Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Šablony zdrojového kódu

Plzeň 2021 Karel Engl

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Karel ENGL**
Osobní číslo:	**A16B0027P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Šablony zdrojového kódu**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou generování fragmentů kódu se záměrem zjednodušit rutinní činnosti a činnosti administrativního charakteru. Dále se seznamte s pravidly výroby programového kódu zadavatele.
2. Analyzujte, které části jsou vhodné pro automatické generování fragmentů kódu.
3. Najděte existující nástroje pro generování programového kódu a zjistěte jejich funkce a možnosti. Identifikujte funkce, které mohou být dle výsledků předchozí analýzy užitečné.
4. Implementujte generování kódu ve vybraných doménách. Kód generujte včetně fragmentů automatických testů.
5. Řešení ověřte na reprezentativním setu realistických příkladů a důsledně dokumentujte.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Petr Přibyl**
CCA Group, a.s.

Konzultant bakalářské práce: **Doc. Ing. Pavel Herout, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **5. října 2020**
Termín odevzdání bakalářské práce: **6. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 26. října 2020

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 5. května 2021

Karel Engl

Abstract

This thesis is focused on generating code from templates (scaffolding). Several scaffolding tools were analyzed and from this analysis and from company recommendation Yeoman was chosen. After validations of tool posibilites and comparation with company needs, suitable areas for code generation were found. Generating was verifed on four software areas. All performed experiments prove, that scaffolding speeds up program development and has other advantages.

Abstrakt

Práce se zaměřuje na problematiku generování kódu ze šablon (scaffolding). Byla provedena analýza několika scaffoldovacích nástrojů a na základě této analýzy a firemního doporučení byl vybrán nástroj Yeoman. Po ověření reálných možností toho nástroje a porovnání s potřebami firmy byly určeny vhodné oblasti pro generování kódu. Celkově bylo generování kódu ověřeno na 4 různých softwarových částech. Všechny provedené experimenty potvrzují, že scaffolding urychlí vývoj programů a má další výhody.

Obsah

1 Úvod

Vývoj aplikací je dlouhá a drahá činnost. Proto je dobré najít způsoby a nástroje, jak vývoj aplikací urychlit. Jedna z možností je urychlení činností, které se často opakují. Příkladem může být zakládání nových projektů, vyvíjení REST služeb, vytváření databáze a jejich entit, ačkoliv je ve všech případech odlišná funkčnost. Tyto úkoly jsou si podobné. Toto často vede ke kopírování částí z předchozích projektů, což může způsobovat chyby.

Byla vyvinuta řada nástrojů na urychlení těchto činností. Tyto nástroje se souhrnně označují „scaffolding“. Jejich výhodou a další důvod proč je vhodné tyto nástroje použít, kromě zautomatizování částí vývoje, je jednota stylu kódu. Tím je zajištěna lepší čitelnost kódu pro všechny programátory ve firmě a zároveň se urychlí zaučení nových nástupů do firmy. Práce se zaměřuje na prozkoumání scaffoldovacích nástrojů a jejich použití v praxi. Cílem práce je urychlit vývoj projektů, ať už jejich zahájením, nebo urychlením vývoje jejich částí.

2 Scaffolding

Scaffolding v programování slouží ke generaci kostry projektu nebo jeho částí pomocí připravených šablon.

2.1 Generování kostry projektu

Generování projektu ve scaffoldingu slouží k vytvoření kostry projektu. Programátorovi se voláním scaffolding nástroje s parametry jako je název a typ projektu vygeneruje kostra projektu, který je se stavitelný a spustitelný. Běžně je výsledkem spustitelný kód, který většinou nemá žádnou činnost. Pokročilejší nástroje umožňují vygenerovat strukturu backendu, frontendu a databáze najednou. Po vygenerování projektu tým definuje vzhled a služby projektu. Tento proces výrazně urychlí zahájení projektů, zejména u menších aplikací například mikroslužeb. Většinu nástrojů na scaffolding lze rozšiřovat a je možné napsat si vlastní generátory.

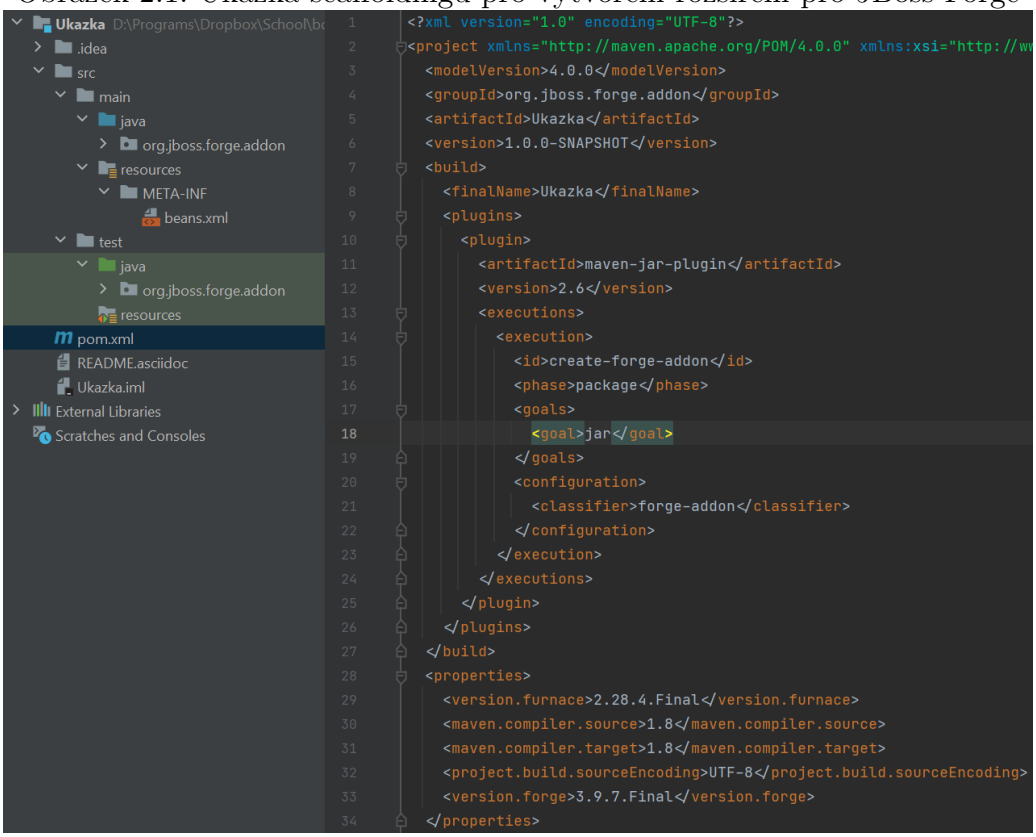
2.2 Fragmenty kódu

Scaffolding jako generátor kódu lze využít v často se opakujících a podobných úlohách. Pro generování kódu scaffolding používá šablony. Šablony určují obsah výsledného kódu. Často jsou parametrizovány a nastavením parametrů lze dále ovlivnit kód, který potřebujeme vygenerovat. Takto lze například při generování určit názvy objektů, proměnných atd. Pokročilejší nástroje umožňují psát do šablon i podmínky.

2.3 Příklady scaffoldingu

Jednoduchá ukázka generování kostry projektu, konkrétně projektu na psaní rozšíření do JBoss Forge, je jeho příkaz `project-new -type forge-addon`. Jak je vidět na obrázku 2.1, je zde vytvořena základní struktura Maven projektu a k tomu jsou v POM souboru připraveny nastavení a závislosti, nutné pro psaní rozšíření pro JBoss Forge. I na takto jednoduchém případu je vidět, že se dá ušetřit čas na vytváření struktury při zakládání projektu a na hledání potřebných závislostí a nastavení.

Obrázek 2.1: Ukázka scaffoldingu pro vytvoření rozšíření pro JBoss Forge



Jako příklad použití šablony je uvedena třída pro CRUD (create, read, update, delete) operace číselníku. Jediné, co se v kódu mění, je jméno číselníku. Ukázky šablon viz listing 2.1 a listing 2.2. Více o příkladu v části 5.2 Příklad.

Listing 2.1: JBoss Forge šablona CRUD operací číselníku

```
@Override
public ResponseEntity<Void> post${name}(
        ${name}Dto ${nameLower}Dto)
{
  ${name}Dto.setKod(${name}Dto.getKod().toUpperCase());
  if (Objects.nonNull(${name}Repository.findByKod(
          ${nameLower}Dto.getKod())))
  {
    throw new RecordAlreadyExistsException(
```

```
            CiselnikCommon.postConflictErrorText(
                    ${nameLower}Dto.getKod()));
  }

  ${name}Ent ${nameLower} = new ${name}Ent();
  ${nameLower}Mapper.map(${nameLower}, ${nameLower}Dto);
  ${nameLower}.setKod(${nameLower}Dto.getKod());

  ${nameLower}Repository.save(${nameLower});

  return new ResponseEntity<>(HttpStatus.CREATED);
}
```

Listing 2.2: Yeoman šablona CRUD operací číselníku

```
@Override
public ResponseEntity<Void> post<%= nazev %>(
       <%= nazev %>Dto <%= nazevLower %>Dto)
{
  <%= nazev %>Dto.setKod(<%= nazev %>Dto.getKod()
      .toUpperCase());
  if (Objects.nonNull(<%= nazev %>Repository.findByKod(
      <%= nazevLower%>Dto.getKod())))
  {
    throw new RecordAlreadyExistsException(CiselnikCommon
      .postConflictErrorText(<%= nazevLower%>Dto.getKod()));
  }

  <%= nazev %>Ent <%= nazevLower%> = new <%= nazev %>Ent();
  <%= nazevLower%>Mapper.map(<%= nazevLower%>,
    <%= nazevLower%>Dto);
  <%= nazevLower%>.setKod(<%= nazevLower%>Dto.getKod());

  <%= nazevLower%>Repository.save(<%= nazevLower%>);

  return new ResponseEntity<>(HttpStatus.CREATED);
}
```

Další scaffolding nástroje je Lombok. Jedná se o Java knihovnu, která přidává několik anotací. Mezi hlavní patří `Getter`, `Setter`, `NoArgsConstructor`, `AllArgsConstructor`, `Builder` `Data`. Anotace `Getter` a `Setter` se dají použít nad konkrétní proměnnou třídy nebo celou třídou, což vytvoří automaticky getry a setry pro všechny proměnné třídy. Anotace `Data` obsahuje anotace `Getter`, `Setter` a `NoArgConstructor`. Při kompilaci kódu jsou vytvořeny základní verze getrů a setrů, případně konstruktorů a builderů. To ušetří mnoho času, jak na vývoji, tak údržbě kódu. Vhodné využití je obzvlášť pro třídy entit nebo DTO (Data Transform Object), viz obrázek 2.3.

Listing 2.3: Lombok ukázka

```
@Data
@Builder
public class OsobaDto {

  private String jmeno;

  private String prijmeni;

  private LocalDate datumNarozeni;

}
```

OpenAPI slouží ke psaní RESTových služeb. Definice se vytváří v souboru ve formátu YAML. Tuto definici lze použít jako vstup pro scaffolding. Pro framework Spring existuje Maven plugin, který vygeneruje kontrolery, DTO a rozhraní pro službu. Poté stačí jen implementovat rozhraní. V rozhraní jsou definovány funkce s názvem určené `operationId` s parametry definovanými v sekci `requestBody` a návratovou hodnotou definovanou v části `content` v `responses`. Toto velice urychluje vývoj REST služeb. V obrázku 2.4 je definice REST služby pro obchod, která pro zadaný `guidObchod` vrátí jeho DTO.

Listing 2.4: Ukázka definice REST služby v OpenAPI

```
openapi: 3.0.1
info:
  title: Registr osob
```

```
  description: ukazka OpenAPI definice REST sluzby
    version: 1.0.0

paths:
  /osoba/{guidOsoby}:
    get:
      operationId: getOsobu
      parameters:
        - in: path
          name: guidOsoby
          required: true
          scheme:
            type: string
            format: uuid
      responses:
        200:
          description: osoba nalezena
          content:
            application/json:
              scheme:
                $ref: '#/components/schemas/OsobaDto'
        404:
          description: osoba nenalezena

components:
  schemas:
    OsobaDto:
      type: object
      properties:
        guid:
          type: string
          format: uuid
       jmeno:
         type: string
       prijmeni:
         type string
       datumNarozeni:
         type: string
         format: date-time
```

3 Zavedená pravidla výroby softwaru

3.1 Vývoj software ve firmě

Firma CCA Group a.s. se zaměřuje na vývoj aplikací na míru, systémovou integraci, IoT (Internet of Things), správu dokumentů a business intelligence. Aktuálně vyvíjené projekty jsou zaměřené na aplikace na míru a správu dokumentů. Softwary jsou za frontend psány v jazyce TypeScript ve frameworku Angular. Backend je psán v Javě ve frameworku Spring. Komunikace mezi frontendem a backendem, popřípadě mezi moduly backendu jsou vedeny přes REST (Representational state transfer) služby. Logování je řešeno pomocí Graylogu. Databáze jsou v jazyce PostgreSQL. Editování databáze je většinou zajištěno pomocí Liquibase scriptů. Občas jsou použity obyčejné SQL (Structured Query Language) scripty. Komunikace mezi Javou a databází je prováděna přes Hibernate. Analýza je prováděna v nástroji Enterprise Architect.

