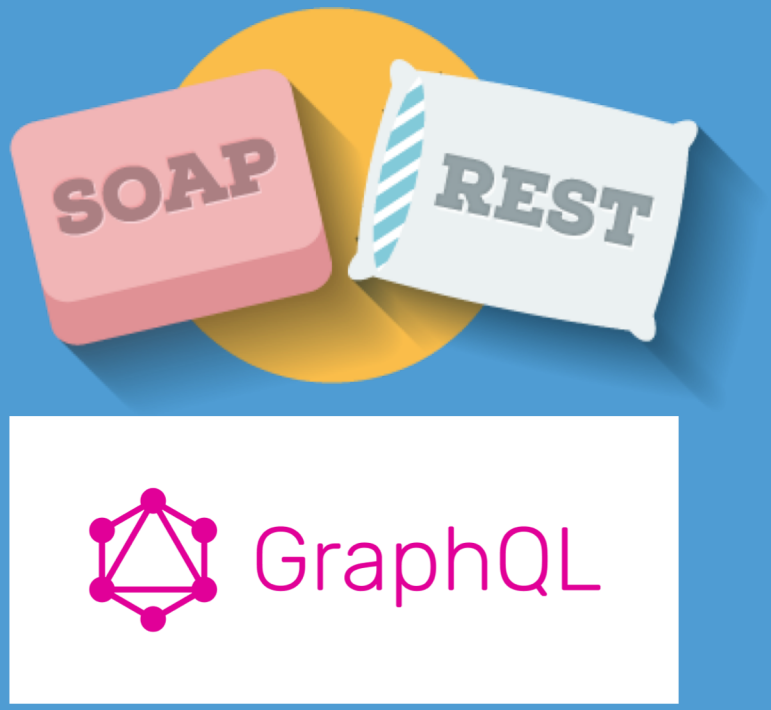
Search . . .



11

Návrhy rozhraní

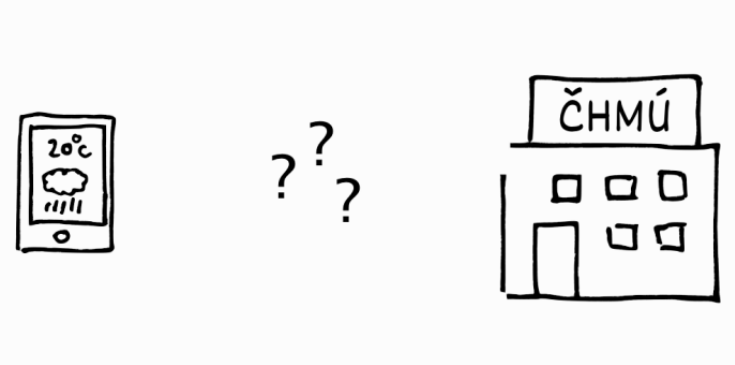
- Behaviorální ekvivalence
- Preconditions a postconditions
- Specifikace
- Výjimky v Java
- Web API
- REST, SOAP, GraphQL
- OpenAPI, Swagger, APIARY

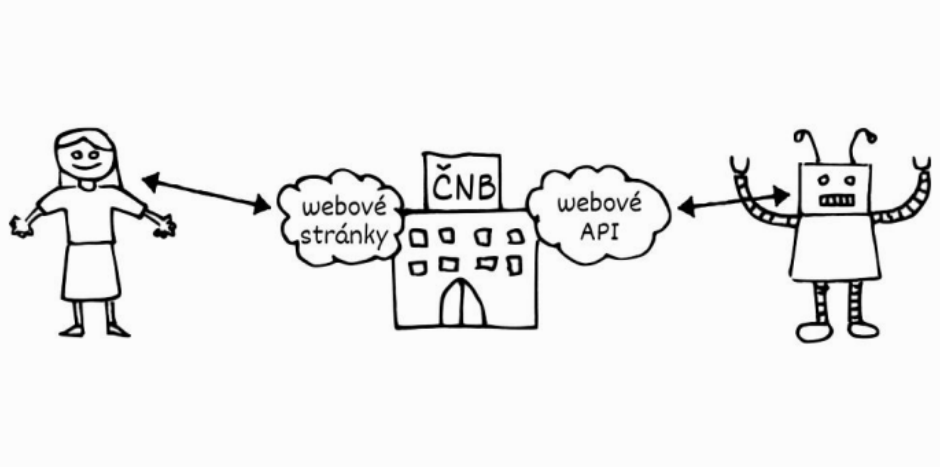
Next >

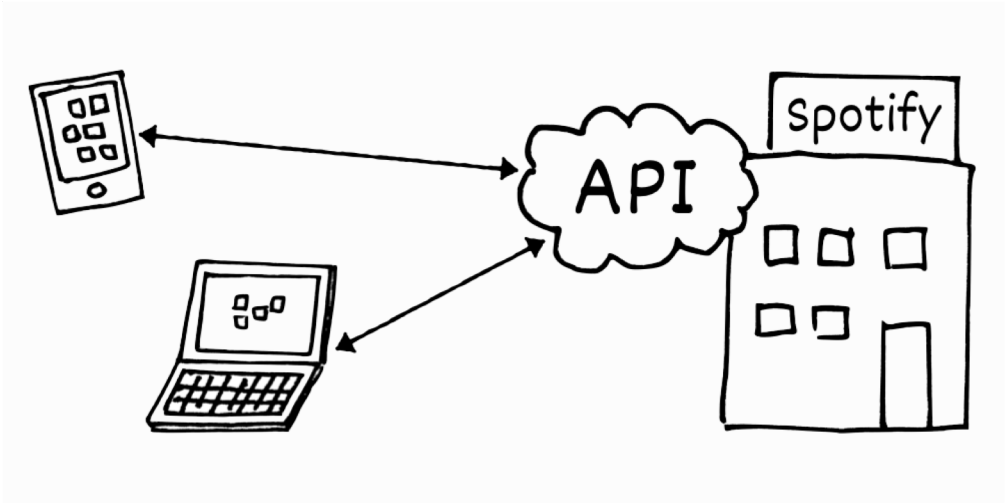
Objektové Modelování - podzim 2022

kadlecd@fel,cvut.cz, www.czm.fel.cvut.cz

11 Co to je API?







11 Motivace

- Proč specifikace?
- Častou příčinou mnoho chyb v chování systému je nedorozumění na rozhraní dvou částí kódu
- Nedokumentované rozhraní je v mysli vývojáře, vývojář druhé části kódu může rozhraní vnímat jinak
- Specifikace rozhraní
  - Dokumentace pro klienta, který rozhraní využívá
    - Nemusí zkoumat kód
  - Odstínění pro vývojáře kódu
    - Vývojář má možnost měnit kód aniž by musel předávat jako dokumentaci nový kód klientům
- Specifikace rozhraní umožní decoupling
- Představuje společný kontrakt, který musí obě strany dodržet
- Obě části - klient volající kód přes rozhraní a vývoj kódu implementace – se mohou vyvíjet nezávisle

11 Proč specifikace

```
public BigDecimal divide(BigDecimal val, int newScale, int roundingMode)
  throws ArithmeticException, IllegalArgumentException
{
  if (roundingMode < 0 || roundingMode > 7)
    throw
  new IllegalArgumentException("illegal rounding mode: " + roundingMode);

  if (intVal.signum () == 0)     // handle special case of 0.0/0.0
    return newScale == 0 ? ZERO : new BigDecimal (ZERO.intVal, newScale);

  // Ensure that pow gets a non-negative value.
  BigInteger valIntVal = val.intVal;
  int power = newScale - (scale - val.scale);
  if (power < 0)
    {
  // Effectively increase the scale of val to avoid an
  // ArithmeticException for a negative power.
      valIntVal = valIntVal.multiply (BigInteger.TEN.pow (-power));
  power = 0;
    }

  BigInteger dividend = intVal.multiply (BigInteger.TEN.pow (power));

  BigInteger parts[] = dividend.divideAndRemainder (valIntVal);

  BigInteger unrounded = parts[0];
  if (parts[1].signum () == 0) // no remainder, no rounding necessary
    return new BigDecimal (unrounded, newScale);

  if (roundingMode == ROUND_UNNECESSARY)
    throw new ArithmeticException ("Rounding necessary");

  int sign = intVal.signum () * valIntVal.signum ();

  if (roundingMode == ROUND_CEILING)
```

divide

```
public BigDecimal divide(BigDecimal divisor,
                         int scale,
                         int roundingMode)
```

Returns a BigDecimal whose value is (this / divisor), and whose scale is as specified. If rounding must be performed to generate a result with the specified scale, the specified rounding mode is applied.

