

Sem vložte zadání Vaší práce.





**FAKULTA  
INFORMAČNÍCH  
TECHNologiÍ  
ČVUT V PRAZE**

Bakalářská práce

# **Věnná města českých královen - API pro předávání grafických modelů**

*Martin Čapek*

Katedra softwarového inženýrství

Vedoucí práce: Jiří Chludil

7. května 2019



---

## Poděkování

Děkuji především své rodině za podporu, svému vedoucímu za časté konzultace a za směřování mě k práci.



---

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. května 2019

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2019 Martin Čapek. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.*

## **Odkaz na tuto práci**

Čapek, Martin. *Věnná města českých královen - API pro předávání grafických modelů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.



---

# Abstrakt

Tato práce popisuje návrh a vývoj API v technologii Node.js.

První část je věnována analýze funkčním a nefunkčním požadavkům Editoru virtuálního historického průvodce, analýze a výběru nástrojů pro návrh API. V další části provedu návrh API s použitím nástroje Swagger, který byl vybrán v analýze, a s ohledem na současný stav databáze projektu Věnná města českých královen. Nakonec je popsán vývoj API a implementace funkčního prototypu. Tento prototyp je podroben testování.

**Klíčová slova** vývoj API, funkční prototyp, virtuální realita, historické modely budov, Node.js, Swagger

---

# Abstract

This bachelor thesis describes design and development of API in technology node.js.

First part is focused on analysis of functional and non-functional requirements of the Virtual historical guide editor, analysis and selection of API design tools. In the next part, I will design the API using Swagger tool from the analysis and with regard to the current state of the database of the project Dowry Towns of the Queens of Bohemia. Finally, API development and

functional prototype implementation are described. This prototype is being tested.

**Keywords** API development, functional prototype, virtual reality, historical models of buildings, Node.js, Swagger

---

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Cíl práce</b>	<b>3</b>
<b>2 Analýza</b>	<b>5</b>
2.1 Požadavky editoru virtuální reality . . . . .	5
2.2 Nástroje pro návrh REST API . . . . .	8
<b>3 Návrh</b>	<b>13</b>
3.1 Srovnání databáze s požadavky . . . . .	14
3.2 Pokrytí funkčních požadavků . . . . .	14
3.3 Návrh ve Swaggeru . . . . .	15
<b>4 Vývoj</b>	<b>19</b>
4.1 Použité technologie . . . . .	19
4.2 Vývojové prostředí pro OS Linux . . . . .	23
4.3 Implementace . . . . .	28
<b>5 Testování</b>	<b>31</b>
5.1 Jest . . . . .	31
5.2 GitLab . . . . .	31
<b>Závěr</b>	<b>33</b>
<b>Literatura</b>	<b>35</b>
<b>A Seznam použitých zkratk</b>	<b>37</b>
<b>B Obsah příloženého CD</b>	<b>39</b>



---

## Seznam obrázků

3.1	Proces volání API . . . . .	13
3.2	Současná databáze . . . . .	14
3.3	Popis koncového bodu ve Swaggeru . . . . .	16
3.4	Vygenerovaný koncový bod ve Swaggeru . . . . .	17
4.1	JWT [1] . . . . .	21
4.2	Proces autorizace . . . . .	22
4.3	Postman . . . . .	25
4.4	PostmanUsers . . . . .	28
4.5	Adresář implementace . . . . .	29



---

## Seznam tabulek

2.1	Metrika GitHub repozitáře [2, 3, 4, 5] . . . . .	11
3.1	Pokrytí funkčních požadavků . . . . .	15





---

# Úvod

Věnná města českých královen je rozsáhlý projekt, jehož cílem je vytvoření specializovaného historického průvodce věnnými městy a jejich městskou krajinou. Momentálně se v tomto projektu vyvíjí editor virtuální reality, který má sloužit jako nástroj historikům a jiným odborníkům, ve kterém budou moci spolupracovat na vytváření modelů, zasazování modelů do krajiny a tím vytvářet historická věnná města. Editor umožní návštěvníkům si tato města prohlížet.

V mojí práci se věnuji analýze požadavků editoru virtuální reality, návrhem API pro editor, implementací jeho prototypu a otestováním prototypu.

Uživatelé v editoru budou potřebovat API pro práci s velkým množstvím informací a grafických modelů, uložených v databázi. API bude také potřeba k řešení přístupových práv, protože některé informace jsou citlivé a s některými modely mohou manipulovat jen lidé s příslušnými právy. Výsledek této bakalářské práce bude vzorem pro vývoj API v rámci projektu Věnná města českých královen.



---

## Cíl práce

Cílem teoretické části práce je analyzovat funkční a nefunkční požadavky editoru virtuální reality. Při analýze se zaměřit na datové úložiště a způsob propagace modelů. Dále provést analýzu nástrojů pro návrh a dokumentaci API a z těchto nástrojů vybrat nejvhodnější a s jeho použitím navrhnout prototyp API, které umožní komunikaci mezi datovým úložištěm a editorem virtuální reality. Dalším cílem práce je popsat vývojové prostředí, které využiji v implementaci. V popisu budou použité technologie a příprava vývojového prostředí.

Cílem praktické části práce je implementace a otestování funkčního prototypu REST API za použití technologie Node.js a využitím technologie continuous integration.



---

# Analýza

## 2.1 Požadavky editoru virtuální reality

### 2.1.1 Funkční

Následující sekce popisuje funkční požadavky editoru virtuální reality. Tedy požadavky, které se vztahují k funkcionalitě na cílovou aplikaci. Tyto požadavky jsem vytvořil s pomocí Patrika Křepinského.

- F1 Přihlášení - V aplikaci musí být možnost přihlášení. Pokud se nepřihlásíte, můžete pokračovat jako host, který nemá žádná práva na grafické modely, tedy si je můžete pouze prohlížet.

- F2 Odhlášení

Stejně jako přihlášení je nutná možnost odhlášení. Dále je potřeba mít možnost zůstat trvale přihlášen na tomto počítači, jinak funguje automatické odhlášení po ukončení aplikace.

- F3 Práva k modelu

Model má svého autora a dále skupinu uživatelů, kteří mají některá z následujících práv:

- pouze čtení a zobrazení
- čtení a zobrazení s možností psaní komentářů?
- měnění poznámek a metadat (popis)
- editace modelu
- kopírování modelu pro svoje potřeby (povolení vytvořit na základě modelu jiný model)
- plná práva

Tuto skupinu uživatelů může autor libovolně editovat. Pokud uživatel není ve skupině znamená to, že nemá žádná práva k tomuto modelu.

- F4 Práva k projektu

Projekt je skupina modelů zasazená do lokace. Takovým projektem může být historický model města. Projekt má skupinu uživatelů, kteří mají některá z následujících práv:

- Změna lokace modelu
- Nahrávání
- Mazání
- Udělovat práva
- Seskupení

Uživatel bude moci zařadit model do skupiny. Tato funkce poslouží k lepší organizaci pracovního prostředí.

