Java technológiák - 10. előadás
JDBC - adatbáziskezelés

ANTAL Margit

Sapientia - EMTE

2010

Az előadás célja

- JDBC API
- Data Access Object (DAO) tervezési minta
- Connection Pool
- DataSource

JDBC API

- Olyan Java típusok (osztályok és interfészek) gyűjteménye amelyek keretet biztosítanak adatbázis alapú rendszerek fejlesztésére
- Platformfüggetlen
- Az adatbázisokkal kapcsolatos kód DBMS-független

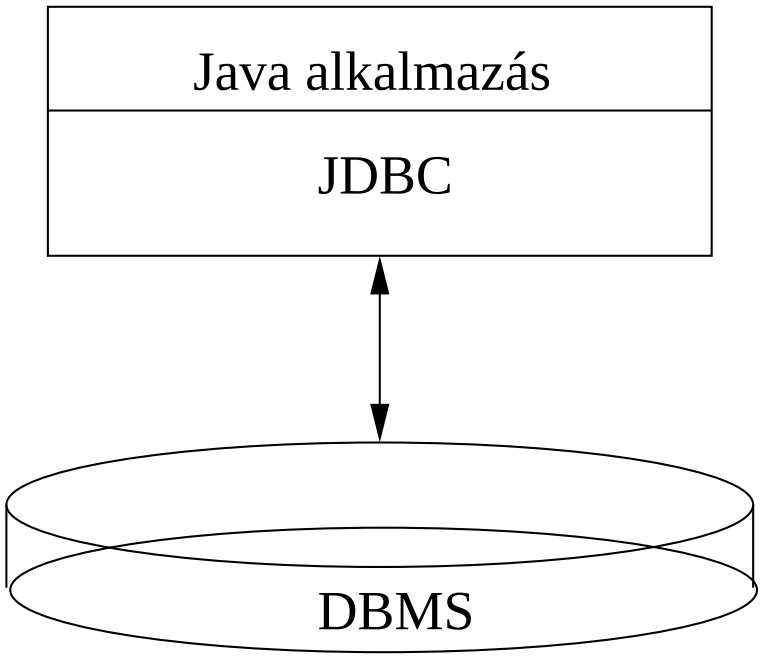
JDBC driver

- A DBMS készítője bocsájtja rendelkezésünkre.
- Feladata: átalakítani a JDBC metódushívásokat natív adatbázis API hívásokká

Driverek:

- Apache Java DB (Derby)
- MySql
- PostgreSQL
- Microsoft SQL server
- ...

