Politechnika Świętokrzyska	
Wydział Elektrotechniki, Automatyki i Informatyki	
Technologie obiektowe - projekt	
Porównianie Spring Data framework oraz biblioteki Room	
1ID21A	Katarzyna Dawiec

1. Wstęp teoretyczny

Biblioteka Room

Baza danych SQLite jest dostarczona domyślnie z systemem Android i korzysta z niej większość urządzeń mobilnych. Do zarządzania można użyć API SQLite, (którego używanie jest skomplikowane) lub skorzystać z biblioteki Room. Biblioteka ta, zapewnia warstwę abstrakcyjną nad SQLite aby umożliwić bardziej sprawny dostęp do bazy danych przy zachowaniu pełnej jej funkcjonalności.
Używana jest do wykonywania operacji na lokalnej bazie danych.
Umożliwia obsługę podstawowych operacji oraz bardziej złożonych za pomocą własnego zapytania.

Posiada trzy główne komponenty:

- Klasa bazy danych – zawiera bazę danych oraz będąca punktem dostępowym do danych aplikacji,
- Klasy encji – reprezentują tabele w bazie danych,
- Obiekty dostępu do bazy danych (ang. *Data Access Object -DAO*) – zapewniają metody umożliwiające wykonywanie zapytań do bazy danych.

Framework Spring Data

Jest to framework bazujący na JPA (ang. *Java Persistence Api - JPA*). Stanowi warstwę abstrakcji nad JPA oraz Hibernate. Zapewnia spójne podejście dostępu do danych relacyjnych, nierelacyjnych, itp.

Jest narzędziem, które na podstawie samej nazwy metody potrafi zbudować zapytanie i obsłużyć zwracane dane. W przypadku, gdy wymagana jest obsługa specyficznej funkcjonalności, framework również umożliwia stworzenie własnego zapytania za pomocą JPQL (ang. *Java Persistence Query Language – JPQL*).

Korzystając ze Spring Data, programista nie potrzebuje implementować obiektów DAO – wystarczy je tylko zdefiniować. Dzięki temu sam framework będzie w stanie znaleźć odpowiedni interfejs i automatycznie stworzyć dla niego implementację.

2. Tworzenie tabel

- Połączenie z baza danych
    - Spring Data
        - Tworzenie bazy danych

Bazę danych utworzono za pomocą panelu phpmyadmin.

Następnie utworzono plik application.properties

```
server.port=8888

spring.datasource.url=jdbc:mariadb://localhost:3306/springlearningdb
spring.datasource.username=root
spring.datasource.password=
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
#spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.hibernate.ddl-auto=create-drop

hibernate.show_sql=true
```

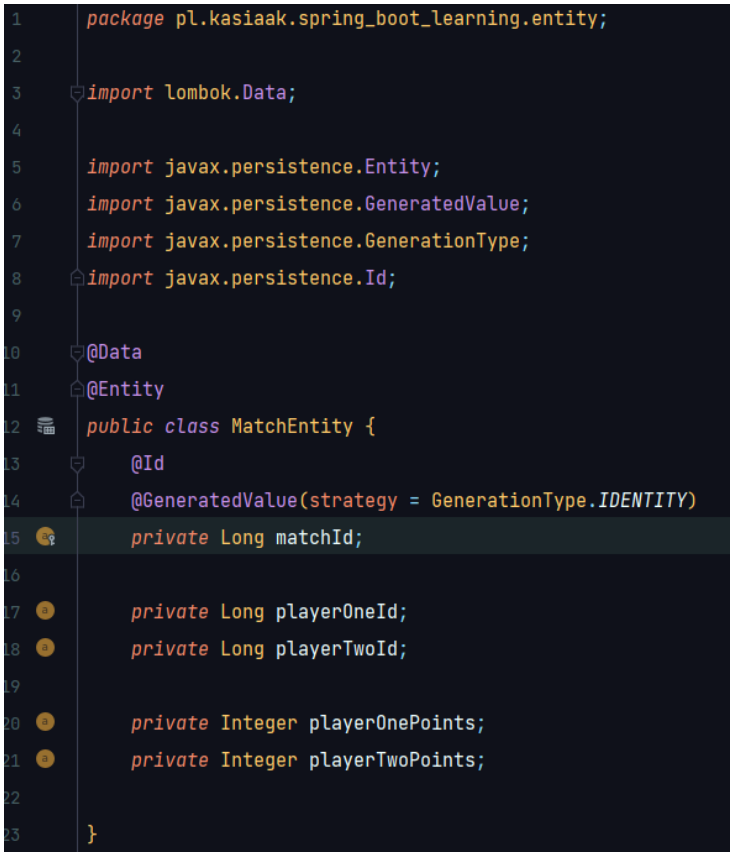
Kolejnym krokiem było utworzenie klasy umożliwiającej uruchamianie aplikacji.

```java
package pl.kasiaak.spring_boot_learning;

import ...

@SpringBootApplication
public class SpringBootLearningApplication {

    public static void main(String[] args) { SpringApplication.run(SpringBootLearningApplication.class, args); }

}
```