The new divide(BigDecimal, int, RoundingMode) method should be used in preference to this legacy method.

Parameters:

divisor - value by which this BigDecimal is to be divided.

scale - scale of the BigDecimal quotient to be returned.

roundingMode - rounding mode to apply.

Returns:

this / divisor

Throws:

ArithmeticException - if divisor is zero, roundingMode==ROUND_UNNECESSARY and the specified scale is insufficient to represent the result of the division exactly.

IllegalArgumentException - if roundingMode does not represent a valid rounding mode.

11 Behaviorální ekvivalence

- Otázka zda se dva zdrojové kódy chovají stejně
- Možnost výměny kódu za jiný beze změny chování systému
- Roli mohou hrát
  - Množina vstupních parametrů
  - Kontext, ve kterém kód běží
  - Specifikace by měla obsahovat popis těchto vlivů
    - ***Preconditions*** – závazek pro klienta
    - ***Postconditions*** – závazek pro implementátora
- Ekvivalence simulačních modelů
- Behaviorální ekvivalence - není obecně testovatelná

Behaviorální ekvivalence je:

- Reflexivní:
  - E ~ e
- Symetrická:
  - If E1 ~ E2 then E2 ~ E3
- TRANSITIVNÍ:
  - If E1 ~ E2 and E2 ~ E3 then E1 ~ E3
- Především transitivnost důležitá pro simulace

11 Preconditions a Postconditions

- *Preconditions* – popisují podmínky pro vstupní hodnoty
- *Postconditions* – popisují, jakým způsobem má být vymezena implementace

```
static int find(int[] arr, int val) {
  requieres: val occurs exactly once in arr
  effects: returns index i such that arr[i] = val
}
```

- V JavaDoc

```
/**
* Find a value in an array.
* @param arr aray to search, requeires that val occurst exactly once in arr
* @param val alue to search for
* @return index i such that arr[i] = val
*/
static int find(int[] arr, int val)
```

11 Silnější vs. slabší specifikace

V případě změny implementace nebo změny vlastní specifikace potřebujeme určit, jak bezpečné je provést změnu v implementaci vzhledem k již existujícím klientům (systémům, které jsou na API již napojené)

Mějme tedy dvě specifikace **S1, S2**:

- **S2 je silnější nebo stejná jako S1** když
  - Preconditions pro S2 jsou slabší než nebo stejné jako pro S1
  - A zároveň postcondition pro S2 jsou silnější nebo stejné jako pro S1, pro stavy, které uspokojí preconditions pro S1

⇒V takovém případě implementace, která uspokojí potřeby specifikace S2, může být použita pro splnění požadavků kladených na specifikace S1

⇒Je tedy bezpečné provést výměnu stávající specifikace S1 za novou specifikaci S2 a existující klienti nebudou zasaženi

Tato specifikace

```
static int findExactlyOne(int[] a, int val)
requires: val occurs exactly once in a
effects: returns index i such that a[i] = val
```

Může být nahrazena – má slabší precondition:

```
static int findOneOrMore,AnyIndex(int[] a, int val)
requires: val occurs at least once in a
effects: returns index i such that a[i] = val
```

A ta může být nahrazena – má silnější postcondition:

```
static int findOneOrMore,FirstIndex(int[] a, int val)
requires: val occurs at least once in a
effects: returns lowest index i such that a[i] = val
```

11 Null reference

- Reference na objekty a pole mohou mít speciální hodnotu – null
- Neukazuje na žádný objekt
- Nelze pro primitivní objekty
- Použití null – potenciální výjimky nullpointerexception v runtime
- Pokud možno se jim vyhýbat, především na rozhraní
- Dobrá praxe pro každou operaci implicitně nepovolovat null v parametrech ani návratových hodnotách
- V případě, že některý parametr povoluje null, pak ve specifikaci explicitně zmínit – totéž platí i pro návratové hodnoty

11 Specifikace výjimek

- Antipattern – předávat informace o chybě nebo okrajovém stavu pomocí návratové hodnoty

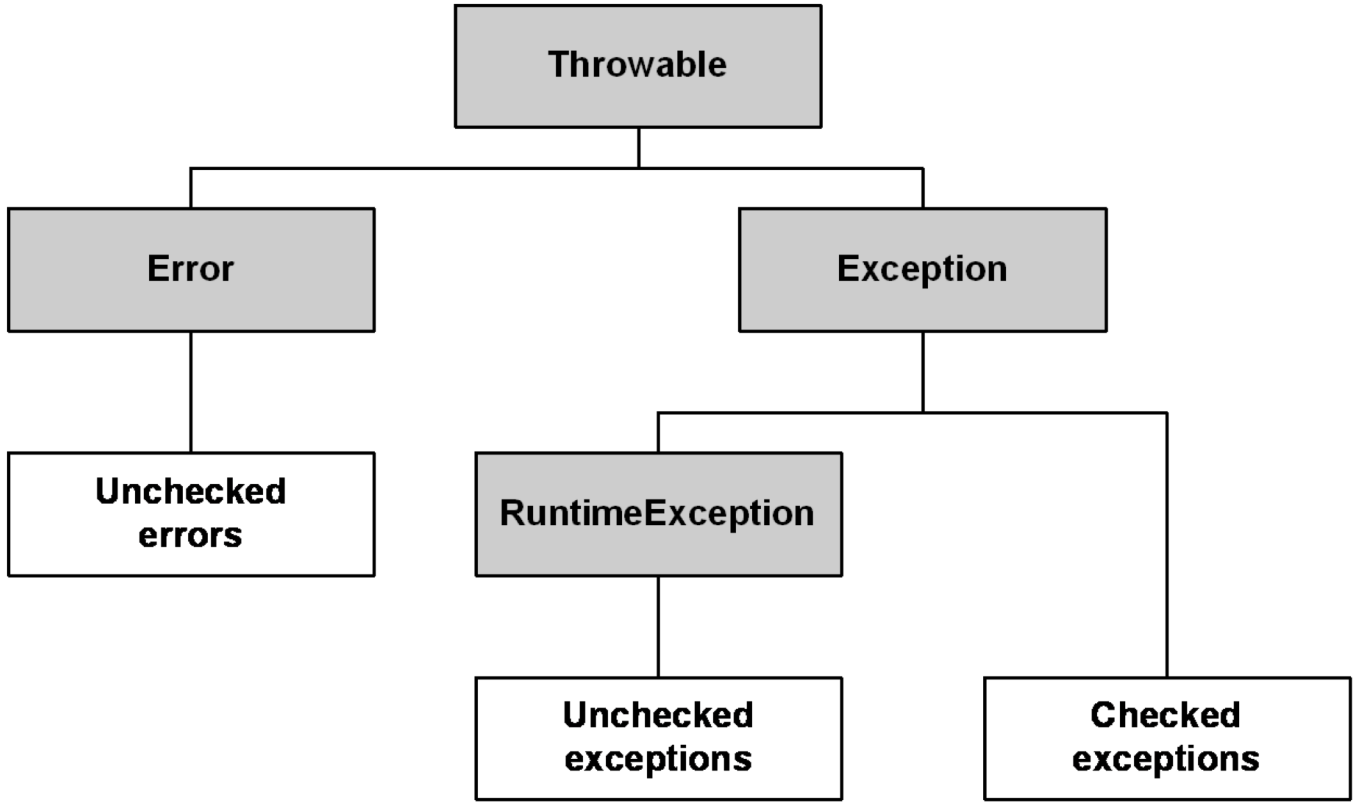
```
class BirthdayBook { LocalDate lookup(String name) { ... } }
```

