Jak funguje REST

Tutoriál k RESTovému API

Projekt BookKeeper má back-end implementovaný na principech architektury REST.

REST = Representational State Transfer.

Viz též <https://en.wikipedia.org/wiki/Representational_state_transfer>

# Obsah

1. Nejprve pár příkladů...
2. Projekt BookKeeperBERest – Vrstva REST API
   1. Přečtení dat o uživateli (HTTP metoda GET)
   2. Aktualizace dat o uživateli (HTTP metoda PUT)
   3. Přidání dat o novém uživateli (HTTP metoda POST)
   4. Smazání dat o uživateli (HTTP metoda DELETE)

# Nejprve pár příkladů...

Jak to celé funguje?

Přestože se jedná o webovou aplikaci, tato aplikace negeneruje žádné HTML stránky, které by si mohl uživatel prohlížet. Protokol HTTP se používá pro komunikaci mezi webovým front-endem a webovým back-endem (tato RESTová appka). Místo HTML si mezi sebou posílají data ve formátu JSON.

Příklad:

FE pošle HTTP request GET <http://localhost:10789/api/users/2> s prázdným tělem.

BE pošle HTTP response se stavovým kódem 200 (OK) a v těle odpovědi budou data v JSONu:

{

"id": 2,

"username": "nasta",

"password": "abc"

}

Další příklad:

HTTP request: POST <http://localhost:10789/api/users> a v těle požadavku budou data v JSONu:

{

"username": "franta",

"password": "heslo"

}

HTTP response: stavový kód 201 (Created), v těle odpovědi je JSON:

{

"id": 17,

"username": "franta",

"password": "heslo"

}

Poslední příklad:

HTTP request: DELETE <http://localhost>:10789/api/users/1 s prázdným tělem.

HTTP response: stavový kód 200 (OK), v těle je JSON:

{

"id": 1,

"username": "igor",

"password": "xyz"

}

Toto jsou některé typické příklady volání CRUD metod nad RESTovým API.

V prvním případě se klient (FE) dožadoval dat (HTTP metoda GET) o uživateli s ID=2. Od serveru (BE) dostal tedy informace o uživateli "nasta" (ID=2).

Ve druhém případě žádal klient o založení nového uživatele jménem "franta" a nějakým heslem (HTTP metoda POST). Server odpověděl kódem 201 (nový záznam byl založen) a kompletními daty o novém uživateli (včetně jeho ID).

Ve třetím případě chtěl klient smazat uživatele s ID=1 (HTTP metoda DELETE). Server odpověděl stavovým kódem 200 (OK) a poslal i data s daným uživatelem, která si vytáhl z databáze předtím, než uživatele odstranil.

# Projekt BookKeeperBERest – Vrstva REST API

Základní principy RESTu lze ukázat na metodách třídy UserController. Jde o typické operace CRUD (create, read, update, delete), tj. přidání dat, přečtení dat, aktualizace dat, smazání dat.

## Přečtení dat o uživateli (HTTP metoda GET)

**Operace GET /api/users**

Nejprve metoda Get:

// REST API path: GET /api/users

// REST API path: GET /api/users/?username=ba

//public IEnumerable<User> Get()

[HttpGet]

public IActionResult Get([FromQuery] User user)

{

IEnumerable<User> users = \_userService.SearchUsers(user);

// HTTP status code: 200 (OK)

return Ok(users);

//return users;

}

V RESTu máme resource, zde např. tabulku uživatelů BK\_USER, které odpovídá business objekt User. Pro metody CRUD (create, read, update, delete) se používají metody POST, GET, PUT a DELETE protokolu HTTP. Operace v rozhraní REST má vždy dvě části: HTTP metodu a tzv. route (část URL).

Příklad: Lokální URL pro vývoj je <http://localhost:10789/api/users>

Operace RESTu je: GET /api/users

Jelikož je na třídě UserController definován atribut [Route("/api/users")], znamená to, že všechny relativní URL s /api/users jsou směrované do metod třídy UserController.

Na metodě Get této třídy je definován atribut [HttpGet]. To znamená, že pokud HTTP požadavek je typu GET, zavolá se právě tato metoda.

Pro URL např. <http://localhost:10789/api/users/?username=baba> se navíc "předvyplní" parametr této metody (objekt User) tak, že do jeho vlastnosti Username se uloží hodnota parametru username z query stringu této URL adresy (zde "baba").

Pro URL bez query stringu (pouze /api/users) má tato RESTová operace vrátit seznam všech uživatelů z tabulky BK\_USER. Pro URL s query stringem (např. /api/users/?username=baba) se vrátí seznam všech uživatelů, jejichž uživatelské jméno obsahuje zadaný řetězec ("baba").

