FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Věnná města českých královen - API pro ditribuci grafických modelů
Student:	Martin Čapek
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Věnná města Českých Královen je platforma zaměřující se na zobrazení historického prostředí v mobilních a webových aplikacích pomocí prvků rozšířené a virtuální reality.
1. Analyzujte funkční a nefunkční požadavky editoru virtuální reality. Zaměřte se především na datové úložiště a způsob propagace modelů.
2. Dle předchozích požadavků navrhněte prototyp API, který umožní komunikaci mezi datovým úložištěm a editorem virtuální reality.
3. Implementujte prototyp REST API rozhraní za použití technologie Node.js, využijte technologie continuous integration.
4. Prototyp podrobte automatizovaným a integračním testům.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Věnná města českých královen - API pro předávání grafických modelů

Martin Čapek

Katedra softewarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

15. května 2019

Poděkování

Děkuji především své rodině za podporu, svému vedoucímu za časté konzultace a za směrování 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 15. května 2019 .....

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 REST API pro editor virtuální reality.

První část je věnována analýze funkčním požadavkům editoru virtuální reality a nefunkčním požadavkům na API, analýze nástrojů pro návrh API a stanovení výběrové metodiky pro tyto nástroje. Z této metriky nám vyjde nejlépe ohodnocený nástroj Swagger.

V návrhové části je vytvořen doménový model databáze, který vychází ze současného stavu databáze projektu Věnná města českých královen. Funkční požadavky editoru virtuální reality jsou pokryty koncové body. Provede se návrh API s použitím nástroje Swagger a popíší se technologie potřebné pro vývoj.

Dále je popsán vývoj API, tutoriál pro vytvoření jednoduchého REST API a implementace funkčního prototypu.

Nakonec je funkční prototyp podroben automatizovaným a integračním testům.

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

Abstract

This work describes the design and development of the REST API for the virtual reality editor.

The first part is devoted to the analysis of the functional requirements of the virtual reality editor and the non-functional requirements of the API, the analysis of API design tools and the determination of the selection methodology for these tools. This selection gives us the best rated tool Swagger.

In the design part I will create a domain model of the database, which is based on the contemporary state of the database of the project . I create endpoints that cover the functional requirements of the virtual reality editor. I will design an API using Swagger and describe the technologies needed for development.

In the design part, a domain model of the database is created, which is based on the contemporary state of the database of the project Dowry Towns of the Queens of Bohemia. The virtual reality editor's functional requirements are covered by endpoints. An API design is performed using Swagger and the technologies needed for development are described.

Next is described API development, tutorial for creating simple REST API and implementation of functional prototype.

Finally, the functional prototype is subjected to automated and integration tests.

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

Obsah

Seznam obrázků

Seznam tabulek

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 garfických 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 a to především pro Věnná města českých královen.

Kapitola 1

Cíl práce

Cíl teoretické části práce je rozdělen na dva celky, analýzu a návrh.

Analýza se provede nad funkčními a nefunkčními požadavky editoru virtuální reality. Dále se analyzují nástroje pro návrh a dokumentaci API a z těchto nástrojů se vybere nejvhodnější a s jeho použitím se navrhne prototyp API, které umožní komunikaci mezi datovým úložištěm a editorem virtuální reality. V návrhu se zaměříme na datové úložiště a způsob propagace modelů.

Cílem praktické části práce je vývoj a otestování funkčního prototypu REST API za použití technologie Node.js a využítím technologie continuous integration. Spoučástí vývoje je popis technologií, které se využijí ve vývoji, postup pro přípravu vývojového prostředí a implementace funkčního prototypu.

Kapitola 2

Analýza

2.1 Požadavky editoru virtuální reality

2.1.1 Funkční

Následující sekce popisuje funkční požadavky editoru virtuální reality. To jsou požadavky, které se vztahují k funkcionalitě pro cílovou aplikaci. Tyto požadavky jsem vytvořil ve spolupráci s Patrikem Křepinským.

- **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 moct 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 moct uživatel třídit podle stavu modelů, přístupových práv, skupin uživatelů, atd.

- **F8 Zobrazení modelů projektu v určitém čase a počasí**

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

- **F9 Vytvoř kopii modelu**

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

- **F10 Vytvoř nový projekt**

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

- **F11 Vytvoř kopii projektu**

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

- **F12 Přidání modelu do projektu**

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

- **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 moct vymodelovat nový model.

- **F14 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)

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 vyvýjet 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 N4.

Byly vybráni čtyři nejrozšířenější kandidáti, které nabízí své služby zdarma. Tyto kandidáty ohodnotím výběrovou metodou, ze které nám vyjde nejvhodnější nástroj pro naše potřeby. 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 funkcionalita, která nebude pokryta v dokumentaci jakoby neexistovala.

- Funkčnost

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

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

  Dále by návrh API měl jít vizualizovat do podoby čitelné 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 [6] je sada nástrojů postavená na OpenAPI Specifikaci. OpenAPI Specifikace, dřívěji 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 vlastonsi Swaggeru. Dokumentace má tutoriály a díky SwaggerHubu si je můžete rovnou vyzkoušet.

- Funkčnost

  Splňuje požadovanou funkčnost.

- Fórum

  Fórum[1] je aktivní a má i GitHub Issues.

2.2.2 Apicurio Studio

Apicurio Studio [7] 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 spolčně v týmu. V editoru můžete přepínat mezi interaktivní vizualizací návrhu a jeho JSON/YAML definicí.

[1] `https://community.smartbear.com/t5/Swagger-Open-Source-Tools/bd-p/SwaggerOSTools`

- 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.

2.2.3 API Workbench

API Workbench [8], 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. Obasahuje 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[2] je aktivní a má i GitHub Issues.