JDBC API használat

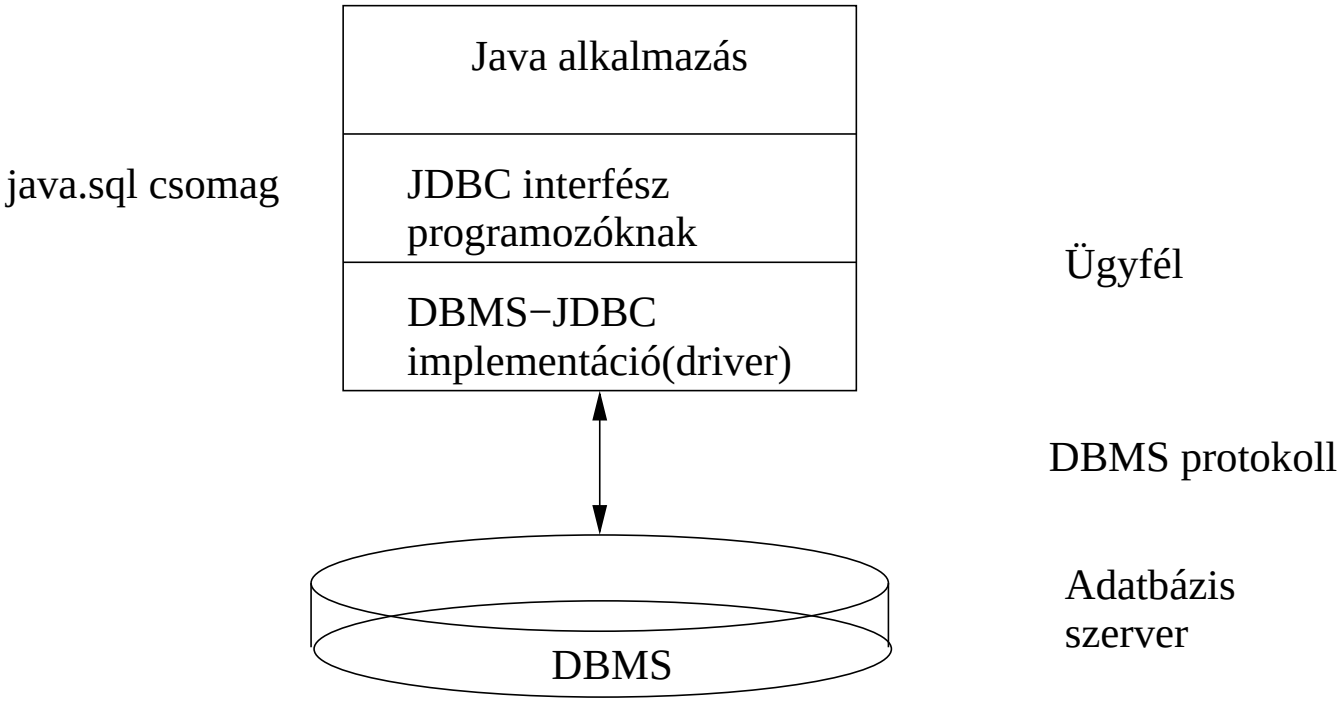


Ügyfél

DBMS protokoll

Adatbázis szerver

JDBC komponensek



Alapvető típusok

Helyük: java.sql csomag

- `java.sql.Connection`
- `java.sql.Statement`
- `java.sql.ResultSet`

JDBC API funkcionális osztályozás

- Adatforrás kapcsolat
- SQL utasítások küldése
- SQL eredmények lekérdezése
- SQL és Java típusok közötti leképzések
- Adatbázis metadadatok szolgáltatása
- Kivételek

Adatforrás kapcsolat

Típus	Ki biztosítja	Megjegyzés
DriverManager	java.sql	JDBC driverek managementje
Driver	Vendor	DBMS kapcsolat
Connection	Vendor	DBMS munkamenet managementje

SQL utasítások végrehajtása

Típus	Ki biztosítja	Megjegyzés
Statement	Vendor	Statikus SQL parancs
PreparedStatement	Vendor	Előfordított SQL parancs
CallableStatement	Vendor	Tárolt eljárás

PreparedStatement vs. CallableStatement

PreparedStatement vs. CallableStatement

- Az előfordított SQL parancs (PreparedStatement) objektum egy előzetesen lefordított SQL parancsot tartalmaz. Végrehajtása gyorsabb, mint a (Statement) objektumé.
- A CallableStatement tárolt eljárás hívására alkalmas.

CallableStatement

```
CREATE DEFINER=`antalm`@`localhost`
PROCEDURE `updatemissedquestions`(IN qid INT)
BEGIN
declare a integer;
set transaction isolation level serializable;
select hanyszor into a
  from missedquestions
       where kerdesid=qid;
if a>0  then
  update missedquestions
      set hanyszor=a+1 where kerdesid=qid;
else
  insert  into missedquestions (kerdesid,hanyszor)
          values (qid,1);
end if;

commit;
end;
```

SQL eredmények lekérdezése

Típus	Ki biztosítja	Megjegyzés
ResultSet	Vendor	

```
Vector<String> users=new Vector<String>();
Connection con=null;
PreparedStatement ps=null;
ResultSet rs=null;
con=datasource.getConnection();
ps=con.prepareStatement(GET_USERS);
rs=ps.executeQuery();
while (rs.next()){
  users.add(rs.getString(1));
}
```

SQL és Java típusok közötti leképzések

Típus	Ki biztosítja	Megjegyzés
Date	java.sql	SQL dátum
Time	java.sql	SQL idő
TimeStamp	java.sql	SQL időbélyeg

Adatbázis metaadatok szolgáltatása

Típus	Ki biztosítja	Megjegyzés
DatabaseMetaData	Vendor	Adatbázis információk
ResultSetMetaData	Vendor	ResultSet információk

Kivételek

Típus	Ki biztosítja	Megjegyzés
SQLException	java.sql	Adatbázis hozzáférés hiba
SQLWarning	java.sql	Adatbázis hozzáférés kivétel

Java DB

Jellemzők:

- nyílt forráskódú Apache DB projekt
- `http:\\db.apache.org`
- a driver: derbyclient.jar

Adatbázis kapcsolat megvalósítása I

1. A szállító cég JDBC driverének regisztrációja. Előzetesen gondoskodnunk kell a driver elérhetőségéről (Ajánlott hozzáadni a projekthez a derbyclient.jar csomagot)

```
Class.forName
("org.apache.derby.jdbc.ClientDriver");
```

2. DBMS munkamenet létrehozása

```
String url =
"jdbc:derby://localhost:1527//distedu";
Connection con =
DriverManager.getConnection(
  url,"username","password");
```

3. SQL lekérdezés objektum létrehozása:

```
Statement stmt = con.createStatement();
```

4. Lekérdezés elküldése

Adatbázis kapcsolat megvalósítása II

```
String query ="SELECT * FROM student";
ResultSet rs = stmt.executeQuery(query);
```

Ha módosítást végzünk az adatbázison:

```
String query =
"DELETE FROM student where id in(2,4,6)";
int nrRecords = stmt.executeUpdate(query);
```

5. Eredmények feldolgozása

```
List<Student> studentList =
new LinkedList<student>();
while( results.next() ){
  String SID   = results.getInt( 1 );
  String name  = results.getString(2);
  String address = results.getString(3);
  Student item  =
       new Student( SID, name, address);
```

Adatbázis kapcsolat megvalósítása III

```
    studentList.add(item);
}
```

6. Objektum lezárások

```
rs.close();
stmt.close();
con.close();
```

DBMS munkamenet

- URL határozza meg az adatbázis típust
- Szintaxis: `jdbc:alprotokoll:alnev`
  - alprotokoll: adatbázis-kezelő függő mechanizmus
  - alnév: az alprotokoll határozza meg ennek formáját
    - `jdbc:derby://localhost:1527/student`
    - `jdbc:odbc:images`

Hordozhatóság növelése

Az adatbázisfüggő adatokat helyezzük el egy tulajdonságfájlban:

```
jdbcDriver =
org.apache.derby.jdbc.ClientDriver

jdbcURL = jdbc:derby://localhost:1527/student

jdbcUser = public

jdbcPassword = public
```

Tulajdonságok betöltése fájlból

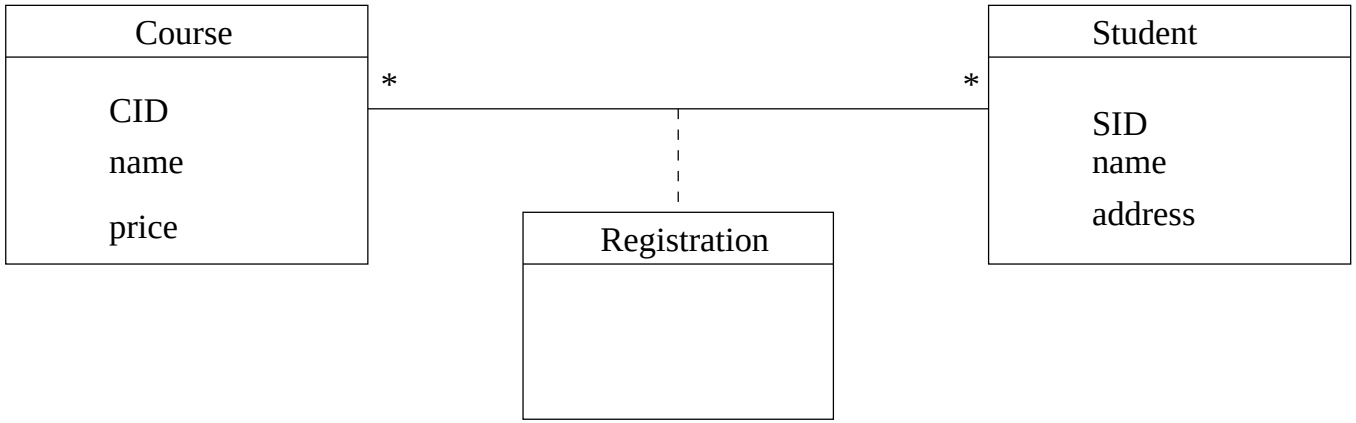
```
Properties p = new Properties();
p.load(new FileInputStream("database"));

String jdbcDriver=p.getProperty
                    ("jdbcDriver");
String jdbcUrl=p.getProperty("jdbcUrl");
String jdbcUser=p.getProperty("jdbcUser");
String jdbcPassword=p.getProperty
                    ("jdbcPassword");
```