- F5 Žádosti o práva

Dále mohou uživatelé žádat o určitá práva k modelu nebo skupině modelů a k projektu. Tuto žádost mohou autoři potvrdit.

- F6 Zobrazení miniatur po přihlášení

Je třeba, aby po přihlášení měl uživatel přístup ke svým modelům. To umožní naše aplikace v podobě miniatur grafických modelů, které se po přihlášení načtou.

- F7 Třídění miniatur

Tyto miniatury si bude moci uživatel třídit podle stavu modelů, přístupových práv, skupin uživatelů, atd.

- F8 Zobrazení modelů projektu v určitém čase a počasí

Uživatel si bude moci zobrazit celý projekt v nějakém čase a procházet si ho. Bude si moci nastavovat počasí a denní dobu. Bude si ho moci procházet i napříč historií.

- F9 Ulož všechny změny v projektu

Uživatel potvrdí a uloží provedené změny na server. Po tomto uložení uvidí změny všichni kdo mají k modelu přístup. U modelu máme dva základní typy změn:

- transformace (rotace, translace, scale)
- informace o modelu (poznámka, autorství)

- F10 Vytvoř kopii modelu

Uživatel bude moci vytvořit model na základě nějakého jiného modelu (pokud k tomu bude mít právo), aby nemusel začínat od začátku.

- F11 Přidání modelu do projektu

Pokud bude moci uživatel editovat nějaký projekt, může přidávat i nové modely.

- F12 Vytvoř kopii projektu

Uživatel vytvoří kopii projektu, pokud k tomu má právo, aby mohl provádět nějaké experimentální úpravy.

- F13 Editace modelu

- smazání

Uživatel smaže model pro který už nemá žádné využití. Po kliknutí na smazání modelu se aplikace ještě zeptá, jestli si je opravdu jist, protože grafický model může představovat desítky hodin práce a může se stát, že uživatel klikne na tlačítko smazat omylem

- vytvoření

Uživateli se zobrazí prázdná pracovní plocha, kde bude moci vymodelovat nový model.

- F14 Vytvoř nový projekt

Uživatel vytvoří prázdnou pracovní plochu, kam bude moci zasazovat modely. Nahrání modelu z lokálního zařízení Uživatel nahraje model z disku a může ho uložit na server.

### 2.1.2 Nefunkční

Tato sekce se zabývá nefunkčními požadavky na REST API. Tedy požadavky, které se zaměřují na nároky cílové aplikace na software a to například z hlediska bezpečnosti, spolehlivosti, či výkonu.

- N1 Rychlost odezvy

Uživatel nesmí na obdržení nebo aktualizování grafického modelu čekat. Je třeba, aby server reagoval rychle.

- N2 Datová nenáročnost při komunikaci

Je třeba minimalizovat množství dat posílané přes API, abychom docílili rychlosti a zbytečně nepřetěžovali spojení s koncovým uživatelem.

- N3 Bezpečnost

Určité koncové body vyžadují oprávnění, jelikož jejich zavoláním se předávají nebo přepisují citlivá data. To se vyřeší tím, že při přihlášení dostane uživatel token, kterým se bude autorizovat u volání citlivých koncových bodů.

- N4 Rozšiřitelnost

API musí umožňovat případné rozšíření o další funkcionality. Také musí být řádně zdokumentováno, aby umožnilo hladší průběh rozšiřování.

- N5 Použité technologie

Bylo zadáno, že se bude vyvíjet v technologii Node.js za využití modulu Express.

## 2.2 Nástroje pro návrh REST API

V této sekci provedeme výběr nejvhodnějšího nástroje, který budeme používat v další kapitole Návrh. Návrh API nám bude rovněž sloužit jako dokumentace, jelikož v návrhu je popsáno co API dělá a jak. Tato dokumentace slouží rozšiřitelnosti API viz nefunkční požadavek.

Bylo vybráno pět nejrozšířenějších kandidátů, které nabízí své služby zdarma. Tyto kandidáty ohodnotím výběrovou metodou, na jejímž základě vyberu nejvhodnějšího. Výběrová metoda se bude skládat z ohodnocení nástroje z hlediska následujících metrik:

- Dokumentace nástroje

Je nutnou podmínkou, aby byl nástroj řádně zdokumentován, protože k nástroji budu přistupovat jako laik. Tudíž veškerá funkcionality, která nebude pokryta v dokumentaci jakoby neexistovala.

- Funkčnost

Popis API záleží na specifikaci, kterou nástroje využívají. Budu ho hodnotit třemi základními kritérii:

- Koncové body
- Zabezpečení s JWT
- Definice objektů, které bude API předávat

Návrh API by měl jít vizualizovat do podoby dokumentace.

- GitHub repozitář

Na repozitář se budeme dívat z ohledu oblíbenosti (na GitHubu mohou uživatelé dávat hvězdičky repozitářům), posledních commitů a řešených problémů



- Fórum

Můžeme narazit na různé problémy, které lze vyřešit na fóru, nebo najít konverzaci na fóru s podobnou tématikou. Budeme tedy hodnotit jak je fórum obsáhlé a jak je živé (poslední příspěvek, rychlost odpovědí).

Popíši každý z vybraných nástrojů a ohodnotím je z pohledu vybraných metrik a na konec dám samostatně metriku GitHub repozitáře.

### 2.2.1 Swagger

Swagger je sada nástrojů postavená na OpenAPI Specifikaci. OpenAPI Specifikace, dříveji Swagger Specifikace, je sada pravidel, které sémanticky popisují API. Nástroje:

- Swagger editor - editor umožňující psát OpenAPI Specifikaci
- Swagger UI - vizualizuje API dokumentaci
- Swagger Codegen - vygeneruje kód clientské nebo serverové části

Metriky:

- Dokumentace

Dokumentace je rozsáhlá. Abyste mohli psát dokumentaci pro API ve Swaggeru, musíte ovládat OpenAPI Specifikaci. Ta je zde detailně popsána a při jejím čtení jsem si připadal jako když se učím nový programovací jazyk (jsou zde popsány datové typy, dědění a polymorfismus). Dále je zde obsáhlá dokumentace SwaggerHubu, což je webová aplikace, která přináší všechny základní vlastnosti Swaggeru. Dokumentace má tutoriály a díky SwaggerHubu si je můžete rovnou vyzkoušet.

- Funkčnost

Splňuje požadovanou funkčnost.

- Fórum

Fórum<sup>1</sup> je aktivní a má i GitHub Issues.

[6]

---

<sup>1</sup><https://community.smartbear.com/t5/Swagger-Open-Source-Tools/bd-p/SwaggerOSTools>

### 2.2.2 Apicurio Studio

Apicurio Studio je open-source editor pro OpenAPI specifikaci. Musíte se přihlásit a vaše návrhy REST API se ukládají do vašeho repozitáře na GitHubu, GitLabu nebo Bitbucketu, což se hodí v případě, když na návrhu pracujete společně v týmu. V editoru můžete přepínat mezi interaktivní vizualizací návrhu a jeho JSON/YAML definicí.

