

Sprawozdanie

Bazy Danych 2 - Projekt

Paweł Gacek i Aleksander Pytel, 24.05.2022

1. Temat projektu

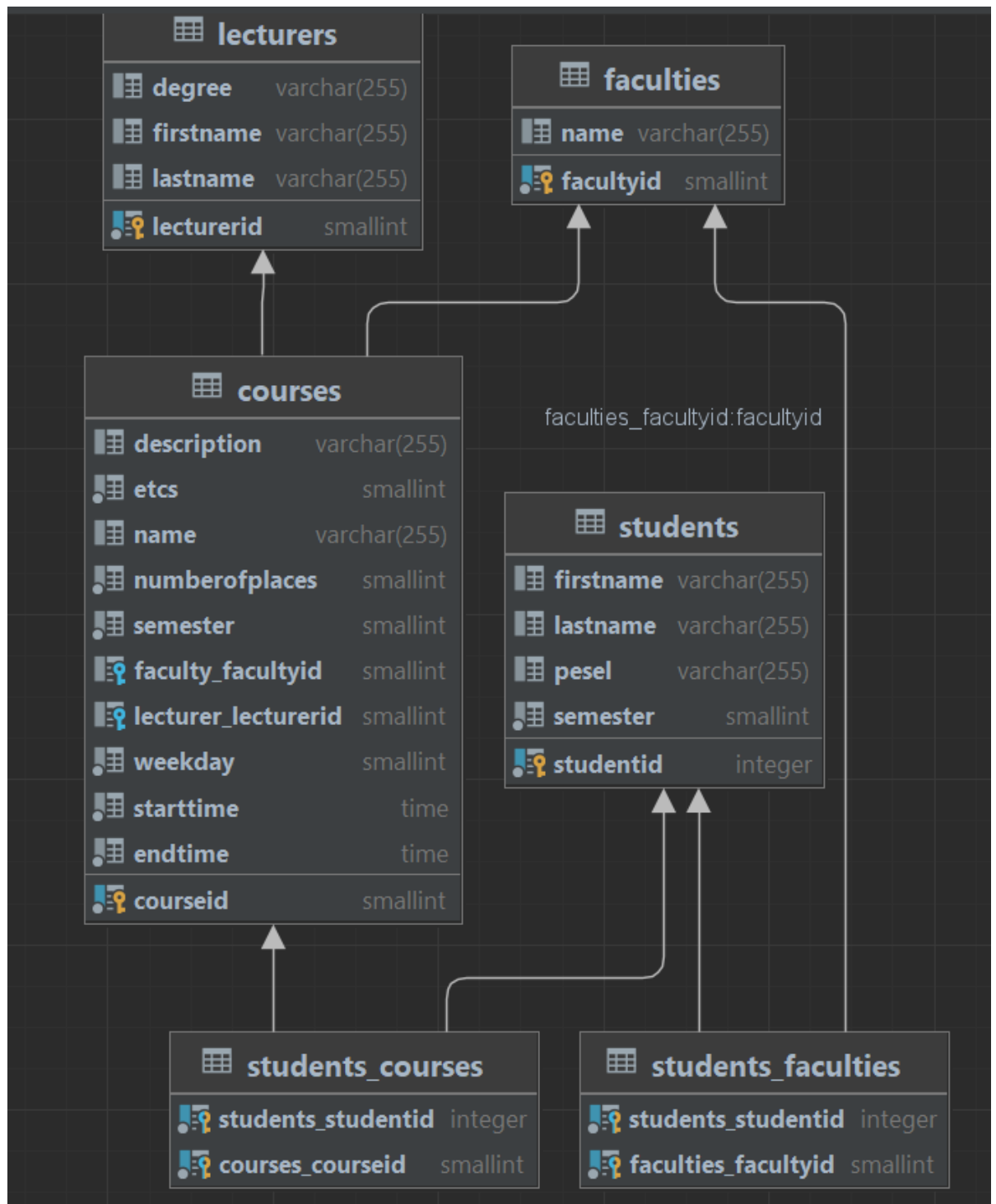
Aplikacja okienkowa umożliwiająca studentom zapisy na zajęcia. (Technologie: PostgreSQL i Hibernate). Student, po zalogowaniu się do aplikacji będzie mógł zapisać się na zajęcia przypisane do swojego wydziału i semestru, a także zobaczyć zajęcia, na które już się zapisał. Aplikacja uniemożliwi zapis na zajęcia, których termin będzie kolidował z innymi zajęciami, na zajęcia, które nie mają wolnych miejsc oraz w przypadku gdy student przekroczy maksymalną ilość punktów ECTS. Po kliknięciu na konkretne zajęcia, studentowi ukażą się szczegółowe informacje.

