VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FRONTEND PRO GENERÁTOR TESTOVACÍCH DAT

FRONT-END FOR GENERATOR OF TEST DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ANDREJ NAŇO
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

21378

Student: **Naňo Andrej**
Program: Informační technologie
Název: **Frontend pro generátor testovacích dat**
Front-End for Generator of Test Data
Kategorie: Web
Zadání:

1. Nastudujte technologie pro tvorbu webových aplikací.
2. Navrhněte infrastrukturu klient-server pro tvorbu testovacích dat využívající existující projekt db-generator-extended. Navrhněte rozhraní REST pro tuto infrastrukturu.
3. Implementujte webové rozhraní pro projekt db-generator-extended.
4. Správnost funkcionality podpořte automatizovanými testy.

Literatura:

- Domovská stránka projektu db-generator-extended. https://pajda.fit.vutbr.cz/testos/db-gen-extended
- Žára, O.: JavaScript - Programátorské techniky a webové technologie. Computer Press, 2015.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Táto bakalárska práca sa zaoberá návrhom a vývojom webovej aplikácie Dataster, ktorá umožňuje generovať dáta pre účely testovania softvéru. Webová aplikácia je súčasťou platformy Testos, ktorá sa zaoberá automatizovaním testovania softvéru. Cieľom projektu bolo vytvorenie webového a aplikačného rozhrania pre existujúci nástroj dbgenx, ktorý je taktiež súčasťou platformy. Poskytnuté riešenie ciele spĺňa a je schopné priameho nasadenia do produkčného prostredia.

Abstract

This bachelor thesis focuses on the design and development of the Dataster web application which provides data generating functionality for use in the testing of software. Web application is a part of the Testos platform which supports automation of software testing. Aim of the project was to create a web and application interface for the existing application dbgenx which is also part of the platform. Provided solution fulfills specified goals and is suitable for the deployment to a production environment.

Kľúčové slová

generátor testovacích dát, webová aplikácia, webový systém, REST API, Vue, frontend, backend, web, docker, kontajnerizácia, mikroslužby

Keywords

test data generator, web application, web system, REST API, Vue, frontend, backend, web, docker, containerization, microservices

Citácia

NAŇO, Andrej. *Frontend pro generátor testovacích dat.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Frontend pro generátor testovacích dat

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Aleša Smrčku, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som pri písaní práce čerpal.

.....
Andrej Naňo
15. mája 2019

Poďakovanie

Rád by som sa poďakoval vedúcemu práce, pánovi Ing. Alešovi Smrčkovi, Ph.D. za konzultácie a cenné rady, ktoré mi poskytol.

Obsah

1 Úvod **2**

2 Analýza požiadaviek projektu **4**
2.1 Vyhodnotenie problému zo zadania 4
2.2 Charakteristika, kontext a rámec projektu 4
2.3 Formálna špecifikácia požiadaviek 8

3 Vývoj webových aplikácií **10**
3.1 Webové technológie 10
3.2 Architektúra webových aplikácií 13
3.3 Docker a virtualizácia založená na kontajneroch 16
3.4 Zhodnotenie podobných alebo existujúcich riešení 16

4 Návrh architektúry a plán práce **18**
4.1 Systémový návrh 18
4.2 Návrh komponentov systému 21
4.3 Návrh virtualizácie pomocou kontajnerov 24
4.4 Organizačná štruktúra vývoja projektu 25

5 Implementácia komponentov systému **27**
5.1 Vývoj a nasadenie v kontajneri 27
5.2 Implementačné problémy jednotlivých komponentov 27
5.3 Ekosystém mikroslužieb 32

6 Užívateľské a aplikačné rozhrania **34**
6.1 Návrh webového rozhrania 34
6.2 Osvedčené postupy a optimalizácie 41

7 Testovanie, integrácia a nasadenie do produkcie **43**
7.1 Testovanie 43
7.2 Nasadenie a konfigurácia spustenia 46

8 Záver **48**

Literatúra **49**

A Konfiguračný súbor – Recipe **51**

B API špecifikácia **53**

Kapitola 1

Úvod

V oblasti testovania softvéru býva často požadované vhodne reprezentovať špecifický stav systému, ktorý je potrebné otestovať. Proces uvedenia systému do určitého stavu je náročný, obzvlášť ak systém pracuje s osobnými dátami užívateľov alebo inými citlivými údajmi. Takéto dáta väčšinou nie je možné jednoducho použiť z existujúcich záznamov databázy systému a je potrebné pre kvalitné testovanie vytvoriť nové, no čo najviac podobné dáta. Vytvorenie množstva špecifických dát je potencionálne náročná aktivita. Tým pádom vzniká potreba generovať dáta reprezentujúce skutočnú podobu produkčnej verzie systému.

Zameraním tejto bakalárskej práce je návrh a tvorba webovej aplikácie **Dataster**, ktorá uľahčuje a automatizuje generovanie vzorových dát pre databázy. Hlavným kontextom využitia aplikácie je v prvom rade prostredie platformy **Testos**[1] (Test Tool Set), ktorá sa zaoberá automatizovaním testovania softvéru [23]. Nástroje v tejto platforme sú zamerané na viacero úrovní a oblastí testovania, medzi ktoré patrí aj dátami riadené testovanie (data-driven testing), zahrňujúce **Dataster**. Tento projekt bude nadväzovať na aplikáciu **dbgenx**[2], ktorá bola taktiež vytvorená v rámci platformy **Testos** [17]. Práve moduly z tejto aplikácie budú tvoriť jadro a základnú aplikačnú logiku pre **Dataster**.

Hlavnou motiváciou pre rozšírenie projektu **dbgenx** je najmä potreba užívateľského rozhrania. Súčasná podoba tohto projektu poskytuje len jednoduché možnosti interakcie s nástrojom pomocou príkazového riadku. Druhou motiváciou je výzva poskytnúť možnosti nástroja verejne, umiestnením nástroja na web.

Globálnym cieľom práce je navrhnutie a implementácia vyššie uvedenej webovej aplikácie. Návrhom aplikácie sa rozumie proces formulovania vhodného abstraktného pohľadu na celkový systém a jeho jednotlivé komponenty, za účelom splnenia vopred stanovených požiadaviek projektu. Výsledok návrhu sa následne v procese implementácie transformuje do reálnej funkčnej podoby.

Požadovanými vlastnosťami projektu sú taktiež podpora správnosti funkcionality pomocou automatizovaných testov, systematická špecifikácia požiadaviek a využitie vhodných webových technológií, pomocou ktorých bude aplikácia implementovaná.

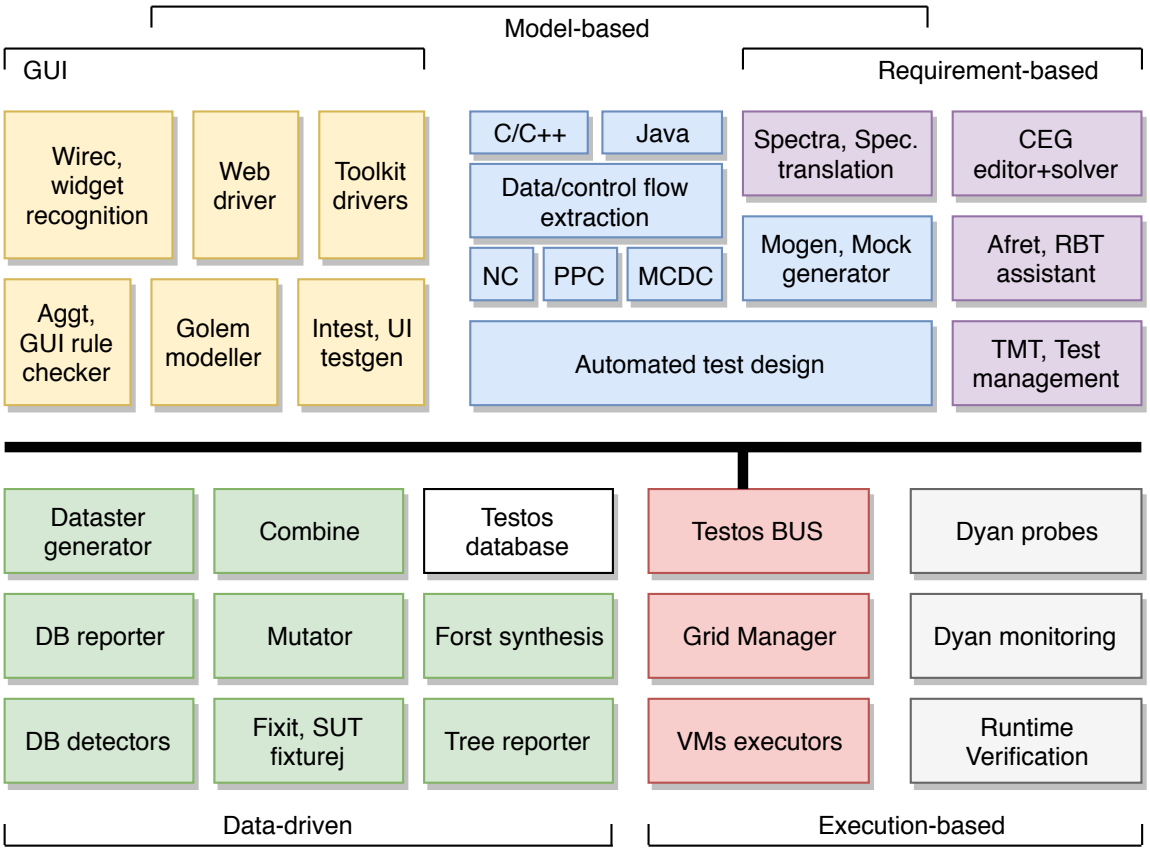
Práca v prvej časti začne analýzou súčasného stavu a požiadaviek pre projekt, na základe ktorých budú stanovené formálne požiadavky v kapitole 2. Na tie sa následne nadviaže uvedením relevantných webových technológií pre projekt v kapitole 3. V kapitole 4 bude predložený návrh architektúry systému a jeho vnútorných komponentov. Implementačné detaily a problémy, ktoré bolo pri vývoji projektu potrebné vyriešiť, sú popísané v kapi-

[1] http://testos.org
[2] https://pajda.fit.vutbr.cz/testos/dbgenx

tole 5. Webové a aplikačné rozhrania budú navrhnuté a špecifikované v kapitole 6. Miera splnenia požiadaviek sa overí pomocou automatizovaných testov na viacerých úrovniach implementácie v kapitole 7.

Pri návrhu aj vývoji by mal byť kladený dôraz na využitie v kontexte platformy Testos ako aj na vytvorenie riešenia, ktoré bude v budúcnosti schopné dodatočného vývoja a rozšírenia. Vzniká tým teda menej priama, ale významná požiadavka na projekt, a to potreba kvalitného vývojového ekosystému pre budúce nadviazanie v rámci platformy.

Hlavným cieľom projektu je teda nielen vytvorenie webovej aplikácie, ale aj súčasne kvalitného softvérového projektu aj po organizačnej a dokumentačnej stránke. Výsledkom projektu bude umiestnenie implementovanej webovej aplikácie Dataster do produkčného prostredia na adrese https://dataster.testos.org.



Obr. 1.1: Nástroje platformi Testos (Test Tool Set). Táto práca sa zaoberá návrhom a vývojom projektu Dataster, ktorý patrí do kategórie dátami riadeného testovania (data-driven testing). Prevzaté z [http://testos.org/].

Kapitola 2

Analýza požiadaviek projektu

Predpokladom pre kvalitný návrh je jasná predstava o požadovaných vlastnostiach pripravovaného softvéru. Táto kapitola sa zaoberá dôkladnou analýzou a vyhodnotením požiadaviek na softvérový projekt – Dataster. Proces analýzy je rozdelený na niekoľko častí, ktorých snahou je získať čo najpresnejšiu predstavu o skutočnej podstate projektu. V jednotlivých etapách bude postupne abstrakcia analýzy znižovaná a predstava o vlastnostiach pripravovaného projektu jasnejšia. Systematicky bude potom možné reagovať na požiadavky konkrétnymi a relevantnými návrhmi a na záver overiť ich splnenie pomocou testov.

2.1 Vyhodnotenie problému zo zadania

Zo zadania projektu je na úvod možné vyhodnotiť dve základne požiadavky:

1. Vytvoriť **aplikačné rozhranie** pre aplikáciu dbgenx.
2. Vytvoriť **webové užívateľské rozhranie** pre túto infraštruktúru.

Môže sa teda zdať, že zjavným problémom a riešením by malo byť navrhnutie a implementovanie jednotlivých rozhraní. Tento popis problematiky je však veľmi nešpecifický a skutočná realizácia sa môže významne odlišovať od skutočných požiadaviek projektu. Je preto dôležité plne pochopiť a ilustrovať všetky významné detaily, prípady použitia, obmedzenia alebo predpoklady o používaní.

2.2 Charakteristika, kontext a rámec projektu

Prvým krokom k lepšiemu pochopeniu podstaty projektu je **zozbieranie požiadaviek** a **definovanie rámca projektu**. Nasledujúce kapitoly sa budú venovať práve týmto aktivitám, s cieľom získania čo najviac uspokojivej predstavy o skutočnej povahe projektu. Väčšina požiadaviek vychádza buď priamo zo zadania, pravidelných stretnutí v rámci skupiny Testos s vedúcim práce alebo postupným rozširovaním rámca projektu a vznikajúcich nejasností pri implementácii.

Zodpovedanie základných otázok o projekte

Počiatočnú predstavu môže poskytnúť zodpovedanie otázok ohľadom veľkosti, rozsahu a prostredia projektu. Tieto otázky som spoločne s vedúcim práce konzultoval a výsledkom sú nasledujúce odpovede, poskytujúce intuitívne pochopenie vlastností projektu.

- **Kto bude systém využívať?**
  Testeri a užívatelia pracujúci v oblasti testovania softvéru.
- **Ako ho budú používať?**
  Najmä prostredníctvom webového rozhrania alebo využitím API cez HTTP klienta.
- **Aké množstvo užívateľov bude systém využívať?**
  Jednotky až desiatky užívateľov.
- **Aké je očakávané správanie a hlavná funkcia systému?**
  Systém by mal umožniť generovanie dát prostredníctvom webového rozhrania.
- **Aké sú vstupy a výstupy pre systém?**
  Vstupom sú predpisy pre generovanie, výstupom sú vygenerované dáta.
- **Aké množstvo a akú veľkosť dát bude systém využívať?**
  Očakávaná veľkosť datasetov pre generovanie nepresiahne jednotky megabajtov.
- **Aké množstvo žiadostí bude systém v daný moment očakávať?**
  Systém by nemal byť zaťažovaný viac ako jednotkovým počtom žiadostí za sekundu.

Role užívateľov

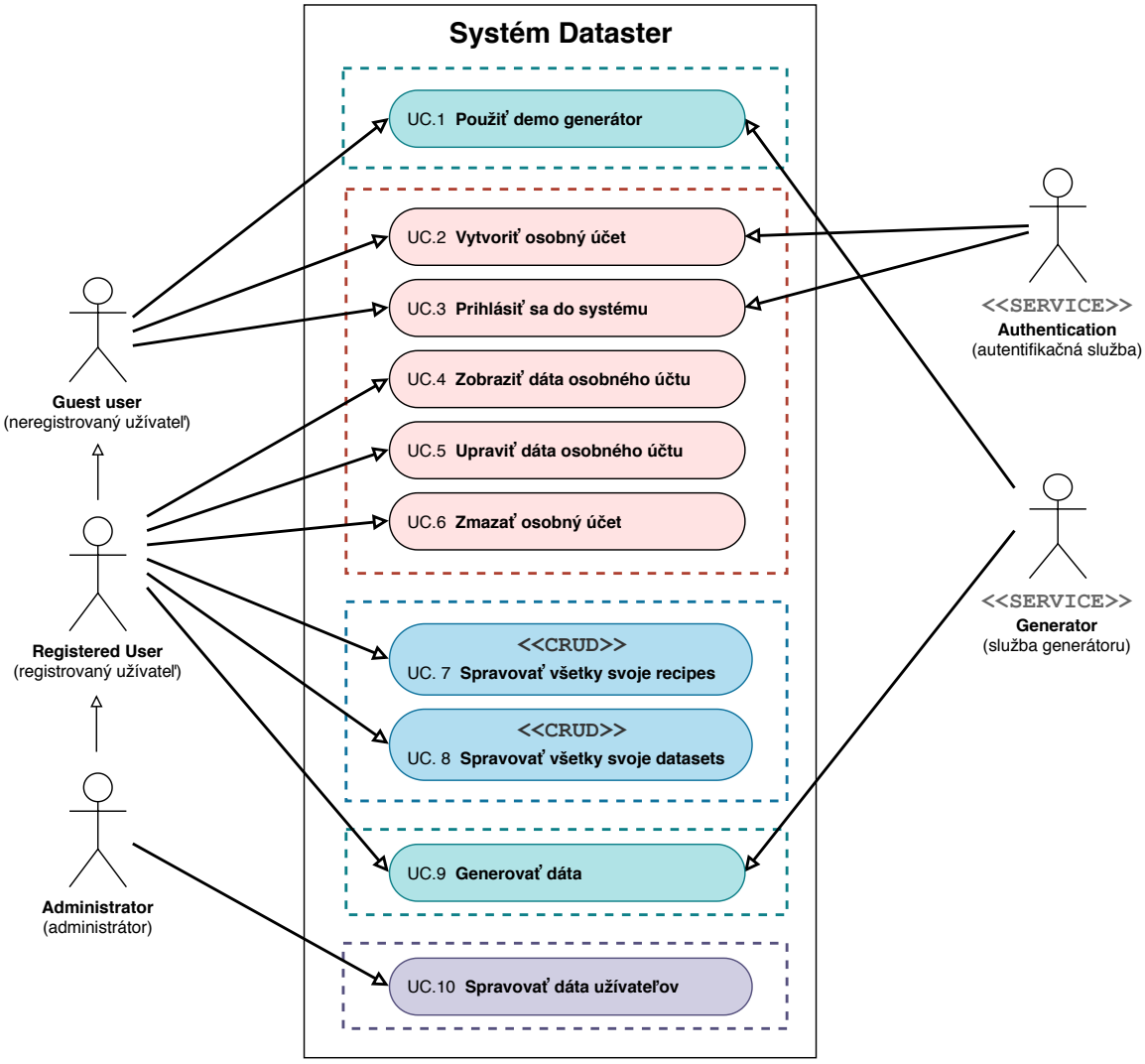
Základná funkcionalita Datasteru by mala byť dostupná aj pre anonymných návštevníkov webového rozhrania, pre ktorých bude v projekte evidovaná rola **Guest user** (neregistrovaný užívateľ).

Pre účely dlhodobejšieho využívania tohto nástroja by však mala existovať možnosť uloženia stavu alebo pracovných dát návštevníkov. Webové rozhranie Datasteru by malo umožňovať vytváranie užívateľských účtov, pre užívateľov, ktorí majú o spomenutú funkcionalitu ukladania dát záujem. Pre tieto účely bude potrebné evidovať rolu **Registered user** (registrovaný užívateľ).

Pre kontrolu, prípadnú intervenciu a manipuláciu nad všetkými užívateľskými dátami by mala následne existovať aj rola **Administrator** (administrátor), ktorá by disponovala väčšími právomocami.

Scenáre a prípady použitia

Lepšiu predstavu o jednotlivých právomociach, úlohách a schopnostiach jednotlivých rolí v systéme môže poskytnúť diagram prípadov použitia na obrázku 2.1.



Obr. 2.1: Diagram prípadov použitia (Use case diagram), popisujúci najdôležitejšie funkcie systému a ich aktérov. Prípady použitia sú v diagrame rozdelené do skupín podľa zamerania daných aktivít.

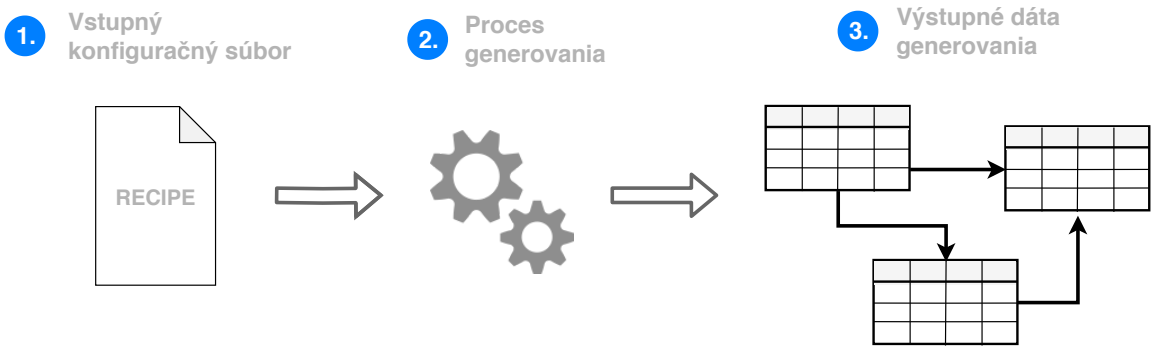
Diagram popisuje najčastejšie a najdôležitejšie prípady použitia aplikácie Dataster. Hlavnými aktérmi v diagrame sú tri typy užívateľov systému a dve služby poskytujúce špecifické potreby systému.

- **GUEST USER** – každý užívateľ, ktorý práve nie je prihlásený.
- **REGISTERED USER** – užívateľ, ktorý si už vytvoril účet a je práve prihlásený.
- **ADMINISTRATOR** – užívateľ, ktorý využíva špeciálny užívateľský účet s pridanými právomocami.

Proces generovania

Jedným z hlavných prípadov použitia Datasteru je prípad **UC.9**, umožňujúci aktérovi **Registered user** (registrovaný užívateľ) generovať dáta. Proces generovania, znázornený

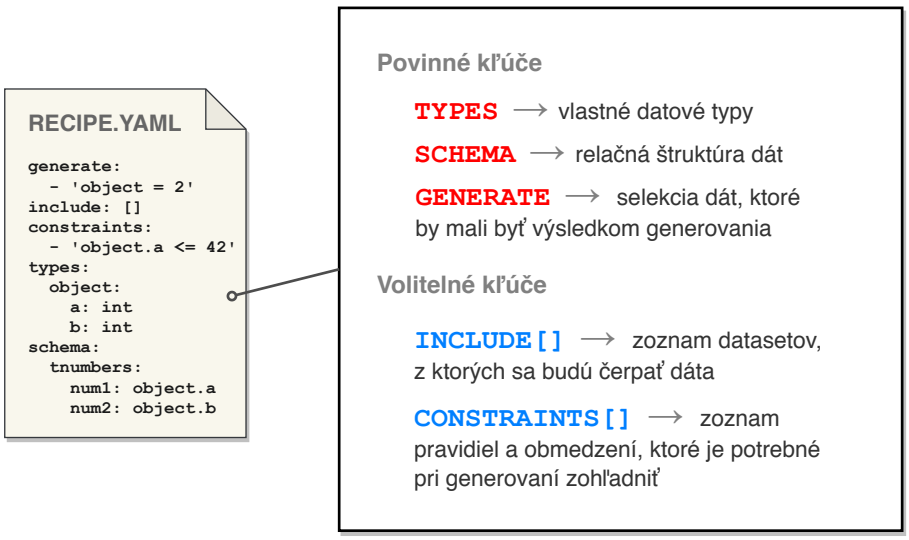
na obrázku 2.2, poskytuje základnú očakávanú funkcionalitu aplikácie, a to prostredníctvom konfiguračného súboru, nazývaného **recipe**, vytvoriť generovaciu úlohu, ktorej výsledkom sú vygenerované dáta.



Obr. 2.2: Vizuálne znázornenie procesu generovania. Vstupný konfiguračný súbor – recipe je vložený do systému, následne prebehne generovanie a systém vráti užívateľovi na výstupe sadu vygenerovaných dát.

Príklad vstupného konfiguračného súboru Recipe

Základným vstupom pre generovanie je konfiguračný súbor **Recipe**. Slúži ako predpis, určujúci, akým spôsobom sa majú dáta vygenerovať. Hlavnými a povinnými súčasťami recipe sú kľúče **types**, **schema** a **generate**. Dodatočne je možné zadať zoznam datasetov, ktoré sa pri generovaní majú použiť v zozname **include** a pravidlá, ktoré by pri generovaní mali byť zohľadnené pomocou **constraints**. Lepšiu predstavu o forme konfiguračného súboru poskytuje ilustrácia schémy na obrázku 2.3.



Obr. 2.3: Schéma konfiguračného súboru recipe.

Príklad kompletného konfiguračného súboru pre generovanie je poskytnutý v prílohe A.

Pojmy špecifické pre projekt

V nasledujúcich kapitolách tejto práce sa budú často vyskytovať určité pojmy, ktoré sú pre Dataster príznačné. Nasledujúci zoznam ich popisuje a vysvetľuje.

- **RECIPE** – Konfiguračný predpis pre generovanie.
- **DATASET** – Sada dát určitej typovej kategórie.
- **FRONTEND** – Klientská časť aplikácie.
- **DEPLOYMENT** – Nasadenie aplikácie do produkčného prostredia.

Obmedzenia

Obmedzeniami projektu sa myslia dopredu známe fakty, ktoré definujú rámec projektu alebo ho nejakým spôsobom obmedzujú. Je to napríklad kontext použitia, konkrétne nasadenie produkčnej verzie, potreba podpory na prehliadačoch atď.

Prostredie nasadenia aplikácie

Aplikácia bude nasadená na virtuálnom stroji Linuxovej distribúcie. Poskytovateľom virtuálneho servéra je platforma Testos, v ktorej bude Dataster nasadený v svojom vlastnom segmente zdieľaného hostingu. Očakáva sa len jedna instancia spustenia aplikácie, ku ktorej budú užívatelia pristupovať prostredníctvom internetu na doméne `dataster.testos.org`.

Predpoklady

Zásadným predpokladom je, že užívateľmi budú testeri, u ktorých sa predpokladajú existujúce zručnosti a poznatky o pojmoch a jazyku z oblasti testovania. Takisto sa predpokladá určitá zručnosť práce so zložitejšími užívateľskými rozhraniami a schopnosť upravovať dáta pomocou editoru zdrojového kódu. Ďalším predpokladom je napríklad aj znalosť jazyka YAML, JSON alebo SQL.

2.3 Formálna špecifikácia požiadaviek

Nasledujúce požiadavky pre projekt, zapísane v tabuľke 2.1, boli vyhodnotené zo zadania a počas konzultácií s vedúcim práce. Ich forma je daná štvoricou údajov, ktoré jednotlivé požiadavky jednoznačne identifikujú a popisujú. Každá požiadavka má svoj identifikátor, popis, kategóriu do ktorej patrí a prípadne závislosť na iných, nadradených požiadavkách. Existujú dve kategórie požiadaviek:

- **Funkcionálne požiadavky** – označované pomocou **F**.
- **Nefunkcionálne požiadavky** – označované pomocou **NF**.

Id.	Popis	Kategória	Závislosť
REQ.1	Aplikácia je spustiteľná ako webová aplikácia dostupná prostredníctvom HTTP/HTTPS protokolu na zadanom porte.	NF	-
REQ.2	Klientská časť systému je spustiteľná samostatne a nezávisle od zvyšku systému.	NF	-
REQ.3	Aplikácia poskytuje webové rozhranie (GUI).	F	-
REQ.4	Aplikácia poskytuje aplikačné rozhranie (API).	F	-
REQ.5	Aplikácia akceptuje žiadosti na generovanie vo formáte JSON.	F	REQ.4
REQ.6	Aplikácia generuje dáta pomocou modulu dbgenx.	F	REQ.4
REQ.7	Aplikácia pri nesprávnom predpise na generovanie odpovie chybovou správou.	F	REQ.4
REQ.8	Aplikácia transformuje žiadosti na generovanie na ekvivalentné volania modulu dbgenx.	F	REQ.6
REQ.9	Webové rozhranie je funkčne ekvivalentné aplikačnému rozhraniu.	F	-
REQ.10	Užívateľ je schopný vytvoriť osobný účet v systéme.	F	-
REQ.11	Užívateľ je schopný prihlásenia do systému prostredníctvom svojho osobného účtu.	F	REQ.10
REQ.12	Užívateľ je schopný upraviť dáta svojho osobného účtu po prihlásení.	F	REQ.11
REQ.13	Užívateľ je schopný zobraziť dáta svojho osobného účtu po prihlásení.	F	REQ.11
REQ.14	Užívateľ je schopný zmazať svoj osobný účet.	F	REQ.11
REQ.15	Užívateľ nie je schopný opätovného prihlásenia po zmazaní účtu.	F	REQ.14
REQ.16	Pri zadaní nesprávnych prihlasovacích údajov systém odpovie chybovou správou.	F	-
REQ.17	Cesty užívateľského API budú vyžadovať autentifikáciu totožnosti.	F	REQ.11
REQ.18	Užívateľ je schopný vytvoriť a uložiť RECIPE pre generovanie.	F	REQ.11
REQ.19	Užívateľ je schopný zobraziť všetky svoje RECIPES.	F	REQ.11
REQ.20	Užívateľ je schopný upraviť každý svoj RECIPE.	F	REQ.11
REQ.21	Užívateľ je schopný zmazať každý svoj RECIPE.	F	REQ.11
REQ.22	Užívateľ je schopný vytvoriť a uložiť DATASET pre generovanie.	F	REQ.11
REQ.23	Užívateľ je schopný zobraziť všetky svoje DATASETS.	F	REQ.11
REQ.24	Užívateľ je schopný upraviť každý svoj DATASET.	F	REQ.11
REQ.25	Užívateľ je schopný zmazať každý svoj DATASET.	F	REQ.11
REQ.26	Aplikačné rozhranie poskytuje koreňovú cestu odkazujúcu na dokumentáciu aplikácie.	F	REQ.4
REQ.27	Komunikácia webového rozhrania so systémom nezastavuje vykresľovanie rozhrania počas získavania dát zo servéru.	F	-
REQ.28	Webové rozhranie aplikácie bude relevantne reagovať na nedostupnosť služieb systému, v prípade ich nedostupnosti.	F	REQ.3
REQ.29	Webové rozhranie aplikácie bude informovať užívateľa o všetkých relevantných aktivitách, pri ktorých prebieha komunikácia so systémom.	F	REQ.3
REQ.30	Webové rozhranie je funkčné vo webových prehliadačoch: Chrome, Firefox a Safari.	F	REQ.3
REQ.31	Webové rozhranie umožňuje zobraziť výsledky generovania vo formátoch: SQL, JSON, CSV a XLSX.	F	REQ.3
REQ.32	Webové rozhranie bude responzívne aj na mobilných zariadeniach.	NF	REQ.3
REQ.33	Systém neumožní prístup a priamu komunikáciu s databázovou instanciou z vonkajšieho prostredia systému.	NF	-

Tabuľka 2.1: Formálna špecifikácia požiadaviek pre Dataster.

Kapitola 3

Vývoj webových aplikácií

Vzhľadom na to, že hlavnou požiadavkou projektu je vytvorenie webového rozhrania a teda aj webovej aplikácie, bude potrebné vhodne zvoliť technológie, ktoré tvorbu webových aplikácií umožňujú. Táto kapitola sa bude zaoberať štúdiom a výberom relevantných webových technológií, ktoré budú poskytovať možnosti vývoja aplikácie podľa zadaných požiadaviek z predchádzajúcej kapitoly 2.

3.1 Webové technológie

Nasledujúce webové technológie boli zvolené ako relevantné nástroje umožňujúce dosiahnutie stanovených cieľov pre webovú aplikáciu Dataster.

JavaScript

JavaScript je multiparadigmatický interpretovaný skriptovací jazyk vytvorený pre kontext využitia na webových stránkach. JavaScript je však možné použiť aj v iných prostrediach ako vo webových prehliadačoch. Najčastejším prostredím pre interpretovanie JavaScriptu je prostredie **Node.js**[1], ktoré umožňuje využitie JavaScriptu aj pri tvorbe aplikácií na strane servera. Štandardom pre JavaScript je špecifikácia **ECMAScript**[2], ktorej 5. verziu podporujú od roku 2012 všetky moderné webové prehliadače [9].

Vue.js

Vue je JavaScriptový framework pre tvorbu užívateľských rozhraní. Jedným z hlavných príncipov a mechanizmov frameworku je použitie takzvaného tieňového DOM (Shadow DOM) modelu a používanie kombinácie viacerých stromových štruktúr DOM. Syntax Vue komponentov je založená na špecifikácii **Web Components**[3][11].

Každá aplikácia začína vytvorením instancie Vue v JavaScriptovom kóde. Ďalej je možné instanciu previazať s HTML dokumentom webovej stránky pomocou identifikátoru vybraného elementu v dokumente, ktorý sa tak stane koreňovým elementom Vue aplikácie. Vue pre tento element vytvorí tieňovú reprezentáciu DOM, ktorá umožňuje k elementom HTML pridávať dáta a funkcie JavaScriptového kódu [11].

[1] https://nodejs.org/
[2] https://www.ecma-international.org/
[3] http://w3c.github.io/webcomponents/spec/

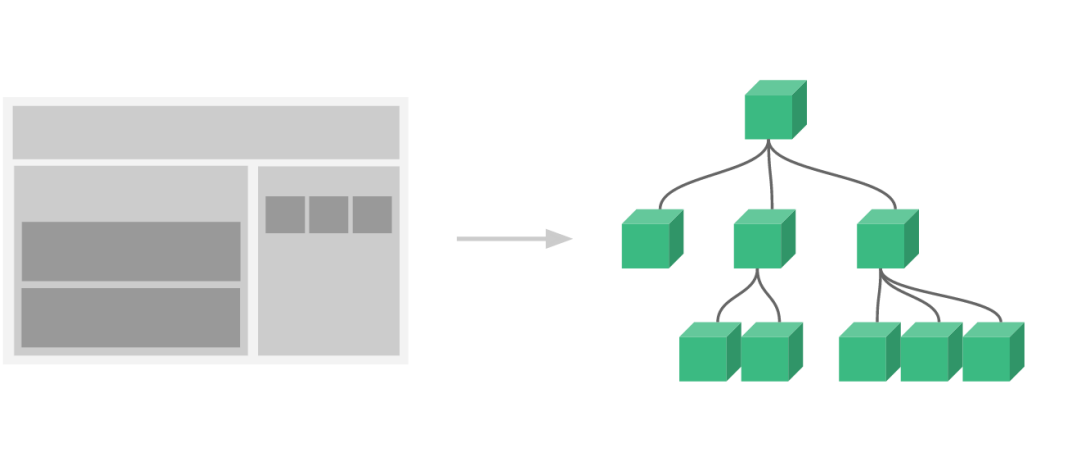
Framework Vue umožňuje vytvárať hierarchiu komponentov, ktoré je možné definovať v súboroch s príponou `*.vue`. Tieto súbory sú zložené z troch častí (elementov), ktoré spoločne vytvárajú jeden ucelený komponent. Element `<template>` poskytuje priestor na vytvorenie HTML šablony alebo vloženia iných Vue komponentov. V elemente `<script>` je možné vložiť JavaScriptový kód štandardným spôsobom, ako je to v bežných HTML dokumentoch. Špecifickým priestorom je v tomto elemente exportovací výraz `export default {}`, v ktorom je možné zadať rozširujúce atribúty pre komponent, ktoré framework Vue ponúka. Posledný element komponentu, `<style>` je priestorom, kde je možné definovať štýly špecifické pre daný komponent v izolovanom kontexte od zvyšku kódu.

```
<template></template>

<script></script>

<style></style>
```

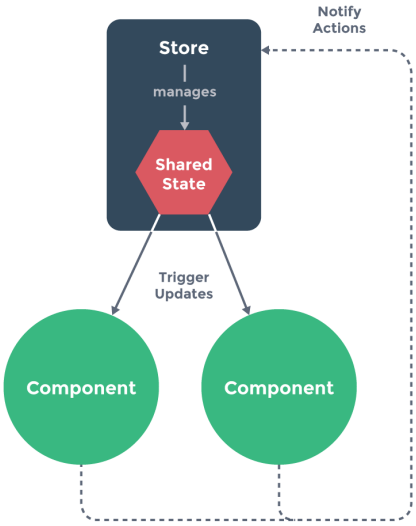
Vue komponenty je medzi sebou možné hierarchicky a rekurzívne používať, dynamicky načítavať a presúvať medzi nimi dáta alebo udalosti. Mapovanie vizuálnych komponentov rozhrania na stromovú štruktúru komponentov Vue je znázornené na obrázku 3.1.



Obr. 3.1: Hierarchia vue komponentov. Prevzaté z [https://vuejs.org/v2/guide/].

Framework je v základnej podobe veľmi jednoduchý no poskytuje dodatočné rozšírenia, ktoré jeho schopnosti významne rozširujú, najmä pre komplexné webové aplikácie.

Najviac významnými rozšíreniami Vue sú **Vue Router**, ktorý poskytuje možnosti takzvaného **Client-side routing** a rozšírenie **Vux Store**, ktoré umožňuje vytváranie a spravovanie globálneho stavu aplikácie. Práca s globálnym stavom pomocou Vuex Store je naznačená na obrázku 3.2.



Obr. 3.2: Globálny stav aplikácie. Prevzaté z [https://vuejs.org/v2/guide/].

Quasar.js

Quasar[4] je vývojové prostredie a framework, založený na prostredí **Node.js** a vývojovom nástroji **Webpack**. Zdokonaľuje a zrýchľuje vývoj aplikácií pre rôzne kontexty nasadenia (SPA, PWA, SSR, Electron a Cordova aplikácie).

Python

Vzhľadom na to, že ústrednou aplikačnou logikou aplikácie je modul **dbgenx**, ktorého vnútorná implementácia je v jazyku Python, bude vhodné na už existujúcu implementáciu nadviazať a modul dbgenx využiť ako závislosť. Bude tak možné využívať funkcie modulu viac flexibilne.

Flask

Flask[5] je jednoduchý, ľahko upraviteľný a pamäťovo nenáročný framework pre jazyk Python uľahčujúci tvorbu základných funkcií webovej aplikácie akými sú webový servér, nastavenie ciest rozhrania API a práca so žiadosťami a odpovedami protokolu HTTP.

Webpack

Webpack[6] je nástroj na takzvaný **bundling** (zabalenie) všetkých závislostí použitých modulov do sady statických súborov, ktoré jednoducho budú predané klientovi. Umožňuje

[4] https://quasar-framework.org/
[5] http://flask.pocoo.org/
[6] https://webpack.js.org/

štruktúrovať projektové súbory v prehľadnej forme a klientovi do prehliadača predať naopak optimalizovanú, no neprehľadnú formu kódu z projektu.

Poskytuje voliteľné možnosti optimalizácie, kompresiu, minifikáciu ... Viac o konkrétnych formách optimalizácie je uvedené v kapitole 6.

Nginx

Nginx je servérový softvér so širokým využitím pre rôzne kontexty. Poskytuje veľmi veľké množstvo funkcií, vhodných najmä pre webové a sieťové aplikácie. V základnej podobe je Nginx najmä HTTP servérom. Dokáže ale taktiež smerovať sieťové žiadosti, pracovať ako takzvaná reverzná proxy. Disponuje taktiež schopnosťou filtrovať žiadosti, komprimovať dáta odpovedí alebo zabezpečovať HTTP prenos pomocou SSL protokolu [10].

Databázové systémy

Keďže aplikácia bude do veľkej miery pracovať s persistentnými dátami, ktoré je niekam potrebné ukladať, je vhodné zvoliť uložiskový systém vo forme databázy.