- Dokumentace

Dokumentace není nijak obsáhlá a není z ní zjevné, co tento nástroj dokáže a co naopak neumí. Odkaz na vyzkoušení online verze editoru nefungoval (chyba DNS\_PROBE\_FINISHED\_NXDOMAIN).

- Funkčnost

Splňuje požadovanou funkčnost.

- Fórum

Fórum pouze v podobě GitHub Issues.

[7]

### 2.2.3 API Workbench

API Workbench, narozdíl od výše zmíněných nástrojů, pracuje s RAML. RAML je akronym RESTful API Modeling Language a je to typ souboru podobný YAML, který specifikuje návrh API. API Workbench je v beta verzi a funguje jako balíček pro Atom, což je textový editor.

- Dokumentace

Dokumentace je stručná, přehledná, výstižná. Obsahuje tutoriál jak si napsat návrh pro jednoduché API, ale není zde vidět jak vypadá vizualizovaný návrh. Pro vizualizaci si musíte nainstalovat další nástroj API Designer.

- Funkčnost

Splňuje požadovanou funkčnost.

- Fórum

Fórum<sup>2</sup> je aktivní a má i GitHub Issues.

[8]

---

<sup>2</sup><https://community.smartbear.com/t5/Swagger-Open-Source-Tools/bd-p/SwaggerOSTools>

### 2.2.4 Apiary

Api editor je založeno na API Blueprint, což je podobně jako OpenAPI Specifikace sada pravidel popisující API. API Blueprint specifikace má vlastní formát APIB, který má syntaxi podobnou Markdownu a využívá MSON<sup>3</sup>. Apiary disponuje interaktivní dokumentací, webový editor má možnost přepnutí do tmavého režimu. Dále zajišťuje propojení s GitHub repozitářem.

- Dokumentace

Dokumentace pro apiary editor je jedna stránka bez tutoriálu. Obsahuje jak vypadá editor, popis základní funkcionality a seznam klávesových zkratk. Má tedy nejmenší dokumentaci ze všech nástrojů.

- Funkčnost

Splňuje požadovanou funkčnost.

- Fórum

Nic jako fórum jsem nenašel, pouze FAQ.

[9]

### 2.2.5 GitHub repozitář

Metriku GitHub repozitář jsem zapsal do tabulky Metrika GitHub repozitáře. Nástroj Apiary nemá GitHub repozitář, tudíž ho nemůžu hodnotit. U Swaggeru jsem hodnotil jak repozitář pro editor tak pro UI a to tím způsobem, že hvězdičky jsem zprůměroval, poslední commit je nejposlednější commit obou repozitářů a Issues jsem sečetl. Data jsou ze dne 14.4.2019.

Tabulka 2.1: Metrika GitHub repozitáře [2, 3, 4, 5]

	Swagger	Apicurio Studio	API Workbench
Hvězdičky	9 799	310	219
Poslední commit	14.4.2019	12.4.2019	3.4.2018
Issues otevřené/uzavřené	417/4723	153/456	133/350
Další poznámky		one man show Smíšené různé jazyky - JavaScript 32.7% Java 30.9% TypeScript 22.9%	

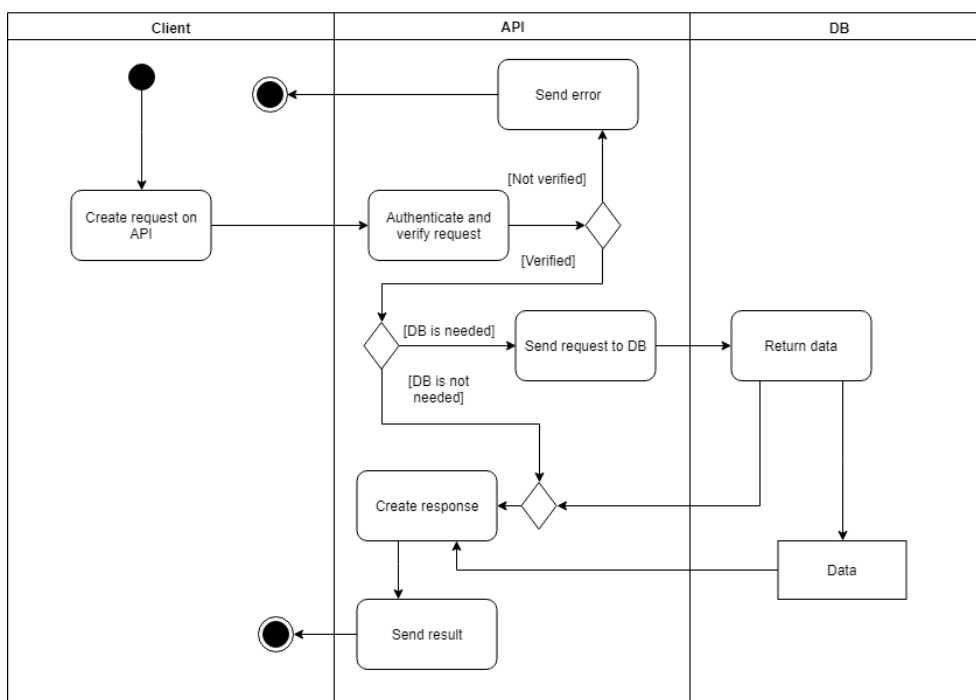
<sup>3</sup><https://github.com/apiaryio/mson>

### 2.2.6 Výsledek výběru nástroje pro návrh API

Vítězem tohoto pomyslného souboje se stal **Swagger**. Vyhrál v kategoriích Dokumentace a GitHub repozitář. Fórum vyhovuje našim požadavkům a funkcionalitu splnily všechny nástroje.

## Návrh

V této kapitole provedu návrh REST API na základě analýzy funkčních požadavků. Průběh volání API je zobrazen na obrázku 3.1.



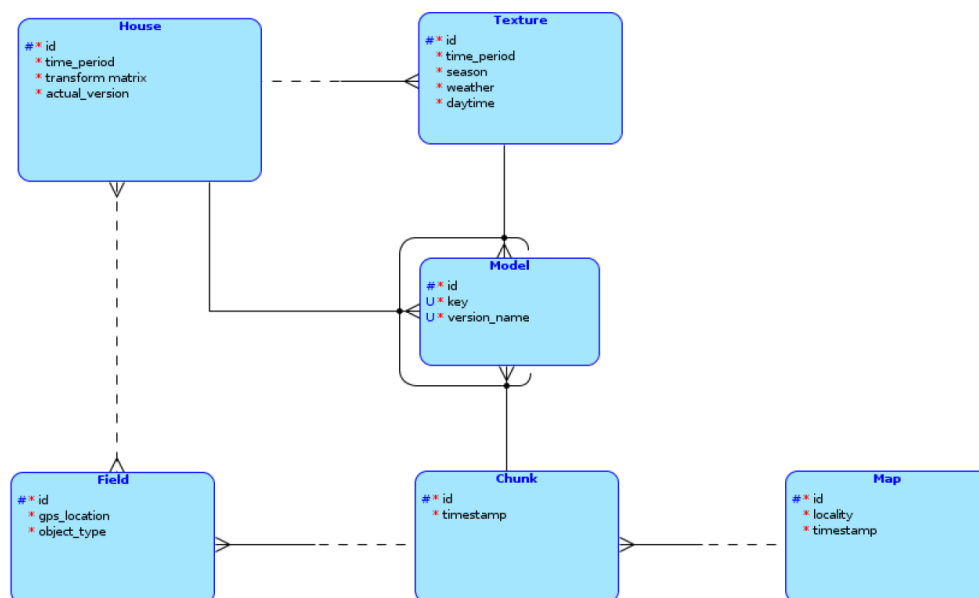
Obrázek 3.1: Proces volání API

Z obrázku je patrné, že API musí znát databázi, aby na ní mohlo vytvářet požadavky, a editor musí znát naše API, aby ho mohl správně volat. Databáze projektu Věnná města českých královen je zachycena na obrázku 3.1.

Nacházíme se ve stavu, kdy známe schéma databáze i jak vypadají požadavky

### 3. NÁVRH

---



Obrázek 3.2: Současná databáze

na API, takže nám nic nebrání v jejím návrhu. Předem se ale musíme ujistit, že databázové schéma pokrývá požadavky.

#### 3.1 Srovnání databáze s požadavky

Editor virtuální reality požaduje správu práv a uživatelských účtů. U entity modelu chybí informace o právech uživatelů a chybí entita uživatelů. Dále Editor požaduje miniatury modelů, které se ve schématu nenacházejí. Bude tedy potřeba provést změnu v databázovém schématu pro plnou podporu Editoru.

#### 3.2 Pokrytí funkčních požadavků

Pokrytí funkčních požadavků jsem zachytil v tabulce 3.1, kde ke každému požadavku je uveden koncový bod API. Funkční požadavky Odhlášení a Třídění miniatur nebudeme řešit v tomto návrhu, protože se o ně postará Editor. Odhlášení se provede v aplikaci odstraněním JWT. Třídění miniatur následuje po Zobrazení miniatur po přihlášení, miniatury tedy budou uloženy v aplikaci a o jejich třídění se postará aplikace.

Tabulka 3.1: Pokrytí funkčních požadavků

Funkční požadavek	API endpoint
Přihlášení	GET /login
Smaž model	DELETE /model/{modelId}
Vytvoř nový model	PUT /model
Editovat práva k modelu	POST /model/{modelId}
Editovat práva k projektu	POST /project/{projectId}
Seskupení	POST /group/{groupId}
Žádosti o práva	GET /rights/model/{modelId}
Zobrazení miniatur po přihlášení	GET /miniatures
Zobrazení modelů projektu v určitém čase a počasí	GET /model/{modelId}/time/{time}/weather/{weather}
Ulož všechny změny v projektu	POST /project/{projectId}
Vytvoř kopii modelu	PUT /model
Přidání modelu do projektu	POST /project/{projectId}
Vytvoř nový projekt	PUT /project
Vytvoř kopii projektu	PUT /project
Nahrání modelu z lokálního zařízení	PUT /project

### 3.3 Návrh ve Swaggeru

Na základě této tabulky jsem vytvořil návrh ve Swaggeru. Uvedu zde příklad zapsání jednoho koncového bodu, zbytek najdete v elektronické příloze TBD. Na obrázku 3.2 je zapsaný koncový bod v OpenAPI Specifikaci a na obrázku 3.3 je vidět jeho vizualizace.

### 3. NÁVRH

---

```
paths:
  /login:
    post:
      summary: Login user
      security: []
      requestBody:
        content:
          application/json:
            schema:
              type: object
              required:
                - username
                - password
              properties:
                username:
                  type: string
                  nullable: false
                  example: Batman
                password:
                  type: string
                  nullable: false
                  example: Batman's passwd

      responses:
        '200':
          description: successful operation
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
                    example: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
                      .eyJ1c2VySwQiOiJlMDhmODZhZi0zNWRLTQ4ZjItOGZlYi1jZWYzOTA0NjYwYmQifQ
                      .-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
        '400':
          $ref: '#/components/responses/BadRequest'
        '401':
          description: Login failed wrong user credentials
```

Obrázek 3.3: Popis koncového bodu ve Swaggeru



### 3.3. Návrh ve Swaggeru

The image shows a Swagger UI interface for a **POST /login** endpoint. The endpoint is described as "Login user".

**Parameters:** No parameters are defined for this endpoint.

**Request body:** The content type is set to **application/json**. The example value is a JSON object:

```
{
  "username": "Batman",
  "password": "Batman's passwd"
}
```

**Responses:** The following table lists the expected responses:

Code	Description	Links
200	<b>successful operation</b> The response body is <b>application/json</b> . Example Value: <pre>{   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInp0eSI6IjYyLjE2VyswQioiLCJpdHm00ZzhZi0ZNMhLTQ4ZjItOGZhYi1jZWYzOTA0NjYyYmQifQ.-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM" }</pre>	No links
400	<b>Bad Request</b>	No links
401	<b>Login failed wrong user credentials</b>	No links

Obrázek 3.4: Vygenerovaný koncový bod ve Swaggeru



# Vývoj

V této kapitole se budu věnovat technologiím, které použiji při implementaci, přípravě vývojového prostředí a vlastní implementaci.

## 4.1 Použité technologie

### 4.1.1 Rest API

REST, neboli Representational State Transfer, je architektonický styl, který umožňuje přistupovat k datům a využívá již existující HTTP. REST implementuje čtyři základní metody, známé pod akronymem CRUD, jsou to Create (vytvoření dat), Retrieve (získání požadovaných dat), Update (změnu) a Delete (smazání). Dokáže vrátit XML, JSON, YAML nebo jiné formáty podle klientových požadavků. [?]

V naší implementaci budeme používat JSON, rozhodnuto bylo na základě autorových zkušeností a článku [?].

### 4.1.2 Node.js

Node.js je open-source multiplatformní JavaScriptové prostředí postaveno na Chrome V8 JavaScript enginu. Primární účel Node.js je tvorba serverové části webových aplikací, které vychází z paradigmatu "JavaScript everywhere".

Node.js využívá událostmi řízenou architekturu a neblokující I/O operace. Tento návrh optimalizuje výkon a škálovatelnost programů s častými požadavky na I/O operace.

Jádro celého Node.js tvoří smyčka událostí, která běží na jednom vlákně. Ta podporuje desettisíce souběžných připojení bez nutnosti neustálého přepínání kontextu díky neblokujícímu I/O. Nedochází zde k žádnému zamykání, tudíž nemusíme mít obavy z deadlocku systému. [10]

### 4.1.3 npm

Node.js využívá správce balíčků npm, jehož pomocí můžeme obecně instalovat i spravovat závislosti a spouštět skripty. Npm se chlubí tím, že je největším balíčkovacím správcem a aktuálně obsahuje skoro 800 000 balíčků. [11]

### 4.1.4 Express

Express je framework pro Node.js, který nám umožňuje jednoduše napsat webovou aplikaci, či API. Těší se velké popularitě a podle [12] je na něm závislých 21873 balíčků.