2. Schemat bazy i opis tabel

a) Baza składa się z 6 tabel:

- **Students** – zawiera podstawowe informacje o studencie, kluczem głównym jest indeks
- **Lecturers** – zawiera informacje o wykładowcach prowadzących zajęcia. Informacje te pokazywane będą po wejściu przez studenta w dany kurs.
- **Courses** – zawiera informacje o zajęciach takie jak nazwa, ilość wolnych miejsc, liczba punktów ECTS, opis, termin
- **Faculties** – zawiera nazwy wydziałów
- **Students_Courses** – zawiera informację o tym, jaki student jest zapisany na jakie zajęcia
- **Students_Faculties** – zawiera informację na jakich wydziałach studiuje student (student może studiować na wielu wydziałach)

b) Schemat



3. Funkcje, procedury, triggerery

a) Funkcje

- `contains_student_id (student_id)` - zwraca informację czy istnieje student o danym ID
- `get_courses (student_id)` - zwraca zbiór kursów, które dotyczą danego studenta, są z odpowiedniego wydziału oraz semestru.
- `is_student_enrolled(student_id, course_id)` - zwraca informację czy student jest zapisany na podane zajęcia
- `get_used_ects(student_id)` - zwraca liczbę wykorzystanych punktów ECTS danego studenta; sumę wartości z zajęć na które jest zapisany
- `collision_exists (student_id, course_id)` - zwraca informację czy student jest zapisany na zajęcia, które odbywają się w tym samym czasie co podane
- `no_available_places(course_id)` - zwraca liczbę wolnych miejsc na podanych zajęciach.

b) Triggerery

- `enroll_trigger_function` - przy zapisywaniu studenta na zajęcia sprawdź czy istnieją kolizje, czy są wolne miejsca oraz czy student ma jeszcze punkty ECTS to wykorzystania.

Kod

a) Funkcje

- contains_student_id (student_id)

```
create or replace function contains_student_id(student_id bigint)
returns bool
language plpgsql
as
$$
declare
    student students%ROWTYPE;
begin
    select * into student from students where studentid = student_id limit 1;
    if student is null then
        return false;
    end if;
    return true;
end
$$;
```

- get_courses (student_id)

```
create or replace function get_courses(student_id bigint)
returns setof public.courses as $$
declare
    ids smallint[];
    student students%rowtype;
begin
    select * into student from students as s where student_id = s.studentid;
    if student is null then
        raise exception 'Student not found';
    end if;
    select ARRAY(select sf.faculties_facultyid from students_faculties as sf where sf.students_studentid = student_id) into ids;
    return query
        select * from courses as c
        where c.semester = student.semester and c.faculty_facultyid = any(ids);
end
$$ language plpgsql;
```

- is_student_enrolled(student_id, course_id)

```
create or replace function is_student_enrolled(student_id integer, course_id integer)
returns bool as
$$
declare
    enrollment students_courses%ROWTYPE;
begin
    select * into enrollment from students_courses as sc
    where sc.students_studentid = student_id and sc.courses_courseid = course_id;
    if enrollment is null then
        return false;
    end if;
    return true;
end;
$$ language plpgsql;
```

- `get_used_ects(student_id)`

```
create or replace function get_used_ects(student_id bigint)
returns Integer as
$$
    declare
        used_ects smallint;
    begin
        if not contains_student_id( student_id: student_id) then
            raise exception 'Student not found';
        end if;
        select into used_ects sum(c.ects) from courses as c
            inner join students_courses sc on c.courseid = sc.courses_courseid
        where sc.students_studentid = student_id;
        if used_ects is null then
            return 0;
        end if;
        return used_ects;
    end;
$$ language plpgsql;
```

- `collision_exists (student_id, course_id)`

```
create function collision_exists(student_id integer, course_id integer) returns boolean
language plpgsql
as
$$
declare
    chosen_course courses%rowtype;
    no_collisions smallint;
begin
    if not contains_student_id( student_id: student_id) then
        raise exception 'student not found';
    end if;
    select into chosen_course * from courses as c where c.courseid = course_id limit 1;
    if chosen_course is null then
        raise exception 'course not found';
    end if;

    select into no_collisions count(c.*) from courses as c
        inner join students_courses sc on c.courseid = sc.courses_courseid
    where sc.students_studentid = student_id and c.weekday = chosen_course.weekday
        and not (c.endtime < chosen_course.starttime or chosen_course.endtime < c.starttime ) ;

    if no_collisions = 0 then
        return false;
    end if;
    return true;
end;
$$$;
```