2.2.4 Apiary

Apiary [9] 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[3]. Apiary disponuje interaktivní dokumentací, webový editor má možnost přepnutí do tmavého režimu. Dále zajišťuje propojení s GitHub repozitářem.

- Dokumentace

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

[2] `https://community.smartbear.com/t5/Swagger-Open-Source-Tools/bd-p/SwaggerOSTools`

[3] `https://github.com/apiaryio/mson`

- Funkčnost

  Splňuje požadovanou funkčnost.

- Fórum

  Nic jako fórum jsem nenašel, pouze FAQ.

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		Smíšené ruzné jazyky - JavaScript 32.7%, Java 30.9%, TypeScript 22.9%	

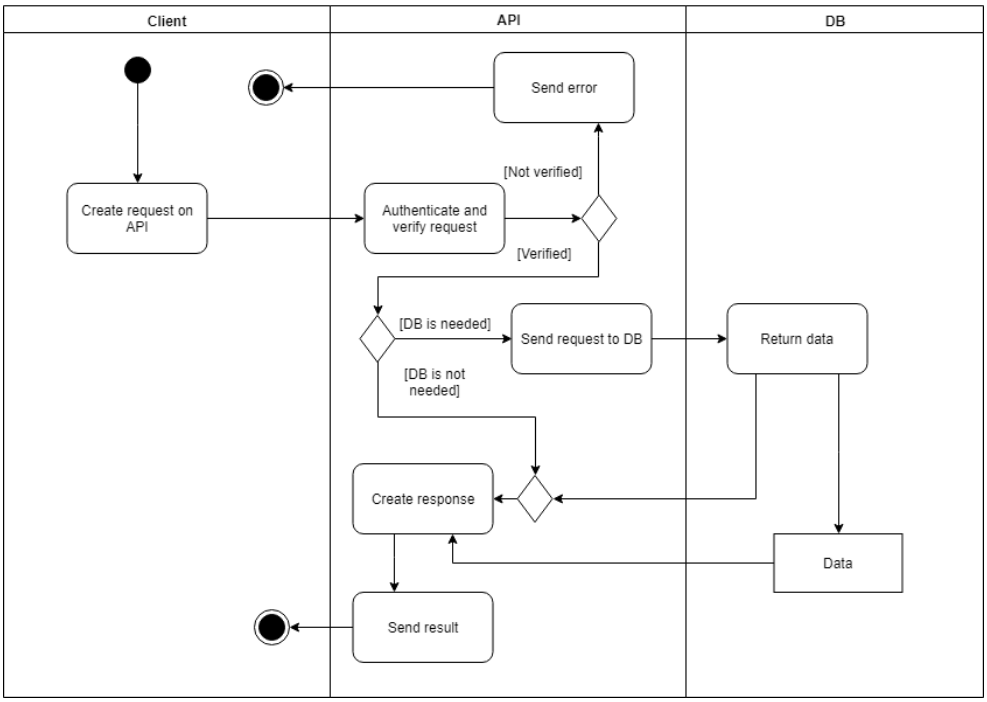
2.2.6 Výsledek výběru nástroje pro návrh API

Vítězem tohoto pomyslného souboje se stal **Swagger**. Vyhrál v kategoriích Dokumentace a GitHub repozitář. Fórum vyhovoje naším požadavkům a fukcionalitu splnily všechny nástroje.

Kapitola 3

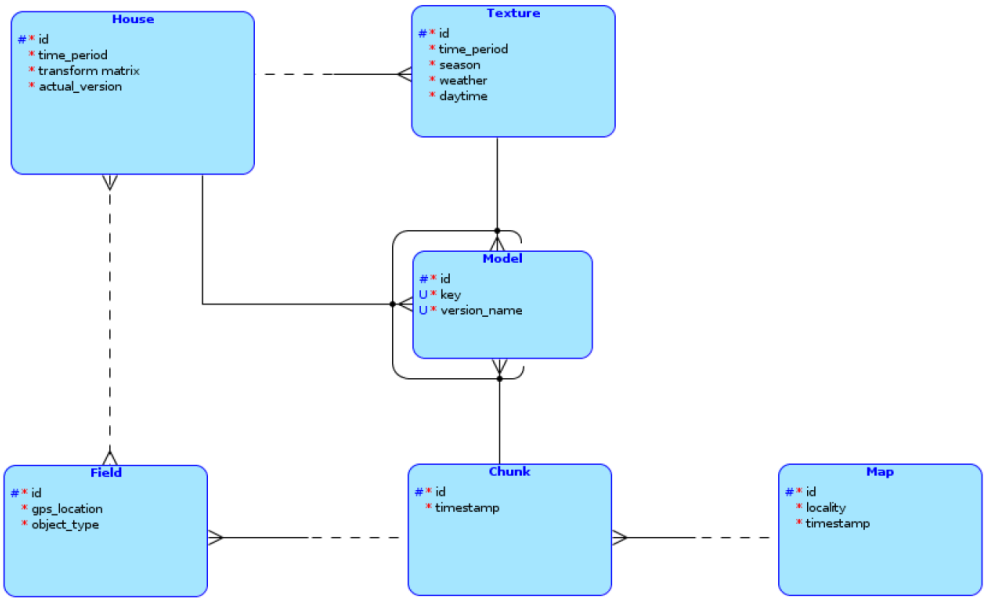
Návrh

V této kapitole provedu návrh REST API na základě analýzy funkčních požadavků. Obecný 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 3.2.