- Atipattern - pokud v birthdaybook neexistuje dané jméno vrací místo localdate null v horším případě nějaké domluvené datum, které po dobu života systému nepředpokládáme, že nastane 9/9/9999

```
LocalDate lookup(String name) throws NotFoundException {
...
if ( ...not found... ) throw new NotFoundException();
...
```

- Lépe, aby metoda vracela výjimky
- Výjimky se mohou vyskytnout při chybě nebo jako speciální výsledek
- V prvním přiblížen
  - Checked výjimky – pro speciální výsledky
  - Unchecked výjimky – pro chybové stavy případně tam, kde klient bude předcházet speciálním výsledkům ošetřením před voláním metody

11 Hierarchie výjimek v Javě



11 Použití výjimek antipattern

- Nepoužívat výjimky tam, kde se nejedná o výjimečné stavy

```
try {
  int i = 0;
  while (true)
    a[i++].f();
} catch ArrayIndexOutOfBoundsException e) {
}
```

- Není vhodné zde použít výjimku, lepší například for cyklus přes prvky pole
- *ArrayIndexOutOfBoundException* může navíc nastat například jinak než v kódu očekáváme -> těžko odhalitelná chyba

11 Specifikace metod mutujících data

- Metoda, která mění vstupní data

```
static boolean addAll(List<T> list1, List<T> list2)
requires: list1 != list2
effects: modifies list1 by adding the elements of list2 to the end of it, and returns true if
    list1 changed as a result of call
```

- Zásada – neměnit vstupní data
- Raději udělat kopii vstupních dat, změnu provést v kopii a vrátit odkaz na kopii s provedenou úpravou
- Když přeci jen nastane situace, kdy specifikace rozhraní počítá se změnou vstupních dat, uvést ve specifikaci, která vstupní data a k jaké mutaci vstupních dat dochází

11 Význam specifikace

- Předchází chybám – dobrá specifikace jasně dokumentuje předpoklady, na kterých staví klient i implementátor. Chyby často vznikají z nesprávně pochopeného rozhraní a specifikace tomuto přechází. Používání strojem kotrolovaného popisu ve specifikaci jako typová kontrola a použití výjimek může ještě více redukovat chyby
- Srozumitelnost - krátká jednoduchá specifikace je mnohem více srozumitelná než vlastní implementace. Ostatní vývojáři nemusí studovat kód, aby mohli rozhraní využít
- Připravenost na změny - specifikace zavádí kontrakt mezi různými částmi kódu. Tyto části se mohou, pokud dodrží kontrakt rozhraní rozvíjet nezávisle

11 Přesnost popisu specifikace

- Popisuje specifikace pouze jeden možný výstup pro daný vstup nebo dává volnost vývojáři vybrat z množiny možných hodnot?
- Deterministic – jednoznačně určený výstup

```
static int findExactlyOne(int[] arr, int val)
requires: val occurs exactly once in arr effects:
returns index i such that arr[i] = val
```

```
static int findOneOrMore,AnyIndex(int[] arr, int val)
requires: val occurs in arr
effects: returns the first index i such that arr[i] = val
```

- Undetermined – není jednoznačně určeno, vývojář má jistou volnost implementace

```
static int findOneOrMore,AnyIndex(int[] arr, int val)
requires: val occurs in arr
effects: returns index i such that arr[i] = val
```

11 Deklarativní vs. operativní specifikace

- **Operativní** – specifikace popisuje kroky implementace a dává předpis pro vývojáře implementovat
- **Deklarativní** – specifikace nepopisuje detaily interních kroků implementace, popisuje výstup na základě vstupů do metody

Ve většině případů je preferovaný deklarativní přístup

Specifikace s deklarativním přístupem

- Obyčejně kratší
- Lépe srozumitelná
- Neodhaluje vnitřní implementaci, na které by mohl pak klientský kód záviste a byl tak více citlivý na změny v implementaci

11 Dobrý návrh specifikace rozhraní

- Specifikace je klíčový kontrakt mezi implementátorem a klientem
- Měla by být srozumitelná
- Měla by být koherentní – metody by měly mít vždy jednu jasně definovanou zodpovědnost
- Návratová hodnota by měla nést jasnou a jednoznačnou informaci
- Specifikace by měla využívat abstraktní typy
- Zabraňuje chybám při volání rozhraní způsobených různým výkladem rozhraní
- Připravena na změny

11 Testování a specifikace

- Testy musí splňovat podmínky specifikace
- Znalost implementace může být vhodným doplňkem pro testování
  - Může například pomoci s testováním specifických podmínek, které nejsou ve specifikaci uvedeny
  - **Testy nesmí být podřízeny implementaci, ale výhradně specifikaci rozhraní**

```
static int find(int[] arr, int val)
requires: val occurs in arr e
ffects: returns index i such that arr[i] = val
```

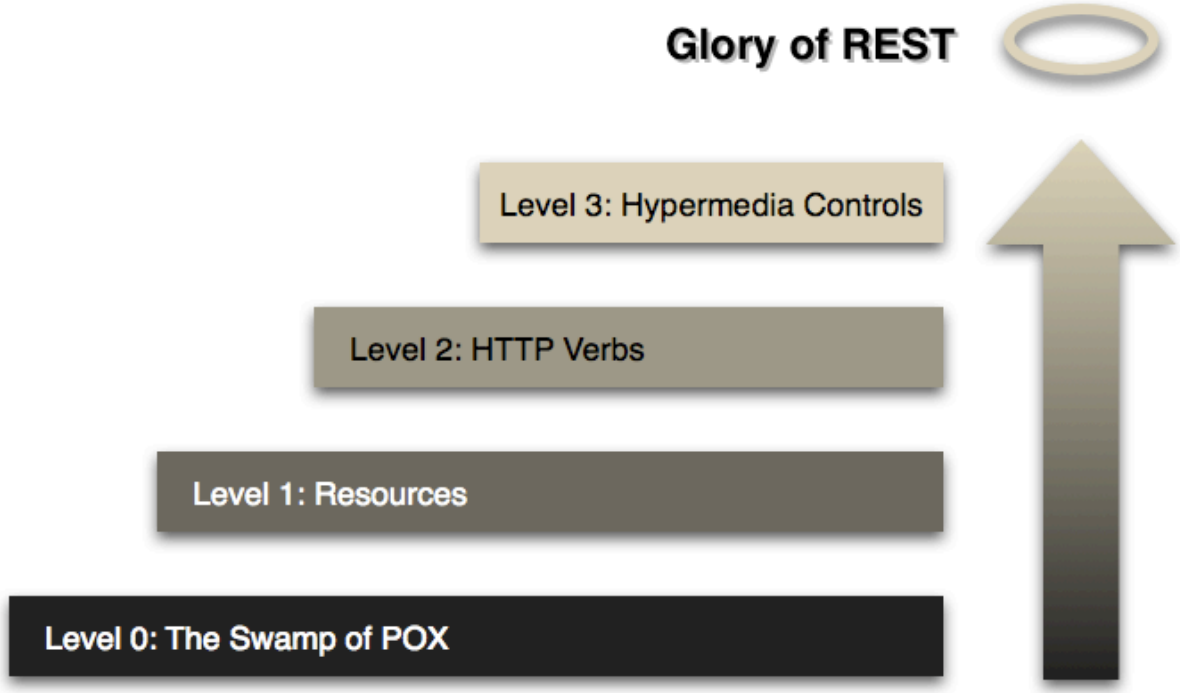
```
int[] array = new int[] { 7, 7, 7 };
assertEquals(0, find(array, 0)); // bad test case: violates the spec
assertEquals(7, array[find(array, 7)]); // correct
```