Firma aktuálně využívá moduly. Každý modul se stará o malou část aplikace. Například správa spisu se stará o spisy, číselníky se starají o čtení číselníkových hodnot z databáze, registr subjektů se stará o registraci osob a jejich databázi. Cílem je vytvořit nezávislé moduly. Když se na trhu objeví objednávka na software, použijí se podle požadavků existující moduly, upraví se podle požadavků zadavatele a propojí se. Použití stále stejných modulů a jejich následná modifikace umožní urychlení vzniku softwaru.

3.2 Analýza možného použití scaffoldingu

Vzhledem k modulárnímů vývoji je zakládáno mnoho modulů. Všechny moduly mají velké množství stejný souborů a řadu velice podobných. Tudíž vytváření nových modulů probíhalo kopírováním z existujících modulů. To vede často k chybám. Například je možné, že některé soubory nejsou zkopírovány, či následně upraveny, nebo dojde k překlepu během úprav. Nevýhodou tohoto procesu je jeho časová náročnost, obtížnost a nároky na vysoké znalosti programátora, aby věděl co zkopírovat a kde je co potřeba upravit. Proto prvním místem možného použití scaffoldingu by bylo generování koster

modulů. Generování koster pro modul je uvedeno v kapitole 7. V kapitole 9 je generátor kostry pomocné knihovny pro modul.

Další možnosti, kde se dá scaffolding uplatnit, je u vytváření všeho potřebného pro CRUD operace. Jedná se o nejčastější typ vytvářeného kódů v komponentách. Zde bývá celý kód velice podobný a jediné, co se mění, je definice entity, pro kterou je CRUD vyroben, a s tím spojené názvy tříd. Generátor je v kapitole 8.

Činnost, kterou velice často opakují frontend vývojáři, je psaní HTML gridů. Gridem se rozumí přehledné zobrazení více záznamů ve formě tabulky. Velká část v HTML je stejná, jednotlivé sloupečky se liší v názvech a typech. Tudíž po zadaní těchto proměnných pro všechny sloupečky je možné grid vygenerovat. Generátor gridu se nachází v kapitole 10.

Nepochybně by bylo možné najít další místa pro scaffolding, ale práce je zaměřena na výše zmíněné příklady, které byly vybrány z důvodu diverzity mezi jednotlivými problémy.

4 Existující nástroje

Pro účely práce bylo třeba vybrat nástroje, které splňují určité podmínky. Byly hledány nástroje, které se dají použít ve více programovacích jazycích. Výstupy nástroje musí být možné konfigurovat. Vhodné by bylo, aby byl nástroj vyvíjen a udržován. Dále jsou uvedeny ty, které byly v rámci práce zkoumány.

4.1 Yeoman

Yeoman umožňuje generování základů projektů nebo částí projektů ze šablon. Jedná se aktuálně o velice populární nástroj a díky tomu existuje mnoho rozšíření. Jedním z nich je JHipster, [6] jedná se o velmi používaný scaffolding framework na zakládání projektů. Podporuje několik jazyků frontendu, backendu a několika možností databází. Krom mnoha rozšíření existuje pro tento nástroj rozsáhlá dokumentace a mnoho tutoriálů na použití a psaní vlastních generátorů. Rozšíření jsou psaná v JavaScriptu, ale vygenerovaný kód lze snadno získat v jakémkoliv požadovaném jazyce nebo jeho nástavbě, jako jsou různé frameworky například Spring, Angular a podobně. Do šablon kódu lze vkládat JavaScript kód, který proběhne během generování. Tudíž v šablonách je například možné použít `if`, pokud chceme určitou část kódu vygenerovat, nebo se také dají použít cykly. Například když se kód generuje podle pole parametrů. Spuštění je v příkazové řádce příkazem `yo` a lze přidat rovnou název rozšíření, které chceme použít. Zajímavá vlastnost Yeoman je, že při generování souborů, pokud dojde ke kolizi, si uživatel může zvolit, jestli chce soubor přepsat nebo přeskočit.

4.2 JBoss Forge

JBoss Forge [5] byl doporučen k prozkoumání vedoucím bakalářské práce. JBoss Forge nabízí několik základních generátorů pro založení projektů, vytvoření databázových entit, i generací fragmentů kódu. Samostatně obsahuje zejména příkazy pro generování fragmentů JavaEE aplikací. Spuštění je přes příkazovou řádku, kde se aktivuje příkazem `forge`. Poté je možné používat jeho jednotlivé příkazy.

Pro JBoss Forge lze psát rozšíření, ve kterých je možné vytvořit vlastní příkazy. Generátory se vytváří v Java Maven projektu se závislostmi na JBoss Forge generátor knihovny. Lze generovat kódy jiných jazyků, ale přijdeme tím o výhody, které JBoss Forge pro Javu má, jako jsou code injection (vložení kódu do existujícího souboru), generování entit, podmínky pro spuštění příkazu a podobně. Tyto vlastnosti by bylo možné vytvořit i pro jiné jazyky, ale zahrnovalo by to obrovské množství práce.

4.3 Celerio

Celerio [2] je nástroj na generování kódu ze šablon. Šablony jsou postaveny na frameworku Apache Velocity. Pomocí šablon autoři vytvořili ukázky pro založení projektu v Angularu 4, JavaEE a podobně. Případy, na kterých to ukazovali, jsou velice specifické a je jednoduší použít jiný nástroj na založení daných projektu. Navíc tento nástroj není již dále vyvíjený. Další jeho nevýhodou je stručná dokumentace a nedostatek návodů na internetu k použití tohoto nástroje.

4.4 Spring Roo

Spring Roo [7] slouží jako scaffolding nástroj pro Spring. Tento nástroj jsem zkoumal z důvodů, že skoro všechny backend části aplikací ve firmě jsou vyvíjeny ve Springu. Roo umí vytvořit projekt a připravit napojení na databázi, lze i ručně napsat entity. Výhoda této možnosti je možnost vygenerovat repository a základní služby. Dokonce je možné napsat i služby na vyhledávání podle jiných parametrů než jen primárního klíče. Ve skriptu lze i vygenerovat kostry pro testy tříd nebo metod. Spring Roo je velice zajímavý nástroj, bohužel jeho využití je pouze pro Spring.

4.5 Telosys

Telosys [8] je další z nástrojů, který využívající pro své šablony framework Velocity. Telosys pracuje buď s modelem databáze nebo s doménově specifickým jazykem (DSL). V případě práce s modelem databáze, se Telosys na databázi napojí a poté z tabulek a šablon vygeneruje kód. Druhá možnost práce s Telosys je napsat si DSL model a pro ten vygenerovat daný kód. Ačkoliv se v celku jedná o zajímavý nastroj, je jeho použití omezené na generování služeb pro databázi nebo služby s ní spojené.

4.6 Angular CLI

Angular CLI [1] obsahuje sadu příkazů. Některé z nich slouží jako scaffolding, který umožňuje několik základních úkonů. Například jako založit kostru projektu, vytvořit strukturu pro stránku nebo vygenerovat fragment kódu ze šablony. Další zajímavá vlastnost je schopnost aktualizovat závislosti aplikace. Tento nástroj dokáže i věci mimo scaffolding, například spuštění aplikace a testů. Nástroj je základ pro používání Angularu, ale neumožňuje přidat vlastní příkazy.

4.7 Ostatní

Další nástroje, které byly prozkoumány a stojí za zmínku jsou:

1. JHipster [6] – jedná se o pokročilý scaffolding generátor vytvořený v nástroji Yeoman, slouží ke generování koster projektů, má mnoho možností pro frontend, backend a typy databází
2. Freemaker [3] – slouží ke generování kódů pro HTML stránky
3. Grails [4] – framework pro Groovy nebo Javu webové aplikace, sám o sobě není scaffoldovací nástroj, ale má scaffolding možnosti generovat svůj kód

5 Výběr nástroje

5.1 Zúžení výběru

Rovnou vyřazený byl Angluar CLI. Je spíše nástroj pro vývoj Angular aplikace než scaffoldovací nástroj. Při výběru nástroje bylo uvažováno několik kritérií.

1. možnosti použití na více programovacích jazyků (nejdůležitější kritérium)
2. dostupnost dokumentace a návodů
3. nástroj je stále vyvíjen a udržován

Nástroj	Více programovacích jazyků	Dokumentace	Udržován
Yeoman	ano	rozsáhlá	vyvíjen
JBoss Forge	ano, ale ztrácí výhody	stručná	udržován
Celerio	ano	velice stručná	ne
Spring Roo	pouze Spring	stručná	ne
Telosys	ano, ale jen s databází	stručná	vyvíjen

Tabulka 5.1: Hodnocení nástrojů

Po sepsání do tabulky a jejím vyhodnocení vyšly jako nejlepší možnosti Yeoman viz 4.1 a JBoss Forge viz 4.2. Yeoman, protože podporuje jakýkoliv programovací jazyk, je vyvíjen a má udržovanou detailní dokumentaci. Yeoman má také velkou komunitu kolem sebe, která vytváří generátory. V době psaní dokumentace byly na oficiálních stránkách odkazy na zhruba 10 000 veřejně dostupných generátorů komunity.

JBoss Forge, protože je použitelný na více jazyků, je udržovaný a byl doporučen vedoucím práce. JBoss Forge sice nemá nejlepší dokumentaci a ztrácí nějaké své výhody v jiných jazycích, ale stále je použitelný. Pro tyto dva zvolené nástroje byl vytvořen jednoduchý testovací generátor. Cílem bylo zjistit možnosti nástrojů, obtížnost vyvíjení vlastních rozšíření a vybrat finální nástroj, ve kterém budou implementovány generátory práce.

5.2 Příklad

Jako příklad byla zvolená třída pro správu číselníku. Jde o RESTovou webovou službu, která umožňuje CRUD operace nad jednoduchým číselníkem. V rámci příkladu bylo nutné umožnit zadat název číselníku. Zadaný název číselníku musí být na různých místech kódu modifikován co do velikosti písmen. V příkladu byl pro ilustraci uveden číselník Státu.

Listing 5.1: Část cílového kódu, při zadání názvů číselníku Stát

```
@Override
public ResponseEntity<Void> postStat(StatDto statDto)
{
  StatDto.setKod(StatDto.getKod().toUpperCase());
  if (Objects.nonNull(StatRepository.findByKod(
    statDto.getKod())))
  {
    throw new RecordAlreadyExistsException(
      CiselnikCommon.postConflictErrorText(statDto.getKod()));
  }

  StatEnt stat = new StatEnt();
  statMapper.map(stat, statDto);
  stat.setKod(statDto.getKod());

  statRepository.save(stat);

  return new ResponseEntity<>(HttpStatus.CREATED);
}
```

5.2.1 JBoss Forge

JBoss Forge [5] má pro vygenerování kostry projektu příkaz `project-new -type forge-addon`, kde `type` v tomto případě určuje, že se jedná o JBoss Forge rozšíření. Ukázku vygenerované kostry projektu je možné vidět na obrázku 2.1. Poté pomocí příkazu `addon-new-ui-command` se vygeneruje struktura pro příkaz. U třídy s příkazem se dá nastavit, podle toho od jaké třídy dědí, za jakých podmínek je příkaz možné spustit. Příklad možností je

Maven projekt, JavaEE projekt, projekt s Gitem, bez podmínky spustitelný kdykoliv a podobně. Část šablony použité v tomto příkladu je na obrázku 2.1.

Pro to, aby bylo možné generátor použít, je potřeba nejdříve projekt sestavit a příkazem `addon-build-and-install` nainstalovat. Poté je takto vytvořený příkaz možné v prostředí Forge používat. Pro vyzkoušení je potřeba nejdříve zapnout Forge. Vytvořený příkaz se jmenuje `ciselnik`. Forge se zeptá na jméno číselníku viz obrázek 5.1. Poté se použije šablona a vygeneruje se Spring služba pro daný číselník. Do jména třídy se použije název s velkým písmenem a do jmen proměnných je použito písmeno malé.

Obrázek 5.1: Příklad spuštění JBoss Forge generátoru

```
D:\Programs\Dropbox\School\bc\testovani\forge>forge
Using Forge at D:\Programs\JBoss Forge\forge-distribution-3.9.7.Final

    _____
   |  __ \                    _
   | |__) |__  _ __ __ _  ___ | |_
   |  ___/ _ \| '__/ _` |/ _ \| __|
   | |  | (_) | | | (_| |  __/| |_
   |_|   \___/|_|  \__, |\___| \__|
                    __/ |
                   |___/

JBoss Forge, version [ 3.9.7.Final ] - JBoss, by Red Hat, Inc. [ https://forge.jboss.org ]
Hit '<TAB>' for a list of available commands and 'man [cmd]' for help on a specific command.

To quit the shell, type 'exit'.

[forge]$ ciselnik
***INFO*** Required inputs not satisfied, entering interactive mode
* CiselnikName (Enter ciselnik name):  Stat
***SUCCESS*** Command 'generate' successfully executed!
```

5.2.2 Yeoman

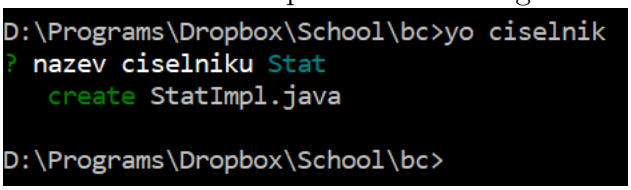
Yeoman [10] má taktéž scaffolding možnosti pro vytvoření generátoru, které připraví kostru projektu. Nejdříve je ale potřeba připravit složku se jménem generator-*název*, kde *název* je jméno našeho generátoru. V této složce je potřeba buď příkazem `npm init` nebo ručně vytvořit soubor `package.json`. Tento soubor slouží podobně jako Maven POM soubor. Jsou zde definované závislosti, jméno a verze projektu a podobně. V tomto souboru musí část `files` obsahovat `generators`, části `keywords` a `dependencies` musí obsahovat `yeoman-generator`, u dependencies je potřeba dodat i verzi, viz listing 5.2. Potom se příkazem `npm install -save yeoman-generator` vytvoří kostra projekt pro Yeoman generátor. Hlavní soubor `index.js` je zapotřebí mít v adresáři `generators/app/`.

Listing 5.2: Ukázka package.json souboru

```
{
  "name": "generator-cca-rocket-component",
  "version": "1.0.0",
  "description": "generator kostry projektu komponent",
  "main": "index.js",
  "files": [
    "generators"
  ],
  "dependencies": {
    "yeoman-generator": "^2.0.5"
  },
  "keywords": [
    "yeoman-generator"
  ],
  "author": "Engl Karel",
  "license": "ISC"
}
```

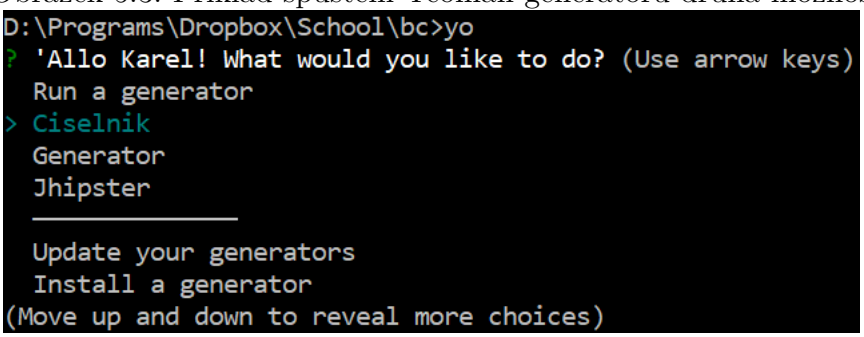
Rozšíření se píší v JavaScriptu. Generátory jsou rozděleny na 8 částí [11]. Pro příklad stačí použít části `prompting` a `writing`. Další části pro tuto práci nemají význam a proto nejsou uvedeny. Část `prompting` slouží ke komunikaci s uživatelem, zde v příkladu uživatel nastaví jméno číselníku. Část `writing` slouží k samostatnému generování kódu, zde se v příkladu vygeneruje REST služba pro číselník. K instalaci lokálně vytvořeného generátoru je zapotřebí ve složce projektu zadat příkaz `npm link` [10].

Obrázek 5.2: Příklad spuštění Yeoman generátoru



Pro spuštění generátoru jsou dvě možnosti. První je příkazem `yo` a zadáním jména generátoru viz obrázek 5.2, nebo spuštěním Yeomena příkazem `yo` a vybráním z nainstalovaných generátorů viz obrázek 5.3. Generátor se uživatele zeptá na jméno, poté je vygenerovaná Java Spring služba pro číselník.

Obrázek 5.3: Příklad spuštění Yeoman generátoru druhá možnost



5.3 Zhodnocení příkladu

5.3.1 JBoss Forge

Při psaní generátoru v JBoss Forge byl problém s nedostatkem dokumentace a tutoriálů. To vedlo k tomu, že jsem psaní generátoru dělal metodou pokus omyl a zkoumáním kódu JBoss Forge knihoven, abych zjistil, jak se mají použít. Oficiální tutoriál [5] ke psaní rozšíření nástroji byl psán ještě k verzi 2.x.x. a občas s poznámkami jak dané kroky udělat v beta verzi 3.0.0 s tím, že aktuální verze JBoss Forge je 3.9.7. Další problém je, že jakmile opustíme Java projekty, ztratíme výhody mnoha vytvořených knihoven pro JBoss Forge. To bohužel není komunitou pro ostatní jazyky vynahrazeno. Generátorů je malý počet a bývají zastaralé, mnohdy několik let od poslední aktualizace. Výhoda může být pro jazyk Java v případě, že využije generování do kódu.

5.3.2 Yeoman

Zkoušení psaní generátoru v Yeomen bylo jednodušší díky rozsáhlé dokumentaci a návodech na oficiálních stránkách, ale i uživatelských návodech. Yeoman má za sebou velkou komunitu, která vytváří značné množství generátorů. Velkou výhodou je také možnost psaní JavaScript kódu do šablon a tím upravit výsledný kód. Další výhodou je obecnost, nezáleží na jazyku výsledného kódu. Nevýhodou může být naopak psaní generátorů v JavaScriptu. JavaScript nemá silnou typovou kontrolu a je náchylný k chybám.

5.3.3 Finální výběr

Vyhodnocením poznatků popsaných v částech 5.3.1 a 5.3.2 bylo rozhodnuto dál používat Yeoman.

6 Principy psaní generátorů

Vytváření generátorů se dá rozdělit podle typové úlohy. Tyto úlohy se v každém generátoru řeší podobně.

6.1 Čtení argumentů z příkazové řádky

Argumenty z příkazové řádky jsou definovány v konstruktoru generátoru [12]. Při nastavení jejich definice se parsuje `args` a zjistí se, jestli byl vyplněn. V případě, že ano, lze jeho hodnotu získat z `opts.<název argumentu>`. Pokud ne, tak je použita defaultní hodnota, a pokud ani ta není, tak je hodnota nastavena na `null`.

Listing 6.1: Ukázka konstruktoru s argumentem v příkazové řádce jménem outputFilename

```
constructor(args, opts) {
  super(args, opts);

  this.argument("outputFilename", {
    type: String,
    required: false,
    description: "Filename to where is the grid generated",
    default: "grid.html"
  });

  outputFile = opts.outputFilename;
}
```

6.2 Čtení argumentů zadané uživatelem v rozhraní Yeomana

Pro čtení těchto argumentů v Yeomanu existuje funkce `prompting` [12], ve které se nadefinují jednotlivé argumenty. Každý argument by měl obsahovat alespoň základní definici, a to:

- `type` – typ argumentu, mezi základní patří:
  - `checkbox` – možnosti se nadefinují na úrovni jako `type` do pole `choices`, každý má `name` - jméno a `value` - hodnotu
  - `input` – vstup je text, který zadá uživatel
  - `confirmation` – potvrzovací dialog ano/ne, zajímavé je, že jakýkoliv text začínající y/Y je brán jako ano (yes), cokoliv jiného je ne.
- `name` – název argumentu
- `message` – zpráva, která se zobrazí uživateli po spuštění generátoru, při zadávání argumentu

Listing 6.2: Ukázka 3 parametrů zadaných v rozhraní yeomana

```
await this.prompt([
  //definition of common inputs
  {
    type: "confirm",
    name: "editable",
    message: "Is the grid editable ? ",
   },
   {
    type: "input",
    name: "gridName",
    message: "Grid name, format (camelCase)"
   },
   {
    type: "input",
    name: "number",
    message: "Number of colums",
   },
 ]);
```

6.3 Psaní šablon

Šablony se dají rozdělit na část textu a kódu. Text je ve finální verzi stejný. Během generování se provede kód a na jeho základě se vytvoří text. Jedna

možnost je mít nějaký atribut a ten v textu použít. To se dělá pomocí `<%= název atributu %>`.

Listing 6.3: Příklad vložení atributu

```
public class <%= entityName %>Ent
```

Další možnost je psát, JavaScript kód přímo do šablony. Zahájení psaní JavaScriptu je pomocí `<%` a ukončení `%>`.

Listing 6.4: Příklad použití podmínky

```
<% if (atribute.nullable === false) {
  %>, nullable = false<%
} %>
```

Tento segment do šablony vloží text `, nullable = false`, za podmínky, že `atribute.nullable` je `false`. Krom podmínek je užitečné použít cykly.

Listing 6.5: Příklad použití cyklu

```
<% for (type of types) {
    if (type.javaImport == null) {
        continue;
    }
    for (atribute of atributes) {
        if (atribute.type === type.java) {
            %>import <%= type.javaImport %>;<%
            break;
        }
    }
} %>
```

Tento segment se stará o to, aby entita měla správné importy. Prochází nadefinované typy a zjistí, jestli nějaký atribut je daného typu. Pokud ano, a tento typ má nějaký Java import, tak je použit. Jak je vidět, v šablonách je možné lehce použít JavaScriptu k doplnění potřebných dat do výsledného kódu.

6.4 Generování souborů

Soubory se generují v Yeomanu ve funkci `writing` [9]. Pomocí funkce `this.fs.copyTpl` se generují jednotlivé soubory. Při volání této funkce jsou dva povinné parametry. První parametrem je cesta k šabloně `this.templatePath`. Pokud je cesta zadaná relativně, je výchozí bod adresář `<název generátoru>/generators/app/templates`. Druhý povinný parametr určuje kam se má soubor vygenerovat `this.destinationPath`. Relativní cesta je od místa, kde byl generátor použit. Pokud je zadaná cesta do adresáře, který neexistuje, potřebné adresáře se vytvoří.

Jako další je možné předat pole objektů, které pak můžeme v kódové části šablon používat, nebo jednoduše vypsat viz kapitola 6.3 Psaní šablon.

Listing 6.6: Ukázka generování souboru ze šablony grid do souboru v proměnné outputFile s argumenty columns a gridName

```
this.fs.copyTpl(
  this.templatePath('grid.html'),
  this.destinationPath(outputFile), {
    columns: columns,
    gridName: gridName,
  }
);
```

7 Generátor kostry komponenty

Cílem generátoru je vytvořit kostru backend komponenty. Backend komponenta poskytuje aplikační logiku vybrané části aplikace a je přístupná přes REST rozhraní. Komponenta musí obsahovat vše potřebné k tomu, aby ji po vygenerování bylo možné ji zkompilovat a nasadit na Wildfly server. U komponenty musí být možné nastavit název, který se bude propagovat do všech částí aplikace vycházející z tohoto jména. Dále musí komponenta obsahovat integrační test, který ověří, že se spustí Spring kontext.

7.1 Důvod vytvoření generátoru

Komponenta funguje jako modul aplikace. Každou komponentu je ale možné samostatně nasadit a spustit na Wildfly serveru. Komponenta má na starost malou část aplikace. Například komponenta správa subjektů se stará jen subjekty a nic víc. Po seskládání komponent vznikne cílová aplikace. Protože každý modul se stará o relativně malou část, je během vytváření projektu vyrobeno velké množství modulů. Zakládání komponent je činnost, během které se kopíruje mnoho souborů a programátor může snadno zapomenout někde upravit název souboru, nastavení a podobně. Toto vede ke zbytečným chybám a zdržování.

7.2 Vstupy

7.2.1 Příkazová řádka

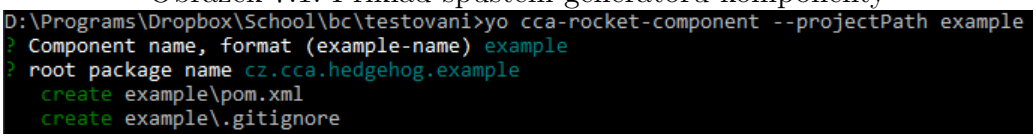
Generátor má pouze jeden vstupní parametr a to `--projectPath <cesta ke složce>`. `Project path` je cesta ke složce, kam se má kostra komponenty vygenerovat.

7.2.2 Argumenty zadané v rozhraní Yeomana

Po spuštění generátoru Yeoman požaduje 2 argumenty.

1. název komponenty, kde jednotlivá slova jsou oddělena pomlčkou
2. název kořenového balíčku, jednotlivé balíčky jsou odděleny tečkou

Obrázek 7.1: Příklad spuštění generátoru komponenty



7.3 Výstup

Výstupem generátoru je zkompilovatená komponenta, kterou lze nasadit na Wildfly server. Kostra projektu obsahuje 14 Java tříd, z niž 3 jsou v části pro testy, `.gitignore`, `pom.xml`, 7 souborů ve složce `main/resources`, 13 souborů ve složce `test/resources`, a 4 soubory ve složce `webapp`. Vygenerovaný kód dodržuje strukuturu Spring Maven projektu. *Název* v souborech je název komponenty zadané v prostředí Yeomena před generováním.

- kořenová složka
  - `pom.xml` – standardní POM soubor, obsahující potřebné pluginy a závislosti pro komponentu
  - `.gitignore` – zde jsou určené, jaké soubory má `git` ignorovat, např jsou to soubory vývojových prostředí, složka `target` a podobně
- složka `main/java`
  - kořenový balíček
    - *Název*`Application.java` – Main třída, spouští Spring context
    - `ServletInitializer.java`
  - balíček `config`
    - `WebConfig.java`
    - `SwaggerConfig.java` – zde je nastaven Swagger, ten se stará o zobrazení a popis všech dostupných endpointů aplikace
    - `MethodSecurityConfig.java`
    - `SecurityConfig.java` – zde je nastaveno zabezpečení aplikace
    - *Název*`DbConfig.java` – konfigurace databáze, obsahuje nastavení datových zdrojů, entit manageru a podobně
    - *Název*`PhysicalNamingStrategy.java`

        * `ParameterConfig.java`
    - balíček `db`
        * `ParametrRepository.java`
        * `ParametrSkupinaRepository.java`
- složka `main/resources`
    - `.openapi-generator-ignore`
    - `application.properties`
    - `log4j2.xml` – obsahuje konfiguraci logování
    - `messages.properties`
    - `messages_cs_CZ.properties`
    - `messages_en_US.properties`
    - *název*`-db.changelog-master.xml` - master liquibase changelog, zde jsou odkazy na ostatní changelogy a pořadí v jakém se mají spustit
- složka `main/webapp`
    - `index.html` – úvodní stránka aplikace
    - `jboss-deployment-structure.xml`
    - `jboss-web.xml` – zde je nadefinovaný URL kontextový kořen aplikace
    - `web.xml`
- složka `test/java`
    - `SpringContextTest.java` – test, který ověřuje spuštění Spring kontextu
    - *Název*`Test.java` – anotace obsahující veškeré nastavení pro spuštění integračního testu komponenty
    - `TestDataSourceConfig.java` – konfigurace testovacího datového zdroje
- složka `test/resources`
    - `activiti.cfg.xml`
    - `application-test.properties` – obsahuje upravené properties pro testy a definice testovací datového zdroje

  - `log4j2.xml` – upravená konfigurace log4j2, bez odesílání logu na Graylog
  - `process.xml`
- složka `test/resources/changelog`
  - `db.changelog-parametr-init.xml` – obsahuje changesety s inicializace tabulek parametrů
  - `db.changelog-test.xml` – master changelog pro testovací databázi, odkazuje na master changelog komponenty a zároveň inicializaci parametrů, možné je pak ještě doplnit odkazy na scripty vytvářející testovací data
  - `dbchangelog-parametry.sql` – obsahuje inicializace základních parametrů
- složka `test/resources/docker`
  - `Dockerfile` – spouštěcí soubor dockeru, obsahuje definici všeho k tomu potřebného, docker složí k vytvoření testovací lokální databáze
  - `init-create-tablespaces.sh` – skript s inicializací testovací databáze
  - `init-postgres.sql` – inicializace rolí a rozšíření testovací databáze
  - `czech.affix`
  - `czech.dict`
  - `czech.stop`

Výstupem generátoru je kostra komponenty. Kostra obsahuje vše potřebné, aby bylo možné rovnou začít vyvíjet endpointy pro daný modul. Tudíž je možné na komponentu použít generátor (viz kapitolu 8), a tím si vytvořit CRUD služby pro entitu.

7.3.1 Fragmenty testů

Generátor vytvoří kostru projektu, který zatím nemá implementovanou žádnou funkčnost. Jediné, co je tudíž možné testovat, je spouštění aplikace. To se ve firmě testuje integračním testem. Tento test, který se nachází v souboru `SpringContextTest.java`, má lokální databázi v Dockeru. Docker nejdříve spustí changesety, tím se jednak ověří, že jsou bez syntaktických

chyb, a zároveň se připraví testovací databáze. Poté test zkusí nasadit aplikaci. Pokud úspěšně proběhne, víme že aplikaci jde nasadit na Wildfly server.

7.3.2 Kroky po vygenerování

Po vygenerování už není třeba žádných dodatečných kroků pro správnou funkčnost.

7.4 Validace výsledků

Cílem mojí práce bylo zjistit, jestli výsledný vygenerovaný kód plní stanovené cíle. Nejdříve jsem vygeneroval komponentu „správu spisu“. Jméno vychází z již existující komponenty, stejně tak i jméno kořenového balíčku.

Obrázek 7.2: Vygenerování správy spisu

```
C:\Development\testovani>yo cca-rocket-component
? Component name, format (example-name) sprava-spisu
? root package name cz.cca.hedgehog.spis
```

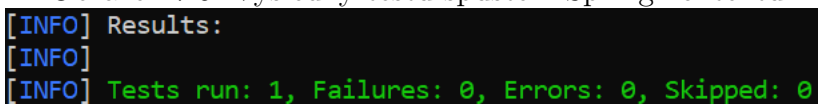
Následně jsem zkusil vygenerovanou komponentu zkompilovat a spustit její test. Sestavení projektu se provede příkazem `mvn clean install`, tento příkaz promaže soubory na sestavení komponenty (pokud existují), sestaví aplikaci a spustí testy. V tomto případě proběhne jen jeden test, a to integrační test na spuštění Spring kontextu. Na obrázku 7.3 je vidět spuštění Maven příkazu. Po určité době se začnou spouštět testy – na obrázku 7.4 je vidět zapínání Springu. Na obrázku 7.5 lze vidět, že test proběhl v pořádku, tudíž Spring se úspěšně zapnul. Na posledním obrázku 7.6 je vidět úspěch celého Maven buildu.

Obrázek 7.3: Spuštění buildu a testů

```
C:\Development\testovani>mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< cz.cca.hedgehog:sprava-spisu >--------------
[INFO] Building Hedgehog: komponenta sprava spisu Backend 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------
```

Obrázek 7.4: Ukázka části se startováním Spring kontextu

Obrázek 7.5: Výsledky testu spuštění Spring kontextu



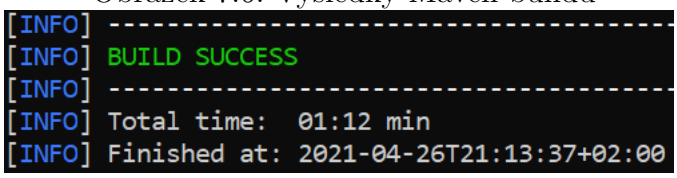
Dále bylo mým cílem pokusit se nasadit komponentu na Wildfly server. Tento server obsahuje potřebná nastavení, jako je nastavení pro připojení do databáze. Na výpisu konzole 7.1 je zahájení nasazování `war` souboru, obsahující komponentu.

Listing 7.1: Logovací záznam zkoušky nasazení, zahájení deploymentu

```
21:20:00,026 INFO [org.jboss.as]
  (Controller Boot Thread) WFLYSRV0060:
    Http management interface listening on
    http://127.0.0.1:9990/management
21:20:00,027 INFO [org.jboss.as]
  (Controller Boot Thread) WFLYSRV0051:
    Admin console listening on http://127.0.0.1:9990
21:20:49,224 INFO [org.jboss.as.repository]
  (DeploymentScanner-threads - 1) WFLYDR0001:
    Content added at location C:\DevTools\servers\
    wildfly-20.0.1.Final-rocket\standalone\data\content\41
    \67371541e124065da3adba0f3aac0153cc3753\content
21:20:49,269 INFO [org.jboss.as.server.deployment]
  (MSC service thread 1-1) WFLYSRV0027:
    Starting deployment of "sprava-spisu-0.0.1-SNAPSHOT.war"
    (runtime-name: "sprava-spisu-0.0.1-SNAPSHOT.war")
```

Z dalšího výpisu konzole 7.2 se dá vyčíst, že komponenta byla úspěšně nasazena na server a má endpointu `/rocket/sprava-spisu`.

Obrázek 7.6: Výsledky Maven buildu



Listing 7.2: Úspěšné nasazení na Wildfly server

```
21:21:10,802 INFO [org.wildfly.extension.undertow]
  (ServerService Thread Pool -- 85) WFLYUT0021:
    Registered web context: '/rocket/sprava-spisu'
    for server 'default-server'
21:21:10,987 INFO [org.jboss.as.server]
  (DeploymentScanner-threads - 1) WFLYSRV0010:
    Deployed "sprava-spisu-0.0.1-SNAPSHOT.war"
    (runtime-name : "sprava-spisu-0.0.1-SNAPSHOT.war")
```

Na závěr jsem prošel zdrojové kódy aplikace a zkontroloval jsem, jestli kód obsahuje *název* komponenty v kódu dle očekávání. Ve částech kódu jsou ukazovány jen nejdůležitější části. Některé části jsou porovnávány s nově vygenerovanou komponentou Example viz A.1. Začal jsem POM souborem. Definice ID (identification) v POM lze vidět na listingu A.1 a listingu A.2. `GroupID` je kořenový adresář bez posledního balíčku. `ArtifactID` je jméno komponenty, víceslovné názvy jsou odděleny pomlčkou. Main soubor aplikace se nachází na listingu A.3, zde pokud balíček nezačíná `ComponentScan`, je rozšířen o kořenový balíček komponenty viz listing A.4.

Jedna z nejdůležitějších tříd je *Název*`DbConfig`, která obsahuje nastavení databáze. Na tomto místě programátoři nejčastěji dělají chyby při ručním psaní komponenty. Vygenerovaný kód lze vidět na listingu A.5. Hlavní věci, které se zde mění, je balíček s enitami, v anotaci `EnableJpaRepositores` v části `basePackages`, a balíček s repositáři ve funkci `spravaSpisuEntityManagerFactory` na konci v `return builder` v části `packages`. Důležité je správně pojmenovat `bean` funkce a správně na ně odkazovat, jako je dříve zmíněný `spravaSpisuEntityManagerFactory`, nebo dále `spravaSpisuLiquibase`, `spravaSpisuDataSource` a podobně.

8 Generátor backend CRUD služby

Cílem tohoto generátoru je na základě definice entity vygenerovat vše potřebné pro službu, která implementuje CRUD operace krom delete. Delete není generován z toho důvodu, že systémy firmy většinou data nemažou, ale spíš se zneplatní číselníkovou hodnotou nebo datem platnosti. Vygenerovaný kód musí mít připravené kostry testů pro jednotlivé odpovědi endpointů.

8.1 Důvod vytvoření generátoru

Ve webových aplikacích firmy je programování CRUD operací jeden z nejčastějších úkonů programátora. To vede k opakování podobné práce i několikrát denně. Práce se opakuje i v rámci psaní jedné skupiny CRUD operací, a to konkrétně u entit. Vytváří se `changeset` pro založení entity do databáze, vytváří se třída entity v Javě a nakonec je pro danou entitu vytvářena i DTO třída. Pro jednu definici entity se tudíž třikrát opakuje velice podobná práce.

8.2 Vstupy

8.2.1 Příkazová řádka

Generátor backendu má dva nepovinné parametry v příkazové řádce. Jsou to `--projectPath <cesta ke složce projektu>` a `--mappingFile <cesta k souboru kofigurace>`. `Project path` je složka, kam se má cílový kód vygenerovat. V případě, že není zadán, jsou soubory vygenerovány do aktuální složky. `Mapping file` je JSON soubor, který obsahuje konfiguraci typu proměnných. Obsahuje pole jménem `types`. V tomto poli jsou nadefinované všechny typy proměnných. Každý typ se skládá se 4 povinných a 3 nepovinných parametrů.

Povinné

- `ea` – jméno typu v prostředí Enterprise Architect
- `java` – jméno typu v Javě
- `openApi` – jméno typu v OpenAPI

- `db` – jméno typu v Liquibase scriptu

Nepovinné

- `length` – maximální délka, používá se při různě dlouhých definic Stringu
- `javaImport` – jaký import je v Javě potřeba pro zkompilovatelný kód, příklad `java.time.LocalDate`
- `openApiFormat` – formát v OpenAPI, například UUID má typ `string` a formát `uuid`

8.2.2 Argumenty zadané v rozhraní Yeomana

Po spuštění generátoru se Yeoman zeptá na 4 parametry.

1. cesta k JSON souboru s definicí entity; soubor je možné vygenerovat z Enterprise Architecta
2. název kořenového balíčku, jednotlivé balíčky jsou odděleny tečkou
3. název schématu, kterému entita v databázi patří; vše je zapsáno malými písmeny, jednotlivá slova jsou oddělena znakem podtržítka _
4. kontextový kořen URL dané komponenty

Obrázek 8.1: Ukázka vstupu pro generátor CRUD operací

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-rocket-backend
? path to json file with entity definition Kniha.json
? root package name cz.cca.hedgehog.knihovna
? scheme name, format (three_word_scheme) knihovna
? URL context root of component rocket/knihovna
```

Formát souboru s definicí entity

Formát definice entity se skládá z `entityName` obsahující název entity a pole `atributes`. Každý atribut se má 3 povinné parametry.

- `name` – název atributu
- `type` – typ atributu v Enterprise Architectu
- `nullable` – true/false, jestli může atribut být null

8.3 Výstup

Výstupem je 8 souborů, z nich jsou 5 Java třídy, 1 YAML soubor s definicí endpointů a DTO a 2 textové soubory. *Název* je `entityName` z definice entity zadané při zadávaní parametrů do Yeomana, upraveno do formátu „camel case“. Camel case je formát, kde slova nejsou oddělena mezerou. Každé slovo začíná velkým písmenem. Výjimkou může být první písmeno prvního slova.

- *Název*`Ent.java` – obsahuje definici entity v databázi
- *Název*`Repository.java` – Spring repository dané entity
- *Název*`Mapper.java` – mapper entity, slouží k přemapování vygenerovaného DTO do entity a obráceně
- *Název*`ServiceImpl.java` – implementace služby s CRUD operacemi
- *Název*`Test.java` – připravená kostra testů všech běžných odpovědí endpointu
- `api-`*název*`.yaml` – obsahuje OpenApi 3 definice endpointů a definici DTO
- `pom.txt` – obsahuje část XML kódu, kterou je potřeba přidat do souboru `pom.xml`, pro generování tříd, nadefinované `api-`*název*`.yaml`
- `changelog.txt` – obsahuje Liquibase `changeSet` se založením tabulky v databázi, obsah je zapotřebí přidat do vhodného `changelog` souboru.

Struktura a kód jsou vygenerovány podle pravidel platných ve firmě. Všechny soubory jsou vygenerovány do složek, kam podle Maven projektů patří. To je zajištěno pomocí parametru `projectPath` a vstupního parametru názvu balíčku. Kód je bez spuštění testů zkompilovatelný a spustitelný. Po založení tabulky s entitou v databázi, ať už přes `changelog.txt` nebo jiným způsobem, jsou endpointy plně funkční.

8.3.1 Fragmenty testů

Generátor připraví kostry testů pro všechny běžně očekávané odpovědi endpointů. Všechny testy mají předpřipravené volání endpointu a očekávaný kód statusu.

- GET entity v pořádku
- GET enitity na neexistující entitu

- PUT entity v pořádku
- PUT na neexistující entitu
- PUT s konfliktem verze posílané a uložené entity
- PUT s konfliktem guidem enitity v JSON a URL
- POST entity v pořádku

Testy je potřeba doimplementovat podle dále uvedeného vzoru. Nejdříve je potřeba nad testovací třídu přidat anotaci *NázevKomponenty*`Test` viz soubor *Název*`Test` z generátoru komponenty sekce 7.3. Aby na tento krok nebylo zapomenuto, je ve vygenerované třídě `TODO` komentář. Pro zprovoznění testů je potřeba vytvořit testovací data, JSON soubory na volání služeb PUT a POST, a u služeb GET a PUT doplnit `guid` do cesty. Jeden test je už kompletně připravený jako vzor, a to `get`*Název*`NotFound`. *Název* je opět `enitityName` ze vstupního souboru.

Listing 8.1: Test GETu entita nenalezena

```
@Test
public void get<%= entityName %>NotFound() {
  RestAssured.get(COMMON_ENDPOINT +
                        UUID.randomUUID().toString())
                  .then()
                  .statusCode(404);
}
```

Aby se nezapomnělo na doimplementování těl testů, je vytvořen test, který nikdy neprojde. Jedná se o test `implementUs`.

Listing 8.2: Test implementUs

```
@Test
public void implementUs() {
   Assertions.assertTrue(false);
}
```

8.3.2 Kroky po vygenerování

Pro správnou funkčnost je potřeba přesunout obsah ze souboru `pom.txt` do souboru `pom.xml` do `plugins`, `plugin`, `openapi-generator-maven-plugin`, `executions`. Pokud chcete vytvořit entity v databázi, je potřeba obsah souboru `changelog.txt` vložit do nějakého vhodného `changesetu` a ten spustit. Také je potřeba přidat práva určené role k přístupu na tabulku. Jako poslední je potřeba implementovat testy viz 8.3.1.

8.4 Validace výsledků

Pro ověření výsledků jsem nejdříve vygeneroval CRUD operace pro entitu `Kniha`. Generování lze vidět na obrázku 8.1. Definice entity se nachází na listingu A.6. Použil jsem defaultní mapovací soubor viz listing A.8. Pro možnost porovnání jsem vygeneroval druhou sadu CRUD operací. Její generování lze vidět na obrázku A.2, definici entity na listingu A.7 a její mapovací soubor na listingu A.9. V mapovacích souborech jsou různé typy dat a jejich importy, které chceme ve finálním kódu použít.

Jak lze vidět po porovnání entit vygenerovaného kódu viz listing A.10 a listing A.11, jsou zde použity typ dat podle mapovacích souborů. Soubory obsahují i správný `import`. Na obrázku A.12 je možné vidět definice všech endpointů a jejich odpovědí. Také je zde definice DTO souboru. Na obrázku A.13 lze vidět stejnou informaci pro registr subjektů. Zde se v definicí endpointů liší jen cesta a názvy. DTO jsou podle očekávání odlišná, neboť odpovídají definicím entit. Na listingu A.14 je zobrazen changeset. Když je proveden, založí tabulku `Kniha` do databáze. Účelem všech ostatních souborů je, aby fungovaly služby CRUD operací. Jejich implementaci lze vidět na listingu A.16 a listingu A.17. Jak je vidět, kód je krom názvů identický. Na A.18 se nachází OpenAPI `execution`, který se vkládá do OpenAPI Maven pluginu v souboru `pom.xml`. Na poslední ukázce je možné vidět kostry integračních testů CRUD služeb knihy viz listing A.19.

9 Generátor klienta komponenty

Cílem generátoru je vytvořit kostru knihovny backend klienta ke komponentě. Generátor připraví strukturu bez definic jednotlivých endpointů komponenty. Vygenerovaná kostra musí obsahovat vše, aby programátorovi stačilo doplnit jen volání jednotlivých „interních“ endpointů komponenty. Interní endpointy slouží na komunikaci mezi komponenty.

9.1 Důvod vytvoření generátoru

Vzhledem modulárnímu vývoji jsou komponenty samostatné aplikace, tudíž pokud komponenty chtějí získat data, musí spolu komunikovat. To je zajištěno pomocí REST služeb. Pro každou komponentu, která nabízí nějaké interní služby ostatním komponentám, je potřeba udělat klienta. Většina komponent má alespoň nějaké interní endpointy. Protože máme mnoho komponent, vzniká i mnoho klientů. Vygenerováním kostry knihovny klienta tudíž ušetříme čas a zároveň se vyhneme možným chybám vzniklých při kopírování.

9.2 Vstupy

9.2.1 Příkazová řádka

Generátor má pouze jeden vstupní parametr a to `--projectPath <cesta ke složce>`. `Project path` je cesta ke složce, kam se má kostra klienta vygenerovat.

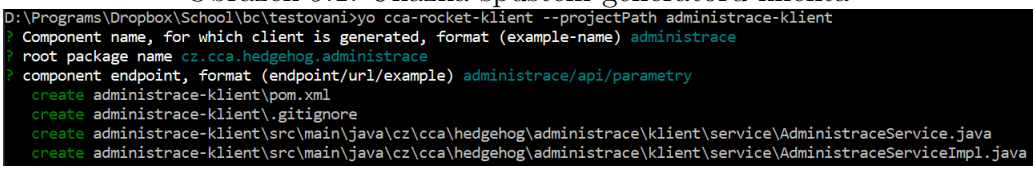
9.2.2 Argumenty zadané v rozhraní Yeomana

Po spuštění generátoru se Yeoman zeptá na 3 parametry.

1. název komponenty, pro kterou je klient generován, jednotlivá slova mají být oddělena pomlčkou
2. název kořenového balíčku, jednotlivé balíčky jsou odděleny tečkou

3. společný základ endpointů služeb komponenty, pro kterou je klient generován. URL začíná textem a poté je standardně oddělován znakem lomítka /

Obrázek 9.1: Ukázka spuštění generátoru klienta



9.3 Výstup

Výsledný kód obsahuje 4 soubory. *Název* je první parametr zadaný v rozhraní Yeomana. Na rozdíl od zadaného názvu je ve formátu „camel case“.

- *Název*`Service.java` – založí rozhraní
- *Název*`ServiceImpl.java` – založí třídu implementující rozhraní *Název*`Service.java` a připraví globání promněné potřebné v implementaci klienta
- `.gitignore` – standardní gitignore používaný v Java projektech ve firmě
- `pom.xml` – standardní POM soubor používaný v klientech, obsahuje všechny potřebné pluginy a závislosti pro klienta komponenty

Výstupem je připravená kostra Maven projekt knihovny klienta ke komponentě. Kód je zkompilovatelný.

9.3.1 Fragmenty testů

Vzhledem k tomu, že výstupem je pouze nejzákladnější kostra knihovny klienta, která nemá žádnou funkcionalitu, není možné nic testovat, tudíž ani nemá smysl vytvářet fragmenty testů.

9.3.2 Kroky po vygenerování

Po vygenerování je ještě zapotřebí implementovat endpointy REST služby dané komponenty. V rozhraní *Název*`Service.java` je potřeba pro každý endpoint připravit definici funkce. Pokud jsou potřeba nějaká DTO, je nutné vytvořit balíček `dto` a do něj je vložit. Následně je potřeba implementovat rozhraní do *Název*`ServiceImpl.java`. Po těchto krocích je klient připraven k použití.

9.4 Validace výsledků

Vzhledem k tomu, že generátor vytvoří jen kostru projektu, jediné, co bude ověřováno, je, zda se podaří knihovnu zkompilovat. Na obrázku 9.2 se nachází spuštění generování. Parametry jsou nastaveny tak, aby byla vygenerována kostra klienta ke komponentě `sprava-spisu`. Vygenerovaný kód je zkompilován Maven příkazem `mvn clean install`, jak je ukázáno na obrázku 9.3. Výsledek kompilace můžeme vidět na obrázku 9.4, jak je vidět, tak se knihovnu klienta podařilo úspěšně zkompilovat.

Obrázek 9.2: Spuštění generátoru klienta pro správu-spisu

```
C:\Development\bc\klient>yo cca-rocket-klient
? Component name, for which client is generated, format (example-name) sprava-spisu
? root package name cz.cca.hedgehog.spis
? component endpoint, format (endpoint/url/example) sprava-spisu
```

Obrázek 9.3: Spuštění generátoru klienta pro správu-spisu

```
C:\Development\bc\klient>mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< cz.cca.hedgehog:sprava-spisu-klient >
[INFO] Building Hedgehog: klient Sprava Spisu 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------
```

Obrázek 9.4: Spuštění generátoru klienta pro správu-spisu

```
[INFO] ------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------
[INFO] Total time:  7.651 s
[INFO] Finished at: 2021-04-30T12:54:47+02:00
[INFO] ------------------------------------
```

10 Generátor HTML gridu

Cílem generátoru je vytvořit HTML grid pro frontend. Během vytváření gridu musí být možné nastavit typ gridu, a to buď „zobrazovací“ nebo „editační“. Následně musí být možné definovat n-počet sloupečků a pro každý definovat název a typ. U editačních gridů musí být ještě možné nastavit, jestli je sloupeček editovatelný. Frontend obsahuje vzorové prvky pro vývojáře. Výsledný kód musí splňovat strukturu vzorového gridu.

10.1 Důvod vytvoření generátoru

Ve frontend aplikacích je grid velmi často používaný. Nachází se na všech „přehledových“ stránkách, ale může se vyskytovat i jinde, jako jsou stránky „detailu“. Téměř každá entita v aplikaci má svojí přehledovou stránku. Přehledové stránky zobrazují list entit se základními údaji, zatímco stránky detailu zobrazují jen jednu entitu, ale všechny její potřebné informace.

10.2 Vstupy

10.2.1 Příkazová řádka

Generátor gridu v HTML má pouze jeden argument z příkazové řádky. Argument je nepovinný. Jmenuje se `--outputFilename`. Určuje, do kterého souboru se má grid vygenerovat. V případě, že není zadán, je grid vygenerován v aktuální složce do souboru `grid.html`.

10.2.2 Argumenty zadané v rozhraní Yeomana

Po spuštění se Yeoman zeptá na 3 základní dotazy

1. jestli je grid editační, volba ano/ne
2. jméno gridu, zadává se ve formátu camel case
3. počet sloupečků v gridu

Následně se pro každý sloupeček zeptá na

1. jméno sloupečku
2. typ sloupečku, na výběr jsou možnosti, `text`, `date`, `numeric` a `boolean`
3. pokud je grid editační, tak se vyplňuje, jestli se daný sloupeček dá editovat volbou ano/ne

Obrázek 10.1: Příklad spuštění generátoru gridu

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-eisir-grid --outputFilename example/grid.txt
? Is the grid editable ?  Yes
? Grid name exampleToShow
? Number of colums 4
? 1. column name name
? 1. column type (Use arrow keys)
> text
  date
  numeric
  boolean
```

10.3 Výstup

Výstupem je jeden soubor, který se defaultně jmenuje `grid.html`. Tento soubor obsahuje HTML fragment kódu. Ve fragmentu se nachází grid definovaný programátorem. Kód si programátor vloží do kódu aplikace kam potřebuje. Fragment je vygenerován podle standardů firmy.

10.3.1 Fragmenty testů

Vzhledem k tomu, že je vygenerován jen malý fragment HTML kódu, který si programátor vkládá kam potřebuje, není víceméně možné napsat nějaký použitelný fragment testu.

10.3.2 Kroky po vygenerování

Po vygenerování gridu je potřeba daný kód překopírovat do cílového kódu. Pokud se jedná o editační grid, je ještě potřeba nastavit maximální délky textů u `text` sloupečku při editaci a minimální a maximální číselné hodnoty u `numeric` sloupečků. Maxima hodnot by bylo možné přidat i generátorů, ale místo toho bylo požadováno u nich nastavit malou hodnotu například u délky textu 3 znaky.

10.4 Validace výsledků

Ověření generátoru, provedu porovnáním vzoru gridu a generovaným gridem. Na HTML kódu viz listing 10.1 je vzorový grid. Obrázek 10.2 ukazuje spuštění generátoru s parametry tak, aby vznikl vzorový grid. Vygenerovaný kód lze vidět na A.20 a po jeho porovnáním se vzorem zjistíme, že je kód identický.

Obrázek 10.2: Spouštění generátoru pro vytvoření vzorového zobrazovacího gridu

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-eisir-grid
? Is the grid editable ?  No
? Grid name, format (camelCase) grid1
? Number of colums 4
? 1. name poradi
? 1. type numeric
? 2. name nazev
? 2. type text
? 3. name datumZapisu
? 3. type date
? 4. name datumVymazu
? 4. type date
```

Listing 10.1: Vzor zobrazovacího gridu

```
<hedgehog-grid [data]="dataGrid1"
               gridId="Grid1"
               [take]="take"
               [isDataLoading]="isLoading">
  <kendo-grid-column field="poradi"
                     title="poradi"
                     filter="numeric"
                     [width]="220">
  </kendo-grid-column>
  <kendo-grid-column field="nazev"
                     title="nazev"
                     [width]="220">
  </kendo-grid-column>
  <kendo-grid-column field="datumZapisu"
                     title="datumZapisu"
                     filter="date"
                     [width]="220"
                     [format]="'date' | getDateFormat">
```

```
  </kendo-grid-column>
  <kendo-grid-column field="datumVymazu"
                     title="datumVymazu"
                     filter="date"
                     [width]="220"
                     [format]="'date' | getDateFormat">
  </kendo-grid-column>
</hedgehog-grid>
```

Další typ je editační grid. Tento grid má opět svůj vzor, proti kterému je porovnáván. Na obrázku A.3 je vygenerování editačního gridu se stejnými parametry jako má vzor. Jak je vidět na kusech kódu vzoru viz listing A.21 a vygenerovaném kódu viz listing A.22, je kód identický, čímž generátor splňuje základní podmínku, že plní strukturu vzorového gridu.

Ve vzorových gridech není typ sloupečku `boolean`, proto na dalších příkladech budou ukázky vygenerovaného kódu s ním. Dále budou prezentována další nastavení generátoru, která dosud nebyla použita.

Jako další příklad je vygenerován zobrazovací grid do souboru `Kniha.html` obsahující všechny 4 základní typy sloupečků. Jeho spuštění je vidět na obrázku A.4. Vygenerovaný kód je možné vidět na listingu A.23. Jak je vidět, kód se úspěšně vygeneruje a dodržuje strukturu vzorových gridů.

Jako poslední příklad je vytvořen editační grid. Na obrázku A.5 je vidět jeho vygenerování. V tomto gridu jsou zobrazeny všechny možnosti, které nebyly ve vzorovém gridu. Všechny sloupečky krom sloupečku `kDispozici` nejsou editovatelnými. Vygenerovaný kód je zobrazen na listingu A.24. Při bližším průzkumu je vidět, že editační sloupeček `boolean` se téměř neliší od needitovatelného. To je dáno tím, že v době psaní generátoru se v projektu nikde neobjevil editovatelný sloupeček `boolean` a tudíž implementace nebyla zatím provedena. Prozatím je připraveno potřebné zapouzdření `ng-template`. Zbytek by bylo možné v budoucnu doplnit do šablony. Další čeho je možné si všimnout, že pokud sloupeček není editovatelný, je strukturou stejný jako ze zobrazovacího gridu.

11 Závěr

V rámci práce byly identifikovány oblasti, kde by bylo ve firmě možné a užitečné použít scaffolding nástroje. Po důkladné analýze byl vybrán nástroj Yeoman. S jeho využitím byly vytvořeny 4 generátory různých oblastí tvorby zdrojového kódu. Jedná se o generátor kostry projektu, generátor implementace CRUD operace nad entitou, generátor HTML gridu a generátor REST klienta ke komponentě.

Bylo ověřeno, že generátory úspěšně generují kód podle stanovených cílů každého generátoru.

Výsledky byly prezentovány ve firmě a byly dobře přijaty. Připravuje se zavedení Yeomana k urychlení vývoje. Aktuálně se připravuje integrace s Enterprise Architect.

V textu jsou mimo jiné zapsány i výhody použití scaffoldingu, které byly při zpracování potvrzeny. Jedná se o urychlení částí vývoje, předcházení zbytečným chybám jako jsou překlepů, chyby při kopírování z předchozích projektů a podobně. Další výhodou je dodržování standardních programátorských postupů firmy, neboť šablony strukturu dodržují. Vygenerované fragmenty nutí programátora napsat testy, jako například u generátoru implementace CRUD operací.

Práce splnila všechny body zadání a její výsledky budou využívány v praxi.

Literatura

[1] Angular CLI Documentation 2020. *Angular CLI*. https://angular.io/cli

[2] Celerio Documentation 2020. *Celerio Reference Guide*. http://www.jaxio.com/documentation/celerio/

[3] Freemaker Documentation 2020. *Freemaker Getting Started*. https://freemarker.apache.org/docs/pgui_quickstart.html

[4] Grails Documentation 2020. *Grails Scaffolding Documentation*. https://docs.grails.org/4.0.10/guide/scaffolding.html

[5] JBoss Forge Documentation 2020. *Hands on Lab*. https://forge.jboss.org/document/hands-on-lab

[6] JHipster 2020. *JHipster*. https://www.jhipster.tech/

[7] Spring Roo Documentation 2020. *Spring Roo Reference Documentation*. https://docs.spring.io/spring-roo/docs/current/reference/html/

[8] Telosys 2020. *Telosys*. http://www.telosys.org/

[9] Yeoman Creating generator - File system 2020. *Working with the file system*. https://yeoman.io/authoring/file-system.html

[10] Yeoman Creating generator - Getting started 2020-12-15. *Writing your own Yeoman generator*. https://yeoman.io/authoring/index.html

[11] Yeoman Creating generator - Running context 2020. *Generator runtime context*. https://yeoman.io/authoring/running-context.html

[12] Yeoman Creating generator - User interactions 2020. *Interacting with user*. https://yeoman.io/authoring/user-interactions.html

Přehled zkratek

- SQL – Structured Query Language, jazyk na komunikaci s databázi
- DTO – data trasport object, objekt který se používá pro výměnu dat mezi moduly aplikace
- REST – Representational state transfer, typ HTTP komunikace
- POM – Project Object Model, XML soubor obsahující definice Maven projektu
- XML – Extensible Markup Language, datový formát
- CRUD – Create, read, update, delete, zkratka pro základní operace nad entitou
- YAML – Ain't Markup Language, značkovací jazyk
- DSL – Domain Specific Language, programovací jazyk
- CLI – Command line interface, rozhraní příkazové řádky
- HTML – HyperText Markup Language, značkovací jazyk
- JSON – JavaScript Object Notation, objekt programovacího jazyku JavaScript
- URL – Uniform Resource Locator, reference na webový zdroj
- HTTP – Hypertext Transfer Protocol, protokol na přesun hypertextových dokumentů jako HTML
- IoT – Internet of things, jedná se o síť „věcí", jako jsou sensory, domácí spotřebiče, která si vyměňují data s ostatními zařízeními přes Internet
- ID – Identification, identifikační hodnota

A Přílohy

Zde budou uvedeny úplné části vygenerovaných kódů. V textech předchozích kapitol byly uvedeny jen jejich části potřebné pro pochopení. Přílohy jsou uvedeny jako ukázky kódu. Význam použití vyplývá z textu práce, kde jsou uvedeny formou odkazů. Zobrazované části kódu mohou mít oproti vygenerovanému kódu jiné zalomení řádek nebo počet mezer z důvodů omezené šířky textu v dokumentaci.

A.1 Přiložené soubory

K práci je přiložen soubor `generatory.zip`. Soubor obsahuje všechny vytvořené generátory. Generátory číselníku viz sekce 5.2.1 a viz sekce 5.2.2 byli vytvořeny čistě pro testování možností nástrojů a mohou obsahovat programátorské prohřešky. Dále je v souboru generátor komponenty viz kapitola 7, generátor CRUD operací viz kapitola 8, generátor kostry knihovny klienta viz kapitola 9 a generátor gridu viz kapitola 10. Každý z těchto 4 generátorů obsahuje uživatelskou příručku v souboru `README.md`

A.2 Generátor komponenty

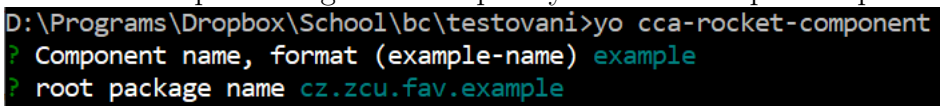
Listing A.1: POM definice ID správy spisu

```
<groupId>cz.cca.hedgehog</groupId>
<artifactId>sprava-spisu</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Listing A.2: POM definice ID example

```
<groupId>cz.zcu.fav</groupId>
<artifactId>example</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Obrázek A.1: Spouštění generátoru pro vytvoření Example komponenty



Listing A.3: Main aplikace Správa spisu

```
@SpringBootApplication
@ComponentScan(basePackages = "cz.cca.hedgehog")
public class SpravaSpisuApplication {
  public static void main(String[] args) {
    SpringApplication.run(SpravaSpisuApplication.class, args);
  }
}
```

Listing A.4: Main aplikace Example

```
@ComponentScan(basePackages = {"cz.cca.hedgehog",
                               "cz.zcu.fav.example"})
public class ExampleApplication ...
```

Listing A.5: Správa spisu konfigurace databáze

```
@Configuration
@EnableTransactionManagement(
    order = Ordered.LOWEST_PRECEDENCE - 1)
@EnableJpaRepositories(
    entityManagerFactoryRef =
        "spravaSpisuEntityManagerFactory",
    transactionManagerRef = Constants.TRANSACTION_MANAGER,
    basePackages = {"cz.cca.hedgehog.spis.db.repo",
      "cz.cca.hedgehog.user.db.repo"},
    repositoryBaseClass = ExtendedSimpleJpaRepository.class
)
public class SpravaSpisuDbConfig {

  @Primary
  @Bean(name = "spravaSpisuEntityManagerFactory")
  @DependsOn("spravaSpisuLiquibase")
  public LocalContainerEntityManagerFactoryBean
    spravaSpisuEntityManagerFactory(
          EntityManagerFactoryBuilder builder,
          @Qualifier("spravaSpisuDataSource")
              DataSource dataSource) {
    Map<String, Object> properties = new HashMap<>();
    properties.put("hibernate.hbm2ddl.auto", "validate");
    properties.put("hibernate.physical_naming_strategy",
        "cz.cca.hedgehog.spis.config." +
        "persistence.SpravaSpisuPhysicalNamingStrategy");
    return builder
            .dataSource(dataSource)
            .persistenceUnit("spravaSpisuPersistenceUnit")
            .packages("cz.cca.hedgehog.spis.db.domain",
                   "cz.cca.hedgehog.core.db.domain.konfigurace",
                   "cz.cca.hedgehog.user.db.domain")
```

```
            .properties(properties)
            .build();
    }

...

    @Primary
    @Bean(name = "spravaSpisuDataSource")
    public DataSource spravaSpisuDataSource() {
      return ProxyDataSourceBuilder
              .create(new JndiDataSourceLookup()
                  .getDataSource(spravaSpisuJndi()
                  .getJndiName()))
              .name(Constants.DB_NAME)
              .logQueryBySlf4j(SLF4JLogLevel.DEBUG)
              .build();
    }

    @Bean("spravaSpisuLiquibaseDataSource")
    @ConditionalOnExpression("T(Boolean).parseBoolean(" +
        "'${sprava.spisu.liquibase.enabled}')")
    public DataSource spravaSpisuLiquibaseDataSource() {
      return new JndiDataSourceLookup()
          .getDataSource(spravaSpisuLiquibaseJndi()
          .getJndiName());
    }

    @Bean
    @DependsOn("spravaSpisuDataSource")
    public SpringLiquibase spravaSpisuLiquibase(
          @Qualifier("spravaSpisuLiquibaseDataSource")
          @Autowired(required = false) DataSource dataSource,
          @Value("${spring.profiles.active:PROD}")
              String activeProfiles
    ) {
      boolean shouldRun = dataSource != null;
      SpringLiquibase springLiquibase = new SpringLiquibase();
      springLiquibase.setShouldRun(shouldRun);
      if (shouldRun) {
          springLiquibase.setDataSource(dataSource);
```

```
        springLiquibase.setContexts(activeProfiles);
    }
    springLiquibase.setChangeLog("classpath:changelog/" +
       "sprava-spisu-db.changelog-master.xml");
    return springLiquibase;
  }
...
```

A.3 Generátor CRUD operací

Obrázek A.2: Spouštění generátoru pro vytvoření Example komponenty

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-rocket-backend
--mappingFile mapping.json --projectPath sprava-subjektu
? path to json file with entity definition Subjekt.json
? root package name cz.cca.hedgehog.subjekt
? scheme name, format (three_word_scheme) registr_subjektu
? URL context root of component rocket/subjekt
```

V definici entity je část `type` bez diakritiky na rozdíl od skutečného kódu.

Listing A.6: Definice entity kniha

```
{
  "atributes": [
    {
      "name":"Nazev",
      "type":"Stredni text",
      "nullable": false
    },
    {
      "name":"Pocet Stran",
      "type":"Cislo",
      "nullable": true
    },
    {
      "name":"Datum Vydani",
      "type":"Datum",
      "nullable": true
    },
```

```
    {
      "name":"Datum a cas vypujceni",
      "type":"Datum a Cas",
      "nullable": true
    },
    {
      "name":"autor",
      "type":"Identifikator",
      "nullable": true
    },
        {
      "name":"datum zakoupeni",
      "type":"Datum",
      "nullable":true
    }
  ],
  "entityName": "Kniha"
}
```

Listing A.7: Definice entity kniha

```
{
  "atributes": [
    {
      "name":"Jmeno",
      "type":"Stredni text",
      "nullable": false
    },
        {
      "name":"Prijmeni",
      "type":"Stredni text",
      "nullable": false
    },
    {
      "name":"Datum a cas registrace",
      "type":"Datum a Cas",
      "nullable": true
    },
        {
```

```
      "name":"Poznamka",
      "type":"Dlouhy text",
      "nullable": false
    }
  ],
  "entityName": "Subjekt"
}
```

Ukázka v části `ea` je bez diakritiky na rozdíl od skutečného kódu.

Listing A.8: Defaultní mapovací soubor

```
{
  "types": [
    {
      "ea": "Kratky text",
      "java": "String",
      "openApi": "string",
      "db": "VARCHAR(50)",
      "length": 50
    },
    {
      "ea": "Stredni text",
      "java": "String",
      "openApi": "string",
      "db": "VARCHAR(255)",
      "length": 255
    },
    {
      "ea": "Dlouhy text",
      "java": "String",
      "openApi": "string",
      "db": "VARCHAR(4000)",
      "length": 4000
    },
    {
      "ea": "Cislo",
      "java": "Integer",
      "openApi": "integer",
      "openApiFormat": "int32",
```

```
      "db": "INTEGER"
    },
    {
      "ea": "Datum",
      "java": "LocalDate",
      "javaImport": "java.time.LocalDate",
      "openApi": "string",
      "openApiFormat": "date",
      "db": "DATE"
    },
    {
      "ea": "Datum a Cas",
      "java": "LocalDateTime",
      "javaImport": "java.time.LocalDateTime",
      "openApi": "string",
      "openApiFormat": "date-time",
      "db": "TIMESTAMP WITHOUT TIME ZONE"
    },
    {
      "ea": "Priznak",
      "java": "Boolean",
      "openApi": "boolean",
      "db": "BOOLEAN"
    },
    {
      "ea": "Identifikator",
      "java": "UUID",
      "openApi": "string",
      "openApiFormat": "uuid",
      "db": "UUID"
    }
  ]
}
```

Listing A.9: Upravený mapovací soubor

```
{
  "types": [
    {
```

```
  "ea": "Kratky text",
  "java": "String",
  "openApi": "string",
  "db": "VARCHAR(50)",
  "length": 50
},
{
  "ea": "Stredni text",
  "java": "String",
  "openApi": "string",
  "db": "VARCHAR(127)",
  "length": 127
},
{
  "ea": "Dlouhy text",
  "java": "String",
  "openApi": "string",
  "db": "VARCHAR(2000)",
  "length": 2000
},
{
  "ea": "Cislo",
  "java": "Integer",
  "openApi": "integer",
  "openApiFormat": "int32",
  "db": "INTEGER"
},
{
  "ea": "Datum",
  "java": "LocalDate",
  "javaImport": "java.time.LocalDate",
  "openApi": "string",
  "openApiFormat": "date",
  "db": "DATE"
},
{
  "ea": "Datum a Cas",
  "java": "OffsetDateTime",
  "javaImport": "java.time.OffsetDateTime",
  "openApi": "string",
```

```
      "openApiFormat": "date-time",
      "db": "TIMESTAMP WITH TIME ZONE"
    },
    {
      "ea": "Priznak",
      "java": "Boolean",
      "openApi": "boolean",
      "db": "BOOLEAN"
    },
    {
      "ea": "Identifikator",
      "java": "UUID",
      "openApi": "string",
      "openApiFormat": "uuid",
      "db": "UUID"
    }
  ]
}
```

Listing A.10: Kniha entita

```
import java.util.UUID;
import java.time.LocalDate;
import java.time.LocalDateTime;

@Entity
@Data
@NoArgsConstructor
@Table(name = "kniha", schema = "knihovna")
public class KnihaEnt extends VersionAwareEntity<UUID> {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "guid", nullable = false)
    private UUID guid;

    @Column(name = "nazev", nullable = false, length = 255)
    private String nazev;
```

```
    @Column(name = "pocet_stran")
    private Integer pocetStran;

    @Column(name = "datum_vydani")
    private LocalDate datumVydani;

    @Column(name = "datum_a_cas_vypujceni")
    private LocalDateTime datumACasVypujceni;

    @Column(name = "autor")
    private UUID autor;

    @Column(name = "datum_zakoupeni")
    private LocalDate datumZakoupeni;
```

Listing A.11: Subjekt entita

```
import java.util.UUID;
import java.time.OffsetDateTime;

@Entity
@Data
@NoArgsConstructor
@Table(name = "subjekt", schema = "registr_subjektu")
public class SubjektEnt extends VersionAwareEntity<UUID> {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "guid", nullable = false)
    private UUID guid;

    @Column(name = "jmeno", nullable = false, length = 127)
    private String jmeno;

    @Column(name = "prijmeni", nullable = false, length = 127)
    private String prijmeni;

    @Column(name = "datum_a_cas_registrace")
    private OffsetDateTime datumACasRegistrace;
```

```
    @Column(name = "poznamka", nullable = false, length = 2000)
    private String poznamka;
```

Popisky endpointů jsou na rozdíl od vygenerovaného kódu zde bez diakritiky.

Listing A.12: Kniha YAML

```
openapi: 3.0.1
info:
  title: Rocket template
  description: Kniha CRUD sluzby
  version: 1.0.0
servers:
  - url: http://localhost/api
tags:
  - name: Kniha

paths:
  /kniha:
    post:
      tags:
        - Kniha
      operationId: postKniha
      summary: zalozi novou Kniha
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/KnihaDetailDto'
      responses:
        200:
          description: ok
          content:
            application/json:
              schema:
                type: string
                format: uuid
```

```
/kniha/{guidKniha}:
  get:
    tags:
      - Kniha
    operationId: getKniha
    summary: vrati detail Kniha
    parameters:
      - in: path
        name: guidKniha
        required: true
        schema:
          type: string
          format: uuid
    responses:
      200:
        description: entita nalezena
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/KnihaDetailDto'
      404:
        description: Kniha nenalezena

  put:
    tags:
      - Kniha
    operationId: putKniha
    summary: aktualizuje Kniha
    parameters:
      - in: path
        name: guidKniha
        required: true
        schema:
          type: string
          format: uuid
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/KnihaDetailDto'
```

```
      responses:
        200:
          description: Kniha aktualizovana
        404:
          description: Kniha nenalezena
        409:
          description: konflikt v guidech nebo verzi

components:
  schemas:
    KnihaDetailDto:
      type: object
      properties:
        guid:
          type: string
          format: uuid
        nazev:
          type: string
          maxLength: 255
        pocetStran:
          type: integer
          format: int32
        datumVydani:
          type: string
          format: date
        datumACasVypujceni:
          type: string
          format: date-time
        autor:
          type: string
          format: uuid
        datumZakoupeni:
          type: string
          format: date
        version:
          type: integer
          format: int32
```

Listing A.13: Subjekt YAML

```
openapi: 3.0.1
info:
  title: Rocket template
  description: Subjekt CRUD sluzby
  version: 1.0.0
servers:
  - url: http://localhost/api
tags:
  - name: Subjekt

paths:
  /subjekt:
    post:
      tags:
        - Subjekt
      operationId: postSubjekt
      summary: zalozi novou Subjekt
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SubjektDetailDto'
      responses:
        200:
          description: ok
          content:
            application/json:
              schema:
                type: string
                format: uuid

  /subjekt/{guidSubjekt}:
    get:
      tags:
        - Subjekt
      operationId: getSubjekt
      summary: vrati detail Subjekt
      parameters:
        - in: path
```

```
      name: guidSubjekt
      required: true
      schema:
        type: string
        format: uuid
  responses:
    200:
      description: entita nalezena
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/SubjektDetailDto'
    404:
      description: Subjekt nenalezena

put:
  tags:
    - Subjekt
  operationId: putSubjekt
  summary: aktualizuje Subjekt
  parameters:
    - in: path
      name: guidSubjekt
      required: true
      schema:
        type: string
        format: uuid
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/SubjektDetailDto'
  responses:
    200:
      description: Subjekt aktualizovana
    404:
      description: Subjekt nenalezena
    409:
      description: konflikt v guidech nebo verzi
```

```
components:
  schemas:
    SubjektDetailDto:
      type: object
      properties:
        guid:
          type: string
          format: uuid
        jmeno:
          type: string
          maxLength: 127
        prijmeni:
          type: string
          maxLength: 127
        datumACasRegistrace:
          type: string
          format: date-time
        poznamka:
          type: string
          maxLength: 2000
        version:
          type: integer
          format: int32
```

Listing A.14: Kniha changeset

```
<changeSet id="init tabulky kniha"
    author="generator-cca-rocket-backend">
    <createTable tableName="kniha" schemaName="knihovna">
        <column name="guid" type="UUID">
            <constraints primaryKey="true"
                primaryKeyName="kniha_pkey"/>
        </column>
        <column name="nazev" type="VARCHAR(255)">
            <constraints nullable="false"/>
        </column>
        <column name="pocet_stran" type="INTEGER"/>
        <column name="datum_vydani" type="DATE"/>
        <column name="datum_a_cas_vypujceni"
```

```
            type="TIMESTAMP WITHOUT TIME ZONE"/>
        <column name="autor" type="UUID"/>
        <column name="datum_zakoupeni" type="DATE"/>
        <column name="datum_vlozeni"
            type="TIMESTAMP WITHOUT TIME ZONE">
            <constraints nullable="false"/>
        </column>
        <column name="vlozil" type="VARCHAR(255)">
            <constraints nullable="false"/>
        </column>
        <column name="datum_aktualizace"
            type="TIMESTAMP WITHOUT TIME ZONE"/>
        <column name="aktualizoval" type="VARCHAR(255)"/>
        <column name="version" type="INTEGER">
            <constraints nullable="false"/>
        </column>
    </createTable>
</changeSet>
```

Listing A.15: Subjekt changeset

```
<changeSet id="init tabulky subjekt"
    author="generator-cca-rocket-backend">
    <createTable tableName="subjekt"
        schemaName="registr_subjektu">
        <column name="guid" type="UUID">
            <constraints primaryKey="true"
                primaryKeyName="subjekt_pkey"/>
        </column>
        <column name="jmeno" type="VARCHAR(127)">
            <constraints nullable="false"/>
        </column>
        <column name="prijmeni" type="VARCHAR(127)">
            <constraints nullable="false"/>
        </column>
        <column name="datum_a_cas_registrace"
            type="TIMESTAMP WITH TIME ZONE"/>
        <column name="poznamka" type="VARCHAR(2000)">
            <constraints nullable="false"/>
```

```
        </column>
        <column name="datum_vlozeni"
            type="TIMESTAMP WITHOUT TIME ZONE">
            <constraints nullable="false"/>
        </column>
        <column name="vlozil" type="VARCHAR(255)">
            <constraints nullable="false"/>
        </column>
        <column name="datum_aktualizace"
            type="TIMESTAMP WITHOUT TIME ZONE"/>
        <column name="aktualizoval" type="VARCHAR(255)"/>
        <column name="version" type="INTEGER">
            <constraints nullable="false"/>
        </column>
    </createTable>
</changeSet>
```

Listing A.16: Služba implementující CRUD operace pro entitu Kniha

```
@Service
@Transactional
public class KnihaServiceImpl implements KnihaApiDelegate {

  @Autowired
  private KnihaRepository knihaRepository;

  @Autowired
  private KnihaMapper knihaMapper;

  @Override
  public ResponseEntity<KnihaDetailDto>
      getKniha(UUID guidKniha)
  {
    KnihaEnt kniha = knihaRepository.findById(guidKniha)
            .orElseThrow(() ->
            new RecordNotFoundException(
                guidKniha, KnihaEnt.class));

    return new ResponseEntity<>(knihaMapper.map(kniha),
        HttpStatus.OK);
  }

  @Override
  public ResponseEntity<Void>
      putKniha(UUID guidKniha,
                  KnihaDetailDto knihaDetailDto) {
    RequestValidator.putSameGuid(guidKniha,
        knihaDetailDto.getGuid());

    KnihaEnt kniha = knihaRepository.findById(
        guidKniha, knihaDetailDto.getVersion())
            .orElseThrow(() ->
            new RecordNotFoundException(
                guidKniha, KnihaEnt.class));

    knihaMapper.map(kniha, knihaDetailDto);

    return new ResponseEntity<>(HttpStatus.OK);
```

```
  }

  @Override
  public ResponseEntity<UUID>
      postKniha(KnihaDetailDto knihaDetailDto)
  {
    KnihaEnt kniha = new KnihaEnt();

    knihaMapper.map(kniha, knihaDetailDto);
    kniha = knihaRepository.save(kniha);

    return new ResponseEntity<>(
        kniha.getGuid(), HttpStatus.CREATED);
  }
}
```

Listing A.17: Služba implementující CRUD operace pro entitu Subjekt

```
@Service
@Transactional
public class SubjektServiceImpl implements SubjektApiDelegate
{

  @Autowired
  private SubjektRepository subjektRepository;

  @Autowired
  private SubjektMapper subjektMapper;

  @Override
  public ResponseEntity<SubjektDetailDto>
         getSubjekt(UUID guidSubjekt) {
    SubjektEnt subjekt = subjektRepository
           .findById(guidSubjekt)
           .orElseThrow(() -> new RecordNotFoundException(
               guidSubjekt, SubjektEnt.class));

    return new ResponseEntity<>(subjektMapper.map(subjekt),
           HttpStatus.OK);
```

```
  }

  @Override
  public ResponseEntity<Void>
         putSubjekt(UUID guidSubjekt, SubjektDetailDto
                subjektDetailDto) {
    RequestValidator.putSameGuid(guidSubjekt,
           subjektDetailDto.getGuid());

    SubjektEnt subjekt = subjektRepository
           .findById(guidSubjekt,
              subjektDetailDto.getVersion())
           .orElseThrow(() -> new RecordNotFoundException(
                  guidSubjekt, SubjektEnt.class));

    subjektMapper.map(subjekt, subjektDetailDto);

    return new ResponseEntity<>(HttpStatus.OK);
  }

  @Override
  public ResponseEntity<UUID>
         postSubjekt(SubjektDetailDto subjektDetailDto) {
    SubjektEnt subjekt = new SubjektEnt();

    subjektMapper.map(subjekt, subjektDetailDto);
    subjekt = subjektRepository.save(subjekt);

    return new ResponseEntity<>(subjekt.getGuid(),
           HttpStatus.CREATED);
  }
}
```

Listing A.18: OpenAPI execution pro endpointy entity knihy

```
<execution>
    <id>kniha</id>
    <goals>
        <goal>generate</goal>
```

```
    </goals>
    <configuration>
        <inputSpec>
            ${project.basedir}/src/main/resources/
            specifikace/api-kniha.yaml
        </inputSpec>
        <configOptions>
            <apiPackage>
              cz.cca.hedgehog.knihovna.modules.kniha.api
            </apiPackage>
            <modelPackage>
              cz.cca.hedgehog.knihovna.modules.kniha.api.dto
            </modelPackage>
            <invokerPackage>
              cz.cca.hedgehog.knihovna.modules.kniha.handler
            </invokerPackage>
            <configPackage>
              cz.cca.hedgehog.knihovna.modules.kniha.config
            </configPackage>
            <packageName>
              cz.cca.hedgehog.knihovna.modules.kniha
            </packageName>
        </configOptions>
    </configuration>
</execution>
```

Listing A.19: Kostry integračních testů pro jednotlivé odpovědi endpointů

```
//todo pridat anotaci testu komponenty
@RunWith(SpringRunner.class)
public class KnihaTest {

    private static final String COMMON_ENDPOINT =
            "rocket/knihovna/api/kniha/";

    @Test
    public void implementUs() {
        Assertions.assertTrue(false);
    }
```

```
//get testy
@Test
public void getKnihaOk() {
    RestAssured.get(COMMON_ENDPOINT)
            .then()
            .statusCode(200);
}

@Test
public void getKnihaNotFound() {
    RestAssured.get(COMMON_ENDPOINT +
        UUID.randomUUID().toString())
            .then()
            .statusCode(404);
}

//put testy
@Test
public void putKnihaOk() {
    String putJson = "{}";
    RestAssured.given()
            .contentType(ContentType.JSON)
            .body(putJson)
            .put(COMMON_ENDPOINT)
            .then()
            .statusCode(200);

    RestAssured.get(COMMON_ENDPOINT)
            .then()
            .statusCode(200)
            .body("guid", CoreMatchers.equalTo(""));
}

@Test
public void putKnihaNotFound() {
    String putJson = "{}";
    RestAssured.given()
            .contentType(ContentType.JSON)
            .body(putJson)
```

```
            .put(COMMON_ENDPOINT)
            .then()
            .statusCode(404);
}

@Test
public void putKnihaVersionConflict() {
    String putJson = "{}";
    RestAssured.given()
            .contentType(ContentType.JSON)
            .body(putJson)
            .put(COMMON_ENDPOINT)
            .then()
            .statusCode(409);
}

@Test
public void putKnihaGuidConflict() {
    String putJson = "{}";
    RestAssured.given()
            .contentType(ContentType.JSON)
            .body(putJson)
            .put(COMMON_ENDPOINT)
            .then()
            .statusCode(409);
}

//post testy
@Test
public void postKnihaOk() {
    String postJson = "{}";
    RestAssured.given()
            .contentType(ContentType.JSON)
            .body(postJson)
            .post(COMMON_ENDPOINT)
            .then()
            .statusCode(200);
}
```

A.4 Generátor gridu

Listing A.20: Vygenerovaný kód se stejnými parametry jako má vzorový zobrazovací grid

```
<hedgehog-grid [data]="dataGrid1"
               gridId="Grid1"
               [take]="take"
               [isDataLoading]="isLoading">
    <kendo-grid-column field="poradi"
                       title="poradi"
                       filter="numeric"
                       [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="nazev"
                       title="nazev"
                       [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="datumZapisu"
                       title="datumZapisu"
                       filter="date"
                       [width]="220"
                       [format]="'date'␣|␣getDateFormat">
    </kendo-grid-column>
    <kendo-grid-column field="datumVymazu"
                       title="datumVymazu"
                       filter="date"
                       [width]="220"
                       [format]="'date'␣|␣getDateFormat">
    </kendo-grid-column>
</hedgehog-grid>
```

Listing A.21: Vzor editačního gridu

```
<form #formGrid2"ngForm">
  <hedgehog-grid [data]="dataGrid2"
                 [isDataLoading]="isLoading"
                 gridId="Grid2"
                 [form]="formGrid2"
```

```
            [isEditable]="true"
            (saveNewRecord)="onSaveNewRecord($event)"
            (updateRecord)="onUpdateRecord($event)">
<kendo-grid-column field="poradi"
                title="poradi"
                filter="numeric"
                [width]="220"
  <ng-template kendoGridEditTemplate
  let-dataItem="dataItem">
    <div fxLayout="column">
      <kendo-numerictextbox hedgehogNumerictextbox
                id="poradiGridGrid2"
                name="poradiGridGrid2"
                class="w-100"
                #poradiGridGrid2Ctrl="ngModel"
                [min]="0"
                [max]="3"
                [(ngModel)]="dataItem.poradi">
      </kendo-numerictextbox>
      <hedgehog-form-error [controller]="poradiGridGrid2"
                [minValue]="0"
                [maxValue]="3">
      </hedgehog-form-error>
    </div>
  </ng-template>
</kendo-grid-column>
<kendo-grid-column field="nazev"
                title="nazev"
                [width]="220"
  <ng-template kendoGridEditTemplate
  let-dataItem="dataItem">
    <div fxLayout="column">
      <input kendoTextBox type="text" class="w-100"
             #nazevGridGrid2Ctrl="ngModel"
             id="nazevGridGrid2" name="nazevGridGrid2"
             [(ngModel)]="dataItem.nazev"
             maxlength="3"
             required/>
      <hedgehog-form-error
                [controller]="nazevGridGrid2Ctrl"
```

```
                    [maxLength]="3">
          </hedgehog-form-error>
        </div>
      </ng-template>
    </kendo-grid-column>
    <kendo-grid-column field="datumZapisu"
                    title="datumZapisu"
                    filter="date"
                    [width]="220"
                    [format]="'date'␣|␣getDateFormat">
      <ng-template kendoGridEditTemplate
      let-dataItem="dataItem">
          <hedgehog-datepicker
                    [datepickerId]="'datumZapisuGridGrid2'"
                    [(date)]="dataItem.datumZapisu"
                    [adjustSize]="true">
          </hedgehog-datepicker>
      </ng-template>
    </kendo-grid-column>
    <kendo-grid-column field="datumVymazu"
                    title="datumVymazu"
                    filter="date"
                    [width]="220"
                    [format]="'date'␣|␣getDateFormat">
      <ng-template kendoGridEditTemplate
      let-dataItem="dataItem">
          <hedgehog-datepicker
                    [datepickerId]="'datumVymazuGridGrid2'"
                    [(date)]="dataItem.datumVymazu"
                    [adjustSize]="true">
          </hedgehog-datepicker>
      </ng-template>
    </kendo-grid-column>
  </hedgehog-grid>
</form>
```

Listing A.22: Vygenerovaný editační grid

```
<form #formGrid2"ngForm">
```

```
<hedgehog-grid [data]="dataGrid2"
               [isDataLoading]="isLoading"
               gridId="Grid2"
               [form]="formGrid2"
               [isEditable]="true"
               (saveNewRecord)="onSaveNewRecord($event)"
               (updateRecord)="onUpdateRecord($event)">
  <kendo-grid-column field="poradi"
                     title="poradi"
                     filter="numeric"
                     [width]="220"
    <ng-template kendoGridEditTemplate
    let-dataItem="dataItem">
      <div fxLayout="column">
        <kendo-numerictextbox hedgehogNumerictextbox
                     id="poradiGridGrid2"
                     name="poradiGridGrid2"
                     class="w-100"
                     #poradiGridGrid2Ctrl="ngModel"
                     [min]="0"
                     [max]="3"
                     [(ngModel)]="dataItem.poradi">
        </kendo-numerictextbox>
        <hedgehog-form-error [controller]="poradiGridGrid2"
                     [minValue]="0"
                     [maxValue]="3">
        </hedgehog-form-error>
      </div>
    </ng-template>
  </kendo-grid-column>
  <kendo-grid-column field="nazev"
                     title="nazev"
                     [width]="220"
    <ng-template kendoGridEditTemplate
    let-dataItem="dataItem">
      <div fxLayout="column">
        <input kendoTextBox type="text" class="w-100"
                #nazevGridGrid2Ctrl="ngModel"
                id="nazevGridGrid2" name="nazevGridGrid2"
                [(ngModel)]="dataItem.nazev"
```

```
                maxlength="3"
                required/>
        <hedgehog-form-error
                    [controller]="nazevGridGrid2Ctrl"
                    [maxLength]="3">
        </hedgehog-form-error>
       </div>
      </ng-template>
    </kendo-grid-column>
    <kendo-grid-column field="datumZapisu"
                    title="datumZapisu"
                    filter="date"
                    [width]="220"
                    [format]="'date'␣|␣getDateFormat">
      <ng-template kendoGridEditTemplate
      let-dataItem="dataItem">
          <hedgehog-datepicker
                    [datepickerId]="'datumZapisuGridGrid2'"
                    [(date)]="dataItem.datumZapisu"
                    [adjustSize]="true">
          </hedgehog-datepicker>
      </ng-template>
    </kendo-grid-column>
    <kendo-grid-column field="datumVymazu"
                    title="datumVymazu"
                    filter="date"
                    [width]="220"
                    [format]="'date'␣|␣getDateFormat">
      <ng-template kendoGridEditTemplate
      let-dataItem="dataItem">
          <hedgehog-datepicker
                    [datepickerId]="'datumVymazuGridGrid2'"
                    [(date)]="dataItem.datumVymazu"
                    [adjustSize]="true">
          </hedgehog-datepicker>
      </ng-template>
    </kendo-grid-column>
  </hedgehog-grid>
</form>
```

Listing A.23: Vygenerovaný editační grid

```
<hedgehog-grid [data]="dataKnihkupectvi"
               gridId="Knihkupectvi"
               [take]="take"
               [isDataLoading]="isLoading">
    <kendo-grid-column field="nazev"
                       title="nazev"
                       [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="datumVydani"
                       title="datumVydani"
                       filter="date"
                       [width]="220"
                       [format]="'date' | getDateFormat">
    </kendo-grid-column>
    <kendo-grid-column field="cena"
                       title="cena"
                       filter="numeric"
                       [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="kDispozici"
                       title="kDispozici"
                       filter="boolean"
                       [width]="220"
    </kendo-grid-column>
</hedgehog-grid>
```

Listing A.24: Editační grid, s needitovatelnými sloupečky

```
<form #formCastecnyEdit"ngForm">
  <hedgehog-grid [data]="dataCastecnyEdit"
                 [isDataLoading]="isLoading"
                 gridId="CastecnyEdit"
                 [form]="formCastecnyEdit"
                 [isEditable]="true"
                 (saveNewRecord)="onSaveNewRecord($event)"
                 (updateRecord)="onUpdateRecord($event)">
    <kendo-grid-column field="nazev"
                       title="nazev"
```

```
                    [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="datumVydani"
                    title="datumVydani"
                    filter="date"
                    [width]="220"
                    [format]="'date' | getDateFormat">
    </kendo-grid-column>
    <kendo-grid-column field="dil"
                    title="dil"
                    filter="numeric"
                    [width]="220"
    </kendo-grid-column>
    <kendo-grid-column field="kDispozici"
                    title="kDispozici"
                    filter="boolean"
                    [width]="220"
      <ng-template kendoGridEditTemplate
      let-dataItem="dataItem">
      </ng-template>
    </kendo-grid-column>
    <kendo-grid-column field="ceskyAutor"
                    title="ceskyAutor"
                    filter="boolean"
                    [width]="220"
    </kendo-grid-column>
  </hedgehog-grid>
</form>
```

Obrázek A.3: Spouštění generátoru pro vytvoření vzorového editačního gridu

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-eisir-grid
? Is the grid editable ?  Yes
? Grid name, format (camelCase) grid2
? Number of colums 4
? 1. column name poradi
? 1. column type numeric
? 1. column is editable Yes
? 2. column name nazev
? 2. column type text
? 2. column is editable Yes
? 3. column name datumZapisu
? 3. column type date
? 3. column is editable Yes
? 4. column name datumVymazu
? 4. column type date
? 4. column is editable Yes
```

Obrázek A.4: Vygenerování zobrazovacího gridu se všemi typy sloupečků

```
D:\Programs\Dropbox\School\bc\testovani>yo cca-eisir-grid --outputFilename Kniha.html
? Is the grid editable ?  No
? Grid name, format (camelCase) Knihkupectvi
? Number of colums 4
? 1. name nazev
? 1. type text
? 2. name datumVydani
? 2. type date
? 3. name cena
? 3. type numeric
? 4. name kDispozici
? 4. type boolean
   create Kniha.html
```

Obrázek A.5: Vygenerování editačního gridu s needitovatelnými sloupečky