REST

Marcin Flis, Kamil Osuch

REST

REpresentational State Transfer

- Styl architektury oprogramowania – zbiór zasad, ograniczeń i porad (w przeciwieństwie do SOAP, który jest protokłem)
- Nie jest standardem – nie ma ISO/RFC
- zazwyczaj oparty o protokół HTTP
- bazuje na prostych operacjach typu CRUD
- bezstanowa architektura

HETAOS

Hypermedia As The Engine Of Application State
Example:

Example Request:

```
GET /accounts/12345 HTTP/1.1
    Host: bank.example.com
    Accept: application/xml
```

response:

```
    <account>
       <account_number>12345</account_number>
       <balance currency="usd">100.00</balance>
       <link rel="deposit" href="https://bank.example.com/
       <link rel="withdraw" href="https://bank.example.com
       <link rel="transfer" href="https://bank.example.com
       <link rel="close" href="https://bank.example.com/ac
     </account>
```

other response:

```
  <account>
        <account_number>12345</account_number>
        <balance currency="usd">-25.00</balance>
        <link rel="deposit" href="https://bank.example.com
    </account>
```

JAX-RS

Jest to ustandaryzowane API definujące adnotacje do tworzenia rozwiązań REST.
Popularne implementacje:

- Jersey (Glassfish)
- RESTeasy (Jboss)
- Apache CXF

Resteasy - dependency

```
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>jaxrs-api</artifactId>
    <scope>provided</scope>
</dependency>
```

Adnotacje podstawowe

- @Path – ścieżka do zasobu, może być nad klasą albo nad metodą klasy
- @GET, @PUT(...) - określa, na jakie zapytania odpowie nasz zasób
- @Produces – mówi, jaka będzie reprezentacja zasobu
- @Consumes – jakie reprezentacje możemy przyjąć

Adnotacje vol. 2

- @PathParam – przypisuje kawałek adresu do parametru metody
- @QeryParam – cos.pl/list?zmienna=wartosc
- @FormParam – z formularza
- @CookieParam – z ciasteczka
- @HeaderParam -z nagłówka HTTP
- @DefaultValue – jak nic innego się nie udało

Przykładowy endpoint

```
@PUT
@Path("/{index}/course")
public void addCourse(@QueryParam("course-name") @DefaultValue(DEFAULT_COURSE_NAME) String name,
                      @QueryParam("ects") @DefaultValue("1") Integer ects,
                      @PathParam("index") String indexNumber) {
    Student student = studentsRepository.getStudent(indexNumber);
    String courseName = name;
    if(DEFAULT_COURSE_NAME.equals(courseName)){
        courseName += indexNumber + "_";
        long number = student.getCourses().stream()
                .filter(course -> course.getName().contains(DEFAULT_COURSE_NAME))
                .count();
        courseName += (number + 1);
    }
    studentsRepository.getStudent(indexNumber).addCourse(new Course(courseName, ects));
}
```

Application - wymagane żeby serwis zadziałał

```
@ApplicationPath("/app")
public class MyApplication extends Application {

}
```

Wysyłanie plików - multipart

```
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-multipart-provider</artifactId>
    <version>3.1.2.Final</version>
     <scope>provided</scope>
</dependency>
```

Wysyłanie plików - formularz

```
<html>
<body>
<h1>Send photo form</h1>

<form action="app/students/photo" method="post" enctype="multipart/form-data">

    <p>
        Index number : <input type="text" name="index" />
    </p>
    <p>
        Select a file : <input type="file" name="uploadedFile" size="50" />
    </p>

    <input type="submit" value="Upload It" />
</form>

</body>
</html>
```

MultipartDataForm

```java
public class FileUploadForm {

    public FileUploadForm() {
    }

    private byte[] data;

    private String index;
    public byte[] getData() { return data; }

    @FormParam("uploadedFile")
    @PartType("application/octet-stream")
    public void setData(byte[] data) { this.data = data; }


    @FormParam("index")
    public void setIndex(String index) { this.index = index; }

    public String getIndex() { return index; }

}
```

Wysyłanie plików - multipart, endpoint

```java
@POST
@Path("/photo")
@Consumes(MediaType.MULTIPART_FORM_DATA)
public Response uploadPhoto(@MultipartForm FileUploadForm fileUploadForm) {
    Student student = studentsRepository.getStudent(fileUploadForm.getIndex());
    try {
        String savePath = SAVE_PATH + fileUploadForm.getIndex() + ".png";
        writeFile(fileUploadForm.getData(), savePath);
        studentsRepository.addStudentsPhoto(student.getIndexNumber(), savePath);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return Response.status(Response.Status.OK).build();
}
```

Protocol Buffers

- protokół de/serializacji danych stworzony przez Google
- oparty na schematach wiadomości zapisywanych w plikach `*.proto`
- pliki te sa poźniej kompilowane do kodu źródłowego w wybranym języku(Java, Python, C++, etc),który obsługuje de/serializację oraz manipulację danymi

Przykładowy plik `*.proto`

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Przewaga nad XML

- prostszy format
- 3 - 10 razy mniejszy rozmiar wiadomości
- 20-100 szybsza de/serializacja
- posiada kompilator generujacy kod do obsługi protokołu i danych w Java/Python/C++/etc. na podstawie `*.proto`
- można dodawać nowe pola do wiadomości nie tracąć kompatybilności wstecznej

Generowane metody

```
// required string name = 1;
public boolean hasName();
public String getName();


// repeated .tutorial.Person.PhoneNumber phone = 4;
public List<PhoneNumber> getPhoneList();
public int getPhoneCount();
public PhoneNumber getPhone(int index);
```

Generowane metody (builder)

```
// required string name = 1;
public boolean hasName();
public java.lang.String getName();
public Builder setName(String value);
public Builder clearName();

// repeated .tutorial.Person.PhoneNumber phone = 4;
public List<PhoneNumber> getPhoneList();
public int getPhoneCount();
public PhoneNumber getPhone(int index);
public Builder setPhone(int index, PhoneNumber value);
public Builder addPhone(PhoneNumber value);
public Builder addAllPhone(Iterable<PhoneNumber> value);
public Builder clearPhone();
```

Metody na wiadomości

- `isInitialized()`: checks if all the required fields have been set.
- `toString()`: returns a human-readable representation of the message, particularly useful for debugging.
- `mergeFrom(Message other)`: (builder only) merges the contents of other into this message, overwriting singular scalar fields, merging composite fields, and concatenating repeated fields.
- `clear()`: (builder only) clears all the fields back to the empty state.

```
<person>
  <id>123</id>
  <name>John Doe</name>
  <email>jdoe@example.com</email>
</person>
```

ProtoBuf Text (for debugging)

```
person {
 id: 123
 name: "John Doe"
 email: "jdoe@example.com"
}
```

ProtoBuf binary [Hex]

```
0a1e0a084a6f686e20446f65107b1a106a646f65406578616d
706c652e636f6d
```

Generowanie klas Java z plików proto - Maven

```xml
<plugin>
    <groupId>org.xolstice.maven.plugins</groupId>
    <artifactId>protobuf-maven-plugin</artifactId>
    <version>0.5.0</version>
    <configuration>
        <protocExecutable>protoc</protocExecutable>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>compile</goal>
                <goal>test-compile</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

```xml
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
    <version>3.2.0</version>
</dependency>
```

Generowanie klas Java z plików proto - Gradle

protobuf{}

```
plugins {
    id "com.google.protobuf" version "0.8.1"
    id "java"
    id "idea"
}

group 'com.arch'
version '1.0-SNAPSHOT'


repositories {
    mavenCentral()
}

dependencies {
    compile 'com.google.protobuf:protobuf-java:3.0.0'
}

protobuf {
    // Configure the protoc executable
    protoc {
        // Download from repositories
        artifact = 'com.google.protobuf:protoc:3.0.0'
    }
    generatedFilesBaseDir = "$projectDir/gen"

}

clean {
    delete protobuf.generatedFilesBaseDir
}

idea {
    module {
        sourceDirs += file("${protobuf.generatedFilesBaseDir}/main/java");
    }
}
```

Wysłanie protobuf przez REST API

- Resteasy nie wie domyślnie jak zserializować protobuf'a
- Trzeba zaimplementować w jaki sposób obiekty mają być serializowane
- Osiągamy to za pomocą stworzenia implementacji klas MessageBodyWriter i MessageBodyReader

MessageBodyWriter - przykład dla protobuf

```java
@Provider
@Produces("application/x-protobuf")
public class ProtobufMessageWriter implements MessageBodyWriter<Message> {
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations,
                               MediaType mediaType) {
        return Message.class.isAssignableFrom(type);
    }

    public long getSize(Message m, Class<?> type, Type genericType, Annotation[] annotations,
                        MediaType mediaType) {
        return m.getSerializedSize();
    }

    public void writeTo(Message m, Class<?> type, Type genericType, Annotation[] annotations,
                        MediaType mediaType, MultivaluedMap<String, Object> httpHeaders, OutputStream entityStream) throws IOException,
            WebApplicationException {
        entityStream.write(m.toByteArray());
    }
}
```

MessageBodyReader - przykład dla protobuf

```java
@Provider
@Consumes("application/x-protobuf")
public class ProtobufMessageReader implements MessageBodyReader<Message> {
    public boolean isReadable(Class<?> type, Type genericType, Annotation[] annotations,
                              MediaType mediaType) {
        return Message.class.isAssignableFrom(type);
    }

    public Message readFrom(Class<Message> type, Type genericType, Annotation[] annotations,
                            MediaType mediaType, MultivaluedMap<String, String> httpHeaders, InputStream entityStream) throws IOExceptio
            WebApplicationException {
        try {
            Method newBuilder = type.getMethod( name: "newBuilder");
            GeneratedMessageV3.Builder<?> builder = (GeneratedMessageV3.Builder<?>) newBuilder.invoke(type);
            return builder.mergeFrom(entityStream).build();
        }
        catch (Exception e) {
            throw new WebApplicationException(e);
        }
    }
}
```

- @Provider - pozwala na automatyczne wykrycie przez JAX-RS w runtime
- @Produces()/@Consumes() - określają dla jakiego Content-Type mają być wykorzystane provider'y

KONIEC