- no_available_places(course_id)

```
create or replace function no_available_places (course_id integer)
returns smallint
language plpgsql
as $$
DECLARE
    takenPlaces integer;
    totalPlaces integer;

Begin
    if not exists(select * from courses as c where course_id = c.courseid) then
        raise exception 'Course not found';
    end if;
    select c.numberofplaces into totalPlaces
    from courses as c
    where c.courseid = course_id limit 1;

    select COUNT(*) into takenPlaces
    from students_courses as sc
    where sc.courses_courseid = course_id;

    return totalPlaces - takenPlaces;
end
$$;
```

b) Triggery

- enroll_trigger_function

```
create or replace function enroll_trigger_function()
returns trigger as
$$
declare
    chosen_course courses%rowtype;
BEGIN
    if no_available_places( course_id: new.courses_courseid) = 0 then
        raise exception 'no available place';
    end if;
    if collision_exists( student_id: new.students_studentid, course_id: new.courses_courseid) then
        raise exception 'schedule collision';
    end if;
    select into chosen_course * from courses as c where c.courseid = new.courses_courseid;
    if get_used_etcs( student_id: new.students_studentid) + chosen_course.etcs > 30 then
        raise exception 'not enough etcs';
    end if;
    return new;
end
$$language plpgsql;
```

```
create trigger enroll_trigger before insert
on students_courses for each row
execute procedure enroll_trigger_function();
```

4. Hibernate

Zgodnie ze schematem bazy danych dodaliśmy klasy:

- Student

```
@Entity
@Table(name = "students", schema = "public", catalog = "enroll_database")
public class Student {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private int studentId;
    private String firstName;
    private String lastName;
    private String pesel;
    private short semester;
    @ManyToMany(fetch = FetchType.EAGER)
    Set<Course> courses;
    @ManyToMany
    Set<Faculty> faculties;
    public Set<Faculty> getFaculties() { return faculties; }
    public Set<Course> getCourses() { return courses; }
    public void enroll(Course course) { courses.add(course); }
    public int getStudentId() { return studentId; }
    public String getFirstName() { return firstName; }
```

- Course

```
@Entity
@Table(name = "Courses", schema = "public", catalog = "enroll_database")
public class Course {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private short courseId;
    private String name;
    private short numberOfPlaces;
    private short etcs;
    private String description;
    private short semester;
    private short weekDay;
    private Time startTime;
    private Time endTime;
    @ManyToOne
    private Faculty faculty;
    @ManyToOne
    private Lecturer lecturer;
    @ManyToMany(mappedBy = "courses", fetch = FetchType.EAGER)
    private Set<Student> students;
```

- Faculty

```
@Entity
@Table(name = "Faculties", schema = "public", catalog = "enroll_database")
public class Faculty {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private short facultyId;
    private String name;
    @ManyToMany(mappedBy = "faculties")
    private Set<Student> students;
```

- Lecturer

```
@Entity
@Table(name = "Lecturers", schema = "public", catalog = "enroll_database")
public class Lecturer {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private short lecturerId;
    private String firstName;
    private String lastName;
    private String degree;
```

Plik konfiguracyjny Hibernate'

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:postgresql://localhost:5433/enroll_database</property>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.hbm2ddl.auto"> update</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">postgres</property>

        <mapping class="hibernate_classes.Course"/>
        <mapping class="hibernate_classes.Faculty"/>
        <mapping class="hibernate_classes.Lecturer"/>
        <mapping class="hibernate_classes.Student"/>

    </session-factory>
</hibernate-configuration>
```


5. Klasa CommunicationUtil

Klasa ta umożliwia komunikację z bazą danych jak również przechowuje informację na temat zalogowanego studenta.

Tworzenie sesji do komunikacji z bazą:

```
private static final SessionFactory ourSessionFactory;
private static Session session = null;
static {
    try {
        Configuration configuration = new Configuration();
        configuration.configure();

        ourSessionFactory = configuration.buildSessionFactory();
    } catch (Throwable ex) {
        throw new ExceptionInInitializerError(ex);
    }
}

public static Session getSession() throws HibernateException {
    if(session == null){

        System.out.println();
        System.out.println("OPENING NEW SESSION");
        System.out.println();
        return ourSessionFactory.openSession();
    }else if(!session.isOpen()){

        System.out.println("SESSION WAS CLOSED, OPENING NEW SESSION");
        return ourSessionFactory.openSession();
    }

    return session;
}
```

Logowanie się do aplikacji, wraz ze sprawdzaniem czy istnieje student o danym indeksie

