assertFalse(teacher.login(password: "bad"));
22     }
23 }
```

UniversityTest.java

```
8  class UniversityTest { Karol Pietrów *
9      private University uni; 20 usages
10     private Student st; 9 usages
11     private Teacher th; 6 usages
12     private Course c1, c2; 10 usages
13
14     @BeforeEach new *
15     void setup() {
16         uni = new University();
17         st = new Student(id: "100", name: "Damian Kowalski", login: "damian", password: "d");
18         th = new Teacher(id: "200", name: "Jan Nowak", login: "jan", password: "j");
19         uni.addStudent(st);
20         uni.addTeacher(th);
21         c1 = new Course(name: "Chemia", description: "Chemia 2025", th);
22         c2 = new Course(name: "Biologia", description: "Biologia 2025", th);
23         uni.addCourse(c1);
24         uni.addCourse(c2);
25     }
26
27     @Test new *
28     void addStudent() {
29         Student student = new Student(id: "1", name: "Test", login: "test", password: "test");
30         uni.addStudent(student);
31         assertNotNull((uni.getStudentById(student.getId())));
32     }
33
34     @Test new *
35     void addTeacher() {
36         Teacher teacher = new Teacher(id: "1", name: "Test", login: "test", password: "test");
37         uni.addTeacher(teacher);
38         assertNotNull((uni.getTeacherByLogin(teacher.getLogin())));
39     }
```

```
41 @Test new *
42 void addCourse() {
43     Course course = new Course( name: "course", description: "desc", new Teacher( id: "test", name: "test", login: "test", password: "test"));
44     uni.addCourse(course);
45     assertTrue(uni.getAllCourses().contains(course));
46 }
47
48 @Test new *
49 void testGetStudentByLogin() {
50     assertEquals(st, uni.getStudentByLogin("damian"));
51     assertNull(uni.getStudentByLogin("nope"));
52 }
53
54 @Test new *
55 void testGetTeacherByLogin() {
56     assertEquals(th, uni.getTeacherByLogin("jan"));
57     assertNull(uni.getTeacherByLogin("x"));
58 }
59
60 @Test new *
61 void testGetCoursesByTeacher() {
62     List<Course> list = uni.getCoursesByTeacher(th);
63     assertEquals( expected: 2, list.size());
64     assertTrue(list.containsAll(List.of(c1, c2)));
65 }
66
67 @Test new *
68 void testGetCoursesByStudent() {
69     assertFalse(uni.getCoursesByStudent(st).isEmpty());
70     c1.addStudent(st);
71     List<Course> list = uni.getCoursesByStudent(st);
72     assertEquals( expected: 1, list.size());
73     assertTrue(list.contains(c1));
74 }
```

```
@Test new *
void testRemoveCourseCleansStudents() {
    c1.addStudent(st);
    assertTrue(c1.getEnrolledStudents().contains(st));

    boolean removed = uni.removeCourse(c1);
    assertTrue(removed);
    assertFalse(uni.getAllCourses().contains(c1));
    assertFalse(c1.getEnrolledStudents().contains(st));
}
```