11 REST (Representational State Transfer)

- Navrhl Roy Fielding v roce 2000
- REST je nezávislý a nemusí být nutně vázaný na HTTP protokol, ale ve většině implementací využívá http protokol a jeho vlastnosti
- REST API jsou navrženy na základě resources, což jsou objekty, data nebo služby, na které může přistupovat klient
- Každý resurce má identifikátor – URI, které resource identifukuje např. Http://application/onlinereservation/book/10231
- Klient si se službou vyměňuje reprezentaci resource (mnoho implementací používá JSON - {"orderid":1,"ordervalue":99.90,"productid":1,"quantity":1}
- Rest api používá jednotný interface, aby byl zajištěn decoupling klienta a serveru (v HTTP implementaci je použito verb v hlavičce – GET, PUT, POST, DELETE, PATCH)
- Rest používá bezestavový model

11 REST (Representational State Transfer)

- Leonard Richardson (2008) definoval úrovně implementace rest:
- Level 0: definuje jedno URI pro všechny operace a všechny operace jsou post volání na toto URI
- Level 1: jsou definována separátní URI pro jednotlivé resouce
- Level 2: používá HTTP metody pro definici operací na resource
- Level 3: používá hypermedia odkazů (Hypertext As The Engine Of Application State - HATEOAS)




```
<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
    <link rel = "/linkrels/slot/book" uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
    <link rel = "/linkrels/slot/book" uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

11 WEB API

- WEB API je application programming interface na webovém serveru nebo webovém klientu
- Server-side WEB API sestává z jednoho nebo více endpointů
- Request-response koncept
- Datový formát JSON nebo XML
- SOAP versus rest
- Ve většině řešení je postavený na HTTP(s), využívá HTTP response kódů
- Web řešení více než SOAP web services využívají RESTful web resources

11 Webové služby, SOAP, WSDL

- SOAP (Simple Object Access Protocol) definovaný v r. 1998 - Dave Winer, Don Box, Bob Atkinson a Mohsen Al-Ghosein za podpory Microsoftu
- Základní vrstva komunikace mezi webovými službami
- Popis rozhraní WSDL - web services description language
- WSDL – popisuje
  - Abstrakci – datové typy (XML Schema), message, operation
  - Přenosový protokol – konkrétní popis formátu a endpointů služby

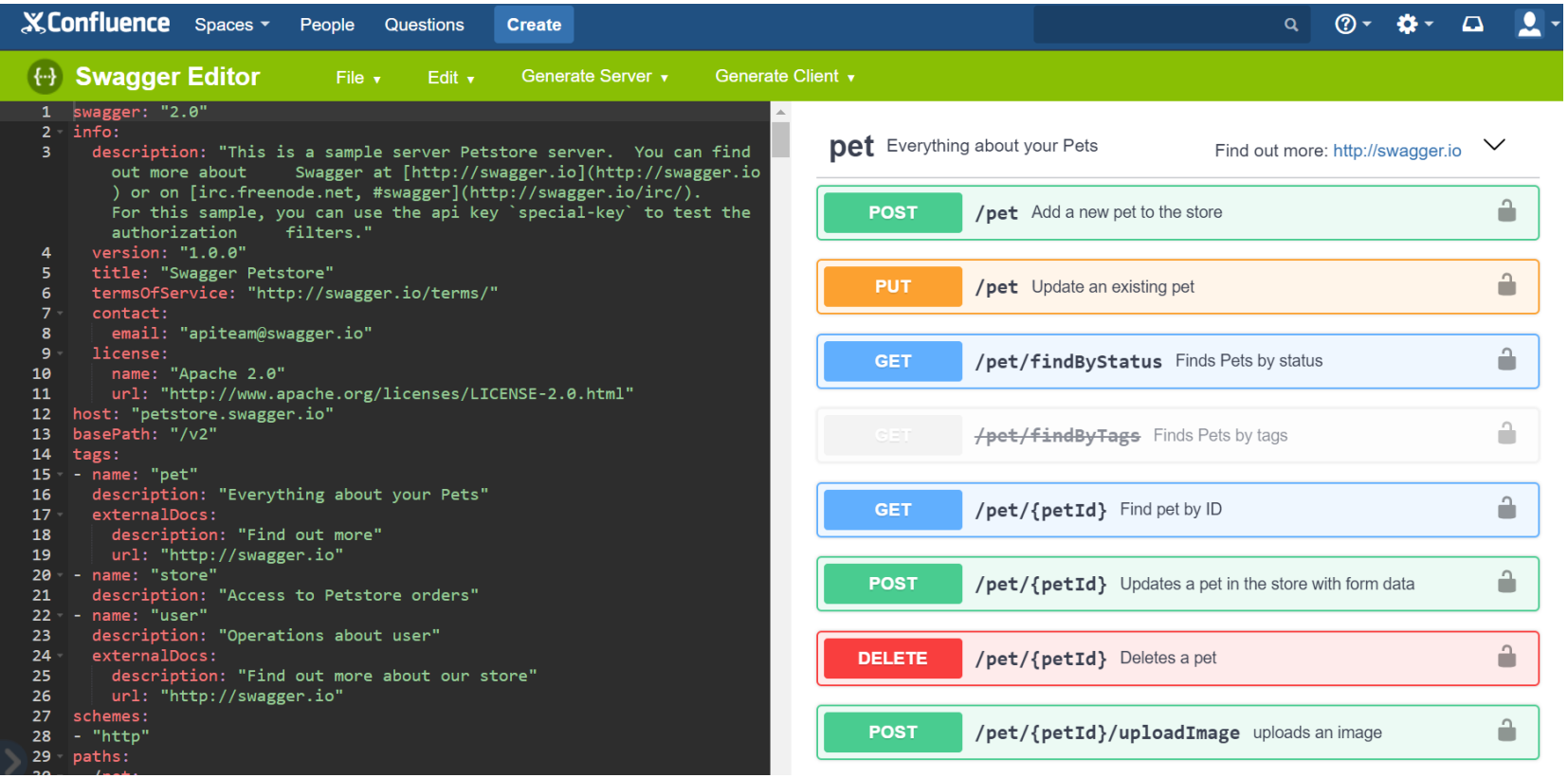
11 Open API

- OpenAPI Initiative (OAI)
  - https://www.openapis.org/
  - Zaměřuje se na vytvoření, zkvalitňování a propagaci formátu pro popis rozhraní, které bude nezávislé na dodavateli
  - Vydavatel OpenAPI Specification (https://github.com/OAI/OpenAPI-Specification)
- OpenAPI Specification
  - Definuje standard pro popis rozhraní pro REST API, nezávislé na programovacím jazyku
  - Umožňuje jak stroji, tak člověku prohlížet a porozumět službě a jejímu významu aniž by musel zkoumat zdrojový kód nebo dokonce sledovat síťový provoz
  - Dobře napsaná specifikace umožní konzumentovi služby porozumět a interagovat se vzdálenou službou s minimálním množstvím implementační logiky
  - OpenAPI definice může být použita
    - Pro generování klientského i serverového kódu pro různé programovací jazyky
    - Generování testovacích nástrojů a testovacích případů
    - Generování dokumentace
    - Validování obsahu zprávy

11 Swagger

- Framework pro definici rozhraní podle specifikace OpenAPI
- Společnost SmartBear věnovala swagger v roce 2015 iniciativě OpenAPI
- Nejen vytvoření popisu - pokrývá celý životní cyklus
  - Návrh
  - Dokumentace
  - Testování
  - Deployment
- Swagger editor – návrh
- Swagger codegen – build
- Swagger inspector – testování
- Swagger UI - dokumentace

11 Swagger editor



- Strukturovaná definice rozhraní
- JSON
- YAML
- Editační okno, okno s průběžnmým zobrazením generované dokumentace
- Možnost testování specifikace
- Možnost generování kódu serverové části
- Možnost generování kódu klientské části
- Generování do velkého množství programovacích jazyků a frameworků

11 Swagger editor - model

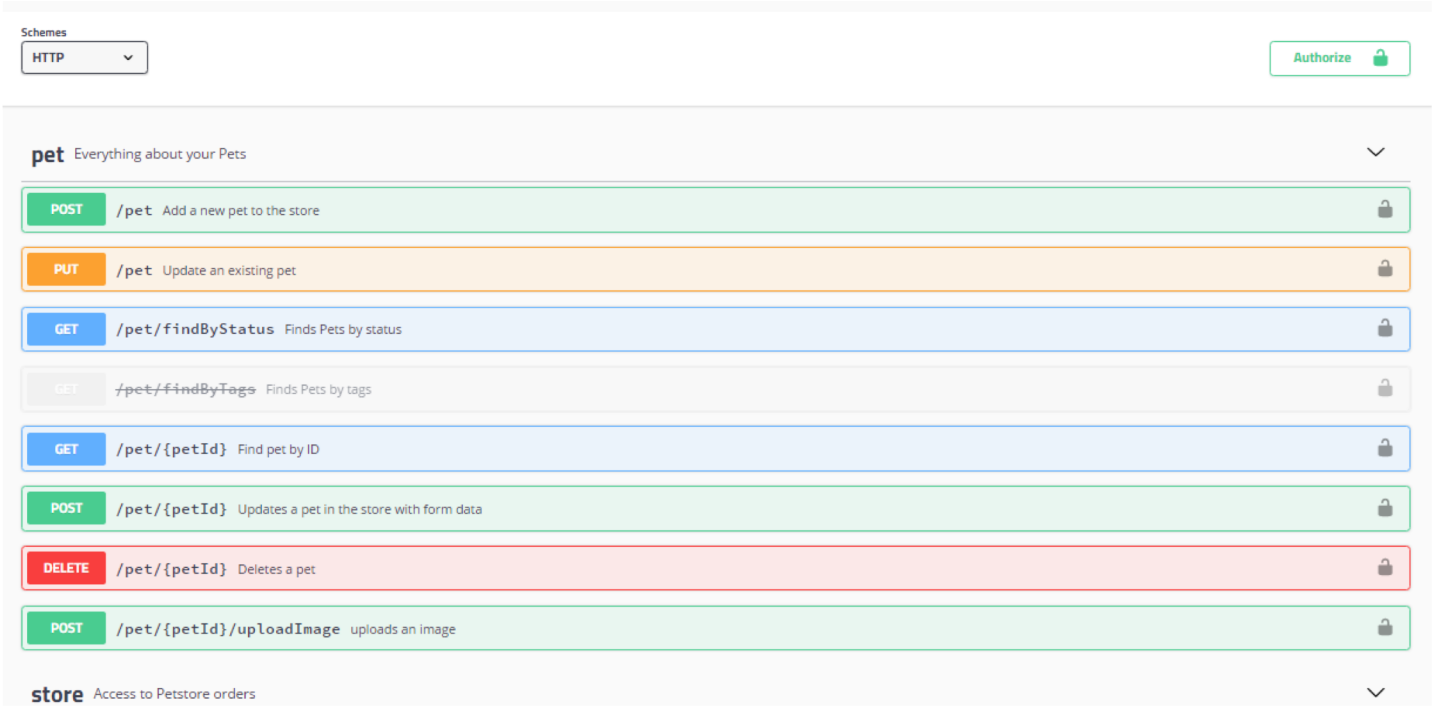
- Definice datových struktur
- Primitivní typy
  - Integer
  - String
- Datové struktury

```
Category:
    type: object
    properties:
        id:
            type: integer
            format: int64
        name:
            type: string
        xml:
            name: Category
```

11 Swagger editor – REST rozhraní

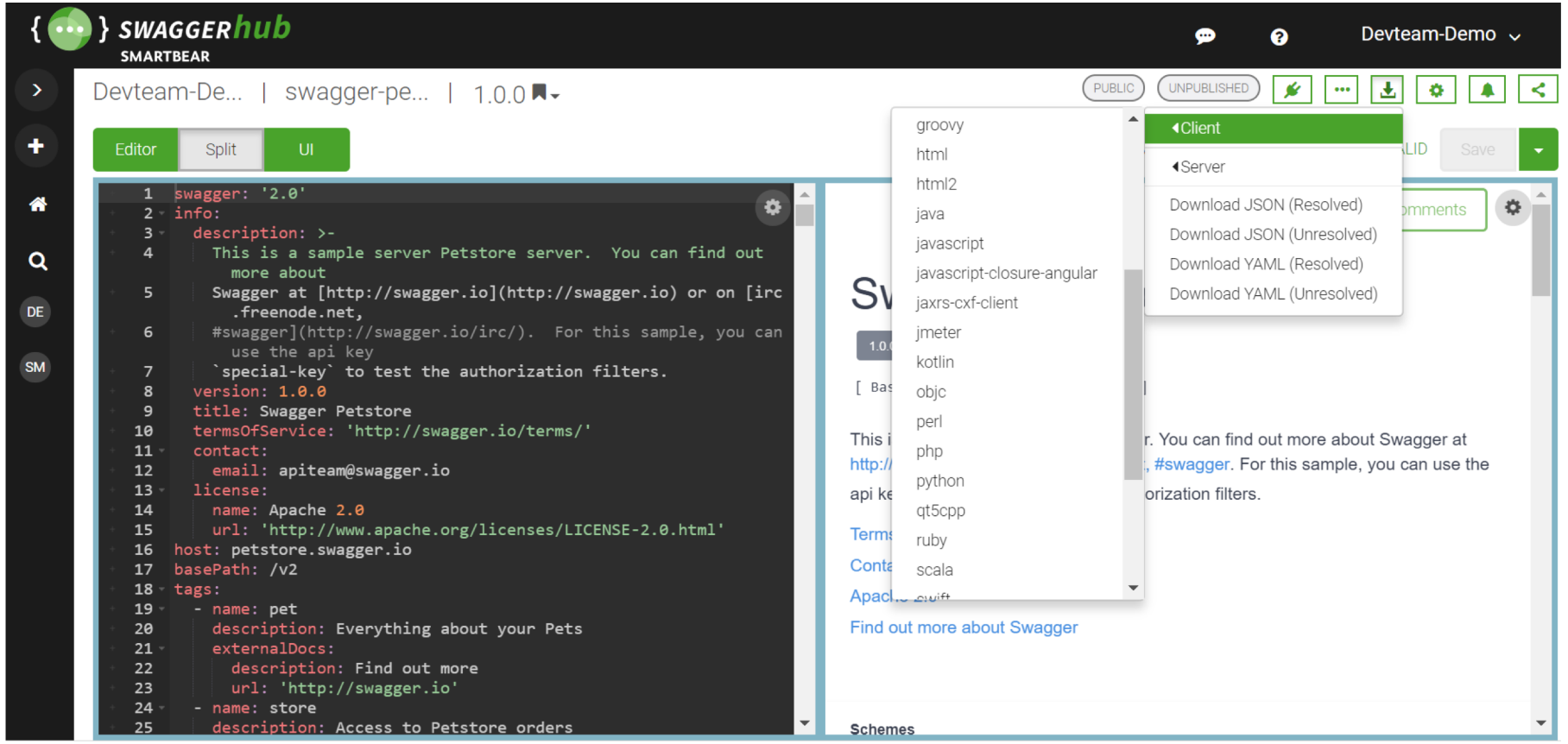
- Definice operací na resource
- Odkaz na datové struktury pomocí REF

```
responses:
    200:
        description: successful operation
        schema:
            type: array
                items:
                    $ref: '#/definitions/Pet'
```



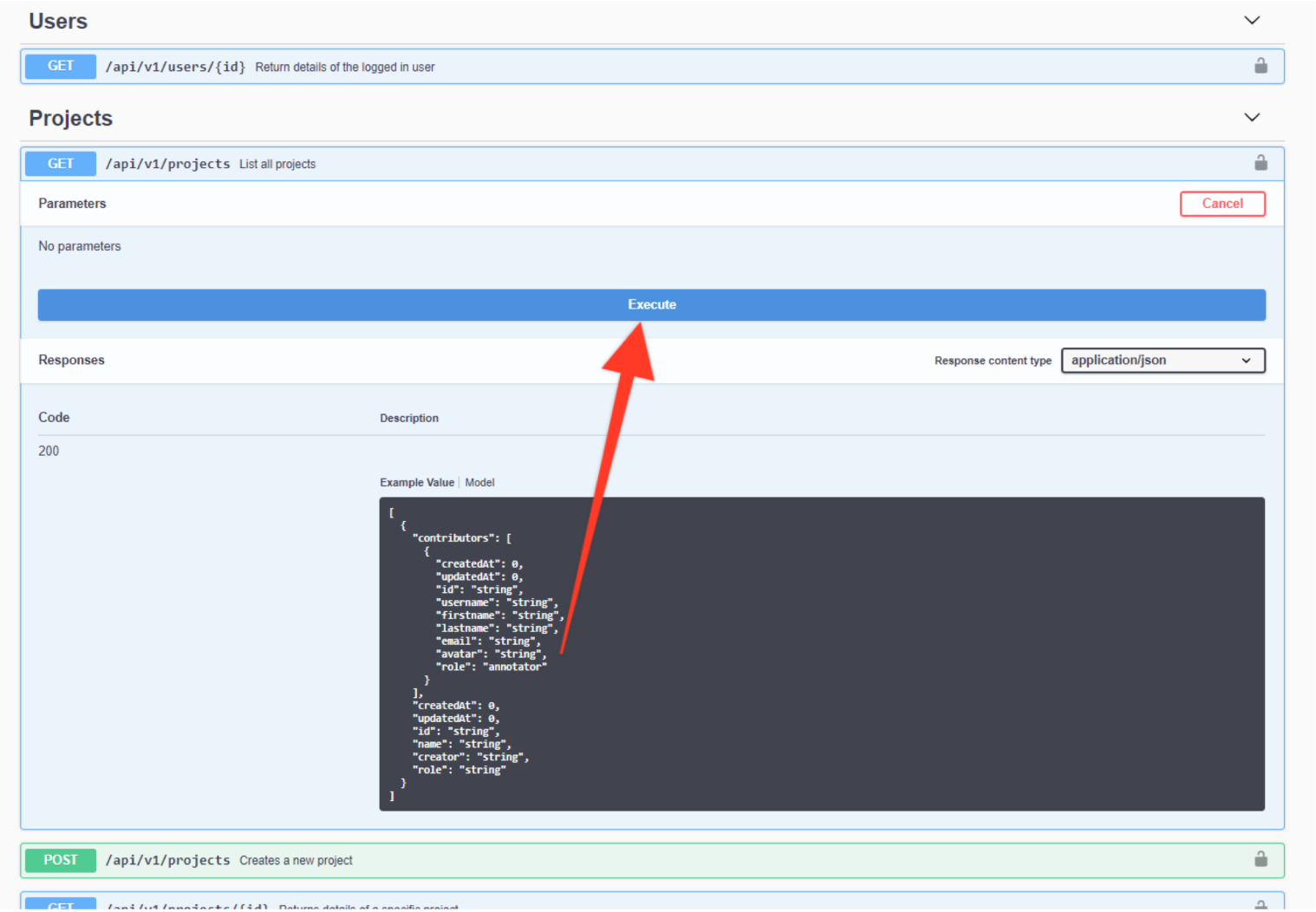
11 Swagger codegen

- Umožní generovat kostru
  - Aplikace pro implementaci služby
  - Klienta, který bude službu volat



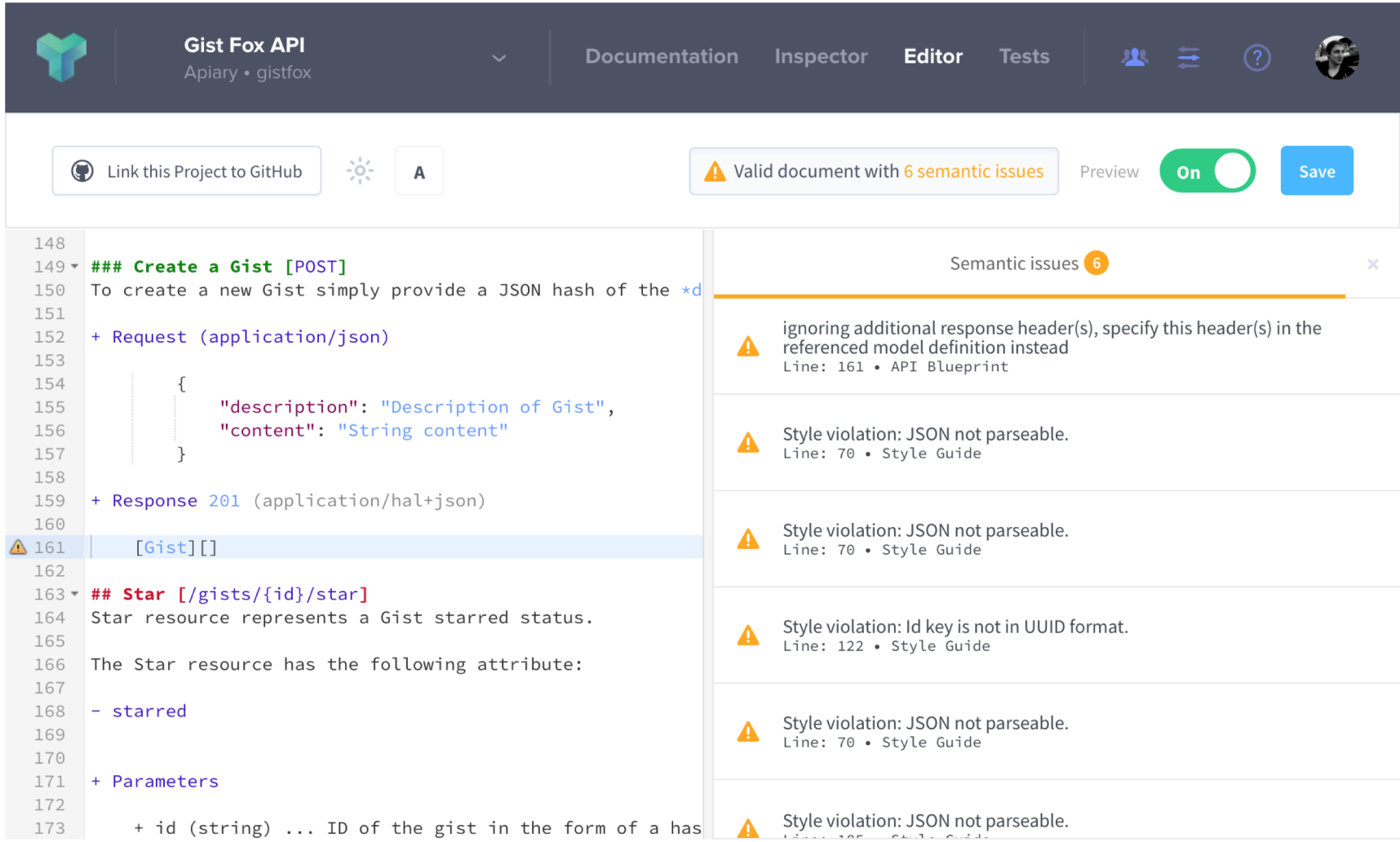
11 Swagger inspector

- Umožní spustit simulaci API. Zobrazuje REST request, response



11 APIARY

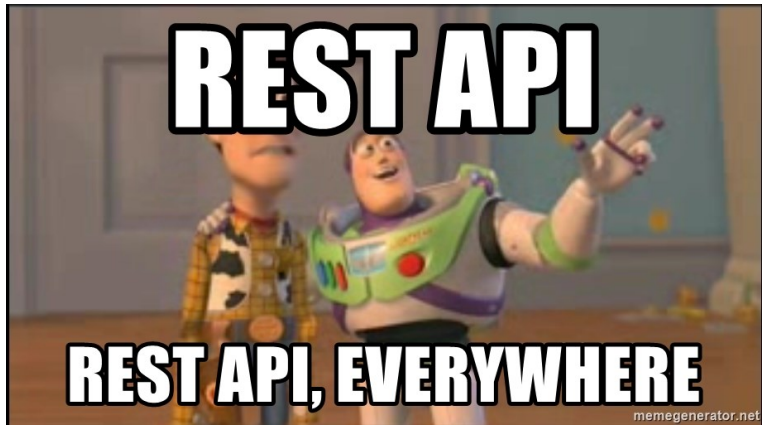
- APIARY – obdobný nástroj pro definici rozhraní
- Podobně jako Swagger – editor strukturovaného popisu rozhraní
- Dokumentace
- Testování



11 Reference

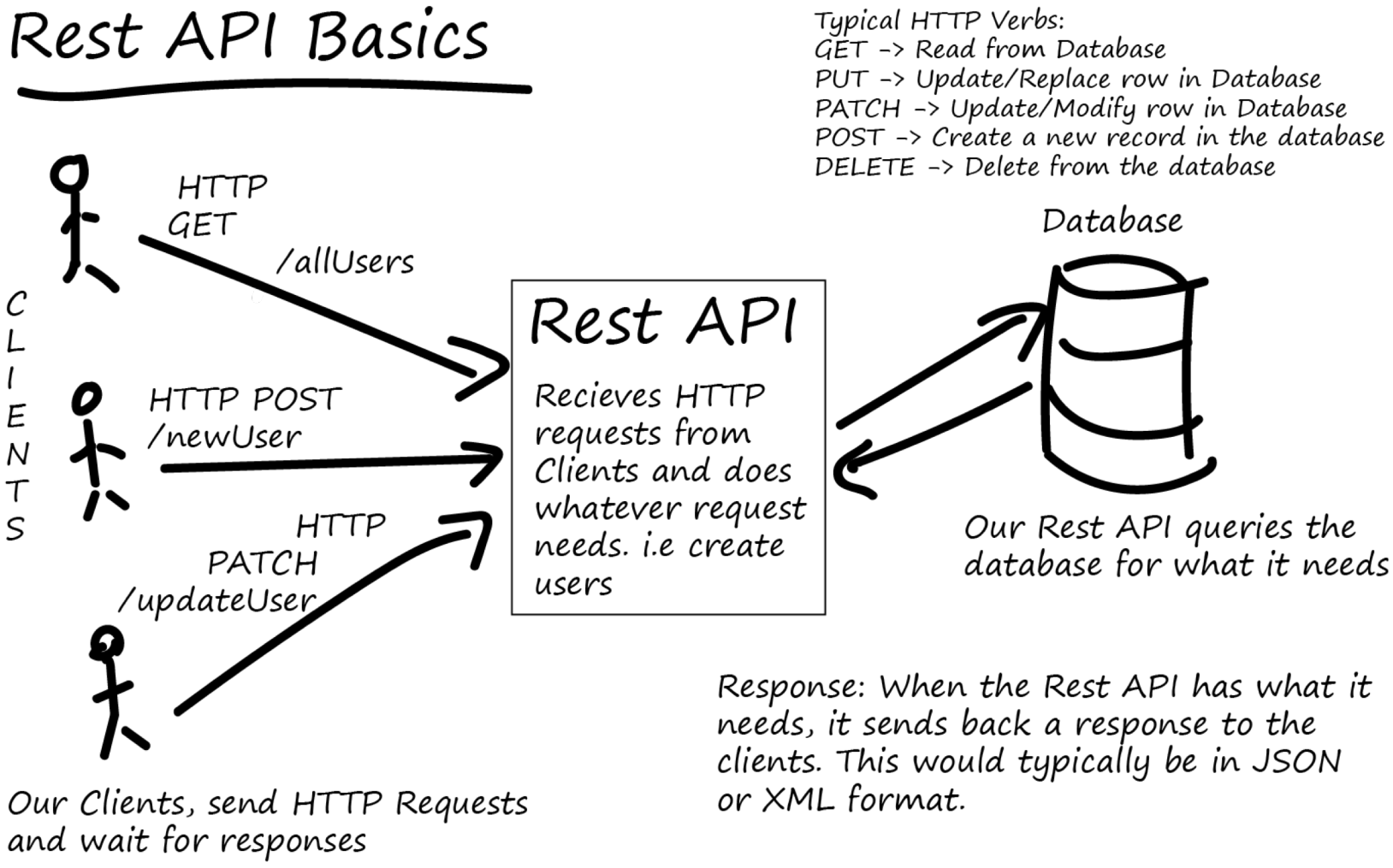
- https://ocw.mit.edu/ans7870/6/6.005/s16/classes/06-specifications/specs/
- https://ocw.mit.edu/ans7870/6/6.005/s16/classes/07-designing-specs/
- Behaviorální ekvivalence
  - http://www.cs.mcgill.ca/~hv/articles/discreteevent/p82-yucesan.pdf
- http://microservices.io/patterns/microservices.html
- https://martinfowler.com/articles/richardsonmaturitymodel.html

11 Co je REST API?



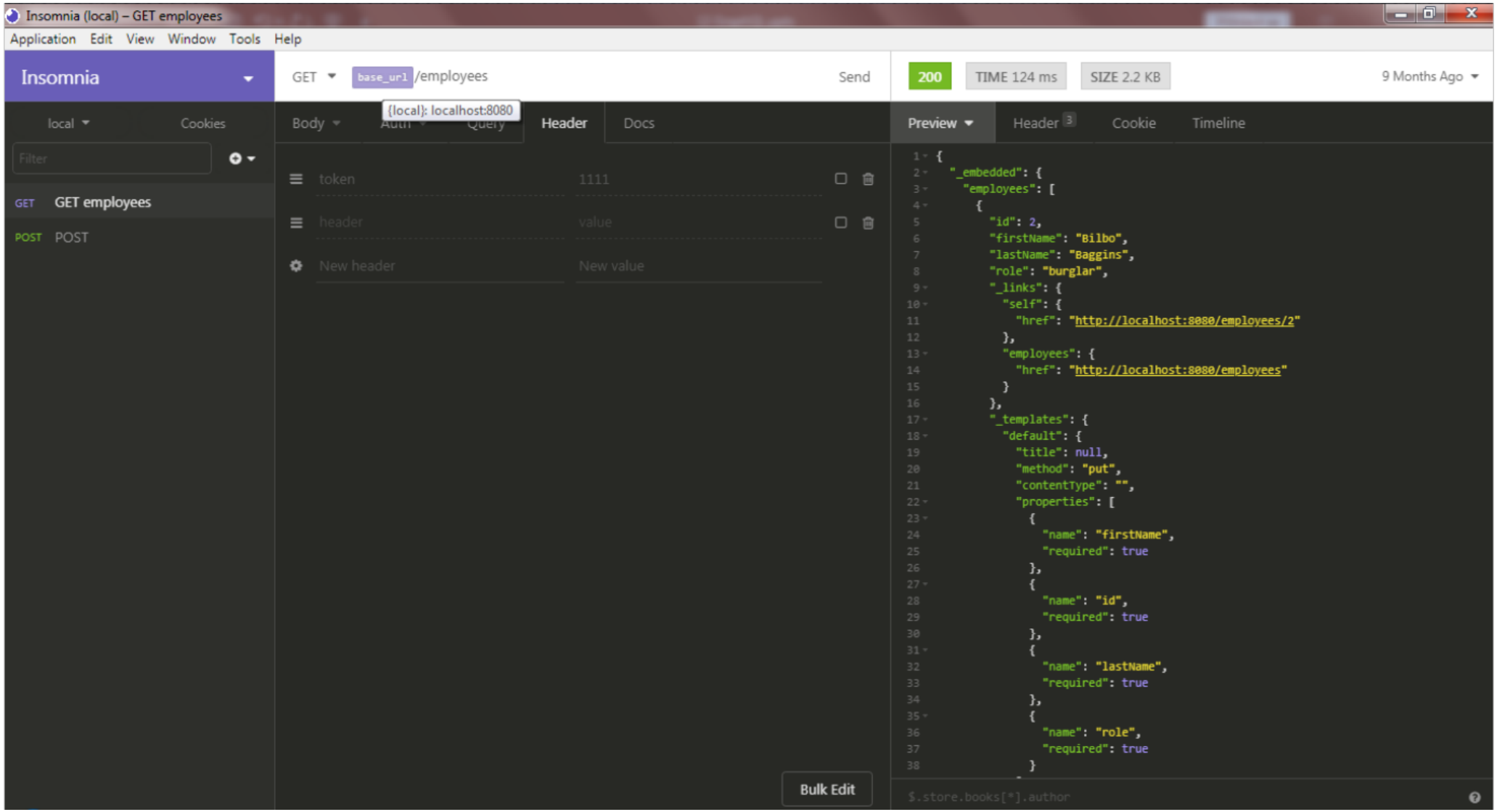
- Architektura, která umožňuje přistupovat k datům pomocí HTTP
- Je orientovaná datově, nikoli procedurálně (oproti SOAP, Cobra…)
- Metody pro přístup ke zdrojům
  - GET
  - PUT
  - POST
  - DELETE
  - PATCH

11 Rest API


Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
HTTP
GET
/allUsers
CLIENTS
HTTP POST
/newUser
HTTP
PATCH
/updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Database
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses

11 Rest API GET

11 Rest API GET



11 Co je SOAP?

- Je protokolem pro výměnu zpráv založených na XML
- Nástupce XML-RPC

- Request

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>827635</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

- Response

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productName>Čokoláda, sada 3 chutí</productName>
        <productID>827635</productID>
        <description>Čokoláda hořká, bílá a smetanová</description>
        <price>98,50</price>
        <inStock>ano</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

11 SOAP vs. REST?

SOAP

- Je to protokol, kdy 2 počítače komunikují sdílením XML dokumentu
- Pouze XML
- Čtení nemůže být cachované
- Je jako běžná desktop aplikace úzce spojená se serverem
- Je pomalejší než REST

REST

- Běží na HTTP
- Rest je architektonická služba a design pro síťově orientovanou softwarovou architekturu
- Podporuje mnoho datových formátů
- Čtení může být cachované
- Klient je jako prohlížeč – ví, jak standardizovat metody a aplikace musí sedět
- Je rychlejší než SOAP
- Užívá HTTP hlavičku pro držení metadat

11 Co je GraphQL?

- Je velmi efektivním nástupcem REST API zavedený Facebookem
- Používaný Githubem, Pinterest a Shopify
- GraphQL je dotazovací jazyk na úrovni aplikační vrstvy pro API
- Dává klientským aplikacím možnost vyžádat si přesně ta data, která potřebují a nic navíc – když operace vrací objektový graf, tak si mohu specifikovat do jaké hloubky se daný objektový graf vrací a jaké podmnožiny atributů jsou vrácené v jednotlivých entitách
- Má podporu pro stránkování
- GraphQL klienti (framework, který realizuje komunikaci mezi klientskou aplikací a serverem) mohou automaticky odpovědi na pozadí cachovat

11 GraphQL specifické requesty

```
{
  hero {
    name
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

```
{
  hero {
    name
    height
    mass
  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker",
    "height": 1.72,
    "mass": 77
  }
}
```

11 GraphQL typy

```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

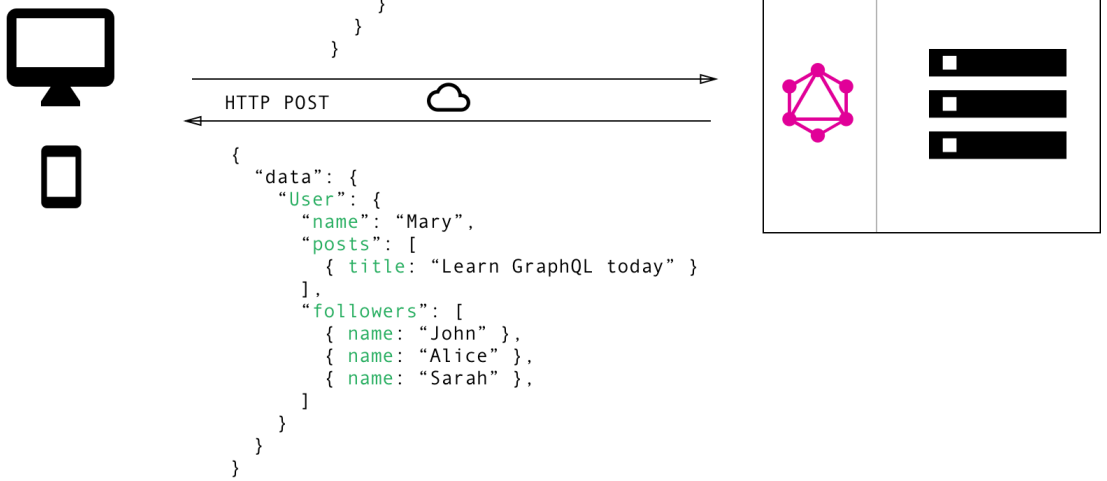
```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

11 REST vs. GraphQL



11 REST vs. GraphQL

Výkon

Zatížení	Nízké		Střední		Vysoké		
Počet připojení	1		10		100		
Celkem dotazů	10		100		1000		
	APDEX	Průměr	APDEX	Průměr	APDEX	Průměr	Propustnost
GraphQL	0.950	118 ms	0.975	290 ms	0.373	1205 ms	49.76 r/s
REST	0.750	496 ms	0.435	1254 ms	0.003	12771 ms	6.98 r/s

Vzorec

- $\frac{s+\frac{t}{2}}{c}$
- S - udává počet požadavků, které byly provedeny v uspokojivém čase, což je v tomto případě 500 ms
- t - udává počet požadavků splněných v tolerovaném čase, který je ve výpočtu nižší než 1 500 ms
- c - udává celkový počet provedených požadavků

=>

- Efektivnější komunikaci klienta se serverem má GraphQL (+ zrychlit prvotní načtení aplikace cca 40%)
- GraphQL zvládá lépe vyšší zátěž
- ..