V těle HTTP odpovědi realizované metodu Get ve třídě UserController bude tedy seznam uživatelů (všech nebo filtrovaný) ve formátu JSON. HTTP status kód odpovědi bude 200 (tj. OK).

**Operace GET /api/users/3**

Třída UserController obsahuje ještě jednu metodu Get, která se použije pro specializovanější RESTovou operaci (pokud požadujeme pouze jediného uživatele s určitým IDčkem).

// REST API path: GET /api/users/3

//public User Get(int id)

[HttpGet("{id:int}")]

public IActionResult Get(int id)

{

User user = new User { ID = id };

try

{

user = \_userService.LoadUser(id);

}

catch (Exception)

{

// No such user (non-existing ID).

// HTTP status code: 404 (Not Found)

return NotFound(new { id = user.ID });

}

// HTTP status code: 200 (OK)

return Ok(user);

// return user;

}

V tomto případě se kombinuje route-a celého controlleru, tj. [Route("/api/users")] s route-ou metody, tj. [HttpGet("{id:int}")], celá route-a pro tuto metodu tedy bude /api/users/{id:int}.

Například tedy /api/users/3, pokud požadujeme data o uživateli s ID=3.

Infrastruktura ASP.NET Core zajistí, že při zavolání metody Get budeme mít v jejím parametru id hodnotu z route-y (zde 3).

Další CRUD operace realizují další části třídy UserController spolu s dalšími metodami protokolu HTTP.

## Aktualizace dat o uživateli (HTTP metoda PUT)

**Operace PUT /api/users { "id": 2, "username": "nasta", "password": "noveheslo" }**

Ve třídě UserController máme metodu Put:

// REST API path: PUT /api/users

// Data is in the request body in JSON format.

// Therefore, we have an HTTP header of "Content-Type", with a value of "application/json".

[HttpPut]

public IActionResult Put(User user)

{

// ...

}

Je-li route-a /api/users (stejná jako při žádosti o seznam všech uživatelů), avšak místo metody GET použijeme PUT a v těle requestu předáme aktualizovaná data uživatele (minimálně s jeho IDčkem), infrastruktura ASP.NET Core zařídí, že se zavolá tato naše metoda Put a do jejího parametru (objekt User) se do příslušných vlastností tohoto business objektu uloží data o daném uživateli (převzatá z těla requestu v JSONu).

Pokud uživatel s předaným ID existuje a aktualizace záznamu o uživateli se podaří, metoda vrátí v response HTTP stavový kód 204 (No Content), protože veškerá užitečná data už měla být v requestu. Pokud je zadané ID uživatele neplatné, je v response HTTP stavový kód 404 (Not Found).

## Přidání dat o novém uživateli (HTTP metoda POST)

**Operace POST /api/users { "username": "franta", "password": "heslo" }**

Dále je ve třídě UserController metoda Post:

// REST API path: POST /api/users

// Data is in the request body in JSON format.

// Therefore, we have an HTTP header of "Content-Type", with a value of "application/json".

[HttpPost]

public IActionResult Post(User user)

{

// ...

}

Opět – pro stejnou route-u (/api/users), avšak tentokrát s HTTP metodou POST se vyvolá ve třídě UserController tato naše metoda Post. Do jejího parametru user se předá instance business objektu User s daty, které přišly jako JSON spolu s requestem. V těchto datech by neměl být údaj pro IDčko (případně by to mělo být ID=0), protože se jedná o nového uživatele, kterému jeho ID přidělí databáze.

## Smazání dat o uživateli (HTTP metoda DELETE)

**Operace DELETE /api/users/3**

Nakonec máme ve třídě UserController metodu Delete:

// REST API path: DELETE /api/users/3

[HttpDelete("{id:int}")]

public IActionResult Delete(int id)

{

// ...

}

U metody pro smazání dat se v REST API očekává route-a typu /api/users/3, tj. na konci je ještě jedna část s IDčkem uživatele, který se má odstranit z databáze. HTTP metoda musí být DELETE. Pokud zadané ID existuje, uživatel s tímto ID se smaže a v response se vrátí HTTP stavový kód 200 (OK). Pokud uživatel s daným ID neexistuje, metoda vrátí stavový kód 404 (Not Found).

Poznámka:

Na názvech metod v controlleru nezáleží. Jediné dvě věci, které rozhodují o tom, která metoda se vyvolá při které operaci RESTového API, jsou použitý atribut na metodě ([HttpGet], [HttpPut], [HttpPost] nebo [HttpDelete]) a route-a z URL v HTTP requestu. Zde jsme měli pouze dva typy route-y:

/api/users

nebo

/api/users/17

V URL s query stringem se query string do route-y nepočítá. Proto /api/users/?username=baba je stejná route-a jako /api/users.

Více o RESTu se píše např. zde:

<https://restfulapi.net/http-methods/>