Obrázek 3.2: Současná databáze

Nacházíme se ve stavu, kdy známe schéma databáze i jak vypadají požadavky 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. Bude tedy potřeba provést změnu v databázovém schématu pro plnou podporu Editoru. Byl vytvořen doménový model pro databázi, který pokrývá požadavky Editoru, znázorněný na obrázku 3.3.
Vysvětlení jednotlivých tabulek:

- Model

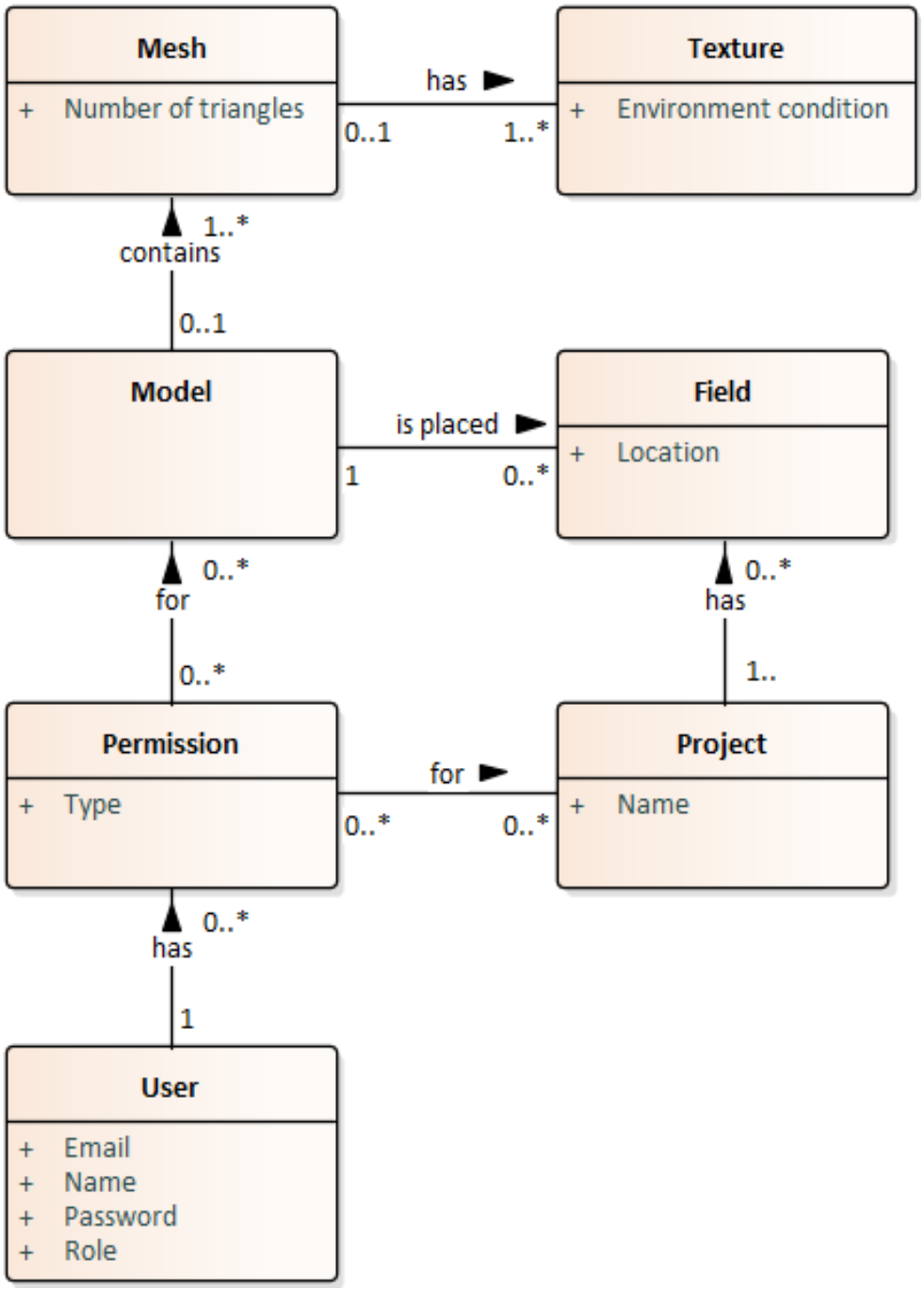
  Model je grafický model.

- Field

  Field reprezentuje umístění objektu. Snadno si ho lze představit jako půdorys.

- Project

  Projekt reprezentuje kolekci Fieldů, tedy zasazení grafických modelů. Může se jednat například o město v určitém historickém období.



Obrázek 3.3: Doménový model

- User

  User, nebo–li uživatel, je někdo, kdo vytváří či upravuje grafické modely nebo zasazuje a upravuje pozici modelů v projektu.

- Permission

  Aby uživatel mohl pracovat s modely a projekty musí k nim mít příslušná práva.

- Mesh

  Polygon mesh (doslovně přeloženo jako „polygonální síť“) se používá v počítačové grafice a reprezentuje tvar objektu. Je to soubor vrcholů, hran a povrchů, které dohromady vytváří mnohostěn.

- Texture

  Textura jednotlivého domu. Předpokládáme textury pro různá počasí, různé roční období, různé denní doby apod.

Tento doménový model bude inspirací pro můj návrh API a jeho implementaci.

3.2 Pokrytí funkčních požadavků

Pokrytí funkčních požadavků jsem zachytil v tabulce 3.1, kde ke každému požadavku je uveden koncový bod API, zapsaný syntaxí používanou ve Swaggeru. 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 třídení 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/{projectlId}
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/{projectlId}
Vytvoř kopii modelu	PUT /model
Přidání modelu do projektu	POST /project/{projectlId}
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 3.4 je zapsaný koncový bod v OpenAPI Specifikaci a na obrázku 3.5 je vidět jeho vizualizace.

3.4 Použité technologie

Zde se budu věnovat technologiím, které použiji při vývoji API.