### 4.1.5 JWT

JWT (JSON Web Token) je standart, který zajišťuje bezpečný přesun informací zapsaných v JSON.

JWT se skládá ze 3 částí:

- Header  
obsahuje typ tokenu a šifrovací algoritmus.
- Payload  
obsahuje naše informace a může obsahovat ještě dodatečné informace jako expirace tokenu, vydavatel atd.
- Signature  
obsahuje zakódované předešlé části a secret – náš tajný klíč.

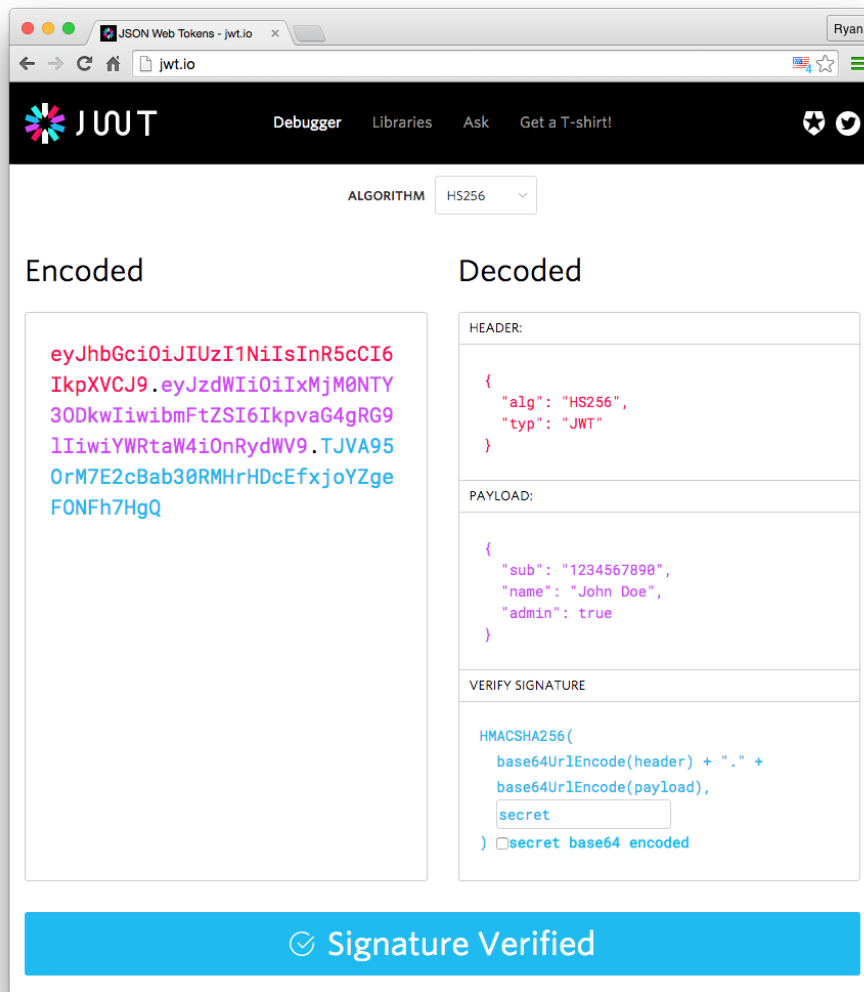
V naší aplikaci využijeme tuto technologii pro ověřování totožnosti uživatelů. Naše aplikace vytvoří JWT po přihlášení a uživatel se jím bude nadále identifikovat. [13]

### 4.1.6 TSLint

TSLint je nástroj pro statickou analýzu kódu TypeScriptu. Detekuje potenciální chyby a udržuje jednotné formátování kódu.[?]

### 4.1.7 dotenv-safe

Tato knihovna umožňuje pracovat s proměnnými prostředí. Proměnné prostředí budeme zapisovat do souboru ".env"v kořenovém adresáři a tento soubor přidáme do gitignore souboru, protože zde budou citlivé informace. Zároveň bude existovat soubor ".env.example", který bude vypadat stejně jako soubor ".env", jenom nebude obsahovat hodnoty proměnných. Soubor ".env.example"slouží jako vzor k vytvoření ".env". [?]

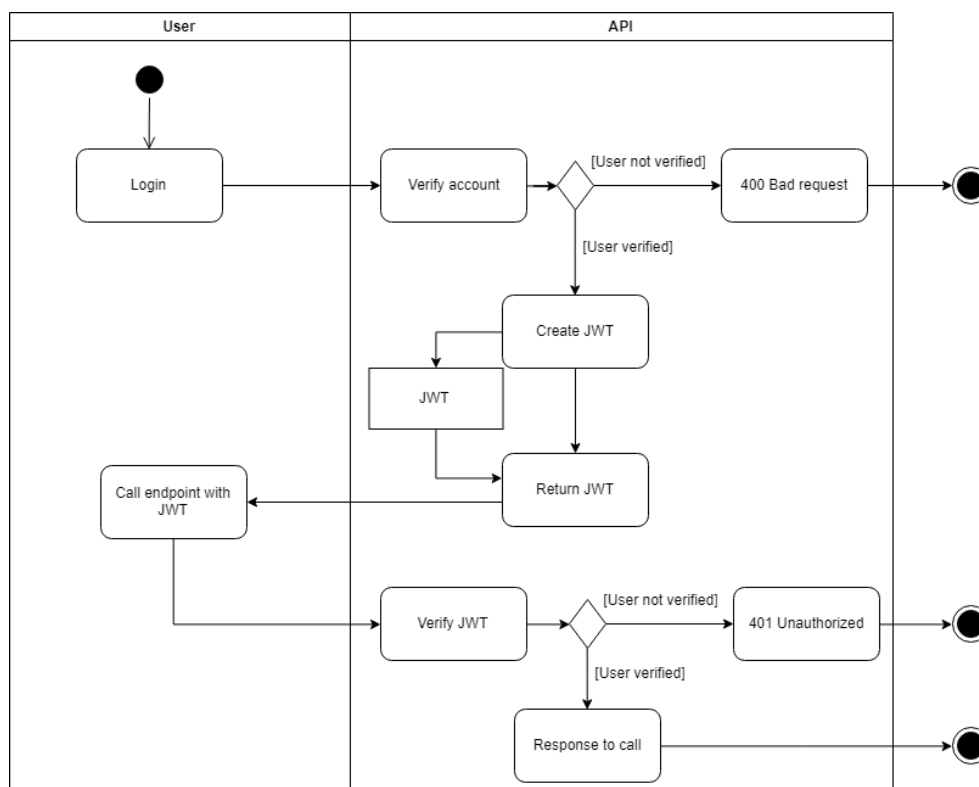


Obrázek 4.1: JWT [1]

#### 4.1.8 Implementační jazyk

Pro vývoj byl vybrán jazyk TypeScript, který je kompilovatelný do JavaScriptu. Tento jazyk je vyvíjen firmou Microsoft a jeho hlavní myšlenkou je být nadstavbou JavaScriptu, která přidává statické typování a další funkcionalitu.

