Korištenje baza podataka u Java web aplikacijama

Uvod

- Za trajno pohranjivanje podataka web aplikacije uglavnom koriste baze podataka koje se mogu nalaziti u memoriji same aplikacije (*in-memory embedded database*) ili na poslužitelju s bazom podataka (*database server*)
- Tijekom razvoja je često vrlo korisno imati bazu podataka koja se kreira prilikom pokretanja poslužitelja, kako bi se lako mogle dodavati izmjene bez potrebe za promjenom na „fizičkoj” bazi podataka

Uvod

- U slučaju korištenja fizičke baze podataka podaci za pristupanje nalaze se na aplikacijskom poslužitelju (konfiguraciji Tomcat poslužitelja, unutar „server.xml” i „context.xml” datoteka)
- U tom slučaju se u Java kodu nalazi samo JNDI identifikator za *data source* (bazu podataka)

Resursi i JNDI

- *Resurs* predstavlja objekt koji omogućava spajanje na sustave poput baze podataka
- JNDI – engl. *Java Naming and Directory Interface*: servis koji omogućava komponentama lociranje drugih komponenata i resursa
- JDBC resurs se često naziva i *data source*
- Svaki resurs se naziva JNDI imenom koji se može lako zapamtiti, npr. „jdbc/biljeskeDatasource”

Resursi i JNDI

- JNDI resursi definiraju se unutar JNDI namespacea (kod Tomcata unutar „<Resource>” tagova u „context.xml” datoteci i „<GlobalNamingResources>” unutar „server.xml” datoteke)
- Unutar tih konfiguracijskih datoteka moguće je definirati više različitih JNDI resursa (baza podataka) koje se koriste u svim aplikacijama *deployanim* na Tomcat poslužitelju

Konfiguriranje Tomcata

- Unutar „server.xml” konfiguracije Tomcata moguće je definirati globalni resurs koji definira detalje vezane uz bazu podataka i JNDI resurs:

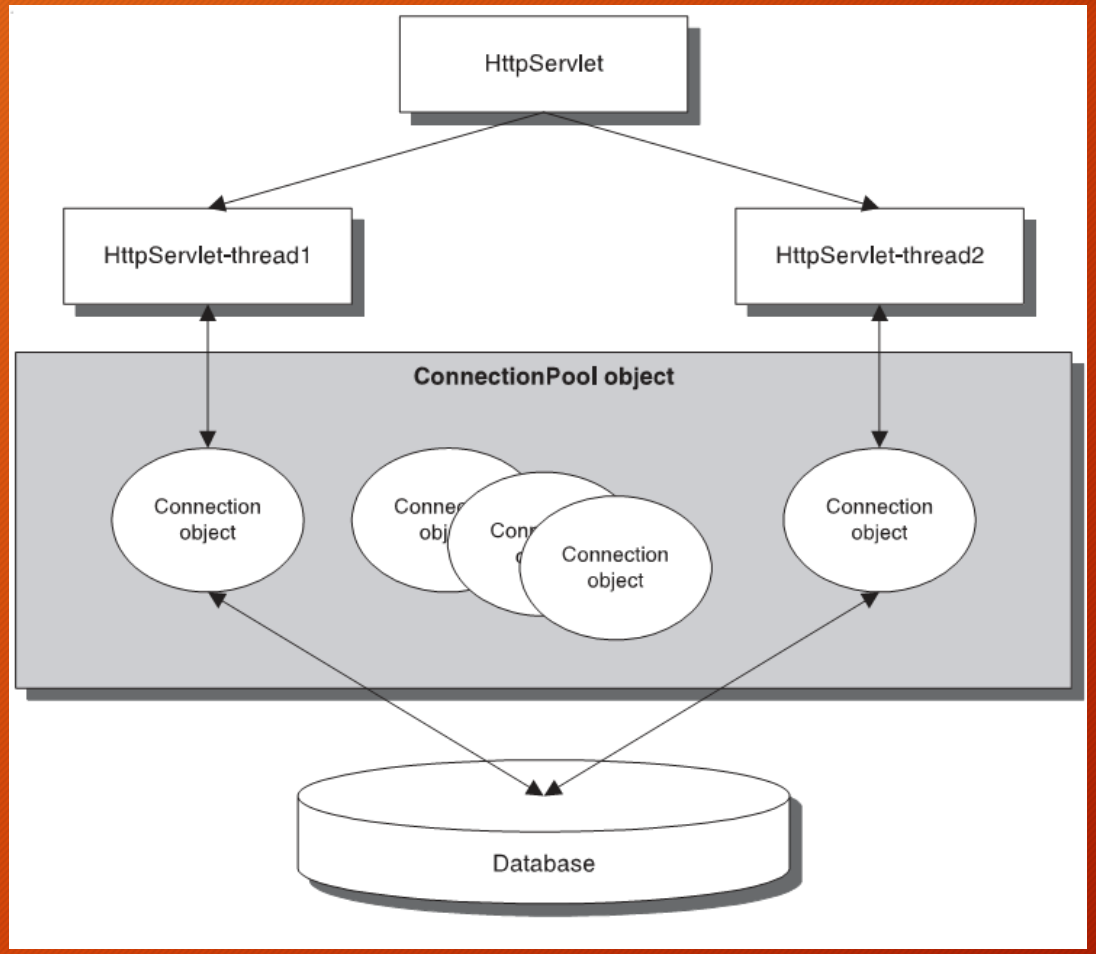
```
<Resource auth="Container"
driverClassName="org.h2.Driver"
maxActive="8"
name="biljeznicaDatabase"
username="student"
password="student"
type="javax.sql.DataSource"
url="jdbc:h2:tcp://localhost/~/BiljeznicaDB" />
```

