API

Hello!

Michał Nowakowski

lead software engineer @EPAM

michal@nowakowski.me.uk

Agenda

- API
- REST
- SOAP

API
Application Programming Interface

- Interfejs programistyczny aplikacji
- Zespół reguł opisujących wzajemną komunikację programów komputerowych
- Definiowane na poziomie kodu źródłowego
- Może dotyczyć konkretnej aplikacji, biblioteki, bazy danych, systemu operacyjnego

API
Application Programming Interface

- Specyfikacje podprogramów
- Struktury danych
- Klasy obiektów
- Protokoły komunikacyjne

API
Przykłady

- DirectX
- OpenGL
- POSIX
- Windows API
- C++ Standard Template Library
- JPA – Java Persistance API
- Java Collection API

API
Rodzaje

- API systemu operacyjnego – opisuje interfejs między aplikacją a systemem operacyjnym (np. POSIX, WinAPI)
- Zdalne API – zdalne wywoływanie metod, ang. RMI – Remote Method Invocation (np. CORBA, JSON-RPC, XML-RPC, DCOM)
- Web API – wykorzystanie protokołu HTTP i formatu JSON lub XML do komunikacji pomiędzy użytkownikiem a systemem zlokalizowanym na serwerze

API
Web API

- REST – Representational State Transfer, wykorzystanie różnych metod protokołu HTTP i formatu JSON w celu uzyskania dostępu do **zasobów** i ich modyfikacji (JAX-RS)

- SOAP – Simple Object Access Protocol, kompletny protokół opisujący sposób wymiany informacji w formacie XML, **najczęściej** z wykorzystaniem protokołu HTTP

API
Dlaczego?

- Hermetyzacja – ukrywanie implementacji, nieistotnych z punktu widzenia użytkownika detali
- Separacja modułów – każdy moduł opisuje swój interfejs (dosłownie Java **interface**)
- Separacja funkcji klas - tzw. loose coupling

API
Dlaczego?

- Zrównoleglenie prac nad aplikacją – każdy zespół tworzy osobny moduł, z góry wiadomo w jaki sposób ma odbywać się komunikacja pomiędzy modułami
- Każdy moduł może być testowany i utrzymywany osobno
- Poprawianie działania konkretnych modułów w sposób niewidoczny dla użytkownika

API
Projektowanie

- Możliwie niski poziom dostępu
- Nie zwiększać poziomu dostępu w celu testowania (jeśli nie ma innej opcji – przeanalizować i zaprojektować od nowa!)
- Nigdy, nigdy, przenigdy nie deklarować pól publicznych – jeśli już, to tworzyć pola statyczne i finalne (typy proste lub niemutowalne)

API
SOLID

- **S**ingle Responsibility Principle – klasa odpowiedzialna za jedno zadanie, nigdy nie powinien istnieć więcej niż jeden powód do modyfikacji klasy
- **O**pen/Closed Principle – klasa powinna być otwarta na rozszerzenia i zamknięta na modyfikacje
- **L**iskov Substitution Principle – klasy bazowe mogą być zastępowane klasami po nich dziedziczącymi

API
SOLID

- **I**nterface Segregation Principle – wiele dedykowanych interfejsów jest lepsze niż jeden ogólny

- **D**ependency Inversion Principle – wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych, zależności powinny wynikać z abstrakcji

API
Sposoby opisu

- Metody publiczne, Java interface
- Pliki java, XML, JSON, YAML, JavaDocs, WSDL
- Dokument, np. https://cloud.google.com/translate/docs/reference/rest

API
Postman

- Wtyczka do przeglądarki Chrome: https://www.getpostman.com/
- Aplikacja dla Windows, Linux, MacOS
- Budowanie zapytań
- Modyfikacja nagłówków
- Historia zapytań

API
Ćwiczenie 1

- Google Translate API - https://cloud.google.com/translate/docs/reference/rest
- Przy użyciu Postmana zbuduj link pobierający listę dostępnych języków
- API KEY: AIzaSyAD3fH67Y64AT6hbi1z2kXG2VNIkt8H9o4

API
Ćwiczenie 2

- Google Translate API - https://cloud.google.com/translate/docs/reference/rest
- Przy użyciu Postmana zbuduj link tłumaczący dowolną frazę
- API KEY: AIzaSyAD3fH67Y64AT6hbi1z2kXG2VNIkt8H9o4
- Co się stanie jak zmienimy typ metody HTTP?