## 4. VÝVOJ



Obrázek 4.2: Proces autorizace

### 4.1.9 PostgreSQL

Realizátory datové vrstvy v projektu Věnná města českých královen byla vybrána databáze PostgreSQL.

PostgreSQL je objektově-relační databázový systém pod MIT licencí. Je pověstný svou spolehlivostí a vysokou bezpečností. Na PostgreSQL wiki<sup>4</sup> lze nalézt rozsáhlý seznam dostupného open-source software, sloužícího k administraci a monitoringu PostgreSQL databází. Nejrozšířenější a velmi přehledný je pgAdmin. [?]

### 4.1.10 TypeORM

Jak již název napovídá jedná se o objektově relační mapování, což nám umožní jednodušeji pracovat s databází. TypeORM podporuje PostgreSQL a má npm balíček, který budeme moci využít při implementaci v Node.js. [14]

<sup>4</sup>[https://wiki.postgresql.org/wiki/Community\\_Guide\\_to\\_PostgreSQL\\_GUI\\_Tools](https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools)

### 4.1.11 GitLab

todo

## 4.2 Vývojové prostředí pro OS Linux

V této sekci se budu věnovat přípravě vývojářského prostředí v Node.js a popíši zde jak zprovoznit API. Stejným postupem byla provedena i Implementace.

Nejprve si musíme nainstalovat Node.js a správce balíčků npm.

```
sudo apt install nodejs -y
sudo apt install npm -y
```

V implementaci je použita verze Node.js 8.10.0 a npm 3.5.2. Vaši verzi si můžete zkontrolovat pomocí následujících příkazů.

```
nodejs -v
npm -v
```

Spustíme příkaz `npm init -y` v adresáři, kde chceme mít náš projekt. Ten nám zde vygeneruje jednoduchý `package.json`. `Package.json` nám definuje náš projekt, obsahuje například název projektu, popis, licenci, autora, závislosti na balíčcích. Můžeme ho upravovat v textovém editoru nebo příkazy:

```
npm set init.author.email "example-user@example.com"
npm set init.author.name "example_user"
npm set init.license "MIT"
```

Příklad souboru `package.json` najdeme v el. příloze. Dále budeme instalovat balíčky. Následujícím příkazem se nám balíček stáhne i s knihovnami, na kterých závisí, do složky `node_modules` a taky se nám zapíše do `package.json` ve formátu „název: verze“.

```
npm i "název balíčku"
```

U příkazu `npm i` můžeme přidat přepínač `-D`, tímto označíme balíček, který není potřeba pro běh samotné aplikace, takto označíme balíčky potřebné pro vývoj a testování např. `tslint` (nástroj pro statickou analýzu kódu), `jest` (nástroj pro testování). Nainstalujeme si balíčky potřebné pro naši aplikaci.

```
npm i express typescript dotenv-safe
```

Dále si nainstalujeme balíčky pro statické typování viz <https://github.com/DefinitelyTyped/DefinitelyTyped>.

```
npm i -D @types/node @types/express
```

## 4. VÝVOJ

---

Vytvoříme si `tsconfig.json` pro konfiguraci TypeScript. Inspirovat se můžete v elektronické příloze TBD. Dokumentaci najdete na stránce <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>.

V `package.json` přidáme:

```
"scripts": {  
  "start": "ts-node src/app.ts",  
}
```

Vytvoříme soubor v našem adresáři `src/app.ts`, který bude vypadat následovně.

```
import * as express from 'express';  
import * as path from 'path';  
  
require('dotenv-safe').config({  
  path: path.resolve(__dirname, '../.env'),  
  sample: path.resolve(__dirname, '../.example.env'),  
});  
  
const app = express();  
  
app.get('/', (req, res) => res.send('Hello World!'));  
app.listen(process.env.LISTEN_ON);
```

Zbývá ještě specifikovat náš port. V souboru `.example.env` stačí mít pouze `"LISTEN_ON"`.

```
echo "LISTEN_ON=4000" > .example.env && cp .example.env .env
```

Teď máme vše připraveno, abychom spustili naše jednoduché API s jedním koncovým bodem. Nastartujeme aplikaci, která bude poslouchat na portu 4000 příkazem:

```
npm start
```

Můžeme otestovat funkčnost v Postmanu<sup>5</sup>, tak že zavoláme koncový bod, který nám vrátí Hello World! Příklad na obrázku 4.3.

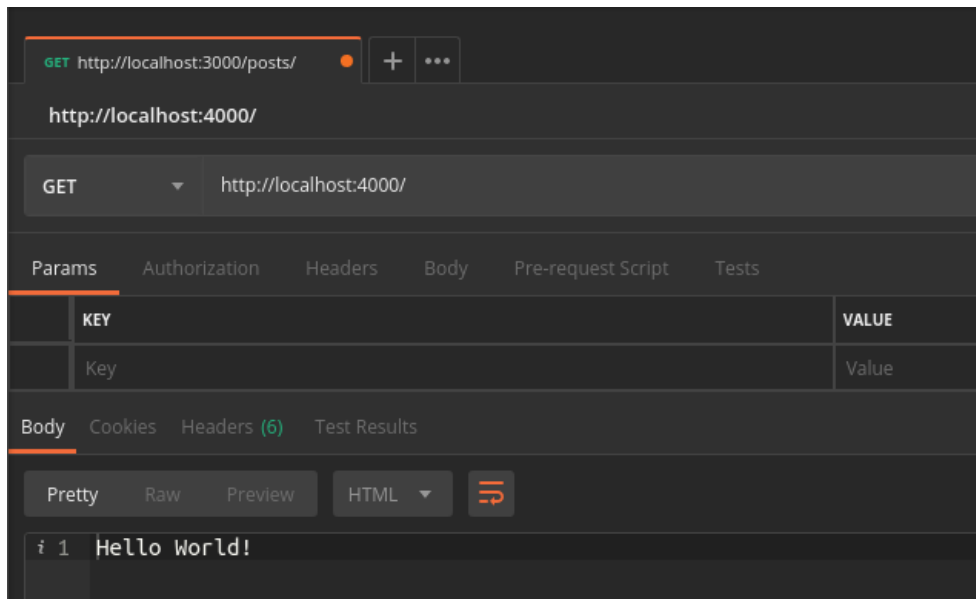
### 4.2.1 Databázové připojení s TypeORM

Začneme s vytvořením souboru `ormconfig.json` v našem kořenovém adresáři, kde bude uložena konfigurace databázového připojení. Můžete se inspirovat souborem `ormconfig.json` v el. příloze, zde je použita PostgreSQL databáze. V této konfiguraci budeme definovat naše schéma. To zdefinujeme pomocí popsání entit. V `ormconfig.json` bude zapsáno, kde se naše entity nachází.

---

<sup>5</sup><https://www.getpostman.com>





Obrázek 4.3: Postman

```
"entities": ["src/entity/**/*.ts"]
```

Vytvoříme si soubor src/entity/User.ts.

```
import {Entity, PrimaryGeneratedColumn, Column} from "typeorm";
```

```
@Entity()
```

```
export class User {
```

```
  @PrimaryGeneratedColumn()
```

```
  id: number;
```

```
  @Column({
```

```
    unique: true,
```

```
  })
```

```
  username: string;
```

```
  @Column({
```

```
    unique: true,
```

```
  })
```

```
  email: string;
```

```
  @Column()
```

```
  password: string;
```

```
}
```

Připíšeme do souboru `src/app.ts` vytvoření databázového připojení.

```
createConnection().then(async () => {  
    const app = express();  
    app.use(bodyParser.json());  
    app.listen((process.env.LISTEN_ON && +process.env.LISTEN_ON) || 8080);  
    console.log(`> Ready on http://${process.env.LISTEN_ON}`);  
}).catch(error => console.log("TypeORM connection error: ", error));
```

Naší databázi můžeme nechat prázdnou a TypeORM nám vytvoří schéma podle našich entit. Pokud máte funkční databázi a správně připojení, mělo by se vám v konzoli po spuštění aplikace zobrazit:

```
> Ready on http://4000
```

### 4.2.2 Volání endpointu

V této sekci si vytvoříme koncový bod, který nám vrátí všechny uživatele.

Nejprve si vytvoříme adresář `src/controller`, zde se budou nacházet koncové body. Vytvoříme si zde soubor `user.t`, kde zadefinujeme koncový bod `getAllUsers` a funkci `createBasicUsers`, která nám vytvoří uživatele v databázi.

```
import {Request, Response} from "express";  
import {getManager, getRepository} from "typeorm";  
import { User } from '../entity/User';  
  
export async function getAllUsers(request: Request, response: Response) {  
    const postRepository = getManager().getRepository(User);  
  
    const users = await postRepository.find();  
  
    response.send(users);  
}  
  
function createUser(username: string, email: string, password: string): User {  
    const user = new User();  
    user.username = username;  
    user.email = email;  
    user.password = password;  
    return user;  
}
```

```
export async function createBasicUsers(): Promise<void> {
  await getRepository(User).save(createUser("Aman", "a@man.cz", "1234"));
  await getRepository(User).save(createUser("Batman", "batman@example.com", "1234"));
}
```

Vytvoříme soubor `src/routes.ts`, kde se budou nacházet všechny cesty pro koncové body.

```
import { getAllUsers } from "../controller/user";

export const AppRoutes = [
  {
    path: "/users",
    method: "get",
    action: getAllUsers
  },
];
```

Nakonec přidáme do `src/app.ts` následující kód, který zavolá funkci pro přidání uživatelů a přidá koncové body z `src/routes.ts`.

```
import { Request, Response } from "express";
import { AppRoutes } from "../routes";
import { createBasicUsers } from "../controller/user";

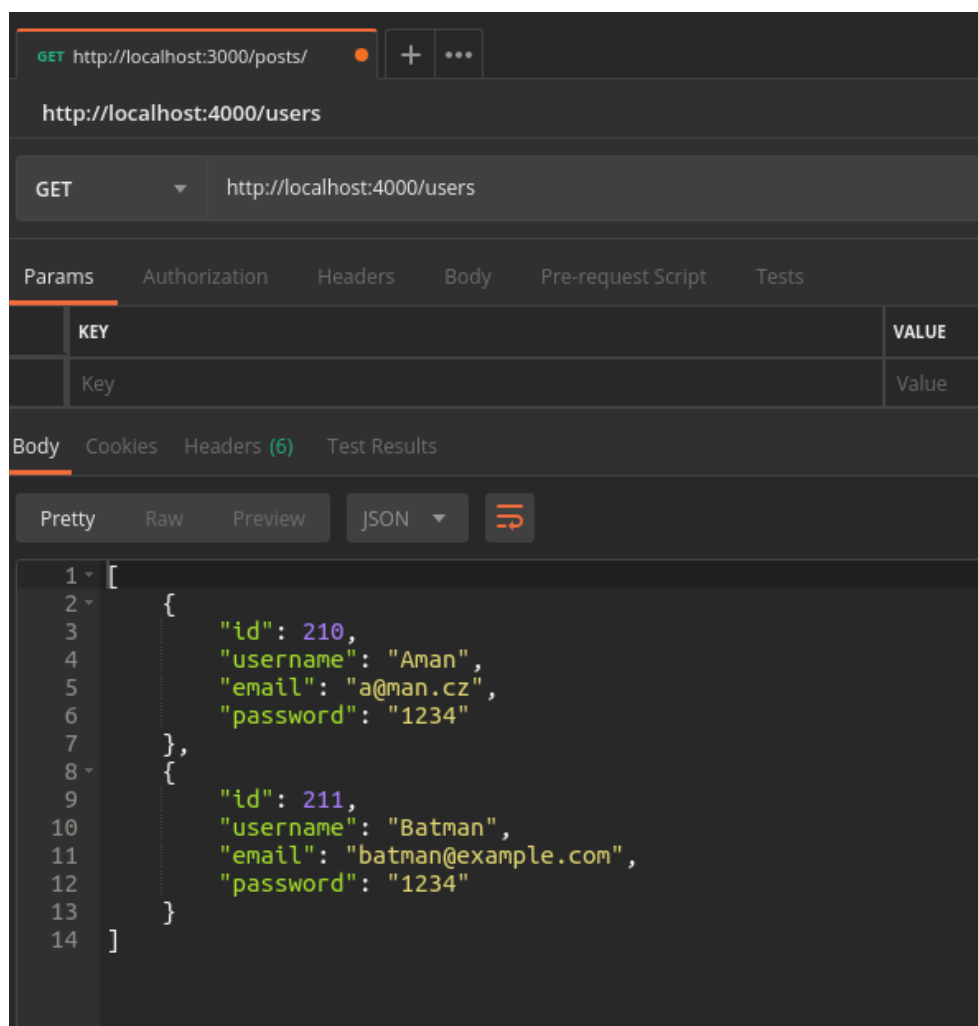
await createBasicUsers();
AppRoutes.forEach(route => {
  app[route.method](route.path, (request: Request, response: Response, next: Function) => {
    route.action(request, response)
      .then(() => next)
      .catch(err => next(err));
  });
});
```

Nastartujeme naši aplikaci a funkčnost můžeme otestovat v Postmanu<sup>6</sup>. Zaolání koncového bodu by mělo vypadat tak, jak je zachyceno na obrázku 4.4.

---

<sup>6</sup><https://www.getpostman.com>

## 4. VÝVOJ



Obrázek 4.4: PostmanUsers

### 4.3 Implementace

Zde popíšeme jak vypadá prototyp Rest API pro Editor virtuální reality. Naše implementace navazuje na předchozí sekci Vývojové prostředí pro OS Linux.

Jelikož se na databázi v projektu Věnná města českých královen ještě pracuje, bylo rozhodnuto, že v prototypu budou pouze entity User a Model.

Implementovány jsou koncové body:

- POST `/login`

Vyžaduje jméno a heslo uživatele. Vrací JWT, pomocí kterého se uživatel autorizuje. JWT expiruje po 4 hodinách od vytvoření.

- DELETE /model/:modelID  
Vyžaduje autorizaci. Vymaže danou entitu z databáze.
- PUT /model  
Vyžaduje autorizaci. Vytvoří entitu v databázi.
- POST /model/:modelID  
Vyžaduje autorizaci. Změní danou entitu v databázi.
- GET /model/:modelID  
Vyžaduje autorizaci. Vrátí danou entitu.

Autorizace probíhá pomocí JWT (viz sekce JWT), který se pošle v hlavičce požadavku.

Struktura adresáře je zachycena na obrázku 4.5.

```

src
├── controller..... akce koncových bodů
├── entity..... databázové schéma
├── app.ts..... hlavní spustitelný soubor
├── routes.ts..... specifikace URI koncových bodů
├── .example.env..... názvy proměnných prostředí
├── .gitignore..... ignorované soubory pro verzování
├── .gitlab-ci.yml..... skript pro GitLab CI
├── ormconfig.json..... připojení na databázi
├── package.json..... definování projektu a jeho závislostí
├── tsconfig.json..... konfigurace TypeScriptu
└── tslint.json..... nastavení linteru

```

Obrázek 4.5: Adresář implementace

Prototyp pokrývá pouze část návrhu v el. příloze - Swagger. Prototyp ovšem může sloužit jako základní struktura, na kterou se dodělá zbytek schématu a koncových bodů.

Hesla uživatelů se momentálně ukládají v plaintextu, což není ideální.



## Testování

Pro ověření funkčnosti našeho prototypu provedeme jednotkové testování. Jednotka v našem případě znamená koncový bod. Pro testování našeho API použijeme framework Jest.

### 5.1 Jest

Jest je framework pro testování JavaScriptových aplikací. Hodí se pro testování Node.js a typescriptu. Pro náš prototyp jsem vytvořil dva konfigurační soubory pro testování, které se liší pouze tím, že v jednom se analyzuje pokrytí kódu testy. Testy bez pokrytí se použít příkazem:

```
npm test
```

A testy s pokrytím:

```
npm run test-with-coverage
```

Pokrytí kódu testy se nám uloží do složky coverage. Můžeme si v internetovém prohlížeči otevřít soubor `coverage/lcov-report/index.html` viz el. příloha, ve kterém vidíme statistiku pokrytí kódu testy a můžeme si interaktivně procházet adresář a dívat se na pokrytí konkrétních souborů.

Axios?

### 5.2 GitLab

Pro verzování teoretické i praktické části jsem používal GitLab. GitLab nabízí nástroj pro průběžnou integraci, který dovoluje při každém commitu spustit skript který je zapsaný v souboru pojmenovaném `“.gitlab.yml”` uloženém v kořenovém adresáři projektu. My budeme chtít, aby se nám po commitu aplikace sestavila a spustili se na ní naše testy.





---

## Závěr

V práci byla provedena analýza funkčních požadavků editoru virtuální reality, nefunkčních požadavků na API, technologií použitých při implementaci, nástrojů pro návrh API, z těchto nástrojů byl vybrán nejvhodnější kandidát na základě hodnotících metrik a tím je Swagger. S tímto nástrojem byl proveden návrh API a tím i jeho dokumentace. Tato dokumentace může sloužit jako vzor komukoli, kdo chce dokumentovat API. V návrhu se vyskytl problém, že současná databáze Věnných měst českých královen nepostačuje požadavkům editoru virtuální reality, byl proveden návrh bez ohledu na současný stav databáze avšak implementace prototypu pokryla jen část návrhu. Tento prototyp je připraven na další rozšíření. Dále tato práce obsahuje popstup vývoje API v Node.js, který může být inspirací budoucím řešitelům implementace API v Node.js. Nakonec byla provedena implementace funkčního prototypu API z předešlého návrhu a jeho otestování.



---

## Literatura

- [1] JSON Web Token Introduction. *JSON Web Token Introduction*, [Online; cit. 2015-10-31]. Dostupné z: <https://jwt.io/introduction>
- [2] GitHub: *swagger-api/swagger-ui*. [Online; cit. 2019-04-14]. Dostupné z: <https://github.com/swagger-api/swagger-ui>
- [3] GitHub: *swagger-api/swagger-editor*. [Online; cit. 2019-04-14]. Dostupné z: <https://github.com/swagger-api/swagger-editor>
- [4] GitHub: *Apicurio/apicurio-studio*. [Online; cit. 2019-04-14]. Dostupné z: <https://github.com/Apicurio/apicurio-studio>
- [5] GitHub: *mulesoft/api-workbench*. [Online; cit. 2019-04-14]. Dostupné z: <https://github.com/mulesoft/api-workbench>
- [6] *Swagger.io*. [Online; cit. 2019-04-197]. Dostupné z: <https://swagger.io>
- [7] *Apicurio Studio*. [Online; cit. 2019-04-197]. Dostupné z: <https://apicur.io>
- [8] *Apiworkbench.com*. [Online; cit. 2019-04-197]. Dostupné z: <http://apiworkbench.com/docs>
- [9] *Apiary.io*. [Online; cit. 2019-04-197]. Dostupné z: <https://apiary.io>
- [10] Node.js: *Node.js*. [Online; cit. 2019-04-197]. Dostupné z: <https://nodejs.org/en/about>
- [11] *XXX*. [Online; cit. 2019-04-197]. Dostupné z: <http://www.modulecounts.com>
- [12] *Gist*. [Online; cit. 2019-04-197]. Dostupné z: <https://gist.github.com/anvaka/8e8fa57c7ee1350e3491>

## LITERATURA

---

- [13] *JSON Web Tokens*. [Online; cit. 2019-04-197]. Dostupné z: <https://jwt.io>
- [14] *Typeorm.io*. [Online; cit. 2019-04-197]. Dostupné z: <https://typeorm.io>

## Seznam použitých zkratk

**API** Application Programming Interface

**CI** Continuous Integration

**IDE** Integrated Development Environment

**I/O** Input/Output

**JSON** JavaScript Object Notation

**npm** Node.js package manager

**OAS** OpenAPI Specification



## Obsah přiloženého CD

	readme.txt .....	stručný popis obsahu CD
	exe .....	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis .....	zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	text .....	text práce
	thesis.pdf .....	text práce ve formátu PDF
	thesis.ps .....	text práce ve formátu PS