- Biblioteka Room
  - Tworzenie bazy danych
    W przeciwieństwie do frameworku Spring Data, tworzenie bazy danych odbywa się z poziomu kodu.

```java
@Database(entities = {Player.class, MatchEntity.class, Tournament.class}, version = 1, exportSchema = false)
public abstract class RoomComparisonDatabase  extends RoomDatabase {

    public abstract MatchDao matchDao();

    private static volatile RoomComparisonDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 4;
    public static final ExecutorService databaseWriteExecutor = Executors.newFixedThreadPool(NUMBER_OF_THREADS);

    public static RoomComparisonDatabase getDatabase(final Context context) {
        if(INSTANCE == null) {
            synchronized (RoomComparisonDatabase.class) {
                if(INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(), RoomComparisonDatabase.class,  name: "room_comparison_db")
                            .addCallback(sRoomComparisonDatabaseCallback)
                            .build();
                }
            }
        }
        return INSTANCE;
    }

    private static RoomDatabase.Callback sRoomComparisonDatabaseCallback = onCreate(db) -> {
            super.onCreate(db);

            databaseWriteExecutor.execute(() -> {
                MatchDao matchDao = INSTANCE.matchDao();
            });
    };
}
```

- Tworzenie tabel w bazie danych
    - Spring Data
        - Tabela MatchEntity



```java
package pl.kasiaak.spring_boot_learning.entity;

import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@Entity
public class MatchEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long matchId;

    private Long playerOneId;
    private Long playerTwoId;

    private Integer playerOnePoints;
    private Integer playerTwoPoints;

}
```

        - Tabela Player

```java
package pl.kasiaak.spring_boot_learning.entity;

import ...

@Data
@Entity
public class Player {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long playerId;

    private String name;
}
```

- Tabela Tournament

```java
package pl.kasiaak.spring_boot_learning.entity;

import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@Entity
public class Tournament {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long tournamentId;

    private String tournamentName;
    private String gameType;
}
```

- Biblioteka Room
  - Tabela MatchEntity

```java
package com.example.roomcomparison.entity;

import ...

@Entity(tableName = "MatchEntity")
public class MatchEntity {
    @PrimaryKey
    @NonNull
    public String matchUUID;

    @ColumnInfo(name = "player_one_id")
    public String player1Id;

    @ColumnInfo(name = "player_two_id")
    public String player2Id;

    @ColumnInfo(name = "player_one_points")
    public int player1Points;

    @ColumnInfo(name = "player_two_points")
    public int player2Points;

}
```

- Tabela Player

```java
package com.example.roomcomparison.entity;

import ...

@Entity
public class Player {
    @PrimaryKey
    @NonNull
    public String playerUUID;

    @ColumnInfo(name = "player_name")
    public String playerName;
}
```