Konfiguriranje Tomcata

- Unutar „context.xml” konfiguracije Tomcata moguće je definirati vezu na globalni resurs definiran unutar „server.xml” konfiguracije:

```
<ResourceLink
        name="jdbc/BiljeznicaDatabase"
        global="biljeznicaDatabase"
        type="javax.sql.DataSource " />
```

Connection pool



Connection pool

- Kod inicijalizacije *connection poola* kreira se jedan „ConnectionPool” objekt koji sadrži više „Connection” objekata
- Prilikom pristupa korisnika servletu kreira se po jedan „Thread” koji dohvaća „Connection” objekt iz *poola* za pristupanje bazi podataka
- Kad „Thread” servleta više ne treba „Connection objekt”, vraća ga natrag u *pool*

Connection pool kod Tomcata

- Tomcat koristi DBCP (engl. *DataBase Connection Pool*) koji je ugrađen pomoću „tomcat-dbcp.jar” datoteke (*Jakarta-Common* projekt)
- Omogućava definiranje parametara o broju aktivnih konekcija unutar *connection poola*, broj konekcija koje mogu biti slobodne, maksimalno vrijeme koje se čeka dobivanje konekcije, ali i postavljanje opcije koja oslobađa konekcije koje nisu zatvorene (removeAbandoned=„true”) itd.

Primjer konfiguracije *connection poola*

```
<Context path="/DBTest" docBase="DBTest"
      reloadable="true" crossContext="true">

   <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
            maxActive="100" maxIdle="30" maxWait="10000"
            username="javauser" password="javadude"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/javatest"/>

   <!-- maxActive: Maximum number of dB connections in pool. Make sure you
        configure your mysqld max_connections large enough to handle
        all of your db connections. Set to -1 for no limit.
        -->

   <!-- maxIdle: Maximum number of idle dB connections to retain in pool.
        Set to -1 for no limit.
        -->
```

Primjer konfiguracije *connection poola*

```
<!-- maxWait: Maximum time to wait for a dB connection to become available
     in ms, in this example 10 seconds. An Exception is thrown if
     this timeout is exceeded.  Set to -1 to wait indefinitely. -->

<!-- username and password: MySQL dB username and password for dB
      connections -->

<!-- driverClassName: Class name for the old mm.mysql JDBC driver is
     org.gjt.mm.mysql.Driver - we recommend using Connector/J though.
     Class name for the official MySQL Connector/J driver is
     com.mysql.jdbc.Driver. -->

<!-- url: The JDBC connection url for connecting to your MySQL dB.
     -->

</Context>
```

Konfiguriranje korisničkog repozitorija pomoću JNDI resursa

- Kako bi bilo moguće koristiti JNDI resurs definiran na razini konfiguracije Tomcata, potrebno je dodati „jdbc-user-service” *tag*:

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/KonferencijaDatabase" />

<jdbc-user-service id="userService"
                   data-source-ref="dataSource" />
```

- U slučaju korištenja Spring boot programskog okvira, ta konfiguracija se podrazumijeva i nije potrebna

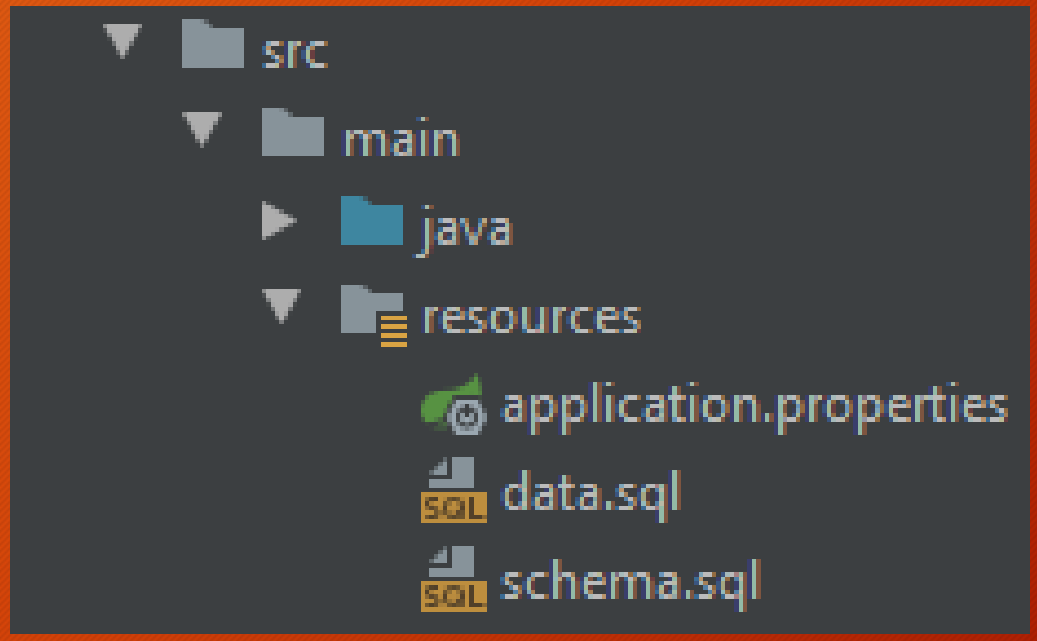
In-memory baza podataka

- H2 baza podataka je idealna za korištenje u *in-memory* načinu rada
- Konfigurira se na način da se definira „jdbc:embedded-database” konfiguracija na način da se definiraju datoteke koje sadrže SQL skritu za kreiranje i popunjavanje baze podataka:

```
<jdbc:embedded-database id="dataSource" type="H2">
   <jdbc:script location="classpath:biljeznica/db/jdbc/schema.sql" />
   <jdbc:script location="classpath:biljeznica/db/jdbc/test-data.sql" />
</jdbc:embedded-database>
```

In-memory baza podataka

- U slučaju korištenja Java konfiguracije skripte kod korištenja Spring boota, za kreiranje i popunjavanje baze podataka dovoljno je postaviti unutar „resources” mape:



Implementacija „Repository” sloja

- Kao zamjenu za „Mock” implementaciju klase koja uvijek vraća „zahardkodirane” podatke, implementacija „Repository” sloja podataka započinje na način da je prvo potrebno definirati sučelje s popisom metoda, npr.:

```
interface StudentRepository {
    Set<Student> findAll();
    Optional<Student> findByJMBAG(String JMBAG);
    Optional<Student> save(Student student);
    Optional<Student> update(String JMBAG, Student updatedStudent);
    void deleteByJMBAG(String JMBAG);
}
```

Implementacija „Repository” sloja

- Implementacija navedenog sučelja temelji se na SQL upitima i korištenju objekta klase „JdbcTemplate” i „SimpleJdbcInsert”:

```
@Primary
@Repository
class JdbcStudentRepository implements StudentRepository{

    private static final String SELECT_ALL =
            "SELECT id, jmbag, first_name, last_name, ects_points, date_of_birth FROM student";

    private final JdbcTemplate jdbc;
    private final SimpleJdbcInsert inserter;

    public JdbcStudentRepository(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
        this.inserter = new SimpleJdbcInsert(jdbc)
                .withTableName("student")
                .usingGeneratedKeyColumns("id");
    }
```

Implementacija „Repository” sloja

- Ostale metode je moguće implementirati kako slijedi:

```
@Override
public Set<Student> findAll() { return Set.copyOf(jdbc.query(SELECT_ALL, this::mapRowToStudent)); }

@Override
public Optional<Student> findByJMBAG(String JMBAG) {
    try{
        return Optional.ofNullable(
                jdbc.queryForObject( sql: SELECT_ALL + " WHERE jmbag = ?", this::mapRowToStudent, JMBAG)
        );
    } catch (EmptyResultDataAccessException e){
        return Optional.empty();
    }
}
```

Implementacija „Repository” sloja

```java
@Override
public Optional<Student> save(Student student) {
    try {
        student.setId(saveStudentDetails(student));
        return Optional.of(student);
    } catch (DuplicateKeyException e){
        return Optional.empty();
    }
}
```

```java
private long saveStudentDetails(Student student) {
    Map<String, Object> values = new HashMap<>();

    values.put("first_name", student.getFirstName());
    values.put("last_name", student.getLastName());
    values.put("jmbag", student.getJmbag());
    values.put("date_of_birth", student.getDateOfBirth());
    values.put("ects_points", student.getNumberOfECTS());

    return inserter.executeAndReturnKey(values).longValue();
}
```

Implementacija „Repository” sloja

```java
@Override
public Optional<Student> update(String JMBAG, Student updatedStudent) {
    int executed = jdbc.update( sql: "UPDATE student set " +
                    "first_name = ?, " +
                    "last_name = ?, " +
                    "ects_points = ?, " +
                    "date_of_birth = ? " +
                        "WHERE jmbag = ?",
                    updatedStudent.getFirstName(),
                    updatedStudent.getLastName(),
                    updatedStudent.getNumberOfECTS(),
                    updatedStudent.getDateOfBirth(),
                    updatedStudent.getJmbag()
            );

    if(executed > 0){
        return Optional.of(updatedStudent);
    } else {
        return Optional.empty();
    }
}
```

Implementacija „Repository” sloja

```java
@Override
public void deleteByJMBAG(String JMBAG) { jdbc.update( sql: "DELETE FROM student WHERE jmbag = ?", JMBAG); }

private Student mapRowToStudent(ResultSet rs, int rowNum) throws SQLException {
    return new Student(
            rs.getLong( columnLabel: "id"),
            rs.getString( columnLabel: "first_name"),
            rs.getString( columnLabel: "last_name"),
            rs.getString( columnLabel: "jmbag"),
            rs.getDate( columnLabel: "date_of_birth").toLocalDate(),
            rs.getInt( columnLabel: "ects_points")
    );
}
```

Tablice u bazi podataka

- U datoteci „schema.sql” za opisati „Repository” sloj potrebno je kreirati sljedeće tablice:

```
create table if not exists student (
   id identity,
   jmbag varchar(10) not null unique,
   first_name varchar(100) not null,
   last_name varchar(100) not null,
   ects_points number not null,
   date_of_birth date not null
);

create table if not exists course (
   id identity,
   name varchar(100) not null,
   ects_points number not null
);

create table if not exists student_course (
   id identity,
   student_id bigint,
   course_id bigint,
   constraint fk_student foreign key (student_id) references student(id),
   constraint fk_course foreign key (course_id) references course(id)
)
```

Tablice u bazi podataka

- U datoteci „data.sql” za opisati „Repository” sloj potrebno je kreirati sljedeće tablice:

```
insert into student (id, first_name, last_name, jmbag, ects_points, date_of_birth)
            values (1, 'Ivo', 'Ivić', '0246053232', 120, NOW());
insert into student (id, first_name, last_name, jmbag, ects_points, date_of_birth)
            values (2, 'Pero', 'Perić', '0246053435', 120, NOW());

insert into course (id, name, ects_points)
            values (1, 'Web aplikacije u Javi', 6);
insert into course (id, name, ects_points)
            values (2, 'Programiranje u jeziku Java', 5);

insert into student_course (id, student_id, course_id)
            values (1, 1, 1);
insert into student_course (id, student_id, course_id)
            values (2, 2, 1);
insert into student_course (id, student_id, course_id)
            values (3, 2, 2);
```

Pitanja?