```
public void login(int indexNumber) {
    Transaction tx = null;

    try {
        session = getSession();
        tx = session.beginTransaction();
        tx.setTimeout(5);

        StoredProcedureQuery q = session.createStoredProcedureQuery("contains_student_id")
            .registerStoredProcedureParameter(1, Integer.class, ParameterMode.IN)
            .setParameter(1, indexNumber);
        logged = (boolean) q.getSingleResult();
        if (logged) {
            TypedQuery<Student> q2 =
                session
                    .createQuery("from hibernate_classes.Student as student where student.id = :id", Student.class);
            q2.setParameter("id", indexNumber);
            loggedStudent = q2.getSingleResult();
            System.out.println("successfully logged in!");
            System.out.println("Hello " + loggedStudent.getFirstName());
        } else {
            tx.commit();
            throw new IllegalArgumentException("There is no student with this IndexNumber");
        }
        tx.commit();
    } catch (RuntimeException e) {

        if(e.getClass() == IllegalArgumentException.class)
            throw new IllegalArgumentException("There is no student with this IndexNumber");
        try {
            if (tx != null) tx.rollback();
        } catch (RuntimeException rbe) {
            System.out.println("Couldn't roll back transaction");
        }
        if(session != null) session.close();
    }
}
```

Pobieranie listy przedmiotów dostępnych dla zalogowanego studenta

```
public List<Course> getCourses() {

    Transaction tx = null;
    List<Course> courses = null;

    try {
        session = getSession();

        tx = session.beginTransaction();
        tx.setTimeout(5);

        StoredProcedureQuery q = session.createStoredProcedureQuery("get_courses", Course.class)
            .registerStoredProcedureParameter(1, Integer.class, ParameterMode.IN)
            .setParameter(1, loggedStudent.getId());
        courses = q.getResultList();
        tx.commit();

    } catch (RuntimeException e) {
        try {
            if (tx != null) tx.rollback();
        } catch (RuntimeException rbe) {
            System.out.println("Couldn't roll back transaction");
        }
        if(session != null)session.close();
    }

    return courses;
}
```

Pobieranie listy dostępnych miejsc na dany przedmiot

```
public short getNumberOfAvailablePlaces(Course course){
    Transaction tx = null;
    short result = 0;
    try{
        session = getSession();

        tx = session.beginTransaction();
        tx.setTimeout(5);
        StoredProcedureQuery q = session
            .createStoredProcedureQuery("s: no_available_places")
            .registerStoredProcedureParameter("i: 1, Short.class, ParameterMode.IN")
            .setParameter("i: 1, course.getCourseId());

        result = (short) q.getSingleResult();
        tx.commit();
    } catch (RuntimeException e) {
        try {
            if (tx != null) tx.rollback();
            System.out.println(e.getMessage());
        } catch (RuntimeException rbe) {
            System.out.println("Couldn't roll back transaction");
        }
        if(session != null) session.close();
    }
    return result;
}
```

Sprawdzanie czy student jest zapisany na dany przedmiot

```
public boolean isEnrolled(Course course) {

    Transaction tx = null;
    boolean result = false;

    try {
        session = getSession();

        tx = session.beginTransaction();
        tx.setTimeout(5);

        StoredProcedureQuery q = session.createStoredProcedureQuery("is_student_enrolled")
            .registerStoredProcedureParameter(1, Integer.class, ParameterMode.IN)
            .registerStoredProcedureParameter(2, Short.class, ParameterMode.IN)
            .setParameter(1, loggedStudent.getStudentId())
            .setParameter(2, course.getCourseId());

        result = (boolean) q.getSingleResult();
        tx.commit();

    } catch (RuntimeException e) {
        try {
            if (tx != null) tx.rollback();
            System.out.println(e.getMessage());
        } catch (RuntimeException rbe) {
            System.out.println("Couldn't roll back transaction");
        }
        if(session != null) session.close();
    }
    return result;
}
```

Funkcja do zapisania studenta na dane zajęcia, w razie problemu funkcja rzuca wyjątek.