API
Ćwiczenie 3

- REST Countries - https://restcountries.eu/#api-endpoints
- Pobierz informacje o Polsce (lub dowolnym innym kraju) na podstawie waluty (np. PLN)

API
JAX-RS

- Java API for RESTful Web Services
- Wsparcie dla obsługi serwisów REST (klient / serwer)
- Standard Java EE

API
JAX-RS - Użycie

- ClientBuilder.newClient() tworzy nowego klienta (typ Client)
- WebTarget target = client.target(adres) tworzy odniesienie do serwisu („celu”) na podstawie URI
- Response response = target.request().get() wywołuje serwis i przechowuje odpowiedź w obiekcie typu Response

API
JAX-RS - Użycie

- String value = response.readEntity(String.class) konwertuje odpowiedź do wybranego typu

- response.close() ← !!!

API
Ćwiczenie 4

- `$ git clone https://github.com/infoshareacademy/jjdd2-materialy-api-rest-soap.git`
- Uzupełnij ciało metody GoogleTranslate.translate() tak, aby zwracała odpowiedź serwera jako String (użyj metody GET)

API
Ćwiczenie 4 - rozwiązanie

```
public String translate(String input, String source, String target) {
    final Client client = ClientBuilder.newClient();
    final WebTarget webTarget = client.target(
        "https://translation.googleapis.com/language/translate/v2/?q="
            + input +
            "?&target="
            + target +
            "&source="
            + source
            + "&key="
            + API_KEY);

    final Response response = webTarget.request().get();
    final String responseValue = response.readEntity(String.class);
    response.close();
    return responseValue;
}
```

API
Ćwiczenie 5

- Parametry zapytania (query parameters) nie wyglądają w tym wypadku najlepiej ☹
- Zaimplementuj metodę GoogleTranslate.translate() tak, aby wysyłała zapytanie POST
- Parametry przekazuj przy użyciu obiektu Form (target.request().post(Entity.form(…)))

API
Ćwiczenie 5 - rozwiązanie

```
public String translate(String input, String source, String target) {
        final Client client = ClientBuilder.newClient();
        final WebTarget webTarget = client.target(
            "https://translation.googleapis.com/language/translate/v2");

        final Form params = new Form();
        params.param("q", input);
        params.param("source", source);
        params.param("target", target);
        params.param("key", API_KEY);

        final Response response = webTarget.request().post(Entity.form(params));
        final String responseValue = response.readEntity(String.class);
        response.close();
        return responseValue;
    }
}
```

API
Ćwiczenie 6

- Zaimplementuj metodę GoogleTranslate.translate() tak, aby przechodziła test
- Zamodeluj klasy odzwierciedlające odpowiedź serwisu – data, translations

API
Ćwiczenie 6 - rozwiązanie

```
public String translate(String input, String source, String target) {
    final Client client = ClientBuilder.newClient();
    final WebTarget webTarget = client.target(
        "https://translation.googleapis.com/language/translate/v2");

    final Form params = new Form();
    params.param("q", input);
    params.param("source", source);
    params.param("target", target);
    params.param("key", API_KEY);

    final Response response = webTarget.request().accept(MediaType.APPLICATION_JSON_TYPE)
        .post(Entity.form(params));
    final GoogleTranslateResponse responseValue = response
        .readEntity(GoogleTranslateResponse.class);
    response.close();
    return responseValue.getData().getTranslations().get(0).getTranslatedText();
}
```

API
Ćwiczenie 7

- Serwis zwracający stany USA: http://services.groupkt.com/state/get/USA/all
- Zaimplementuj metodę, która pobierze listę stanów zapytaniem GET i zmapuje odpowiedź do listy obiektów POJO

Thanks!!

Q&A