3.4.1 Rest API

REST, neboli Representional 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 vracet XML, JSON, YAML nebo jiné formáty podle klientových požadavků. [10]

V naší implementaci budeme používat JSON, jelikož se používá ve Věnných městech českých královen.

```yaml
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
                      .eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ
                      .-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
        '400':
          $ref: '#/components/responses/BadRequest'
        '401':
          description: Login failed wrong user credentials
```

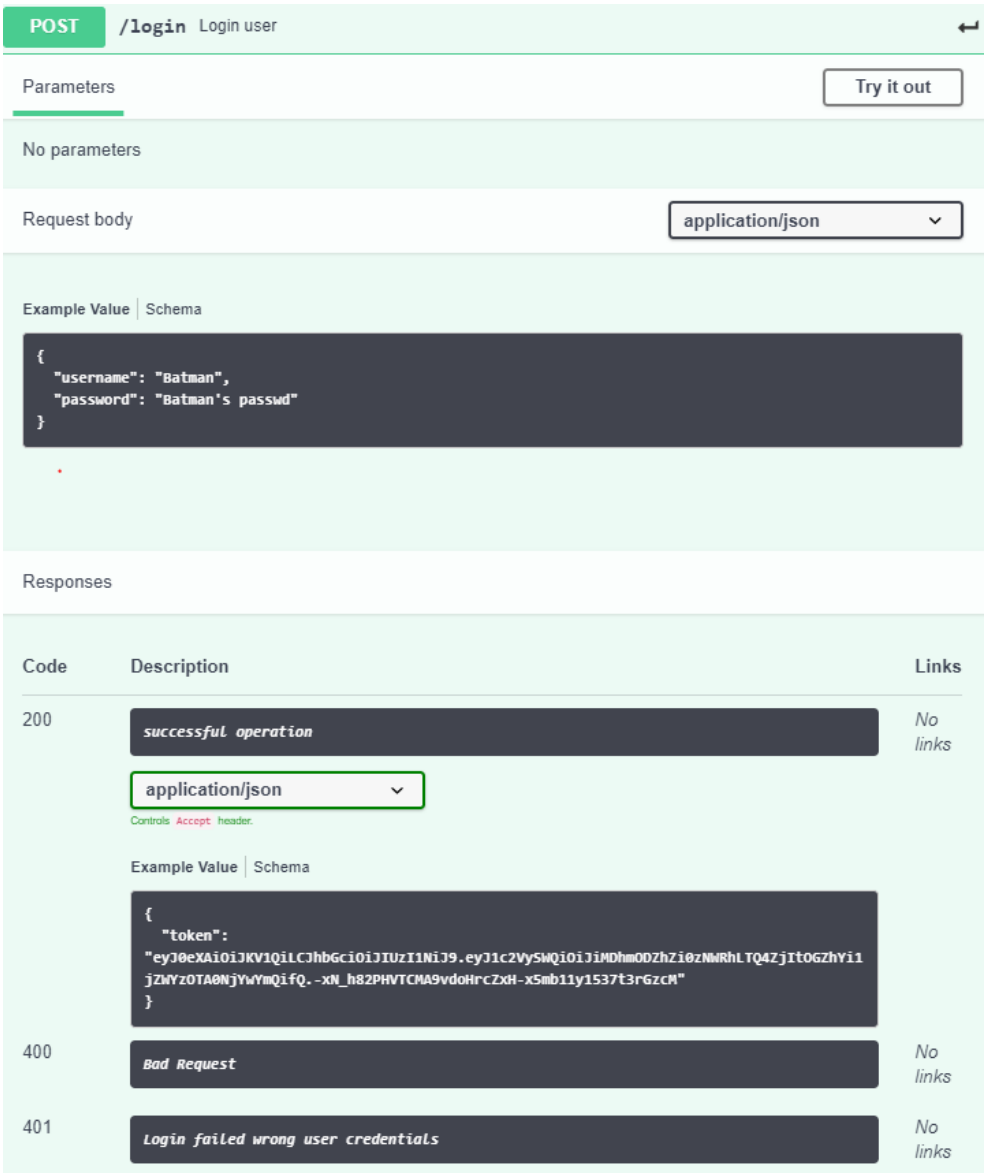
Obrázek 3.4: Popis koncového bodu ve Swaggeru

3.4.2 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 desetitisíce souběžných připojení bez nutnosti neustálého přepínání kontextu díky neblokujícímu I/O. Nedochází zde k žádnému zamykání, tudíž nemusíme mít obavy z deadlocku systému. [11]



Obrázek 3.5: Vygenerovaný koncový bod ve Swaggeru

3.4.3 Implementační jazyk

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

3.4.4 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[4] lze nalézt rozsáhlý seznam dostupného open-source software, sloužícího k administraci a monitoringu PostgreSQL databází. Nejrozšířenější a velmi přehledný je pgAdmin. [12]

3.4.5 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. [13]

3.4.6 Express

Express je framework pro Node.js, který nám umožnuje jednoduše napsat webovou aplikaci, či API. Těší se velké popularitě a podle [14] je na něm závislých 21873 balíčků.

3.4.7 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

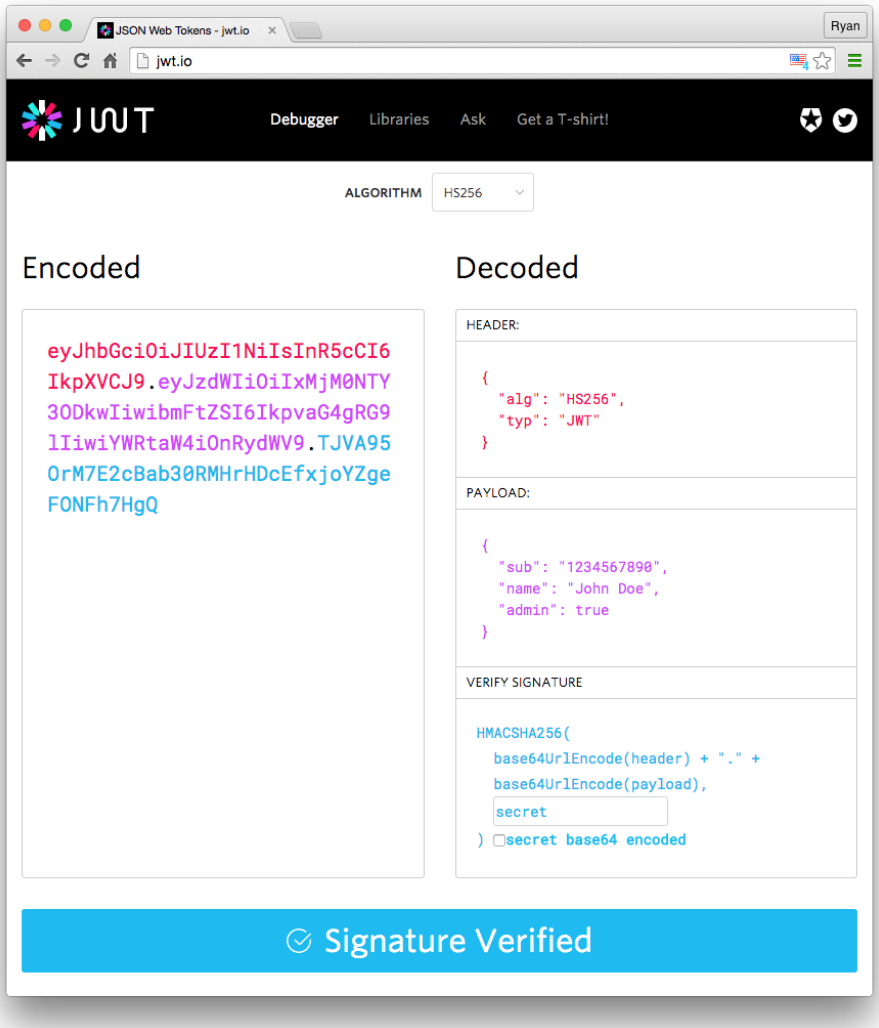
  obsahuje naše informace a může obsahovat ještě dodatečné informace jako expirace tokenu, vydavatel atd.

- 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. [15]

[4] `https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools`



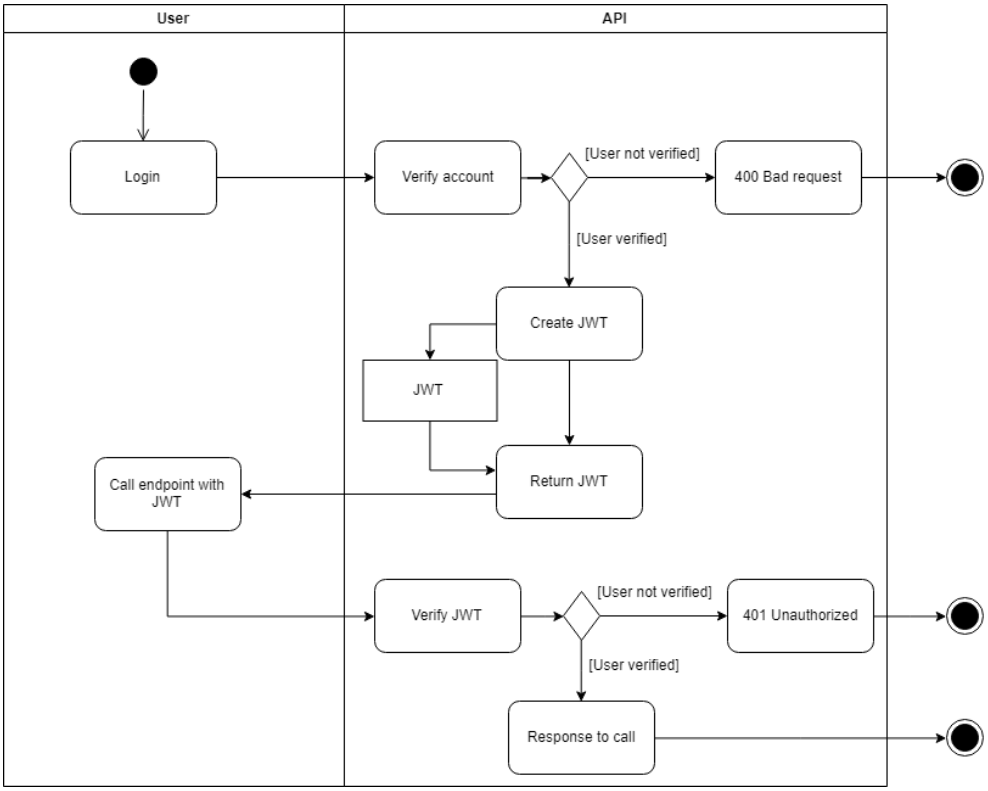
Obrázek 3.6: JWT [1]

3.4.8 TSLint

TSLint je nástroj pro statickou analýzu kódu TypeScriptu. Detekuje potencionální chyby a udržuje jednotné fomátování kódu.[16]

3.4.9 dotenv-safe

Tato knihovna umožňuje pracovat s proměnnými prostředí. Proměnné prostředí budeme zapisovat do souboru ”.env”v kořenovém adresáři a tento soubor



Obrázek 3.7: Proces autorizace

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”. [17]

3.4.10 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. [18]

Kapitola 4

Vývoj

V této kapitole se budu věnovat přípravě vývojového prostředí, tutoriálu jak zpovoznit REST API a implementaci funkčního prototypu.