Aplikácia nebude ukladať veľké objemy dát, dôležitým faktorom by mala byť rýchlosť a jednoduchosť používania DB systému.

MongoDB

MongoDB je dokumentovo orientovaná databázová služba. Nejedná sa teda o relačný databázový model ale dokumentový databázový model, ktorý využíva štruktúry medzi dokumentami považované za ekvivalentné relačnému databázovemu modelu.

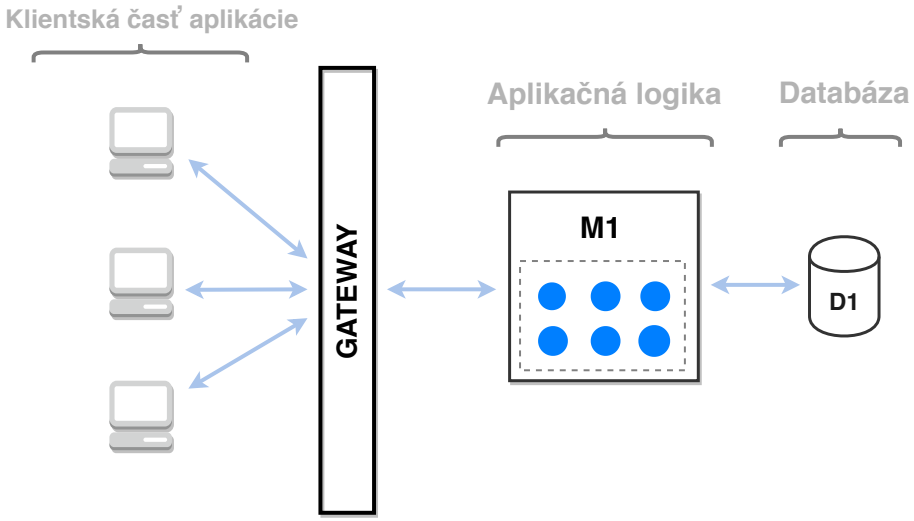
Základnými prvkami databázovej instancie MongoDB sú **kolekcie** (Collections), ekvivalenté tabuľkovým štruktúram. Každá kolekcia v sebe ukladá dáta v objektoch, nazývaných **dokumenty** (Documents). Každý dokument reprezentuje jeden záznam, ktorý sa skladá z niekoľkých párov v tvare **kľúč:hodnota**. Formát uloženia dát je veľmi podobný jazyku JSON [12].

3.2 Architektúra webových aplikácií

S narastajúcou zložitosťou vyvíjaných aplikácií je vhodné udržovať ich celkovú štruktúru podľa vopred stanovených pravidiel. V oblasti vývoja webových aplikácií sú infraštruktúra a systémový pohľad obzvlášť dôležité, keďže je potrebné už pri základných a jednoduchých aplikáciách komunikovať s klientom užívateľa. Minimálne teda vždy existuje predpoklad dvoch komponentov celkového webového komunikačného systému – klienta a serveru. Ak sa postupom vývoja zložitosť implementácie zvyšuje, vzniká tendencia separovať systém na ešte menšie submoduly a umiestniť ich do samostatných funkčných celkov a vytvárať tým prepojený systém komponentov.

Monolitická architektúra.

Monolitická architektúra umiestňuje všetky moduly do jedného spoločného priestoru, v ktorom sú dané moduly vzájomne úzko previazané. Systém takto pracuje buď ako celok alebo nepracuje vôbec.



Obr. 3.3: Monolitická architektúra.

Výhody monolitického návrhu

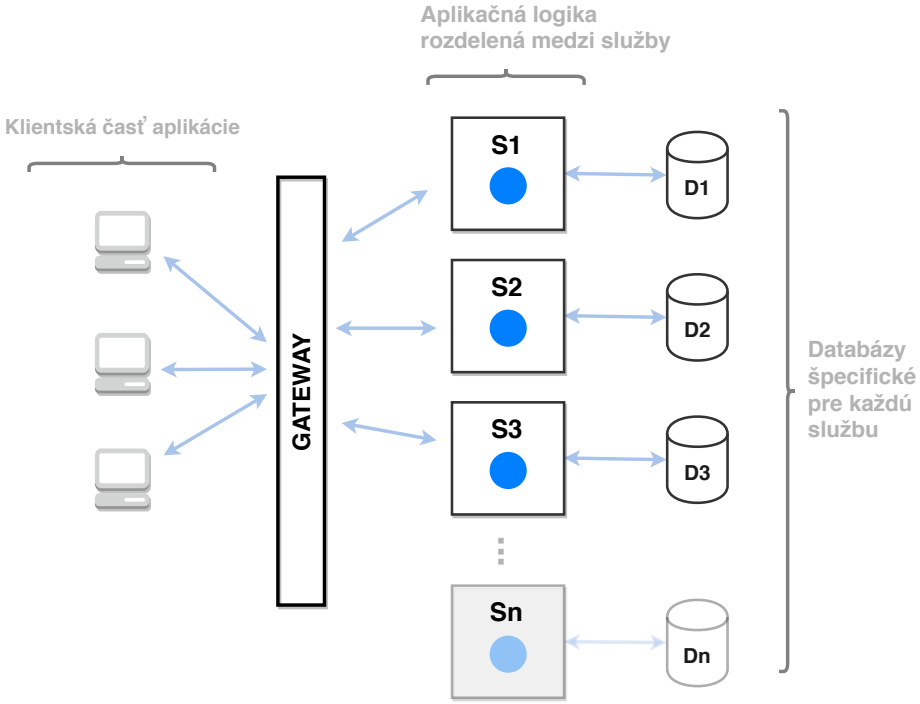
- Jednoduchšia architektúra.
- Len jeden deployment.
- Rýchlejšie lokálne volania.

Nevýhody monolitického návrhu

- Množstvo kontextu, zložitejšie na nadväzujúci vývoj v budúcnosti.
- Zložitejší a pomalší deployment, časté zmeny kódu, rôzne testovacie sady pre jednu službu.
- Riziko jediného bodu zlyhania.
- Tesné previazanie kódu medzi modulmi.

Architektúra založená na mikroslužbách

Architektúra mikroslužieb rozdeľuje komplikované aplikácie na ľahké a voľne prepojené komponenty [25].



Obr. 3.4: Architektúra mikroslužieb.

Výhody návrhu mikroslužieb

- Rozdelenie zodpovednosti medzi menšie celky.
- Jednoduchšie pre vývoj projektu v budúcnosti.
- Prehľadnejší analytický pohľad na systém.
- Jednoduchší a flexibilnejší deployment.
- Jednoduchšie spúšťanie automatizovaných testov.

Nevýhody návrhu mikroslužieb

- Komplexnejšia architektúra.
- Viac častí deploymentu.
- Pomalšie sieťové volania.

Zhodnotenie

Rozdiel medzi obidvomi návrhovými vzormi by sa dal zhrnúť nasledovne. Monolitický návrh vkladá všetku svoju funkcionalitu do jediného procesu. Architektúra mikroslužieb vkladá každý element funkcionality do samostatnej služby [18]. Obidva prístupy majú pre svoj kontext využitie. Pre kontext projektu Dataster je však zvolený návrhový vzor mikroslužieb,

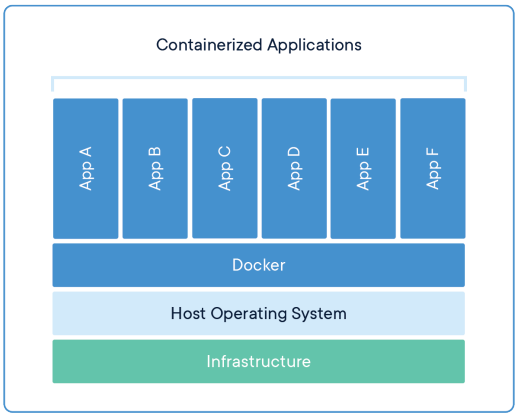
keďže práve vlastnosti tejto architektúry sú na základe požiadaviek najvhodnejšie. Mikroslužby predstavujú možnosť dekompozície jednotlivých služieb do samostatných, izolovaných segmentov, ktoré je možné v budúcnosti ďalej nezávisle rozvíjať alebo na ne nadväzovať. A práve táto kvalita potenciálu jednoduchého nadviazania a vývoja v budúcnosti je pre Dataster dôležitá.

V nasledujúcej kapitole 4 bude návrhový proces pracovať práve s návrhovým vzorom mikroslužby.

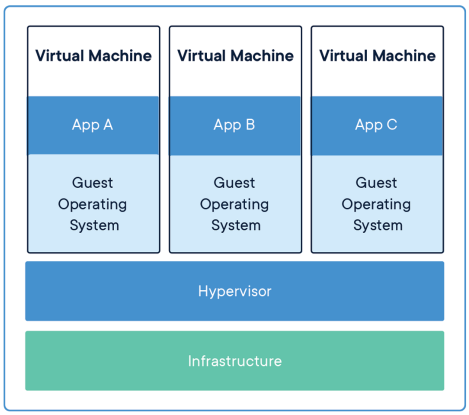
3.3 Docker a virtualizácia založená na kontajneroch

V súčasnej dobe je čoraz viac populárnejšia tzv. kontajnerizácia alebo virtualizácia založená na kontajneroch. Konkrétnym nástrojom umožnujúcim práve takúto formu virtualizácie je napríklad Docker.

Kontajnery zdieľajú nielen fyzické prostriedky, ale taktiež priamo operačný systém a taktiež podporné knižnice a moduly. Na rozdiel od tradičných virtuálnych strojov tzv. *Hypervisor based VMs*, ktoré poskytujú abstrakciu len na hardvérovej úrovni. Prínos kontajnerizácie v kombinácii napríklad s architektúrou mikroslužieb prináša možnosti zlepšenia, škálovateľnosti a elasticity vývoja aplikácií [25].



Obr. 3.5: Kontajnerizácia.



Obr. 3.6: Virtuálny stroj.

Obrázky 3.5 a 3.6 sú prevzaté z [https://docs.docker.com/].

3.4 Zhodnotenie podobných alebo existujúcich riešení

V oblasti generovania vzorových a náhodných dát existujú v súčasnej dobe viaceré nástroje. Množstvo z nich poskytuje veľmi solídne možnosti generovania v určitých aspektoch. Každý z nástrojov má aj svoje nevýhody, ktoré by pri tvorbe alternatívneho nástroja, akým by mal byť Dataster, mali byť zohľadnené.

generatedata.com[7]

Generate Data je webová aplikácia a služba implementovaná v JavaScripte, PHP a MySQL. Hlavnou funkciou je náhodné generovanie dát zo širokej ponuky datasetov. Rozhranie

[7] http://www.generatedata.com

umožňuje pre jednotlivé datasety vytvoriť stĺpce v tabuľke a vygenerovať tak množstvo komplexných záznamov podľa zadaného formátu tabuľky [5].

Výhodou tohto nástroja je skutočne široké spektrum datasetov, rôznych dátových typov a možností regionálnych jazykových tvarov dát. Ďalej ponúka množstvo výstupných formátov, v ktorých je možné dáta zobraziť.

Konfiguráciu pre generovanie dát je však možné zadať len pre jedinú tabuľku. Nie je teda možné vygenerovať dáta, ktoré by reprezentovali ich skutočnú podobu v databáze, v ktorej sú navzájom dané záznamy dát závisle prepojené. Nástroj taktiež neposkytuje možnosť zavádzať obmedzenia navzájom medzi stĺpcami tabuľky. Obmedzenia pre jednotlivé datasety platia len v rámci určitého stĺpca a komplexnejšie schémy nie je možné vytvoriť.

mockaroo.com[8]

Mockaroo je taktiež webovým nástrojom určeným na generovanie náhodných dát [6]. Podobne ako generatedata umožňuje definovanie konfiguračnej schémy pre jednu tabuľku, poskytuje na výber množstvo rôznych datasetov a formátov dát. Dodatočne umožňuje určiť štatistickú absenciu položky pre daný stĺpec vo vygenerovaných dátach.

Výhodou tohto nástroja je možnosť pridať do procesu generovania faktor pravdepodobnostného rozloženia. Docieli sa tým reálnejšia podoba dát, ktoré nebudú v priemere disponovať rovnakými hodnotami ale pri väčších množstvách riadkov, ktoré nástroj vygeneruje, sa pri generovaní prihliada na normálne pravdepodobnostné rozloženie a dáta pôsobia reálnejšie.

Nevýhodou tohto nástroja je opäť možnosť generovať dáta len pre jednu špecifickú štruktúru (tabuľku) a neschopnosť generovať relačné vzťahy medzi viacerými tabuľkami a ani stĺpcami.

[8] http://www.mockaroo.com

Kapitola 4

Návrh architektúry a plán práce

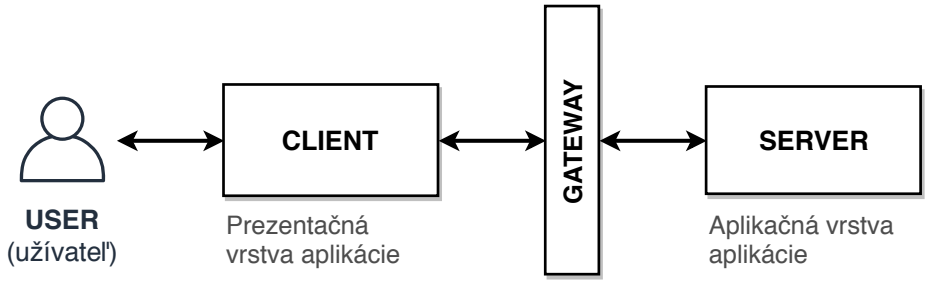
Návrh architektúry v tejto kapitole bude priamo nadväzovať na požiadavky z kapitoly 2 a poznatky o vývoji webových technológií získané v kapitole 3. Proces návrhu infraštruktúry bude prebiehať na niekoľkých vrstvách. Nasledujúce podkapitoly budú postupne prechádzať z vyšších vrstiev abstraktného pohľadu na nižšie. Proces bude zavŕšený detailným návrhom jednotlivých komponentov v systéme a pripravením základu pre proces implementácie.

4.1 Systémový návrh

Z konceptuálneho pohľadu je Dataster systém zložený z viacerých komponentov (podsystémov) a ich vzájomných prepojení. Predpokladom pre navrhnutie systémovej architektúry je vyhodnotenie jeho požiadaviek. Dobrý návrh modelu pre tento systém by mal brať do úvahy existenciu a vzájomnú súhru ako komponentov, tak aj ich vzájomných prepojení. Okrem toho bude systém vždy vložený do určitého prostredia a kontextu. Je teda potrebné myslieť aj na okolie systému.

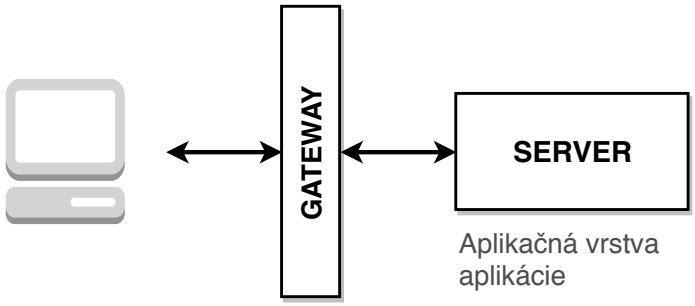
Požiadavky pre systém

V nasledujúcich odsekoch bude používaný pojem "služba", pomenúvajúci funkčné komponenty v rámci systému z pohľadu klienta. Ak by bol systém využívaný len prostredníctvom webového rozhrania, dalo by sa o systéme uvažovať ako o Klient-Server architektúre pomocou modelu 4.1.



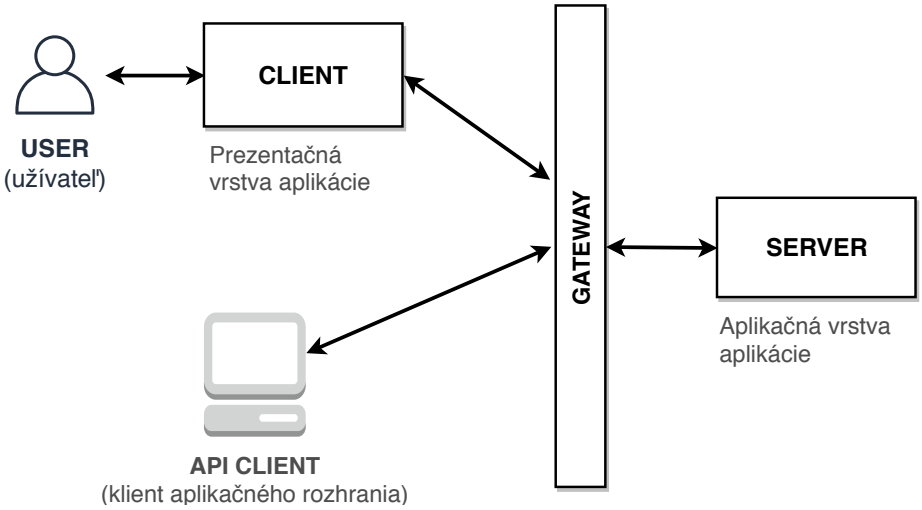
Obr. 4.1: Klient-Server - konceptuálny návrh pre webové rozhranie Dataster.

Keďže ale systém vyžaduje aj priame aplikačné rozhranie pre službu **generator** (požiadavka **REQ.4**), funkčné aj v prípade že webové rozhranie nie je dostupné, je potrebné vytvoriť návrhový model aj pre tento typ interakcie so systémom 4.2.



Obr. 4.2: Klient-Server - konceptuálny návrh pre API rozhranie Dataster.

Ak je požiadavkou súčasne aj to, aby tieto dva typy rozhrania existovali v systéme súčasne, architektúru systému možno vyjadriť modelom 4.3.



Obr. 4.3: Konceptuálny návrh celého systému Dataster.

Komponenty v systéme

Ako už bolo spomenuté, z požiadaviek plynie, že systém musí mať užívateľské a API rozhrania súčasne a nezávisle od seba. Zostáva však ešte niekoľko ďalších požiadaviek, ktoré treba zohľadniť.

Požiadavka **REQ.10**, vytvára potrebu o užívateľsky správny systém, umožňujúci užívateľom vytvárať osobné účty, ukladať pracovné dáta a zabezpečiť prístup k nim pomocou prihlasovacích údajov.

Súčasne požiadavky **REQ.8** a **REQ.9** vyžadujú, aby mohli užívatelia generovať dáta cez užívateľské ale aj API rozhranie. Vzniká teda rozdelenie funkcionality aplikačnej vrstvy na dva hlavné celky.

Služba **web-backend**, bude mať pod kontrolou všetky užívateľské účty, autentifikačný systém a bude taktiež ukladať všetky pracovné dáta užívateľov. K službe by mala taktiež patriť aj priľahlá databáza, v ktorej by dáta mali zostať perzistentne uložené.

Služba **generator**, bude mať výhradne na starosti generovanie dát pomocou submodulu dbgenx. Pre tento submodul vytvorí jednoduchý webový server a API rozhranie. V základ-

nej podobe pre túto službu nie je databázový systém potrebný, avšak systém by mal byť navrhnutý tak, aby bolo v prípade potreby možné databázu pridať.

Všetky komponenty (služby) systému Dataster:

- Služba **web-backend**.
- Služba **web-frontend**.
- Služba **generator**.

Systém môže disponovať aj inými komponentami, akými sú napríklad databázový systém, klient pre spúšťanie testov, atď.

Takýto návrh bude poskytovať lepšiu organizáciu internej štruktúry projektu, každá služba bude mat samostatný segment a funkcionalitu. Ďalej sa tým vytvorí voľné previazanie kódu a funkcionality medzi službami, čo znamená, že bude náročnejšie systém dostať do chybového stavu, v ktorom by bol kompletne neresponzívny alebo nefunkčný.

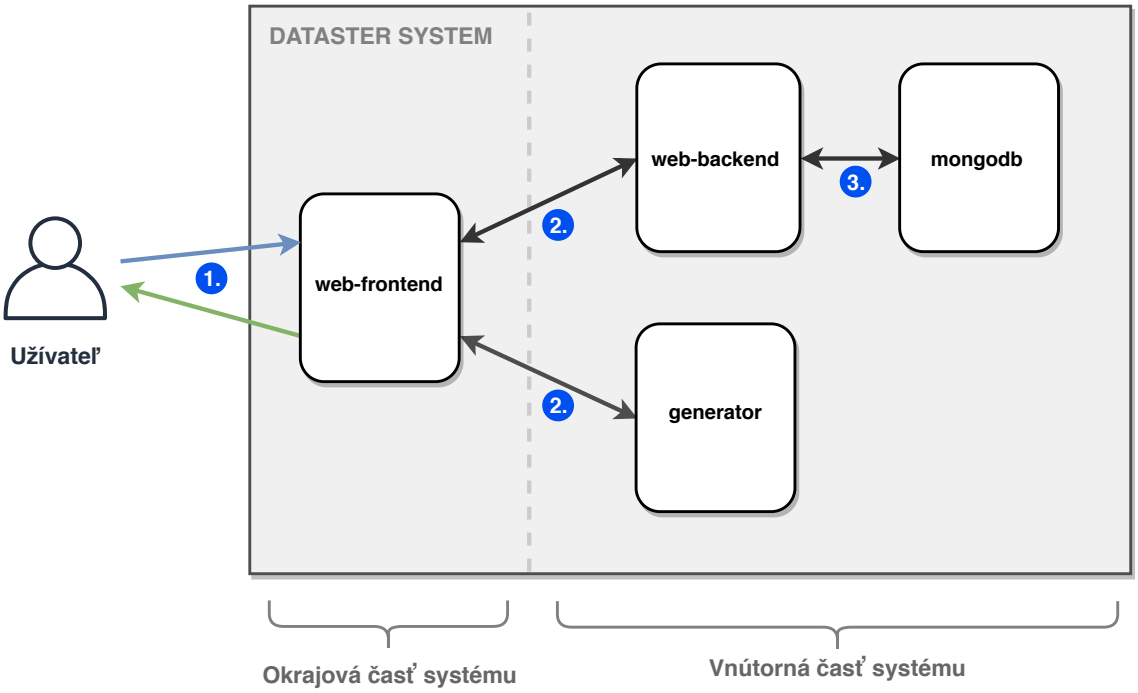
Väzby komponentov systému

Systém vyžaduje prostriedky pre externú a aj internú komunikáciu medzi komponentami. Služba web-frontend musí byť schopná komunikovať externe s klientmi pomocou protokolov HTTP a HTTPS. Vyžadované je taktiež to, aby mali užívatelia možnosť komunikovať s API pre službu generator taktiež z vonkajšieho prostredia. Pre *zjednotenie jedného prístupového bodu* by mala byť určená len jedná hraničná služba, ktorá by žiadosti na iné interné služby delegovala alebo prípadne filtrovala. Pre tento účel je vybraná služba web-frontend, ktorá bude pracovať ako reverzná proxy služba a okrem poskytovania statických súborov klientského rozhrania taktiež funguje ako smerovač žiadostí na web-backend a službu generator.

Jediný komponent, ktorý by mal komunikovať s okolím systému by mal byt **web-frontend**. Všetky ostatné komponenty by mali byť pre vonkajšie prostredie neviditeľné a ich funkcionalita prístupná len prostredníctvom služby web-frontend. V prípade existencie dodatočných služieb v systéme, ako napríklad databázovej služby alebo systému, by takáto služba mala byť pripojená len k tej službe, ktorá ju využíva. Služba **web-frontend** musí byť jediná služba, ktorá komunikuje s vonkajším prostredím aplikácie.

Finálny návrh systémovej architektúry

V priebehu práce bol model systémovej architektúry viackrát pozmenený. Finálny návrh celkového systému je predstavený na obrázku 4.4.



Obr. 4.4: Návrh architektúry systému Dataster. Architektúra systému je tvorená troma základnými komponentami: web-frontend, web-backend a generator. Dodatočne je v systéme zastúpený databázový systém mongodb, ktorý sa viaže k službe web-backend. Väzby medzi komponentami sú nasledovné: (1) web-frontend komunikuje s klientom, (2) web-fronted smeruje žiadosti od klienta na web-backend, (3) web-backend spravuje dáta z databázy mongodb, (4) web-frontend smeruje žiadosti od klienta na generator.

4.2 Návrh komponentov systému

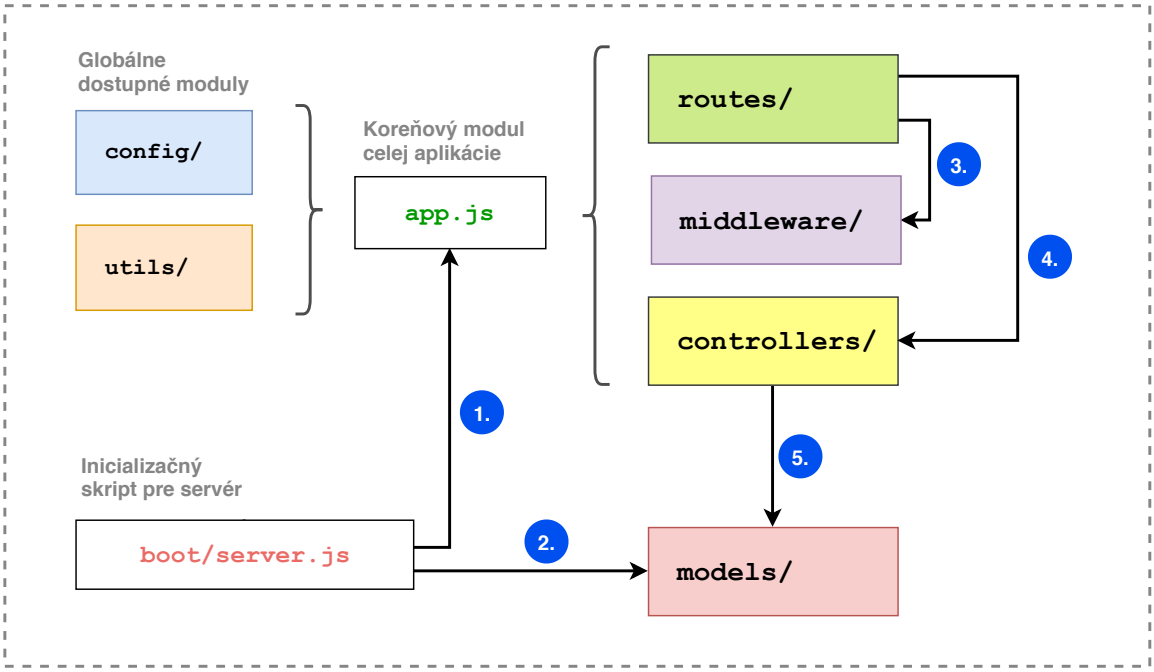
Každý komponent má iné potreby alebo požiadavky a tak je prirodzené aj pri návrhu infraštruktúry postupovať rozdielne.

Web-backend

Serverová časť webovej aplikácie by mala mať na starosti správu užívateľov, autentifikáciu pri prístupe do časti pre prihlásených užívateľov a taktiež ukladanie **recipes** a **datasets** vytvorených užívateľmi. Rozhraním pre aplikáciu by malo byť REST API rozhranie na vhodne stanovených cestách.

Pre tieto potreby je navrhnutá vnútorná štruktúra zdrojového kódu sčasti inšpirovaná návrhovým vzorom MVC (Model-View-Controller)[1]. Štruktúra je ilustrovaná na obrázku 4.5.

[1] https://en.wikipedia.org/wiki/Model-view-controller



Obr. 4.5: Štruktúra serverovej časti webovej aplikácie.

Vnútorná architektúra backendu je inšpirovaná návrhovým vzorom MVC. Avšak priamo neobsahuje časť *view*, tá je kompletne separovaná do samostatného projektu a služby **web-frontend**.

Miesto toho poskytuje rozhranie vo forme čistého REST API [14]. Jednotlivé cesty tohto rozhrania definuje modul `routes`, na ktorý sa priamo viažu jednotlivé kontroléry (modul `controllers`). Tie pomocou schém pre dátové modely, definované v module `models`, spravujú dáta v databáze. Modul `middleware` poskytuje priebežnú kontrolu alebo zachytenie určitých žiadostí počas ich toku od klienta k serveru. Vedlajšími modulmi sú moduly `config` a `utils`, v ktorých sa nachádzajú definície pomocných funkcií a nastavení pre konštatné premenné aplikácie pri spustení.

Web-frontend

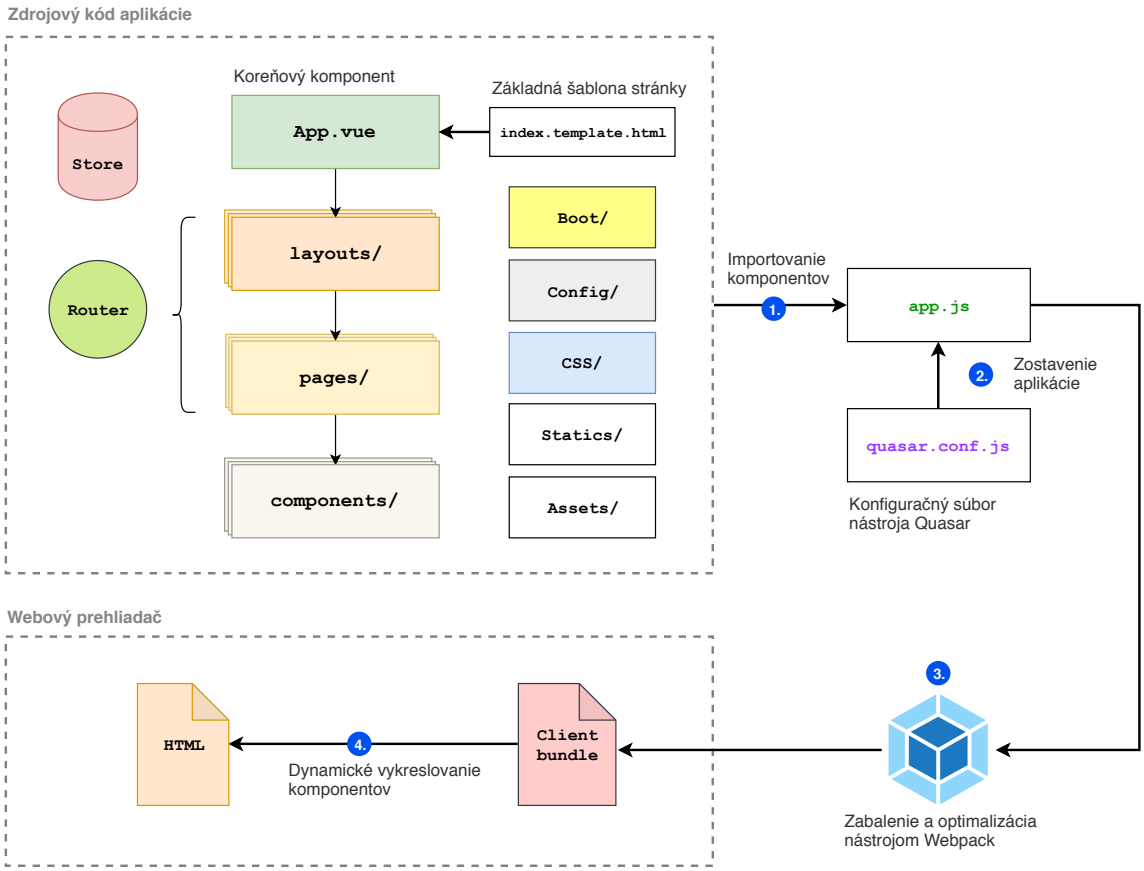
Hlavným zmyslom služby **web-frontend** je poskytnutie webového rozhrania, tak ako to vyžaduje požiadavka **REQ.3**.

Vnútorná štruktúra klientskej časti aplikácie závisí od fázy a kontextu, v ktorom sa práve projekt nachádza. Pri vývoji je štruktúra projektu rozvrhnutá do prehľadnej stromovej štruktúry, v produkčnej fáze sa z projektu bude používať len výstupná sada súborov určených na doručenie ku klientovi.

Počas vývoja bude mať aplikácia štruktúru tvorenú hierarchiou Vue komponentov. Okrem toho sa v nej nachádza modul Router (obsluhujúci client-side routing), Store (spravujúci vnútorný stav klientskej časti rozhrania) a dodatočné statické súbory, používané naprieč celým rozhraním. Takáto abstrakcia umožní prehľadnejšie a modulárnejšie písanie kódu. Výsledný, zostavený a zabalený, kód aplikácie pomocou nástroja **Webpack** bude

umiestnený do adresára `/dist/`, v ktorom sa budú nachádzať už len produkčné statické súbory určené na nasadenie. Bližšiu predstavu môže poskytnúť obrázok 4.6.

Na druhej strane, pri umiestnení aplikácie do produkcie, bude služba operovať ako server, ponúkajúci práve statické súbory vygenerované z predchádzajúcej fázy.



Obr. 4.6: Štruktúra klientskej časti webovej aplikácie a proces transformácie zdrojového kódu aplikácie na balík súborov určených na nasadenie do produkčného prostredia.

Generator

Hlavným zmyslom služby **generator** je poskytnutie aplikačného rozhrania (API) pre funkciu generovania pomocou modulu **dbgenx**. Pre tento účel je v službe vytvorený webový server využívajúci submodul dbgenx.

Navrhnuté riešenie reaguje na požiadavku **REQ.6**, ktorá požaduje, aby generovanie prebiehalo za pomoci modulu **dbgenx**. Služba **generator** zabezpečí, aby sa žiadosti predané protokolom HTTP transformovali na ekvivalentné generovacie úlohy z modulu **dbgenx**, čím sa zabezpečí splnenie požiadavky **REQ.8**.

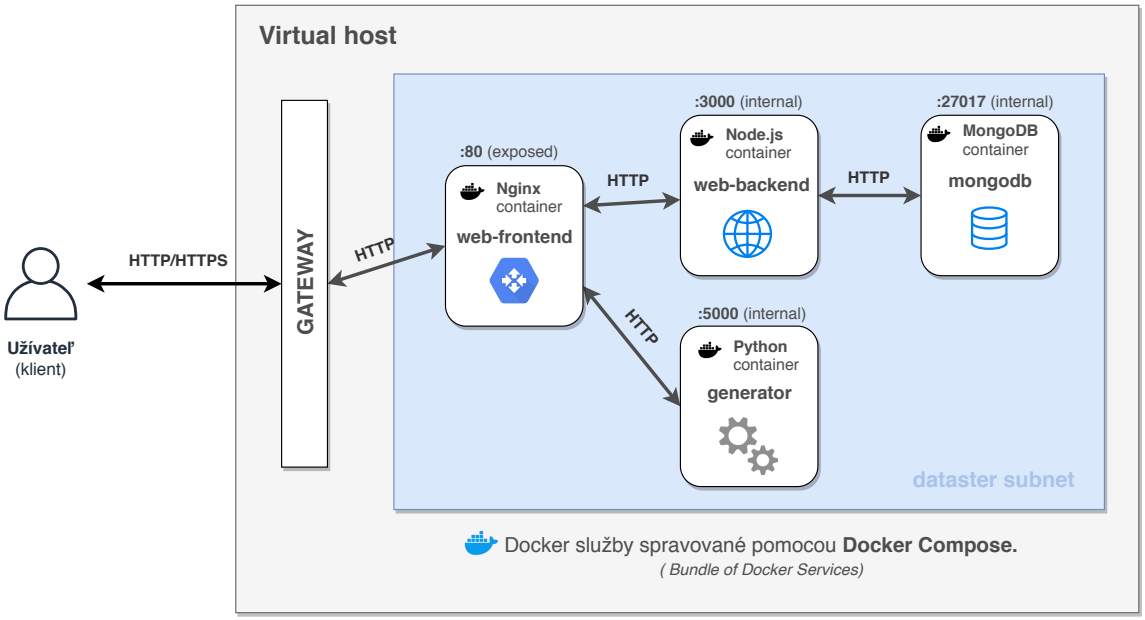
4.3 Návrh virtualizácie pomocou kontajnerov

Pre systém mikroslužieb je kontajnerizácia vhodným prístupom, ako jednotlivé služby a ich implementáciu separovať do samostatných celkov, s vlastnými závislosťami, rozhraniami a potrebami špecifickými pre nasadenie do produkcie.

Pre navrhnutú architektúru z abstraktného hľadiska, by teda mala byť navrhnutá aj modelová reprezentácia služieb nasadených do kontajnerov a kontextu spustenia pomocou nástroja Docker.

Kompozíca služieb v nástroji Docker Compose

Docker Compose[2] je nástroj, ktorý stavia na schopnostiach technológie Docker a pre služby, ktoré majú byť umiestnené v kontajneroch vytvára určitú kompozičnú schému. Jednak umožňuje prehľadne zadefinovať všetky služby v distribuovanej aplikácii, ale taktiež poskytuje centralizované konfiguračné možnosti, nastavenia rozhraní pre služby a vytvorenie internej podsiete, cez ktorú môžu služby komunikovať [7]. Pre kontext navrhnutej architektúry pre Dataster v tejto kapitole je kompozičná schéma kontajnerov znázornená na obrázku 4.7.



Obr. 4.7: Kontajnerová architektúra systému vrátane implementačných detailov.

Okrem iného je v tomto návrhu riešená požiadavka **REQ.2**, ktorá vyžaduje aby klientská časť aplikácie – **web-frontend** bola spustiteľná samostatne a nezávisle od zvyšku systému. Nasadenie služby, ktorá klientskú časť aplikácie zabezpečuje, do kontajneru vytvorí virtuálne prostredie pre Nginx server, ktorého beh nebude ovplyvnený dostupnosťou zvyšných služieb systému. Ďalej sa v návrhu reaguje na požiadavku **REQ.33**, ktorá vyžaduje aby systém neumožnil priamy prístup k databázovej službe z vonkajšieho prostredia. V celom systéme je jedinou službou, ktorá má pre vonkajšie prostredie otvorený port služba **web-frontend**, zvyšné služby majú porty otvorené len pre vnútornú podsieť Docker Compose.

[2] https://docs.docker.com/compose/

4.4 Organizačná štruktúra vývoja projektu

Z návrhu vyplýva, že architektúra systému bude zložená z niekoľkých samonosných častí. Aby sa predišlo neprehľadnosti pri vývoji jednotlivých častí, je zavedená prehľadová štruktúra popisujúca hlavné celky projektu, na ktorých bude prebiehať vývoj.

Vývojové línie projektu

Nasledujúci zoznam popisuje základné vývojové línie projektu a poskytuje nadhľad nad všetkými podstatnými časťami projektovej štruktúry.

1. **Organizácia projektu** - repozitár *testos/dataster*.
   Meta informácie, jasný pohľad na systém a plánovanie všetkých vývojových častí projektu.
   - Požiadavky projektu.
   - Dokumentácia projektu.
   - Inštalácia, zostavenie, nasadenie do produkcie.
   - Nastavenie produkčného prostredia.
   - CI/CD.
2. **Web Frontend** - repozitár *testos/dataster*.
   Samonosná aplikácia/služba klientskej časti webovej aplikácie - frontendu.
   - Užívateľské grafické rozhranie.
   - Server pre statické súbory webového rozhrania.
   - Projektové súbory Vue.js/Quasar.js.
   - Reverzná proxy (Nginx).
3. **Web Backend** - repozitár *testos/dataster*.
   Samonosná aplikácia/služba backendu.
   - Užívateľský správny systém.
   - Autentifikačný server.
   - Správa "Recipesä "Datasets".
4. **Generator** - repozitár *testos/dataster-service*.
   Samonosná aplikácia/služba generátoru dát.
   - Využitie aplikačnej logiky **dbgenx**.
   - Webový server Flask a API rozhranie.

Vývojové repozitáre

Keďže je štruktúra aplikácie distribuovaná medzi viaceré služby, dá sa predpokladať, že vývoj bude prebiehať paralelne v jednotlivých častiach nezávisle od seba.

Ak by boli všetky zdrojové súbory umiestnené v jedinom repozitári, bolo by náročnejšie zavádzať zmeny, testovať moduly v izolácii a opravovať chyby, špecifické len pre niektorú službu. Z tohto dôvodu boli zavedené samostatné repozitáre pre službu **generator** a **dbgenx**. Služby **web-frontend**, **web-backend** a všetky prvky dôležité pre systém ako celok budú umiestnené v hlavnom, koreňovom repozitári **testos/dataster**, ktorý ostatné repozitáre ukladá v stromovej štruktúre ako git submoduly[3].

```
pajda.fit.vutbr.cz/testos
└─ dataster
   └─ dataster-service
      └─ dbgenx
```

[3] https://git-scm.com/docs/gitsubmodules

Kapitola 5

Implementácia komponentov systému

Táto kapitola sa zameriava na implementačné detaily jednotlivých komponentov ale aj celého systému.

5.1 Vývoj a nasadenie v kontajneri

Jednou z najsilnejších výhod mechanizmu kontajnerizácie je ekvivalencia prostredí, v ktorých sú aplikácie a služby spúšťané, nezávisle od hostiteľského systému. Vďaka tomu je možné vývojové a produkčné prostredie simulovať na akomkoľvek systéme a pracovať s identickými vlastnosťami prostredia, v ktorom bude finálne systém nasadený.

Pri vývoji Datasteru je možnosť spustenia v kontajneri aktívne využívaná a poskytuje možnosť efektívne pracovať so zdrojovým kódom aplikácie a súčasne vidieť reflektované zmeny do prostredia systému a jeho služieb.

5.2 Implementačné problémy jednotlivých komponentov

V rámci procesu transformácie návrhu systému a jeho komponentov do reálnej a funkčnej podoby, je potrebné vyriešiť niekoľko implementačných detailov a problémov špecifických pre kontext každého komponentu zvlášť.

Web-backend

Všetky hlavné funkcie je potrebné sprístupniť prostredníctvom API rozhrania. Hlavnými funkciami služby **web-backend** sú:

- autentifikácia,
- správa osobných účtov a
- správa užívateľských dát ako sú recipes a datasety.

Datové modely užívateľských a pracovných dát

Služba **web-backend** pracuje s troma datovými modelmi (definicia schém sa nachádza v adresári `/models`).

Pre každý dátový model je vytvorená špecifická schéma, zložená z povinných a nepovinných položiek. Nasledujúce schémy, naznačené vo výpisoch 5.1, 5.2 a 5.3, používa v službe **web-backend** modul **mongoose**[1], ktorý uľahčuje prácu s databázou MongoDB a poskytuje možnosť dátové objekty jednoducho modelovať.

```
{
    firstname: {type: String},
    lastname: {type: String},
    username: {type: String},
    email: {type: String},
    password: {type: String}
}
```

Výpis 5.1: **User scheme** – schéma dátového modelu pre užívateľské dáta.

```
{
    recipename: {type: String},
    description: {type: String},
    owner: {type: mongoose.Schema.Types.ObjectId},
    configuration: {
        types: {type: Object},
        schema: {type: Object},
        generate: [{type: String}],
        include: [{type: String}],
        constraints: [{type: String}]
    }
}
```

Výpis 5.2: **Recipe scheme** – schéma dátového modelu pre recipe.

```
{
    datasetname: {type: String},
    description: {type: String},
    owner: {type: mongoose.Schema.Types.ObjectId},
    location {
        type: String
    }
}
```

Výpis 5.3: **Dataset scheme** – schéma dátového modelu pre dataset.

Vďaka týmto dátovým modelom, ktoré systém dokáže spravovať, je možné pokryť funkcionálne požiadavky **REQ.10**, **REQ.12**, **REQ.13**, **REQ.14**, **REQ.18**, **REQ.19**, **REQ.20**, **REQ.21**, **REQ.22**, **REQ.23**, **REQ.24** a **REQ.25**.

Autentifikácia

Pre účely overovania totožnosti sa v implementácii služby web-backend využíva mechanizmus JSON Web Tokens[2]. Ide o spôsob reprezentovania pohľadávok (claims) medzi dvoma stra-

[1] https://mongoosejs.com/
[2] https://jwt.io/

nami pri sieťovej komunikácii. Pohľadávky JWT je možné digitálne podpísať a zabezpečiť tak integritu pri komunikácii medzi dvoma stranami [16].

JWT technológia sa využíva vo väčšine prípadoch použitia služby web-backend, kedy je daný token vždy očakávaný v tele prichádzajúcej žiadosti. Či už ide o správu užívateľských dát alebo úpravu pracovných dát ako sú recipes a datasets.

Token sa vytvára pri vytváraní účtu alebo prihlasovaní (**UC.2** a **UC.3**), pričom jeho hodnota sa spolu s odpoveďou vracia ku klientovi. Klient by si mal token uložiť a pri každej nasledujúcej žiadosti ho pri komunikácii so serverom pridať do hlavičky správy. Webové rozhranie by malo token ukladať v lokálnej pamäti webového prehliadača a pri odhlásení ho z pamäte odstrániť.

Služba **web-backend** tak vytvára mechanizmus, vďaka ktorému je možné splnenie požiadaviek **REQ.11** a **REQ.17**. Dodatočne by mala služba v prípade neoverenia totožnosti, pri zadaní nesprávnych alebo neexistujúcich údajov, odpovedať chybovým kódom a žiadny token užívateľovi neposkytnúť. Tým sa splnia požiadavky **REQ.15** a **REQ.16**. Pre ukážkové a testovacie účely je v produkčnej konfigurácií systému zavedený demonstračný užívateľský účet, do ktorého je možné sa prihlásiť pomocou údajov v tabuľke 5.1.

Rola	Prihlasovacie meno	Prihlasovacie heslo
Registrovaný užívateľ	**tester**	**password**

Tabuľka 5.1: Vzorové prihlasovacie údaje, ktoré je možné použiť pre prihlásenie do produkčnej verzie Dataster, ktorá je dostupná na adrese `https://dataster.testos.org/`.

Web-frontend

Hlavnou funkciou služby **web-frontend** je poskytnutie webového rozhrania. V samotnom webovom rozhraní prebieha množstvo akcií, pre ktoré je potrebné ukladať stav. Na tieto účely je použité rozšírenie Vue – Vuex Store. Ďalšiou významnou funkciou webového rozhrania je tzv. client-side routing, ktorý zabezpečuje rozšírenie Vue – Vue Router. Toto rozšírenie poskytuje spracovanie URL cesty vrámci klientského JavaScript kódu, nie je teda potrebné kontaktovať server pri navigovaní stránky [3].

Vuex Store

Na ukladanie globálneho stavu klientskej časti aplikácie využíva aplikácia rozširujúci modul frameworku Vue, **Vuex Store**[3]. Vrámci Vuex Store sú ďalej vytvorené submoduly, podľa kategórie dát s ktorými v aplikácii pracujú. Ich počiatočný stav je popísaný vo výpise 5.4.

```
store
├── auth
├── user
├── gen
├── recipe
└── dashboard
```

[3] `https://vuex.vuejs.org/`

```
auth: {
    authToken: '',
    status: '' // loading, success, error
},
dashboard: {
    colorMode: 'light-mode' // or 'dark-mode'
},
gen: {
    result: {},
    status: '' // loading, success, error
},
recipe: {
    recipeData: {}
    status: ''
},
user: {
    userData: {},
    recipeCount: 0,
    datasetCount: 0,
    generatedCount: 0,
    status: '' // loading, success, error, updating, none
}
```

Výpis 5.4: Počiatočný stav aplikácie.

Nginx server

Všetky dáta určené pre klienta a zároveň smerovanie na zvyšné služby systému má server Nginx, bežiaci v službe web-frontend. Okrem poskytovania statických súborov a smerovania na služby web-backend a generator, má na starosti napríklad aj obmedzovanie množstva a intenzity žiadostí metódou **traffic shaping** (riadenia prevádzky). Traffic shaping je technika spravovania šírky pásma pri komunikácii so serverovou službou. Proces traffic shapingu sa dá popísať ako proces cieleného oneskorenia určitých paketov v rámci prúdu dát za účelom podriadenia prúdu podľa určitého profilu [13].

V rámci konfigurácie pre nginx server, uloženej v súbore `nginx.conf`, je špecifikované obmedzenie pre množstvo povolených žiadostí od jedného klienta za časovú jednotku. Výťah z konfigurácie je poskytnutý vo výpise 5.5. Dodatočne pre každú službu, na ktorú nginx vo funkcii reverznej proxy smeruje, je pridané rozširujúce nastavenie obmedzenia povolených žiadostí o parameter zhluku (burst), ktorý určité množstvo akceptovaných, no hneď neobslúžených žiadostí.

Týmito obmedzeniami sa vytvára profil založený na algoritme **leaky bucket**, ktorý každú prichádzajúcu žiadosť zaradí do čakacej rady na obslúženie podľa **first-in-first-out** (FIFO) plánovacieho algoritmu. Jednotlivé žiadosti budú obslúžené, až keď im to algoritmus povolí a žiadosti, ktoré sa do čakacej rady nezmestia, budú serverom odmietnuté a neobslúžené [21].

```
limit_req_zone $request_uri zone=zone1:10m rate=5r/s;
```

Výpis 5.5: Nastavenie limitujúce množstvo žiadostí za sekundu v konfigurácií NGINX.

Generator

Služba **generator** očakáva na API rozhraní HTTP žiadosti s telom vo formáte JSON, z ktorých práve tieto JSON dáta deserializuje a ďalej ich predáva na generovanie modulu **dbgenx**. Týmto postupom sa rieši požiadavka **REQ.5**. Okrem toho musí služba v prípade neúspešného generovania, z dôvodu nesprávneho formátu recipe, relevantne odpovedať pomocou chybovej správy. V impleméntacii sa takáto sitúacia rieši odchytením vyvolanej výnimky v prípade chybného spracovania konfiguračného súboru modulom dbgenx a navrátením chybového kódu v HTTP odpovedi. Tým sa pokrýva aj požiadavka **REQ.7**.

Ukážku približnej vnútornej implementácie transformácie žiadosti na generovanie na generovaciu úlohu pomocou modulu dbgenx môže poskytnúť výpis 5.6.

```
def generate():
    # (1.) deserialize JSON data from request
    recipe = request.get_json()
    # (2.) parse generates/constraints from str -> smt format
    generates = get_generates(recipe.get('generate', []))
    constraints = get_constraints(recipe.get('constraints', []))
    # (3.) parse recipe data from request
    input_data = {
        'dataset_names': recipe.get('include', []),
        'types': recipe.get('types', None),
        'generates': generates,
        'constraints': constraints,
        'schema': recipe.get('schema', None)
    }
    # (4.) create relational db table scheme
    sql_generator = SQL(input_data.get('schema'))
    # (5.) initiate generating task
    generate_data(
        dataset_names=input_data.get('dataset_names'),
        types=input_data.get('types'),
        generates=input_data.get('generates'),
        constraints=input_data.get('constraints'),
        sql_generator=sql_generator,
        dataset_path=dataset_path
    )
    # (6.) return data with response
    return app.response_class(
        response=json.dumps({
            'recipe': input_data,
            'sql': sql_generator.sql
        })
    )
```

Výpis 5.6: Naznačenie vnútornej implementácie API pre modul generátoru dbgenx.

5.3 Ekosystém mikroslužieb

Spustenie systému je vždy ovplyvnené určitými okolnými faktormi. Či už kontextom, z pohľadu umiestnenia na sieti, v ktorom je aplikácia nasadená, alebo špecifickými premennými pre danú instanciu aplikácie. Nasledujúce vlastnosti projektu sa týkaju systému ako celku a vytvárajú určitú dynamickosť pri spustení systému Dataster v závislosti na kontexte.

Premenné prostredia

Významnou súčasťou spustenia behu celého systému Dataster v produkčnej konfigurácii sú premenné prostredia. **Environment variables** (premenné prostredia) sú premenné, ktorých hodnota je nastavená vo vonkajšom prostredí bežiaceho procesu. Env. variable je zložená z dvojice `kľúč:hodnota`, vďaka čomu je možné podľa názvu kľúča danú env. variable referencovať v zdrojovom kóde programu a počas jeho behu, sa referencie nahradia konkrétnymi hodnotami, ak daná premenná prostredia skutočne existuje [19].

V rámci aplikácie Dataster sa používaju env. variables veľmi často, najmä v kontexte spustenia v produkčnej konfigurácii, kedy je potrebné nastaviť nutne špecifické údaje o prostredí, v ktorom je aplikácia spúštaná. Miestom, kde je možné nastaviť všetky dôležité env. variables pre aplikáciu, je súbor `.env` v koreňovom adresáry projektu. Z tohto súboru si všetky potrebné premenné načíta Docker Compose pri spúšťaní produkčnej konfigurácie celého systému.

Šablona obsahujúca všetky potrebné env. variables, ktoré je potrebné nastaviť, je uložená v súbore `production_template.env` v koreňovom adresári projektu. Ukážka jej obsahu je zobrazená vo výpise 5.7.

```
DATASTER_HTTP_PORT=8080
MONGO_USER=root
MONGO_PASSWORD=root_password
JWT_SECRET=sampleJWTsecret
JWT_EXPIRATION=10000
BACKEND_API_URL=http://localhost:8080
GENERATOR_API_URL=http://localhost:8080
GENERATOR_API_KEY=secret_api_key
```

Výpis 5.7: Šablona pre nastavenie premenných prostredia v produkcii.

Pomocou týchto premenných je možné nastaviť potrebné údaje ako je napríklad sieťový port, na ktorom by mala byť aplikácia dostupná prostredníctvom HTTP protokolu (požiadavka **REQ.1**).

Perzistencia dát

Pre dlhodobé zachovanie dát aplikácie aj po opakovanom reštartovaní celého systému je používaný mechanizmus **Docker Volume**[4], ktorý umožňuje perzistentné zachovanie dát vytvorených a používaných kontajnermi. Dáta obsiahnuté vo *volume* zostávajú uložené na súborovom systéme hostiteľského systému, nezávisle od celého systému.

V projekte Dataster sú dve lokality používané ako *volumes*. Pre dáta databázy MongoDB a služby **mongo**, ktorá ju používa, je vyhradený volume v adresári `/data/db`. Druhý volume

[4] https://docs.docker.com/storage/volumes/

s adresárom `/data/datasets` je úložiskom pre datasety používané službou **generator**. Obidva adresáre je možné voľne presúvať medzi instanciami Datasteru a vytvárať tak ekvivalentný stav v rôznych instanciách spustenia.

Tolerancia chýb v produkcii

Aby sa predišlo nedostupnosti niektorých služieb systému, bude v produkcii aplikovaná v určitej miere tolerancia chýb a výnimiek, ktoré v špeciálnych prípadoch môžu nastať. Hlavné aplikačné procesy služieb **generator** a **web-backend** budú spravované pomocou nástroja spravovania procesov, **PM2**[5], ktorý umožňuje spravovať stav aplikácie.

V produkčnej konfigurácii je pre obidve služby nastavený správca procesov tak, aby v prípade chybového návratového kódu proces aplikácie reštartoval a zaručil tak neustálu dostupnosť služieb aj v prípade, že nastane vnútorné zlyhanie.

[5] http://pm2.keymetrics.io/

Kapitola 6

Užívateľské a aplikačné rozhrania

Z pohľadu užívateľa je najdôležitejšou časťou celého systému práve jeho rozhranie, cez ktoré môže s aplikáciou komunikovať. V rámci aplikácie Dataster sa nachádzajú dva typy rozhraní, webové grafické rozhranie a aplikačné rozhranie (API). V tejto kapitole prebehne návrh a popis obidvoch typov rozhraní.

6.1 Návrh webového rozhrania

Webové rozhranie je grafické užívateľské rozhranie umiestnené na webovej stránke. Cieľom webového rozhrania by malo byť uľahčenie práce s aplikáciou, abstrakcia detailov, ktoré bežný užívateľ nemusí ovládať. V svojej podstate by malo takéto rozhranie mapovať schopnosti rozhrania API na vizuálne komponenty webovej stránky (požiadavka **REQ.9** 2.1) a tak priniesť intuitívne a pre človeka známe ovládanie aplikácie.

Návrh užívateľského rozhrania sa dá v určitých aspektoch prirovnať k návrhu architektúry budov. Podobnosťou medzi na prvý pohľad odlišnými odvetviami je silný dôraz na popredie, fasádu budovy v oblasti architektúry. Najmä z dôvodu, že práve fasáda uvádza a predstavuje vstup do budovy a je zároveň prvým prostriedkom interakcie, s ktorým sa väčšina ľudí dostane do styku a vytvorí si dojem o celej budove. Funguje tak ako prostriedok, artikulujúci vzťah medzi vonkajškom a vnútrom [24].

Obdoba tohto princípu platí aj pre digitálne užívateľské rozhrania, ktoré sú podobne časťou systému, ktorú užívatelia uvidia ako prvú vec a vytvoria si tak dojem o celej aplikácií.

Dodatočne z tejto analógie plynie aj ďalší poznatok, a to že užívatelia vnímajú esteticky príťažlivé rozhrania ako indikáciu **použiteľných systémov** [24]. Pri tvorbe a návrhu webového rozhrania by teda mala byť zohľadnená aj estetická kvalita rozhrania popri jeho použiteľnosti. Zvýši sa tým vnímaná kvalita použiteľnosti rozhrania z pohľadu užívateľa.

Ďalšou požiadavkou (**REQ.32**) na rozhranie je očakávaná responzívnosť zobrazenia na mobilných zariadeniach. Neočakáva sa, že by bol nástroj prostredníctvom mobilných webových prehliadačov používaný na samotné generovanie, no minimálne zobrazenie statických stránok rozhrania, prípadne najpodstatnejších dát osobného účtu by malo byť umožnené.

Základná štruktúra webového rozhrania

Prvá fáza pri návrhu rozhrania je návrh jeho globálnej štruktúry, v rámci celej webovej stránky. Pre aplikáciu Dataster, ktorá je zložená z niekoľkých diskrétnych častí, je potrebné

logicky rozčleniť jednotlivé časti rozhrania do samostatných celkov, medzi ktorými bude možné intuitívne navigovať.

Proces návrhu celkovej globálnej štruktúry vychádza z niekoľkých základných scenárov použitia, pri ktorých sa dá predpokladať určitá nadväznosť akcií užívateľa. Tieto scenáre sa snažia reflektovať a pokryť vopred špecifikované prípady použitia v diagrame 2.1.

USE CASE SCENARIO [1]: Základný predpokladaný scenár používania. Užívateľ si vytvorí účet a vygeneruje dáta.

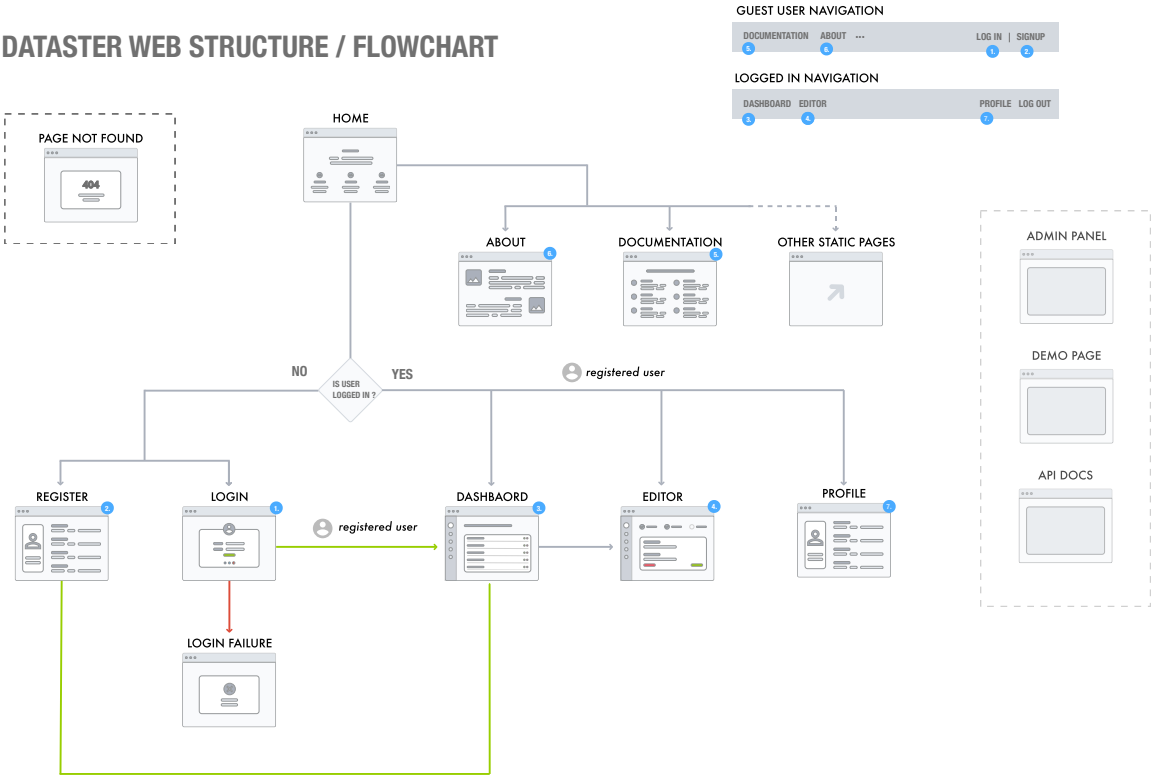
Obr. 6.1: Prvým scenárom je kombinácia vytvorenia nového osobného účtu, následne vytvorenia nového konfiguračného súboru recipe a na záver spustenie generovania dát a zobrazenie výsledku. Scenár popisuje prípady použitia: **UC.2**, **UC.7** a **UC.9** 2.1.

USE CASE SCENARIO [2]: Užívateľ sa opätovne prihlási, vyhľadá a upraví nejaký svoj recipe a vygeneruje dáta.

Obr. 6.2: Druhým scenárom je postupnosť prihlásenia sa do systému, nasledovaná zobrazením a upravením daného recipe, po čom sa opäť spustí generovanie a zobrazenie výsledku. Scenár popisuje prípady použitia: **UC.3**, **UC.7**, **UC.9** 2.1.

Pre ostatné prípady použitia boli vytvorené podobné scenáre manuálne, vyššie znázornenie slúži pre ilustráciu toho, ako typický scenár pri návrhu rozhrania vyzerá.

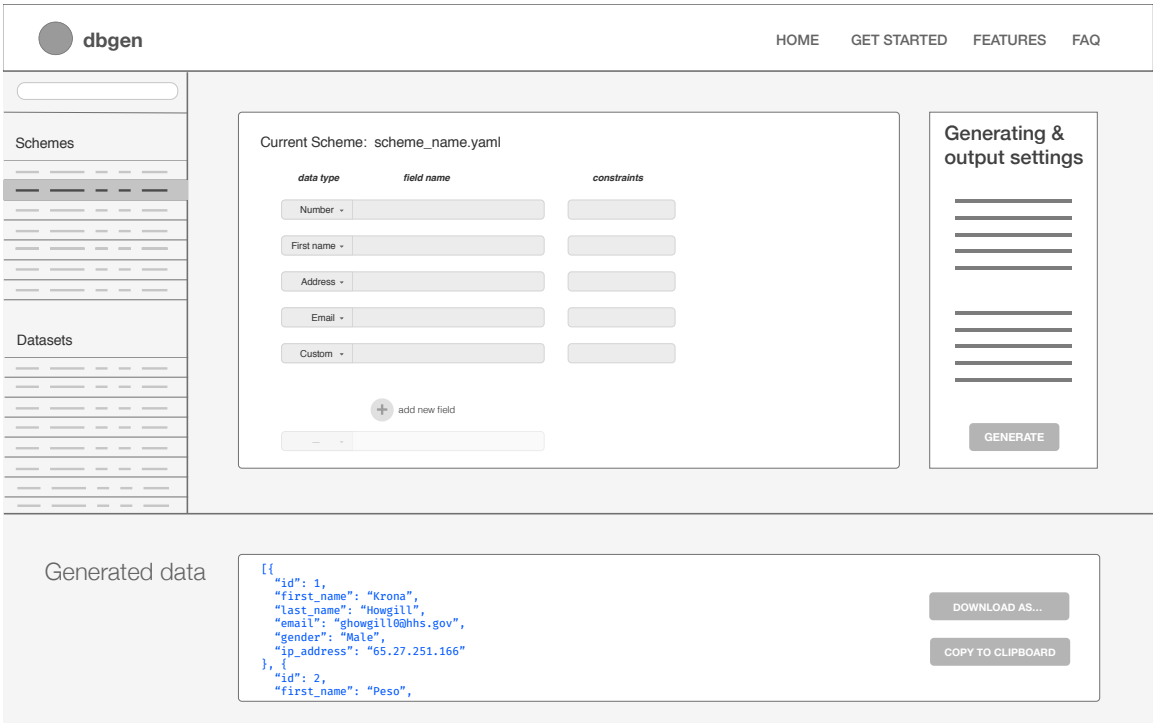
Po zvážení všetkých podstatných scenárov použitia a požiadavky **REQ.26**, ktorá vyžaduje aby bolo webové rozhranie funkčne ekvivalenté aplikačnému rozhraniu, bola stanovená štruktúra webového rozhrania, znázornená na obrázku 6.3, ktorá taktiež ilustruje, základný tok užívateľských aktivít vo webovom rozhraní.



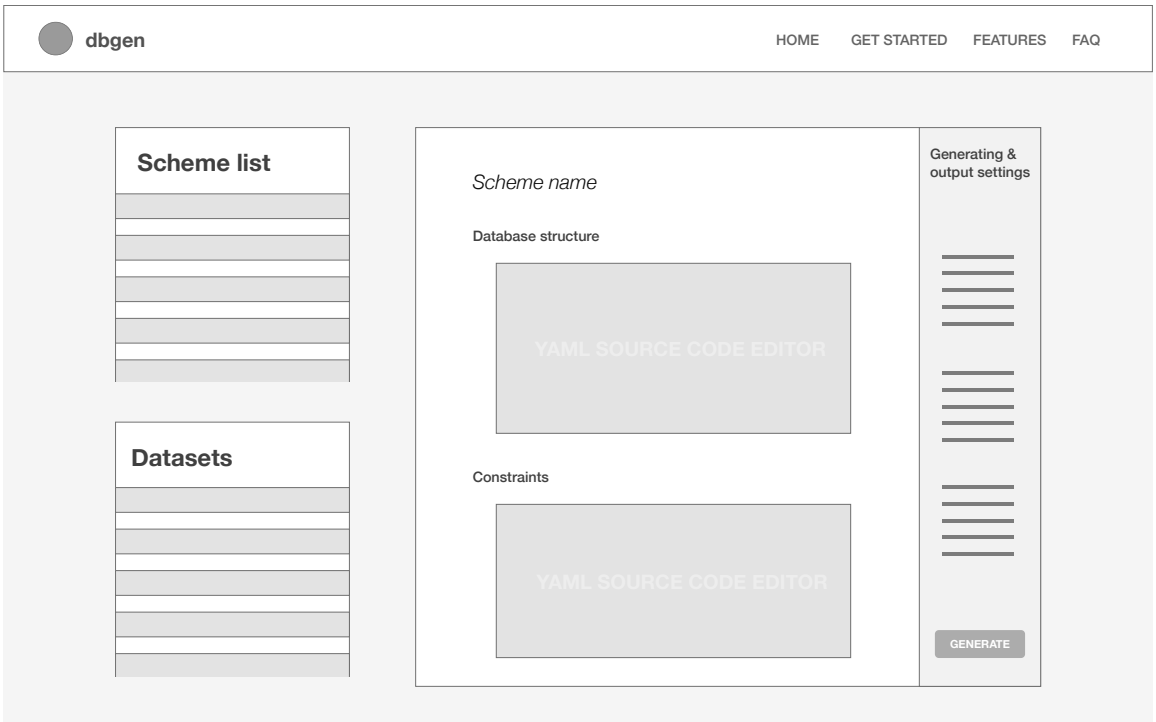
Obr. 6.3: Diagram rozloženia webového obsahu a rozhrania.

Návrhy

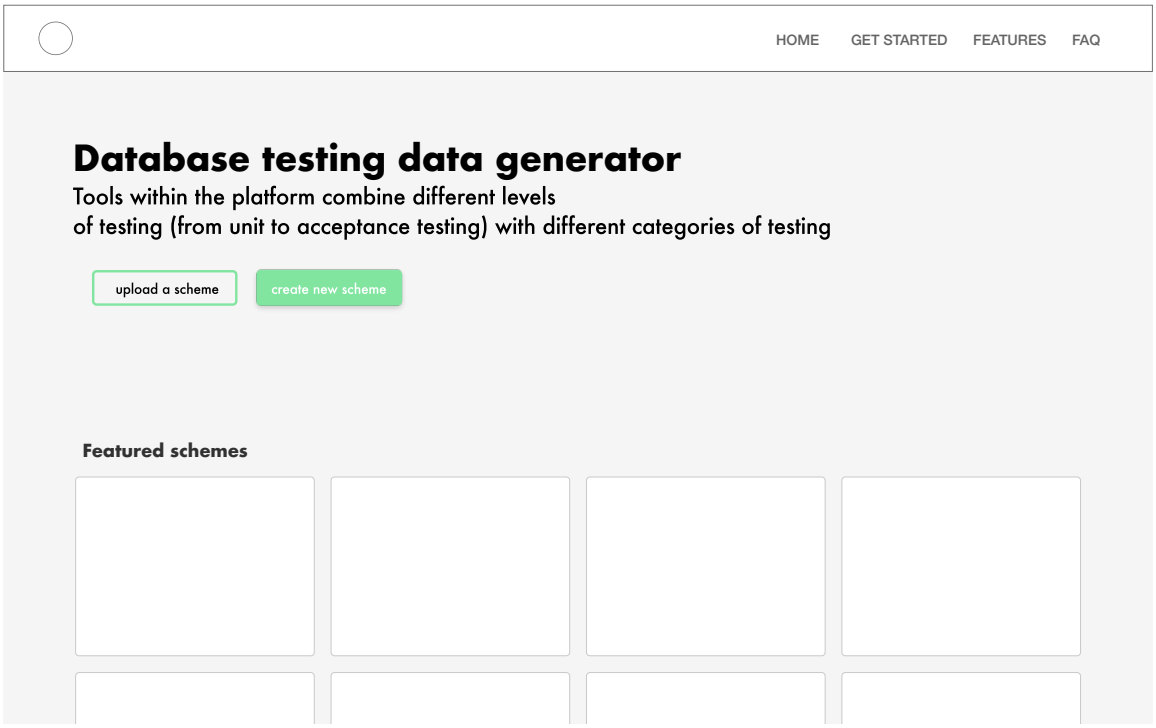
Špecifickej implementácii webového rozhrania predchádzalo vytvorenie prototypov rozhrania, akými sú napríkľad prototypy na obrázkoch 6.4, 6.5, 6.6 a 6.7. Finálna implementácia sa značne od týchto prototypov odlišuje, avšak počas práce tvorbe rozhrania poskytovali tieto prototypy nápomocný nadhľad o dôležitých častiach rozhrania.



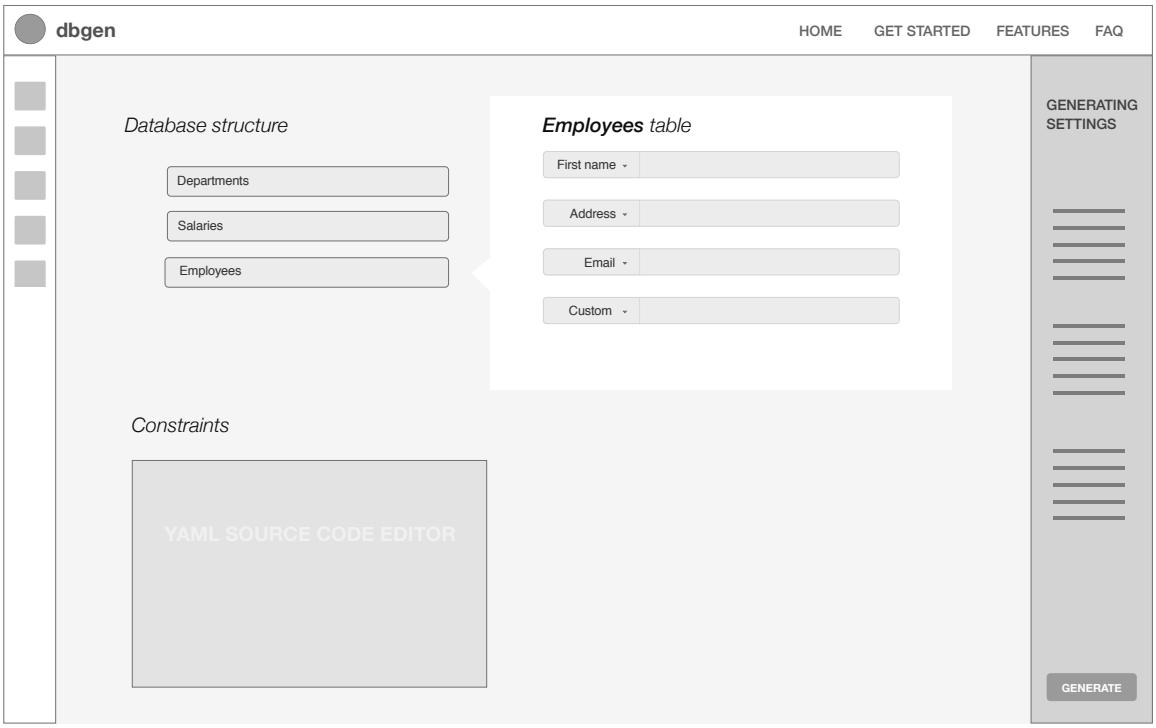
Obr. 6.4: Prototyp 1.



Obr. 6.5: Prototyp 2.



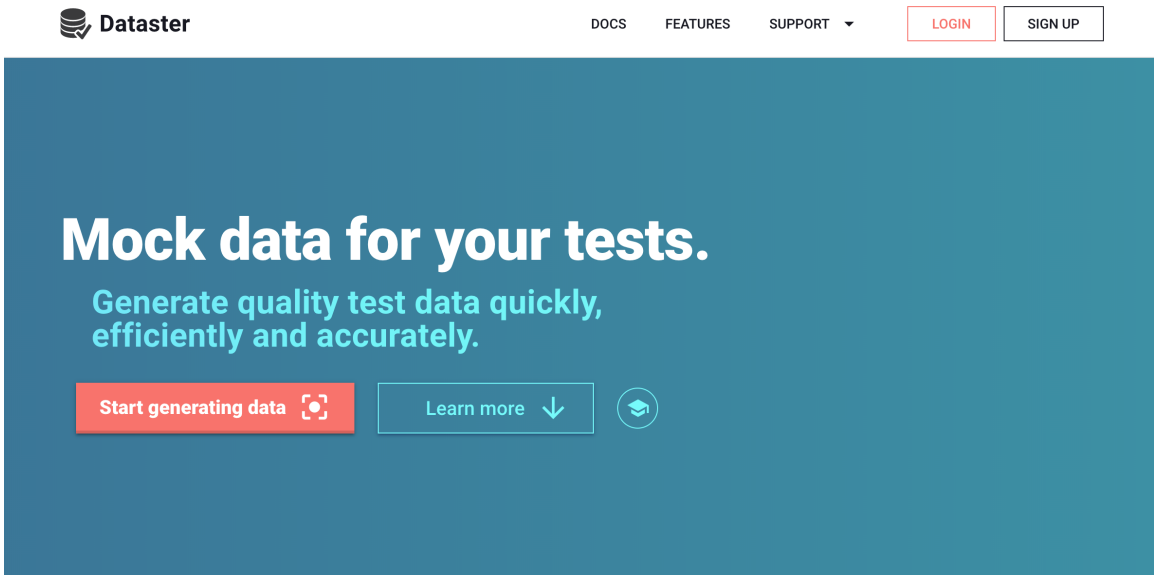
Obr. 6.6: Prototyp 3.



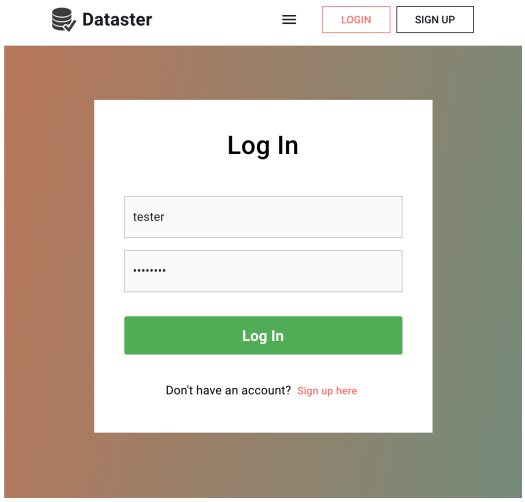
Obr. 6.7: Prototyp 4.

Detailné rozhrania jednotlivých podstránok

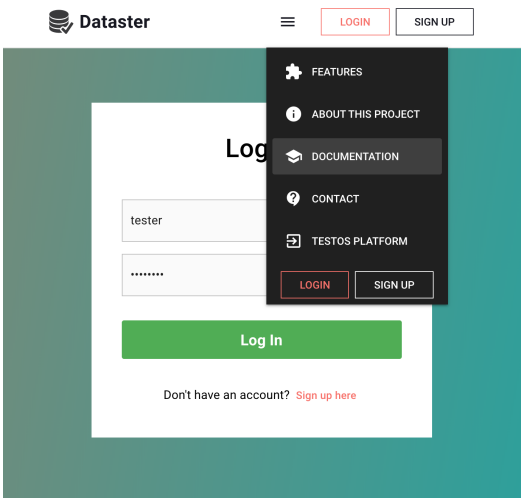
Pri návrhu a implementácií komponentov webového rozhrania bolo brané do úvahy, aby boli splnené požiadavky **REQ.28**, **REQ.29**, **REQ.30**, **REQ.31**. Ukážku výslednej podoby webového rozhrania je možné vidieť na obrázkoch 6.8, 6.11, 6.12 a 6.13. Návrh a vývoj rozhrania najviac ovplyvnila požiadavka **REQ.32**, ktorá vyžaduje responzívnosť rozhrania aj pre mobilné zariadenia. Väčšina komponentov bola tým pádom vyvíjaná s dôrazom na správne vykreslovanie pri dynamických zmenách rozlíšenia obrazovky. Niektoré komponenty, ako napríklad navigačný panel sa v prípade menších rozlíšení skrývajú a nahradzujú komponentami vhodnými pre malé rozmery obrazky ako je to naznačené na obrázkoch 6.9 a 6.10.



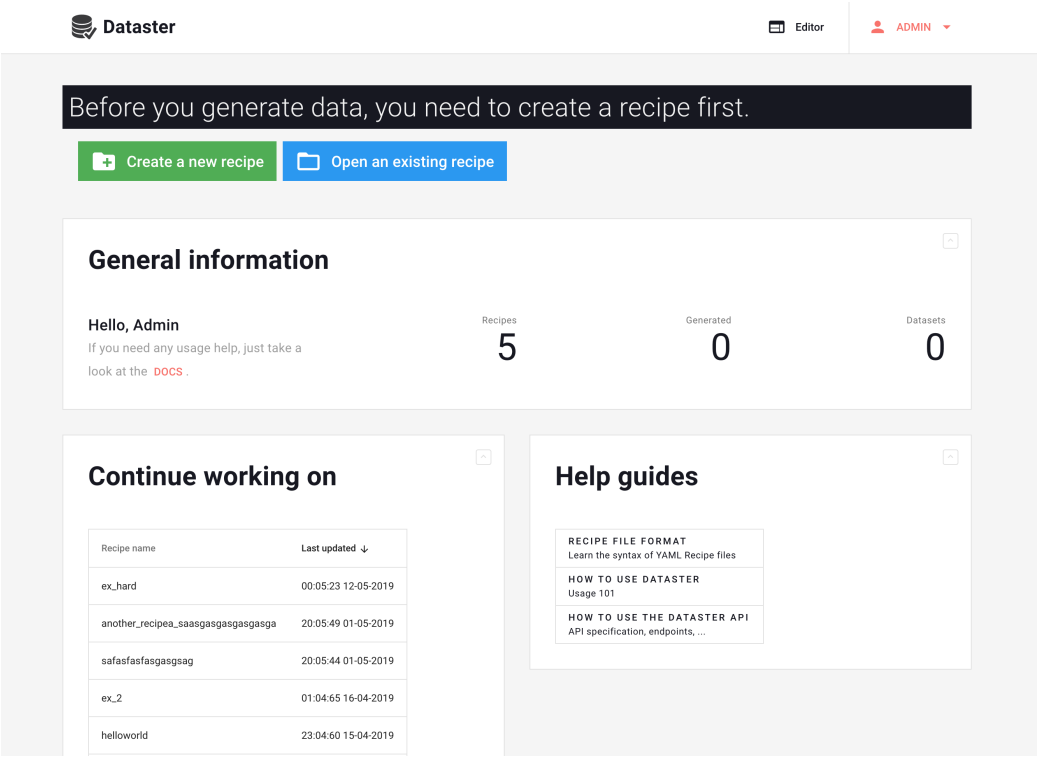
Obr. 6.8: Domovská stránka webového rozhrania Dataster.



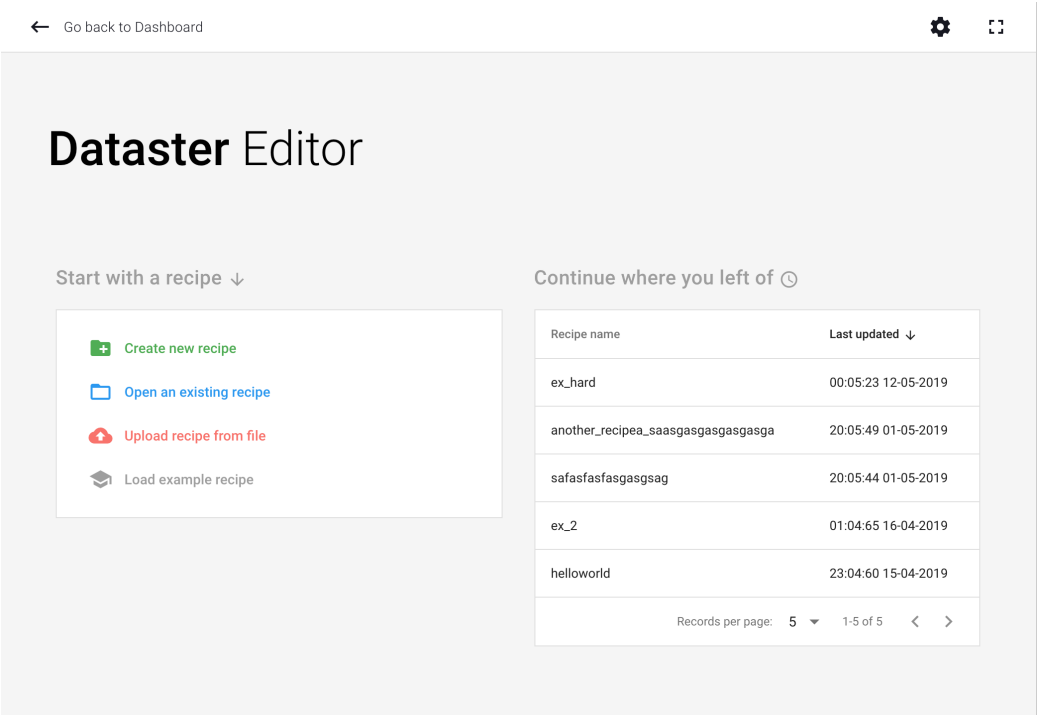
Obr. 6.9: Prihlasovacia stránka s formulárom.



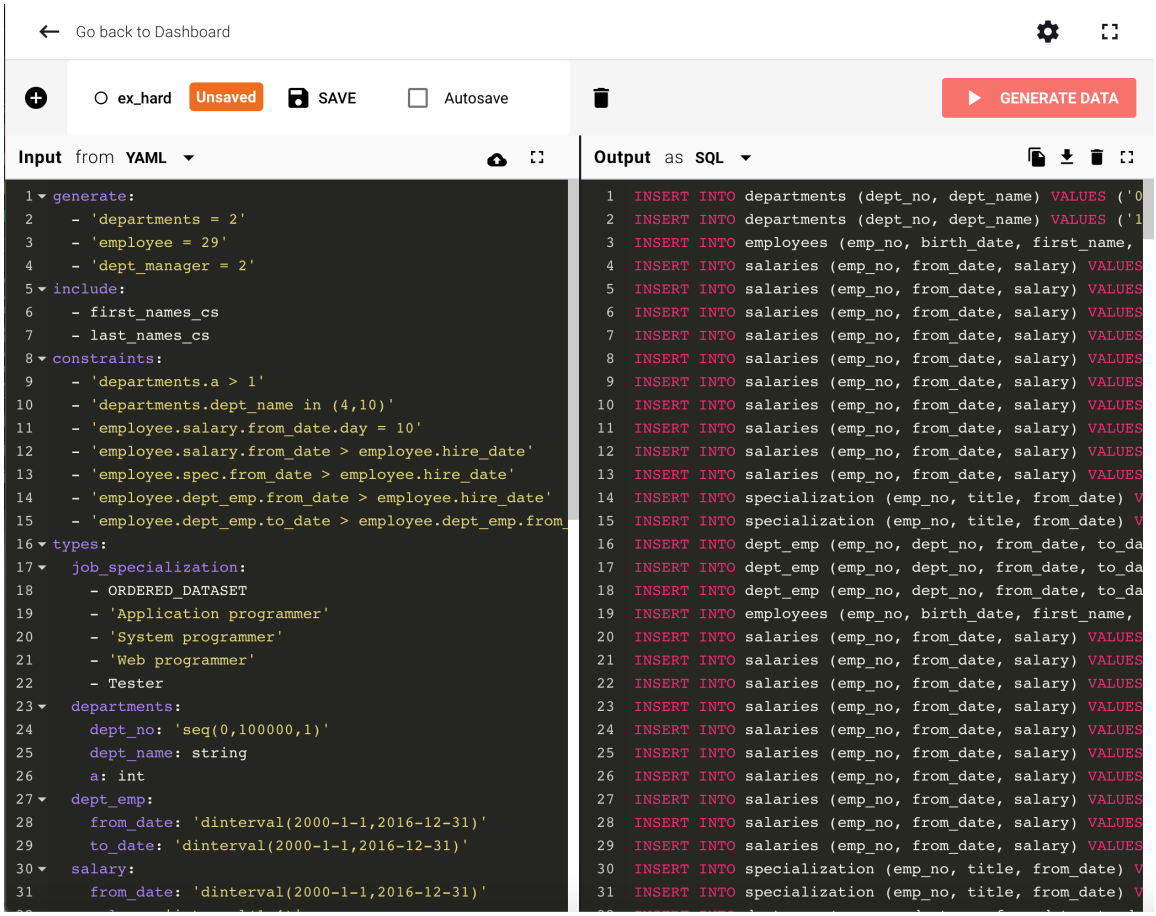
Obr. 6.10: Zobrazenie navigácie responzívne.



Obr. 6.11: Dashboard webového rozhrania Dataster, poskytujúci prehľad nad pracovnými dátami a umožňujúci vykonať rýchle akcie hneď po prihlásení.



Obr. 6.12: Uvítacia stránka editoru, zobrazená v prípade, že práve v pracovnom prostredí nie je načítaný žiadny recipe.



Obr. 6.13: Hlavná stránka editoru pre úpravu recipes a spúšťanie generovania je jedným z najkomplikovanejších komponentov rozhrania. Okrem upravovania kódu vo viacerých formátoch, poskytuje možnosť generované dáta exportovať a zvoliť zo štyroch formátov výstupu. Rozhranie editoru kódu je flexibilné a jeho veľkosť je možné meniť.

Kontext spustenia aplikácie

Framework **Quasar** umožňuje zvoliť viacero možností konfigurácie pre zostavenie a deployment aplikácie. Okrem jednoduchej SPA (Single Page Application) webovej aplikácie umožnuje napríklad nasadenie vytvorenej implementácie do kontextu mobilnej aplikácie na rôznych platformách.

V súčasnom stave je nastavené zostavenie len pre SPA konfiguráciu, avšak táto možnosť je ľahko meniteľná v konfigurácii služby web-frontend.

6.2 Osvedčené postupy a optimalizácie

Významnou kvalitou webového rozhrania je jeho rýchlosť načítania a responzívnosť na akcie užívateľa. Aby sa predišlo dlhým intervalom pri komunikácii so serverom, prebieha načítanie prvkov rozhrania len raz, po čom je všetka komunikácia so serverom vykonávaná asynchronne. Neprebieha teda nikdy prekresľovanie ako dôsledok opätovného odoslania žiadosti na server.

Ďalším spôsobom ako zvýšiť rýchlosť načítavania webového rozhrania už pri prvom stiahnutí jeho zdrojov je napríklad kompresia statických súborov.

Kompresia HTTP odpovedí

Kompresia odpovedí protokolu HTTP/HTTPS je jednou z možností ako zmenšiť veľkosť prenášaných dát medzi klientom (webovým prehliadačom) a serverom (webovou aplikáciou). V súčasnej dobe využíva kompresiu viac než 77.7% webových stránok [8]. Pre kontext Datasteru, môže kompresia statických súborov prebiehať na viacerých miestach, v závislosti od konfigurácie. Predvolene prebieha kompresia v službe **web-frontend**, v rámci servéru Nginx. Ten využíva na kompresiu metódu GZIP.

Pre prípadný iný kontext nasadenia aplikácie je však možné aplikovať kompresiu na statické súbory frontendu vopred. [4]

Minifikovanie kódu

Ďalším spôsobom na zmenšenie veľkosti súborov klientskej časti webovej aplikácie je minifikovanie zdrojového kódu webovej stránky. Tento proces odstráni všetky syntakticky nepotrebné prázdne znaky, medzery a zalomovacie znaky pre nový riadok.

Výsledkom je menší, no pre človeka nečitateľný súbor (napr. CSS/HTML/JS). Na druhej strane pre webové prehliadače je kód stále čitateľný a prináša úsporu objemu prenášaných dát pri HTTP komunikácii.

Špecifikácia ciest API rozhrania

Aplikácia poskytuje identické možnosti na používanie prostredníctvom API rozhrania. Pre správu užívateľských dát je možné využívať nasledujúce cesty pomocou HTTP klienta. V nasledujúcich ukážkach je použitý nástroj CURL[1]. Pre kompletnú špecifikáciu API rozhrania je poskytnutý zoznam v prílohe B.

Pri tvorbe webového rozhrania bolo zohľadnené, aby sa splnila požiadavka **REQ.27**, ktorá vyžaduje, aby sa pri získavaní dát z API webové rozhranie zbytočne dlho neprekreslovalo alebo nezastavovalo vykreslovanie. Riešením je striktné používanie asynchronnej komunikácie so serverom a jeho API rozhraním. Webové rozhranie teda nikdy synchronne nečaká na odpoveď a webové rozhranie zostáva stále vykreslené.

`POST /users/`

```
curl -X POST http://localhost:3000/api/users/ \
       -H 'Content-Type: application/json' \
       -H 'cache-control: no-cache' \
       -d '{
              "email": "tester@testos.com",
              "username": "tester",
              "password": "password"
              }'
```

[1] https://curl.haxx.se/

Kapitola 7

Testovanie, integrácia a nasadenie do produkcie

Pomyselným cieľom každého softvérového projektu je finálne nasadenie systému alebo softvéru do praxe (produkčného prostredia). Jednoducho povedané, ide o stav, v ktorom sa systém alebo softvér ocitne, keď bude dostupný na použitie užívateľmi. Proces dosiahnutia tohto cieľa sa nazýva **deployment** (nasadenie) a začína po ukončení procesu vývoja [20]. Pred tým, než začne, ale taktiež aj počas procesu deploymentu, by malo prebehnúť niekoľko dôležitých krokov, ktorými sa bude zaoberať táto kapitola.

7.1 Testovanie

Testovaním sa rozumie akákoľvek aktivita s cieľom ohodnotenia vlastností alebo schopností určitého programu alebo systému a rozhodnutie, či spĺňa očakávané výsledky [15].

Procesom testovania je možné systematicky rozhodnúť, či výsledky vývoja vyhovujú stanoveným požiadavkám a či systém alebo vyvíjaný softvér spĺňa jeho očakávané správanie, schopnosti použitia a potreby užívateľa [2].

Testovací plán a metodológia

Testovacím plánom sa rozumie plán projektu pripravovanej testovacej aktivity. Slúži najmä ako pomyselný návod pre chápanie významu a pripomínanie dôležitých oblastí, ktorých by sa testy mali týkať. Neobsahuje detailnú špecifikáciu návrhu testov ani kolekciu konkrétnych testovacích prípadov, testovacích sád alebo iných implementačných detailov [1].

Príkladom testovacieho plánu môže byť testovací plán podľa štandardu *IEEE 829*, ktorý by mal obsahovať informácie ako:

- jednoznačný identifikátor plánu,
- rámec testovania; celkový kontext a podmienky testov,
- vymenovanie testovaných vlastností systému,
- vymenovanie vlastností systému, ktoré netreba testovať,
- testovacie kritéria a
- špecifikáciu požiadaviek ... [2].

Pre kontext projektu však bude testovací plán len vyššie spomenutým štandardom inšpirovaný. Hlavnou kostrou preň bude kombinácia formálnej špecifikácie požiadaviek, testy reagujúce na jednotlivé požiadavky a ich následné vyhodnotenie podľa úspešnosti.

Metodológia testovania sa bude najviac opierať o **Requirements-Based Testing** (Testovanie na základe požiadaviek) – ďalej ako **RBT**, ktoré kladie dôraz na kvalitnú špecifikáciu požiadaviek a následne návrh testovacích sád pokrývajúcich dané požiadavky [22]. Kvantifikovanou mierou kvality výsledného systému bude teda percentuálny pomer pokrytia špecifikácie požiadaviek.

Typy testov

Vzhľadom na to, že systém je zložený z niekoľkých menších častí, správne testovanie by malo overiť správanie nielen celého systému ako celku, ale aj jednotlivých komponentov a ich rozhraní. Práve preto bude kvalita systému podporená testami na nasledujúcich úrovniach:

- **Systémové testy** – overujúce správne fungovanie celého systému ako celku.
- **Integračné testy** – overujúce správne fungovanie rozhraní komponentov a ich vzájomnej komunikácie.
- **Jednotkové testy** – overujúce správne fungovanie menších fragmentov kódu v rámci komponentov systému.

Systémové testy

Testovanie funkcionality celého systému je možné pomocou dodatočných kontajnerových služieb, využívajúcich najmä nástroj Selenium[1]. Pri testovaní sa používa kontajnerová architektúra znázornená na obrázku 7.1.

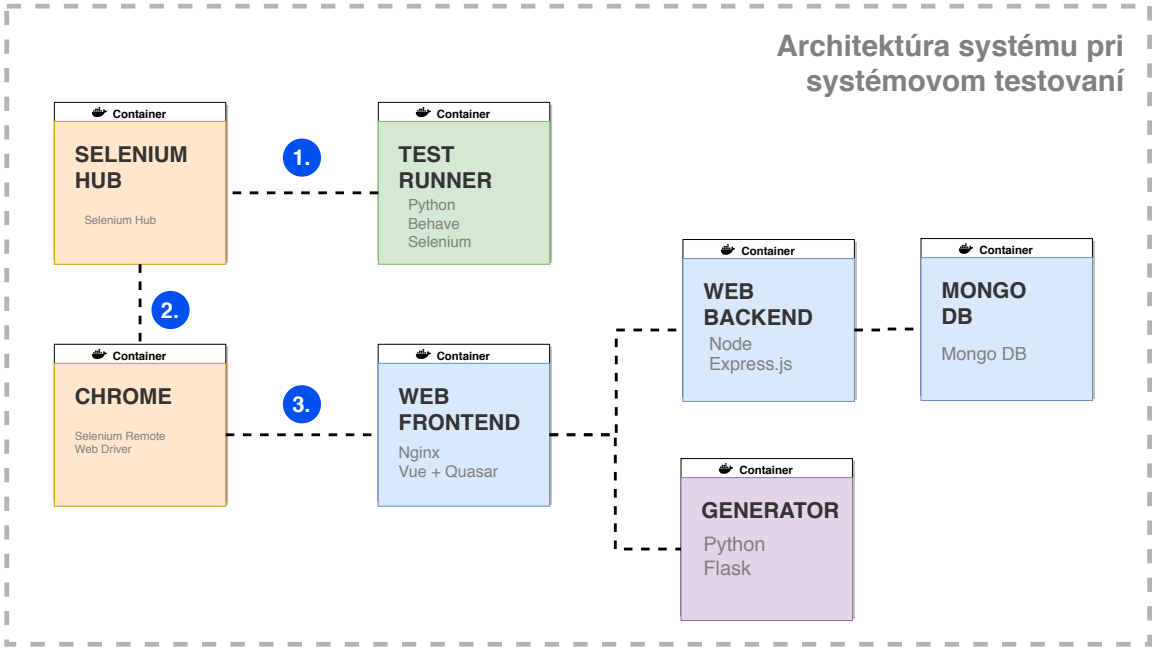
Integračné testy

Vrámci projektu prebiehajú integračné testy pre službu **web-backend**. Testuja sa v nich rozhranie služby a jej vnútorná komunikácia s priľahlou službou **mongodb**. Na spúšťanie testov sa využíva nástroj Jest[2] a definície jednotlivých testovacích prípadov sa nachádzajú v adresároch pre submoduly služby, ktorých sa týkajú. Hlavnou náplňou testov je automatizovaným spôsobom otestovať API rozhranie služby a popritom sledovať správanie vnútornej implementácie služby.

Jednotkové testy

Pre službu **web-backend** je možné väčšinu kódu pokryť v rámci integračného testovania, no pri testovaní pomocných funkcií, napríklad modulu `utils`, v ktorom sa nachádzajú funkcie pre návrat normalizovaných odpovedí od servera, sa používajú aj jednotkové testy. Snahou týchto testov je overiť správne fungovanie fragmentov kódu a všetkých riadiacich vetiev funkcií modulu.

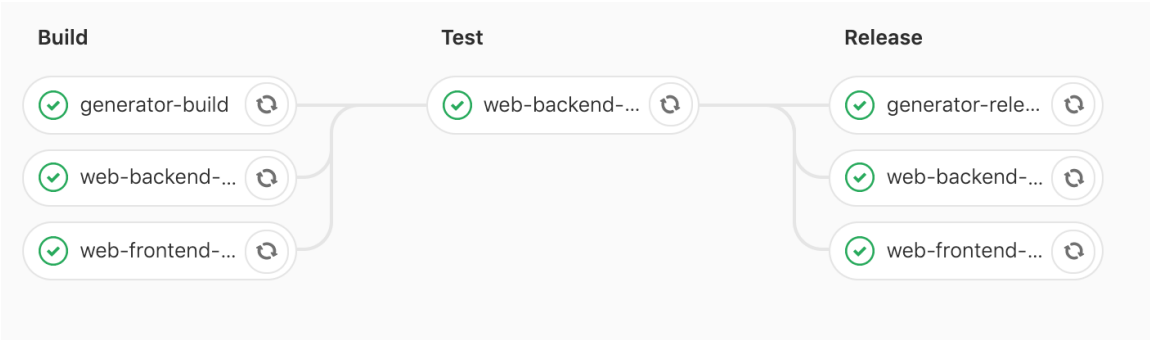
[1] https://www.seleniumhq.org/
[2] https://jestjs.io/



Obr. 7.1: Kontajnerová architektúra pri systémovom testovaní.

Automatizované testy pri CI/CD

Projekt aktívne využíva možnosti vývojového repozitára služby GitLab[3] a jej možností priebežnej integrácie. Vždy po zavedení zmien do kódu repozitára prebehne niekoľko integračných úloh, ktoré najprv overia, či je jednotlivé služby systému možné zostaviť, následne vykonajú automatizované testy pomocou testovacej konfigurácie kontajnerového systému a v prípade, že všetko prebehne úspešne, označia Docker obrazy pre jednotlivé služby značkou `release`. Bližšiu predstavu o etapách priebežnej integrácie môže poskytnúť obrázok 7.2.



Obr. 7.2: Zreťazené spracovanie pri priebežnej integrácii projektu na službe GitLab. Konfigurácia pre priebeh integrácie je definovaná v súbore `.gitlab.ci.yml`.

[3] https://gitlab.com

Manuálne testy

Väčšina aktivít spojených s testovaním prebehla taktiež aj manuálnym testovaním najmä pomocou nástroja **Postman**[4] alebo manuálnym využívaním webového rozhrania aplikácie. HTTP klient nástroja Postman poskytol prehľad nad správaním rozhrania API a po manuálnej kontrole návratových hodnôt všetkých ciest API, sa dá skonštatovať že boli splnené všetky požiadavky týkajúce sa API rozhrania (**REQ.10 - REQ.25**).

7.2 Nasadenie a konfigurácia spustenia

Projekt je možné spustiť v niekoľkých konfiguráciách spustenia v závislosti od kontextu a prostredia. Prostredie spustenia musí spĺňať určité predpoklady, ktoré sú popísané v sekcií 7.2. Ak bude projekt spúšťaný v produkčnej konfigurácii, je potrebné správne nastaviť premenné prostredia. Vývojová konfigurácia má premenné prostredia nastavené predvolené a nie je potrebné ich meniť, ak to nie je nutné.

Predpoklady pre spustenie

Základnými predpokladanými závislosťami, ktoré je potrebné mať na hostiteľskom systéme sú:

- **Git**[5] – pre získanie všetkých projektových súborov priamo z repozitára,
- **Docker**[6] a **Docker Compose**[7] – pre možnosti kontajnerizácie.

Vývojová konfigurácia

Pre účely vývoja je možné projekt spustiť vo vývojovej konfigurácii, pri ktorej je možné do zdrojového kódu priamo zasahovať, pridávať alebo upravovať moduly a premenné prostredia. Všetky služby sa automaticky reštartujú a aktualizujú zmenené dáta.

Spustenie vývojovej konfigurácie: `$ make dev-build && make dev`.

Po zostavení a spustení kontajnerov je možné overiť dostupnosť služieb systému.

Zobrazenie spustených služieb: `$ make ps`.

Pri spúšťaní systému vo vývojovej konfigurácii je taktiež k dispozícii skript `tmuxdev.sh`, ktorý využíva nástroj **Tmux**[8], umožňujúci súčasné zobrazenie viacerých terminálových okien. Poskytuje tak nadhľad nad výpismi o aktivite služieb počas vývoja, ako je možné vidieť na obrázku 7.3.

Spustenie vývojového zobrazenia: `$ ./tmuxdev.sh`.

[4] https://www.getpostman.com/
[5] https://git-scm.com/
[6] https://www.docker.com/
[7] https://docs.docker.com/compose/
[8] https://github.com/tmux/tmux

```
dataster-dev:1:mak... ⌘1
web-frontend    |
 DONE  Compiled successfully in 44719ms          6:38:35 PM
web-frontend    |
web-frontend    |
web-frontend    |  N  App dir........... /app
web-frontend    |     App URL........... http://localhost:8888/
web-frontend    |     Dev mode.......... spa
web-frontend    |     Pkg quasar........ v1.0.0-beta.19
web-frontend    |     Pkg @quasar/app... v1.0.0-beta.20
web-frontend    |

generator       | server-0  |  * Environment: production
generator       | server-0  |    WARNING: Do not use the development server in a production environmen
t.
generator       | server-0  |    Use a production WSGI server instead.
generator       | server-0  |  * Debug mode: off
generator       | server-0  |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
generator       | ==============================================================================
generator       | --- PM2 development mode -------------------------------------------------------
generator       | Apps started         : server
generator       | Processes started    : 1
generator       | Watch and Restart    : Enabled
generator       | Ignored folder       : node_modules
generator       | ==============================================================================
generator       | server-0  |  * Serving Flask app "api" (lazy loading)
generator       | server-0  |  * Environment: production
generator       | server-0  |    WARNING: Do not use the development server in a production environmen
t.
generator       | server-0  |    Use a production WSGI server instead.
generator       | server-0  |  * Debug mode: off
generator       | server-0  |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

web-backend     | --- PM2 development mode -------------------------------------------------------
web-backend     | Apps started         : server
web-backend     | Processes started    : 1
web-backend     | Watch and Restart    : Enabled
web-backend     | Ignored folder       : node_modules
web-backend     | ==============================================================================
web-backend     | server-0  | |-----------------------------------------------------------------------------
web-backend     | server-0  | | UP!
web-backend     | server-0  | |  Listening on:  http://0.0.0.0:3000/
web-backend     | server-0  | |-----------------------------------------------------------------------------
web-backend     | server-0  |
web-backend     | server-0  | |-----------------------------------------------------------------------------
web-backend     | server-0  | | [dataster]  ✅  Connection to database successful!
web-backend     | server-0  | |  Backend  URI: mongodb://mongo:27017/DatasterDevelopment
web-backend     | server-0  | |-----------------------------------------------------------------------------
web-backend     | server-0  | GET /api/users 200 95.878 ms - 234
web-backend     | server-0  | GET /api/recipes 200 31.571 ms - 8109
web-backend     | server-0  | GET /api/users 200 6.315 ms - 234
web-backend     | server-0  | GET /api/datasets 204 7.582 ms - -
web-backend     | server-0  | GET /api/recipes 200 8.662 ms - 8109
web-backend     | server-0  | GET /api/recipes 200 9.142 ms - 8109
web-backend     | server-0  | GET /api/users 200 9.593 ms - 234

1:make*                                                                          14/05  23:40:02
```

Obr. 7.3: Súčasné zobrazenie výpisov zo služieb systému vo vývojovej konfigurácii nástrojom tmux.

Produkčná konfigurácia spustenia

Produkčná konfigurácia nastavuje systém tak, aby ho bolo možné nasadiť do produkčného prostredia a ďalej sa o jeho beh nestarať. Vytvára bezpečnostnú izoláciu pre väčšinu služieb systému a pre vonkajšie prostredie poskytuje prístupový bod len na jedine službe – **web-frontend**. Pre všetky služby zabezpečuje stály beh, aj v prípade, že v nich nastane interná chyba.

Spustenie produkčnej konfigurácie: `$ make production-build && make production`.

Kapitola 8

Záver

Vytváranie vzorových dát pre testovacie účely je náročná úloha. Automatizácia tohto procesu je významnou nápomocou pri testovaní softvéru, najmä ak je možné výsledné dáta získať rýchlo a efektívne. Webová aplikácia Dataster, ktorú popisuje táto práca, poskytuje možnosť generovania dát prostredníctvom ako webového tak aj aplikačného rozhrania. V aplikácii je možné vytvoriť osobný užívateľský účet, ku ktorému je možné priradiť a uložiť pracovné dáta pri práci s generátorom a editorom konfiguračných súborov. Dáta užívateľov zostanú perzistentne uložené a pripravené na ďalšie použitie v budúcnosti. Nástroj umožňuje výstupné dáta generovania exportovať do viacerých formátov, špecifických pre zobrazenie tabuľkových dát.

Webový systém bol navrhnutý tak, aby bol v prípade potreby v budúcnosti ľahko rozšíriteľný a jeho primárne komponenty ľahko vymeniteľné. Aplikácia je schopná okamžitého nasadenia v produkcii, no taktiež jednoduchého spustenia vo vývojovej konfigurácii, v ktorej je možné priamo zasahovať do zdrojového kódu všetkých služieb systému a ich zmeny vidieť reflektované v celom systéme za behu.

Projekt tak predstavuje nielen výslednú webovú aplikáciu ale aj komplexné vývojové prostredie so širokými možnosťami rozvoja aplikácie v budúcnosti. Požiadavky projektu boli splnené a overené kombináciou manuálneho a automatizovaného testovania.

Webová aplikácia Dataster bola na záver úspešne nasadená do produkčného prostredia na adrese `https://dataster.testos.org`, kde ju je možné aktívne a voľne využívať.

Literatúra

[1] What is the purpose and importance of test plans in software testing? [Navštívené 13.05.2019].
URL http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/

[2] IEEE Standard for Software and System Test Documentation. *IEEE Std 829-2008*, July 2008: s. 1–150, doi:10.1109/IEEESTD.2008.4578383.

[3] Server-side vs Client-side Routing. May 29 2017, navštívené 14.05.2019.
URL https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f

[4] Can gzip Compression Really Improve Web Performance? 2018.
URL https://royal.pingdom.com/can-gzip-compression-really-improve-web-performance/

[5] Generate Data. 2018.
URL https://www.generatedata.com/

[6] Mockaroo - Random Data Generator. 2018.
URL https://mockaroo.com/

[7] Docker - Enterprise Container Platform for High-Velocity Innovation. 2019.
URL https://www.docker.com/

[8] GZIP Compression Survey. 2019.
URL https://w3techs.com/technologies/details/ce-gzipcompression/all/all

[9] JavaScript. May 2019.
URL https://developer.mozilla.org/en-US/docs/Web/JavaScript

[10] Official NGINX Homepage. 2019.
URL https://nginx.org/

[11] Vue JS Documentation guide. 2019.
URL https://vuejs.org/v2/guide/

[12] Agrawal, S.; Verma, J.; MAHIDHARIYA, B.; aj.: SURVEY ON MONGODB: AN OPEN-SOURCE DOCUMENT DATABASE. *International Journal of Advanced Research in Engineering and Technology (IJARET)*, ročník 6, 12 2015: str. 1 to 11.

[13] Blake, S.; Black, D. L.; Carlson, M. A.; aj.: An Architecture for Differentiated Services. RFC 2475, RFC Editor, December 1998,

http://www.rfc-editor.org/rfc/rfc2475.txt.
URL http://www.rfc-editor.org/rfc/rfc2475.txt

[14] Fredrich, T.: Learn REST: A RESTful Tutorial. 2018.
URL https://www.restapitutorial.com/

[15] Hetzel, W.: *The Complete Guide to Software Testing*. QED Information Sciences, druhé vydanie, Január 1988, ISBN 0-89435-242-3.

[16] Jones, M.; Bradley, J.; Sakimura, N.: JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015, http://www.rfc-editor.org/rfc/rfc7519.txt.
URL http://www.rfc-editor.org/rfc/rfc7519.txt

[17] Kotyz, J.: Testos / DB data generator extended. 2018.
URL https://pajda.fit.vutbr.cz/testos/dbgenx

[18] Lewis, J.; Fowler, M.: Microservices. 2014.
URL https://martinfowler.com/articles/microservices.html

[19] Medlock, J.: 12 2018.
URL https://medium.com/chingu/an-introduction-to-environment-variables-and-how-to-use-them-f602f66d15fa

[20] Pressman, R. S.: *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, piate vydanie, 2001, ISBN 0072496681.

[21] Rawdat, A.: Rate Limiting with NGINX and NGINX Plus. June 2017, navštívené 14.05.2019.
URL https://www.nginx.com/blog/rate-limiting-nginx/

[22] Skokovic, P.; Rakic-Skokovic, M.: Requirements-based testing process in practice. *International Journal of Industrial Engineering and Management*, ročník 1, 01 2010: s. 155–161.

[23] Smrcka, A.: Testos Platform. 2018.
URL http://testos.org

[24] Tractinsky, N.; Katz, A.; Ikar, D.: What is beautiful is usable. *Interacting with Computers*, ročník 13, č. 2, 2000: s. 127 – 145, ISSN 0953-5438, doi:https://doi.org/10.1016/S0953-5438(00)00031-X.
URL http://www.sciencedirect.com/science/article/pii/S095354380000031X

[25] Wan, X.; Guan, X.; Wang, T.; aj.: Application deployment using Microservice and Docker containers: Framework and optimization. *Journal of Network and Computer Applications*, ročník 119, 2018: s. 97 – 109, ISSN 1084-8045, doi:https://doi.org/10.1016/j.jnca.2018.07.003.
URL http://www.sciencedirect.com/science/article/pii/S1084804518302273

Príloha A

Konfiguračný súbor – Recipe

```
include:
    - first_names_cs
    - last_names_cs
types:
    job_specialization: [
        "ORDERED_DATASET",
        "Application programmer",
        "System programmer",
        "Web programmer",
        "Tester"
    ]

    departments:
        dept_no: seq(0,100000,1)
        dept_name: string
        a: int

    dept_emp:
        from_date: dinterval(2000-1-1,2016-12-31)
        to_date: dinterval(2000-1-1,2016-12-31)

    salary:
        from_date: dinterval(2000-1-1,2016-12-31)
        salary: interval(1,4)

    specialization:
        title: job_specialization
        from_date: dinterval(2000-1-1,2016-12-31)

    employee:
        emp_no: seq(0,100000,1)
        birth_date: dinterval(1950-1-1,1997-12-31)
        first_name: first_names_cs
        last_name: last_names_cs
        email:
            format: "{}.{}@company.com"
            values:
                - employee.first_name
                - employee.last_name
        hire_date: dinterval(2000-1-1,2016-12-31)
        spec: "[specialization]{1,2}"
        salary: "[salary]{0,10}"
```

```
        dept_emp: "[dept_emp]{1,3}"

    dept_manager:
        active: bool

schema:
    employees:
        emp_no: employee.emp_no
        birth_date: employee.birth_date
        first_name: employee.first_name
        last_name: employee.last_name
        email: employee.email
        hire_date: employee.hire_date

    salaries:
        emp_no: employee.emp_no
        from_date: employee.salary.from_date
        salary: employee.salary.salary

    specialization:
        emp_no: employee.emp_no
        title: employee.spec.title
        from_date: employee.spec.from_date

    departments:
        dept_no: departments.dept_no
        dept_name: departments.dept_name

    dept_emp:
        emp_no: employee.emp_no
        dept_no: FROM departments.dept_no
        from_date: employee.dept_emp.from_date
        to_date: employee.dept_emp.to_date

    dept_manager:
        emp_no: FROM dept_emp.emp_no
        dept_no: FROM dept_emp.dept_no
        from_date: FROM dept_emp.from_date
        to_date: FROM dept_emp.to_date
        active: dept_manager.active

constraints:
    - departments.a > 1
    - departments.dept_name in (4,10)
    - employee.salary.from_date.day = 10
    - employee.salary.from_date > employee.hire_date
    - employee.spec.from_date > employee.hire_date
    - employee.dept_emp.from_date > employee.hire_date
    - employee.dept_emp.to_date > employee.dept_emp.from_date

generate: ["departments = 2", "employee = 10", "dept_manager = 2"]
```

Príloha B

API špecifikácia

V tejto prílohe sú popísané všetky API cesty dostupné cez HTTP protokol.

Užívateľia

`POST /users/` – Vytvorenie nového osobného účtu (UC.2)

```
curl -X POST http://localhost:3000/api/users/ \
       -H 'Content-Type: application/json' \
       -H 'cache-control: no-cache' \
       -d '{
              "email": "tester@testos.com",
              "username": "tester",
              "password": "password"
              }'
```

`GET /users/` – Získanie dát osobného účtu (UC.4)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X GET http://localhost:3000/api/users/ \
   -H 'Content-Type: application/json' \
   -H 'Authorization: Bearer eyJhbG...' \
   -H 'cache-control: no-cache'
```

`PUT /users/` – Úprava dát osobného účtu (UC.5)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X PUT http://localhost:3000/api/users/ \
   -H 'Content-Type: application/json' \
   -H 'Authorization: Bearer eyJhbG...' \
   -H 'cache-control: no-cache' \
   -d '{
       "email": "newemail",
       "username": "newusername",
   }'
```

`DELETE /users/` – Zmazanie osobného účtu (UC.6)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X DELETE http://localhost:3000/api/users/ \
   -H 'Content-Type: application/json' \
```

```
    -H 'Authorization: Bearer eyJhbG...' \
    -H 'cache-control: no-cache'
```

POST /users/login – Prihlásenie do systému (UC.3)

```
curl -X POST http://localhost:3000/api/users/login \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -d '{
        "username": "tester123",
        "password": "secretPassword",
    }'
```

Recipes

Nasledujúce cesty umožňujú prihlásenému užívateľovi spravovať jeho recipes.

POST /recipes/ – Vytvorenie nového recipe (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X POST http://localhost:3000/api/recipes \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...' \
    -d '{
        "recipename": "myrecipe23",
        "description": "sample description",
        "configuration": {}
    }'
```

GET /recipes/ – Zobrazenie všetkých recipes (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X POST http://localhost:3000/api/recipes \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```

GET /recipes/:recipe_id – Zobrazenie konkrétneho recipe (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X GET http://localhost:3000/api/recipes/:recipe_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```

PUT /recipes/:recipe_id – Upravenie konkrétneho recipe (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X PUT http://localhost:3000/api/recipes/:recipe_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...' \
    -d '{
        "recipename": "updated_recipe_name",
```

```
        "description": "new description",
        "configuration": {}
    }'
```

DELETE /recipes/:recipe_id – Zmazanie konkrétneho recipe (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X DELETE http://localhost:3000/api/recipes/:recipe_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```

Datasets

Nasledujúce cesty umožňujú prihlásenému užívateľovi spravovať jeho datasety.

POST /datasets/ – Vytvorenie nového datasetu (UC.8)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X POST http://localhost:3000/api/datasets \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...' \
    -d '{
        "datasetname": "mydataset",
        "description": "sample dataset description",
        "location": "first_cz_names.yaml"
    }'
```

GET /datasets/ – Zobrazenie všetkých datasetov (UC.8)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X POST http://localhost:3000/api/datasets \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```

GET /datasets/:dataset_id – Zobrazenie konkrétneho datasetu (UC.8)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X GET http://localhost:3000/api/datasets/:dataset_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```

PUT /datasets/:dataset_id – Upravenie konkrétneho datasetu (UC.8)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X PUT http://localhost:3000/api/datasets/:dataset_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...' \
    -d '{
        "datasetname": "updated_dataset_name",
```

```
        "description": "new description",
        "location": "first_names.yaml"
    }'
```

`DELETE /datasets/:dataset_id` – Zmazanie konkrétneh datasetu (UC.7)

Vyžaduje použitie autorizačného JWT tokenu.

```
curl -X DELETE http://localhost:3000/api/datasets/:dataset_id \
    -H 'Content-Type: application/json' \
    -H 'cache-control: no-cache' \
    -H 'Authorization: Bearer eyJhbG...'
```