Adatbázisok használata webalkalmazásokban

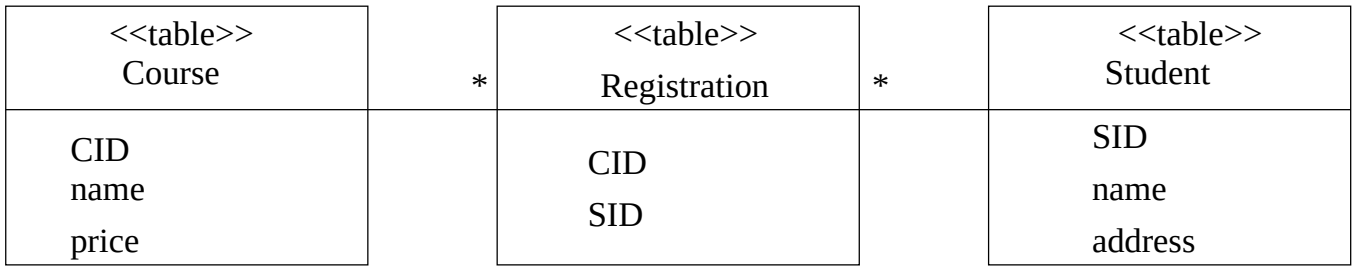
- Az alkalmazás modell elemeinek megtervezése.
- A modell elemek leképzése adattáblákra.
- Adatelérési tervezési minta használatára épülő üzleti szervíz komponensek megtervezése.

Modell elemek leképzése adattáblákra

Domain Objects



Database Tables



Adatelérési objektum tervezési minta(DAO)

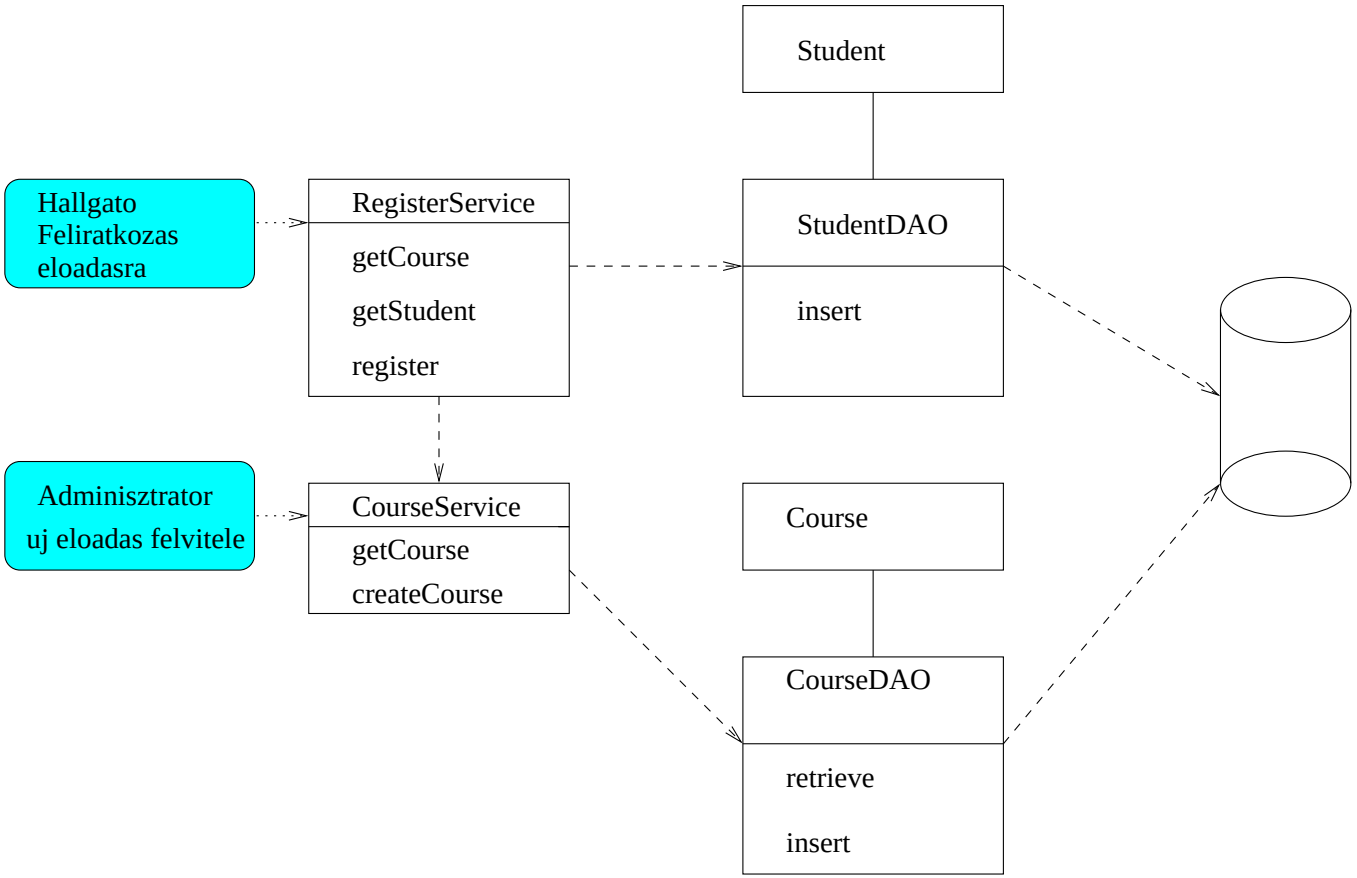
Data Access Object

Megkönnyíti az alkalmazás karbantartását, elválasztva az üzleti logikát az adatelérési logikától.
Általában minden egyes entitáshoz (domain object) készül egy-egy DAO

Előnyök:

- üzleti logika és adatelérési logika szétválasztása
- az adatelérési logika újrafelhasználható

DAO alkalmazása



A hagyományos adatbázis kapcsolódás kritikája

- Egy Java EE alkalmazásban több helyen is szükség van adatbázis kapcsolatra: webréteg (webkonténer), üzleti réteg (alkalmazásszerver)!!!
- Ha egy szervlet hoz létre egy adatbázis kapcsolatot, akkor ez a kapcsolat megmarad, viszont párhuzamos kiszolgálás esetén szinkronizálást igényel. Csak a webrétegben érhető el!!!
- Egy másik lehetőség egy kapcsolathalmaz (connection pool) állandó fenntartása, ahonnan igény szerint lehet kapcsolatot kivenni, majd használat után visszatenni. A kapcsolathalmazt lehet a webalkalmazás hatókörében tárolni. Ekkor viszont csak a web komponensek férnek hozzá, az üzleti réteg komponensei viszont nem.

ResourcePool - erőforrás gyűjtemény

Java EE példák:

- alkalmazásszerver (webkonténer): ThreadPool - szálgyűjtemény
- alkalmazásszerver : EJBPool - Enterprise JavaBeans gyűjtemény
- alkalmazásszerver : ConnectionPool - adatbázis kapcsolat gyűjtemény

Milyen problémát old meg?

- növeli az alkalmazás skálázhatóságát
- hatékony memóriakezelés: osztott memória

Adatbázis kapcsolat

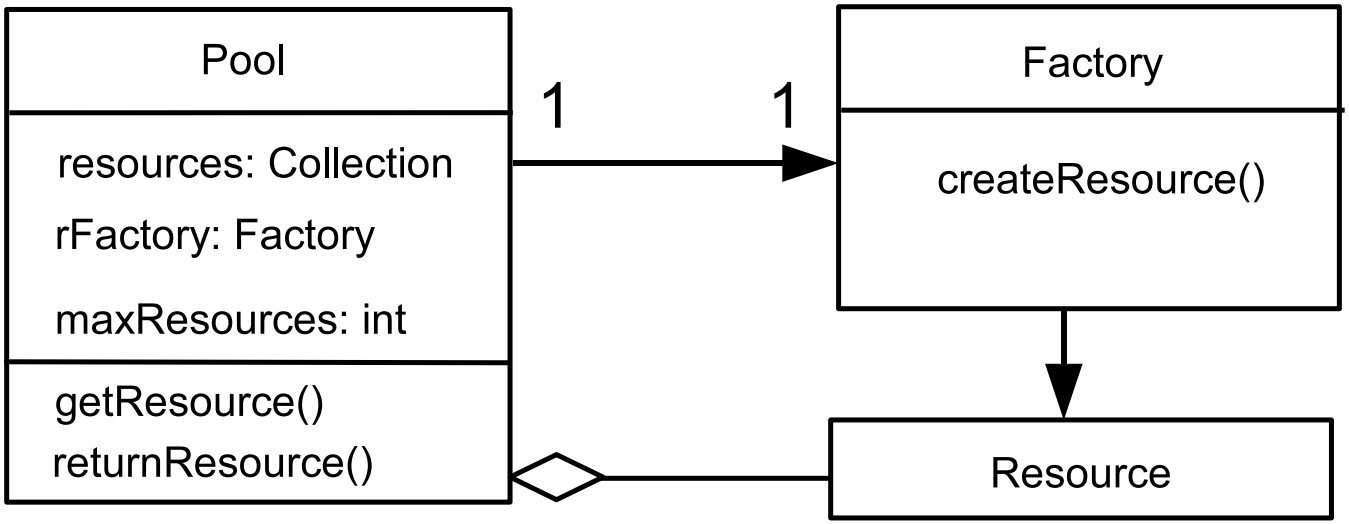
1. Adatbázis kapcsolatobjektum létrehozása.
2. Adatbázis elérési-, illetve felhasználó hitelesítéséhez szükséges adatok beállítása.
3. Hálózati kapcsolat létrehozása az adatbázissal.
4. Felhasználó hitelesítéséhez szükséges adatok elküldése.
5. Felhasználó hitelesítése.

$\Rightarrow$ Időigényes

Megoldás

- Az alkalmazás indításakor létrehozunk egy kapcsolathalmazt (Connection Pool).
- Az alkalmazás komponese szükség esetén kivesz egy felépített kapcsolatot a halmazból.
- A műveletek elvégzése után visszateszi a kapcsolatot a halmazba.
- A kapcsolathalmaz (Conection Pool) mérete állítható az igényeknek megfelelően - ezt minden alkalmazásra be kell hangolni.

Connection Pool



Adatforrás és JNDI API

DataSource

Olyan erőforrás, amely egy adatbázis kapcsolathoz szükséges információkat tartalmazza:

- URL
- driver
- username
- password

JNDI - Java Naming and Directory Interface

- Névadáshoz és könyvtárak eléréhez használt API.
- Enterprise alkalmazások használják a komponensek elérésére.
- Lehetővé teszi a komponensek regisztrációját és elérését.
- Minden Java EE alkalmazásszerver biztosítja ezt a szolgáltatást. (a Tomcat is !!!)

Adatforrás használatának előnyei

- Kevesebb munka - Nem kell megírni a ”connection pool” mechanizmust
- Hordozhatóság - bármely alkalmazásszerveren fog működni

Adatforrás konfiguráció

- Alkalmazásszerverfüggő

Netbeans 6.1:

```
File--->New File--->GlassFish
    JDBC Connection Pool
    JDBC Resource
```

Adatforrás komponens használata

- JNDI szolgáltatás - adatforrás komponens megkeresése
- `getConnection` metódus hívása
- JDBC API használata az adatbázissal való kommunikáció lebonyolítására

A DataSource interfész

```
package javax.sql;
import java.sql;

public interface DataSource{
  Connection getConnection();
  Connection getConnection(
    String user, String password);
}
```

Adatforrás használata -(1)

```
public class DAO {
  protected DataSource datasource;
  public DAO(){
    try {
       Context ctx = new InitialContext();
       if( ctx == null )
          throw new RuntimeException
          ("JNDI Context coluld not be found");
       datasource =  (DataSource)
                ctx.lookup("jdbc/distedudb");

    }
    catch (Exception e) {
            e.printStackTrace();
    }
  }
}
```

Adatforrás használata -(2)

```
public class CourseDAO extends DAO{
    public CourseDAO(){
        super();
    }
    public List<Course> getCourses(){
        ...
        Connection con=null;
        PreparedStatement ps = null;
        ResultSet rs=null;
        try{
            con=datasource.getConnection();
            ps = con.prepareStatement(GET_ALL_COURSES)
            ...
        }
        ...
    }
}
```