4.1 Vývojové prostředí pro OS Linux

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

Nejprve si musíme nainstalovat Node.js a správce balíčk 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 následujích 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 elektronické 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 fomátu „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 naší 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
```

Vytvoříme si tsconfig.json pro konfiguraci TypeScript. Inspirovat se můžete v elektronické příloze TBD. Dokumetaci 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);
```

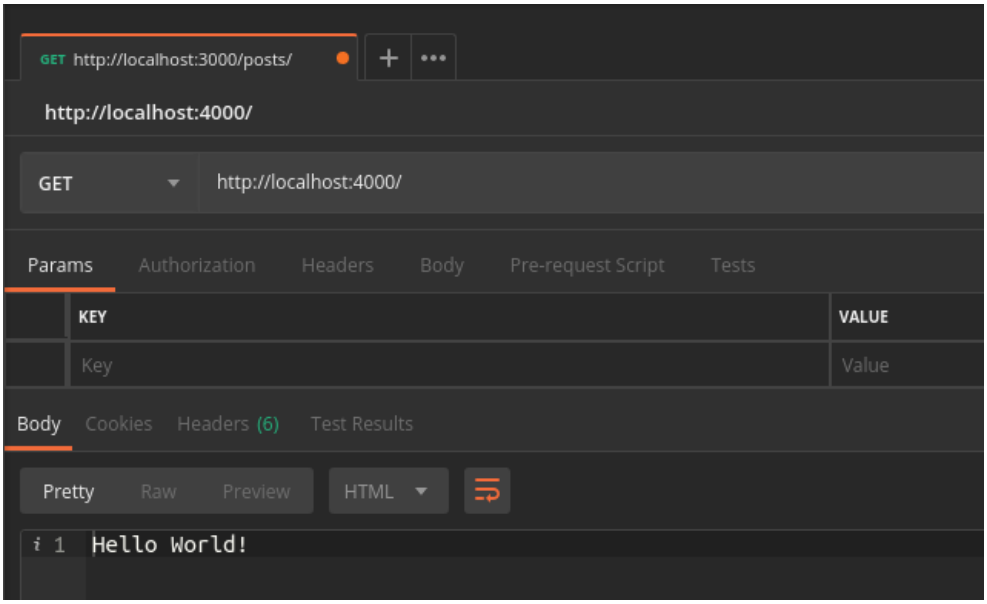
Zbýva ještě specifikovat náš port. V souboru .example.env stačí mít pouze "LISTEN_ON".

```
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 fukčnost v Postmanu[5], tak že zavoláme koncový bod, který nám vrátí Hello World! Příklad na obrázku 4.3.



Obrázek 4.1: Postman

4.1.1 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 elektronické příloze, zde je použita PostgreSQL databáze. V této konfiguraci budeme definovat naše schéma. To zadefinujeme pomocí popsání entit. V ormconfig.json bude zapsáno, kde se naše entity nachází.

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

Vytvoříme si soubor src/entity/User.ts.

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

@Entity()
export class User {

    @PrimaryGeneratedColumn()
```

[5] https://www.getpostman.com

```
    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 databází a správně připojení, mělo by se vám v konzoli po spuštění aplikace zobrazit:

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

4.1.2 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
    },
];
```

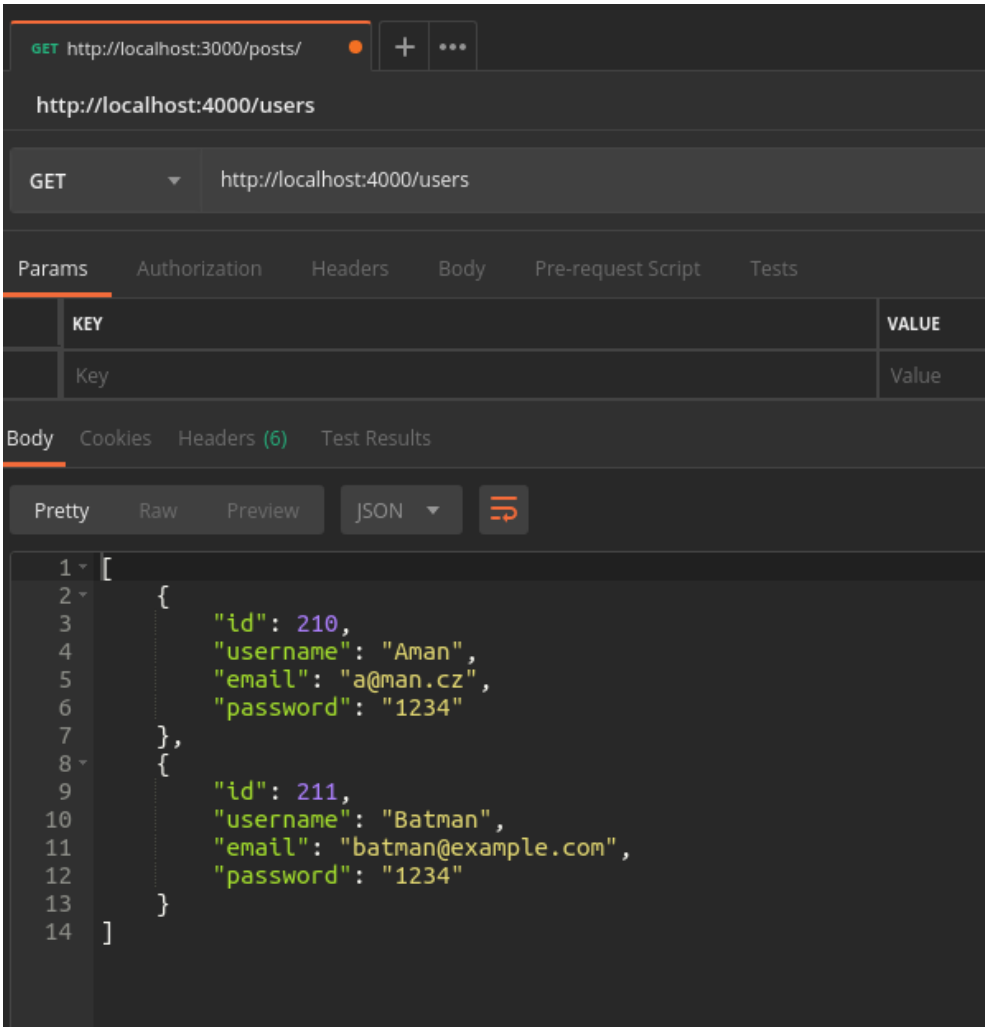
Nakonec přidáme do src/app.ts následující kód, který zavolá funkci pro přidání uživatelů a přidá koncové body z src/routes.ts.

```
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: Fu
            route.action(request, response)
                .then(() => next)
                .catch(err => next(err));
```

```
    });
});
```

