Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Aplikace s databází omezených tanečních figur WDSF (World Dance Sport Federation)

s generátorem tanečních sestav

2020

Jakub Komárek

Vedoucí práce: Mgr. Radek Janoštík

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Jakub Komárek
Název práce:	Aplikace s databází omezených tanečních figur WDSF (World Dance Sport Federation) (s generátorem tanečních sestav)
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Radek Janoštík
Počet stran:	43
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Jakub Komárek
Title:	An aplication with an implemented database of basic figures recognized by WDSF (World Dance Sport Federation) (with a dance choreography generator)
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Radek Janoštík
Page count:	43
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Smyslem mojí práce je vytvořit aplikaci pro jednoduché a přehledné prohlížení figur, které by bylo prakticky využitelné v tanečním světě a navíc by nabízela možnost nechat si vygenerovat taneční sestavy.

Synopsis

The aim of my thesis is to create an application for easy and organized dance figure browsing, which would be useful in dance world and moreover it would provide a tool for generating dance choreographies.

Klíčová slova: server; Java; Android; React; tanec

Keywords: server; Java; Android; React; dance

Děkuji Mgr. Radkovi Janoštíkovi za příkladné vedení mé práce a za trpělivost, kterou se mnou musel mít.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **8**

2 Použité technologie **9**
2.1 Programovací jazyky . . . 9
2.1.1 Java . . . 9
2.1.2 JavaScript . . . 10
2.1.3 HTML a CSS . . . 10
2.2 Frameworky a knihovny . . . 10
2.2.1 Spring . . . 10
2.2.1.1 Spring Boot . . . 10
2.2.1.2 Spring Data JPA . . . 11
2.2.2 ReactJS . . . 11
2.2.3 React Native . . . 11

3 Server **13**
3.1 Kontrolery . . . 13
3.1.1 REST . . . 15
3.2 Entity . . . 16
3.3 Repozitáře . . . 17

4 Webová a Android aplikace **19**
4.1 Webová aplikace . . . 19
4.1.1 Třída Body . . . 21
4.1.1.1 Metoda renderH . . . 21
4.1.1.2 Metoda handleClick . . . 22
4.1.1.3 Metoda render. . . . 23
4.1.2 Třída Main . . . 25
4.1.2.1 Lifecycle metody . . . 25
4.1.2.2 Metoda fetchData . . . 26
4.1.2.3 Metoda filterName . . . 26
4.1.3 Třída Auth . . . 27
4.1.4 Třída Generator . . . 27
4.2 Android aplikace . . . 28
4.2.1 Třída App . . . 28
4.2.2 Třída Main . . . 29
4.2.3 Třída Generator . . . 30

5 Generátory **30**
5.1 Abstraktní třída Generator . . . 30
5.1.1 Vstupní metoda generate . . . 30
5.1.2 Metoda publicSelect . . . 30
5.1.3 Metoda checkFoot . . . 31
5.1.4 Metoda changeFoot . . . 31

5.2 Generátor Anglického Waltzu 32
5.2.1 Hlavní metoda třídy 33
5.2.2 Hledání možných navazujících figur 34
5.2.3 Vybrání figury 36
5.3 Generátor Samby 37

6 Plány do budoucna **38**

Závěr **39**

Conclusions **40**

Literatura **41**

A Obsah přiloženého CD/DVD **43**

Seznam obrázků

1 Java Virtual Machine 9
2 Jednoduché znázornění architektury REST. 15
3 Ukázka mého webového rozhraní. 19

Seznam zdrojových kódů

1 Ukázka kódu v ReactJs. 12
2 Úvodní třída Application. 13
3 Ukázka kontroleru se Spring anotacemi. 14
4 Ukázka entity ve Spring frameworku. 16
5 Ukázka použití JpaRepository. 18
6 Zakomponování komponent do HTML. 21
7 Příklad metody renderující jednotlivé položky menu nalevo. 22
8 Část metody handleClick. 23
9 Část vykreslující metody render. 24
10 Lifecycle metoda. 25
11 Ukázka použití Fetch API. 26
12 Filtrování figur podle názvu. 27
13 Třída App. 29
14 Metoda checkFoot. 31
15 Metoda changeFoot. 32
16 Konstanty ve třídě SlowWaltzGenerator. 32
17 Strategie pro výjimečné stavy. 33
18 Metoda findNextPossibleFigures. 34
19 Metoda checkDirection. 35
20 Metoda afterSelection. 37

1 Úvod

Čas jsou peníze. V tanečním světě bych to ale trochu poupravil a řekl čas jsou výsledky. Každý taneční pár má nějaký domovský klub, kam chodí trénovat, ať už na skupinové lekce, sami, nebo s trenérem, což je ten nejdůležitější trénink, a právě na trénink s trenérem bych se chtěl zaměřit. Na tomto tréninku se nejdříve musí sestavit taneční sestava, kterou bude pár tančit na soutěžích a poté, na dalších lekcích, se zlepšuje provedení figur v sestavě.

V současné době musí trenér buďto figury a jejich rozdělení podle tříd znát zpaměti, nebo složitě listovat v docela rozsáhlých tištěných provedeních, což bere čas jinak využitelný na trénink. Za každou individuální lekci se platí a takové vyhledávání tedy nepřímo stojí peníze.

Také na tanečních soutěžích, kdy odborný dozor musí hlídat, jestli tanečníci netační figury z vyšších tříd, než kterou mají, je problémem příliš mnoho figur a tam vůbec není čas na listování papíry.

Základní termíny:

- WDSF (World Dance Sport Federation) je mezinárodní vůdčí tělo tanečního sportu a tanečního sportu pro invalidy, uznané Mezinárodním olimpijským výborem a Mezinárodním paralympijským výborem.
- ČSTS (Český svaz tanečního sportu) je občanské sdružení tanečníků a tanečních funkcionářů z České republiky.
- Taneční sestava je posloupnost figur o volitelné délce, které na sebe musí vhodně navazovat (stojná noha, pozice partner/partnerka, směr pohybu).
- Délka sestavy může být určena počtem taktů nebo figur.
- Takt je část skladby, za kterou se provede jistý počet dob, například ve 3/4 taktu jsou tři doby.
- Taneční krok je přenesení váhy z jedné nohy na druhou s výsledkem pohybu. Většinou je jeden krok na dobu, ale občas se doba rozdělí a tančí se vícero kroků v jedné době, nebo naopak.
- Povinné figury jsou v generování sestav figury, které musí být obsaženy v sestavě.

Kdybych to měl shrnout, pomocí mé aplikace chci snížit čas na trénincích a soutěžích strávený listováním v papírech a dát tanečnímu světu nástroj pro rychlé generování sestav, ať už cvičných, nebo soutěžních.

2 Použité technologie

2.1 Programovací jazyky

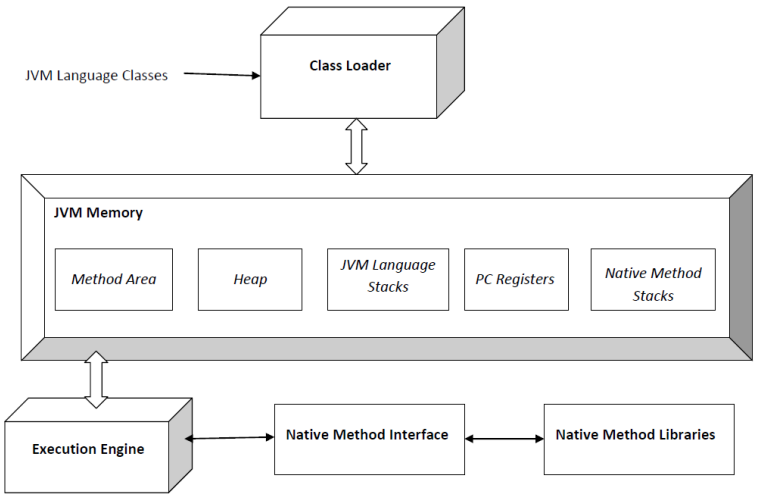
Při výběru techlogií, pomocí kterých budu mou práci vytvářet, jsem se velmi rychle rozhodl pro Javu, se kterou se mi už od začátku, co jsem jazyk poznal, pracovalo velmi dobře.

V průběhu práce mi došlo, že se vytváření webu pomocí HTML a CSS chci co nejvíce vyhnout, v čemž mi perfektně pomohl ReactJS, pomocí kterého jsem se vyhnul naprosté většině HTML. Odnož Reactu (React Native) se navíc dá použít pro vývoj aplikací pro Android, čehož jsem také využil.

2.1.1 Java

Java je obecně použitelný objektově orientovaný programovací jazyk. Jednou z hlavních myšlenek Javy je, aby programátoři kód napsali jednou a rozběhli ho kdekoliv (write once, run anywhere), znamenající, že zkompilovaný Java kód může běžet na jakékoliv platformě, podporující Javu, bez nutnosti rekompilace. Je to díky tomu, že Java se nepřekládá do strojového kódu, ale do tzv. java bajtkódu, který může být spuštěn na jakémkoliv JVM (Java Virtual Machine). Java je svou syntaxí velmi podobná jazykům C, C++ a C#, byla vyvinuta Jamesem Goslingem v Sun Microsystems a byla vydána v roce 1995.

Java Virtual Machine je virtuální počítač definovaný specifikacemi. Použitý garbage-collector algoritmus, popřípadě jakékoliv vnitřní optimalizace JVM nejsou specifikovány. Je to proto, aby implementátoři nebyli zbytečně omezováni. Jakákoliv Java aplikace může běžet pouze na nějaké konkrétní implementaci těchto abstraktních specifikací JVM. Graficky znázorněný JVM můžete vidět na obrázku 1.



Obrázek 1: Java Virtual Machine

Javu jsem si pro svou bakalářskou práci vybral proto, že je lehce pochopilná a mně osobně se v ní velmi dobře píše. Navíc, podle průzkumu Githubu, je Java jedna z nejpoužívanějších programovacích jazyků na světě, využívaná zejména na client-server webové aplikace, což se pro mou práci hodí. [3]

2.1.2 JavaScript

Javascript je programovací jazyk, syntakticky patřící do rodiny C/C++/Java, avšak sémanticky jde o zcela jiný jazyk, Java v názvu je pouze marketingový tah. Standardizovaná verze JavaScriptu se nazývá ECMAScript. Nejvíce se JavaScript využívá při vývoji webu, popřípadě webové aplikace.

JavaScript v mé aplikaci využívám hojně, a to hlavně knihovnu ReactJS, díky které jsem vytvořil webové i mobilní uživatelské rozhraní. [4]

2.1.3 HTML a CSS

I přes snahu vyhnout se jim bylo i v mé bakalářské práci nutno použít HTML s CSS, naštěstí však pouze na definici hlavičky dokumentu a úvodního tagu v těle.

2.2 Frameworky a knihovny

2.2.1 Spring

Spring je nejpopulárnější framework na vývoj aplikací pro Java EE. Miliony vývojářů kolem světa využívají Spring framework na vytvoření vysoce výkonných, lehce testovatelných a znovupoužitelných kódů. [5] Největší výhody Spring frameworku jsou:

- Spring umožňuje vyvíjet enterprise-class aplikace s využitím POJO, není tedy nutné mít EJB (Enterprise JavaBean) prostředí.
- Velmi dobře využívá již rozšířené technologie, jako některé ORM, JEE, Quartz a další.
- Spring Web framework je velmi dobře napsaný webový MVC framework.
- Nabízí nám pohodlné API na předklad technologicky specifických výjimek (z JDBC, Hibernate, JDO,..) do konzistentních nekontrolovaných výjimek.

2.2.1.1 Spring Boot

Spring Boot je projekt postavený na vrcholu Spring frameworku. Zajišťuje vývojářům mnohem rychlejší a jednodušší cestu k založení, konfiguraci a rozběhnutí jak jednoduché, tak webové aplikace. Je to Spring modul zajišťující RAD (Rapid Application Development) rys do Spring frameworku. Zkráceně je Spring Boot složený z Springu a vložených serverů, bez nutnosti použít XML konfiguraci. Používá konvenci před konfigurací (convention over configuration) softwarové konstrukční paradigma. [6]

2.2.1.2 Spring Data JPA

Spring Data je na Springu založený programovací model pro přístup k datům. Snižuje množství potřebného kódupro práci s databázemi a úložištěmi dat. Spring Data JPA zjednodušuje vývoj Spring aplikací které používají JPAtechnologii.

Se Spring Data definujeme rozhraní úložiště pro každou entity domény v aplikaci. Úložiště (repository) obsahuje metody na provádění CRUD (Create, Read, Update, Delete) operací. [8]

- ORM, Object-relational mapping, je programovací technika v softwarovém inženýrství pro konverzi dat mezi nekompatibilními typovými systémy používající objektově orientovanéprogramovací jazyky. Hlavním cílem ORM je synchronizace mezi používanými objekty v aplikaci a jejich reprezentací v databázovém systému tak, aby byla zajištěna perzistence dat. [18]
- Java Persistence API (JPA), je specifikace programovacího rozhraní Java aplikace, které popisuje management relačních dat v aplikacích používajících Java SE (Standard Edition) a Java EE (Enterprise Edition). [18]
- API, Application Programming Interface, je počítačové rozhraní, které definuje kominukaci mezi vícero softwarovými prostředníky. [16]

2.2.2 ReactJS

ReactJS (také React.js) je JavaScriptová knihovna pro vytvoření uživatelského rozhraní. Je vyvíjen hlavně společností Facebook a komunitou individuálních vývojářů a společností. React může být použit jako základ pro vytvoření single-page webu, popřípadě mobilní aplikace (Android i iOS). [7] Zprvu jsem doufal, že se verze z webu bude dát přenést i pro android, ale, ikdyž je to stejný jazyk, je to velmi rozdílné.

React může být založen na třídách (lze také využívat pouze funkce, bez tříd), které představují jednotlivé komponenty rozhraní, uchovávají si svůj vlastní vnitřní stav a jedna s druhou dokáží komunikovat pomocí metod. Komponenty Reactu mají hierarchickou strukturu, ve které rodič (vyšší komponenta) přímo vidí a může upravovat vnitřní stav potomka (nižší komponenta), druhým směrem to jde pouze přes odkazy na metody rodiče, které rodič posílá jako argumenty při konstrukci potomka.

S Reactem se mi pracovalo velmi dobře, tutoriál na oficiálních stránkách dá člověku více, než jen základní pohled do fungování a poté už je programování více méně intuitivní. Komunita okolo ReactJS je rozšířená a tak není problém se na ni obrátit, když člověk neví, kudy dál. Ukázku kódu napsaného v ReactJS můžete vidět v 2.2.3.

2.2.3 React Native

React Native je open-source framework pro mobilní aplikace vytvořený Facebookem. Je používaný pro vývoj Android, iOS, Web a UWP díky možnosti použít

React zároveň s nativními platformními možnostmi. [9]

```
1  import React, { Component } from 'react';
2  class App extends Component {
3    constructor(props) {
4      super(props);
5      this.state = { count: 1 };
6    }
7
8    render(){
9      return(
10       <p>Count: {this.state.count}</p>
11     );
12   }
13 }
```

Zdrojový kód 1: Ukázka kódu v ReactJs.

3 Server

Tzv. backend mé aplikace, prostředník mezi frontendem a databází s daty. Založen čistě na Javě a jejím frameworku Spring. Serverová část mé aplikace se skládá z entit, kontrolerů, repozitářů a generátorů, které popíšu ve vlastní kapitole, plus jedna nezařená třída Application, přes kterou se vše spouští a kterou můžete vidět v ukázce 3.

```
1 @SpringBootApplication
2 public class Application {
3
4    public static void main(String[] args) {
5       SpringApplication.run(Application.class, args);
6    }
7
8 }
```

Zdrojový kód 2: Úvodní třída Application.

(1) Meta-anotace která spojuje průzkum komponent, autokonfiguraci a majetkovou podporu (property support). [6]

(2) Název třídy.

(4) Název metody.

(5) Volání, kterým spustíme celou aplikaci.

3.1 Kontrolery

Kontrolery jsou třídy, ve kterých se spojují všechny části serveru, tj. generátory a repozitáře, obecně modely, a vstupní/výstupní body. V mé práci najdete tři kontrolery, FigureController, jehož část kódu je ukázaná v 3.1, obsluhující požadavky na figury, UserController obsluhující uživatele a GeneratorController, zajišťujicí generování tanečních sestav.

```
1  @RestController
2  @CrossOrigin(origins="*")
3  public class FigureController {
4     @Autowired
5     private UserFigureRepository userFigureRepository;
6     @Autowired
7     private FigureRepository figureRepository;
8
9     @GetMapping(path="/All")
10    public Iterable<Figure> getAllFiguresOrderById(@RequestParam(
          required = false) String user) {
11       List<Figure> result = figureRepository.findByOrderByIdAsc();
12       if(user != null)
13          result.addAll(Figure.userFigureToFigure(
               userFigureRepository.findByUser(user)));
14       return result;
15    }
16 }
```

Zdrojový kód 3: Ukázka kontroleru se Spring anotacemi.

(1) @RestController anotace byla představena pro zjednodušení vývoj RESTful webových služeb. Spojuje dohromady @Controller a @ResponseBody, což eliminuje nutnost anotovat každou metodu obsluhující požadavky s @ResponseBody. [10]

(2) @CrossOrigin nám umožňuje sdílení prostředků z různých zdrojů. [11] Bez ní bychom nemohli využívat například Fetch API v ReactJS.

(3) Název třídy, tento kontroler obsluhuje požadavky na vše, co se týká figur.

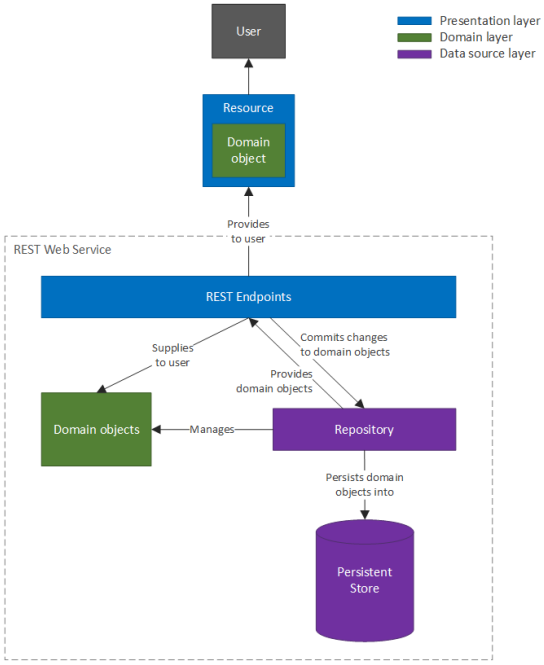
(4)-(7) @Autowired anotace nám dává přesnou kontrolu kde a jak by mělo automatické propojování komponent probíhat. [13]

(9) @RequestMapping je anotace používaná na mapování webových požadavků na metody Spring kontroleru. GetMapping je zjednodušená RequestMapping (method = RequestMethod.GET). [14]

(10)-(15) Metoda pro získání dat z databáze a vrácení výsledku, díky anotaci, namapovaného do JSON formátu. @RequestParam nám říká, jaké parametry by měly být v požadavku dodány, required = false nám říká, že jsou nepoviné.

3.1.1 REST

REpresentational State Trasnfer (REST), softwarový architektonický styl který definuje set omezení pro použití při vytváření webových služeb. Webové služby, které se přizpůsobují REST architektonickému stylu, nazývající se RESTful služby, nabízejí interoperabilitu mezi počítačovými systémy na internetu. REST byl navržen Royem Fieldingem v jeho disertační práci v roce 2000. V REST architektuře, klienti posílají požadavky na získání nebo modifikaci dat a server posílá odpovědi na takové požadavky.Máme 4 základní HTTP slovesa, která používáme v požadavcích na komunikaci se zdroji v REST systému. [15] Architekturu REST můžete vidět znázorněnou na obrázku 2.



Obrázek 2: Jednoduché znázornění architektury REST.

- GET metoda požaduje reprezentaci určitých zdrojů/dat. Požadavky používající tuto metodu by měly pouze získávat data. [12]
- POST se používá pro odeslání entit do specifikovaného zdroje, často způsobující změnu stavu nebo jiné postranní efekty. [12]
- PUT nahradí všechny aktuální reprezentace cíleného zdroje payloadem požadavku. [12]
- DELETE smaže specifikovaný zdroj. [12]

GET se považuje za bezpečnou metodu, nedochází při ní k modifikaci dat, je idempotentní a cachovatelná. Idempotentní znamená, že stejný požadavek,

provedený jednou, či vícekrát za sebou, má stejný účinek a nechává server ve stejném stavu, jako předtím. Na rozdíl od GET je metoda POST datově nebezpečná a není idempotentní. PUT a DELETE nejsou bezpečné metody, dochází také k modifikaci dat, ale už jsou idempotentní.

3.2 Entity

Persistenční Entita je klasická třída Javy (POJO), ukázka v 3.2, jejíž stav je typicky perzistován k tabulce v relační databázi. Instance takovéto entity se shodují s řádky dané tabulky. Entity mívají obvykle vztahy k dalším entitám a tyto vztahy jsou vyjádřeny pomocí objektově/relačních metadat.Objektově/relační metadata mohou být vyjádřena pomocí anotací, nebo v separátním XML deskriptorovém souboru distribuováném společně s aplikací. [16]

- POJO, neboli Plain Old Java Object, je klasický Java objekt, nevázaný žádnými jinými speciálnimi restrikcemi než nucenými specifikacemi jazyka Java. POJO jsou používány pro jejich lehkou čitelnost a znovupoužitelnost v kódu. [17]

```
1  @Entity
2  @Table(name="figure")
3  public class Figure {
4     @Id
5     @GeneratedValue(strategy=GenerationType.AUTO)
6     @Column(name="ID")
7     private Integer id;
8
9     @Column(name="name")
10    private String name;
11
12    public static List<Figure> userFigureToFigure(List<UserFigure>
          list){
13       List<Figure> result = new ArrayList<>();
14       for(UserFigure uf : list) {
15          result.add(Figure.oneUFToF(uf));
16       }
17       return result;
18    }
19 }
```

Zdrojový kód 4: Ukázka entity ve Spring frameworku.

(1) Anotace označující třídu jako Entitu.

(2) V případě, že se název tabulky liší od názvu třídy, pomůže @Table, která Springu řekne, v jaké tabulce hledat.

(3)-(20) Název a tělo třídy.

(4) Anotace určující primární klíč tabulky, každá Entita by ho měla mít.

(5) Určuje, že primární klíč je automaticky alokován.

(6), (9) Udává název sloupce, ke kterému se proměnná váže.

(7), (10) Názvy proměnných instancí.

(12)-(18) Metoda třídy.

3.3 Repozitáře

Repozitáře zajišťují komunikaci s databází. Díky využití JpaRepository interface bylo pracování s databází velmi jednoduché.

JpaRepository rozšiřuje CRUDRepository a také PagingAndSortingRepository. Každé z nich definuje svou vlastní funkcionalitu:

- CRUDRepository zajišťuje CRUD operace. CRUD:
  - Create - vytváří záznamy v tabulce
  - Read - dokáže záznamy v tabulce přečíst
  - Update - pozměňuje záznamy v tabulce
  - Delete - maže záznamy z tabulky
- PagingAndSortingRepository se stará o stránkování a třídění záznamů.
- JpaRepository zajišťuje metody související s JPA jako například splachování (flushing) perzistentního kontextu a mazání záznamů ve várce.

Díky tomuto dědičnému vztahu JpaRepository obsahuje plnou API CRUDRepository a PagingAndSortingRepository. Když nepotřebujeme plnou funkcionalitu zajištěnou JpaRepository a PagingAndSortingRepository, můžeme použít základní CRUDRepository. [19] Jak se dá použít JpaRepository uvidíte v ukázce 3.3.

```
1 @Repository
2 public interface FigureRepository extends JpaRepository<Figure,
      Integer> {
3
4  List<Figure> findByDanceAndLeagueGreaterThanEqual(String dance,
       String league);
5
6  @Query(value = "SELECT DISTINCT f.league FROM figure f WHERE f.
       dance = ?1 ORDER BY LENGTH(f.league) DESC", nativeQuery = true)
7  List<String> findLeagues(String dance);
8 }
```

Zdrojový kód 5: Ukázka použití JpaRepository.

(1) Anotace @Repository se používá k naznačení, že třída poskytuje mechanismus pro ukládácí, vyhledávací, updatovací a mazací operace na objektech. Je to specializace @Component anotace, takže Spring repozitáře jsou automaticky detekovány Spring frameworkem přes classpath sken.

(2) Název rozhraní rozšiřující JpaRepository, ke kterému musíme zadat <T,ID>, tj. entitu znázorňující tabulku a formát primárního klíče, v mém případě Integer.

(3) Základní a jednoduché použití JpaRepository.

(5)-(6) Pro složitější operace si můžeme nadefinovat vlastní dotaz.

Informativní data do databáze jsem získal z veřejně dostupného souboru na stránkách WDSF a ČSTF, které jsou oba součástí přiloženého CD. Soubor z webu WDSF obsahuje popis figur, zatímco soubor od ČSTS obsahuje jejich rozdělení do tříd. Oba soubory jsou ve formátu PDF a jelikož jsou docela rozsáhlé a zadávat hodnoty ručně by mohlo trvat věčnost, vytvořil jsem si na to pomocný program, jehož zdrojový kód také najdete na CD. V této části bylo složité jen parsování WDSF souboru, protože v tomto oficiálním dokumentu nezachovali jednotné formátování a to mi přihodilo pár podmínek v kódu navíc.

Data pro generátory jsem však už musel zadávat sám ručně, protože jsem nevymyslel způsob, jak tuhle část automatizovat. Problém byl v tom lidsky si přeložit poznámky a poté z informací vytvořit pro generátor vhodné objekty.

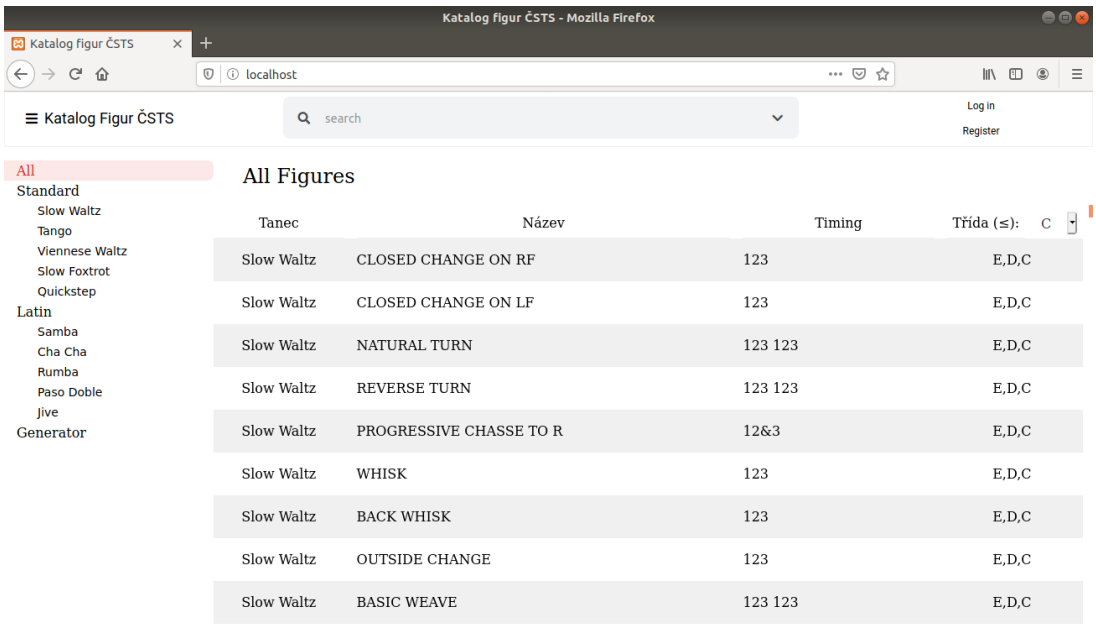
4 Webová a Android aplikace

4.1 Webová aplikace

Webové rozhraní u mé práce slouží hlavně k prohlížení dat a přidávání, upravování a odebírání vlastních figur. Ukázku rozhraní můžete vidět na obrázku 3. Hned na začátek této kapitoly bych rád uvedl důvod, proč je tohle webové rozhraní webová aplikace a ne webová stránka.

- Webové stránky jsou typicky informativní. Jejich hlavní úkol je sdělit informaci koncovému uživateli, ať už to je formou zpráv, jako CNN, nebo receptů. Jako obecné pravidlo platí, že je zde málo nebo žádná interakce na straně návštěvníka.
- Webové aplikace jsou obvykle zodpovědné za jistou formu interakce s návštěvníky. Zde již návštěvníci mohou s daty, co se jim dostávají, něco udělat. Skrze interakci s aplikací mohou poslat požadavek na jiný typ informací, nebo dokonce manipulovat s daty. [20]

Z tohoto pohledu je moje webové rozhraní webovou aplikací, jelikož uživatelé mohou data jak zobrazit, tak přidávat, sebou přidané data upravovat a mazat.



Obrázek 3: Ukázka mého webového rozhraní.

Jak jsem již psal výše, chtěl jsem se vyhnout psaní mnoha řádků HTML s CSS a to se mi povedlo díky JavaScriptu, konkrétně tedy knihovně ReactJS, ve kterém je napsané prakticky celé webové rozhraní.

Knihovna ReactJS poskytuje možnost objektově orientováného programování webového rozhraní.

React je knihovna pro vytváření skládajících uživatelských rozhraní. Podporuje vytváření znovupoužitelných UI (User Interface) komponent, které představují data, která se, během času, mění. Mnoho lidí používá React jako V v MVC architektuře. Knihovna nám bere dále od DOMu, nabízejíc nám jednodušší programovací model a lepší výkon. React se vykresluje na serveru užitím Node a může také napájet nativní aplikace použitím React Native [26].

- DOM - Document Object Model je multiplatformní a jazykově nezávislé rozhraní které bere XML nebo HTML dokument jako stromovou strukturu, ve které je každý uzel objekt reprezentující část dokumentu. DOM reprezentuje dokument logickým stromem, každá větev stromu končí uzlem a každý uzel obsahuje objekt. [21]
- HTML - Hypertext Markup Language, je standardní značkovací jazyk pro dokumenty, určený pro zobrazení ve webovém prohlížeči. Může být doprovázen technologiemi jako CSS a skriptovací jazyky, například JavaScript. [22]
- XML - Extensible Markup Language je, stejně jako HTML, značkovací jazyk, který definuje set pravidel na kódování dokumentů ve formátu, který je čitelný jak člověkem, tak počítačem. [23]
- CSS - Cascading Style Sheet je stylovací jazyk používaný pro popis prezentace dokumentu napsaném ve značkovacím jazyce, například HTML. CSS je jeden ze základních kamenů WWW společně s HTML a JavaScriptem. [24]
- WWW - World Wide Web, obecně známý jako web. [25]

Vlastnosti Reactu:

- JSX: je Javascriptové rozšíření syntaxe. Není nutné JSX využít při vyvýjení s použitím ReactJS, ale je doporučené.
- Komponenty: vše v Reactu se točí kolem komponent. Je důležité o všem přemýšlet jako komponentách, pomůže to později při udržování kódu, když se pracuje na projektech větších rozměrů.
- Jednosměrný proud dat a Flux: implementuje jednosměrný proud dat a Flux, což je vzor, který pomáhá udržovat data jednosměrná.
- Licence: je licencovaný pod Facebook Inc.

Výhody Reactu:

- Využívá virtuálního DOM, který je JavaScriptový objekt. Tento fakt vylepšuje výkon, vzhledem k tomu, že JavaScriptový virtuální DOM je rychlejší než ten obvyklý.
- Může být použitý na straně klienta, serveru a také s jinými frameworky
- Vzory komponent a dat zlepšují možnosti čtení, což napomáhá udržování větších aplikací.

JavaScriptovou část mám rozdělenou do několika tříd, které spolu vzájemně komunikují. Komponenty do HTML základu vložíme pomocí kódu, ukázané v 4.1.

```
1 ReactDOM.render(
2   <Body/>,
3   document.getElementById('root')
4 );
```

Zdrojový kód 6: Zakomponování komponent do HTML.

(1) Zde říkáme ReactDOMu (JavaScriptovému virtuálnímu DOMu), že má něco vykreslit. Co a kam záleží na argumentech.

(2) Prvním argumentem je komponenta, kterou budeme vykreslovat, v našem případě je to třída Body.

(3) Druhým argumentem je místo, kam komponentu budeme vykreslovat. Zde využijeme klasického JavaScriptu na nalezení příslušného uzlu.

4.1.1 Třída Body

Tato třída je základní část rozhraní. Vizuálně zajišťuje menu na levé straně a hlavičku na horní straně. Díky svému vnitřnímu stavu, měnícímu se podle návštěvníkova rozhodnutí, určuje, jaká komponenta bude vykreslena do hlavního pole, na obrázku 3 vpravo dole. Celý kód je příliš dlouhý na to ho tu popisovat, ale popíšeme si nejdůležitější části.

4.1.1.1 Metoda renderH

Tato metoda je sice krátká, ale dostatečně dlouhá na ukázání základní práce s odkazovanými metodami. kód můžete vidět v 4.1.1.1.

```
1 renderH(arg, selected){
2   return(
3     <Aside
4       value={arg}
5       onClick={() => this.handleClick(arg)}
6       selected={selected}
7     />
8   );
9 }
```

Zdrojový kód 7: Příklad metody renderující jednotlivé položky menu nalevo.

(1) Název metody a její parametry.

(2) Návratová hodnota metody.

(3) Název jiné komponenty (potomka), jehož hodnotu vrací.

(4)-(6) Argumenty předané potomkovi.

(5) Nejzajímavější argument, důvod, proč tu tahle metoda je. Jednotlivé komponenty mohou různými způsoby měnit svůj vlastní stav a stav potomků, ale nemohou přímo měnit stav rodičovské komponenty. Proto tu předáváme odkaz na metodu, která se nachází v těle třídy Body a kterou můžeme vidět v 4.1.1.2. Tento argument neříká, že když klikneme na vrácenou část DOMu z Aside třídy, tak se provede metoda handleClick, pouze předáváme odkaz, který komponenta Aside využije po svém. Rezervované slovo 'this' označuje, že jde o metodu téhle třídy, bez něj by šlo o funkci mimo třídu.

4.1.1.2 Metoda handleClick

Metoda handleClick má za úkol zjistit, pomocí předaného argumentu, na kterou komponentu (nebo její část) bylo kliknuto a adekvátně změnit stav třídy Body. Je to méně zajímavá metoda, ale důležitá, pro další vývoj stavu komponenty.

Při používání setState však musíme mít v paměti, že setState se provádí asynchronně, což znamená, že se stav komponenty nemusí změnit okamžitě, většinou dokonce komponenta čeká, jestli se nenajde ještě další setState, aby změnila honoty najednou. Tenhle fakt mi dělal občas docela potíže, ale existují postupy, kterými se s tím dá vypořádat.

```
1  handleClick(arg){
2    if(arg === 'All'){
3      this.setState({event: '/All', all: true, value: 'All Figures'});
4    }
5    ...
6    var a = document.getElementsByClassName('selected');
7    if(a !== undefined && a.length != 0){
8      a[0].className = "";
9    }
10   document.getElementById(arg).className = "selected";
11   ...
12 }
```

Zdrojový kód 8: Část metody handleClick.

(1) Název metody s argumenty.

(2) Zjištění hodnoty parametru.

(3) setState metoda slouží k změně vnitřního stavu komponenty, je to metoda ze třídy Component, kterou třída Body rozšiřuje.

(6)-(10) Část kódu zajišťující změnu třídy DOM uzlů.

4.1.1.3 Metoda render.

A tím se dostáváme k poslední důležité části třídy Body, a to k metodě render. V metodě render se může skrývat menší logika komponenty, ikdyž se to, pro zachování přehlednosti, nedoporučuje, ale hlavně obsahuje return příkaz. Ovšem, jak uvidíve níže 4.1.1.3, je občas výhodnější a přehlednější si předpřipravit proměnnou a pak pouze vložit její hodnotu.

```
1  render(){
2    var a;
3    if(this.state.generator){
4      a = (<Generator/>);
5    } else if (this.state.auth) {
6      a = (<Auth name={this.state.name} value={this.state.value} userOk
           ={name => this.userOk(name)} />);
7    } else {
8      var src = this.state.event;
9      if(this.state.name !== ""){
10       if(src.search("\\?") === -1){
11         src = src + "?user=" + this.state.name;
12       }
13       else {
14         src = src + "&user=" + this.state.name;
15       }
16     }
17     a = (<Main value={this.state.value} src={src} all={this.state.
           all} filter={this.state.filter} />)
18   }
19   return(
20     ...
21       </aside>
22       {a}
23     </div>
24   ...
25 );}
```

Zdrojový kód 9: Část vykreslující metody render.

(1) Název metody.

(2) Proměnná, jejíž hodnotu později dosadíme do výsledného vykreslení a její velice nápadité jméno.

(3),(5),(7) Zjišťujeme, kterou komponentu chceme vyrenderovat. Komponenta Generátor nám dává přístup k generátoru a generování sestav, Auth zajišťuje přístup do uživatelské sekce, kde si můžeme hrát s daty a Main je hlavní zobrazovací komponenta.

(4),(6),(17) Odkaz na jinou komponentu, jaký jsme již viděli v 4.1.1.1.

(8)-(16) Úprava budoucího argumentu v závisloti na tom, zda-li je uživatel přihlášen.

(19)-(25) Return příkaz.

(22) Zakomponování předpřipravené, nápaditě pojmenované proměnné a.

Na řádcích 6 a 17 v 4.1.1.3 vidíme, stejně jako na řádku 4, odkaz na jinou komponentu, avšak tady se ukazuje krása propojenosti komponent Reactu. Zde můžete vidět předávání stavu komponenty Body do props (zkrácenina properties) potomka. Při změně stavu rodičovské komponenty, jak je ukázáno třeba v 4.1.1.2, se změní i tyto hodnoty a pokud na nich nějak závisí data, jež se zobrazují v komponentě Auth/Main, tak se potomek znovu vykreslí s novými daty. Je to velmi hezká ukázka komunikace jednoho potomka s jiným a propojenosti komponent v ReactJS.

4.1.2 Třída Main

Třída Main je komponenta pro zobrazení dat získáných pomocí Fetch API. Data získává ze serveru pomocí adresy předané komponentě jako argument (4.1.1.3) a má dva režimy zobrazení. První zobrazuje všechny figury ze skupiny dané bočním menu, druhý rozbrazuje detailnější info o figuře dle výběru.

4.1.2.1 Lifecycle metody

V ReactJS je hned několik Lifecycle metod, které vývojářům výrazně ulehčují práci s knihovnou. Zde jsou ty hlavní z nich.

- contructor, metoda, při které nastavíme základní stav komponenty a zajistíme propojení metod s komponenou. 4.1.2.1 Je zavolána Reactem pokaždé, kdy vytvoříme komponentu.
- componentDidMount je metoda, která se zavolá ihned po prvním vykreslení komponenty. Zde se umisťují příkazy, které potřebují, aby už byla komponenta zařazená do DOMu, jak je ukázáno v 4.1.2.1.
- shouldComponentUpdate vrací boolean hodnotu, jestli by se komponenta měla pokračovat se znovuvykreslováním nebo ne.
- componentDidUpdate je zavolána hned poté, co byla komponenta pozměněna v DOMu. Aby se předcházelo zacyklení, používají se parametry prevProps a prevState, které jsou pole předchozích hodnot a podle porovnání těchto hodnot s aktuálními lze zabránit zacyklení. [28]

```
1 componentDidMount(){
2   this.fetchData(this.props.src, ['data', 'dataAll']);
3 }
```

Zdrojový kód 10: Lifecycle metoda.

4.1.2.2 Metoda fetchData

Fetch API je jednoduché rozhraní pro získávání dat. Fetch je jednodušší na vytváření požadavků a ošetření odpovědí než starší XMLHttpRequest, které obvykle potřebuje další logiku (například na ošetření přesměrování). [27] Jak jednoduché je používat Fetch API můžete vidět v kódu 4.1.2.1.

```
1 fetchData(string, key){
2   fetch(address + string)
3     .then(response => this.checkResponse(response))
4     .then(response => response.json())
5     .then(data => key.map(k => this.setState({[k]: data})))
6   .catch(err => {});
7 }
```

Zdrojový kód 11: Ukázka použití Fetch API.

(1) Název metody s parametry.

(2) Zavolání funkce fetch s argumenty.

(3) Hodnotu, kterou dostaneme, předáme metodě, která ošetřuje výjimky a pokud výjimka nenastala, vrátí původní hodnotu.

(4) Data, která jsme získali převedeme do JSONu, pokud v něm už nejsou.

(5) Pomocí funkce map, kterou lze použít na pole hodnot uložíme data do stavu komponenty.

(6) Pojistka na zachycování výjimek, avšak nezachytí všechny, proto je zde i metoda na řádku 3.

4.1.2.3 Metoda filterName

Tato část ukazuje, jak jsem vyřešil filtrování figur podle názvu, což můžete vidět v 4.1.2.3. Filtrování podle třídy je velice podobné.

```
1  filterName(){
2    var text = this.props.filter;
3    if(!text){
4      text = "";
5    }
6    var newData = this.state.dataAll.filter(item => {
7      const itemData = item.name.toLowerCase();
8      const textData = text.toLowerCase();
9      return itemData.indexOf(textData) > -1;
10   });
11   this.setState({
12     data: newData,
13   });
14 }
```

Zdrojový kód 12: Filtrování figur podle názvu.

(1) Název metody.

(2) Do proměnné načteme stav komponenty, který se shoduje s hledaným výrazem.

(3)-(5) Ošetření hodnoty null.

(6)-(10) Filtrování figur pomocí funkce filter.

(6),(7) Převedení filtru i filtrovaného na malá písmena.

(8) Jednoduché vyhledávání podřetězce.

(11)-(13) Změna stavu komponenty, výměna všech figur za figury, které vyhovují filtru.

4.1.3 Třída Auth

Tato komponenta se stará o registraci nových a přihlášení již registrovaných uživatelů. Po příhlášení může uživatel přidávat, upravovat a mazat sebou přidané figury. Je to také jediná komponenta, kde využívám XMLHttpRequest, a to abych poslal požadavek metodou POST, čehož se mi s využitím Fetch API moc nedařilo.

4.1.4 Třída Generator

Poslední třídou je třída Generator, která má za účel vytvořit uživatelské rozhraní pro generování tanečních sestav. Uživatel by si měl povybírat hodnoty, podle kterých by chtěl sestavu vygenerovat a spustit generaci. Hodnoty však vybrat

ani nemusí, protože v serverové části jsou nastaveny defaultní hodnoty.

4.2 Android aplikace

React Native, je adaptace ReactJS pro Android a iOS. Celá moje aplikace pro Android je psaná čistě v React Native. Práce s React Native se od klasického React JS liší hlavně v:

- Zatímco ReactJS je nástroj pro vývoj webu, který můžeme přidat do webové stránky ve formě znovupoužitelných komponent, které mohou být integrovány s jinými technologiemi, React Native není nástroj, který přidáme do existující projektu, je to nástroj použitý na vytvoření aplikace od základu.
- Přibývá nám zde nutnost instalovat balíčky pomocí npm nebo Yarn.
- V React Native se při pozicování a upravování vzhledu prvků nepracuje přímo s CSS, ale s StyleSheet, která se inicializuje pomocí objektu. Při jednoduché StyleSheet může to může být jednoduchý objekt, který v sobě uchovává hodnoty, nebo to může být, jako v mém případě, objekt složený, který v sobě uchovává další objekty, které už v sobě mají hodnoty. Takovéto stupňování lze udělat několikanásobné.
- Názvy jednotlivých uzlů DOMu se zásadně liší.

Jen co založíte projekt, tak v kořenovém adresáři naleznete index.js, který funguje jako index.html, tedy přesměruje nás na první defaultní stránku, v tomhle případě tedy App.js.

4.2.1 Třída App

Komponenta App, ukázaná v 4.2.1, zajišťuje menu nepříč všemi pohledy, je to základní komponenta celé aplikace. Jako navigátor jsem zvolil DrawerNavigator, tedy panel vytahující se z levého boku, popřípadě pomocí tlačítka v Appbaru nahoře. Navigátor mi přišel přehledný a dobře se hodící pro mou aplikaci.

```
1  export default class App extends Component {
2    render(){
3      return (
4        <NavigationContainer>
5          <Drawer.Navigator initialRouteName="Home">
6            <Drawer.Screen name="Slow Waltz" component={SlowW} />
7        ...
8          </Drawer.Navigator>
9        </NavigationContainer>
10     );
11   }
12 }
13
14 function SlowW ({ navigation }) {
15   return(
16     <Main source={address + "/Dance?dance=Slow Waltz"} navigation=
         {navigation}/>
17   );
18 }
```

Zdrojový kód 13: Třída App.

(1) Název třídy.

(2) Název metody.

(3)-(10) Return výraz.

(4),(9) Tagy kontejneru obsahujícího navigaci.

(5),(8) Tagy kontejneru.

(6) Navigační prvek odkazující na konkrétní pohled.

(14) Název funkce.

(15)-(17) Return výraz obsahující volání komponenty Main. Navigation argument musíme uvést, abychom měli možnost dostat se k menu a tedy i na jiné pohledy.

4.2.2 Třída Main

Třída Main ze souboru Home.js má za úkol zobrazovat data, která přijme za pomocí Fetch API. Jelikož je mobilní zařízení obecně menší než monitor počítače, šlo tu hlavně o přehlednost a dostatečnou velikost, aby se dalo dobře orientovat mezi figurami. Je zde zase zasazen filtr, ke kterému lze přistoupit po kliknutí na vyhledávací ikonu v Appbaru. také je zde zasazen filtr pomocí třídy figur, což je

většinou ten hlavní, prakticky více využitelný filtr.

4.2.3 Třída Generator

Pohled Generator představuje rozhraní pro generování sestav. Jako ve webové verzi si zde můžeme zadat jednotlivé parametry, včetně figur, které se v sestavě mají objevit. Zde byl také kladen důraz na přehlednost a dostatečnou velikost jednotlivých prvků.

5 Generátory

Tato kapitola pojednává o algoritmu generování tanečních sestav. Nakonec jsem vytvořil generátory dva, pro Slow Waltz (Anglický Valčík) a Sambu. Tyto dva tance jsem vybral nejen proto, že jsou v pořadí první ve své skupině, ale také proto, že jsou dostatečně odlišné, a zatímco u Standardních tanců jsou všechny, z pohledu generátoru, zhruba na stejné úrovni obtížnosti, Samba je jeden z nejtěžších tanců pro generování, těžší už je jenom Paso Doble, které ale tančím teprve krátkou dobu a tak by nebylo v mých silách pro něj vytvořit generátor. Samba je oproti většině ostatních Latinskoamerických tanců těžší z důvodu rozdělení hudby a také pro svou povahu postupového tance. Hudba v Sambě je rozdělena po osmi taktech a hodně záleží na tom, aby při konci každého osmého skončila i figura.

5.1 Abstraktní třída Generator

Tato třída zajišťuje předka obou Generátorů. Definuje vstupní metodu, kterou volá GeneratorController. Kromě téhle definuje abstraktní metodu GenerateList, kterou generování pokračuje, a také několik menších metod využívaných oběma generátory, například na převedení názvů figur z pole na ArrayList, nebo změnu stojné nohy.

5.1.1 Vstupní metoda generate

Tato metoda byla doplněna až v pozdějších stádiích vývoje. Když zadáme nějaké množství figur, které chceme mít v sestavě, je zde šance, že generátor nestihne vyčerpat všechny nutné figury dříve, než překročí limit taktů/limit počtu figur. Proto je zde pojistka, která kromě prvního generování zkusí vygenerovat ještě tři další sestavy a vrátí první s odpovídající délkou, nebo nejkratší z nich. V případě nezadání povinných figur se tohle neprovádí.

5.1.2 Metoda publicSelect

Nenechte se zmást, metoda není public, ale protected, aby ji mohli použít rozšiřující třídy, tedy oba generátory, ale ne široké okolí. Je to vcelku nezajímavá

metoda, která pouze rozlišuje prioritu figur. Kolekce figur, kterou přijme, rozdělí podle jejich třídy a následně vybere jednu figuru ze skupiny s nejvyšší třídou.

5.1.3 Metoda checkFoot

Tato metoda, ukázaná v 5.1.3, zjišťuje, jestli se k sobě dvě figury hodí, co se týče stojné nohy.

```
1 protected boolean checkFoot(GenInfo gn, GenInfo actual){
2   if(actual.getFootFinish().contains("behind") && gn.getFootStart().
        contains("fwd")){
3     return false;
4   }
5   String foot = changeFoot(actual.getFootFinish());
6   return (gn.getFootStart().contains(foot));
7 }
```

Zdrojový kód 14: Metoda checkFoot.

(1) Název třídy, jako argumenty bere poslední figuru v sestavě a potencionální další figuru.

(2)-(4) Pokud volnou nohou (ta, která není stojná) skončíme za stojnou nohou, nemůžeme pokračovat vpřed.

(5) Ze stojné nohy poslední figury zjistíme, jakou nohou, popřípadě kam, můžeme pokračovat 5.1.4.

(6) Vrátíme hodnotu porovnání.

5.1.4 Metoda changeFoot

Tato krátká metoda zjistí, jakou nohou může budou figura začínat. Ukázku kódu vidíte v 5.1.4.

```
protected String changeFoot(String foot) {
  if (foot.contains("LF")) {
    foot = foot.replace("LF", "RF");
  } else {
    foot = foot.replace("RF", "LF");
  }
  foot = foot.replace(" closes", "");
  foot = foot.replace(" behind", "");
  return foot;
}
```

Zdrojový kód 15: Metoda changeFoot.

(1) Název třídy, jako argument bere řetězec nohy poslední figury.

(2)-(6) Změní stojnou nohu na nohu, kterou musíme vyrazit vpřed.

(7) Pokud aktuální figura skončila přísunem nohy k noze, můžeme vyrazit kterýmkoliv směrem, dopředu i dozadu.

(8) Stojnou nohu před volnou jsme řesili už v předchozí metodě 5.1.3, tady už ji jen odstraníme.

(9) Vrátíme řetězec označující vhodnou výchozí nohu, popřípadě i směr.

5.2 Generátor Anglického Waltzu

Tato třída rozšiřuje abstraktní třídu Generator a přidává dodatečné metody. Při sestavování sestavy pracuje třída se stupni, aby se dokázala orientovat v prostoru, takže hned na začátku třídy máme definované konstanty, jak je ukázané v 5.2. Pro praktičnost se nejedná o obvyklé rozmezí stupňů 0-360, ale -90-270, protože rozmezí -80-45 (původně -90-45) je v jedné řadě, zatímco 270-45 ne.

```
private static int LOWERFRONT = -80;
private static int UPPERFRONT = 45;
private static int LOWERBACK = 125;
private static int UPPERBACK = 225;
private static int[] STARTINGDIRECTION = new int[]{0, 45};
private static String dance = "Slow Waltz";
private String league;
private int multiplier = 1;
```

Zdrojový kód 16: Konstanty ve třídě SlowWaltzGenerator.

(1)-(2) Tyto konstanty určují rozmezí, do kterého se musíme dotočit, pokud končíme figuru posunem vpřed, nebo přísunem.

(3)-(4) Konstanty určující rozmezí pro figuru končící posunem vzad, nebo přísunem.

(5) Pole výchozích hodnot. Ve Slow Waltzu se obvykle začíná šikmo ke zdi ve směru tance.

(6) Tanec, který generujeme.

(7) Třída, pro kterou sestavu generujeme.

(8) Při dlouhých sestavách se po určitém množství figur znovu obnoví všechny výchozí figury.

5.2.1 Hlavní metoda třídy

Je volána uvozovací metodou z abstraktní třídy Generator. Je to rozsáhlá metoda a i přes mé snahy ji rozdělit do vícero menších celků stále zůstává velmi dlouhou, proto ji zde nemůžu ukázat celou.

Na začátku této metody získáme data z databáze pomocí GenInfoRepository. Získáme tak všechny figury a všechny možné počáteční figury. Z množiny počátečních figur jednu vybereme pomocí select a provedeme povýběrové úpravy proměnných pomocí afterSelect metody. Poté začne for cyklus, který běží tak dlouho, dokud nesplníme délku a nevyčerpáme povinné figury.

```
1  possible = findNextPossibleFigures(genInfo, direction, actual);
2  if (possible.size() == 0) {
3   possible = findNextPossibleFigures(repeating, direction, actual);
4   if (possible.size() == 0) {
5     genInfo = genInfoRepository.findByDanceAndLeagueGreaterThanEqual(
          dance, league);
6     genInfo.remove(actual);
7     repeating.clear();
8     possible = findNextPossibleFigures(genInfo, direction, actual);
9     if (possible.size() == 0) {
10      GenInfo fail = new GenInfo();
11      fail.setName("Oops, something went wrong! I wasn't able to find
            next figure.");
12      result.add(fail);
13      return result;
14     }
15   }
16 }
```

Zdrojový kód 17: Strategie pro výjimečné stavy.

Při každém průchodu for cyklem vyhledáváme možné navazující figury na naposledy vybranou figuru pomocí metody findNextPossibleFigures. Pokud nastane, že by metoda nenašla žádnou vhodnou figuru, je zde několik strategií pro nalezení další figury, jak je ukázáno v 5.2.1. První strategie spočívá v pokusu najít další vhodnou figuru v kolekci již použitých opakovatelných figur. Pokud nevyjde, další možností je pokusit se najít příští figuru ze všech možných figur, vyjma figury aktuální. Když ani tato strategie nevyjde, vložíme jako poslední figuru nový objekt, kterému zadáme pouze název, za který dosadíme chybovou hlášku, a vrátíme výsledek.

Po získání kolekce možných figur je předáme metodě select a pozměníme proměnné. Pokud určujeme délku podle počtu taktů, provedeme výpočet pomocí hodnoty timing vybrané figury.

Po tomto výpočtu, pokud je délka sestavy příliš dlouhá, obnoví kolekci figur získanou z databáze a pokračuje v dalším kole cyklu.

5.2.2 Hledání možných navazujících figur

Metoda findNextPossibleFigures slouží jako rozcestník pro další metody, které zjišťují, zda-li jsou v kolekci nějaké vyhovující figury. Metodu checkFoot jsme již viděli ve třídě Generator a metoda checkPosition pouze porovnává řetězce pozice konce aktuální a začátku potencionální figury. Metodu si můžete prohlédnou v 5.2.2.

```
1 private List<GenInfo> findNextPossibleFigures(List<GenInfo> list,
     int[] direction, GenInfo actual) {
2   List<GenInfo> possible = new ArrayList<>();
3   for (GenInfo gn : list) {
4     if (checkFoot(gn, actual) && checkPosition(gn, actual) &&
         checkDirection(direction, gn)) {
5       possible.add(gn);
6     }
7   }
8   return possible;
9 }
```

Zdrojový kód 18: Metoda findNextPossibleFigures.

(1) Název metody s parametry. Jako argumenty bere kolekci volných figur, aktuálně poslední figuru a směr, kam momentálně míříme.

(2) Inicializace výsledné proměnné, do které ukládáme vhodné kandidáty.

(3)-(7) Cyklus procházející celou kolekci.

(4) Otestování, jestli je figura vhodný kandidát.

(5) Pokud ano, přidáme ji do výsledné proměnné a tu po skončení cyklu vrátíme.

Poslední metoda, tj. checkDirection, zobrazená v kódu 5.2.2, je společně s metodou calculateDirection od ostatních odlišná, protože zde nejde pouze o to porovnat řetězce, ale jde o složitější výpočet. Hned na začátku metody checkDirection vypočítáme potencionální směr pomocí metody calculateDirection, která je však příliš dlouhá, abych zde předvedl kód a samostatné části jí by nemusely dávat smysl.

```
1  private boolean checkDirection(int[] direction, GenInfo gn){
2    int[] tmp = calculateDirection(gn, direction);
3    String finish = gn.getFootFinish(), start = gn.getFootStart();
4    if ((start.contains("fwd") && (LOWERBACK <= direction[0] &&
        direction[1] <= UPPERBACK)) ||
5       (start.contains("bwd") && (LOWERFRONT <= direction[0] &&
           direction[1] <= UPPERFRONT) && finish.contains("closes"))) {
6     return false;
7    } else
8    return ((LOWERFRONT <= tmp[0] && tmp[0] <= tmp[1] && tmp[1] <=
        UPPERFRONT) && !finish.contains("bwd")) ||
9     ((LOWERBACK <= tmp[0] && tmp[0] <= tmp[1] && tmp[1] <= UPPERBACK)
         && !finish.contains("fwd")) ||
10    (gn.getName().equals("Hover Corte") && (LOWERFRONT <= direction[0]
          && direction[1] <= UPPERFRONT));
11 }
```

Zdrojový kód 19: Metoda checkDirection.

(1) Název metody s parametry. Jako argumenty bere současný směr a potencionální figuru.

(2) Výpočet budoucího směru, kdybychom použili tuhle figuru.

(3) Zjištění počáteční a koncové nohy/směru.

(4)-(7) Opatření proti figurám končícím přísunem. Nechceme se posunem vpřed/vzad pohybovat tak moc proti tanečnímu směru.

(8)-(10) Vrátíme pravdivostní hodnotu, jestli figura splňuje podmínky, nebo ne.

Hned na začátku calculateDirection vypočítáme možný úhel sečtením nejnižších hodnot a nejvyšších, přičemž jejich pořadí v poli je pevně dané. Tím získáme relativně velký rozsah, kam bychom mohli dotančit. Také zjistíme, kam chceme dotančit, pokud končíme pohybem vpřed, musím skončit čelem dopředu a naopak. Ve zbytku metody už jen počítáme, jaký výřez o velikosti 45 stupňů z toho rozsahu vezmeme za koncový.

Pokud bychom se trefili a v rozsahu byl směr 0 stupňů pro pohyb vpřed, nebo 180 stupňů pro pohyb vzad, dostaneme se do ideálního stavu a a už pouze zjišťuje, zda může být střed oněch 45 stupňů na 0/180, nebo to je nějak posunuté.

Kdybychom se netrefili, což je častější případ, tak zjišťujeme, jestli se vůbec trefíme do výřezu zadaného statickými konstantami třídy. Tahle část je ještě rozdělená na část, kdy končíme figuru přísunem a kdy končíme pohybem vpřed/-vzad.

5.2.3 Vybrání figury

Když máme množinu možných navazujících figur, zbývá už jen jednu vybrat. V metodě select se nejdřív zkouší, jestli není jedna z povinných figur v kolekci možných, pokud ano, zavolá metodu publicSelect s kolekcí těchto povinných figur, pokud ne, zavolá ji s původní kolekcí, která ji byla předána jako parametr.

Kratší metoda pro upravení proměnných po selekci figury. Byla vytvořena pro zlepšení orientace v hlavní metodě a je zobrazena v 5.2.3.

```
1  private int[] afterSelection(GenInfo actual, List<GenInfo> genInfo,
       List<GenInfo> result, List<GenInfo> repeating, int[] direction){
2    genInfo.remove(actual);
3    if (actual.getRepeating()) {
4      repeating.add(actual);
5    }
6    result.add(actual);
7    if (genInfo.size() == 0) {
8      genInfo.addAll(genInfoRepository.
           findByDanceAndLeagueGreaterThanEqual(dance, league));
9      repeating.clear();
10   }
11   return calculateDirection(actual, direction);
12 }
```

Zdrojový kód 20: Metoda afterSelection.

(1) Název metody s parametry.

(2) Z kolekce možných figur odebereme právě vybranou.

(3)-(5) Pokud je možné figuru opakovat, přidáme ji do kolekce opakovatelných.

(6) Přidáme vybranou figuru do výsledku.

(7)-(10) Pokud už není na výběr žádná další figura, znovu je všechny načteme z databáze. Pročistíme opakovatelné.

(11) Vrátíme výsledý směr.

5.3 Generátor Samby

U samby je strategie generování trochu odlišná. Všechny figury jsou dělané tak, aby se pohybovaly po tanečním směru a jelikož je to tanec ze skupiny Latinsko-amerických tanců, je mnohem lehčí se nějak pootočit. Díky tomu jsem nemusel řešit složité směrování, o to víc jsem však musel rozšířit data v databázi, kam jsem musel přidat dodatečné sloupce.

Hlavní metoda generátoru je velmi podobná té u Slow Waltzu, tady však se nepracuje s kolekcemi v ArrayListu, ale v HashMap, a to z důvodu rozdělení Samby po osmi taktech. Klíči v HashMap jsou jednotlivé figury a hodnotami je délka v taktech.

Při vybírání vhodných navazujících figur ověřujeme oproti Slow Waltzu navíc délku tak, aby nepřesáhla právě osm taktů. Ostatní ověřování jsou stejné, nebo velmi podobné.

Vybírání správné navazující figury už je trochu složitější. Je preferováno, aby

za sebou byly figury patřící k sobě (třeba po zášvihu vlevo bude zášvih vpravo) a až poté se řeší další možnosti.

Protože figury mají v Sambě definováno, jakým směrem začínají a jakým končí, nemusíme počítat směrování, ale některé figury mohou začít jakýmkoliv směrem a proto je zde metoda handleDirection, která vypočítá výsledný směr z předchozí figury a míry otočení.

Generátor Samby je programátorsky menší, ale to kompenzuje složitější strukturou dat v databázi a vícero záznamy v ní. Jelikož má každá figura definovaný směr na začátku i konci, je složitější skládat figury za sebe a tvoří se tak mnoho podmíněných kroků v kódu, což jej dělá více nepřehledným a na oko ne moc hezkým.

6 Plány do budoucna

- Do budoucna plánuji vytvořit aplikaci pro iOS, kterou jsem zatím nevyvíjel, protože nemám žádné zařízení na otestování a můj pracovní notebook nepodporuje virtuální zařízení.
- Chtěl bych zlepšit ověření uživatele a přístup do soukromé sekce. Zatím je to trochu improvizované, protože se mi nepovedlo plně napojit Fetch API na Spring Security.
- Nejsem grafik, ale věřím, že bych dokázal vylepšit grafickou stránku webového rozhraní.
- Do Android aplikace a budoucí iOS aplikace bych chtěl přidat offline mod s integrovanou databází v aplikaci.

Závěr

Zezačátku jsem byl názoru, že největší práci si budu muset dát s jádrem serveru, které bude těžké vytvořit. Nakonec mě však mile překvapil Spring Boot, pomocí kterého byla tahle část mnohem jednodušší. Delší dobu zabraly generátory, kde jsem vymýšlel, jak by to mohlo fungovat.

Co jsem nečekal, je kolik času mi zabere webové rozhraní. Značnou chvíli trvalo, než jsem se naučil s ReactJS natolik, abych mohl reálně postoupit s mou aplikací.

React Native a aplikace pro android nebyla složitá na vytvoření, ale hodně času zabralo seznámit se s jednotlivými komponentami androidu, abych mohl tvořit prostředí. Právě vývoj aplikace pro android byl důvod, proč jsem ze svého novějšího počítače s Windows přešel na starý notebook s Ubuntu.

Celkově je smysl aplikace ulehčit život trenérům a odbornému dozoru na soutěžích a myslím, že přesně tohohle je moje aplikace schopná. Vždy je však přítomen prostor na zlepšení a upřímně se těším, až aplikaci ještě vylepším a uvidím v ostrém provozu.

Zatím je aplikace nasazená na adrese komi.weby.cz, kde si můžete prohlédnout webové rozhraní.

Conclusions

At the beginning I used to think the server core would be the most difficult part to create. But in the end, I was pleasantly surprised by Spring Boot and with its help was this part much more easier. The generators, on the other hand, took more time, because i had to invent a way to make it work.

What I was not expecting was how much time would it take to create web interface. It took quite a while before I learned ReactJS well enough to continue with developing the application.

React Native application for android was not really hard to create, but I had to learn new components to create the interface. It was at this point, that I have switched notebooks from Windows OS to Ubuntu.

The overall aim of this application was to make lives of trainers and professional supervisors easier and in my opinion, I am quite sure that my application is capable of just that. However, there is always a room for improvement and I am genuinely excited to upgrade my application and to see it go live.

At the moment the application is running on address komi.weby.cz, where you can visit the web interface.

Literatura

[1] WIKIPEDIE, *Český svaz tanečního sportu - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://cs.wikipedia.org/wiki/Český_svaz_tanečního_sportu

[2] WIKIPEDIE, *World DanceSport Federation - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/World_DanceSport_Federation

[3] WIKIPEDIE, *Java - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/Java_(programming_language)

[4] WIKIPEDIE, *Javascript - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://cs.wikipedia.org/wiki/JavaScript

[5] TUTORIALS POINT, *Spring framework - overview*,
Dostupné z: https://www.tutorialspoint.com/spring/spring_overview.htm

[6] JAVA POINT, *Spring Boot tutorial*,
Dostupné z: https://www.javatpoint.com/spring-boot-tutorial

[7] WIKIPEDIE, *React (web framework) - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/React_(web_framework)

[8] ZETCODE, *Spring Boot JpaRepository tutorial*,
Dostupné z: http://zetcode.com/springboot/jparepository/

[9] WIKIPEDIE, *React Native - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/React_Native

[10] BAELDUNG, *The Spring Controller and RestController Annotations*,
Dostupné z: https://www.baeldung.com/spring-controller-vs-restcontroller

[11] SPRING, *Enabling Cross Origin for a RESTful Web Service*,
Dostupné z: https://spring.io/guides/gs/rest-service-cors/

[12] MOZILLA, *HTTP request methods*,
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[13] TUTORIALS POINT, *Spring Autowired Annotation*,
Dostupné z: https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm

[14] BAELDUNG, *Spring RequestMapping*,
Dostupné z: https://www.baeldung.com/spring-requestmapping

[15] CODE ACADEMY, *What is REST*,
Dostupné z: https://www.codecademy.com/articles/what-is-rest

[16] WIKIPEDIE, *Application Programming Interface - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/Application_programming_interface

[17] GEEKSFORGEEKS, *POJO vs. Java Beans*,
Dostupné z: https://www.geeksforgeeks.org/pojo-vs-java-beans/

[18] WIKIPEDIE, *Java Persistence API - Wikipedie: Otevřená Encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/Java_Persistence_API

[19] BAELDUNG, *Spring Data Repositories*,
Dostupné z: https://www.baeldung.com/spring-data-repositories

[20] MODEEFFECT, *Key Differences Between a Website and Web App*,
Dostupné z: https://modeeffect.com/key-differences-between-website-web-app/

[21] WIKIPEDIE, *Document Object Model - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/Document_Object_Model

[22] WIKIPEDIE, *HTML - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/HTML

[23] WIKIPEDIE, *XML - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/XML

[24] WIKIPEDIE, *Cascading Style Sheets - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/Cascading_Style_Sheets

[25] WIKIPEDIE, *WWW - Wikipedie: Otevřená encyklopedie*,
Dostupné z: https://en.wikipedia.org/wiki/World_Wide_Web

[26] TUTORIALSPOINT, *ReactJS Overview - Tutorials Point*,
Dostupné z: https://www.tutorialspoint.com/reactjs/reactjs_overview.htm

[27] GOOGLE, *Working with the Fetch API - Google*,
Dostupné z: https://developers.google.com/web/ilt/pwa/working-with-the-fetch-api

[28] W3SCHOOLS, *React Lifecycle - W3Schools*,
Dostupné z: https://www.w3schools.com/react/react_lifecycle.asp

A Obsah přiloženého CD/DVD

server/	
thesis.jar	Spustitelný .jar soubor se serverem.
src/	Zdrojový kód serverové části.
android/	
thesis.apk	Podepsaný .apk soubor.
src/	Zdrojový kód části s android aplikací.
web/	
src	Zdrojový kód webu.
PDFloader/	
src/	Zdrojový kód pomocného programu.
WDSF.pdf	Katalog figur podle WDSF.
CSTS.pdf	Katalog figur podle ČSTS.
kidiplom/	Zdrojový kód tohoto dokumentu.
README.txt	Instalační instrukce.