```
public void enroll(Course course) {
    if (!isEnrolled(course)) {

        Transaction tx = null;

        try {

            session = getSession();
            tx = session.beginTransaction();
            tx.setTimeout(10);

            loggedStudent.enroll(course);
            course.addStudent(loggedStudent);
            session.update(loggedStudent);
            session.update(course);

            tx.commit();

        } catch (RuntimeException e) {
            try {
                loggedStudent.getCourses().remove(course);
                course.getStudents().remove(loggedStudent);
                session.update(loggedStudent);
                session.update(course);
                if (tx != null) tx.rollback();
            } catch (RuntimeException rbe) {
                System.out.println(rbe.getMessage());

                System.out.println("cant rollback");
            }

            if(session != null)session.close();
            throw e;
        }
    }
}
```

Funkcja do wypisania studenta z danego przedmiotu

```
public void unroll(Course course) {
    if (isEnrolled(course)) {

        Transaction tx = null;

        try {
            session = getSession();

            tx = session.beginTransaction();
            tx.setTimeout(5);

            loggedStudent.getCourses().remove(course);
            course.getStudents().remove(loggedStudent);
            session.update(loggedStudent);
            session.update(course);
            tx.commit();

        } catch (RuntimeException e) {
            try {
                System.out.println(e.getMessage());

                if (tx != null) tx.rollback();
            } catch (RuntimeException rbe) {
                System.out.println("Couldn't roll back transaction");
            }
            if(session != null)session.close();
        }
    }
}
```

6. Instrukcja do uruchomienia aplikacji

a) klonujemy repozytorium:

```
git clone https://github.com/pwgacek/projekt-bd2.git
```

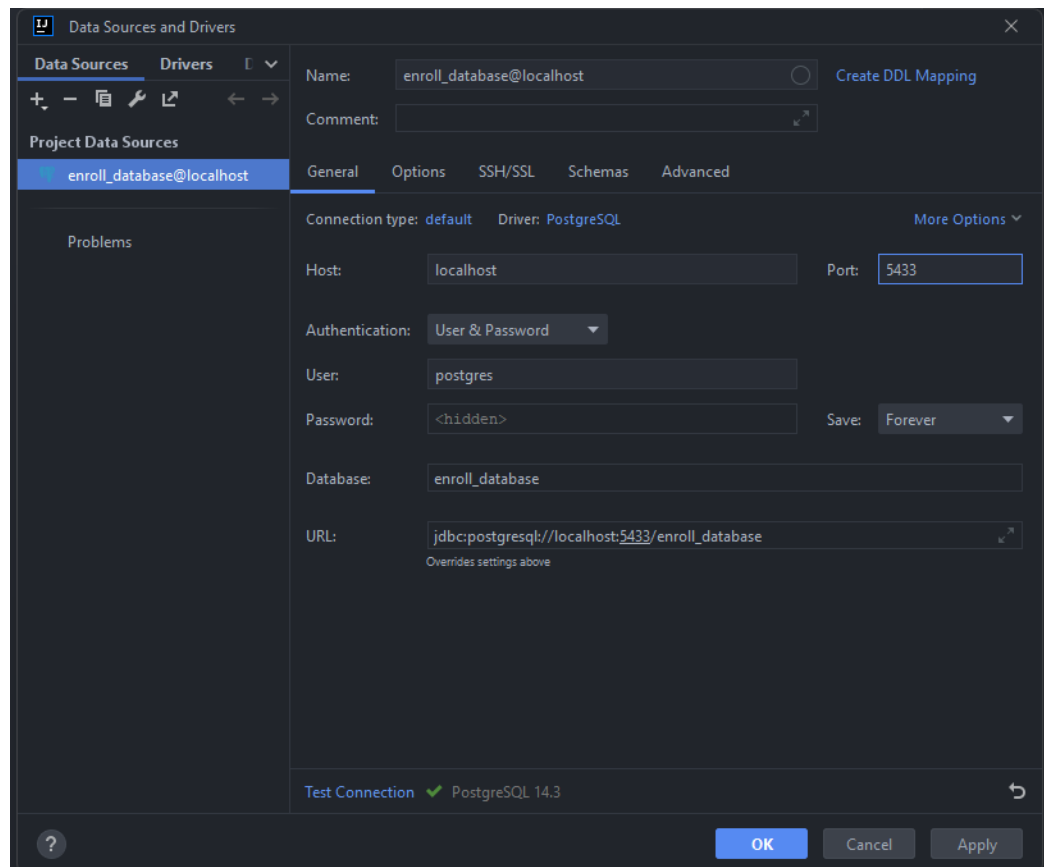
b) pobieramy bazę z dockera i odpalamy ją na porcie np 5433:

```
docker pull pwgacek/db_project
```

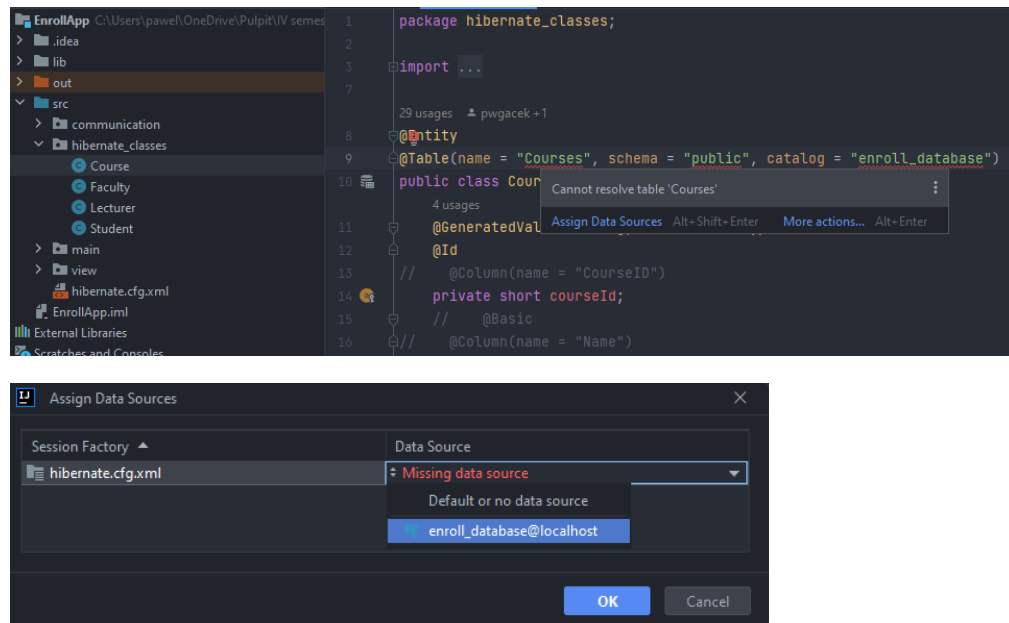
```
docker run --name db_container --env PGDATA=postgres -d -p 5433:5432 -i
```

```
pwgacek/db_project
```

c) łączymy się z bazę (hasło: postgres, login : postgres, nazwa bazy: enroll_database):



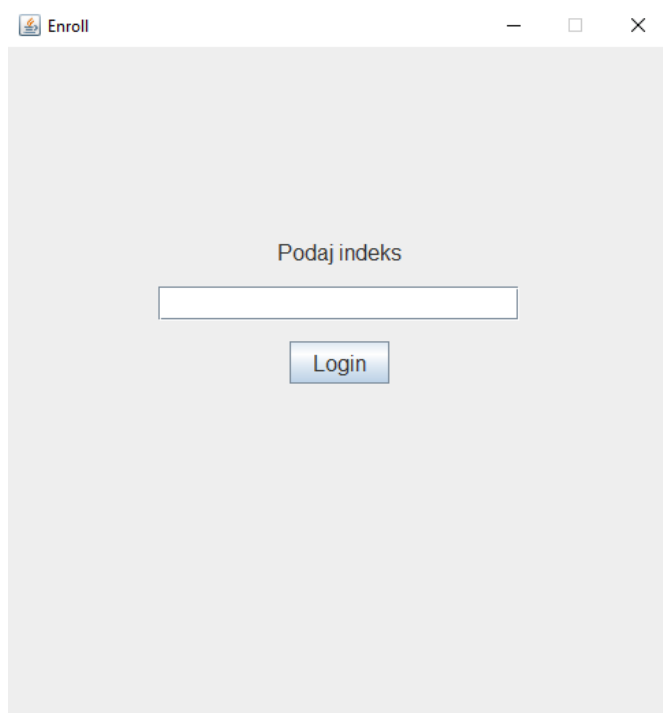
d) w jednej z klas Hibernate'a po najechaniu na błąd wybieramy opcję Assign Data Source -> enroll_database



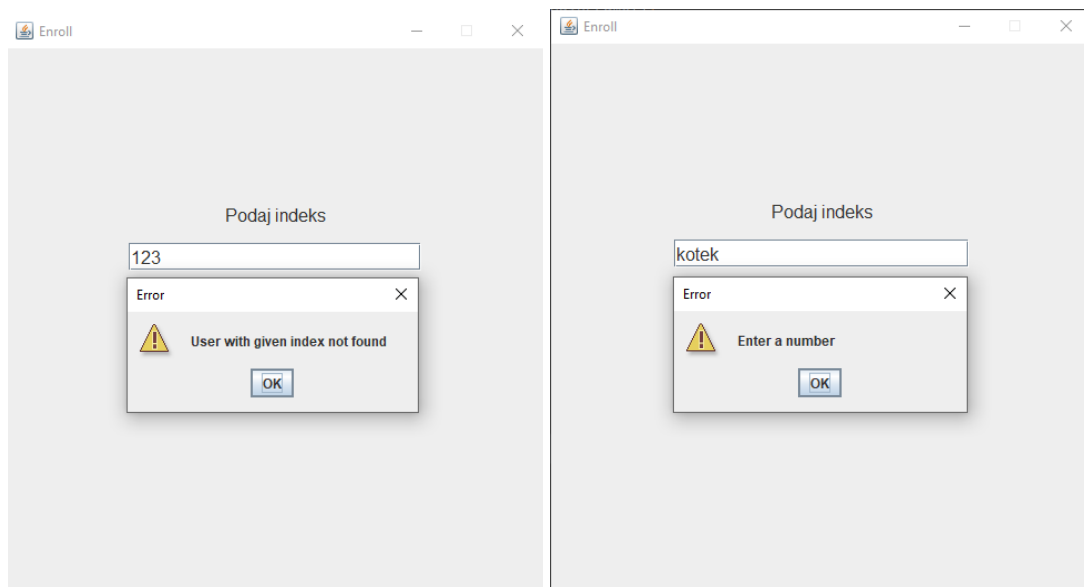
e) aplikacja jest gotowa do użycia, przykładowe loginy: 100123, 100124, 100128

7. Korzystanie z aplikacji

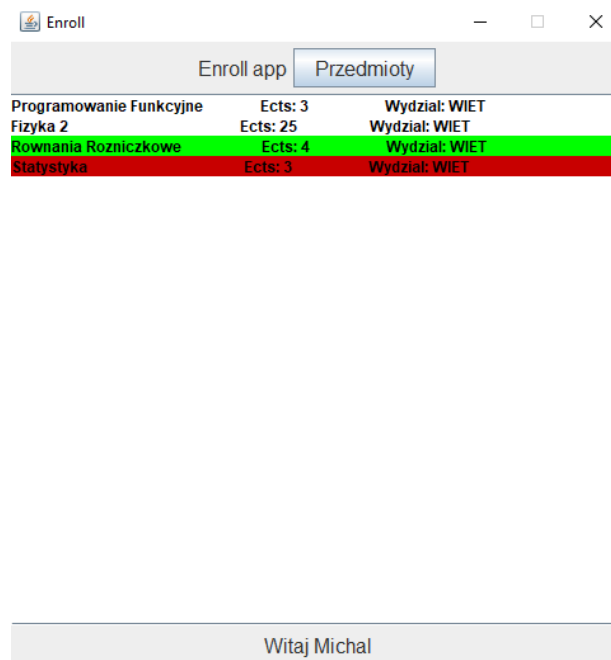
Po uruchomieniu aplikacji widzimy ekran logowania.



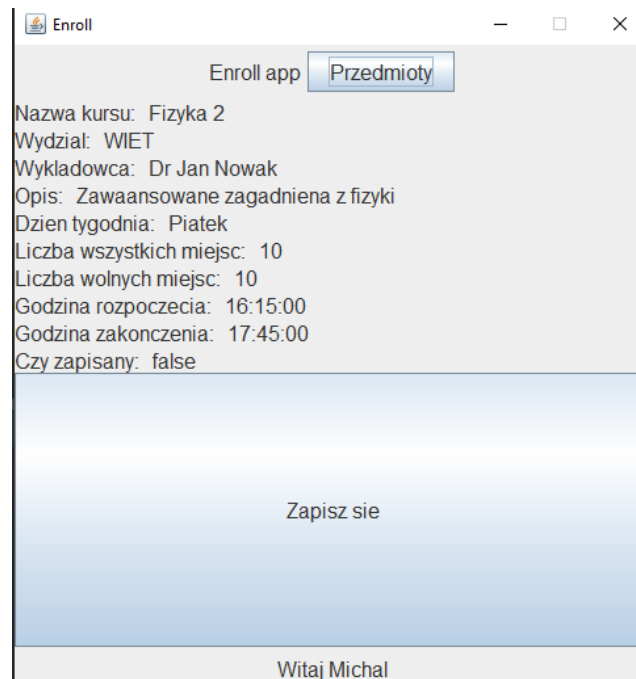
Podanie błędnego indeksu generuje komunikat o błędzie



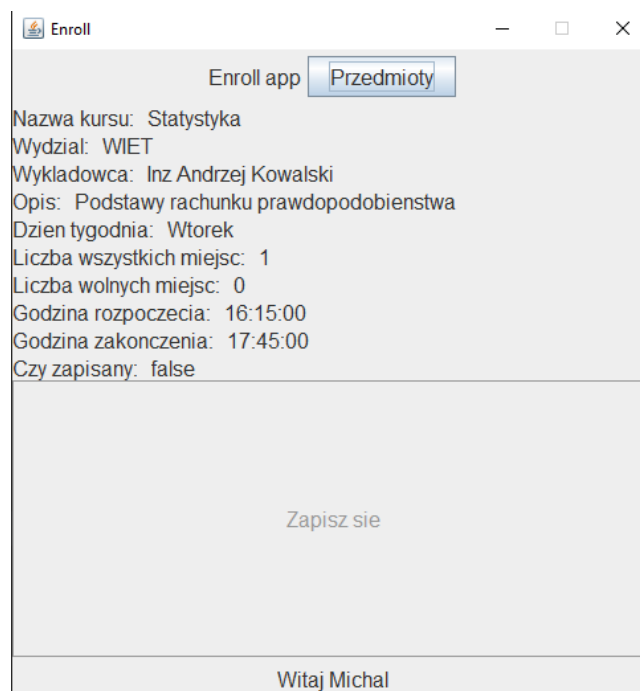
Po zalogowaniu widzimy listę przedmiotów, które są realizowane na naszym semestrze oraz na naszym wydziale. Przedmioty na które jesteśmy zapisani są zaznaczone zielonym kolorem, natomiast przedmioty, na których nie ma już wolnych miejsc czerwonym.



Po wybraniu danego przedmiotu dostajemy szczegółowe informacje na jego temat wraz z możliwością zapisania się na zajęcia (jeżeli zostały wolne miejsca) lub wypisania się - jeżeli byliśmy już wcześniej zapisani