Nastartujeme naší aplikaci a funkčnost můžeme otestovat v Postmanu[6]. Zaolání koncového bodu by mělo vypadat tak, jak je zachyceno na obrázku 4.4.



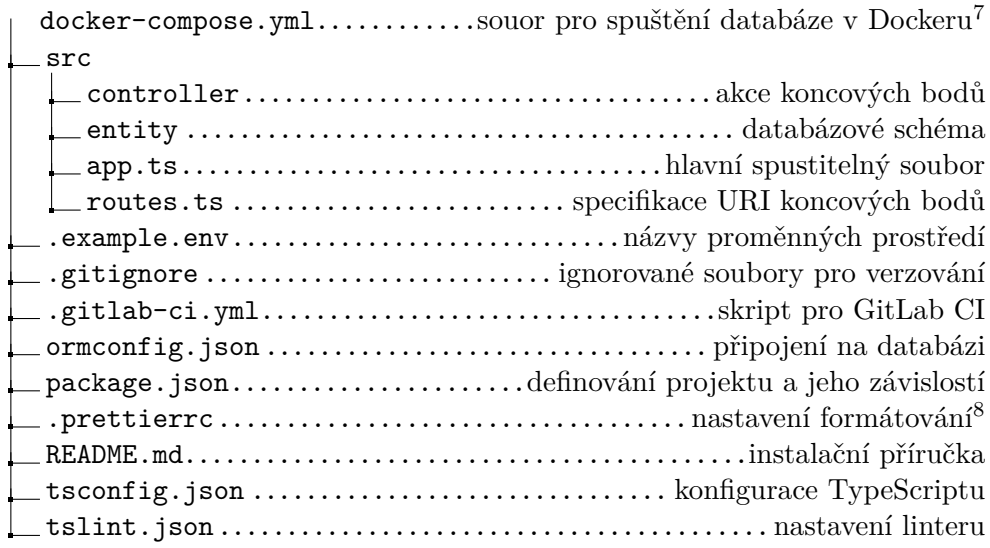
Obrázek 4.2: PostmanUsers

[6] https://www.getpostman.com

4.2 Implementace

Zde popíši jak vypadá prototyp Rest API pro Editor virtuální reality. Jeho instalační příručku najdete v elektronické příloze README.md. Naše implementace navazuje na předchozí sekci Vývojové prostředí pro OS Linux.

Struktura adresáře implementace je zachycena na obrázku 4.5.



Obrázek 4.3: Adresář implementace

Zaměřil jsem se na nejpodstatnější část pro funkčnost editoru virtuální reality a tím je předávání grafických modelů, správa projektů a umisťování modelů do projektů. Práva by se musela řešit ke každému koncovému bodu a to vyžaduje rozsáhlejší analýzu. Práva v implementaci tedy řeším tak, že existuje koncový bod *login*, který vrací JWT), jež expiruje po 4 hodinách od vytvoření. Tímto tokenem se uživatel bude autorizovat při volání zbylých koncových bodů. Znamená to tedy, že pokud se uživatel přihlásí, tak je oprávněn ke všem akcím.

Pro tvorbu databázového schématu jsem vyšel z doménového modelu zachyceném na obrázku 3.3. Neimplementoval jsem pouze entitu Permission, z výše zmíněného důvodu.

Implementovány jsou koncové body pro CRUD operace entit Model, Project a Field. A k nim již zmíněný login.

Fukční prototyp splňuje požadavky editoru virtuální reality až na spávu práv. Spávu práv a uživatelských účtů pro REST API si dokáži představit, jako samostatnou bakalářskou práci. Dalším nedostatkem prototypu je, že ukládá uživatelská hesla v plaintextu.

Kapitola 5

Testování

Pro ověření funkčnosti našeho prototypu provedeme integrační testování. Integrační testování v našem případě vypadá jako sled volání koncových bodů (např. přihlaš se, vytvoř model, změň model, vymaž model). Pro testování našeho API použijeme testovací framework Jest[9] a npm modul Axios[10] pro vytváření HTTP požadavků. Zavolání jednoho koncového bodu v testování vypadá následovně:

```
const secret = process.env.JWT_SECRET || '';
it('should response with 200', async () => {
  await axios({
    method: 'post',
    url: 'http://localhost:4000/login',
    data: {
      username: 'Batman',
      password: '1234',
    },
  })
    .then(response => {
      expect(typeof jwt.verify(response.data, secret)).toBe('object');
      expect(response.status).toBe(200);
    })
    .catch(error => {
      throw error;
    });
});
```

Všechny testy můžeme najít v elektronické příloze todo.

[9] https://jestjs.io
[10] https://github.com/axios/axios

5.1 Jest

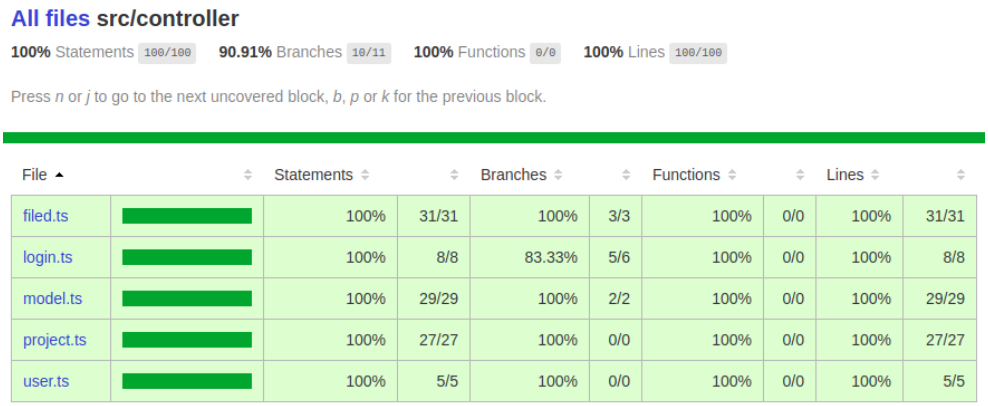
Jest je framework pro testování JavaScriptových apilkací. Hodí se pro testování Node.js a typescriptu. Pro náš prototyp jsem vytvořil dva konfigurační soubory pro testování, které se liší pouze tím, že v jednom je navíc pokrytí kódu testy. Testy bez pokrytí se pouští příkazem:

```
npm test
```

A testy s pokrytím:

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

Pokrytí kódu testy se nám uloží do složky coverage. V internetovém prohlížeči si můžeme otevřít soubor src/impl/coverage/lcov-report/index.html, který najdete v elektronické příloze, jeho podoba je zachycena na obrázku 5.1. Uvidíme



Obrázek 5.1: Pokrytí kódu testy

v něm statistiku pokrytí kódu testy a můžeme si interaktivně procházet adresář a dívat se na pokrytí konkrétních souborů.

5.2 GitLab

Pro verzování teoretické i praktické části jsem používal GitLab. GitLab nabízí nástroj pro průběžnou integraci, který při každém commitu nahraném na GitLab spustí 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 první části teoretické práce (analýze) proběhla analýza funkční a nefunkční požadavků editoru virtuální reality. Funkční požadavky byly vytvořeny ve spolupráci s Patrikem Křepinským, který momentálně pracuje na editoru virtuální reality. Dále byl vybrán nástroj Swagger jakožto nástroj pro návrh a dokumentaci REST API. V tomto výběru byla provedena analýza čtyř nástrojů pro návrh API a zavedení výběrové metodiky, kterou byly nástroje ohodnoceny.

V druhé části teoretické práce (návrhu) byl proveden návrh REST API. V návrhu se vyskytl problém, že současná databáze Věnných měst českých královen nepostačuje požadavkům editoru virtuální reality, byl tedy proveden nový návrh databáze.

V první části praktické práce (vývoji) byly popsány použité technologie. Dále tato část 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 na základě předešlého návrhu.

V druhé části praktické práce (testování) byly naimplementovány integrační testy pro náš prototyp. Všechny testy proběhly úspěšně. Dále je zde popsané GitLab CI, které bylo využíváno při verzování této práce.

Literatura

[1] JSON Web Token Introduction. *JSON Web Token Introduction*, [Online; cit. 2019-04-19]. 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. 2019-04-19]. Dostupné z: https://swagger.io

[7] *Apicurio Studio*. [Online; cit. 2019-04-19]. Dostupné z: https://apicur.io

[8] *Apiworkbench.com*. [Online; cit. 2019-04-19]. Dostupné z: http://apiworkbench.com/docs

[9] *Apiary.io*. [Online; cit. 2019-04-19]. Dostupné z: https://apiary.io

[10] Malý, M.: REST: architektura pro webové API - Zdroják. *Zdroják*, [Online; cit. 2019-04-19]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[11] Node.js: *Node.js*. [Online; cit. 2019-04-19]. Dostupné z: https://nodejs.org/en/about

[12] *Postgresql.org*. [Online; cit. 2019-04-19]. Dostupné z: https://www.postgresql.org

[13] *Modulecounts.com*. [Online; cit. 2019-04-19]. Dostupné z: http://www.modulecounts.com

[14] *Gist*. [Online; cit. 2019-04-19]. Dostupné z: https://gist.github.com/anvaka/8e8fa57c7ee1350e3491

[15] *JSON Web Tokens*. [Online; cit. 2019-04-19]. Dostupné z: https://jwt.io

[16] palantir/tslint. *GitHub*, [Online; cit. 2019-04-19]. Dostupné z: https://github.com/palantir/tslint

[17] rolodato/dotenv-safe. *GitHub*, [Online; cit. 2019-04-19]. Dostupné z: https://github.com/rolodato/dotenv-safe

[18] *Typeorm.io*. [Online; cit. 2019-04-19]. Dostupné z: https://typeorm.io

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CI Continuous Integration

IDE Integrated Development Environment

I/O Input/Output

JSON JavaScript Object Notation

npm Node.js package manager

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `docs` ..... adresář se Swaggger dokumentací
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF