BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRNÖK INFORMATIKUS SZAK

Rendszerfejlesztés szakirány

Önálló laboratórium 1. (BMEVIIIM814)

Webalkalmazás fejlesztése JavaEE alapokon

Készítette:

Szirmay-Kalos Barnabás (H5F31Y)

Konzulens:

Dr. Goldschmidt Balázs

IRÁNYÍTÁSTECHNIKA ÉS INFORMATIKA TANSZÉK

2014

TABLE OF CONTENTS

1. ABSTRACT

In this thesis I will demonstrate how to use Java Enterprise Edition's Spring Framework to develop a web applications. I will present a working example of an educational website helping people to improve their English pronunciation, and I will demonstrate how I utilized the framework in the case of this pronunciation trainer application.

2. INTRODUCTION

Before going into the technical details of the Java toolset I used, I'd like to briefly present the motivations and the concept behind the pronunciation trainer application.

2.1. Motivations

The biggest motivating factor for creating the application is the fact that English is becoming more and more the global language. Almost 40% of people in the European Union speak English as a second language, and this percentage is even higher among the young population [1]. The increasing use of the English language globally drives an increasing demand for educational software.

Secondly, even though English pronunciation may be harder to master then vocabulary or grammar, it might be the most important part of oral communication. A bad pronunciation not only gives away the fact that one isn't a native speaker immediately, it's often considered an indicator of a bad English skill all together.

Furthermore, even native speakers may be interested in learning new accents, they might be able to distinguish the different types of English accents, but they are usually unable to imitate them properly.

2.2. English pronunciations

The English language, its accents and pronunciations is a widely researched field of linguistics. I have found that the best method to improve one's pronunciation is considered to be that one records and replays his own voice while trying to pronounce a phrase [2]. This is why I'll focus the application's main functionality around these use cases.

1. Users can record their voice, and replay it
2. Users can fetch native samples from the server and play them
3. Users can request meta-information on native phrase samples
4. Users can select certain native samples based on the speaker's profile (region of origin, sex, etc.)
5. Users should have feedback on how well they have pronounced a certain phrase

3. JAVA ENTERPRISE EDITION

After painting a picture of the intended web application, I will go into the technological details of developing Java web applications, starting with a brief overview of the Java Enterprise Edition platform and its main features.

The Java programming language is a high-level object-orientated programming language. Java technology has three platforms, Java Standard Edition (SE), Java Enterprise Edition (EE) and Java Micro Edition (ME), each with its own intended environment to be used in [3].

The Java EE platform provides an API and a runtime environment for large-scale, multi-tiered, scalable, reliable, and secure network applications. The Java EE platform is based on the Java SE platform, and therefore it has the benefit of "write once deploy anywhere" application development. Additionally the technology is open-source, and it has received tremendous support and a large set of libraries from the open-source community.

3.1. Multi-tier applications

Firstly, as web application development and in general server side programming is a complex task, maintainability of the application architecture is usually a key concern. Modularity, designing the application using interacting parts with well-defined interfaces is a typical method to improve maintainability and decoupling in the case of Java EE applications.

Two-tier, thick-client applications, with separated business logic and data access layers are easier to develop but harder to maintain as any change in the business logic has to be deployed to the client. A more maintainable solution is the three-tiered application pattern, which is when the server hosts presentation layer [4].

As the next step, modern web applications have web-enabled their services in the business logic layer. Using the browser's client-side script languages like Javascript takes the burden of rendering from the server, and only the rendering scripts have to be moved to the client. Client side scripts are not only capable of more sophisticated rendering, like animating or asynchronous communication with the server, but they can take provide better scalability.

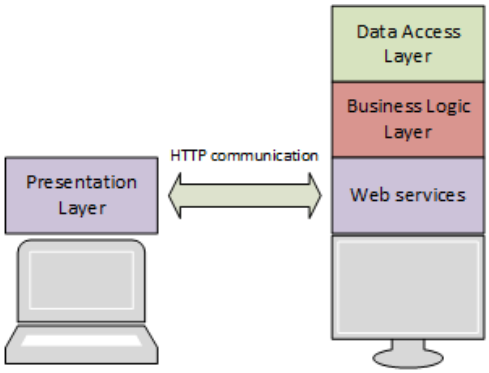

Figure 3.1 Architecture overview of the pronunciation trainer application

For the pronunciation trainer application I've chosen a multi-tier architecture (see figure 3.1), with separated presentation layer implemented in Javascript, web service layer implemented using Spring MVC, a business logic layer implemented in Java and a data access layer using the Java Persistence API.

3.2. Enterprise Java Beans

One of the introduced novelties of the Java EE framework was Enterprise Java Beans. Almost any Java object can be treated as a bean. Beans are contextual, meaning that they are instantiated and managed by a statefull context that they belong to.

Beans have a type and a qualifier to distinguish themselves from other beans of the same type. Beans have a scope, which determines their life cycle, singleton beans are application scoped, other beans can be transaction scoped, session scoped, etc. [5]

The main advantage of beans is their ease of use. If they depend on other beans, their dependencies will be automatically injected from the container. This not only eliminates the need to initialize business objects in the code, but helps to decouple the application modules. As dependencies can be injected using their type, the modules will not depend on the implementation of an interface as long as one is found in the bean container.

3.3. Java Persistence API

Another welcomed feature introduced in the Java EE framework was the Java Persistence API, which brought a standardized method to handle relational databases.

The Java Persistence API provides a library for persisting POJOs (Plain Old Java Objects). By providing the appropriate annotations for the class and its fields, it takes care of persisting and reading them from relation databases. This doesn't only come with all the advantages of object-relational mapping (ORM), but also decouples the business logic layer from the actual database used.

In a later chapter I will demonstrate how the pronunciation trainer application benefits from the Java Persistence API. For example I'll show how it makes the data access layer's implementation completely decoupled from the specific relational database used, which allows the web application to easily switch between an in-memory relation database or a MySQL server.

3.4. Summary

In conclusion, the Java Enterprise Edition already comes with a wide range of tools helping the development of web applications. It extends the Java SE API with useful new libraries helping to develop maintainable applications and forces commonly accepted design patterns to be used.

In the next chapters I will present the various open-source tools I researched and used for the pronunciation trainer application.

4. MAVEN

The original purpose of Maven was to provide a simple, cross-platform tool for building complex Java applications. It follows the *"convention over configuration"* principle, meaning that it enforces a certain library and file structure. Maven also offers a shared repository for downloading common JAR components, allows using custom plugins and provides various tools for quality control. [5]

4.1. Project descriptor

Each Maven project has to define the Project Object Model (pom) file at the project's root containing the project's meta data. The header of the file contains the modules contained by the parent artifact and its name, version, etc.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.webther.pronun</groupId>

    <artifactId>pronun-parent</artifactId>

    <version>1.0-SNAPSHOT</version>

    <name>Pronunciation Trainer</name>

    ...

    <modules>

        <module>pronun-voice</module>

        <module>pronun-data</module>

        <module>pronun-webapp</module>

    </modules>

    ....
```

When building the project, it first builds all its modules and it tries to resolve its defined dependencies from either the local Maven repository or from one of the configured remote maven repositories.

```
<dependencyManagement>
    <dependencies>
        <!-- Modules -->
        <dependency>
            <groupId>com.webther.pronun.voice</groupId>
            <artifactId>pronun-voice</artifactId>
            <version>${project.version}</version>
        </dependency>
        ....
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
        ...
    </dependencies>
</dependencyManagement>
```

4.2. Built-in plugins

For functionality not provided by Maven out-of-the box it's necessary to use plugins. For example, the pronunciation trainer application depends on the *TarsosDSP* voice processing library. It can't be expected that it’s deployed in the local Maven repository on the build machine and it's not in any central Maven repository either.

In this case I have used the exec-maven-plugin to deploy the JAR file in the local repository before the compilation phase. Adding the following plugin definition to the POM file will run the *maven install:install-file* command just before compiling the modules and running the defined tests.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2.1</version>
    <inherited>false</inherited>
    <executions>
        <execution>
            <phase>validate</phase>
            <goals>
                <goal>exec</goal>
            </goals>
            <configuration>
                <executable>mvn</executable>
                <arguments>
                    <argument>install:install-file</argument>
                    <argument>-Dfile=${basedir}/lib/TarsosDSP-1.7.jar</argument>
                    <argument>-DgroupId=tarsosdsp</argument>
                    <argument>-DartifactId=tarsosdsp</argument>
                    <argument>-Dversion=1.7.0</argument>
                    <argument>-Dpackaging=jar</argument>
                </arguments>
            </configuration>
        </execution>
    </executions>
</plugin>
```

4.3. Custom plugins

As one of the main uses cases of the pronunciation trainer application is that the users should be able to listen to native samples, these native samples have to be accessible to the application. As the development of the application is already in the early stages I wanted it

not to depend on an existing database, so I've implemented a custom Maven plugin which gathers the samples during the packaging phase of the build.

This plugin requires a file containing comma separated values of the native samples to be gathered, and downloads their native samples and their International Phonetic Alphabet (IPA) representation into the directory and file specified. As you'll see this directory is the *webapp* folder, which is going to be added to the web archive assembled by Maven.

```
<plugin>
    <groupId>com.webther.pronun</groupId>
    <artifactId>pronun-loader-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <inherited>false</inherited>
    <configuration>
        <directory>${basedir}/pronun-webapp/src/main/webapp/samples/</directory>
        <csvFile>${basedir}/pronun-
webapp/src/main/resources/puzzles.csv</csvFile>
        <outFile>${basedir}/pronun-
webapp/src/main/resources/entityList.csv</outFile>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>load</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

4.4. Conclusion

By properly configuring Maven I've managed to simplify the compilation, verification and deployment of the application, meaning that by issuing a few simple commands the development environment can be duplicated in any machine running JVM.

5. SPRING FRAMEWORK

Spring Framework is an open source application initially aimed for the Java platform. It extends the Java EE API and it also addresses the difficulties of developing complex, scalable, multi-tier applications.

The Java EE platform's Java Beans, the Java Persistence API, Java EE's component based structure and other popular features were already a huge success, some of its difficulties have forced the development community to look for alternatives. The platform's complexity has led to lot of unnecessary boilerplate code to be written, the fact that Java EE components run inside the servlet container has made test-driven development difficult and the misuses of distributed objects have often led to even a simple application not performing to its desired level [4].

The Spring Framework attempts to provide an alternative to classic Java EE development by fixing the difficulties and flaws of the platform.

5.1. Inversion of Control

The Spring Inversion of Control container is the heart of the framework. Its principle is that custom-written code portions receive the control flow from the core library, and it's not the custom-written code which invokes the library [4].

For example in case of the pronunciation trainer application, we implement various REST controllers which communicate with the client side scripts. One of these REST controllers for example is responsible for processing sound samples recorded by the user's microphone and returning the interval of time where actual speech was detected.

In this case by configuring Spring to scan for annotated Spring Beans in a specified package, the core will process the received HTTP message and invoke the custom controller implementation with the desired arguments.

```
@Controller
@RequestMapping("/voice")
public class WAVUploadController {
    ...
    @RequestMapping(method = RequestMethod.POST)
    public @ResponseBody ResponseEntity<SpeechInterval> upload(
            @RequestHeader("X-Voice-Session") String sessionId,
            @RequestHeader("X-Puzzle-Id") String puzzleId,
            MultipartHttpServletRequest request) throws InvalidUploadFormatExcep-
tion {
                ...
                return new ResponseEntity<SpeechInterval>(interval,
HttpStatus.OK);
                ....
    }
}
```

When the servlet receives a POST request to the specified URI, it parses its HTTP headers, and passes them as classic method arguments. Finally it converts the POJO returned by the method and sends a JSON response to the client. It's not only not the coder's responsibility to deal with HTTP communication, the Inversion of Control design principle eliminates the need to manually manage the controller, which leads to more decoupled code.

5.2. Dependency Injection

The Dependency Injection is a design pattern which takes the factory method pattern a step further. Similarly to Java Enterprise Beans, the framework enables defining Java objects as Spring Beans, and the framework will handle instantiation of the object. Additionally a Spring Bean may define other Spring Beans as its dependencies, in which case the framework will automatically inject these dependencies.

In case of the pronunciation trainer example, I've defined a service class which loads entities from the database with a given name. To promote loose decoupling, the implementation and the interface of this service is separated. As the current implementation depends on a JPA repository for accessing entities in the database, this dependency is injected by the framework.

```
@Service
public class PuzzleServiceImpl implements PuzzleService {
    @Autowired
    private PuzzleRepository repository;

    @Override
    public PuzzleEntity getPuzzle(String puzzleCode) throws PuzzleEntityNot-
FoundException {
        ...
    }
    ...
}
```

5.3. Spring Data

Spring Data is an extension for the Spring Framework, which works alongside the Java Persistence API in order to provide improved support for relation database technologies. But in order to understand Spring Data's place in the data access layer, I have to talk about JPA in more detail first.

Before the Java Persistence API (JPA) the two main technologies available for implementing a persistence layer was the Java Database Connectivity (JDBC) API or third party ORM libraries like Hibernate. The use of ORM libraries freed the developers from constructing queries and dealing with model objects returned by the query. The Java Persistence API is a standardized implementation for the persistence layer that uses relation database. Most of its features were originally introduced in third party ORM libraries [6].

The API defines persistent objects as *entities*, each entity class is generally represented by a table in the database. Entities are managed by an *entity manager*, it's responsible for reading and persisting the entities.

In the case of the pronunciation trainer application, the application uses the Java Persistence API to store and access meta information on the native pronunciation samples. These ensembles of meta information are called puzzles, as the user is expected to solve them by pronouncing the word correctly. In the example below, I'll demonstrate how I've used annotations to define the PuzzleEntity JPA entity.

```
@Entity
@Table(name = "puzzles")
public class PuzzleEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "puzzle_id")
    private Long puzzleId;

    @Column(name = "puzzle_text")
    private String puzzleText;
    ...
}
```

Ultimately this means that whenever the application needs to access or modify records in the database, it can do so using simple Java objects, and the framework will handle processing the native database queries and mapping the data to the entity's fields.

The Java Persistence API also defined its own, platform independent and object-orientated query language the Java Persistence Query Language (JPQL). Although this was already a large step forward from writing native database queries, a lot of functionality dealing with basically the same thing had to be rewritten when implementing JPA repositories.

Spring Data takes over the responsibility of implementing these repositories, as long as it's interface's methods are named following the frameworks conventions. For example, if we'd like to provide searching for PuzzleEntity by their system code or we'd like to find PuzzleEntities where their phrase contains a certain word, we'd only need to implement the following interface, and the framework will provide the implementation wherever it's injected to Spring Bean.

```
public interface PuzzleRepository extends JpaRepository<PuzzleEntity, Long>{


    public List<PuzzleEntity> findByPuzzleCode(String puzzleCode);


    public List<PuzzleEntity> findByPuzzleTextLike(String puzzleText);
}
```

6. APPLICATION

The application itself consists of a single page, with three buttons in the header next the challenge phrase to be pronounced by the user. The *"Play Native"* button will play the sample for the phrase, a recorded sample for the British pronunciation of the word.

The *"Let me try!"* will activate the user's microphone, record his/hers speech for 10 seconds and upload the recorded content to the server. Afterwards the server analyzes the sample, selects the interval where actual speech was detected, and returns to start and end time of this interval to the browser.

The *"Replay mine"* button replays the recorded sample, but only the part where speech was detected. Meaning that even if the user only attempted to pronounce the phrase 8 seconds after clicking the *"Let me try!"* button, it will only play back the last few seconds.

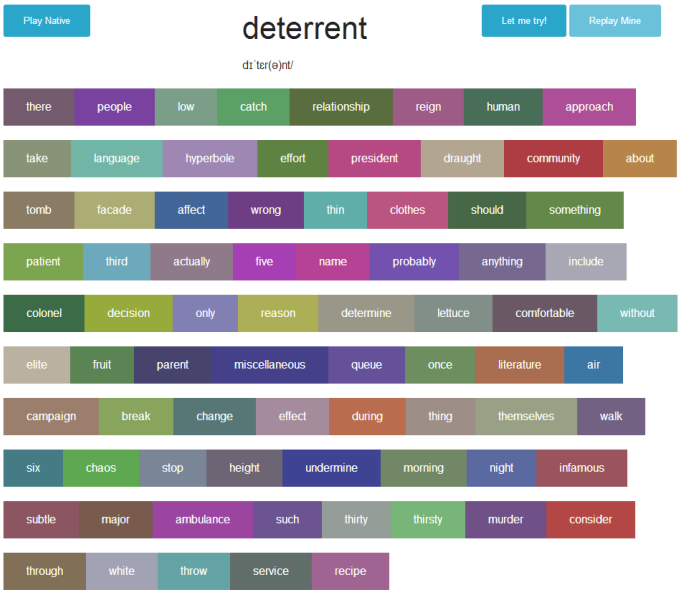

Figure 6.0 Example screen for the pronunciation trainer application

Currently the demo supports approximately 50 different puzzles (see figure 6.0), each can be loaded by clicking one of the boxes below the header [7]. The actual phrases presented here are easy to configure, the **pronun-loader-plugin** Maven plugin loads them from a file, if its configured to load the phrases from a different file, it will download the native samples for those phrases and update the databases with their phonetic representation.

7. WORK LOG

7.1. 2nd week

By the second week I familiarized myself with the technologies used to record and play back audio in the browser. I'll have studied available libraries and tools, and select found that TarsosDSP fits my needs the best.

7.2. 4th week

By the 4th week I've implemented a demo Spring application capable if accepting Multipart HTTP requests containing sound samples and executing some simple sound processing.

7.3. 6th week

I've worked on the client side scripts. I've studied and tried the Foundation CSS framework and the Require.js Javascript framework. I've also studied the HTML5 sound API and implemented a sandbox application which can record and play back the user's voice in the browser.

7.4. 8th week

I've researched possible methods for gathering native sound samples, and I've implemented a Java program which looks for native samples in known locations.

7.5. 10th week

At this point I familiarized myself with the Java Persistence API and Spring Data. I've implemented the needed repositories and configured the Spring Framework to work with H2 base.

7.6. 12th week

At this point I was mainly focused on writing a report on my research findings and on the application's features so far. I've also implemented few new features.

8. REFERENCES

[1] http://ec.europa.eu/public_opinion/archives/ebs/ebs_237.en.pdf European Commission: Europeans and Language

[2] http://reallifeglobal.com/7-tips-to-drastically-improve-your-pronunciation-in-english Real Life: 7 Tips to Drastically Improve Your Pronunciation in English

[3] http://docs.oracle.com/javaee/ Oracle Corporation: Java Platform, Enterprise Edition (Java EE) Technical Documentation

[4] Dhrubojyoti Kayal, Apress, 2008: Pro Java™ EE Spring Patterns

[5] Peter A. Pilgrim, Packt Publishing, 2013: Java EE 7 Developer Handbook

[6] Petri Kainulainen, Packt Publishing, 2012: Spring Data

[7] https://github.com/wther/pronun, Barnabas Szirmay, How I Sound? – English Pronunciation Trainer