W przypadku braku wolnych miejsc przycisk do zapisu jest nieaktywny:



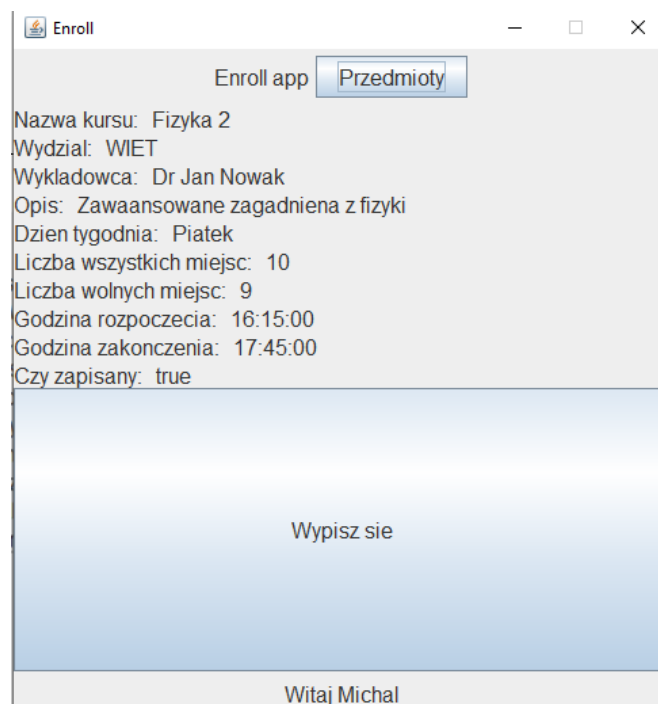
Enroll app **Przedmioty**

Nazwa kursu: Statystyka
Wydział: WIET
Wykładowca: Inz Andrzej Kowalski
Opis: Podstawy rachunku prawdopodobieństwa
Dzień tygodnia: Wtorek
Liczba wszystkich miejsc: 1
Liczba wolnych miejsc: 0
Godzina rozpoczęcia: 16:15:00
Godzina zakończenia: 17:45:00
Czy zapisany: false

Zapisz sie

Witaj Michał

Widok przedmiotu na który jesteście zapisani:



Enroll app **Przedmioty**

Nazwa kursu: Fizyka 2
Wydział: WIET
Wykładowca: Dr Jan Nowak
Opis: Zawaansowane zagadnienia z fizyki
Dzień tygodnia: Piątek
Liczba wszystkich miejsc: 10
Liczba wolnych miejsc: 9
Godzina rozpoczęcia: 16:15:00
Godzina zakończenia: 17:45:00
Czy zapisany: true

Wypisz sie

Witaj Michał

Podczas próby zapisania się na zajęcia może wystąpić problem, np. kolizja z innymi zajęciami. Dostajemy wtedy komunikat o błędzie.