- Tabela Tournament

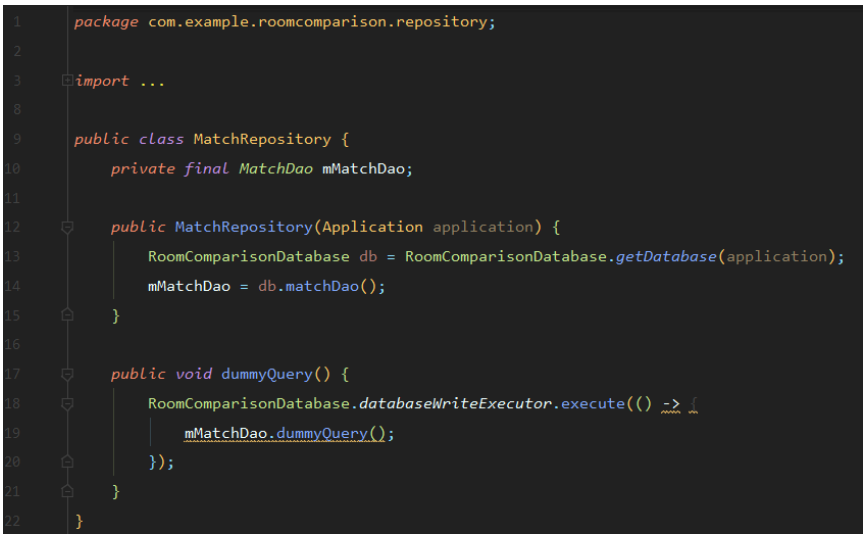
```java
package com.example.roomcomparison.entity;

import ...


@Entity(tableName = "Tournament")
public class Tournament {
    @PrimaryKey
    @NonNull
    public String tournamentUUID;

    @ColumnInfo(name = "tournament_name")
    public String tournamentName;

    @ColumnInfo(name = "game_mode")
    public String gameMode;

}
```

Aby wyświetlić utworzone tabele w „Database Inspector” należało również dla jednej z tabel (w tym przypadku dla tabeli Match) utworzyć klasy DAO, Repository oraz ViewModel. W celu uruchomienia aplikacji konieczna również była klasa MainActivity.

- Klasa MatchDao

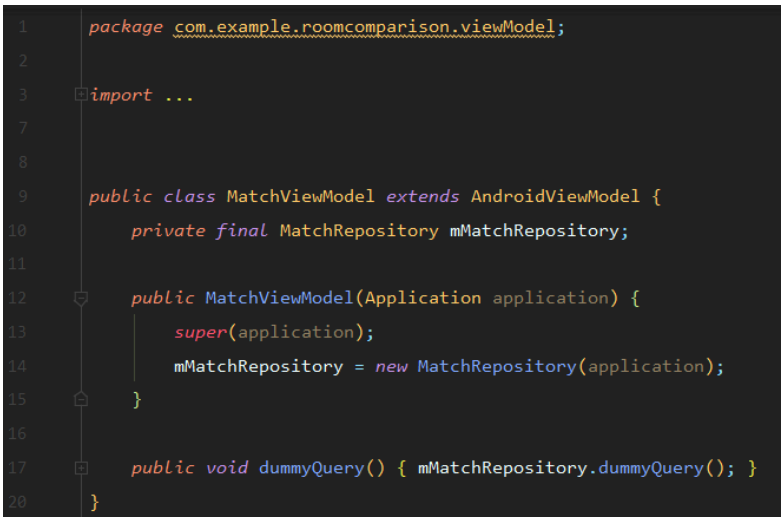
```java
package com.example.roomcomparison.dao;

import ...


@Dao
public interface MatchDao {
    @Query("SELECT 1")
    String dummyQuery();
}
```

- Klasa MatchRepository



```
package com.example.roomcomparison.repository;

import ...

public class MatchRepository {
    private final MatchDao mMatchDao;

    public MatchRepository(Application application) {
        RoomComparisonDatabase db = RoomComparisonDatabase.getDatabase(application);
        mMatchDao = db.matchDao();
    }

    public void dummyQuery() {
        RoomComparisonDatabase.databaseWriteExecutor.execute(() -> {
            mMatchDao.dummyQuery();
        });
    }
}
```

- Klasa MatchViewModel



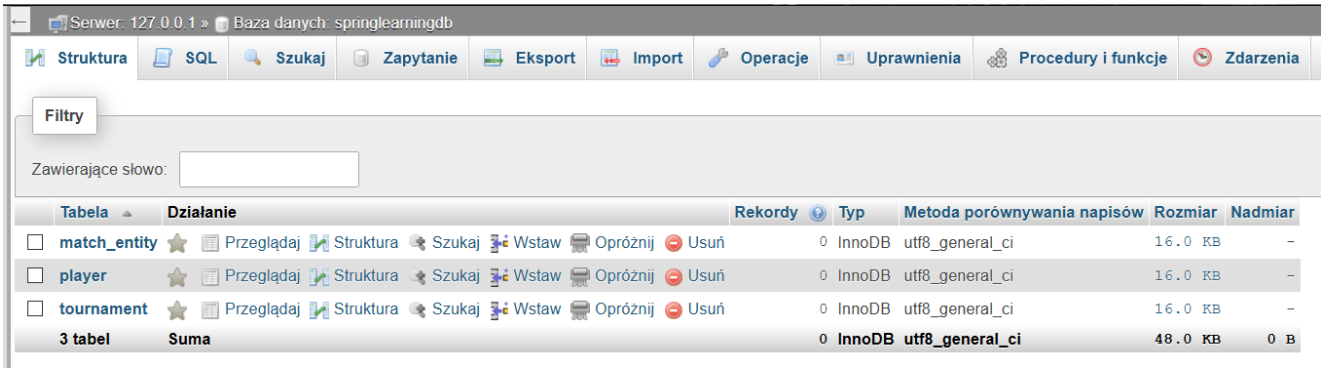
```
package com.example.roomcomparison.viewModel;

import ...


public class MatchViewModel extends AndroidViewModel {
    private final MatchRepository mMatchRepository;

    public MatchViewModel(Application application) {
        super(application);
        mMatchRepository = new MatchRepository(application);
    }

    public void dummyQuery() { mMatchRepository.dummyQuery(); }
}
```

- Klasa MainActivity

```java
package com.example.roomcomparison;

import ...

public class MainActivity extends AppCompatActivity {

    MatchViewModel mMatchViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mMatchViewModel = new ViewModelProvider( owner: this).get(MatchViewModel.class);
        mMatchViewModel.dummyQuery();
    }
}
```

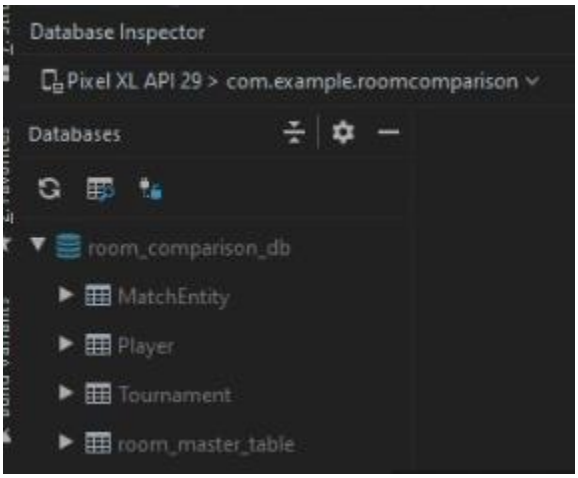
- Uruchomienie aplikacji
    - Spring Data

      Uruchomienie aplikacji spowoduje utworzenie wcześniej określonych tabel w bazie danych. Będą widoczne z poziomu panelu phpmyadmin



- Biblioteka Room

Utworzone tabele po uruchomieniu aplikacji można zobaczyć z poziomu wbudowanego w Android Studio Database Inspector.



W porównaniu do frameworku Spring Data, uruchomienie bez zdefiniowanej chociaż jednej klasy DAO, Repository, ViewModel dla tabeli oraz MainActivity spowoduje, że w przypadku uruchomienia Database Inspector po pomyślnym uruchomieniu aplikacji, „utworzone” przez użytkownika tabele nie pojawią się. Jest to spowodowane tym, że baza danych do utworzenia tabel wymaga dodatkowo wykonania jakiejś operacji. Dlatego konieczne było stworzenie wyżej wymienionych klas dla tabeli MatchEntity na tak wczesnym etapie.

3. Tworzenie relacji

- Relacja wiele do wielu
    - Spring Data
        - Tabela mecz

```
@ManyToMany(mappedBy = "matchParticipation")
List<Player> playersAssignedToMatch;
```

        - Tabela gracz

```
@ManyToMany
@JoinTable(
        name = "tournament_participation",
        joinColumns = @JoinColumn(name = "playerId"),
        inverseJoinColumns = @JoinColumn(name = "tournamentId"))
List<Tournament> tournamentParticipation;

@ManyToMany
@JoinTable(
        name = "match_participation",
        joinColumns = @JoinColumn(name = "playerId"),
        inverseJoinColumns = @JoinColumn(name = "matchId"))
List<MatchEntity> matchParticipation;
```

        - Tabela turniej

```
@ManyToMany(mappedBy = "tournamentParticipation")
List<Player> playersAssignedToTournament;
```

- Biblioteka Room
  - Tabela pośrednia obsługująca relację mecz – gracz

```
@Entity(primaryKeys = {"playerUUID", "matchUUID"})
public class MatchPlayerCrossRefEntity {
    @NonNull
    public String playerUUID;
    @NonNull
    public String matchUUID;
}
```

  - Klasa obsługująca relację miedzy tabelami mecz – gracz -> „mecz z graczami”

```
public class MatchWithPlayers {
    @Embedded
    public MatchEntity match;
    @Relation(
            parentColumn = "matchUUID",
            entityColumn = "playerUUID",
            associateBy = @Junction(MatchPlayerCrossRefEntity.class)
    )
    public List<Player> players;
}
```

  - Tabela pośrednia obsługująca relację turniej – gracz

```
@Entity(primaryKeys = {"tournamentUUID", "playerUUID"})
public class TournamentPlayerCrossRefEntity {
    @NonNull
    public String tournamentUUID;
    @NonNull
    public String playerUUID;
}
```

- Klasa obsługująca relację między tabelami turniej – gracz -> „turnieje z graczami”

```
public class TournamentWithPlayers {
    @Embedded public Tournament tournament;
    @Relation(
            parentColumn = "tournamentUUID",
            entityColumn = "playerUUID",
            associateBy = @Junction(TournamentPlayerCrossRefEntity.class)
    )
    public List<Player> listOfTournamentsWithPlayers;
}
```

- Klasa obsługująca relację między tabelami gracz – turniej -> „gracze z turniejami”

```
public class PlayerWithTournament {
    @Embedded public Player player;
    @Relation(
            parentColumn = "playerId",
            entityColumn = "tournamentId",
            associateBy = @Junction(TournamentPlayerCrossRefEntity.class)
    )
    public List<Tournament> listOfPlayersWithTournaments;
}
```

- Relacja jeden do wielu
  - Spring Data
    - Tabela turniej – relacja “mecze z turniejami”

```
@OneToMany(targetEntity = MatchEntity.class)
List<MatchEntity> matchesInTournament;
```

- Biblioteka Room
  W przypadku relacji „jeden do wielu”
  - Klasa obsługująca relację między tabelami turniej – mecz -> „turnieje z meczami”

```
public class TournamentWithMatches {
    @Embedded public Tournament tournament;
    @Relation(
            parentColumn = "tournamentUUID",
            entityColumn = "tournamentUUIDFK"
    )
    public List<MatchEntity> matchList;
}
```