FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Informační systém pro podporu chovu šneků
Student:	Mgr. Mikolas Teska
Vedoucí:	Mgr. Petr Matyáš
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

* Seznamte se s existujícími aplikacemi pro podporu chovu zvířat a sepište jejich řešerši.
* Navrhněte webovou informační systém pro podporu chovu šneků.
* Aplikace bude umožňovat:
- správu uživatelů,
- správu chovných skupin šneků i jednotlivců,
- genealogické výstupy pro chované jedince,
- chovatelské statistiky a zobrazení událostí na časové ose,
- export existujících dat.
* Pro navrženou aplikaci implementujte datovou a aplikační vrstvu s vystaveným API pro potřebné služby pomocí Spring Boot a PostgreSQL.
* Veškeré softwarové výstupy důkladně otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 15. února 2021 v Praze.

Bakalářská práce

INFORMAČNÍ SYSTÉM PRO PODPORU CHOVU ŠNEKŮ

Mikoláš Teska

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Mgr. Petr Matyáš
13. května 2021

Odkaz na tuto práci: Mikoláš Teska. *Informační systém pro podporu chovu šneků.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat především panu magistru Petru Matyášovi za jeho pomoc a podporu při tvorbě této bakalářské práce. Velká část vděku také patří mé rodině a přátelům za jejich trpělivost a podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2020 .....

Abstrakt

Tato práce se zabývá tvorbou datového modulu a REST API pro informační systém zaměřený na podporu chovu šneků. V první části práce jsou analyzovány podobné aplikace a následně jsou určeny požadavky na aplikaci a sestaveny případy užití. V další části je zvolena architektura, databáze, je představen datový model a návrh REST API pro komunikaci s uživatelským prostředím. Následuje popis implementace a testování, kde jsou diskutovány zvolené technologie, popsány principy organizace kódu a části implementace. V závěrečné kapitole je zhodnocen výstup bakalářského projektu a diskutován jeho přínos a možné rozšíření.

Klíčová slova informační systém, šnek, REST API, Spring Boot, PostgreSQL

Abstract

This study deals with the creation of a data module and REST API for an information system supporting snail breeding. In its first part, similar applications are analyzed and the application requirements and use cases are determined. In the next part, the architecture and database are chosen, and the data model and the REST API communicating with the user's interface are introduced. In the next two chapters, the implementation and testing are described. Implementation focuses on selected technologies, code organization, and programming logic. The final chapter evaluates the output of the bachelor's project and discusses its benefits and possible expansion.

Keywords information system, snail, REST API, Spring Boot, PostgreSQL

Seznam zkratek

AJAX	Asynchronous JavaScript + XML
API	Application Programming Interface
BSON	Binary JSON
CRUD	Create, Read, Update, and Delete
CSS	Cascading Style Sheets
CSV	Comma-separated value
DTO	Data To Object
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
JAR	Java Archive
JDK	Java Development Kit
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Token
MPA	Multi Page Application
NoSQL	Not only SQL
PDF	Portable Document Format
POJO	Plain Old Java Object
POM	Project Object Model
REST	Representational State Transfer
SHA-1	Secure Hash Algorithm 1
SPA	Single Page Application
SQL	Structured Query LanguageL
URL	Uniform Resource Locator
WAR	Web Application Resource
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Kapitola 1
Úvod

Webové aplikace slouží k podpoře mnohých lidských aktivit. V Čechách i ve světě je velmi populární chov domácích mazlíčků a stoupající popularitě se těší i chov šneků, přesněji nekomerční chov afrických šneků. Mezi nejčastěji chované druhy patří oblovka žravá (archatina fulica), oblovka síťovaná (archatina reticulata) nebo oblovka obrovská (archatina archatina). [1] Tyto druhy šneků jsou jedny z největších zástupců plžů. Největší zaznamenaný jedinec oblovky obrovské měřil na délku 39,3 cm a vážil 900 gramů. [2] Oblovky mají původ v zemích střední Afriky, ale v dnešní době jsou chovány v domácích chovech po celé planetě. Lidé, kteří oblovky chtějí pěstovat, se musí vyvarovat jejich vypouštění do volné přírody, protože jsou považovány za významný invazivní druh s dopadem na zemědělský průmysl. [3] Chov těchto mazlíčků probíhá v chovatelských boxech a je s ním spojeno značné množství aktivit při péči o jedince. Chov šneků tudíž produkuje velké množství informací, které si pěstitelé evidují v různých necentralizovaných formách. Ať už zde mluvíme o sešitech a poznámkách o chovu, nebo o tabulkách aplikace Microsoft Excel a jiných digitálních dokumentech.

Tato bakalářská práce si dává za cíl přispět k lepší organizaci a centralizaci pěstitelských dat, a současně nabídnout pěstitelům platformu, pomocí které budou moci informace o svém chovu mezi sebou sdílet. Takováto platforma by byla, dle mých informací, mezi českými chovateli šneků unikátní.

Pro úspěšnou realizaci cílů této bakalářské práce je podstatné modelování dat a tvorba datové a logické vrstvy aplikace, následováno celkovým procesem tvorby softwarového řešení ve zvoleném frameworku. Vedle těchto technických nároků je důležitá i analýza neznámé domény chovatelství šneků.

Tato práce bude rozdělena do několika tematických celků. V první části Analýza bude představeno několik aplikací s podobnou tématikou, následováno analýzou požadavků a sestavením případů užití. V další kapitole Návrh budou diskutovány možnosti technického řešení a bude představen datový model a struktura REST API rozhraní. Poté již budou představeny jednotlivé body realizace v části Implementace a celé to bude zakončeno závěrem.

Tato práce je koncipována jako dokumentace vývoje a implementace datové vrstvy a REST API pro informační systém pro podporu chovu šneků. Na tuto práci navazuje bakalářská práce kolegy Jakuba Rigociho, který vypracovává frontendovou část aplikace.

Kapitola 2

Analýza

2.1 Průzkum stávajících aplikací

Nejprve jsem se snažil najít existující aplikaci zabývající se přímo chovem šneků. Protože se mi však nepodařilo takto specificky orientovanou webovou aplikaci nalézt, rozhodl jsem se pro přehled chovatelských aplikací. Kromě chovatelských aplikací je do přehledu zařazena také jedna genealogická aplikace orientované na lidi.

2.1.1 GesPet

Aplikace GesPet [1] slouží k evidenci a organizaci pro pěstitele psů. Má velice široké portfolio funkcionalit. Počínaje evidencí psů a jejich křížení, přes doručovací službu pro jedince. Na úvodní obrazovce je kalendář s volbou dne, dále vyhledávací pole pro zvíře a pro zákazníka. Obrazovka rozdělena do několika vertikálních části, kde jsou postupně zobrazeny data pro: doručování psů, blížící se porody, březí fenky, události, úkoly a připomínky. U každé ze sekcí je možnost volby přímé editace položky nebo otevření subsekce pro příslušnou oblast. Na horní liště se nachází menu rozděleno do pěti sekcí (chovatel, nový, ukaž, reporty a grafy, nástroje, nastavení). První menu „chovatel“ umožňovalo přecházet na jednotlivé další moduly (hotel, cvičitel, školka, obchod, kadeřnictví, veterinář). Volba „nový“ umožňuje založení nových položek pro (chovaný jedinec, zákazník, dodavatel, veterinář, šlechtitel, březost, páření, vrh štěňat, plánovaný vrh, rezervace, prodeje, výdaje). Shodné položky jsou i v sekci náhled spolu s několika doplňujícími položkami, mezi které patří: rodokmen, koeficient křížení a plánované doručení. Podobně tomu je u reportů a grafů kde se nachází široký přehled reportu o datech ze zmiňovaných oblastí. V sekci nástroje se nacházeli tyto položky (deník, úkoly, připomínky a varování, dokumenty, zařízení, správa emailů, nápověda). V těchto sekcích je možné procházet, editovat a přidávat jednotlivé položky. V sekci nastavení je možné upravit záznamy o zaměstnancích, plemenech, detailech o společnosti a fakturačních číslech. Také je v této sekci možné nastavit jazyk aplikace, platební metody a správu hesel. Celkově tato aplikace zahrnuje široké pole funkcionalit a nabízí bohatou škálu reportů. Její záběr je širší než plánuji v rámci tohoto projektu, ale i tak je mnoho poznatků zajímavých. Například volba příslušného modulu v rámci menu. Při takovém přístupu by vedle modulu tvořeného touto bakalářskou prací mohly v budoucnu vzniknout i další části aplikace.

[1] www.gespet.com

2.1.2 ZooEasy

U této aplikace [2] je při registraci nutná volba typu chovaných zvířat. Je možné vybírat mezi savci a ptáky. Na úvodní obrazovce je vidět výpis chovaných jedinců. Na horní liště je volba pro: přidáni jedince, pole pro vyhledávání jedinců podle jména a volba pro export, tisk nebo uložení do souboru. Druhý rozměr menu se nachází na levé vertikální liště a je možné volit přehled chovaných jedinců, páry pro šlechtění, kontakty jedinců, výsledky, finance, export dat ve formátu csv, správa, můj účet a data o mém přihlášení. Většina voleb používá stejnou šablonu, kdy nové záznamy se přidávají volbou z horní horizontální lišty a následným vyplněním modální obrazovky, následně stvrzeného volbou pro uložení. Odstranění záznamů probíhá přímo volbou v seznamu a editace je možná po otevření detailu záznamu. Při vkládání nového jedince do systému je možné vyplnit tyto atributy: identifikace, pohlaví, druh, barva, datum narození, datum smrti, otec, matka, chovatel. Pro existující záznamy lze přidávat soubory, obrázky, vkládat poznámky a komentovat identifikované mutace. Pro jedince je možné zobrazit grafický rodokmen a dopočítávají se pěstitelské hodnoty jako: Ancestor-Loss coefficient (koeficient ztráty genetické vazby na předky), Koeficient inbreedingu (koeficient křížení).

Obecně lze říci, že aplikace ZooEasy je starší a má jednodušší formu než již zmiňovaný GesPet. Zároveň však mnohé základní principy jako jsou vložení záznamu, editace, odstranění záznamu jsou velmi podobné jako u GesPet.

2.1.3 Farmbrite

Tato aplikace [3] pokrývá různé aspekty provozu farmy a chov zvířat jen jednou z mnoha funkcionalit. Při přidání zvířete se volí z typických zvířat, která se chovají na farmě. Při přidaní jedince se vyplňují tyto atributy: identifikátor, druh zvířete, interní identifikátor, klíčová slova, status, pěstitelské plemeno, pohlaví, plemeno, barva, datum narození, porodní váha, zakoupeno/vychováno. Genealogická data se přidávala ve zvláštní sekci. Tato aplikace se zaměřuje spíše na finanční a organizační stránku provozu farmy.

2.1.4 Genealone

Poslední aplikace v tomto přehledu Genealone [4] , je zaměřena na rodokmeny lidí. Tato aplikace byla přidána do přehledu z důvodu, že plánovanou součástí tohoto projektu jsou i genealogické výstupy chovaných jedinců. U aplikace Genealone je bezplatně k dispozici pouze demo, ale přes to lze prohlédnout mnoho výstupů: kalendář s daty narození, úmrtí zobrazené k vybranému datu, třídění podle příjmení, třídění podle lokace, hledání příbuznosti dvou osob, zobrazení diagramem genealogického stromu, zobrazení výsledků na časové ose. Bohužel není možné si prohlédnout, jak se data do aplikace nahrávají, ale i tak jsou tyto výstupy velmi podnětné při úvahách nad možnostmi zobrazení dat o příbuznosti jedinců. Při opakované návštěvě URL na začátku května 2021 již nebylo demo přístupné.

2.1.5 Shrnutí

Pro většinu aplikací, které byli v této části analyzovány, platil základní model přidávání jedinců s příslušnými informacemi a následná tvorba vztahu mezi jedinci. Tyto vztahy bylo možné zobrazit nejčastěji v rámci profilu jedinců. Při přidávání nového záznamu se zobrazil seznam atributů, které je nutné nebo možné vyplnit. Nechyběla zde políčka pro volný text a tento přístup byl použit také v tomto projektu a to často použitým atributem „komentář“. Genealogická aplikace

[2] www.zooeasy.com
[3] www.farmbrite.com
[4] www.genealone.com/demo/

se týkala příbuzenských vztahů mezi lidmi kde některé charakteristiky, jako jsou třídění podle lokace nebo příjmení nejsou použitelná u šneků. Otázku možného zaznamenávání příbuznosti mezi šneky je třeba ještě hlouběji analyzovat.

2.2 Analýza požadavků

Jasné stanovení požadavků na produkt je nutné k dosažení úspěšné implementace. V některých případech je programátorovi prezentováno již vypracované zadáním a jeho úkolem je implementace řešení jasně zadaných požadavků. Často je ale tvorba takového zadání procesem, na kterém se podílejí všechny části projektového týmu. V případě tohoto projektu vznikla analýza požadavků na základě komunikace mezi mnou, školitelem a komunitou chovatelů šneků. Požadavky lze členit různými způsoby, zvoleno bylo dělení na funkční a nefunkční požadavky.

2.2.1 Funkční požadavky

Funkční požadavky mají za cíl jednoznačně a konkrétně definovat co bude produkt dělat. Jinými slovy definují, co bude výstupem při definovaném vstupu. Slouží tak k ujasnění cílů mezi všemi skupinami, které se podílejí na vývoji aplikace. Toto ujasnění, je zejména důležité při vývoji softwaru, kde často proběhne značné množství práce dříve, než vznikne první prototyp. Následuje seznam funkčních požadavků na vývoj této aplikace.

F1 – Tvorba a editace záznamu o šnekovi: Aplikace bude umožňovat vytváření záznamů o šnekovi s možností přidání těchto atributů (jméno, barva těla, barva ulity, vzorec ulity, datum narození, datum smrti, datum prodeje, taxonomické určení, původ šneka, zda má být zveřejněn). Dále bude možné připojit soubory. U šneka bude nutné vytvořit vazbu na skupinu. Pokud nový záznam pro šneka vznikne z evidované snůšky, bude přidán záznam o rodné snůšce. Pro existující šneky bude možné editovat vybrané atributy, přidávat záznam o mateřství vzniklé snůšky a vkládat jednotlivá měření (váha a velikost ulity).

F2 – Přidání záznamu o snůšce: Aplikace bude umožňovat přidání záznamu o snůšce, a to mimo jiné s těmito informacemi (datum snesení, počet vajíček, zda je snůška ponechána, perioda líhnutí začátek, perioda líhnutí konec). Pro každou snůšku bude možné přidávat záznamy o selekcích, které budou obsahovat (počet ponechaných vajíček, komentář). Po poslední selekci zůstanou vybrání jedinci pro další chov. Z této skupiny bude možné vytvořit nové záznamy o šnecích s vazbou na snůšku.

F3 – Genealogický profil jedinců: Pro jednotlivé šneky bude dostupný jejich genealogický profil popisující příbuznost v rámci záznamů v aplikaci. Při bližším obeznámení se se situací mezi chovateli se ukázalo, že určení rodičů šneka je velmi komplikované, a to zejména kvůli možnosti chovat více šneků v jednom boxu a schopnosti odloženého oplození. Z těchto důvodů bude pro snůšku evidována pouze chovná skupina, ze které vzešla a matka snůšky. Z těchto důvodů není možné definovat plný genealogický profil jedinců. Tato funkcionalita nebude tedy plně implementována. Pro jedince bude možné identifikovat matku a skupinu, ze které vzešli. Pro vybraného šneka bude možné získat řetěz mateřských generací.

F4 – Přidáni a zobrazení událostí na časové ose pro šneka, box, snůšku, skupinu: Pro jedince bude možné přidat definované události. Definované události bude přidávat uživatel a to k objektu, kterého se týkají. Bude moci přidávat události k boxům, šnekům, skupinám, snůškám. Následně bude možné zobrazit na časové ose definované události.

F5 – Evidování chovatelských boxů: Aplikace bude umožňovat přidávání a správu chovatelských boxů. Při zakládáni a editaci box bude možné nastavit jednotlivé atributy pro zvolený box (jméno, výška, šířka, hloubka, datum pořízení, prodejce). Existující boxy mohou být vázány na jednotlivé chovné skupiny.

F6 – Tvorba chovných skupin: Uživatel bude muset přiřazovat šneky do chovných skupin. Těmto skupinám pak bude třeba přiřadit specifický box. Všichni šneci chovaní v boxu musí patřit do právě jedné skupiny. Skupina muže být tvořena jedním nebo více šneky.

F7 – Přesun šneka do jiné chovné skupiny: Šneka bude možné přesunout do jiné chovné skupiny. Při inaktivaci šneka bude jeho příslušnost k aktuální chovné skupině ukončena a šnek nebude aktivní součástí žádné chovné skupiny. Tento stav nastává při úhynu nebo předání chovateli, který si nepřeje šneka evidovat v tomto informačním systému.

F8 – Export všech záznamů příslušného uživatele: Uživatel bude schopen z aplikace exportovat svoje chovatelská data.

F9 – Administrace: Uživatel s příslušnými právy bude moci spravovat profily uživatelů. Bude aktivovat uživatele po registraci. Bude moci vidět data všech uživatelů a editovat jejich záznamy. Bude schopen editovat taxonomické záznamy a typy událostí pro šneka, box, skupinu a snůšku.

2.2.2 Nefunkční požadavky

Nefunkční požadavky definují systémové požadavky a omezení. Stejně jako v případě funkčních požadavků i pro nefunkční požadavky existuje různá členění. Pro tuto práci popíši uživatelnost, bezpečnost, rozšiřitelnost, udržitelnost.

N1 – Uživatelnost: Modul bude dostupný z internetu prostřednictvím volání REST API URL. Modul je navrhován pro následné propojení s uživatelským prostředím. Tento modul poskytuje rozhraní pro možné implementace uživatelského prostředí, kde implementace uživatelského modulu je s tímto modulem vázaná pouze komunikací přes REST API.

N2 – Bezpečnost: Nedílnou součástí dnešních aplikací je bezpečnost, a to zejména pokud jsou dostupné z internetu. Uživatelé se budou autentizovat pomocí jména a hesla. Po přihlášení bude zabezpečení jednotlivých REST volání zprostředkována pomocí přiděleného JWT tokenu který bude uložen na straně klienta.

N3 – Rozšiřitelnost: Modul bude navržen takovým způsobem, že bude umožňovat budoucí implementaci dalších funkcionalit. Případné přidání dalších modulů bude možné. Komunikace pomoci REST API umožňuje volné spojení.

N4 – Udržitelnost: Modul bude implementován pomocí frameworku Spring Boot. Pro datovou část byla zvolena databáze PostgreSQL. Oba tyto systémy jsou všeobecně známy a používány což přispívá k budoucí udržitelnosti a případným modifikacím pro návaznost na další technologie.

2.3 Přehled uživatelů

V této části bude prezentován přehled typů uživatelů, kteří budou definováni v tomto informačním systému. Pro každý typ uživatele budou nadefinována práva.

2.3.1 Administrátor

Tento typ uživatele bude mít nejvyšší práva. Bude moci vidět a editovat veškerá data. Přidávat a odebírat uživatele. Bude schopen aktivovat uživatele. Editovat společné datové struktury, kterými jsou taxonomické rozdělení a typy událostí.

2.3.2 User

Právo vkládat, editovat a zobrazit svoje data. Právo zobrazit data zpřístupněná od jiných chovatelů. Právo exportovat svoje data.

2.4 Případy užití

Případy užití, jako součást vývoje softwarového řešení, slouží jako nástroj pro podrobnější identifikaci a určení požadavků na systém. Většinou zahrnují tři prvky: uživatele, cíl a systém. Pomocí diagramu nebo slovního popisu jsou vyobrazeny jednotlivé funkční požadavky z pohledu uživatele. Mezi výhody, které tato metoda přináší, patří: přehled o rozsahu systému, přispění k učení systémových požadavků a jejich vhodná forma pro komunikaci mezi členy týmu. [4]

Ačkoliv je tento projekt zaměřen na datový modul, analýza případů užití je důležitá pro implementaci logiky práce s požadavky přicházejícími ze strany uživatele. Případy užití mají zásadní vliv na návrhu REST API pro komunikaci s uživatelským modulem.

2.4.1 Výčet případů užití

- **UC1:** Zobrazení seznamu jedinců pro daného chovatele s možností zobrazení detailu pro vybraného jedince Jedna ze základních funkcionalit, přehledně zobrazující seznam šneků.

  Scénář:

  1. Uživatel si vybere sekci pro seznam šneků.
  2. Následně se zobrazí seznam šneků. Po kliknutí na jméno vybraného šneka se zobrazí obrazovka se souhrnem informací o zvoleném šnekovi.

- **UC2:** Přidání jedince do aplikace

  Scénář:

  1. Uživatel si vybere sekci pro seznam šneků, kde následně pomocí tlačítka „přidej“ jedince”zobrazí formulář pro přidání nového jedince.
  2. V zobrazeném formuláři musí vyplnit povinná pole a případně také nepovinná pole. Lze vytvářet vazbu na snůšku, mateřskou vazbu na snůšku, vazbu na existující chovnou skupinu. Pokud není vybrána chovná skupina, je při přidání jedince vynuceno i založení nové chovné skupiny pro jedince.

  Alternativní scénář:

  1. Nového jedince půjde také vytvořit v detailu snůšky, pokud nebyly vyčerpány všichni jedinci v rámci snůšky, kteří byli selektováni.

- **UC3:** Editace jedince v aplikaci

  Pro jedince evidované v aplikaci bude možné přidávat záznamy a upravovat informace.

  Scénář:

  1. Uživatel si vybere ze seznamu jedinců, na které má práva a po kliknutí na označení vybraného šneka se zobrazí stránka se souhrnem informací o jedinci.
  2. V obrazovce pro detail bude umožněno editovat přípustné atributy pro jedince.

- **UC4:** Smazání jedince z aplikace

  Slouží pro odstranění záznamu o jedinci z aplikace včetně vizuálních materiálů.

  Scénář:

  1. Uživatel si zobrazí seznam jedinců.
  2. Po kliknutí na ikonu „odstranit“, se objeví modální obrazovka s potvrzením volby „odstranění vybraného jedince“. Tato volba bude dostupná také z obrazovky pro detail o jedinci.

- **UC5:** Zobrazení seznamu snůšek a detailu snůšky

  Scénář:

  1. Uživatel si vybere v menu volbu pro správu snůšek.
  2. V následující obrazovce se zobrazí seznam snůšek. Volbou odkazu na označené snůšce se zobrazí detail snůšky.

- **UC6:** Přidání chovné snůšky

  Uživatel přidá chovnou snůšku do systému.

  Scénář:

  1. Uživatel si zvolí volbu pro správu snůšek.
  2. V podnabídce zvolí „přidat snůšku“ a následně vyplnění informace o nové snůšce. Vyplněný formulář odešle k uložení.
  3. Nová snůška se objeví v seznamu snůšek.

- **UC7:** Editace chovné snůšky

  Uživatel upraví atributy chovné snůšky.

  Scénář:

  1. Uživatel si po volbě pro zobrazení seznamu snůšek klikne na vybranou snůšku, a po zobrazení detailů o snůšce bude možné editovat povolené atributy.
  2. Po editaci či přidání zvolených atributů zvolí uživatel volbu pro uložení.

- **UC8:** Odstranění chovné snůšky

  Uživatel odstraní vybranou snůšku. Při volbě pro odstranění bude validováno, zda snůška nemá existující vazbu na šneka. Pokud, existují šneci vzešlí z vybrané snůšky, odstranění nebude povoleno.

Scénář:

1. Uživatel pro vybranou snůšku ze seznamu zvolí volbu „odstranit“.
2. Následně v zobrazeném modálním okně svou volbu potvrdí.

Alternativní scénář:

1. Snůšku lze smazat i volbou v obrazovce pro zobrazení detailů o snůšce.

- **UC9:** Přidání chovné skupiny

  Uživatel vytvoří chovnou skupinu a přiřadí do ní příslušné šneky.

  Scénář:

  1. Uživatel zvolí volbu pro správu chovných skupin.
  2. Zde zvolí vytvořit chovnou skupinu.
  3. Ve formuláři vyplní atributy a přiřadí jedince a zvolí chovný box.

- **UC10:** Rozdělení chovné skupiny

  Uživatel rozdělí chovnou skupinu a vytvoří tak novou chovnou skupinu, do které budou přemístěni vybraní šneci z původní chovné skupiny.

  Scénář:

  1. Uživatel si ze seznamu chovných skupin zobrazí detail vybrané skupiny.
  2. Zvolí volbu „rozdělit“.
  3. Ve formuláři vyplní atributy a zvolí jedince pro přeřazení do nové chovné skupiny.
  4. Na závěr celou operaci potvrdí.

- **UC11:** Editace chovné skupiny

  Vedle rozdělení chovné skupiny bude možné editovat i další parametry. Uživatel bude moci změnit box pro chovnou skupinu odebrat šneka z chovné skupiny nebo přemístit šneka mezi chovnými skupinami.

  Scénář:

  1. Uživatel si zobrazí detail vybrané chovné skupiny ze seznamu chovných skupin.
  2. Zde zvolí „editovat“.
  3. Upraví atributy chovné skupiny.
  4. Potvrdí změny volbou „uložit“.

- **UC12:** Odstranění chovné skupiny

  Uživatel bude moci odstranit chovnou skupinu. Tato akce bude omezena pravidlem, že taková chovná skupina nesměla nikdy mít žádnou vazbu na jedince nebo snůšku.

  Scénář:

  1. Uživatel si zvolí zobrazení detailu vybrané chovné skupiny.

2. Po volbě „odstranit“ se zobrazí modální okno s dotazem na potvrzení akce.
3. Potvrzením volby je chovná skupina odstraněna.

- **UC13:** Přidání chovného boxu

  Scénář:

  1. Uživatel v navigaci vybere sekci pro správu chovných boxů, kde zvolí volbu pro přidání nového chovného boxu.
  2. Po vyplnění atributů ve formuláři volbu potvrdí tlačítkem „uložit“.

- **UC14:** Editace chovného boxu

  Uživatel si vybere box ze seznamu chovných boxů a atributy vybraného boxu může upravit. Volbu potvrdí tlačítkem „uložit“.

  Scénář:

  1. Uživatel si v sekci pro správu chovných boxů vybere ze seznamu box určený pro editaci.
  2. Následně je zobrazena obrazovka s detaily pro daný box.
  3. Vybraná pole doplní nebo upraví.
  4. Volbu potvrdí tlačítkem „uložit“.

- **UC15:** Zobrazení genealogických dat

  Uživatel si zvolí jedince ze seznamu svých jedinců a zobrazí genealogická data pro vybraného jedince.

  Scénář:

  1. Uživatel si v sekci pro správu jedinců vybere jedince.
  2. Volbou jedince se zobrazí obrazovka s detaily o jedincovi.
  3. Volbou „genealogický profil“ se zobrazí mateřská příbuznost pro vybraného jedince.

- **UC16:** Zobrazení časové osy pro chovný box

  Uživatel zobrazí časovou osu s událostmi pro vybraný chovný box.

  Scénář:

  1. Uživatel si vybere box ze seznamu.
  2. Následně otevře obrazovku pro detail vybraného boxu.
  3. Zvolí volbu pro zobrazení časové osy boxu.

- **UC17:** Zobrazení časové osy pro jedince

  Uživatel zobrazí časovou osu pro vybraného jedince

  Scénář:

  1. Uživatel si vybere jedince ze seznamu.
  2. Následně otevře obrazovku pro detail vybraného jedince.
  3. Zvolí volbu pro zobrazení časové osy jedince.

- **UC18:** Export kompletních záznamů příslušného uživatele

  Scénář:

  1. Uživatel si vybere volbu pro export.
  2. Po určení adresáře pro uložení jsou kompletní záznamy exportovány do textového souboru ve formátu JSON.

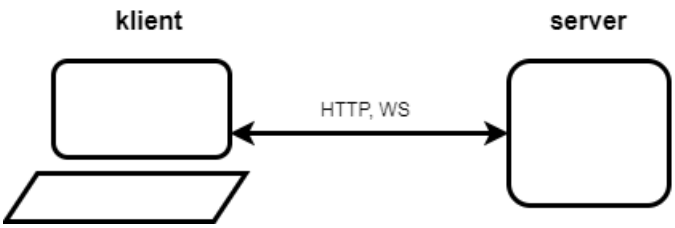
Kapitola 3
Návrh

Tato kapitola se zabývá návrhem architektury vytvářené aplikace. Návrh architektury je zásadní volbou v procesu vývoje softwaru. Důležité je zvolit takovou kombinaci architektury a technologií, která umožní optimální realizaci požadavků na aplikaci a zároveň umožní optimální vývoj z hlediska obtížnosti a realizovatelnosti. Tento návrh bude vycházet z kapitoly 2 Analýza. V první části budou představeny vybrané možnosti architektury. Vzájemně budou porovnány a nejvhodnější z nich bude zvolena spolu s vysvětlením volby. V další části bude představen datový model a návrh REST API.

3.1 Volba architektury pro datový model

Webové aplikace představují specifický typ aplikací, kde lze systém rozdělit na dvě části:

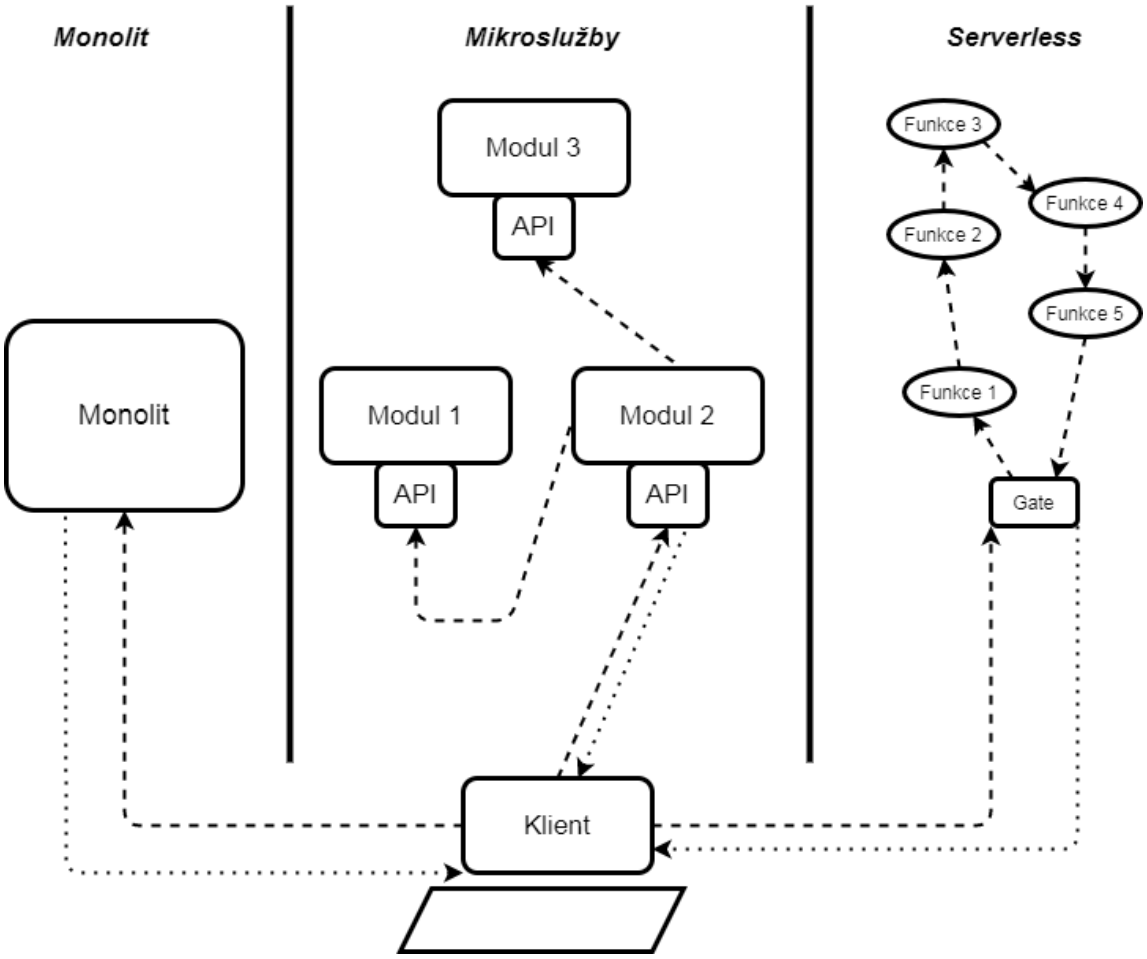
1. **Program běžící na straně klienta:** Tento program běží na počítači uživatele a reaguje na podněty ze strany klienta.
2. **Program běžící na straně serveru:** Tento program běží na počítačích vzdálených od uživatele. S programem na straně klienta komunikuje pomoci http požadavků [5] nebo takzvaného „websocket“ protokolu [6]. Na požadavky ze strany klienta je zasílán kód pro interpretaci na straně klienta. Velmi často jsou na straně serveru spravována data příslušné webové aplikace.



Obrázek 3.1 Vyobrazení architektury klient - server

Na základě tohoto obecného rozložení odpovědností za chod systém vzniká několik architektonických návrhových typů. Každý z těchto typů specificky pokrývá nároky pro danou webovou aplikaci. Každý z těchto návrhových typů má své výhody i nevýhody. V následující části budou popsány a diskutovány jednotlivé typy.

První dělení při volbě architektury proběhne na úrovni vazeb. Pro konstrukci logiky aplikace a vazby této logiky na databázový systém je možné volit mezi několika přístupy



Obrázek 3.2 Architektura podle rozložení komponent

3.1.1 Monolitické aplikace

První z těchto přístupů se nazývá monolit a jedná se o architekturu, kdy logiku aplikace tvoří jedna konsolidovaná aplikace, která běží na serveru. Tato aplikace je schopna vykonat všechny požadavky ze strany klienta. Tento přístup je původní a dříve tvořil naprostou většinu u webových aplikací [7].

Mezi výhody tohoto přístupu patří:

1. Jednodušší vývoj. Celou aplikaci lze vyvíjet v rámci jednoho projektu v jednom repositáři.
2. Jednodušší nasazení na aplikační server. Pro nasazení je třeba nakonfigurovat a spustit pouze tuto jednu aplikaci. Při chybách je jednodušší identifikace původu chyb.
3. Jednodušší hledání chyb a omylů při vývoji stejně tak jako testování.
4. Jednodušší nastavení logování, cachování a transakcí oproti složitějším systémům. Všechny restrikce se řeší v rámci jedné aplikace.

Mezi nevýhody patří:

1. Složitější na udržení přehlednosti a čitelnosti. Složitější monolitické aplikace mohou nabýt až příliš velkého rozsahu, což vede ke zvýšeným nárokům na údržbu.

2. Obtížnost implementace změn. Zavedení složitějších změn do rozsáhlých aplikací může znamenat velice obtížný a časově náročný úkol.

3. Škálovatelnost je možná pouze na celé aplikaci a není umožněna pro jednotlivé komponenty.

4. Složitost modifikací při přechodu na nové technologie. Taková změna často znamená přepsání celé aplikace.

3.1.2 Mikroslužby využívající aplikace

Modernějším přístupem jsou takzvané mikroslužby „microservices“. Architekturou mikroslužeb se rozumí rozdělení funkcionalit aplikace do jednotlivých modulů, které mezi sebou sdílejí data. To je umožněno pomocí specifických komunikačních kanálů, jakými bývají REST API, websocket, message brokers, případně jiné druhy komunikace mezi systémy. Takovéto členění aplikace na jednotlivé moduly je například velmi praktické, pokud vyvíjená aplikace má sloužit jak pro mobilní zařízení, tak pro standardní webové prohlížeče, nebo při komunikaci s dalšími aplikacemi [8].

Mezi výhody tohoto přístupu patří:

1. Rozsah každého z modulů v rámci architektury mikroslužeb bývá menší než u monolitických aplikací. Takový kód bývá přehlednější a lehčí na údržbu.

2. Umožňuje postupné nasazení po částech. To může být výhodou zejména u rozsáhlých projektů.

3. Stabilita systému bývá lepší než u monolitických aplikací. Pokud dojde k problému u určitého počtu modulů, částečná dostupnost služeb může být v určitých situacích zachována.

4. Větší možnosti v oblasti volby technologie. Možnost implementace jednotlivých modulů pomocí různých technologií na základě funkčních a nefunkčních požadavků.

5. Více specifická škálovatelnost aplikace. Je možné rozšířit počet jen u vybraných modulů.

Mezi nevýhody patří:

1. Vývoj distribuovaných aplikací klade na vývojáře vyšší nároky při organizaci a návrhu řešení. Výše zmiňovaná komplexnost řešení v rámci jedné aplikace může být nahrazena komplexností interakcí mezi moduly. Například tvorba požadavku přes více modulů bývá obtížnější než tvorba přímého požadavku.

2. Testování aplikace může být obtížné. Tato komplexita testování může být následkem vazeb mezi moduly.

3. Nasazení takovéto aplikace bývá složitější oproti monolitické aplikaci.

4. Větší požadavky na hardware. Každý z modulů běží v samostatném prostředí.

3.1.3 Serverless aplikace

Rád bych také zmínil poslední přístup. S rozvojem cloudových služeb je častěji využívána možnost takzvané „serverless“ architektury. Lze přeložit jako architektura bez serveru. Různí zprostředkovatelé cloudových služeb nabízejí možnost nasazení samotného kódu aplikace do funkcí, jejichž běh bude obstarávat cloudová služba. Takovéto funkce se spojují mezi sebou a s dalšími službami v rámci prostředí cloudu a jako celek pak tvoří serverless aplikaci. [9]

Mezi výhody tohoto přístupu patří:

1. Odpadá nutnost konfigurace a udržování serverů a případně propojení mezi servery.
2. Nejoptimálnější škálovatelnost. Je možné škálovat jednotlivé funkcionality.
3. Nižší náklady na samotné prostředí oproti udržování serverů.

Mezi nevýhody patří:

1. Může docházet k takzvaným studeným startům, následkem čehož vzniká zpoždění v odezvě systému.
2. Obtížná přenositelnost aplikace, a to jak na odlišnou technologii tak na odlišné cloudové prostředí.
3. Možnosti sledování toku informace mohou být omezenější.

Pro tento bakalářský projekt byla zvolena architektura blízká přístupu mikroslužeb. Přesto, že datový model tvoří pouze jediný modul, návrh se drží charakteristiky architektury mikroslužeb. Výsledkem je tedy datový modul poskytující služby přes vystavené REST API. Druhým modulem je modul uživatelského prostředí. Jedním z důvodů této volby, byl oddělený vývoj datového modulu a modulu uživatelského prostředí. Dalším důvodem byla možnost rozšiřitelnosti do budoucna.

3.2 Volba architektury modulu pro uživatelské prostředí

Přesto, že tato bakalářská práce primárně popisuje tvorbu datového modulu aplikace, bylo by vhodné zdůvodnit i volbu modulu pro interakci s uživatelem, a to z důvodu vazby obou modulů. Uživatel bude pro přístup k aplikaci používat standardní webové prohlížeče, které překládají HTML a CSS texty. HTML je značkovací jazyk definující strukturu webové stránky a CSS popisuje prezentaci webové stránky. [10] Obecně se při implementaci uživatelského rozhraní volí mezi dvěma přístupy. Single Page Aplication zkráceně SPA nebo Multi Page Aplication, zkráceně MPA. Dále bude popsán každý z přístupů.

3.2.1 Vícestránkové aplikace (Multi Page Application)

Tento způsob komunikace mezi klientem a serverem je tradičnější a původnější. Při tomto přístupu je ze strany klienta zaslán požadavek na server. Tento požadavek je zpracován na základě logiky aplikace. Následně je podle typu požadavku vygenerován kód pro zobrazení stránky spolu s příslušnými hodnotami. Tento celek je poté odeslán na stranu klienta kde dojde k zobrazení v příslušném okně. Tento přístup byl během času optimalizován a to i díky AJAX. AJAX je sada funkcionalit, umožňující rozvolnění návaznosti mezi interakcí klienta s uživatelským prostředím a uživatelského prostředí se serverem. [11] V dnešní době je i díky tomu možné ze serveru posílat pouze ty části webové stránky, kterých se aktualizace týká. [12]

3.2.2 Jednostránkové aplikace (Single Page Aplication)

Historii vývoje webových aplikací po celou dobu provází snaha přiblížit celkový dojem z webových aplikací nativním aplikacím v počítači. Pomocí SPA se lze tomuto cíli velmi přiblížit. Hlavní myšlenkou je, že celé uživatelské prostředí tvoří jedna stránka, která se může na základě požadavků a akcí od klienta měnit. Výhodou tohoto přístupu je, že prezentace uživatelského prostředí je pevně oddělena od logiky aplikace na serveru. Další výhodou je, že komunikace se serverem může být orientována pouze na data, zatímco o prezentaci se může plně starat systém na straně klienta. [13] Frontendový modul, který bude vypracováván jiným studentem a je plánován jako SPA.

3.3 Třívrstvá architektura

Jednou z často používaných strategií při tvorbě aplikací je rozdělení funkcionalit do vrstev, mezi kterými se předávají informace od uživatele až po systém pro ukládání dat. Takovým aplikacím se také říká aplikace pomocí n-vrstvé architektury. Výhodou tohoto přístupu je větší přehlednost organizace kódu a zejména nezávislost implementací jednotlivých vrstev jedné na druhé. Například datová vrstva je často vytvořena tak, aby nezávisela na logické vrstvě. Takto, pokud dojde ke změněně databáze, nebo přechodu na jiný systém pro ukládání dat, není třeba měnit implementaci logické vrstvy ale pouze datové vrstvy. [14] Důležitou charakteristikou tohoto konceptu je jeho lineárnost. Tím je myšleno, že data vždy musí procházet stejným pořadím vrstev a nesmí docházet k přeskočení. V tomto bakalářském projektu budou implementovány dvě vrstvy: logická vrstva a datová vrstva. Prezentační vrstva bude implementována jiným studentem.

3.4 Framework pro vývoj datového modulu

Framework pro tvorbu počítačových softwarů ulehčuje programátorovi práci. Umožňuje, zaměřit se pouze na provozní logiku aplikace. Často se stará o bazální funkcionality. Framework není to samé jako knihovna. Například oproti knihovnám, které jsou volány autorem v kódu, framework volá autorský kód. Tomuto principu se říká „inversion of controll“, inverze kontroly. [15] Zvolenému frameworku Spring Boot je věnována sekce v kapitole čtyři 4.3.

3.5 Systém řízení báze dat

Důležitým krokem při návrhu webové aplikace, která má uchovávat uživatelská data je volba systému řízení báze dat. Systému řízení báze dat je někdy také zjednodušeně označován jako databáze. Během vývoje informačních technologií se vytvořili mnohé různorodé způsoby uchování a organizace dat. Lze obecně říci, že tyto systémy se dělí do dvou skupin: Relační databáze a NoSQL databáze.

3.5.1 NoSQL databáze

NoSQL databáze tvoří různorodou skupinu databázových technologií zahrnujících mimo jiné sloupcové databáze, grafové databáze, objektové databáze, databáze klíč-hodnota a asi nejrozšířenější dokumentové databáze. [16] Rád bych zde zmínil princip alespoň dokumentových databází, neboť jsou v dnešní době pro webové aplikace velmi používané. U toho typu databáze jsou data uložena jako dokumenty ve formátu JSON, BSON, XML, YAML nebo v binárním formátu jako třeba PDF. Přístup k jednotlivým dokumentům je definován pomocí unikátního klíče. Tento stejný princip se uplatňuje u databází typu klíč-hodnota, ale u dokumentových databází navíc každý dokument obsahuje interpretační metadata. U dokumentových databází bývají

k dispozici API nebo dotazovací jazyk, které umožňují výběr informací na základě metadat, které obsahují. [17]

3.5.2 Relační databáze

Pravděpodobně nejrozšířenějším typem systému řízení báze dat jsou relační databáze. Poprvé byl tento systém teoreticky popsán v roce 1970 E. F. Coddem. [18] Relační databáze jsou založeny na relačním modelu aparátu relačních množin a predikátové logice. Oproti teoretickému modelu mají navíc nástroj nazvaný SQL schéma. Toto schéma popisuje jednotlivé tabulky. [19] Relační databáze slouží k uložení, načítání, úpravě, mazání, zabezpečení a zajištění integrity dat. Data jsou ukládána v řádcích tabulek. Tyto tabulky mají definovány sloupce s označením a datovým typem ukládaným v příslušném sloupci. Většinou je pro každou tabulku určen jeden nebo více sloupců, které tvoří takzvaný primární klíč. Kombinace hodnot sloupců primárního klíče v každém řádku musí být unikátní. Největší výhodou relačních databází je normalizace dat a vytváření vztahů mezi tabulkami podle vztahů, které existují mezi daty. Tyto vztahy jsou definovány pomocí takzvaných cizích klíčů, které definují vztah mezi dvěma řádky v rámci databáze. Následně lze přes takzvané spojení „JOIN“ tabulky propojovat a takto vyjadřovat vztahy mezi daty pro každou položku. Existuje několik systémů řízení báze dat pro SQL databáze a patří mezi ně MySQL, Oracle, PostgreSQL.

3.5.3 Volba systému řízení báze dat

Pro tento projekt byl vybrán SQL systém řízení báze dat PostgreSQL. Data, které budou chovatele šneků ukládat, mají systémovou provázanost a to lze dobře modelovat pomocí SQL databáze. PostgreSQL bylo zvoleno pro vysoký výkon tohoto systému, pro jeho podporu v rámci licence open source a pro jeho dostupnost.

3.6 Datový model

Důležitou fázi tvorby každé aplikace, která používá databázi pro uchovávání dat, je tvorba datového modelu. Při tvorbě datového modelu se vycházelo z funkčních požadavků a případů užití diskutovaných v kapitole 2 Analýza. Při tvorbě datového modelu bylo třeba dbát pravidel normalizace. [20] Ilustrace databázového modelu je na obrázku 3.3.

snek: Tato tabulka ukládá informace o šnecích. Každý šnek má seznam základních atributů, které popisují reálného jedince. Těmito základními atributy jsou: komentář, jméno, barva těla, barva ulity, vzorec ulity, datum narození, datum smrti, zda byl šnek prodán, komentář o původu šneka a zda může být zveřejněn. Vedle těchto samo vysvětlujících atributů má šnek vazby na dvě další tabulky. První vazbou je cizí klíč na tabulku `taxonomy`. Touto vazbou je určeno přesné taxonomické určení jedince. Dále má cizí klíč na tabulku `snuska`. Tato vazba existuje, pokud šnek pochází ze snůšky evidované v systému. Vedle této možnosti může šnek také přicházet od chovatele, který není registrován v systému. Potom bude tento cizí klíč prázdný. Poslední cizí klíč ukazuje na tabulku `galerie`, pomocí které je tvořena vazba na uložené fotky.

mereni_snek: Tato tabulka uchovává měření pro jednotlivé šneky. Obsahuje tyto atributy: komentář, váha šneka, rozměr ulity a datum měření. Cizí klíč určuje, ke kterému šnekovi měření patří.

skupina: Tabulka skupina zaznamenává tyto základní atributy: jméno a komentář. Tabulka `skupina` slouží k ukládání informací o entitě, která určuje množinu šneků, kteří jsou v systému evidováni jako skupina. Každý šnek musí patřit do právě jedné skupiny. Šneci mohou v čase

měnit skupiny, do kterých patří, a tak je vztah mezi tabulkami `snek` a `skupina` m:n. Pro popis tohoto vztahu slouží tabulka `snek_skupina`.

box: Tabulka `box` popisuje chovatelské boxy, ve kterých chovatelé šneky chovají. Základní atributy ukládané v tabulce `box` jsou komentář, jméno, výška, šířka, hloubka, datum pořízení, prodejce. Tabulka `box` má zásadní vazbu na tabulku `users`. Touto vazbou jsou určeni majitelé všech dalších dat v databázi, protože většina tabulek má přímou nebo zprostředkovanou vazbu na tabulku `box`. Tuto vazbu má i tabulka `skupina`, a to analogicky ke vztahu mezi tabulkami `snek` a `skupina`, pomocí vazby m:n mapované tabulkou `skupina_box`.

snuska: Tato tabulka popisuje jednotlivé snůšky, které mají chovaní šneci. Snůška je soubor nakladených vajíček. Základní atributy ukládané do této tabulky jsou komentář, datum snesení, značka zda byla ponechána, perioda líhnutí začátek, perioda líhnutí konec. Tabulka má dva cizí klíče. První ukazuje na tabulku `skupina` a určuje skupinu, ze které snůška vzešla. Druhý cizí klíč ukazuje na tabulku `snek` a určuje, matku snůšky, pokud je tento údaj zadán.

selekce: V této tabulce se ukládají jednotlivé selekce z příslušné snůšky. Selekce je volba množiny jedinců, kteří budou zachování. Zbytek je zamrazen, čímž je usmrcen. Tabulka ukládá tyto informace: komentář, datum, velikost a cizí klíč ukazující na příslušnou snůšku. Velikost je počet vajíček po selekci.

událost: Tato tabulka uchovává uživatelem definované události, týkající se chovu. Každá ze zaznamenaných událostí musí patřit k právě jednomu z těchto objektů: šnek, skupina, snůška, box. Tento vztah je vyjádřen 4 cizími klíči, z nichž právě jeden má hodnotu ukazující na přidružený objekt. Dále událost obsahuje komentář, datum a cizí klíč odkazující na typ události v tabulce `udalost_typ`.

událost_typ: V této tabulce se uchovávají typy událostí. Má dva atributy: popis a typ události.

taxonomy: Tato tabulka představuje taxonomické členění vybraných druhů šneků. Pro vyjádření vztahů v rámci této živočišné říše slouží rekurzivní odkaz `fk_taxonomy_taxonomy` na vlastní primární klíč. Základními atributy jsou jméno, popis a level.

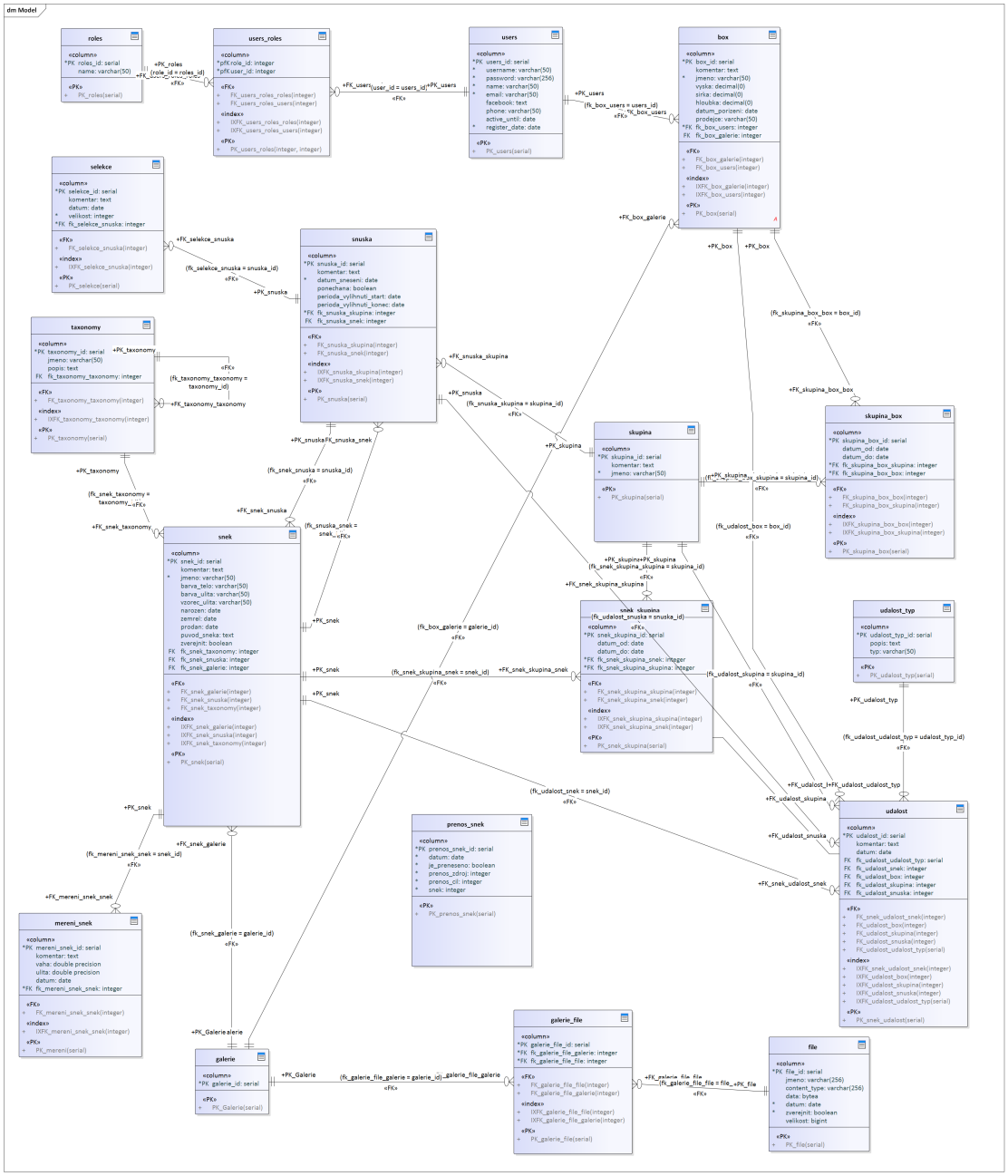
users: V této tabulce jsou uchovávány informace o registrovaných uživatelích. Obsahuje tyto atributy: uživatelské jméno, heslo, jméno, email, facebook, telefon, datum do kdy je aktivní a datum registrace.

roles: Tato tabulka určuje uživatelské role. Na tabulku `users` má vazbu m:n zprostředkovanou tabulkou `users_roles`.

galerie: Tato tabulka ukládá galerie pro objekty šnek a box.

file: Tato tabulka obsahuje nahrané fotografie se jménem, komentářem a mime typem. Na tabulku `galerie` má vazbu m:n zprostředkovanou tabulkou `galerie_file`.

prenos_snek: Tato tabulka obsahuje záznamy o předání šneků v rámci této aplikace. Záznamy v této tabulce ukazují, kdo šneka nabízí, kdo případně přijímá a jestli došlo k předání.



Obrázek 3.3 Datový model

3.7 REST API

Komunikace mezi uživatelským prostředím a datovým modulem bude zprostředkována pomocí REST API. API je způsob, kterým mezi sebou komunikují programy, a to pomocí přesně definovaných pravidel. REST byl poprvé zmíněn v disertační práci R. T. Fieldinga. [21] V dnešní době tvoří jeden z nejpoužívanějších standardů pro API komunikaci. REST API používá http metody POST, PUT, GET, DELETE a další. Za standard se považuje přenos dat pomoci JSON formátu. Pro URL se používají podstatná jména. Pro označení kolekcí se používají podstatná

jména v množném čísle. Hierarchické vrstvení zdrojů v URL se používá pro přenos informace o specifické volbě zdroje. Z důvodu možnosti více verzí se na počátku URL vkládá informace o verzi REST API. HTTP status kód je součástí odpovědi. [22]

3.7.1 Mapování metod REST API

Mapování je rozděleno do bloků podle funkce. Všechny URL mají společný začátek `/api/v1`, po kterém následuje příslušná cesta k metodě, jak je popsáno v následujících subsekcích.

3.7.1.1 admin

URL pro zobrazení všech boxů, šneků, snůšek nebo skupin

GET	/admin/snek	Výpis všech šneků v aplikaci
GET	/admin/box	Výpis všech boxů v aplikaci
GET	/admin/skupina	Výpis všech skupin v aplikaci
GET	/admin/snuska	Výpis všech snůšek v aplikaci

3.7.1.2 auth

URL pro přihlášení a registraci do aplikace

POST	/auth/signup	Registrace uživatele do aplikace
POST	/auth/signin	Přihlášení uživatele do aplikace

3.7.1.3 snek

URL pro operace se šneky a jejich atributy

GET	/snek	Vrátí pole šneků
POST	/snek/{id}	Přidaní šneka do aplikace
GET	/snek/{id}	Vrátí objekt šnek
PUT	/snek/{id}	Upraví šneka
DELETE	/snek/{id}	Smaže šneka
GET	/snek/{snekId}/skupina	Vrátí skupinu ve které se šnek nachází
GET	/snek/{snekId}/snuska	Vrátí seznam snůšek, kterým je šnek matkou
POST	/snek/{snekId}/skupina/{skupinaId}	Změní skupinu šneka
GET	/snek/{snekId}/udalost	Vrátí pole událostí patřících ke šnekovi
GET	/snek/{snekId}/rodokmen	Vrátí rodokmen pouze matek pro vybraného šneka
GET	/snek/{snekId}/mereni	Vrátí pole měření patřících ke šnekovi
POST	/snek/{snekId}/image	Přidaní obrázků šneka do aplikace
POST	/snek/{snekId}/inactivate	Inaktivace šneka. Po níž nenáleží do žádné skupiny

3.7.1.4 box

URL pro operace s chovnými boxy

POST	/box	Přidání boxu do aplikace
GET	/box	Vrátí pole boxu patřících uživateli
GET	/box/{boxId}	Vrátí specificky box
PUT	/box/{boxId}	Upraví box
DELETE	/box/{boxId}	Smaže specifický box
GET	/box/skupina/{skupinaId}	Vrátí specificky box ve kterém je skupina
GET	/box/{boxId}/udalost	Vrátí pole událostí patřících k boxu
POST	/box/{boxId}/image	Přidání obrázku boxu do aplikace

3.7.1.5 skupina

URL pro operace se skupinami

GET	/skupina	Vrátí pole skupin příslušného uživatele
GET	/skupina/{id}	Vrátí skupinu
POST	/skupina/{id}	Přidaní skupiny do aplikace
PUT	/skupina/{id}	Upraví skupinu
DELETE	/skupina/{id}	Smaže specifickou skupinu
GET	/skupina/{skupinaId}/udalost	Vrátí pole událostí patřících ke skupině
POST	/skupina/{skupinaId}/inactivate	Inaktivuje skupinu vyřazením z boxu
POST	/skupina/{skupinaId}/box/{boxId}	Změna boxu u skupiny

3.7.1.6 snuska

URL pro operace se snůškami

GET	/snuska	Vrátí pole snůšek
GET	/snuska/{id}	Vrátí snůšku
POST	/snuska/{id}	Přidání snůšky do aplikace
PUT	/snuska/{id}	Upraví snůšku
DELETE	/snuska/{id}	Smaže specifickou snůšku
GET	/snuska/{snuskaId}/snek	Vrátí pole šneků kteří vzešli ze snůšky
GET	/snuska/{snuskaId}/udalost	Vrátí pole událostí patřících ke snůšce
GET	/snuska/{snuskaId}/selekce	Vrátí pole selekci patřících ke snůšce
POST	/snuska/{snuskaId}/selekce	Přidaní selekce ke snůšce v aplikaci
GET	/snuska/selekce/{id}	Vrátí selekci
PUT	/snuska/selekce/{id}	Upraví selekci
DELETE	/snuska/selekce/{id}	Smaže selekci

3.7.1.7 mereni

URL pro operace s měřeními šneků

GET	/mereni/{id}	Vrátí měření
POST	/mereni/{id}	Přidání měření šneka do aplikace
PUT	/mereni/{id}	Upraví měření
DELETE	/mereni/{id}	Smaže měření

3.7.1.8 prenos

URL pro operace s přenášením šneků mezi uživateli

POST	/prenos	Přidaní přenos záznamu do aplikace
GET	/prenos/nabidka	Vrátí seznam šneků které uživatel nabízí
GET	/prenos/prijem	Vrátí seznam šneků které je možné přijmout
GET	/prenos/{prenosId}	Vrátí přenos
PUT	/prenos/{prenosId}	Upraví přenos
DELETE	/prenos/{prenosId}	Smaže přenos
POST	/prenos/{prenosId}/skupina/{skupinaId}	Příjme přenos a šneka umístí do skupiny

3.7.1.9 udalost

URL pro operace s událostmi

POST	/udalost	Přidání události
GET	/udalost/{udalostId}	Vrátí událost
PUT	/udalost/{udalostId}	Upraví událost
DELETE	/udalost/{udalostId}	Smaže událost

3.7.1.10 udalost-typ

URL pro operace s typy událostí

GET	/udalost-typ	Vrátí seznam entit udalostTyp
GET	/udalost-typ/{udalostTypId}	Vrátí udalostTyp
POST	/udalost-typ/admin	Přidání udalostTyp
PUT	/udalost-typ/admin/{udalostTypId}	Upraví udalostTyp
DELETE	/udalost-typ/admin/{udalostTypId}	Smaže udalostTyp

3.7.1.11 taxonomy

URL pro operace s taxonomickým členěním

GET	/taxonomy	Vrátí pole taxonomy záznamu
GET	/taxonomy/{taxonomyId}	Vrátí taxonomy záznam
POST	/taxonomy/admin	Přidání taxonomy záznamu do aplikace
PUT	/taxonomy/admin/{taxonomyId}	Upraví taxonomy záznam
DELETE	/taxonomy/admin/{taxonomyId}	Smaže specificky záznam taxonomy

3.7.1.12 file

URL pro operace s upload soubory

GET	/file	Vrátí všechny soubory patřící uživateli
GET	/file/{fileId}	Vrátí soubor
DELETE	/file/{fileId}	Smaže soubor
GET	/file/galerie/{galerieId}	Vrátí soubory v příslušné galerii

3.7.1.13 users

URL pro operace s uživateli

GET	/users/{userId}	Vrátí záznam o uživateli
PUT	/users/{userId}	Upraví záznam o uživateli
GET	/users/{userId}/roles	Vrátí uživatelské role
POST	/users/{userId}/changePassword	Změní heslo specifického uživatele
GET	/users/admin	Vrátí pole záznamu o uživatelích
PUT	/users/admin/{id}/roles	Změní role pro specificky záznam o uživateli
PATCH	/users/admin/{id}/scramble	Přepíše slovem ”REMOVED”údaje uživatele
GET	/users/admin/{id}/substitute	Generuje token jako substituci za vybraného uživatele
PATCH	/users/admin/{id}/activeUntil/{date}	Nastaví datum do kdy je uživatel aktivní

3.7.1.14 export

URL pro export dat

GET	/export/{id}	Generuje kompletní uživatelská data ve formátu JSON

Kapitola 4
Implementace

Tato kapitola se zabývá samotnou implementací modulu. Nejprve zde budou popsány jazyk, nástroje a knihovny, které byly pří implementaci použity. Následně bude popsána adresářová struktura projektu a jednotlivé části budou představeny. Na závěr bude diskutováno, jaké jsou možnosti dalšího rozšíření tohoto modulu aplikace.

4.1 JAVA

Java je objektově orientovaný programovací jazyk. Java byla původně vyvíjena v laboratořích Sun Microsystem jako jazyk pro komunikaci spotřebitelských elektronických přístrojů, zejména televizních zařízení, pro které byl však tento vývoj na tu dobu příliš pokročilý. S rozvojem Internetu vývoj toho programovacího jazyka brzy nabral nový směr, který se později ukázal být klíčovým. James Gosling, Mike Sheridan, Patrick Naughton a další členové týmu pracovali mezi roky 1991 a 1995 na vývoji tohoto jazyka, kterému se postupně pracovně říkalo Oak (Dub), Green, a konečně Java, pravděpodobně podle původu kávy která byla v týmu konzumována ve velkém množství. [23] První verze Javy byla zveřejněna 23. května 1995 jako alfa verze a pouze pro operační systém Solaris. [24] Od té doby Java pronikl do velikého množství odvětví s největším podílem v podnikových aplikacích a webových aplikacích. V březnu 2021 vyšlo už šestnácté vydání balíčku pro vývoj v jazyku Java. [25]

Jednou ze stěžejních myšlenek při vývoji jazyku Java byla idea přenositelnosti programu mezi platformami. Tuto ideu ilustroval i slogan „Napiš jednou, spusť kdekoliv“, se kterým přišli Sun Microsystems při propagaci Javy. [26]

Tuto přenositelnost umožňuje koncept JVM, kdy kód v jazyce JAVA je kompilován na takzvaný bytecode, který obsahuje instrukce pro JVM. JVM interpretuje bytecode programu a vytvoří prostředí pro běh programu. JVM má mnoho implementací a je zpětně kompatibilní, tedy kód vytvořený ve verzi Java 5 lze spustit v JVM implementací publikovanou spolu s Java verzí 6. [27]

JVM je také součástí JDK, který slouží k vývoji, ladění a monitorování aplikací psaných v jazyce Java. [28]

Pro implementaci tohoto bakalářského projektu byla zvolena vysoce rozšířená Java verze 8.

4.2 Apache Maven

Je nástroj vyvinutý Apache Software Foundation, sloužící pro správu a automatizaci buildu „sestavení“ projektů určený zejména pro jazyk Java. Kromě Javy může být použit i pro projekty v jazycích jako jsou C#, Ruby či Scala. [29]

Apache Maven byl vyvinut jako odnož projektu Jakarta Turbine. Maven zjednodušuje proces buildu aplikace z pohledu vývojáře. Ke konfiguraci Maven se používá pom.xml soubor, který je většinou referován pomocí zkratky POM. Samotný build probíhá v rámci tohoto POM a takzvaných „pluginů“, které jsou v něm definovány. Každý Maven projekt se tak uvbuildí analogicky. Maven je také schopen generovat build reporty z části z procesů v rámci POM a z části ze zdrojového kódu. Maven přispívá k lepší organizaci při vývoji aplikace a použití některých ověřených postupů. Mezi výhody použití Mavenu patří: jednoduchá a rychlá příprava projektu, organizace knihoven závislostí, veliké množství knihoven v repositářích, rozšiřitelnost, možnost psaní vlastních komponent a možnost integrace s množstvím dnešních vývojových nástrojů. [30]

4.3 Spring Boot

Pro implementaci této bakalářské práce byl zvolen Spring Boot. Spring Boot je open-source projekt JAVA mikro frameworku, který je postaven na starším JAVA frameworku Spring, ale na rozdíl od něj je značně omezena nutnost uživatelských konfigurací. Spring Boot je zaměřen na okamžitý a rychlý vývoj. Spring Boot používá autokonfiguraci na základě použitých závislostí. V případě potřeby je možné tuto autokonfiguraci přepsat. [31]

Spring Boot dále obsahuje vložený Apache Tomcat server což umožňuje rychlejší vývoj bez nutnosti konfigurace nasazení na server. Apache Tomcat je webový server a servlet kontejner který se používá pro běh JAVA webových aplikací. [32] Počátky projektu Spring Boot sahají do roku 2012 kdy v GitHub repositáři projektu Spring, Github repositář je služba pro údržbu a správu repositáře distribuovaného systému pro kontrolu verzí, byl uživatelem Mike Youngstromem vznesen dotaz na „vylepšení podpory pro architekturu webové aplikace bez externích kontejnerů“. Odpovědí od Phila Webba bylo, že místo úprav na projektu Spring se rozhodli vytvořit nový projekt Spring Boot, který bude adresovat tento požadavek spolu s dalšími úpravami. [33] První verze Spring boot 1.1 spatřila světlo světa v roce 2014. Aktuální verze k dubnu 2021 je Spring Boot 2.4.5. [34] Pro tuto práci byla zvolena verze 2.4.0.

4.4 Spring Initializer

Jedním z nejpoužívanějších způsobů jak vytvořit základ pro projekt ve Spring Boot je Spring Initializer. Při použití si vývojář zvolí typ nástroje pro správu balíčků, kdy může zvolit Maven nebo Gradle. Dále zvolí jazyk, kde má výběr z Java, Kotlin nebo Groovy. Zvolí Spring Boot verzi. Výběr je pouze z nejaktuálnějších verzí. Při použití nástroje příkazové řádky Spring Boot CLI je možnost specifikovat jakoukoliv verzi. Dále se vyplní metadat projektu: Group, Artifact, Name, Description, Package name. Zvolí se typ balíčku pro nasazení, kde je výběr mezi JAR nebo WAR. A poslední a důležitou funkcionalitou je přidávání závislostí pro specifickou verzi Spring Boot. Některé ze závislostí vybraných pro tento projekt budou níže popsány. Takto vytvořený projekt se nakonec vyexportuje jako soubor typu ZIP obsahující základní strukturu projektu. [35]

4.5 Spring Security

Spring Security je nastavitelný famework pro správu authentifikace, authorizace a kontroly přístupu. Je považován za standard pro zabezpečení aplikací se serverovou částí ve Spring boot. Velkou výhodou je možnost nastavení a přizpůsobení pro specifické nároky. Pomocí třídy která je potomkem třídy WebSecurityConfigurerAdapter lze nastavit filtry pro jednotlivé URL, určit které role umožní přístup nebo nastavit filtr pro autorizační token.

4.6 Spring Data JPA

Spring Data JPA modul umožňuje snadnou implementaci repositářů založených na JPA. Zprostředkuje vrstvu pro přístup k datům založenou na JPA implementaci. Takto je vývoj aplikace značně urychlen, protože vývojář nemusí trávit čas psaním JPA implementačních tříd. V případě Spring Data JPA stačí nadefinovat rozhraní (Interface) a případně doplnit specifické metody. Důležité je aby repositáře extendovali JpaRepository jak je zobrazeno ve výpisu kódu 4.1.

Výpis kódu 4.1 Rozhraní implementující JpaRepository

```
public interface TaxonomyDao extends JpaRepository<Taxonomy, Integer> {
}
```

I v situaci kdy implementace JPA rozhraní s databází je implementována pomocí Spring Data JPA, musí vývojář dávat pozor na nebezpečné postupy. Například chybným použitím metody `findByUsername`, které, jak je zobrazeno ve výpisu kódu 4.2, by mohl být umožněn útok pomocí SQL vložení (SQL Injection).

Výpis kódu 4.2 Chybné použití metody JpaRepository

```
public interface UsersDao extends JpaRepository<Users, Integer> {
Optional<Users> findByUsername(String username);
}
```

Tato metoda je implementována pomocí JPQL dotazu Spring Data JPA implementuje metodu jako query s vloženou hodnotou textového parametru `select u from Users u where u.username = ?1`. Aby bylo této potenciální hrozbě zamezeno, je třeba pozměnit implementaci metody `findByUsername` tak, aby hodnota parametru username byla vkládána pomocí adresovaného parametru. Tato úprava je vidět ve výpisu kódu 4.3.

Výpis kódu 4.3 Správné použití metody JpaRepository

```
public interface UsersDao extends JpaRepository<Users, Integer> {
    @Query("select␣u␣from␣Users␣u␣where␣u.username␣like␣:username")
    Optional<Users> findByUsername(@Param("username") String username);
}
```

Máme-li však například metodu `findById`, kde Id je typu Integer nebo Long, pak takový útok nehrozí, protože hodnota vstupního parametru je Integer nebo Long a nemůže proto vnést škodlivý kód.

4.7 Autentifikace

Důležitou součástí každé webové aplikace, která obsahuje citlivé nebo soukromé informace, je autentifikace uživatele při užití aplikace. Autentifikací se rozumí ověření identity uživatele. Existuje několik různých způsobů.

Tradičním a dodnes velmi oblíbeným způsobem je autentifikace pomocí takzvaných cookies. Cookies jsou informace, které se ukládají do hlavičky http volání nebo odpovědi. [36] Cookies jsou vygenerovány při přihlášení uživatele a na straně serveru jsou uloženy buď do paměti, nebo do databáze. Takto vygenerované cookies nesou specifický identifikátor, často také identifikátor klienta na serveru a případně i další informace. Po vygenerování jsou v odpovědi zaslány klientovi, kde jsou uloženy, většinou v prohlížeči. Při zaslání dalšího požadavku jsou připojeny do hlavičky,

a tak je server schopen jednoznačně určit, o kterého uživatele se jedná. V dnešní době, kdy neustále dochází k nárůstu počtu uživatelů a ke zvyšování nároků na server strany aplikací, může tento tradiční postup narážet na problémy. Jedním z nich je, že některé dnešní aplikace se rozkládají na mnoha serverech a cookies tudíž nemohou být uloženy pouze na specifickém serveru. Toto částečně řeší takzvané „sticky sessions“, které umožňují navedení požadavku na správnou instanci serveru. Dalším problémem může být, že mnohé dnešní aplikace jsou sítí mikroservis, které mezi sebou vzájemně komunikují. Přenos cookies mezi mikroservisy není často vhodným řešením. [37]

Dalším z možných způsobů autentifikace je autentifikační token. Nejpoužívanějším typem tokenu v dnešní době je JWT token. JWT token je standardem [38], který představuje kompaktní a soběstačný způsob přenosu informace mezi stranami jako JSON objekt. Tato informace může být verifikována a důvěryhodná díky digitálnímu podpisu buď pomocí tajemství s pomocí HMAC algoritmu, nebo pomocí páru veřejného a soukromého klíče a použití RSA nebo ECDSA. Při použití JWT tokenu pro autentifikaci je po přihlášení uživatele vygenerován podepsaný JWT token, který často nese i data dle potřeby. Nejčastěji nese data o uživateli. Takto vygenerovaný token je odeslán jako odpověď klientovi, který ho dále připojuje ke každému požadavku do hlavičky, nejčastěji pod Authorization: Bearer ¡JWT token¿. Server si při přijetí požadavku ověří platnost příslušného tokenu a z informací které nese je schopen autentifikovat a případně autorizovat uživatele. [39]

Struktura JWT tokenu je tvořena třemi částmi:

1. Hlavička, která většinou nese informace o typu tokenu, v tomto případě JWT, a algoritmu který byl použit pro podepsání.
2. Payload „náklad“, který nese Claims, což jsou informace o identitě, a případně další data pokud je to potřeba. Existují tři typy claims:
   a. Registrované claims: iss (vydavatel), exp (čas vypršení platnosti), sub (subjekt), aud (audience) a případně další atributy definované ve standardu.
   b. Veřejné claims: které můžou být definovány dle potřeb komunikujících stran, ale mělo by se předcházet kolizi s ostatními claims.
   c. Soukromé claims: které slouží k přenosu informací mezi dohodnutými stranami a nepatří mezi registrované ani veřejné claims.
3. Podpis, který je vytvořen spojením hlavičky, payload a specifického tajemství serveru a jako celek zakódováno pomocí příslušného algoritmu.

V implementaci je JWT token generován pomocí knihovny jjwt.

4.8 Mapstruct

Jednou ze základních otázek při tvorbě REST API je formát přenášených dat. Nejčastější volbou jsou dva formáty XML a JSON. [40] Oba dva formáty jsou čitelné formy pro přenos dat a objektů mezi informačními systémy. Původnější XML vznikl jako implementace SGML pro přenos dat na webu. Později přišla odlišná forma přenosu dat, která vznikla jako z ECMA Script programovacího jazyku a byla nazvána JSON. Tvoří jí úzká skupina strukturních pravidel pro přenos strukturovaných dat. [41] V dnešní době je JSON vysoce rozšířenou formou pro přenos dat. Při přenosu dat mezi systémy je nutná serializace objektů do formy JSON. Tato serializace je často, pro systémy napsané v jazyce JAVA, implementována za pomoci externích knihoven. Mezi tyto knihovny patří populární Jackson Project. [42] Pro jednoduché POJO je takto zajištěná serializace a deserializace plně dostačující. Pokud ale existuje mezi serializovanými objekty vzájemná křížová reference, vznikla by při serializaci takovéhoto objektu nekonečná smyčka. Takováto

křížová reference často vzniká při implementaci entit databázových tabulek, kde vazba mezi tabulkami je popsána oboustrannou vazbou. Aby bylo možné vzniku takových nekonečných smyček zabránit, jsou často použity DTO entity. U takových objektů je možné vazbu implementovat pouze z jedné strany, čímž je bezpečná serializace umožněna. Mapování entit na tyto DTO třídy je možné implementovat vlastním kódem, nebo pro tento účel dnes existuje několik knihoven. Pro tento projekt byla zvolena populární knihovna Mapstruct, která generuje kód pro třídu obsahující metody pro mapování automaticky na základě pravidel a kódu rozhraní nebo abstraktní třídy.

Výpis kódu 4.4 Mapstruct rozhraní mapující objekt Snek

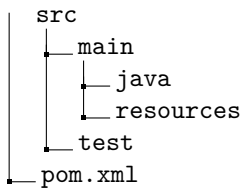
```
public String generateJwtToken(Authentication authentication)
@Mapper(componentModel = "spring", uses = {SkupinaBezSnekuMapper.class,
MereniSnekMapper.class, TaxonomyMapper.class})
public interface SnekMapper {

@Mapping(target = "galerieId", expression = "java(getGalerieId(snek))")
@Mapping(target = "snuskaId", expression = "java(getSnuskaId(snek))")
@Mapping(target = "taxonomyId", expression = "java(getTaxonomyId(snek))")
SnekDto toSnekDto(Snek snek);
List<SnekDto> toSnekDtos(List<Snek> sneks);

Snek toSnek(SnekDto snekDto, @Context SnuskaDao snuskaDao,
@Context TaxonomyDao taxonomyDao);

default Integer getSnuskaId(Snek snek) {
    if(snek.getSnuska()==null)
        return null;
    else
        return snek.getSnuska().getSnuskaId();
}

default Integer getTaxonomyId(Snek snek) {
    if(snek.getTaxonomy()==null)
        return null;
    else
        return snek.getTaxonomy().getTaxonomyId();
}

default Integer getGalerieId(Snek snek) {
    if(snek.getGalerie()==null)
        return null;
    else
        return snek.getGalerie().getGalerieId();
}

@AfterMapping
default void toSnek(SnekDto snekDto, @MappingTarget Snek snek,
@Context SnuskaDao snuskaDao) {
    if (snekDto.getSnuskaId()==null)
        snek.setSnuska(null);
    else {
        Snuska snuska = snuskaDao.findById(snekDto.getSnuskaId())
        .orElseThrow(() -> new RuntimeException("Snuska␣with␣SnuskaId␣:"
        + snekDto.getSnuskaId() + "does␣not␣exist."));
        snek.setSnuska(snuska);
    }
```

```
}

@AfterMapping
default void toSnek(SnekDto snekDto, @MappingTarget Snek snek,
@Context TaxonomyDao taxonomyDao) {
    if (snekDto.getTaxonomyId()==null)
        snek.setTaxonomy(null);
    else {
        Taxonomy taxonomy = taxonomyDao.findById(snekDto.getTaxonomyId())
        .orElseThrow(() ->new RuntimeException("Taxonomy␣with␣TaxonomyId"
        + snekDto.getTaxonomyId() + "␣has␣no␣match"));

        snek.setTaxonomy(taxonomy);
    }
}
}
```

Výpis kódu 4.4 zobrazuje mapovací rozhraní pro objekt šnek. Anotace `@Mapper` signalizuje Mapstruct generátoru kódu, aby vygeneroval implementaci rozhraní při kompilaci kódu. Atribut `componentModel` určuje, ve kterém modelu bude mapper použit. V rámci samotného rozhraní jsou definovány potřebné metody. Metoda `toSnekDto` pro mapování entity na DTO objekt. Ten je možné pak serializovat a data přenést ve formátu JSON. Dále také Metoda `toSnekDtos` která interně využívá metodu `toSnekDto` pro serializaci seznamu entit do seznamu DTO objektů. Pro deserializaci je anlogicky definována metoda `toSnek`. Anotace `@Mapping` určuje specifické mapování, kde target určuje cílový atribut a expression definuje JAVA kód, který má být použit pro dané mapování. Defaultní metody lze implementovat v rozhraní od Javy verze 8. [43] Anotace `@AfterMapping` určuje metodu, která se provede na závěr mapování a `@Context` umožňuje vložení závislosti (Dependency Injection). [44]

4.9 Organizace složky projektu

Zdrojový kód pro aplikace ve Spring Boot je organizován standardně, kde zdrojový kód je obsažen v adresáři `src/main/java`. Konfigurační soubory se nacházejí v adresáři `src/resources` a automatické testy jsou v adresáři `src/test`. V samotném kmenovém adresáři se nachází soubor pom.xml, ve kterém je definována Maven konfigurace.



4.10 Organizace zdrojového kódu

Samotná implementace logiky aplikace je členěna do funkčních celků. V následujícím textu bude obsah jednotlivých balíčku popsán.

```
cz.fit.cvut.snail_application
├── config
├── controller
│   └── error
├── data
│   ├── dao
│   ├── dto
│   │   └── export
│   ├── entity
│   └── mapper
├── enums
│       └── export
├── messages
│   ├── request
│   └── response
├── security
│   ├── jwt
│   ├── userDetails
│   └── utils
├── service
└── snailApplication.java
```

4.10.1 config

Tento balíček obsahuje třídu `ConfigProperties` použitou pro iniciaci proměnných prostředí.

4.10.2 controller

Tento balíček představuje vystavenou stranu REST API rozhraní. Jsou v něm třídy pro implementaci metod jednotlivých REST API unikátních URL. Celkově se jedná o čtrnáct tříd. Velký počet tříd a metod je zvolen v důsledku funkčních požadavků a zároveň s přihlédnutím k architektuře REST API servisy a SPA aplikace pro uživatelské rozhraní. Takovéto rozložení dává větší flexibilitu ohledně možností při implementaci části aplikace pro uživatelské rozhraní. Například entita šnek má velké množství možných vazeb. Dále jsou přítomny možné aktivní nebo neaktivní vazby na skupiny (m:n), možná vazba na snůšku, ze které pochází, potenciální vazby na snůšky, kterým je matkou, vazba na taxonomii, potenciální vazba na měření, která pro danou entitu proběhla, vazba na události, které jsou na tuto entitu napojeny a pomoci galerie vazby na obrázky, které jsou k této entitě připojeny. V takové to situaci, při volání metody GET objekt šnek podle specifického identifikátoru, je možné vrátit plnou entitu včetně všech vazeb. Takový přístup ale omezuje možnosti REST API na předem definovaný obsah struktury. Oproti tomu implementace, která byla zvolena pro tento modul, spočívá v navrácení triviální reprezentace objektu `Snek` se základními atributy a následné možnosti doplnění potřebných objektů pomocí dalších volaní metody GET na příslušné URL metody.

- AdminController

  V této třídě se nacházejí metody pro získání kompletní sady záznamů pro entity šnek, skupina, snůška a box. URL pro tyto metody je omezeno na uživatele s rolí ADMIN.

- AuthController

  V této třídě jsou zabezpečeny dvě metody: `signup` a `signin`. První slouží k registraci nových uživatelů a druhá pro přihlášení do systému. Při registraci jsou informace o novém uživateli validovány a uloženy do databáze. Heslo je zakódováno pomocí metody `encode` ze třídy `PasswordEncoder`. Tato metoda používá SHA-1 hash, případně vyšší hash, pro zakódování hesla. Po aktivaci Administrátorem se příslušný registrovaný uživatel může přihlásit pomocí metody `signin`. Při úspěšné autentifikaci je systémem vygenerován JWT token s definovanou dobou platnosti.

- BoxController

  Tato třída obsluhuje požadavky ohledně entity box a vázaných entit. Standardně se jedná o CRUD operace. Dále je implementováno přidání události k boxu nebo souboru patřícímu k boxu do galerie.

- ExportController

  Tato třída implementuje specifickou metodu, která umožňuje export veškerých dat uživatele ve formě JSON. Pro implementaci této funkcionality byly vytvořeny speciální třídy DTO, které nesou odkaz na entity s vazbou. Takto jsou pak jednotlivé vazby vyjádřeny jako vnořené objekty JSON notace nebo seznam objektů. Takto sestavený JSON je poslán jako odpověď na dotaz.

- FileController

  Tato třída obsluhuje požadavky ohledně uložených fotografií u entit šnek nebo box. Umožňuje nahrání, smazání a poslání příslušného souboru podle identifikátoru v požadavku. Dále také umožňuje poslání všech souborů uživatele jako seznam objektů s informacemi a specifickým URL pro vyžádání každého jednoho souboru.

- MereniSnekController

  Tato třída obsluhuje jednotlivé CRUD operace pro měření šneků.

- PrenosSnekController

  Tato třída zajišťuje předání šneků mezi jednotlivými uživateli. Slouží k tomu tabulka `prenos_snek`. Třída PrenosSnekController umožňuje vypsání, jak šneků kteří byli nabídnuti uživateli k předání, tak šneků, které uživatel k předání nabízí. Pro jednotlivé záznamy nabídky jsou spravovány CRUD operace. Přenos je dokončen takzvanou akceptaci, která má svoje URL v této třídě.

- SkupinaController

  Tato třída obstarává správu skupin, kde je vedle CRUD operací a výpisu seznamu událostí umožněna také změna boxu a inaktivace skupiny. Inaktivace skupiny znamená, že v mapovací tabulce `skupina_box`, která mapuje vztahy mezi skupinami a boxy bude nejnovější záznam vyplněn v atributu `datum_do` hodnotou aktuálního časového záznamu. Taková skupina poté nepatří do žádného boxu.

- SnekController

  Tato třída slouží ke správě entit šnek a přiřazených měření. Pro entity šnek a měření jsou implementovány všechny CRUD operace. Dále je možné změnit skupinu u šneka. Nahrát soubor do galerie příslušející vybranému šnekovi. Vypsat události nebo měření a případně seznam snůšek, kterým je vybraný šnek matkou.

- SnuskaController

  V této třídě jsou implementovány CRUD operace pro entity snůška a selekce. Selekce jsou vždy připojeny k vybrané snůšce. Také je možné vyžádat si události vybrané snůšky nebo seznam šneků kteří se zrodily z příslušné snůšky.

- TaxonomyController

  V této třídě jsou spravovány entity taxonomy. Tyto entity určují, do kterého taxonomického dělení šnek patří. Na tyto URL mají přístup pouze uživatelé s rolí ADMIN.

- UdalostController

  Tato třída slouží ke zprávě entit událost. Přidání nové události je definováno objektem v těle zprávy. Vedle standardních atributů a odkazu na typ události musí v zaslaném objektu `UdalostDto` být nenulový právě jeden z těchto odkazů (snekId, SkupinaId, snuskaId, boxId). Pokud tato podmínka není splněna, je požadavek vyhodnocen jako chybný.

- UdalostTypController

  Tato třída slouží ke zprávě entit typ události. Přístup mají pouze uživatele s rolí ADMIN.

- UsersController

  V této třídě je umožněna správa uživatelů. URL pro metody této třídy je rozděleno na dva typy. Jedny mají v cestě na začátku řetězec admin. Takové metody jsou přístupné pouze uživatelům s rolí ADMIN. Ostatní jsou přístupné všem uživatelům. Mezi metody přístupné všem uživatelům patří: vypsání uživatele, vypsání rolí, úprava vybraných atributů uživatele nebo změna hesla. Mezi metody dostupné pouze administrátorům patří: seznam všech uživatelů, změna rolí uživatele, přepsání osobních údajů uživatele, změna data do kdy je uživatel aktivní a substituce za uživatele. Substituce je provedena vygenerováním JWT tokenu s příslušností k vybranému uživateli.

4.10.3 data

Tento balíček organizuje kód, který definuje datové struktury a rozhraní pro práci s daty. Obsahuje uvnitř několik dalších balíčků.

4.10.3.1 dao

Tento balíček obsahuje rozhraní pro interakci s databází. Pro samotnou implementaci je použito Spring Data JPA, takže každé rozhraní dědí z JpaRepository rozhraní. Základní metody, jakou jsou `findById`, `save`, `flush`, jsou automaticky obsaženy. Dále je možné dodefinovat další metody jako například v rozhraní `SelekceDao`, zobrazeno ve výpisu z kódu 4.5.

Výpis kódu 4.5 Rozhraní selekceDao

```
public interface SelekceDao extends JpaRepository<Selekce, Integer> {

    List<Selekce> findBySnuska_SnuskaIdOrderByVelikost(Integer snuskaId);

}
```

V některých rozhraních jsou také použity SQL dotazovací jazyk jako například v rozhraní `FileDao`, výpis kódu 4.6.

Výpis kódu 4.6 Metoda v dao třídě FileDao

```
@Query(value = "SELECT␣FILE_ID,␣FILE.JMENO,␣FILE.CONTENT_TYPE," +
"NULL␣AS␣DATA,␣FILE.DATUM,␣FILE.ZVEREJNIT,␣FILE.VELIKOST␣FROM␣BOX␣" +
"LEFT␣JOIN␣SKUPINA_BOX␣ON␣+
"(BOX.BOX_ID = SKUPINA_BOX.FK_SKUPINA_BOX_BOX AND BOX.FK_BOX_USERS=?1)"␣+
" LEFT JOIN SKUPINA ON "␣+
"(SKUPINA.SKUPINA_ID = SKUPINA_BOX.FK_SKUPINA_BOX_SKUPINA) "␣+
```

```
"LEFT JOIN SNEK_SKUPINA ON "␣+
"(SKUPINA.SKUPINA_ID = SNEK_SKUPINA.FK_SNEK_SKUPINA_SKUPINA) "␣+
"LEFT JOIN SNEK AS S ON "␣+
"(SNEK_SKUPINA.FK_SNEK_SKUPINA_SNEK = S.SNEK_ID) "␣+
"LEFT JOIN GALERIE ON (S.FK_SNEK_GALERIE = GALERIE_ID) "␣+
"LEFT JOIN GALERIE_FILE ON "␣+
"(GALERIE.GALERIE_ID = GALERIE_FILE.FK_GALERIE_FILE_GALERIE) "␣+
"LEFT JOIN FILE ON (GALERIE_FILE.FK_GALERIE_FILE_FILE = FILE.FILE_ID) "␣+
"UNION "␣+
"SELECT FILE_ID, FILE.JMENO, FILE.CONTENT_TYPE, NULL AS DATA, "␣+
"FILE.DATUM, FILE.ZVEREJNIT, FILE.VELIKOST FROM BOX "␣+
"LEFT JOIN GALERIE ON "␣+
"(BOX.FK_BOX_GALERIE = GALERIE_ID AND BOX.FK_BOX_USERS=?1) "␣+
"LEFT JOIN GALERIE_FILE ON +
"(GALERIE.GALERIE_ID␣=␣GALERIE_FILE.FK_GALERIE_FILE_GALERIE)␣+
"LEFT JOIN FILE ON +
"(GALERIE_FILE.FK_GALERIE_FILE_FILE␣=␣FILE.FILE_ID)", nativeQuery = true)
List<File> selectAllFileByUser(Integer userId);
```

4.10.3.2 dto

Zde jsou definovány datové struktury pro komunikaci přes rozhraní REST API. Speciální podsekci tvoří balíček export, kde jsou definovány modifikované DTO objekty obsahující všechny závislosti pro export kompletních dat uživatele.

4.10.3.3 entity

Tento balíček obsahuje entity reprezentující jednotlivé tabulky v databázi. Pro vztahy m:n je buď použita mapovací tabulka s vlastním primárním klíčem nadefinována v databázovém schématu. V tomto případě bývají přítomny další atributy vedle cizích klíčů. U vztahu mezi tabulkami `roles` a `users` má tabulka `users_roles` kompozitní primární klíč. Toto je implementováno extra třídou `UsersRolesPK` s atributy hodnot obou kompozitních primárních klíčů. Samotná třída `UsersRoles` má poté anotaci `@IdClass(UsersRolesPK.class)`, která zajišťuje vzájemné namapování vztahů mezi třídami `Users` a `Roles`.

4.10.3.4 mapper

Balíček mapper obsahuje rozhraní definující automatické generování mapujících tříd z entit na dto třídy a opačně. Samostatnou skupinu tvoří rozhraní v balíčku export určená pro mapování se závislostmi pro export kompletních dat uživatele. Použití knihovny mapstruct již bylo popsáno v sekci 4.8. Vedle již zmiňovaných rozhraní jsou přítomny i tři napsané implementace v balíčku export.customMapper, a to z důvodu přidání závislosti na jiném mapperu. Tato závislost lze zabezpečit v anotaci `@Mapper` u deklarace rozhraní pomocí parametru `uses` ale pouze pro přímé mapování datových typů. U tří zmíněných implementovaných mapovacích tříd se jedná o zanořené použití vlženého mapperu v java kódu při filtrování kolekce.

4.10.4 enums

Obsahuje dvě enum třídy. Jedna pro uživatelské role a druhá pro atribut stavu v objektu `SnekDto`.

4.10.5 messages

Tento balíček obsahuje balíčky request a response obsahující třídy nesoucí informace pro metody `signup` a `signin`, které se nacházejí ve třídě `AuthController`. Balíček request také obsahuje třídu `ChangePasswordRequest` pro požadavek na změnu hesla. Vedle těchto balíčků je zde třída `MessageConstants`, která obsahuje konstanty pro většinu zpráv v aplikaci.

4.10.6 security

Balíček security obsahuje tři balíčky. Balíček JWT obsahuje třídy pro autorizaci JWT tokenu. Balíček userDetails pro implementaci userDetails a balíček utils. Balíček utils obsahuje třídu `JwtUtils`, která implementuje metody pro práci s JWT tokenem, a třídu `AuthoriseUtils`, která slouží k autorizaci oprávnění přístupu k datům. Data v aplikaci jsou strukturovaná tak, že kromě obecných entit jako jsou taxonomy, přenos šneka a typ události všechny ostatní data patří některému z uživatelů. Pomocí metod ve třídě `AuthoriseUtils` se testuje, zda má příslušný uživatel právo na přístup ke specifické entitě. Například metoda ve výpisu kódu 4.7 testuje, zda příslušný uživatel má právo na přístup k entitě box definované identifikátorem `boxId`.

Výpis kódu 4.7 Metoda authoriseBox

```
public boolean authoriseBox(int boxId, String jwtToken) {
        Users user = getUser(jwtToken);
        if (checkIfAdmin(user))
            return true;
        Box box = boxDao.findById(boxId)
        .orElseThrow(() ->
        new RuntimeException(MessageConstants.BOX_WITH_ID
        + boxId + MessageConstants.NOT_EXIST_ERROR));
        return box.getUser().getUserId().equals(user.getUserId());
    }
```

Balíček security také obsahuje třídu `WebSecurityConfig`, kde je možné pomocí `@Override` metody `configure` s argumentem `HttpSecurity` nadefinovat hodnoty takzvaných „antMatchers“, které pak určují omezení přístupu k jednotlivým URL podle rolí uživatele.

4.10.7 service

Tento balíček obsahuje implementaci logiky aplikace. Jsou zde implementovány všechny metody pro třídy z balíčku controller.

4.11 Možná další rozšíření aplikace

Aplikace je vyvíjena jako volně svázaná REST API služba pro práci s daty aplikace. Jedním z dalších možných rozšíření by mohlo být doplnění e-shop modulu, který by zpravoval obchodování se šneky. V tuto chvíli aplikace umožňuje přenos vlastnictví šneka mezi uživateli aplikace, ale pro plnohodnotný e-shop by bylo třeba upravit datový model a implementovat entity a logiku pro e-shop. Dalším možným rozšířením je přidání dalších druhů zvířat. Zde by stačilo rozšíření

databázového modelu a implementace dalších REST API služeb. U tohoto typu rozšíření je dokonce možné vyčlenit zprávu uživatelů ze stávajícího modelu. Upravit vazbu boxu na uživatele tak, aby fungovala přes odlišné mikroslužby a přidat nový modul jako další mikroslužbu aplikace. Při tomto typu rozšíření není rozsáhlost systému tolik omezena. Uživatelská část aplikace je plánována jako SPA implementace. Vzhledem k tomu, že tato část aplikace je REST API, je do budoucna možné implementovat také uživatelské rozhraní jako aplikaci pro mobilní zařízení a tablety, která by komunikovalo s tímto modulem.

Kapitola 5
Testování

Nedílnou součástí vývoje aplikací je testování. Pro vývojáře je nezbytné použití automatických testů, které mohou být spouště opakovaně a automaticky bez vysoké časové konzumace. Jednou z důležitých funkcí automatických testů při vývoji aplikace je kontrola, zda nedochází ke ztrátám nebo narušení funkčnosti aplikace v rámci vývoje a úpravy zdrojového kódu. Mezi typy automatických testů patří jednotkové testy a integrační testy.

5.1 Jednotkové testy

Tento typ testů se zaměřuje na testování minimálních celků kódu. Účelem těchto testů je potvrdit, že každá taková část kódu produkuje očekávaný výstup. Tyto testy by měli být krátké, samo vysvětlující. Měli by být zaměřené na logiku třídy. Pokud je to možné měli by pokrývat i krajní hodnoty vstupu. Důležitou vlastností jednotkových testů je jejich rychlost. Pouštějí se často, například při zasílání změn na kódu do centrálního repositáře, při kompilaci a sestavování aplikace v Mavenu a případně dalších opakovaných činnostech při vývoji. Pro jazyk Java je velmi oblíbený testovací Framework JUnit s aktuální verzí JUnit 5. Tato verze je kompatibilní s Java 8 a výše. [45] Jednotkové testy se používají pro testovaní minimálních celků, například izolovaných veřejných metod tříd. Třídy ale často nejsou izolované a používají služby a metody jiných tříd. Aby jsme metody takových tříd mohli izolovaně testovat, musíme vazby na služby a metody jiných tříd zastoupit pomocí určité náhrady. Jedna z používaných metod jak toho docílit se nazývá mokování z anglického „mocking“. Principem této metody je, že je zavedena náhrada v podobě mocking objektu, který lze konfigurovat tak, že pro definovaný vstup vrací určený výstup. Takto zavedený mokovací objekt pak může nahradit závislost na zmíněné externí třídě. V tomto projektu je použit framework mockito. Výpis kódu 5.1 zobrazuje jednotkový test pro metodu insert třídy `TaxonomyService`. Anotace `@ExtendWith` zajistí registraci Mockito extenze. Anotace `@InjectMocks` zajistí automatické vložení mocků do testované třídy. Anotace `@Mock` identifikuje objekt pro mokování. Samotná konfigurace mocku je nastavena pomocí sady metod.

Výpis kódu 5.1 Jednotkový test TaxonomyServiceTest

```
@ExtendWith(MockitoExtension.class)
class TaxonomyServiceTest {

    @InjectMocks
    TaxonomyService taxonomyService;

    @Mock
    private TaxonomyDao taxonomyDao;;
```

```
    @Mock
    private TaxonomyMapper taxonomyMapper;;

    @Test
    void insert() throws Exception {
        TaxonomyDto taxDto1 = new TaxonomyDto();
        taxDto1.setTaxonomyId(0);
        taxDto1.setJmeno("tax1");
        taxDto1.setPopis("tax1 popis");

        Taxonomy tax1 = new Taxonomy();
        tax1.setTaxonomyId(0);
        tax1.setJmeno("tax1");
        tax1.setPopis("tax1 popis");

        when(taxonomyMapper.toTaxonomy(taxDto1)).thenReturn(tax1);
        when(taxonomyDao.save(tax1)).thenReturn(tax1);
        when(taxonomyMapper.toTaxonomyDto(tax1)).thenReturn(taxDto1);
        TaxonomyDto taxonomyDto = taxonomyService.insert(taxDto1);
        assertEquals(taxDto1,taxonomyDto);
    }
}
```

5.2 Integrační testy

Integrační testy představují další stupeň testů, kdy je testována integrace mezi jednotlivými třídami a moduly. [46] Pro tento typ testu je třeba nastartovat aplikační komplex. Pro náš typ projektu je také třeba mít testovací data v testovací databázi. Pro náš účel je použita H2 databáze. Po nastartování kontextu je ověřen návrat po volání jednotlivých URL REST API. Výpis kódu 5.2 zobrazuje integrační test metody `getById` pro třídu `TaxonomyController`. Anotace `@SpringBootTest(webEnvironment=SpringBootTest.WebEnvironment.RANDOM_PORT)` umožní nastartování aplikačního kontextu použitelného pro test běžící na náhodném portu. To je dále následováno anotací `@ActiveProfiles("test")`, umožňující použití testovací databáze. Anotace `@BeforeEach` umožňuje určení metody, která je vykonána před každým z testů příslušné třídy. JWT token je vygenerován pro autentifikaci. `TestRestTemplate` je použita pro volání příslušných URI.

Výpis kódu 5.2 Integrační test TaxonomyControllerTest

```
@SpringBootTest(webEnvironment=SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
class TaxonomyControllerTestIT {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    JwtUtils jwtUtils;

    private String jwtToken;

    @Autowired
    private TestRestTemplate restTemplate;
```

```
    @BeforeEach
    public void setup() {
        Authentication authenticationObj = authenticationManager
                .authenticate(
                        new UsernamePasswordAuthenticationToken(
                                "test2", "123456"
                        )
                );
        SecurityContextHolder.getContext()
                .setAuthentication(authenticationObj);
        jwtToken = jwtUtils.generateJwtToken(authenticationObj);
        restTemplate.getRestTemplate()
                .setRequestFactory(
                        new HttpComponentsClientHttpRequestFactory()
                );
    }

    @Test
    void getById() throws JSONException {

        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.setBearerAuth(jwtToken.toString());
        HttpEntity<?> request = new HttpEntity<Object>( requestHeaders);

        ResponseEntity<String> actualresponse = restTemplate
                .exchange(
                    "/taxonomy/1", HttpMethod.GET, request, String.class
                );
        System.out.println(actualresponse.getBody());
        JSONAssert.assertEquals("{\"taxonomyId\":1," +
            "\"jmeno\":\"testDruh\",\"popis\":\"Druh␣pro␣testovani\",\""
           + "fkTaxonomyTaxonomy\":null}",actualresponse.getBody(),true);
    }
}
```

5.3 Pokrytí testy

Jednotkové testy pokrývají velkou část kódu v balíčcích security a service. Implementace testů je nejrozsáhlejší a nejsložitější v případě tříd se složitou logikou a vysokou interakci s dalšími třídami. V balíčku security je takovým příkladem třída `AuthoriseUtils`. U balíčku service je implementace nejrozsáhlejší u `SnekService`.

Integrační testy byly implementovány pro všechny REST API URL metody ve třídách balíčku controller. Pro pokrytí test případů je využit zakládající skript pro testovací databázi. Celkově je implementováno osmdesát integračních testů.

Při spuštění kompletních testů je pokryto přibližně osmdesát tři procent metod. Integrační testy tvoří velkou část tohoto pokrytí. Využití testů napomohlo odhalení některých chyb v rámci vývoje aplikace. Stejně tak se u jednotkových testů projevila jejich ochrana před regresí funkcionality, kdy úpravou kódu došlo k porušení funkcionality. Takový zásah byl odhalen při spuštění testů a pomocí zpětného dohledání změn byl kód upraven takovým způsobem, že původní funkcionalita nebyla porušena. Stejně tak i integrační testy napomohli korektnímu vývoji. U integračních testů je však jistou nevýhodou jejich časová náročnost, protože pro jejich běh se startuje aplikační kontext.

Kapitola 6
Závěr

Cílem tohoto bakalářského projektu bylo navrhnout a implementovat datový modul pro informační systém pro podporu chovu šneků. Tento cíl byl úspěšně splněn.

Pro dosažení tohoto cíle bylo nejprve třeba provést průzkum domény chovu šneků a seznámit se s obdobnými aplikacemi. Následným krokem bylo zjištění požadavků. Pro tento proces bylo postupováno podle zásad tvorby softwaru a výsledkem byl seznam funkčních požadavků, nefunkčních požadavků a případů užití. Klíčovou fází při navrhování webových aplikací využívajících uložení dat v databázi je samotná tvorba datového modelu. Tomu byla věnována značná část návrhu aplikace.

Pro implementaci bylo třeba se obeznámit s metodikou tvorby softwaru v objektově orientovaném jazyce Java a to specificky ve frameworku Spring Boot. Autor této práce si vybral framework Spring Boot nejen pro jeho oblibu při tvorbě webových aplikací, ale také pro osobní zájem se s tímto frameworkem seznámit a naučit se vytvářet software s jeho použitím. Dále bylo třeba pracovat s databázemi PostgreSQL pro ukládání dat a H2 pro integrační testy.

Autor projektu se také seznámil s použitím technologie JWT token pro autentifikaci uživatelů. Při samotné implementaci autor vytvořil prototyp datového modulu aplikace vystavující REST API pro komunikaci s plánovaným modulem uživatelského rozhraní. Autor si vedle práce na implementaci softwaru osvojil také znalosti o chovu šneků. Výsledný produkt by měl přispět k efektivnější evidenci a sledování šnečí populace pro chovatele. Jeho návrh umožňuje další rozšíření v budoucnosti a poskytuje značnou flexibilitu při implementaci uživatelského rozhraní.

Bibliografie

1. HAMMERSCHMIED, Petr. *SUPERZOO – Oblovka - oblíbený a nenáročný mazlíček* [online]. [N.d.] [cit. 2021-03-18]. Dostupné z: https://www.superzoo.cz/skola/oblovka-oblibeny-a-nenarocny-mazlicek/.
2. GUINNESS WORLD RECORD. *Guinness World Record – Largest Snail* [online]. [N.d.] [cit. 2021-03-18]. Dostupné z: https://www.guinnessworldrecords.com/world-records/70397-largest-snail.
3. THIENGO, C. Silvana. Rapid spread of an invasive snail in South America: the giant African snail, Achatina fulica, in Brasil. *Biological Invasions*. 2007. ISSN 1573-1464. Dostupné také z: https://doi.org/10.1007/s10530-006-9069-6.
4. BRUSH, Kate. *Use Case* [online]. [N.d.] [cit. 2021-03-18]. Dostupné z: searchsoftwarequality.techtarget.com/definition/use-case.
5. FIELDING, P. *Hypertext Transfer Protocol – HTTP/1.1* [online]. 1999 [cit. 2021-03-20]. Dostupné z DOI: https://doi.org/10.17487/RFC2616.
6. FETTE, I. *The WebSocket Protocol* [online]. 2011 [cit. 2021-03-20]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc6455.
7. GNATYK, Romana. *Microservices vs Monolith: which architecture is the best choice for your business?* [Online]. 2018 [cit. 2021-03-18]. Dostupné z: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/#:~:text=While%5C%20a%5C%20monolithic%5C%20application%5C%20is,as%5C%20perform%5C%20the%5C%20specific%5C%20functions..
8. RICHARDSON, Chris. *Pattern: Microservice Architecture* [[online]]. 2020 [cit. 2021-03-22]. Dostupné z: https://microservices.io/patterns/microservices.html.
9. CLOUDFLARE. *What is serverless computing? — Serverless definition* [online]. [N.d.] [cit. 2021-03-22]. Dostupné z: https://www.cloudflare.com/learning/serverless/what-is-serverless/#:~:text=Serverless%5C%20computing%5C%20is%5C%20a%5C%20method,bandwidth%5C%20or%5C%20number%5C%20of%5C%20servers..
10. W3. *HTML & CSS* [online]. [N.d.] [cit. 2021-03-22]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss.
11. GARRETT, Jesse James. *Ajax: A New Approach to Web Applications* [online]. 2005 [cit. 2021-03-22]. Dostupné z: https://web.archive.org/web/20150910072359/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/.
12. NEOTERIC. *Single-page application vs. multiple-page application* [online]. 2016 [cit. 2021-03-23]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

13. EMMIT, Scott. *SPA design and architecture: understanding single page web applications*. Manning, 2015. ISBN 1617292435.
14. ASHMORE, Derek C. *The J2EE architect's handbook: how to be a successful technical architect for J2EE applications*. DVT Press, 2004. ISBN 0972954899.
15. SINGH, Vijay. *What is Frameworks? [Definition] Types of Frameworks* [online]. 2020 [cit. 2021-03-23]. Dostupné z: https://hackr.io/blog/what-is-frameworks.
16. NOSQL-DATABASE.ORG. *LIST OF NOSQL DATABASE MANAGEMENT SYSTEMS* [online]. [N.d.] [cit. 2021-03-25]. Dostupné z: https://hostingdata.co.uk/nosql-database/.
17. DRAKE, Mark. *A Comparison of NoSQL Database Management Systems and Models* [online]. 2019 [cit. 2021-03-25]. Dostupné z: https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models.
18. CODD, E. A relational model of data for large shared data banks. *Communications of the ACM*. 1970, roč. 13, č. 6. Dostupné z DOI: 10.1145/362384.362685.
19. DATE C. J., Hugh Darwen. *Databases, Types, and the Relational Model: The Third Manifesto*. Addison Wesley, 2006. ISBN 0321399420. Dostupné také z: https://freecomputerbooks.com/Databases-Types-and-the-Relational-Model-The-Third-Manifesto.html.
20. KENT, William. *A Simple Guide to Five Normal Forms in Relational Database Theory* [online]. 1982 [cit. 2021-03-29]. Dostupné z: http://www.bkent.net/Doc/simple5.htm.
21. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [Dizertační práce, UNIVERSITY OF CALIFORNIA, IRVINE]. 2000 [cit. 2021-03-29]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
22. KAPADNIS, Jay. *REST: Good Practices for API Design* [online]. 2018 [cit. 2021-03-29]. Dostupné z: https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9.
23. NIB, Tech's. *The History of JAVA Technology ||| JAVA History* [online]. 2012 [cit. 2021-04-15]. Dostupné z: http://www.techsnib.com/2012/05/the-history-of-java-technology-java.html.
24. EVANS, Benjamin. *Java: The legend* [online]. 2015 [cit. 2021-04-15]. Dostupné z: https://www.oreilly.com/content/java-the-legend/.
25. ORACLE. *Oracle Announces Java 16* [online]. 2021 [cit. 2021-04-15]. Dostupné z: https://www.oracle.com/news/announcement/oracle-announces-java-16-031621.html.
26. BLEWITT, Alex. *Java 1.0 Turns 25* [online]. 2021 [cit. 2021-04-22]. Dostupné z: https://www.infoq.com/news/2021/01/java-turns-25/.
27. VENNERS, Bill. *Free Online Chapters of Inside the Java Virtual Machine by Bill Venners* [online]. [N.d.] [cit. 2021-04-22]. Dostupné z: https://www.artima.com/insidejvm/ed2/.
28. ORACLE. *Java SE Downloads* [online]. [N.d.] [cit. 2021-04-22]. Dostupné z: https://www.oracle.com/java/technologies/javase-downloads.html.
29. GURU99. *What is Maven? Project Framework ||Uses||ArchitectureTutorial* [online]. [N.d.] [cit. 2021-04-22]. Dostupné z: https://www.guru99.com/maven-tutorial.html.
30. THE APACHE SOFTWARE FOUNDATION. *What is Maven?* [Online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://maven.apache.org/what-is-maven.html.
31. MULDERS, Michiel. *What Is Spring Boot?* [Online]. 2019 [cit. 2021-03-25]. Dostupné z: https://stackify.com/what-is-spring-boot/.
32. BAELDUNG. *Introduction to Apache Tomcat* [online]. 2020 [cit. 2021-03-25]. Dostupné z: https://www.baeldung.com/tomcat.

33. YOUNGSTROM, Mike. *Improved support for 'containerless' web application architectures [SPR-9888]* [online]. 2012 [cit. 2021-04-24]. Dostupné z: https://github.com/spring-projects/spring-framework/issues/14521.
34. MVNREPOSITORY. *Spring Boot* [online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot.
35. VMWARE, Inc. *Spring Initializer* [online]. [N.d.] [cit. 2021-04-24]. Dostupné z: https://start.spring.io/.
36. KRISTOL, David. HTTP Cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology*. 2001, roč. 1. Dostupné z DOI: 10.1145/502152.502153.
37. CHENKIE, Ryan. *5 Steps to Add Modern Authentication to Legacy Apps Using JWTs* [online]. 2015 [cit. 2021-04-24]. Dostupné z: https://auth0.com/blog/5-steps-to-add-modern-authentication-to-legacy-apps-using-jwts/.
38. JONES, M. *JSON Web Token (JWT)* [online]. 2015 [cit. 2021-04-27]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7519.
39. AUTHO. *Introduction to JSON Web Tokens* [online]. [N.d.] [cit. 2021-04-27]. Dostupné z: https://jwt.io/introduction.
40. BRAY, Tim. *Extensible Markup Language (XML) 1.0* [online]. 2008 [cit. 2021-04-27]. Dostupné z: https://www.w3.org/TR/xml/.
41. ECMA INTERNATIONAL. *The JSON data interchange syntax* [online]. 2017 [cit. 2021-04-27]. Dostupné z: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.
42. COWTOWNCODER. *FasterXML/jackson* [online]. [N.d.] [cit. 2021-04-29]. Dostupné z: https://github.com/FasterXML/jackson.
43. KHOJAYE, Muhammad Ali. *Interface Default Methods in Java 8* [online]. 2014 [cit. 2021-04-29]. Dostupné z: https://dzone.com/articles/interface-default-methods-java.
44. MAPSTRUCT. *MapStruct 1.4.2.Final Reference Guide* [online]. [N.d.] [cit. 2021-04-29]. Dostupné z: https://mapstruct.org/documentation/stable/reference/pdf/mapstruct-reference-guide.pdf.
45. BECHTOLD, Stefan. *JUnit 5 User Guide* [online]. [N.d.] [cit. 2021-04-30]. Dostupné z: https://junit.org/junit5/docs/current/user-guide/.
46. SOFTWARETESTINGHELP. *What Is Integration Testing (Tutorial With Integration Testing Example)* [online]. [N.d.] [cit. 2021-04-30]. Dostupné z: www.softwaretestinghelp.com/what-is-integration-testing/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF