Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a implementace webové aplikace pro issue tracking

Bc. Petr Skřivan

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Petr Skřivan

Systémové inženýrství a informatika
Informatika

Název práce

Návrh a implementace webové aplikace pro issue tracking

Název anglicky

Design and implementation of a web application for issue tracking

---

Cíle práce

Hlavním cílem diplomové práce je návrh a implementace webové aplikace pro issue tracking. Aplikace by měla poskytovat základní sadu funkcí potřebných pro issue tracking pro malé a střední firmy. Pro implementaci budou využity technologie Spring framework a Angular.

Metodika

Metodika vypracování teoretické části je založena na studiu odborných informačních zdrojů. Na základně zjištěných informací budou formulována teoretická východiska práce. V praktické části bude navrhnuta a implementována webová aplikace poskytující základní funkce pro issue tracking. Fáze vývoje aplikace budou popsány, aplikace bude otestována a budou navržený případné možnosti jejího dalšího rozšiřování.

Doporučený rozsah práce

60-80 stran

Klíčová slova

Issue tracking, Java, Spring, REST, TypeScript, Angular

Doporučené zdroje informací

Angular [online]. [cit. 2019-06-28]. Dostupné z: https://angular.io/docs

Oracle Java [online]. [cit. 2019-06-28]. Dostupné z: https://www.oracle.com/technetwork/java/index.html

PECINOVSKÝ, R. *Návrhové vzory : [33 vzorových postupů pro objektové programování]*. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

Spring Framework [online]. [cit. 2019-06-28]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html

---

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 18. 3. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 18. 3. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a implementace webové aplikace pro issue tracking" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 6.4.2020 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph. D. za cenné rady, ochotu a vedení práce. Chci taktéž poděkovat mé rodině za možnost studovat a podporu při celém průběhu studia, a především mé sestře za veškerou pomoc a korekturu práce.

Návrh a implementace webové aplikace pro issue tracking

Abstrakt

Diplomová práce se zabývá funkcí issue trackingu při vývoji softwaru a problematikou vývoje webových aplikací. Práce je rozdělená na dvě části, teoretická východiska a praktickou část. V teoretické části je popsán proces issue trackingu a jsou vybráni čtyři zástupci softwaru v této kategorii, kteří jsou popsáni a jsou zhodnoceny jejich výhody a nevýhody. V teorii jsou taktéž představeny technologie a postupy využité v praktické části práce.

V praktické části je nejdříve provedena analýza požadavku na aplikaci. Poté je proveden návrh aplikace, kde je vytvořen diagram případů užití a jeho jednotlivé scénáře, datový model a stavový model, který představuje základní workflow issue v systému. Design aplikace je navržen pomocí wireframů. Podle návrhu je implementována webová aplikace obsahující základní funkce pro potřeby issue trackingu. Webová aplikace se skládá ze dvou částí. Backendová část využívá programovací jazyk Java a framework Spring, zatímco frontendová část je postavena nad frameworkem Angular za použití jazyka TypeScript. Na závěr je zhodnocen průběh práce a jsou zde navržena možná budoucí rozšíření aplikace.

Klíčová slova: Issue tracking, Java, Spring, REST, TypeScript, Angular

Design and implementation of a web application for issue tracking

Abstract

The diploma thesis deals with the function of issue tracking in software development and with the difficulties of web applications development. The thesis is divided into two parts, a theoretical basis and a practical part. In the theoretical part, the process of issue tracking is described and four software representatives in this category are selected, described and their advantages and disadvantages are assessed. Technologies and procedures used in the practical part of the thesis are also introduced.

In the practical part, an analysis of application requirements is performed first. Then, the application design is done, where the use case diagram and its individual scenarios are created, along with a data model and state a model, which represents the basic workflow of an issue in the system. The design of the application is designed using wireframes. According to the design, a web application containing basic functions for issue tracking needs is implemented. The web application consists of two parts. The backend part uses Java programming language and the Spring framework, while the frontend part is built with Angular framework using TypeScript language. Finally, the thesis is evaluated, and possible future extensions of the application are suggested.

Keywords: Issue tracking, Java, Spring, REST, TypeScript, Angular

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Vývoj softwaru je v dnešní době obrovsky finančně i časově náročná činnost. Jedná se o týmy desítek a někdy i stovek lidí, které svým společným úsilím tvoří stále nové a nové aplikace. Aby bylo řízení takového počtu lidí možné a proces vývoje systému měl alespoň naději, že dosáhne stanoveného cíle, je potřeba mít práci velmi dobře zorganizovanou a taktéž musí být viditelné a vysledovatelné, kdo co dělá, dělal a co je potřeba ještě udělat. K tomuto úkolu jsou využívány tzv. issue tracking systémy. Do těchto systémů se zaznamenávají jednotlivé požadavky při vývoji, přiřazují se ke konkrétním pracovníkům týmu a ti je vykonávají.

Nejen že jsou do těchto systémů jednotlivé úkoly zaznamenávány, ale tyto aplikace mají spoustu dalších funkcí, které podporují práci po celý průběh implementace. Jednotlivým issue lze např. přiřadit prioritu, zaměstnance, který by se daným problémem měl zabývat, kdo za celý proces zodpovídá lze sledovat stav a průběh celého řešení. Tím, jak se vývoj softwaru stále posouvá dopředu, tak se i samotné issue trackery neustále vylepšují, absorbují do sebe nové funkcionality a dnes se jedná ve spouště případů už o velmi rozsáhlé a složité aplikace, které je potřeba složitě konfigurovat, aby vyhovovaly procesům organizace, která je využívá.

Cílem této diplomové práce je vytvoření aplikace, která bude obsahovat nezákladnější sadu funkcí, které jsou pro potřeby issue trackingu nezbytné. Sloužit by měla primárně menším firmám o desítkách zaměstnanců, které chtějí mít aplikaci pro issue tracking, nicméně nechtějí a ani nemají čas jí složitě konfigurovat a postrádá pro ně smysl mít obrovské systémy se spoustou funkcí, které stejně pro svoje potřeby nevyužijí. Rozhodně není cílem a ani v možnostech autora, v rámci této práce, vytvořit robustní aplikaci, která bude umět obrovské množství různých funkcí pro všechny možné i nemožné požadavky uživatelů, aby tak mohla konkurovat už zavedeným aplikacím v tomto odvětví.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je provést návrh a poté implementaci webové aplikace pro issue tracking. Aplikace bude poskytovat základní funkce a potřeby pro issue tracking. Pro implementaci budou využity moderní technologie a best practices pro vývoj softwaru.

V teoretické části práce bude cílem popsat samotný proces issue trackingu a jeho funkce. Dále popsat vybrané existující aplikace spadající do této kategorie a představit technologie použité pro implementaci.

2.2 Metodika

Informace potřebné pro dosažení stanovených cílů budou získány z odborné literatury, publikací a informačních zdrojů a také z praktických zkušeností načerpaných za dobu studia. Tyto informace budou zpracovány a poslouží jako hlavní zdroje pro tvorbu teoretické části práce.

V praktické části budou zanalyzovány požadavky, které by aplikace měla splňovat. Z této analýzy bude vycházet návrh aplikace a samotná implementace, pro kterou budou využity postupy softwarového inženýrství. Backend aplikace bude napsán v jazyku Java, konkrétně na to bude využit framework Spring. Pro implementaci frontendu bude využit jazyk TypeScript a framework Angular.

3 Teoretická východiska

3.1 Issue Tracking

Vývoj softwaru je proces, který zahrnuje desítky až stovky lidí, kteří spolu musejí spolupracovat, aby bylo možné výsledný produkt vůbec dodat. Jedná se o projektové manažery, analytiky, vývojáře, testery a další role, které neodmyslitelně k vývoji softwaru patří. Aby v tomto procesu bylo možné sledovat a efektivně organizovat práci členů týmu a taktéž evidovat, řešit a archivovat chyby, které se ve vyvíjené aplikaci v průběhu času objeví, jsou pro tento úkolu používány issue trackery.

Issue tracker lze brát jako centralizovanou databázi, kde se nachází všechny issue, které se projektu týkají. Jedná se o defekty, požadavky na nové funkcionality, úkoly pro členy týmu atd. Postupem času se ze všech těchto informací může stát obrovská znalostní báze celého projektu, kterou mohou členové týmu využívat i k tomuto účelu.[1]

Další velkou výhodu, kterou issue tracker přináší je jasně daná odpovědnost za úkol. Možnost úkol přiřadit konkrétní osobě a mít o tom záznam v daném systému přináší potřebnou dávku odpovědnosti, kterou lidé v týmu potřebují pro to, aby mohli svojí práci vykonávat efektivně a bez zbytečného vyčkávání, zda se o daný problém skutečně starat a řešit ho či to nechat na svých kolezích.[1]

3.1.1 Issue

Samotné issue je prvek, který reprezentuje daný defekt, požadavek atd. a je základním kamenem těchto systémů. Každé issue má řadu atributů. Mezi základní atributy můžeme řadit:

- Název issue
- Na koho je issue přirazeno
- Typ issue
- Kategorie
- Kdo issue založil
- Kdy ho založil
- Popis

- Závažnost
- Priorita
- Komentáře

Konkrétní počet těchto atributů se samozřejmě může lišit dle softwaru, který se používá ale třeba i podle týmu, který daný issue tracker používá. Spousta existujících řešení umožnuje alespoň omezenou konfiguraci atributů.[1]

U každého issue je taktéž možné sledovat historii jeho parametrů, především historii komentářů. Když se tedy např. defekt dostane k vývojáři, může se do komentářů podívat, zda se k problému nevyjádřil už analytik s návrhem, čím by daná chyba mohla být způsobena, či komentář od jiného vývojáře, který defekt už řešil.

3.1.2 Workflow

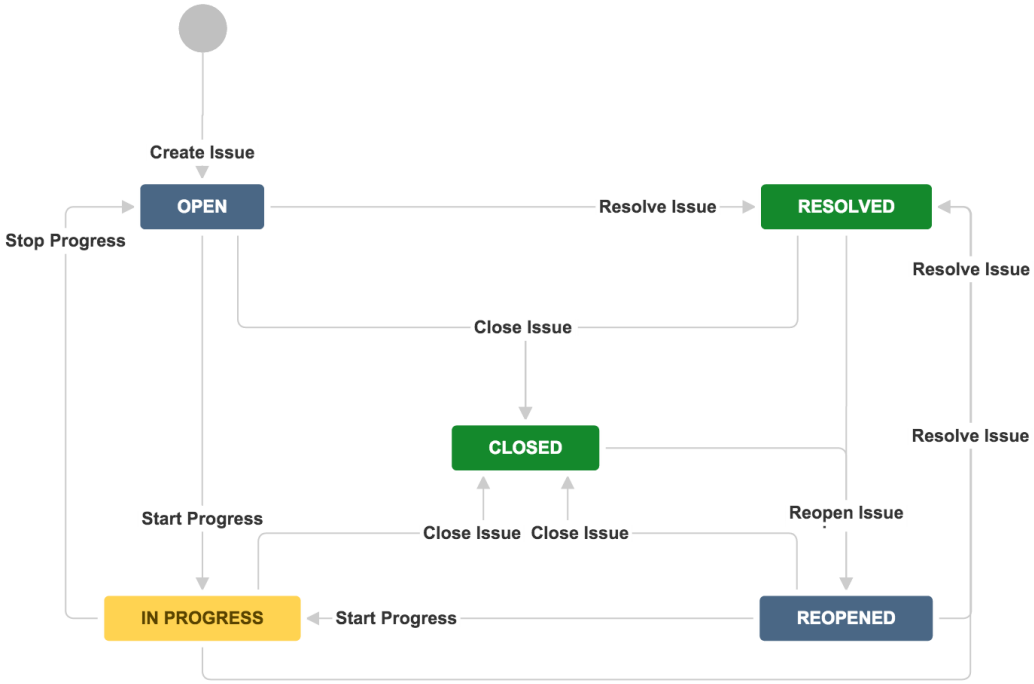
Workflow typicky představuje životní cyklus issue v systému a sestává z řady stavů a přechodů mezi nimi. Každé issue se přes tyto stavy a přechody během svého životního cyklu pohybuje. Tyto stavy by měly jasně vymezovat, kde ve workflow se issue nachází. Klasické workflow issue může obsahovat tyto stavy:

- Nový
- Otevřený
- Vyřešený
- Zavřený

O změně stavu většinou rozhoduje osoba, na kterou je v dané chvíli issue přiřazeno, a se změnou stavu přichází i změna přiřazení osoby. Aby bylo možné přejít z jednoho stavu na druhý, musí mezi nimi existovat spojení. Issue navíc nemusí jít přímou cestou od nového přes zavřený stav, ale může být v některých stavech i několikrát. U výše zmíněných stavů by přechody mohly být následující. Issue je založeno a je ve stavu Nový. Osoba, které je požadavek přiřazen rozhodne, zda se jedná o oprávněný problém, který je třeba řešit, či nikoliv. Pokud se rozhodne, že ho není třeba řešit, může issue rovnou přepnout do stavu Zavřený, případně ho přepnout zpátky na zadavatele s doplňujícími dotazy. Jestli se rozhodne, že by se daná věc měla vyřešit, přepne issue do stavu Otevřený

a rovnou vybere osobu, která se tím má zabývat. Daná osoba ticket převezme a začne na něm pracovat. Po vyřešení požadavku ho přepne do stavu Vyřešený a na zadavatele. Zadávající osoba ověří, že byla podstata problému vyřešena a issue uzavře, anebo naopak s řešením spokojená není a znovu ho otevře.

Takovýto workflow samozřejmě nemusí vyhovovat každému týmu, proto, stejně jako je tomu u parametrů, i workflow je často možné konfigurovat podle svých požadavků. Je možné stavy a přechody přidávat a odebírat způsobem, který bude organizaci co nejvíce vyhovovat a bude zaručovat co nejefektivnější práci.[2]



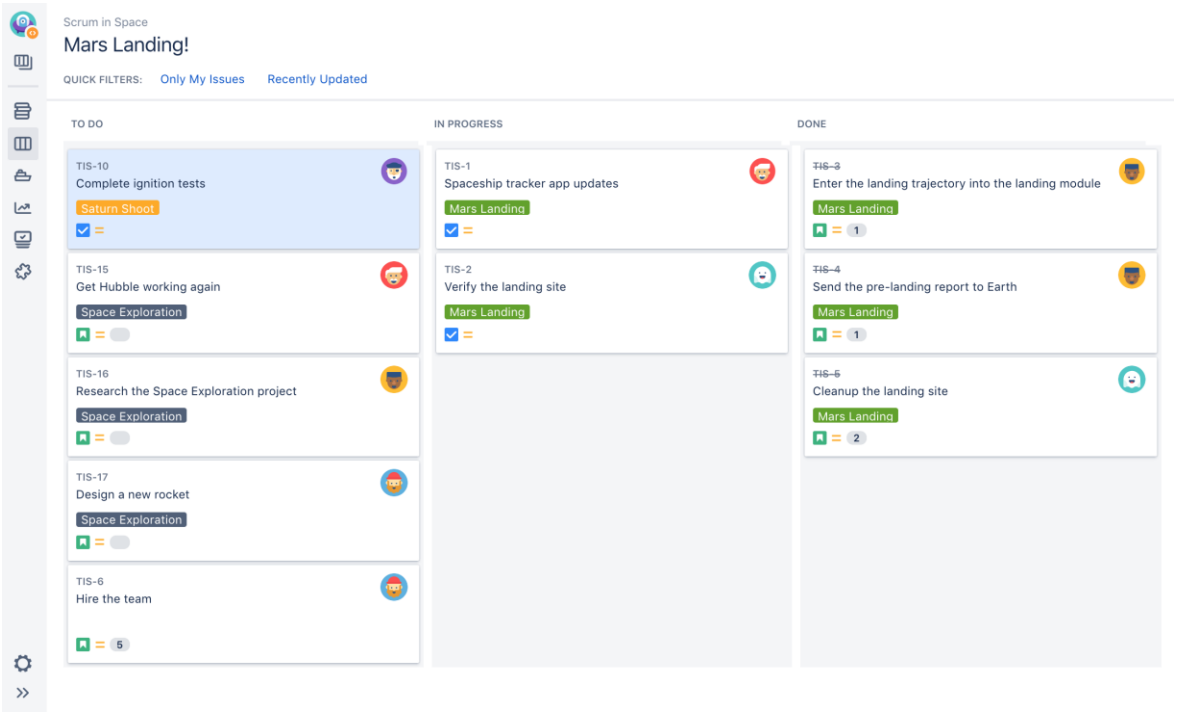
Obrázek 1 Workflow Jira
Zdroj: [2]

3.2 Popis existujících řešení

Na trhu jsou desítky aplikací, které mohou plnit funkci issue trackeru. V této kapitole bude několik takových aplikací představeno, budou popsány jejich funkce, konfigurovatelnost a budou zhodnoceny výhody i nevýhody použití jednotlivých řešení. Důvod výběru těchto konkrétních aplikací je především jejich rozšířenost a každá z nich by měla mít vždy lehce odlišné zaměření.

3.2.1 Jira

Jira je řešení pro projekt management a issue tracking od firmy Atlassian. Implementace je provedena v jazyce Java a první verze Jiry byla vydána v roce 2002. Jednalo se čistě o software pro issue tracking. Později se začala aplikace čím dál tím více vyvíjet i směrem k podpoře vedení projektů a dnes se dle samotného Atlassianu jedná o nejvyužívanější software při vývoji softwaru v agilním prostředí. Velkou výhodou použití Jiry je její snadná integrace s ostatními oblíbenými produkty Atlassianu jako např. Confluence a Bitbucket.



Obrázek 2 Jira
Zdroj: [2]

Jiru lze využívat zdarma, pokud počet uživatelů na danou licenci nepřesahuje deset. Pokud by Jiru mělo využívat více uživatelů, je již potřeba mít placenou licenci. Cena se pohybuje od 10$ za měsíc za deset uživatelů při standardní licenci až po 36 150$ měsíčně při 5000 uživatelů při prémiové licenci. Mimo to Atlassian umožňuje využívat zdarma všechny jejich produkty pro open source projekty, které splňují daná kritéria. Firma taktéž provozuje Atlassian Marketplace, kde lze pro jejich software dokupovat pluginy, které dále rozšiřují a konfigurují funkcionalitu jednotlivých aplikací. Většina těchto pluginů je vyvíjena samotnou komunitou.[2][3]

Jira obsahuje jazyk JQL (Jira Query Language), který je používán pro pokročilé vyhledávání v aplikaci. Z názvu a zkratky by se mohlo zdát, že se jedná o databázový jazyk, ale není tomu tak. JQL pouze používá velmi podobnou syntax jako SQL (Structured Query Language). Díky tomu můžete přenést svoji znalost SQL a přímo ji použít při vyhledávání v aplikaci. JQL obsahuje většinu klíčových slov a operátorů jako SQL.[2]

project = "Název projektu" AND assignee WAS "Uzivatel" AND sprint in openSprints()

Ukázkový příkaz vyhledá issue, které patří do projektu "Název projektu", jsou nebo byly v minulosti přiřazené na uživatele "Uzivatel" a které jsou zároveň v otevřeném sprintu. Lze si všimnout, že v JQL lze používat i funkce, které jsou v aplikaci již předpřipravené. V tomto příkladě se jedná o funkci openSprints(), která, jak již název napovídá, hledá pouze issue, které patří do sprintu, který je momentálně ve stavu Otevřen.[2]

Standardní funkcí issue trackerů je i možnost konfigurovat a vytvářet si svoje vlastní workflow, Jira tuto funkcionalitu taktéž umožňuje. V Jiře se lze setkat s neaktivními a aktivními workflow. Aby bylo možné workflow využívat v projektu, musím být aktivované. Aktivace probíhá pomocí mapování daného workflow na typy issue v projektu, ke kterému je workflow navázané. Neaktivní workflow jsou taková, která nejsou momentálně napojená na žádný projekt a pouze v tomto stavu lze měnit stavy a přechody, jelikož na nich nejsou závislá žádná issue. Workflow je možné exportovat i importovat a díky tomu je velmi jednoduché je přenášet mezi projekty. Jira pro jejich tvorbu obsahuje grafický editor, což práci velmi usnadňuje. [2]

Issue jsou základním prvkem systému a uživatel jich má na výběr hned několik. Jaké typy lze vybrat, záleží na Jira produktu, který uživatel využívá. Může se jednat o Jira Software, která je určená pro projekty vývoje softwaru, Jira Core, která je určená pro byznysové projekty, anebo Jira Service Desk, určená pro technickou podporu. V Jira Software jsou v nabídce tyto typy issue:

- Bug
- Epic
- Story
- Task

- Subtask

Bug je typ reprezentující nalezenou chybu ve vyvíjeném softwaru. Epic je velký požadavek, který potřebuje být rozdělen na menší prvky, které jsou typy Story. Tento typ je částí Epicu a představuje práci, kterou je potřeba splnit, aby byl splněn celý požadavek. Task je celkový úkol, který je potřeba udělat a je rozdělen na Subtasky, což jsou části celku, které musí být dodělány, aby byl Task hotový.[2]

Jira Core obsahuje pouze dva typy, a to Task a Subtask, které jsou totožné jako stejně nazvané prvky v Jira Software. V Jira Service Desk jsou na výběr tyto typy issue:

- Change
- IT help
- Incident
- New feature
- Problem
- Service request
- Service request with approval
- Support

Change je požadavek pro jakoukoliv změnu. Typ IT help využije uživatel, který chce vytvořit issue pro pomoc s problémem tykajícím se obecně IT. Incident slouží pro reportování vzniklého incidentu. New feature lze využít v situacích, kdy máte požadavek pro nové IT vybavení či funkčností softwaru. Problem reprezentuje reportování a hledání důvodu vzniku několika issue typu Incidient. Service request je typ sloužící pro vyžádání pomoci od technické podpory. Service request with approval je totožný, až na fakt, že pro jeho splnění je potřeba vyžádat si povolení od nadřízeného manažera. Posledním typem je Support, který je pro vyžádání pomoci od zákaznické podpory.[2]

Na výběr je i možnost vytvořit si nové typy pro isssue nebo typy editovat, aby týmů vyhovovaly. Typy je taktéž možné i mazat. Na mazání není žádné omezení, ale je doporučeno si vyhledat všechny issue, které jsou daného typu a vybrat u nich jiný typ. Poté je již bezpečné typ issue smazat.[2]

V Jiře je na výběr i možnost přidávat na issue dodatečná pole. Takto přidaná pole jsou vždy volitelná, což znamená, že po přidání není nutné měnit všechny existující issue

v systému. Tyto issue budou mít v poli prázdnou hodnotu, a to i v případě, má-li pole nastavenou defaultní hodnotu. Každé pole má svoje parametry jako např. jméno pole, popis, defaultní hodnota atd. Tyto parametry se dají volně měnit a přizpůsobovat. Je doporučeno nevytvářet spoustu vlastních polí, aby nebyla zhoršena výkonnost Jiry (více jak tisíc polí), a taktéž si dávat pozor na vytváření duplicitních polí s trochu pozměněným názvem, ale de facto stejným významem, jako už obsahují aktivní pole.[2]

V Jira je možné vytvořená pole taktéž smazat. Nicméně je potřeba si na tuto funkci dát pozor, jelikož při smazání pole se tím mohou rozbít již vytvořené JQL filtry či jiné scripty, které se v aplikaci používají.[2]

Pomocí linkování lze vytvářet asociace mezi jednotlivými issue a usnadňovat vyhledávání mezi souvisejícími problémy. Uživatel může potřebovat označit issue jako duplikát jiného issue, nebo že jeden prvek musí být kompletně dokončený, než je možné začít práci na jiném. Je možné využít několik typů vazeb:[2]

- Relates to - „souvisí s“,
- Duplicates / is duplicated by – „duplikuje / je duplikován“
- Blocks / is blocked by – „blokuje / je blokován“
- Clones / is cloned by – „klonuje / je klonován“

Vazba „souvisí“ ukazuje, že spolu dvě issue nějakým způsobem souvisí, ale jsou na sobě nezávislá. Odkaz „Duplikuje“ se přidává k issue, které je duplikací jiného issue v systému. Tomu se přidělí link „je duplikován“. Pokud je nějaký problém blokován, nalinkuje se jako „je blokován“ a prvku, který ho blokuje, se nastaví vazba typu „blokuje“. Poslední možnou vazbou je, pokud je potřeba jeden issue naklonovat. Původnímu issue se nastaví vazba „je klonován“ a novému „klonuje“.[2]

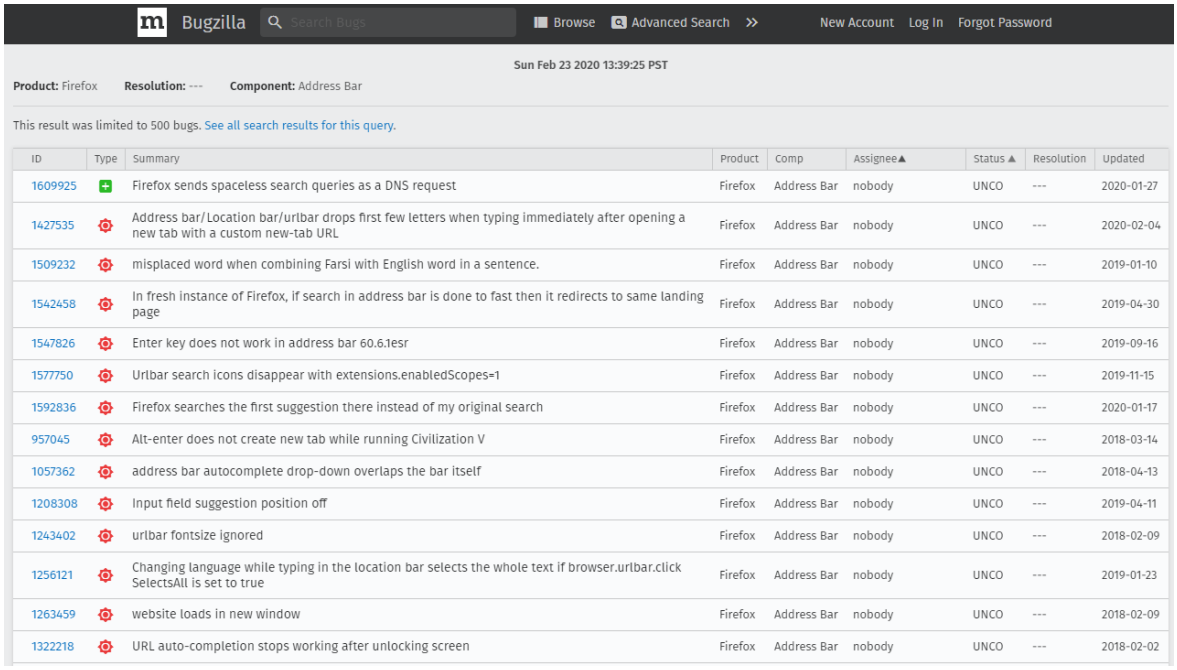
V aplikaci je oficiální, plně podporována česká lokalizace. Mimo funkce typické pro issue tracking, obsahuje Jira velké množství nástrojů zaměřených na agilní metody Scrum, Kanban a agilní reporting.[3]

3.2.2 Bugzilla

Bugzilla je open source software na zaznamenávání defektů, který byl vydán v roce 1998 jako jeden z prvních produktů neziskové organizace Mozilla Foundation. Bugzillu je

možné požívat zdarma pro jakékoliv účely. Napsána je v jazyce Perl, který je tím pádem potřeba mít nainstalovaný na serveru, aby mohla Bugzilla fungovat. Jako databázi lze využít MySQL, PostgreSQL a Oracle. Přestože se spousta jiných aplikací pro issue tracking dnes zaměřuje i směrem k podpoře řízení celého projektu, tým, který má Bugzillu na starosti, si klade za cíl zůstat pouze systém pro správu defektů. [4]

Uživatelské rozhraní je velmi jednoduché, přímočaré a při práci nevytváří žádné nadbytečné překážky. Navigace mezi jednotlivými stránkami aplikace je intuitivní. Mezi velké výhody Bugzilly patří její rychlost, na kterou se vývojový tým speciálně zaměřuje, a to pomocí minimalizování zbytečných volání do databáze a nenačítání nepotřebných dat. [4]



Obrázek 3 Bugzilla
Zdroj: [4]

Bugzilla nabízí dva typy vyhledávání. Lze vyhledávat klasicky full textově nebo taktéž pomocí složitějších dotazů, kde lze konkretizovat, co přesně a podle čeho chci vyhledávat. Typicky to může být vyhledávácí dotaz, kde chce uživatel ukázat všechny defekty, které byly zadány tento týden. Všechna vyhledávání, která uživatel provede, si může uložit. Takto uložená vyhledávání se potom ukazují v zápatí stránky jako odkazy, na které stačí kliknout a zobrazí se jejich výsledky. Uložená vyhledávání je možno sdílet s ostatními uživateli.[4]

Při zadávání defektů je nabízena automatická detekce proti duplikaci. Jakmile uživatel začne psát do pole Summary, v databázi se vyhledávají defekty se stejným textem a ukazují se na obrazovce. Tím zadavatel rychle zjistí, že je chyba v systému už zaevidována a nemusí to pomocí složitého hledání zjišťovat. Zároveň je to pomocný nástroj, pokud se jedná o znovu zanesenou chybu do systému s tím, že se tým může rovnou podívat, jak byla chyba v minulosti opravena. [4]

Od verze 3.0 je podporováno přádávání vlastních polí s libovolným obsahem. Pole lze přidávat k defektům a lze podle nich i vyhledávat. U pole je potřeba vyplnit jeho název, popis, typ, atribut, podle kterého bude umístěno na stránce, zda má být pole zobrazováno při zadávání nového defektu a taktéž zda se má pole zobrazovat v mailu, který se pošle po jeho vytvoření.

Typy polí, které lze přidat jsou několikařádkové pole, jednořádkové pole, multi-select, dropdown a pole s časem nebo datem. V případě, že je pole už vytvořené, nelze mu měnit název ani typ. U typů multi-select a dropdown lze měnit hodnoty, které jsou na výběr. Ostatní pole lze libovolně měnit. Smazat pole lze pouze, pokud nikdy nebylo v projekt použito a má nastaveno atribut, že je zastaralé. Jakmile je pole použito u nějakého defektu, už není možnost, aby bylo z projektu smazáno. Toto je ochrana pro zachování integrity celého projektu a defektů v něm. [4]

K dispozici je taktéž konfigurace workflow. Lze měnit stavy i přechody mezi nimi. Jediné omezení je nemožnost smazání či přejmenování stavu UNCONFIRMED. Tento stav je povoleno pouze zneaktivnit. Co se typu vyřešení defektů týká, nelze smazat ani přejmenovat duplikát (DUPLICATE). Všechny ostatní typy vyřešení omezení nemají.[4]

V Bugzille je možné k defektům nalinkovat vazbu na defekty, které je potřeba vyřešit před řešením daného defektu a taktéž je možné přidat defekty, které jsou daným defektem blokované. Dále je možné využít nepřímou cestu jako nalinkovat jeden defekt s druhým. Pro vytvoření vazby je nutné zadat komentář, do kterého lze vložit odkaz na defekt či na konkrétní komentář. Takový odkaz lze vytvořit pomocí napsání, na co se odkazujeme, a poté číslo dané položky. Příklad může vypadat např. takto:[4]

bug 123456

comment 789

Při zadání čísla, které se v databázi nenachází, se odkaz stejně vytvoří, ale bude to odkaz na stránku, která neexistuje. U defektu nebo komentáře, na který odkazujete, se nic o jeho nalinkování neukáže, „vazbu“ tedy lze vidět pouze na defektu, u kterého jste vytvořili komentář. Takový to postup pro linkování není ideální a lze lehce vytvořit špatný odkaz.[4]

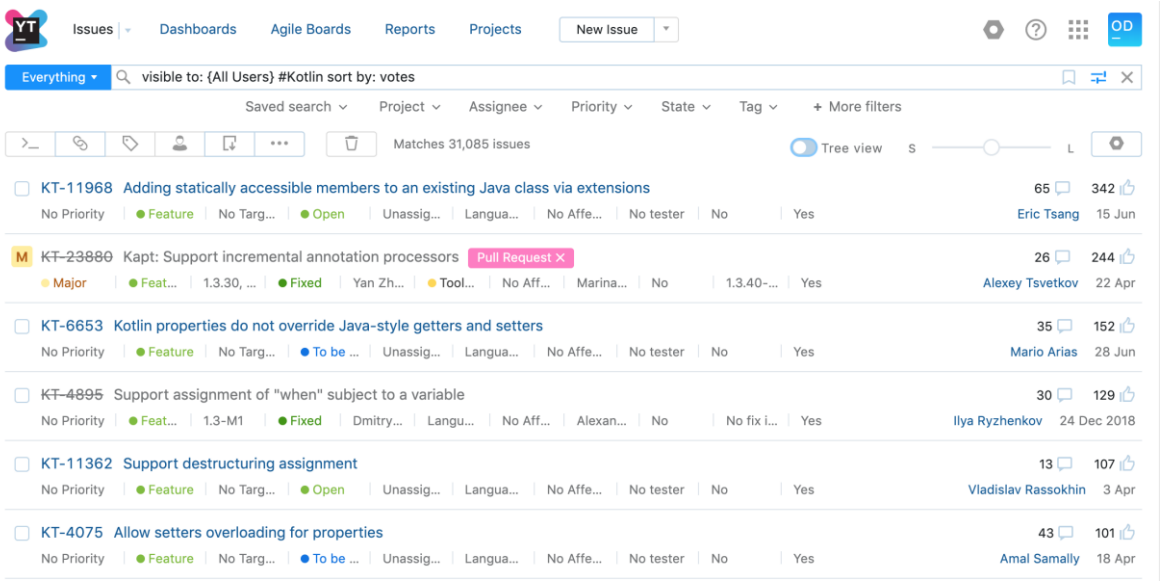
Lokalizace je nabízena v několika jazycích včetně češtiny, nicméně se jedná o práci komunity a jazykové mutace nemusí fungovat na všech verzích aplikace. Bugzilla je zabezpečena proti hrozbám jako je SQL injection a dokáže předcházet útokům XSS (Cross-site scripting).[4]

3.2.3 YouTrack

YouTrack je nástroj pro issue tracking a projektový management, který je stejně jako Jira zaměřen na vývoj v agilním prostředí. První verze byla vydána v roce 2009. Původní implementace byla v jazycích Java a JavaScript. Dnešní implementace již využívá místo Javy jazyk Kotlin. Vyvíjen je firmou JetBrains, mezi jejíž další produkty patří mezi vývojáři velmi populární IDE jako InetillJ IDEA, PhpStrom, WebStorm, nástroj na code review a týmovou kolaborací Upsource, CI/CD nástroj TeamCity a i programovací jazyk Kotlin. To zároveň zaručuje výbornou integraci mezi všemi těmito nástroji.[5][6]

Pro týmy nepřesahující počet tří uživatelů a open source projekty je možné využívat bezplatné licence. Pro komerční projekty s více jak třemi uživateli je nutné za licenci již platit. Začíná se na ceně 108,33 Kč za jednoho uživatele měsíčně při placení jednou ročně. Další ceny jsou odstupňované podle počtu uživatelů. Čím více osob systém využívá, tím méně se za jednotlivé uživatele platí. Jedná se o úrovně 3-99, 100-199, 200-299, 300-399 a 400+ uživatelů. Pokud počet uživatelů dosahuje čísla 1000, je cena za měsíc 43008,33 Kč při ročním placení.[5]

YouTrack disponuje velmi čistým a jednoduchým designem, ve kterém není problém se okamžitě zorientovat a naplno využívat všech funkcí, které aplikace nabízí. Na hlavní obrazovce se nachází hlavní menu, kde jsou odkazy na hlavní stránky v aplikaci, jako např. dashboardy, reporty a projekty. Pod menu je panel pro vyhledávání a dále už se na obrazovce nachází pouze seznam issue.



Obrázek 4 YouTrack
Zdroj: [5]

Panel pro vyhledávání má dva ovladatelné prvky. Prvním je vyhledávání pomocí kontextu. Ten slouží pro omezování vyhledávání pouze na specifickou množinu dat. Jako výchozí hodnota je vybráno vyhledávání ze všech issue. Na výběr je poté možnost vyhledávání omezit na specifický projekt, nebo štítky, které má uživatel uložené v oblíbených, případně vybrat omezení pomocí uloženého vyhledávání, které musí mít uživatel taktéž uložené v oblíbených položkách.[5]

Další ovladatelný prvek je klasické vyhledávací pole, kde si lze volně zadat, co hledáme nebo je možné použít metodu tzv. smart vyhledávání. To umožňuje do vyhledávacího pole psát vyhledávácí dotazy pomocí hodnot pro jednotlivá pole issue. Vyhledávací dotaz může vypadat např. následovně:[5]

created: {This week} project: Projekt #{Assigned to me} sort by: Priority desc

Tento dotaz vyhledá všechny tikety z projektu Projekt vytvořené tento týden, které jsou přiřazené na přihlášeného uživatele a seřadí je dle atributu priorita sestupně. Předpřipravené výrazy se zadávají do složených závorek, v tomto případě {This week}. Ve vyhledávacím dotazu lze použít i jiná vyhledávání, která byla v aplikaci uložena. Uložená vyhledávání se označují symbolem #. V příkladovém dotazu je použito #{Assigned to me}, které vyhledá vše přirazené na přihlášeného uživatel. Takováto vyhledávání lze používat samostatně, anebo jako je tomu v tomto případě zakomponovat je do jiného

vyhledávání. Vyhledávací dotaz může potom být složen pouze z již uložených vyhledávání a ten samý dotaz může být znovu uložen a znovu použit kdykoliv je potřeba. K dispozici je i našeptávač všech atributů, vyhledávacích parametrů i uložených dotazů, není třeba si nic pamatovat.

YouTrack poskytuje možnost přidání vlastních polí na issue. Při vytváření pole je nutné zadat název, typ, alias, zda může být pole prázdné, či privátní. Jako typ pole lze zvolit klasické datové typy jako jsou string, integer, date apod., anebo si lze vybrat enum typ s předdefinovaným listem možných hodnot. Typ pole ovlivňuje i to, jak je možné jeho hodnotu použít i jinde v aplikaci., např. pole s typem date lze využít při generování reportu jen za určitý časový úsek. Alias je využito jako reprezentace pole při vyhledávání. Při nastavení pole jako privátní stane se toto pole viditelné pouze pro uživatele, kteří mají oprávnění privátní pole číst a měnit. Tato funkce může být vhodná, pokud např. vyvíjíte software pro jinou organizaci, u které chcete, aby její zaměstnanci zadávali do vašeho YouTracku chyby, které naleznou. Můžete jim dát základní práva, se kterými privátní pole neuvidí. Jako viditelné pole budou mít jen název, popis a prioritu issue. Všechny ostatní pole jako stav, komu je issue přiřazen, komentáře atd. budou vidět jen členové vašeho týmu. [5]

U vytvořeného pole lze editovat typ a alias položku. Nově vybraný typ pole však musí být kompatibilní s aktuálním typem, jinak změnu nelze provést. Mazat lze všechny pole, a to i základní sadu, která je v každém projektu přednastavená. Při smazání pole, které je v projektu aktivně využito, se smaže ze všech issue i s hodnotami, které jsou v něm obsaženy a nelze tuto operaci vrátit zpět. [5]

Issue lze v YouTracku i smazat. Nicméně je potřeba být s touto možností velmi obezřetný. Jakmile je položka jednou smazána, není již v aplikaci cesta, jak ji obnovit. Lze se pokusit obnovit issue pomocí zálohy databáze, ale tato cesta není stoprocentní a není zaručené, že issue ze zálohy bude mít všechna data aktuální jako právě smazaná položka.[5]

Pro vytvoření spojení mezi položkami lze použít vazby. Vazby mají atribut, který určuje jejich směr:

- Directed – vyjadřuje vztah nadřízenosti jednoho issue nad druhým
- Undirected – vyjadřuje pouze přítomnost vztahu mezi dvěma issue
- Aggregation – vyjadřuje vztah, kde se jedná o kombinaci issue, např. duplikace

Jednotlivé vazby mají podle daného atributu názvy na jejich koncích. Pro atribut Directed to můžou být:[5]

- Depends on – „závisí na“- než je možné pracovat na daném issue, je potřeba vyřešit jeho nadřízenho
- Is requried for – „je vyžadováno“ pro-aby bylo možné začít práci na podřízeném issue, je potřeba toto vyřešit

Jelikož atribut Undirected vyjadřuje pouze obecný vztah, je možné mít jen jednu vazbu:[5]

- Relates to – „souvisí s“-dvě issue spolu souvisí, ale jsou na sobě nezávislé

Atribut Aggregation má několik názvů použitelných pro vazby. Konkrétně to jsou:[5]

- Duplicates – „duplikuje“- označuje, že daný issue je duplikát k propojenému issue
- Is duplicated by – „je duplikován“- dané issue je duplikované
- Subtask of – „pod úkol“- jedná se potomka
- Parent for – „rodič“- rodič, který se dělí na další issue

Na detailu issue se vždy zobrazuje, s jakými dalšími položkami má vztah. Taktéž je možnost v aplikaci vyhledávat položky, které jsou spolu propojené.[5]

Podporována je i možnost úprav a vytváření svých vlastních workflow. V YouTracku jsou workflow složená z tzv. typů pravidel. Na výběr je celkem z pěti kategorií:[5]

- On-change – vykonává se při změně issue
- On-schedule – vykonává se ve stanovený čas
- State-machine – nastavení změn hodnot u políček issue
- Action – provedení akce jako reakce na uživatelův příkaz

- Custom – vytvoření vlastního pravidla

Všechny pravidla jsou napsaná v JavaScriptu a uživateli je umožněno je psát přímo v tomto jazyku. To dává týmům velkou volnost v tom, jak chtějí, aby se workflow chovalo. Nicméně je potřeba, aby člověk, co chce pravidla měnit, měl znalost jazyka daného jazyka. Je možnost vytvářet zcela nová pravidla nebo upravovat již stávající řešení. Přímo v YouTracku je implementován editor pro JavaScript, není tedy nutné pro psaní pravidel používat externí editor. Samotná workflow se poté skládají z vytvořených pravidel. Workflow i samotná pravidla je možné editovat i smazat bez jakéhokoliv omezení, i když jsou momentálně používána v projektu.[5]

Lokalizace je dostupná v několika jazykových mutacích včetně češtiny. Ta je nicméně neoficiální, vytvořena komunitou a JetBrains nezaručuje, že je v aplikaci přeloženo vše a že lokalizace bude fungovat na všech verzích.[5]

3.2.4 Azure DevOps

Azure DevOps je aplikace zaměřené na podporu celého životního cyklu vývoje softwaru. Jedná se o produkt od firmy Microsoft, vyvíjený od roku 2005 a až do roku 2018 nesoucí název Visual Studio Team Foundation Server. Je možné ho využívat pro plánování projektu jako systém pro verzování a revize kódu, CI/CD a zároveň i jako testovací nástroj. Aplikace má skvělou integraci s dalšími produkty a technologiemi od Microsoftu. Aplikace je rozdělena do pěti hlavních kategorií, kterými jsou Azure Boards, Azure Pipelines, Azure Repos, Azure Test Plans, Azure Artifacts.[7][8]

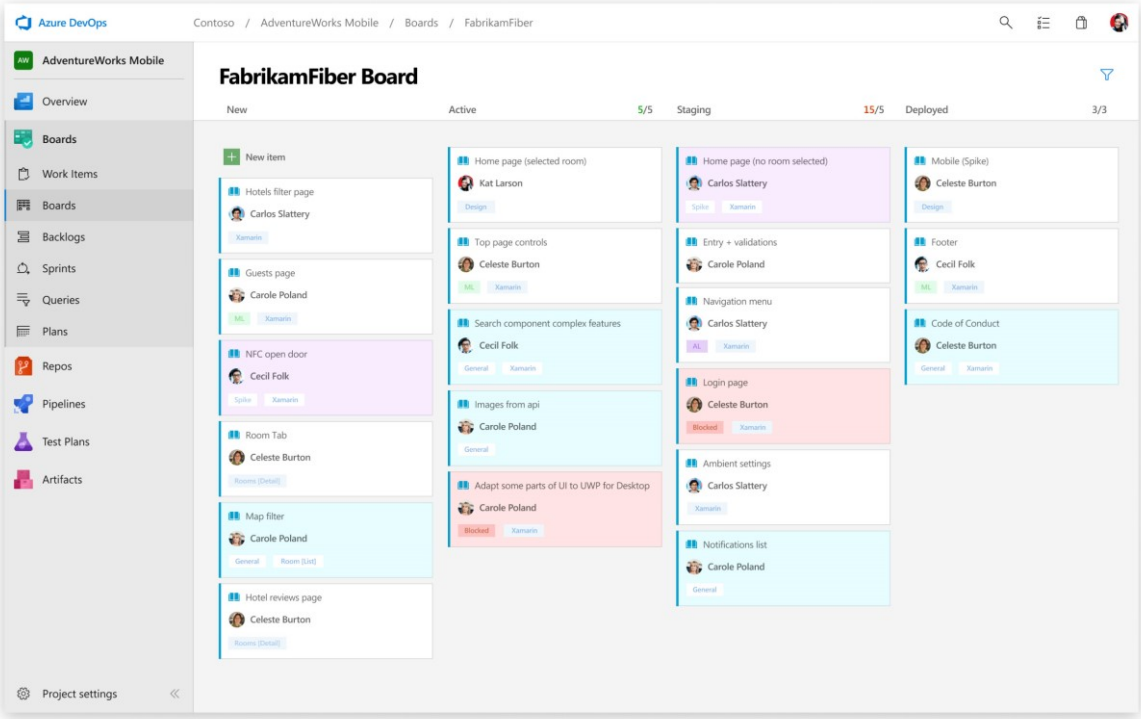
Kategorie Azure Boards poskytuje podporu pro řízení projektu pomocí sledování úkolů, user stories ve sprintu nebo např. požadavků v backlogu. V základu jsou nabízeny dashboardy pro metodologie Scrum a Kanban, ale lze je dále konfigurovat dle týmových potřeb. Na výběr je řada widgetů a grafů informujících o aktuálním stavu projektu, které lze na dashboard umístit a na první pohled mít přehled o nejdůležitějších metrikách, které tým při vývoji zajímají. Součástí Boards je i integrovaná část pro reportování výsledků a průběhu projektu, která je nápomocná především pro vedoucí členy týmů.[8]

Azure Pipelines je určeno pro průběžnou integraci a pro rychlé, jednoduché a bezpečné nasazování vyvíjené aplikace a její automatické testování. Je zde podpora mnoha jazyků, jakými jsou Python, Java, PHP, C# a další.[8]

Azure Repos je nástroj pro verzování zdrojových kódů, který podporuje dva typy verzovacích systémů, kterými jsou Git, nejrozšířenější systém pro distribuované verzování, a Team Foundation Version Control, což je vlastní centralizované řešení od Microsoftu. [8]

Sekce Azure Test Plans je určeno pro využití při plánování a exekuci testování softwaru. Lze vytvářet manuální testy, které lze řadit do testovacích sad a k testům lze taktéž přiřazovat automatické testy. Test Plans nabízí bohatou nabídku grafů a reportů pro sledování průběhu testů.[8]

Poslední kategorií v Azure DevOps je Artifacts. Jedná se o nástroj sloužící k lepší organizaci a přístup k balíčkům pro Maven, npm a NuGet, kde je lze vytvářet nebo stahovat. Přípravné balíčky je poté možné využívat v Piplines pro CI/CD.[8]



Obrázek 5 Auzre DevOps
Zdroj: [7]

Microsoft nabízí několik možností licencování, nicméně některé umožňují koupi pouze jednotlivých služeb a těm se zde nebudeme věnovat. Zbývají tedy dvě varianty. První varianta obsahuje kategorie Azure Boards, Pipelines, Repos a Artifacts a pro prvních 5 uživatelů je zdarma. Za každého dalšího uživatele je již potřeba zaplatit 6 $ měsíčně. Druhá varianta, kterou si lze vybrat, obsahuje navíc i kategorii Test Plans. Tato verze je placená od začátku a cena za jednoho uživatele je 52 $ za měsíc. Součástí služby je i

přístup k Visual Studio Marketpalce, kde je možné zdarma stahovat čí nakupovat rozšíření a doplňující aplikace, které je možné do Azure DevOps integrovat jako např. Slack, SonarQube nebo FlowViz. [7][9]

Základním prvkem je tzv. work item, česky lze přeložit jako pracovní položka, který může mít různé typy. Lze se setkat s těmito typy:

- Epic
- Feature
- User Story
- Task
- Bug
- Issue

Epic je celistvý a rozsáhlý úkol, který nelze stihnout v rámci jednoho sprintu a je nutné ho rozdělit na menší části. Může se jednat např. o úkol implementovat celý issue tracking systém. Feature je část Epicu, která je mnohem detailněji popsána, nicméně stále se jedná o moc velký kus práce, který by nešlo v rámci sprintu stihnout, např. po přihlášení chce uživatel vidět všechny issue, které jsou na něm momentálně přiřazeny. User story je část Feature, která už je tak dostatečné malá, že ji lze implementovat v rámci jednoho sprintu. Jako příklad lze uvést, že uživatel se chce při kliknutí na issue dostat na jeho detail. Task představuje konkrétní úkol, který je potřeba splnit, aby bylo možné implementovat dané User story. Problém, který nějakým způsobem zabraňuje dokončení User story, je v Azure DevOps označen jako Issue. Defekt ve vyvíjeném systému je poté zadán jako Bug.[8]

Jaké budou v projektu dostupné typy, záleží na tom, jaký je vybrán proces. U základního procesu jsou k dispozici typy Epic, Issue a Task. V agilním procesu je k dispozici více typů, konkrétně Epic, Feature, User Story, Task, Bug a Issue.[8]

Ke všem je možné přidávat dodatečná pole. Ta musí mít unikátní jméno a lze si vybrat z několika datových typů, jako např. text, číslo nebo checkbox. U pole lze vybrat, zda je povinné ho vyplnit, a zároveň je možné nastavit jeho výchozí hodnotu. Při ukládání pracovní položky musí být vždy všechna povinná pole vyplněna. Nevýhodou se může zdát nemožnost vytvořené pole měnit. V takovém případě je nutné dané pole smazat a znovu vytvořit takové, které bude splňovat požadavky.[8]

Mimo představené základní typy pracovních položek, se kterými se v Azure lze setkat, je nabízena i možnost vlastních dodatečných, které lze přidat do projektu. Při tvorbě lze vybrat, jaký název pracovní položka ponese, jeho popis, ikonku a její barvu. Vytvořená pracovní položka má v základu vždy stejná pole. I zde lze přidat dodatečná pole.[8]

I workflow je možné podrobit konfiguraci. Každé workflow má čtyři kategorie stavů, kde v každé kategorii musí být vždy alespoň jeden aktivní stav. Kategorie jsou:

- Proposed – navržené
- In Progress – probíhající
- Resolved – vyřešené
- Completed – dokončené

Stavy, které jsou defaultně nabízeny nelze měnit ani je smazat. Tyto stavy jsou:

- New – nově vytvořená pracovní položka, v kategorii Proposed
- Active – práce probíhá, v kategorii In Progress
- Resolved – vyřešená pracovní položka, v kategorii Resolved
- Closed – zavřená pracovní položka, v kategorii Completed

Základní stavy je možné pouze zneaktivnit, pokud ovšem je ve stejné kategorii aktivní stav. Nový stav lze vytvořit v jakékoliv kategorii. U stavu je nutné uvést jeho název, do jaké kategorie stavů spadá a jeho barvu. Jakmile je stav vytvořen, je možné ho okamžitě vybrat pro již vytvořenou pracovní položka v projektu. Nelze však měnit jeho název, pouze kategorii a barvu. Uživatelem vytvořené stavy lze oproti stavům základním i smazat. Při smazání či zneaktivnění stavu se pracovní položky v projektu, které se momentálně v daném stavu nachází, nemění. Pořád mají daný stav vybraný. V okamžiku, kdy chce uživatel něco na pracovní položce změnit, musí zároveň i změnit stav a vybrat takový, který je v projektu momentálně aktivní, jinak nelze provedené změny uložit. Aby nebylo nutné takto všem položkám ručně stav měnit, je možnost na tuto operaci vytvořit příkaz, který to provede automaticky.[8]

Obrazovka detailu pracovní položky je rozdělena do několika záložek. Detaily, kde jsou zobrazeny nejzákladnější informace typu název, komu je položka přirazena, stav, popis, priorita atd. Záložka Historie ukazuje veškeré změny, kterými položka prošla. Třetí

záložka ukazuje, jaké další pracovní položky jsou nalinkované na právě prohlíženou položku a poslední záložka jsou připojené přílohy.[8]

Obrazovku detailu lze taktéž modifikovat. Za prvé lze měnit pozici polí, která se na obrazovce zobrazují. Taktéž je možné vytvořit novou záložku, na kterou lze zobrazovat data, která jsou pro práci na projektu potřeba. Takto vytvořené záložky je možné i smazat, nicméně nelze smazat základní záložku Detaily a taktéž tato záložka musí být vždy jako první. Všechny ostatní záložky na obrazovce jsou až za ní.[8]

Pro pracovní položku je možné změnit její typ. To lze provést velmi jednoduše a není zde nastaveno žádné omezení. Stačí pouze vybrat nový typ a uložit. Pro odstranění položky jsou na výběr dvě možnosti. Při nastavení stavu Removed se položka nezobrazuje mezi ostatními a pokud ji chcete vyhledat, je nutné specifikovat, že hledáte položky s daným stavem. Druhá možnost je již opravdové smazání. Při odstranění se položka vloží do koše. Zde ji buď lze obnovit a tím vrátit zpátky do projektu, či vybrat možnost permanentního smazání a tím jí odstraníte bez možnosti návratu.[8]

Jednotlivé pracovní položky se dají propojit vazbami a zobrazit tak jejich vztah. Vazby, kterými je lze spojit jsou následující:

- Duplicate – duplikát
- Parent/Child – rodič/potomek
- Predecessor-Successor – předchůdce-nástupce
- Related – související

Pokud se v projektu vyskytuje dvojice položek, které zachycují stejnou informaci, měly by být označeny vazbou duplikát, jednu z nich zavřít a ponechat jednu aktivní, se kterou se bude dále pracovat.[8]

Propojení rodič/potomek se primárně využívá na ukázání vztahu, kdy položka patří pod jinou položku. Mělo by být použito, když máme např. pracovní položku Epic, který představuje rodiče a tato položka byla rozdělena na několik položek typu Feature, které jsou v tomto vztahu potomky. Díky tomuto spojení můžeme lehce sledovat, jaké položky je třeba vykonat, aby nadřazený rodič byl splněn.[8]

Další propojení, předchůdce-nástupce se využívá ve chvíli, kdy je potřeba, aby byla jedna položka splněna před tím, než se začne pracovat na další. Položka, kterou je potřeba udělat jako první, označíme jako předchůdce a druhou jako nástupce.[8]

Posledním vztahem mezi položkami je vazba související. Ta pouze vyjadřuje určité, spíše volné spojení dvou či více položek, jejichž vztahy nesplňují definici výše zmíněných vazeb.[8]

Pracovní položky lze vyhledávat velice rychle, a to buď pomocí full-textového vyhledávání, či hledání podle specifického pole. Ve full-textovém vyhledávání lze zadat text a ten je vyhledáván přes všechny políčka pracovních položek a vyhledává i podobná slova tomu, které uživatel zadal. Pokud uživatel zná ID pracovní položky, stačí ho zadat do vyhledávajícího pole a položka je rovnou vyhledána. Druhou možností je vyhledávání pomocí políček. Pro vyhledávání tímto způsobem stačí napsat název pole, dle kterého chceme vyhledávat, a hodnotu, které by dané pole mělo mít. V jednom vyhledávání je i možnost kombinace více polí. Vyhledávácí příkaz může vypadat např. následovně:[8]

CreatedDate = @StartOfMonth Assignedto:@Me State:active

Tento příkaz vyhledá všechny pracovní položky přiražené na přihlášeného uživatele, které byly vytvořeny od začátku aktuálního měsíce a momentálně jsou aktivní.

Pro rychlejší vyhledávání lze používat i zkratky pro dané pole. Zkratky jsou k dispozici pro tyto pole:[8]

Pole	Zkratka
Assigned to:	a:
Created by:	c:
State	s:
Work item type	t:

Tabulka 1 Azure DevOps zkratky
Zdroj: [8]

Vyhledávající dotaz potom může mít tvar:

a:@Me s:active t:feature

Příkaz vyhledá všechny pracovní položky typu Feature, které se momentálně nachází ve stavu aktivní a jsou přiřazené na aktuálního uživatele.

Do vyhledávajících dotazů lze taktéž přidat logické operátory AND, OR, NOT.

1. issue AND tracker
2. issue OR tracker
3. issue NOT tracker

První příkaz vyhledá všechny položky, které obsahují slova issue a zároveň tracker. Druhý příkaz vyhledá všechny položky, které obsahují slovo issue nebo tracker. Poslední příkaz vyhledá všechny položky, kde se nachází slovo issue, ale neobsahují slovo tracker.[8]

Poslední možností jak upravit vyhledávání je pomocí znaků * a ?:

1. issue*
2. issue?tracker

První příkaz vyhledá všechny položky, které na začátku obsahují slovo issue, např. issueTracker, issueSolved. Druhý příkaz vyhledá všechny záznamy, kde je slovo issue první, poté jakýkoliv znak a na konci slovo tracker.[8]

V současné době psaní práce není pro Azure DevOps k dispozici oficiální česká lokalizace. Týmy tedy musí využívat základní anglickou verzi.[8]

3.3 Použité technologie

V této kapitole budou představeny technologie, které budou v praktické části práce použity pro implementaci vlastního řešení issue trackeru. Jelikož se jedná o webovou aplikaci, bude rozdělena na dvě části. Server-side část neboli backend je aplikace, která běží na webovém serveru. Client-side neboli frontend je poté část aplikace, která běží na zařízení uživatele, v tomto konkrétním případě ve webovém prohlížeči.

3.3.1 Java

Java je objektově orientovaný programovací jazyk, který byl poprvé představen již v roce 1995, tehdy vyvíjený firmou Sun Microsystems, dnes již pod hlavičkou Oracle.

Hlavním důvodem pro jeho vznik byla snaha o jazyk, který není závislý na platformě, na které běží, a dal by se použít pro vývoj softwaru pro nejrůznější hardwarová zařízení. Rozmach Javy přišel především s vývojem internetu, pro který se Java jako přenositelný jazyk velmi hodila a zajistila si tím obrovskou popularitu, která přetrvává dodnes. Dle průzkumu na webu Stack Overflow byla Java druhým nepopulárnějším programovacím jazykem za rok 2019, kde se na první příčce umístil JavaScript. Stejně dopadl průzkum za stejný rok i od firmy JetBrains. [5][14][15]

Platforma Javy se dělí na několik části. Dvě neznámější a nejhojněji využívané jsou Java Standard Edition a Java Enterprise Edition. Java SE je primárně určená pro vývoj desktopových a serverových aplikací, zatímco Java EE se používá pro implementaci podnikových aplikací. Dalšími části platformy jsou např. Java TV, která určená pro vývoj aplikací pro televize a set top boxy, anebo Java Micro Edition pro aplikace pro vestavěné a mobilní zařízení pro IoT.[17]

Jako základní přednosti, které z Javy dělají tak lákavý jazyk, tým odpovědný za její návrh a vývoj shrnul takto:[14]

- Jednoduchost
- Bezpečnost
- Přenositelnost
- Objektově-orientovaný
- Robustnost
- Víceválcové programování
- Architektonicky neutrální
- Interpretovatelnost
- Vysoká výkonnost
- Distributovatelnost
- Dynamičnost

Důvodem, proč můžou být programy napsané v jazyce Java přenositelné na různé platformy, je Java Virtual Machine (JVM), což je virtuální stroj, na kterém samotný program běží. Výstupem Java není spustitelný kód, který poběží na počítači či serveru, ale bajtový kód, který je právě určen pro běh na JVM. Kdyby se Java rovnou kompilovala do spustitelného kódu, znamenalo by to, že pro každou platformu bychom potřebovali různé

verze daného programu. Místo toho nám stačí, aby na platformě běžel JVM a tím bezpečně víme, že tam lze spustit aplikaci napsanou v Javě.[14]

Další nespornou výhodou toho, že aplikace běží v prostředí JVM, je taktéž zvýšení bezpečnosti celého procesu, jelikož virtuální stroj může zamezit běžícímu programu ve vykonávání podezřelých operací.[14]

Nevýhodou tohoto přístupu by se mohla zdát pomalejší rychlost programů, jelikož místo toho, aby běželi přímo ve spustitelném kódu, je potřeba, aby je spouštěl JVM a teprve ten vykonával samotné instrukce. Program je sice vykonáván o trochu pomaleji, ale není to zas o tolik znatelné, aby to přebilo výhodu přenositelnosti Java aplikaci.[14]

Jak již bylo zmíněno, jedná se o objektově orientovaný jazyk. To znamená, že při vývoji je snaha v programu vytvářet objekty z reálného světa. Ty se vytvářejí pomocí šablon, které se nazývají třídy. Každá třída definuje, jaký bude mít jí vytvořený objekt strukturu, tedy jaká data bude mít, a chování, jež je reprezentováno pomocí metod, které má. Z podpory objektově orientovaného konceptu má Java stejně jako každý další jazyk, který je podporuje, tři základní rysy. Zapouzdření, dědičnost a polymorfismus.[14]

Zapouzdření je princip ukrytí dat třídy před vnějším kódem tak, aby byly chráněny před nesprávným použitím a neoprávněným zásahem. Třída má atributy, které jsou viditelné jen uvnitř třídy a pro každý atribut jsou vystavené metody, které vrací aktuální hodnotu atributu a které mohou hodnotu atributu nastavit. Pokud chce okolní kód pracovat s těmito atributy, má možnost k nim přistupovat pouze přes tyto metody. Tím, že se hodnota atributu nastavuje jen přes metodu, kterou vystavila daná třída, nemůže se stát, že se do metody pošle špatný typ dat, a tím jsou data uvnitř chráněna. Metody pro získání dat se nazývají gettery a metody pro nastavování dat se nazývají settery.

S tím souvisejí i modifikátory přístupu. Modifikátory lze nastavovat pro třídy, metody a atributy, přičemž je možné mít nastavený vždy pouze jeden modifikátor. V Javě jsou k dispozici modifikátory:[18]

- public – přístup má libovolný kód z celého programu
- protected – přístup má daná třída, potomci třídy a třídy z package, kde se třída nachází
- žádný modifikátor (package-private) – přístup má daná třída a třídy z package, kde se třída nachází
- private – přístup má pouze daná třída

Dle dokumentace Oraclu by se programátoři měli snažit dávat vždy co nepřísnější modifikátor. Pokud nemáte dobrý důvod postupovat jinak, měli byste použít modifikátor private.[18]

Dalším podstatným konceptem objektově orientovaných jazyků je dědičnost. Jak již název napovídá, jedná se o techniku, kdy jedna třída může dědit vlastnosti třídy jiné. Taková třída se pak nazývá potomek nebo také podtřída, a třída, od které dědí, rodič nebo nadtřída. Díky tomu má podtřída všechny vlastnosti, co rodič, a programátor tedy nemusí psát všechny atributy a metody pro novou třídu znovu. Zároveň však, pokud je potřeba některou metodu změnit nebo třídu rozšířit o nový prvek, není to problém.

S dědičností úzce souvisí i třetí princip objektově orientovaných jazyků a to polymorfismus. Ten bývá velmi často popisován obratem „jedno rozhraní, více metod". Jedná se o to, že v aplikaci můžeme mít rozhraní, které má sadu metod. Od tohoto rozhraní dědí řada tříd a každá si dané metody upraví podle svého účelu. Některou z těchto metod poté nad rozhraním zavoláme a podle toho jaké parametry jsme dané metodě poskytly, program za běhu rozhodne, jako metodu z podtříd se pro dané volání použije. To je výhoda pro programátora, který tak nemusí sám ručně určovat, jaká konkrétní implementace metody by se měla zavolat.

3.3.2 Spring

Pro vývoj backendové části aplikace v této práci bude využit Spring Boot, který patří do rodiny open source frameworků jazyka Java primárně určených pro tvorbu podnikových aplikací. Spring obsahuje desítky modulů a je na vývojáři, které z nich chce ve své aplikaci využít. Spring Boot svými vlastnostmi zajišťuje rychlý vývoj. Toho je docíleno pomocí minimální potřeby konfigurace ze strany vývojáře. Všechno už je v základním balíčku přednastaveno a není potřeba s tím ztrácet čas. Nicméně je samozřejmě i možné si veškeré nastavení nakonfigurovat sám podle svých potřeba, a i toto nastavování je značně ulehčeno oproti konkurenčním řešením. Spring Boot navíc v sobě obsahuje servery Tomcat, Jetty a Undertow a aplikaci je tedy možné okamžitě rozběhnout bez potřeby jejího nasazování.[10]

Mimo samotného Spring Boot budou v aplikaci využité i další moduly Spring. Pro objektově relační mapování, které se v Javě nazývá JPA a slouží pro zjednodušení práce

s databázovými operacemi a zároveň udržování konzistence mezi entitami v databázi a třídami v Java kódu, bude využit Spring Data JPA. Tento modul využívá jako základní ORM Hibernate.[10]

Pro zabezpečení aplikace bude využit modul Spring Security, který je de facto standardem pro zabezpečení aplikací postavených na Spring frameworku. Spring Security nabízí ochranu proti útokům fixací session prohlížeče, clickjacking, cross site request forgery a mnohé další. Lze jednoduše řídit autorizaci a autentizaci aplikace, využívat uživatelské role a řídit přístupy pro jednotlivé URI.[10]

3.3.3 Maven

Nástroj Maven je open source projekt od Apache, určen pro automatické sestavování aplikací. Používaný je primárně pro projekty psané v jazyku Java. Založen je na konceptu hlavního souboru POM, kde je v jazyce XML nakonfigurováno, jak se má projekt sestavit a jaké knihovny se mají pro správnou funkčnost aplikaci dotáhnout ze sdílených repositářů. Hlavním cílem Mavenu je udělat sestavování aplikací jednoduché, unifikované a podpořit tím rychlý vývoj softwaru.[11]

3.3.4 SQL

SQL je strukturovaný dotazovací jazyk určený pro práci s daty, která jsou uložena v relačních databázích. Poprvé se představilo již v roce 1974 a první standard jazyka byl publikován v roce 1986.[20]
Jazyk obsahuje pět hlavních kategorií příkazů pro práci s daty:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

DDL je kategorie příkazů, kterými se vytváří a upravuje samotná databáze a taktéž tabulky v ní a jejich struktura, indexy a pohledy. Patří sem příkazy:

- CREATE – vytvoření databáze, tabulky, indexu či pohledu
- ALTER – úprava struktury tabulky, indexu či pohledu
- RENAME – přejmenování databáze, tabulky, indexu či pohledu
- DROP – smazání databáze, tabulky, indexu či pohledu
- TRUNCATE – smazání dat z tabulky, ale ponechání samotné tabulky

DML jsou příkazy, kterými se manipuluje se samotnými daty uvnitř tabulek v databázi. Patří sem příkazy:

- INSERT – vložení nových řádků do tabulky
- UPDATE – změna hodnot uložených v tabulce
- DELETE – smazání hodnot v tabulce
- LOCK – změna přístupu k určité tabulce
- MERGE – vložení nového řádku do tabulky či jeho změna, pokud takový řádek v databázi již existuje

V kategorii DQL se nachází pouze jeden příkaz a to:

- SELECT – výběr dat z databáze a jejich zobrazení uživateli

DCL příkazy jsou určeny pro správu práv uživatelů k práci s tabulkami. Příkazy v této kategorii jsou:

- GRANT – přidělení oprávnění uživateli
- REVOKE – odebrání oprávnění uživateli

Poslední kategorií příkazů jazyka SQL jsou TCL, které jsou určeny pro řízení transakcí v databází. TCL příkazy jsou:

- COMMIT – uložení změn databáze provedených uživatelem
- ROLLBACK – smazání změn databáze provedených uživatelem

- SAVEPOINT – uložení specifického stavu databáze

3.3.5 MySQL

Pro databázovou část aplikace bude využito MySQL. Jedná se o nepoužívanější open source relační databázi na světě. Vyvíjená je již od roku 1995 a je to spolehlivý databázový systém s vynikající výkonností a velmi snadnou použitelností. Díky své rozšířenosti disponuje skvělou integraci i s frameworky z Java světa a lze ji tedy ideálně využít se Springem. [12]

3.3.6 TypeScript

TypeScript je open source programovací jazyk vyvíjený firmou Microsoft. Jedná se nadmnožinu jazyka JavaScript, což znamená, že má totožnou syntax i sémantiku a validně napsaný soubor v JavaScriptu je taktéž validní v TypeScriptu. Soubor je kompilovaný do čistého JavaScriptu. To je obrovská výhoda pro vývojáře, kteří již jsou dobře seznámení s JavaScriptem a nemusí se tak učit kompletně nový jazyk. To hlavní, co přináší, je podpora statické typové kontroly, třídy a moduly.[16]

Validně napsaný program, ještě neznamená, že se vám v TypeScriptu neukáží varování nebo chybové hlášení. To může být dáno rozdíly mezi těmito dvěma jazyky a může to být třeba právě kvůli statické typové kontrole. Nicméně i přesto se takový kód zkompiluje.[19]

Statická typová kontrola jazyka znamená, že jsou automaticky kontrolované typy hodnot, aby tak nedošlo k nevalidnímu nastavení proměnné, která má jiný typ. Této kontrole se lze vyhnout deklarováním tzv. dynamických proměnných. Jinak je možné proměnným nastavit primitivní typy, array, funkce, alias či jakoukoliv jinou komplexní strukturu včetně třídy a rozhraní. Oproti mnoha jiným jazykům, např. Javě, se v TypeScriptu při deklaraci proměnné píše její typ až za název:[19]

```
const projectName: string = 'Issue tracker';
const age: number = 56;
```

V jazyku jsou k dispozici tyto primitivní typy:

- string – sekvence UTF-16 znaků

- boolean – true nebo false
- number – 64-bit číslo s desetinnou čárkou s přesností na dvě desetinná místa
- symbol – jedinečný neměnný symbol

Taktéž je možné se setkat s typy, které představují speciální hodnoty:[19]

- undefined – proměnné ještě nebyla přirazena hodnota
- null – nepřítomnost hodnoty proměnné
- void – hodnota neexistuje, např. při funkci, která nic nevrací
- never – nedosažitelná část kódu

Všechny ostatní typy, které nejsou primitivní jsou poté podtřídou typu objekt. V jazyku je možné ještě využít typ any, který lze použít k reprezentaci kteréhokoliv typu. Je tedy teoreticky možné všechny proměnné opatřit tímto dynamickým typem. U něj však nedochází k automatické kontrole při kompilaci a jeho použitím tedy do jisté míry potlačujete primární výhodu použití samotného jazyka oproti JavaScriptu.[19]

Před příchodem standardu EMCAScript 6 nebyly v JavaScriptu přítomné třídy, což mohlo pro spoustu programátoru, kteří byli z jiných jazyků zvyklí na objektově orientovaný přístup k vývoji, představovat menší potíže. Proto se vývojáři TypeScriptu rozhodli do jazyka třídy zařadit. Třídy jsou v jazyce velmi podobné třídám např. v jazyce Java. Mohou obsahovat specifické konstruktory, mít své pole a metody, a jednotlivé třídy se mohou dědit.[16]

Moduly jsou taktéž věcí, která byla v JavaScriptu představená v EMCAScript 6 a tuto techniku obsahuje i TypeScript. Díky modulům lze proměnné, třídy, funkce atd. zapouzdřit do jejich vlastního jmenného prostoru. Oproti klasickým jmenným prostorům (v angl. namespace), které pouze redukují množství věcí, které přidávají do globálního jemného prostoru, moduly do něj nepřidávají vůbec nic. Části kódu deklarované uvnitř modulu nejsou mimo svůj vlastní jmenný prostor dosažitelné. Při potřebě jejich použití je nutné je explicitně exportovat, a poté na místě jejich zamýšleného použití importovat. Jakýkoliv soubor v TypeScriptu, který má na začátku import nebo export výraz, je považován za modul. Pokud soubor není modulem, je bez omezení dostupný v globálním jmenném prostoru. Moduly jsou hlavním nástrojem pro škálování velkých programů.[16][19]

3.3.7 Angular

Angular bude použit pro implementaci frontendu aplikace. Angular je open source framework vyvíjený společností Google. Je postaven na jazyce TypeScript a primárně určen pro vývoj single-page webových aplikací, což jsou aplikace, kde při kliku uživatele není nutná aktualizace celé stránky, ale obsah se načítá dynamicky, což zlepšuje pocit při práci s webem. Jeho předchůdcem byl framework AngularJS, který byl napsán v jazyce JavaScript. Google se při vývoji druhé verze AngularJS rozhodl celý framework přepsat do TypeScriptu a od verze Angular 2 se již jedná o jiný, samostatný framework, který se dnes nazývá pouze Angular.[13]

Základem frameworku jsou tzv. komponenty. Jedná se o základní stavební bloky uživatelského rozhraní celé aplikace a ty představují prvky stránky jako např. menu, přihlašovací dialogové okno nebo i celou stránku. Samotné komponenty jsou vždy rozděleny na oddělené soubory s HTML, kde jsou popsaný prvky, které se budou v komponentě nacházet, soubory s CSS, kde je pomocí kaskádových stylů popsána grafická úprava komponenty, a soubory s TypeScriptem, kde je implementována potřebná logika pro správnou funkčnost dané komponenty.[13]

Komponenta se definuje pomocí anotace @Component. Většinou má specifikované atributy selector, templateUrl a styleUrls. Selector je identifikátor samotné komponenty. TemplateUrl představuje cestu k HTML šabloně komponenty a styleUrls jsou cesty k CSS souborům komponenty. Životní cyklus komponent, tedy vytváření, vykreslování, reakce na změny či zničení, to vše je řízeno samotným Angularem, který na to poskytuje rozhraní, která mohou komponenty implementovat. Vývojářům je k dispozici několik metod životního cyklu. Nejvíc se lze setkat se dvěma:[13]

- ngOnInit
- ngOnDestroy

Metoda ngOnInit se volá vždy pouze jednou, a to při inicializaci komponenty a její dokončení znamená, že vytvoření je hotové a vstupní vlastnosti komponenty jsou nastavené. V TypeScriptu má každá třída vlastní konstruktor, u kterého by se mohlo zdát, že tvoří dost podobnou funkci jako ngOnInit, nicméně konstruktor je v objektově orientovaném programování určen pro vytvoření instance třídy, zatímco ngOnInit je jedna

z funkcí životního cyklu Angularu, které nám umožňují mít specifický kód v různých fázích života komponenty. [13]

Předtím, než je komponenta zničena, by měla být zavolána metoda ngOnDestroy, která je důležitá pro vyhnutí se přetečení paměti. K tomu může dojít především pokud neprovedeme odhlášení od obesrvable. [13]

Všechny metody životního cyklu jsou implementovány v jednotlivých rozhraních. Konkrétně metoda ngOnInit implementuje rozhraní OnInit a ngOnDestroy implementuje OnDestroy. Z čistě technického hlediska jsou importy těchto rozhraní nepovinné. V JavaScriptu rozhraní neexistují, a proto při zkompilování kódu nemá Angular možnost, jak při běhu program zkontrolovat. Pokud nejsou importy přítomny, všechny metody životního cyklu komponent budou stejně vykonány, ale minimálně pro přehlednost a kompletnost kódu se doporučuje importy rozhrání přidávat všude, kde jsou dané metody využívané.[13]

Koncept observable slouží především k přenosu dat od vydavatele k odběrateli. Princip je, že o tom, kdy odběratel dostane data, která si vyžádal, nerozhoduje on sám, jak by se to mohlo zdát logické a jak je to také implementované v mnoha jiných konceptech v samotném JavaScriptu, ale je to právě vydavatel, který má rozhodovací slovo v tom, kdy svoje data k odběrateli zašle. Toto má svoje využití především při předávání HTTP požadavků v aplikaci.

Aby bylo možné observable využívat, je nejprve nutné vytvořit si prvek, který bude reprezentovat vydavatele. V něm je potřeba implementovat metodu subscribe, díky které se budou moct odběratelé přihlásit k odběru dat. Pokud komponenta chce data odebírat, zavolá si danou metodu a tím získá objekt, který se v nazývá observer. Ten obsahuje informace o tom, jak budou data zasálána. Když se odběratel přihlásí k odběru, vydavatel mu začne zasílat požadovaná data.[13]

Prvkem, který se v Angularu taktéž hojně používá jsou služby. Pod tímto pojmem si lze představit funkce, které jsou v aplikaci využívané. Převážně se jedná o třídy, které mají specifický úkol a pouze ten řeší. Typicky se jedná o požadavky typu vyžádání si dat z backendu, validace formulářových polí, logování záznamů do konzole. Tyto funkce se používají či jsou potřeba v mnoha komponentách. Místo postupu, aby každá komponenta pro ni měla svojí vlastní metodu, vytvoří se služba, která danou funkci zapouzdří a komponenty, která ji chtějí používat, si službu jednoduše injektují. Tento postup zabezpečuje volnější spojení mezi jednotlivými prvky v kódu, zvyšuje celkovou

modularitu a taktéž „přepoužitelnost“ používaných funkcionalit napříč celou aplikací. Tento koncept je postaven na návrhovém vzoru vkládání závislostí, který je popsán v kapitole 3.3.9.1.[13]

3.3.8 REST

Representational State Transfer obecně známý zkratkou REST je architektonický styl pro distribuované systémy, který je hojně využíván pro tvorbu webových aplikací a bude využit při tvorbě aplikace v této diplomové práci.[21]

Aby bylo možné rozhraní považovat za RESTful, je nutné, aby splňovalo šest základních podmínek:[21]

- Jednotné rozhraní
- Klient-server
- Bezstavovost
- Cacheable
- Systém vrstev
- Kód na vyžádání

Princip jednotného rozhraní udává, že API rozhraní, které systém vystavuje pro konzumenty, musí být jasně stanoveno a každý zdroj musí být pouze jedno URI, které na něj odkazuje. Data, která klient dostane, by v sobě měla mít všechny informace, které klient potřebuje pro svoje další úkony, např. změna či smazání dat, a zároveň data mohou obsahovat odkazy na další části API, ke kterým může konzument přistoupit a lze touto cestou dále volat zdroje bez toho, aby o nich měl předchozí informace. Klient nikdy nepracuje se samotným zdrojem, ale pouze s jeho reprezentací, která může mít různé formáty, např. JSON nebo XML.[21][22]

Rozdělením aplikace na klienta a server, se stanou tyto dvě entity nezávislé a lze je vyvíjet zcela separátně. Jediné, co klient potřebuje vědět, jsou URI zdrojů, které vystavuje server. To umožnuje lehčí přenositelnost klientské aplikace a zároveň možnost mít více klientů, kteří komunikují s jednotným rozhraním serveru.[22]

Bezstavovost má REST z protokolu HTTP. Komunikace mezi klientem a serverem neobsahuje žádný stav. Každá žádost od klienta se vykonává zcela samostatně a nezávisle

na jiných žádostech a musí vždy obsahovat všechny informace, které server potřebuje, aby mohl žádost správně zpracovat bez jakýchkoliv dalších informací mimo danou žádost.[21]

Cachable znamená, že každá žádost může být cachovatelná, tedy že si klient může data od serveru ponechat a případně je využít při dalším volání daného zdroje, což značně zlepšuje výkonnost načítání stránek a zároveň to může ulehčit serveru, který nemusí pokaždé posílat vše.[22]

Systém vrstev umožnuje mít systém rozdělen na více komponent a každá komponenta může komunikovat pouze s další přilehlou vrstvou. To umožnuje velkou míru škálovatelnosti a taktéž větší zabezpečení aplikací. Z pohledu klienta nelze rozeznat žádný rozdíl.[21]

Na většinu žádostí bude server odpovídat odesláním statických dat ve formátech JSON, XML atd., nicméně díky poslednímu principu RESTu, má server možnost odpověď formou spustitelného kód, např. JavaScriptu, který klient může využít třeba na vykreslení části uživatelského rozhraní.[21]

REST je postaven na protokolu HTTP a ke komunikaci tak využívá jeho metod. V RESTu je konkrétně možné využít primárně tyto metody:[21]

- GET
- POST
- PUT
- DELETE
- PATCH
- OPTIONS
- HEAD

Metoda GET se používá pro získání dat ze serveru. Pomocí této metody nelze data jakkoliv měnit. Zavoláním stejného zdroje se stejnými parametry by mělo vrátit vždy stejné výsledky. Úspěšná žádost metodou GET by měla vždy vrátit HTTP kód 200 OK. Pokud není zdroj na serveru nalezen, návratový kód by měl být 404 Not Found. V případě, že žádost nemá požadovaný formát, vrátí se kód 400 Bad Request.[21]

Pro vytváření nových záznamů dat, resp. zdroje, slouží metoda POST. Při zpracování žádosti by se měl vrátit HTTP kód s číslem 201 Created a odkazem na umístění

nově vytvořeného zdroje. Je třeba si dávat pozor na fakt, že při poslání dvou identických žádostí POST, dojde k vytvoření dvou separátních zdrojů / záznamů dat. [21]

Metoda PUT slouží pro modifikaci již existujících dat na serveru, nicméně pokud zdroj, který se snažíme měnit, neexistuje, metoda tento záznam vytvoří. V takovém případě by server měl vrátit kód 201 Created, stejně jako když záznam vytváříme pomocí metody POST. V případě modifikace stávajícího zdroje vrací server kód 200 OK nebo 204 No content.[21]

DELETE slouží, jak už název napovídá, pro smazání zdroje ze serveru. Návratový kód může být 200 OK, 202 Accepted nebo 204 No content. Pokud pošleme dvakrát za sebou stejný požadavek na smazání záznamu, druhá žádost by měla skončit kódem 404 Not Found, jelikož zdroj byl smazán s první žádostí, při druhé žádosti daný zdroj již neexistuje a server ho nemůže tím pádem smazat.[21]

Další používanou metodou REST rozhraní je PATCH, slouží oproti metodě PUT pouze pro částečnou modifikaci záznamu dat nikoliv celou kolekci. Návratové kódy jsou obdobné a to 200 OK nebo 204 No content.[21]

Metoda OPTIONS se používá pro dotazování serveru, jaké jsou dostupné metody na daném zdroji. Pokud tedy chceme informaci, zda daný zdroj lze např. modifikovat či smazat, můžeme se dotázat metodou OPTIONS, která nám navrátí seznam použitelných metod. Návratový kód by měl být 200 OK. [26]

Poslední použitelnou metodou restu je HEAD. Tato metoda je identická s GET až ne jeden rozdíl, a to že v metodě HEAD nesmí být navráceno tělo zprávy. Vrátí se pouze návratový kód a informace z hlavičky stránky. Metoda se dá používat převážně na testování, zda zdroj, na který se chceme dotazovat, se na serveru vůbec nachází, je přístupný a lze s ním pracovat, a taktéž pro ověření jeho změn. Návratové kódy jsou taktéž identické s metodou GET, tedy 200 OK, případně 404 Not Found a 400 Bad Request.[26]

3.3.9 Návrhové vzory

Návrhové vzory jsou obecné postupy pro řešení různorodých úloh při vývoji softwaru, které zaručují správnou funkčnost a zároveň čistotu kódu. Existují desítky návrhových vzoru pro různé situace při programování. V následující kapitole budou představeny některé návrhové vzory, které budou využity při implementaci.

3.3.9.1 Dependency Injection

Dependency Injection, česky taktéž vkládání závislostí, je návrhový vzor, který implementuje koncept Inversion of Control, zkráceně IoC. Tento návrhový vzor je velmi využíván ve frameworku Spring, který obsahuje IoC kontejner. Místo toho, aby měl objekt pevné vazby na jiné objekty, díky tomuto principu objekt pouze definuje, jaké objekty chce, a je na frameworku, v tomto případě na Springu, aby mu je dodal. Tomuto vkládání závislostí se říká injektování. Díky Dependency Injection jsou jednotlivé třídy méně svázány mezi sebou, což vede k lepší udržitelnosti, a i škálovatelnosti celé aplikace, jelikož změna části kódu v jedné třídě se nikterak nedotkne třídy, do kterých je daná třída vkládána. Ve Springu je možné vkládání závislostí provést v konstruktoru třídy, v setteru či jako proměnou třídy. [23]

3.3.9.2 Model-View-Controller

Návrhový vzor Model-View-Controller, v českém překladu Model-Pohled-Ovládání, zkráceně MVC rozděluje aplikaci na tři části. Model představuje aplikační logiku aplikace. View uživatelské rozhraní a Controller se stará o reakce na uživatelské požadavky a řídí tok aplikace. Tímto logickým rozdělením kompetencí vzniká čistší kód oproti tomu, aby se jeden celek musel starat o všechny tyto činnosti samostatně. [24]

3.3.9.3 Strategy

Strategy, česky strategie, je návrhový vzor vhodný, pokud máme skupinu podobných algoritmů, které za sebe lze zaměňovat podle toho, co konkrétně klient požaduje. Všechny tyto implementace algoritmů rozdělíme do jednotlivých objektů a „schováme“ za jedno obecné rozhraní. Tím, že algoritmy nejsou v jedné třídě, nám umožňuje je nezávisle upravovat, bez toho, abychom se obávali, že do ostatních zaneseme chybu, a taktéž je velmi jednoduché přidat novou implementaci algoritmu bez nutnosti zasahovat do již vyvinutých částí. Jaká implementace rozhrání bude používat se poté zvolí při běhu programu.[25]

3.3.10 Vývojové principy

V následující kapitole budou popsány obecné principy používané při programování, kterými se bude autor práce řídit při implementaci. Jejich dodržování je většinou považováno za samozřejmost a výsledný kód je poté kvalitnější a zároveň méně náchylný na chyby, které by jinak při jejích nedodržování vznikaly. Taktéž samotným vývojářům, kteří k takovému systému přijdou, se s ním mnohém lépe pracuje a jsou schopní rychleji do něj zavádět změny.

3.3.10.1 SOLID

SOLID není pouze jeden princip, kterým bychom se měli při vývoji softwaru řídit, ale jedná se hned o několik principu. Konkrétně jich je pět a z jejich počátečních písmen vznikl akronym SOLID. V anglickém originále se jedná o:

- Single Responsibility – princip jedné odpovědnosti
- Open / Closed – otevřenost / uzavřenost
- Liskov Substitution – Liskovové princip zaměnitelnosti
- Interface Segregation – rozdělení rozhraní
- Dependency Inversion – obracení závislostí

Princip jedné odpovědnosti nám radí, že třída by měla mít odpovědnost vždy pouze za jednu věc a tu by měla vykonávat. Neměla by řešit nic jiného. Všechny metody, které obsahuje, by měly sloužit právě pro vykonání dané odpovědnosti. Výhodou takového přístupu by mělo být, že v systému je jasné, jaká třída za co zodpovídá, a tím pádem při hledání chyb či nutnosti rozšíření nějaké funkce by mělo být hned zřejmé, jaké třídy se bude úprava týkat.

Princip otevřenosti / uzavřenosti nám říká, že třídy a metody je možné rozšiřovat, ale neměli bychom je měnit. Jakmile už máme třídu či metodu vytvořenou a funkční v systému, její kód by se neměl modifikovat. Nicméně je možné rozšířit její funkčnost. Toho lze docílit např. pomocí dědičnosti. Kód nadtřídy neměníme, zdědí ho její podtřída, kterou již můžeme rozšiřovat bez obavy, že poroucháme funkčnost předka. Nelze samozřejmě tento princip dodržovat absolutně. Během vývoje se vyskytne mnoho situací,

kdy není možné postupovat jinak než současný kód změnit. Cílem by však mělo být co nejvíce se tohoto principu držet.

Aby kód dodržel Liskovové princip zaměnitelnosti, musí platit, že pokud máme třídu, která je podtypem jiné třídy, můžeme místo dané nadtřídy použít jejího potomka bez toho, abychom jakkoliv změnili požadované chování aplikace.

Princip rozdělení rozhraní uvádí, že třídy by neměly být nuceny implementovat rozhraní, jehož metody nebudou potřebovat a používat. Z toho vyplývá, že rozhraní by neměla dosahovat obrovských rozměru, ale naopak by měla být rozdělována na menší prvky tak, aby třídy, které je používají, mohli vždy použít skutečně jen takové rozhraní, které pro svojí funkci potřebují. Díky tomu, že máme více rozhraní, která jsou úzce zaměřená, docílíme i volnějších spojení mezi prvky v kódu.

Poslední princip SOLID je obracení závislostí. Dle tohoto principu by neměla abstraktní vrstva záležet na implementaci, ale naopak implementace by měla záviset na abstrakci. Stejně tak vyšší vrstva abstrakce není závislá na nižších vrstvách. Implementační řešení se může často měnit, a pokud na něm závisí abstraktní vrstva, bude se muset měnit taktéž, aby kód fungoval. Toto by se ale nemělo stát. Budeme-li postupovat podle tohoto principu, je možné změnit implementaci na nižších vrstvách bez toho, abychom museli provádět změny ve vrstvách vyšších.

3.3.10.2 KISS

KISS, jenž je akronymem anglického spojení „keep it simple stupid“, není principem, který by se objevoval pouze v otázce programování, ale lze se s ním setkat obecně ve spoustě aktivit denního života. Hlavní myšlenkou je naprogramovat danou věc co nejjednodušším způsobem, co lze. Vyhnout se obřím komplexním implementacím, které zanedlouho způsobí absolutní nepřehlednost v kódu aplikace. Čím jednodušší výsledné řešení je, tím je přehlednější a méně náchylné k chybám.

3.3.10.3 DRY

Princip DRY je akronymem k anglickému „don’t repeate yourself“. Jde o myšlenku, že by v kódu neměly být totožné funkce na více místech. Pokud nějakou takovou funkci máme, měli bychom ji zapouzdřit, mít ji na jednom místě a ve všech

případech, kde ji potřebujeme, jednoduše zavolat tuto metodu. Výhodou takového přístupu je, že pokud je v dané funkci chyba, či ji potřebujeme změnit, není potřeba, abychom hledali všechny výskyty, kde se daná funkce nachází a ručně je měnili, ale stačí nám změnit pouze jednu danou implementaci, kterou využívá celý systém.

4 Vlastní práce

V praktické části této diplomové práce bude popsán kompletní postup návrhu a implementace webové aplikace pro potřeby issue trackingu. Při jeho implementaci budou využity technologie popsané v kapitole 3.3 Použité technologie a znalosti autora načerpané za dobu studia na vysoké škole.

4.1 Analýza

Při tvorbě nového systému je nejprve potřeba provést analýzu. V této fázi jsou shromážděny všechny požadavky, které by aplikace měla ve své výsledné formě splňovat. Výsledky analýzy poté slouží jako základ pro návrh samotného systému.

4.1.1 Požadavky

Požadavky, které jsou zahrnuty v analýze, se dělí na funkční a nefunkční. Do funkčních požadavků jsou zahrnuté přímo jednotlivé funkčnosti, které musí výsledná aplikace obsahovat. Jako nefunkční požadavky se poté berou všechny požadavky, které nespadají do funkčních požadavků, a jsou to především požadavky na stabilitu, rychlost, použitelnost a bezpečnost aplikace. Aplikace je vytvářena pouze pro potřeby této diplomové práce. Z toho důvodu jsou všechny požadavky od autora práce, nikoliv od externí entity.

4.1.1.1 Funkční požadavky

Issue bude mít pole název, popis, typ, stav, projekt a nastaveného řešitele
V systému budou role uživatel a administrátor

Na issue bude možné přidávat vlastní pole

Na issue bude možné přidávat komentáře

V aplikace je možné změnit workflow

V aplikaci budou dvě role: User a Admin

Do aplikace má přístup pouze přihlášený uživatel

Nepřihlášený uživatel se může přihlásit pomocí formuláře

Uživatel vidí existující issue

Uživatel má možnost zadat nové issue
Uživatel má možnost upravit existující issue
Uživatel má možnost filtrovat a vyhledávat issue
Uživatel má možnost změnit si heslo
Administrátor má všechny funkce, co má role uživatel
Administrátor má možnost spravovat typy issue
Administrátor má možnost spravovat vlastní pole na issue
Administrátor má možnost spravovat stavy issue
Administrátor má možnost spravovat workflow
Administrátor má možnost spravovat projekty
Administrátor má možnost spravovat uživatele v aplikaci
Administrátor má možnost vytvořit uživatele

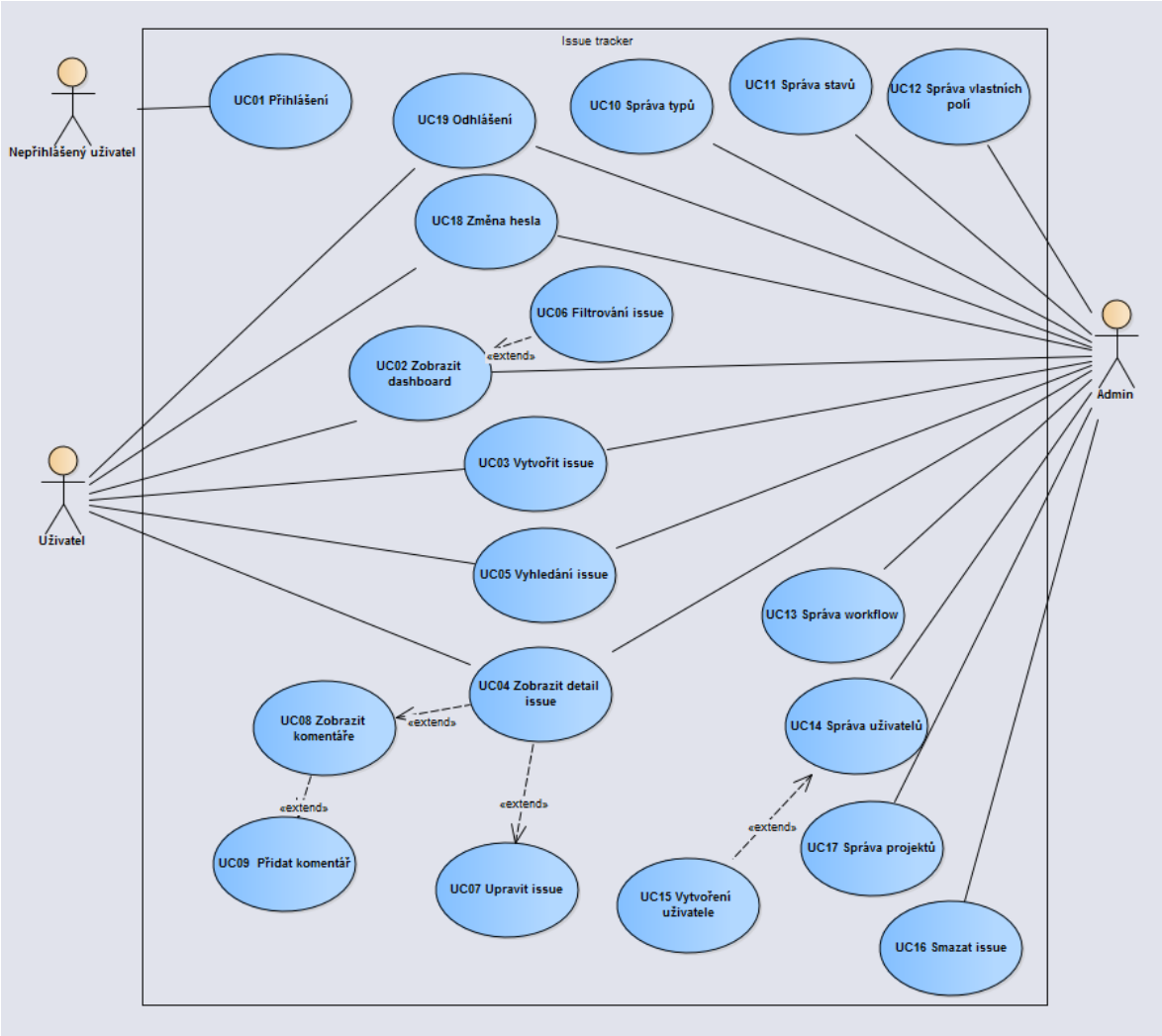
4.1.1.2 Nefunkční požadavky

Systém bude implementován jako webová aplikace
Backend aplikace bude implementován v jazyce Java 11 s frameworkem Spring
Backend aplikace bude vystavovat REST endpointy
Frontend aplikace bude implementován v jazyce TypeScript s frameworkem Angular 8
Zdrojový kód musí splňovat standardní požadavky pro psaní softwaru
Pro databázi bude využito MySQL
Aplikace musí podporovat prohlížeč Google Chrome (od v50)

4.2 Návrh

Jako první věc ve fázi návrhu vytvoříme diagram případů užití. Ten slouží pro zobrazení interakce vnějších uživatelů, tzv. aktérů, s vyvíjeným systémem. Diagram zjednodušeně popisuje, jaké operace můžou jednotliví uživatelé v aplikaci využívat, bez popisu vnitřních procesů, a taktéž pomáhá lépe specifikovat ohraničení celého systémů. Diagram se skládá již ze zmíněných aktérů a případů užití. Aktéři jsou externí objekty komunikující s daným systémem. Může se jednat jak o jednotlivé uživatele, tak i jiné systémy. Případ užití je popis akcí či scénář kroků, které aktér a systém vykonávají k dosažení cíle daného případu užití.

4.2.1 Diagram případů užití



Obrázek 6 Diagram případů užití

Se systémem mohou interagovat celkem tři aktéři. Nepřihlášený uživatel má pouze jedinou možnost, a to se do systému přihlásit. Bez přihlášení nelze se systémem jinak pracovat. Role uživatel má možnost zobrazit dashboard, vytvořit issue, zobrazit detail issue, změnit si heslo a odhlásit. Admin slouží především pro administraci celého projektu. Oproti uživateli může navíc spravovat typy a stavy issue, projekty, vlastní pole, workflow a taktéž vytvořit a spravovat jednotlivé uživatele. Admin taktéž může issue smazat.

Mohla by vyvstanout otázka, proč má pouze administrátor možnost smazat issue a uživatel nikoliv. Takto bylo rozhodnuto z důvodu, že mazání issue není něco, co by mělo být běžným jevem. Jakmile je v systému issue jednou vytvořené, mělo by tam i zůstat. Musí být přítomny opravdu dobré důvody pro jeho smazání. Proto je tato možnost v systému ponechána, ale pouze pro roli administrátora.

4.2.2 Scénáře případů užití

Mimo samotný diagram případů užití jsou dále jednotlivé scénáře případů užití rozepsané i textově. V této formě jsou detailně rozepsány postupy, jak dosáhnout cíle daného scénáře. Každý scénář by měl mít vždy hlavní tok, a poté může mít několik alternativních toků. Tento detailní popis by měl mít každý scénář případu užití. V rámci této kapitoly bude ukázán pouze jeden, aby si čtenář mohl udělat obrázek, jak by takový scénář měl vypadat.

Název	**UC02 Založit issue**	
Popis	Scénář umožňuje uživateli založit issue v aplikaci	
Aktéři	Uživatel Admin	
Vstupní podmínky	Nejsou	
Základní tok		
Číslo kroku	Aktér	Popis
1	Uživatel	Uživatel klikne na odkaz Vytvořit nové
2	Uživatel	Uživatel vyplní všechna povinná pole
3	Uživatel	Uživatel klikne na tlačítko Uložit
4	Systém	Systém zvaliduje data vyplněné uživatelem
5	Systém	Systém issue uloží
6	Systém	Systém přejde na detail právě vytvořeného issue
Alternativní tok 1	Uživatel nevyplní všechna povinná pole	
4.1	Systém	Systém zvaliduje data vyplněná uživatelem
4.2	Systém	Systém issue neuloží
4.3	Systém	Systém zůstane na obrazovce zadávání nového issue
4.4	Systém	Systém vyznačí pole, které musí být vyplněné
Alternativní tok 2	Uživatel vyplní některé z polí nevalidní hodnotou a klikne na tlačítko Uložit	
4.1	Systém	Systém zvaliduje data vyplněná uživatelem
4.2	Systém	Systém issue neuloží

4.3	Systém	Systém zůstane na obrazovce zadávání nového issue
4.4	Systém	Systém vyznačí pole, která jsou vyplněná nevalidní hodnotou
Alternativní tok 3	Uživatel zruší zadávání nového issue	
2.1	Uživatel	Uživatel klikne na tlačítko Zrušit
2.2	Systém	Systém zobrazí dialogové okno s upozorněním, že všechny zadané informace budou smazány, a zda chce uživatel skutečně zadávání zrušit
2.3	Uživatel	Uživatel dialog potvrdí
2.4	Systém	Systém issue neuloží
2.5	Systém	Systém přejde na hlavní obrazovku aplikace
Alternativní tok 4	Uživatel zruší zadávání nového issue, ale nepotvrdí dialogové okno	
2.3.1	Uživatel	Uživatel dialog nepotvrdí
2.3.2	Systém	Systém dialogové okno zavře a zůstane na obrazovce zadávání nového issue
Podmínky pro dokončení	Nové issue je uložené v databázi aplikace	

Tabulka 2 UC02 Založit issue

Název	**UC12 Správa vlastních polí**	
Popis	Scénář umožňuje správu typů vlastních polí issue	
Aktéři	Admin	
Vstupní podmínky	Přihlášený uživatel má roli Admin	
Základní tok		
Číslo kroku	Aktér	Popis
1	Uživatel	Uživatel klikne na odkaz Nastavení
2	Systém	Systém roztáhne menu Nastavení
3	Uživatel	Uživatel klikne na odkaz
4	Systém	Systém přejde na obrazovku Vlastní pole
5	Uživatel	Uživatel klikne na tlačítko Přidat
6	Systém	Systém zobrazí modální okno Nový typ
7	Uživatel	Uživatel vyplní název pole

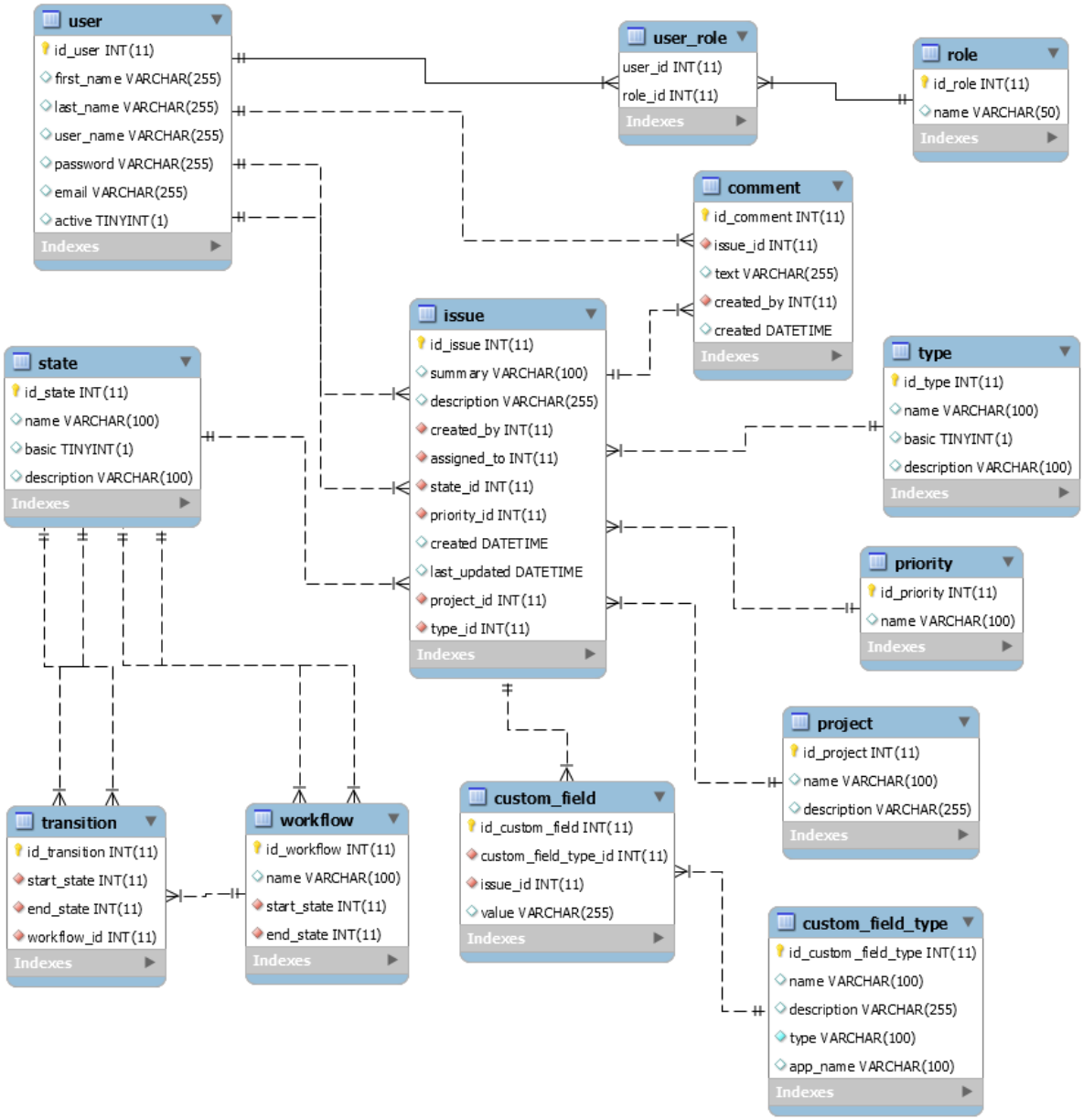
8	Uživatel	Uživatel vybere popis pole
9	Uživatel	Uživatel vybere typ pole
11	Uživatel	Uživatel klikne na tlačítko Uložit
12	Systém	Systém provede validaci, zda se vlastní pole v databázi s daným názvem již nenachází
13	Systém	Systém vlastní pole uloží do databáze
14	Systém	Systém zavře modální okno
15	Systém	Systém zobrazí nový typ vlastního typu na obrazovce
Alternativní tok 1	Uživatel zadá jako název pole název shodný s již vytvořeným typem vlastního pole v databází a klikne na tlačítko Přidat	
12.1	Systém	Systém provede validaci, zda se vlastní pole v databázi s daným názvem již nenachází a nalezne shodu
12.2	Systém	Systém vlastní pole neuloží do databáze
12.3	Systém	Systém zavře modální okno
12.4	Systém	Systém zobrazí upozornění, že nové vlastní pole nemůže mít název stejný jako již vytvořené vlastní pole
Alternativní tok 3	Uživatel zruší zadávání nového typu vlastního pole	
11.1	Uživatel	Uživatel klikne na tlačítko Zrušit
11.2	Systém	Systém zavře modální okno
Alternativní tok 4	Uživatel smaže typ vlastního pole	
5.1	Uživatel	Uživatel u daného klikne na tlačítko Smazat
5.2	Systém	Systém typ vlastního pole smaže z databáze
Podmínky pro dokončení	Nové vlastní pole je uložené v databázi aplikace	

Tabulka 3 UC12 Správa vlastních polí

4.2.3 Datový model

Po vytvoření diagramů užití a specifikace jednotlivých scénářů přichází další fáze analýzy, která se zabývá návrhem databáze aplikace. Je potřeba specifikovat, jaké se budou

v databázi nacházet konkrétní tabulky, jaké atributy budou mít a taktéž je potřeba navrhnout jejich propojení. Je nutné si strukturu databáze velmi dobře promyslet. Jakmile je specifikace jednou vytvořena a začne se pracovat na samotné implementaci, je už velmi složité, a i finančně náročné, provádět ve struktuře databáze změny. Při návrhu je taktéž nutné klást důraz na dodržování obecných zásad pro tvorbu relačních databází.



Obrázek 7 Datový model

4.2.3.1 Tabulky

V následující kapitole budou detailně popsány všechny tabulky, které se nachází v datovém modelu aplikace. Pro vytvoření databáze, tabulek a jejich další správu byla použita aplikace MySQL Workbench 8.0. Jedná se o volně dostupný nástroj pro navrhování databáze od společnosti Oracle. [27]

Tabulka User obsahuje informace o všech uživatelích, kteří se v aplikaci nachází a mohou se do ní přihlásit.

User		
Atribut	Datový typ	Popis
Id_user	INT	Primární klič tabulky User
First_name	VARCHAR	Křestní jméno uživatele
Last_name	VARCHAR	Příjmení uživatele
User_name	VARCHAR	Uživatelské jméno uživatele
Password	VARCHAR	Heslo uživatele
Email	VARCHAR	E-mailová adresa uživatele
Active	TINYINT	Příznak, zda je uživatel aktivní

Tabulka 4 Tabulka User

Tabulka Role obsahuje výčet všech rolí, které může mít uživatel přirazené.
V tabulce se nachází následující role:

- USER – Uživatel
- ADMIN – Administrátor

Role		
Atribut	Datový typ	Popis
Id_role	INT	Primární klíč tabulky Role
Name	VARCHAR	Název role

Tabulka 5 Tabulka Role

Vazební tabulka User_role slouží pro propojení uživatele v tabulce User s jeho přirazenými rolemi v tabulce Role.

User_role		
Atribut	Datový typ	Popis
User_id	INT	Cizí klíč sloupce Id_user v tabulce User

Role_id	INT	Cizí klíč sloupce Id_role v tabulce Role

Tabulka 6 Tabulka User_role

V tabulce Issue se nachází informace o všech issue v aplikaci.

Issue		
Atribut	Datový typ	Popis
Id_issue	INT	Primární klíč tabulky Issue
Summary	VARCHAR	Krátké shrnutí issue
Description	VARCHAR	Detailní popis issue
Created_by	INT	Uživatel, který issue vytvořil. Cizí klíč sloupce Id_user v tabulce User
Assigned_to	INT	Uživatel, na kterého je issue aktuálně přirazené. Cizí klíč sloupce Id_user v tabulce User
State_id	INT	Stav issue. Cizí klíč sloupce Id_state v tabulce Issue_state
Priority_id	INT	Priorita issue. Cizí klíč sloupce Id_priority v tabulce Issiue_priority
Created	DATETIME	Datum vytvoření issue
Last_updated	DATETIME	DATE Datum poslední změny na issue
Project_id	INT	Cizí klíč sloupce Id_project v tabulce Project
Type_id	INT	Cizí klíč sloupce Id_type v tabulce Type

Tabulka 7 Tabulka Issue

Tabulka State obsahuje výčet všech stavů, kterých může issue v aplikaci nabývat.

Výčet základních stavů:

- Nový
- Otevřený
- Znovuotevřený
- Vyřešený
- Zavřený

State		
Atribut	Datový typ	Popis

Id_state	INT	Primární klíč tabulky State
Name	VARCHAR	Název stavu
Basic	TINYINT	Příznak, zda se jedná o základní stav. Pokud ano, nelze ho smazat
Description	VARCHAR	Popis stavu

Tabulka 8 Tabulka State

Tabulka Type obsahuje výčet všech typů, kterých může issue v aplikaci nabývat.

Výčet základních typů:

- TASK
- BUG

Type		
Atribut	Datový typ	Popis
Id_type	INT	Primární klíč tabulky type
Name	VARCHAR	Název typu
Basic	TINYINT	Příznak, zda se jedná o základní typ. Pokud ano, nelze ho smazat
Description	VARCHAR	Popis typu

Tabulka 9 Tabulka Type

Tabulka Priority obsahuje výčet všech priorit, kterých může issue v aplikaci nabývat.

Výčet priorit je následující:

- Kritická
- Vysoká
- Střední
- Nízká

Priority		
Atribut	Datový typ	Popis
Id_priority	INT	Primární klíč tabulky Priority
Name	VARCHAR	Název priority

Tabulka 10 Tabulka Priority

Tabulka Comment je určená pro uchovávání jednotlivých komentářů přidaných k issue.

Comment

Atribut	Datový typ	Popis
Id_comment	INT	Primární klíč tabulky Comment
Issue_id	INT	Issue, ke kterému je komentář přirazen. Cizí klíč sloupce Id_issue v tabulce Issue
Text	VARCHAR	Text komenáře
Created_by	INT	Uživatel, který komentář přidal. Cizí klíč sloupce Id_user v tabulce User
Created	DATETIME	Datum vytvoření komentáře

Tabulka 11 Tabulka Comment

Tabulka Project uchovává informace o projektu v aplikaci.

Project		
Atribut	Datový typ	Popis
Id_project	INT	Primární klíč tabulky Project
Name	VARCHAR	Název projektu
Description	VARCHAR	Popis projektu

Tabulka 12 Tabulka Project

V tabulce Workflow jsou uloženy informace o workflow.

Workflow		
Atribut	Datový typ	Popis
Id_workflow	INT	Primární klíč tabulky Workflow
Name	VARCHAR	Název workflow
Start_state	INT	Stav, ve kterém workflow začíná. Cizí klíč sloupce Id_state v tabulce State
End_state	INT	Stav, ve kterém workflow končí. Cizí klíč sloupce Id_state v tabulce State

Tabulka 13 Tabulka Workflow

Tabulka Transition uchovává informace o jednotlivých přechod mezi stavy.

Transition		
Atribut	Datový typ	Popis
Id_transition	INT	Primární klíč tabulky Transition
Start_state	INT	Stav, ze kterého se přechází. Cizí klíč

		sloupce Id_state v tabulce State
End_state	INT	Stav, na který se přechází. Cizí klíč sloupce Id_state v tabulce State
Workflow_id	INT	Workflow, ke kterému přechod patří. Cizí klíč sloupce Id_workflow v tabulce Workflow

Tabulka 14 Tabulka Transition

Do tabulky Custom_field_type se ukládají pole vytvořená v aplikaci

Custom_field_type		
Atribut	Datový typ	Popis
Id_custom_field_type	INT	Primární klíč tabulky Custom_field_type
Name	VARCHAR	Název vlastního pole
Description	VARCHAR	Popis vlastního pole
Type	VARCHAR	Typ vlastního pole
App_name	VARCHAR	Název vlastního pole v lower case bez mezer

Tabulka 15 Tabulka Custom Field Type

Tabulka Custom_field slouží pro ukládání hodnot vlastních polí na jednotlivých issue

Custom_field		
Atribut	Datový typ	Popis
Id_custom_field	INT	Primární klíč tabulky Custom_field
Custom_field_type_id	INT	Cizí klíč sloupce Id_custom_field_type v tabulce Custom_field_type
Issue_id	INT	Cizí klíč sloupce Id_issue v tabulce Issue
Value	VARCHAR	Hodnota vlastního pole

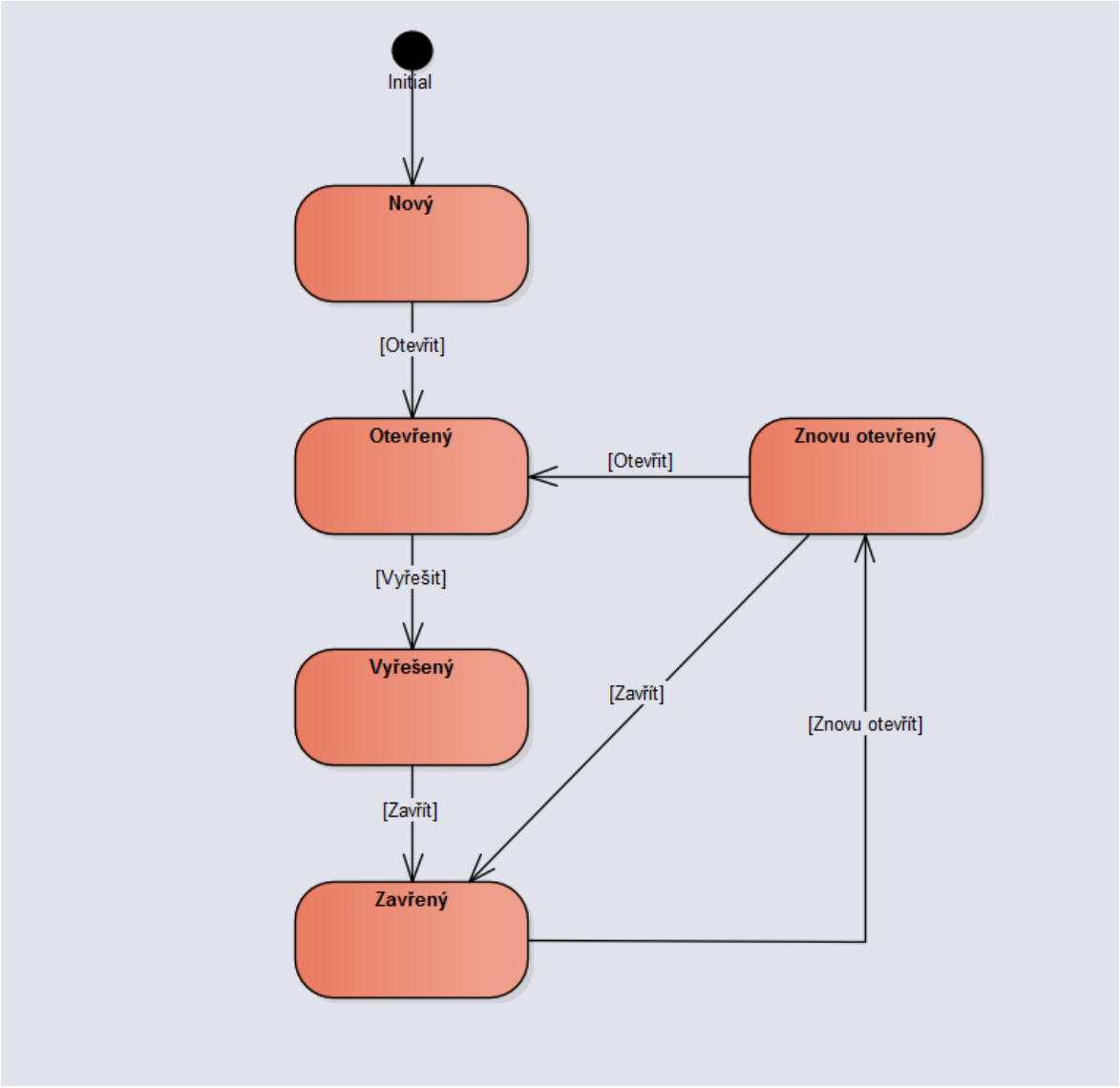
Tabulka 16 Tabulka Custom Field

4.2.4 Stavový diagram

Prvkem v systému, pro který je velmi přínosné vytvořit stavový diagram je samotné issue. Tento diagram bude de facto sloužit jako popis základního workflow v systému.

Samozřejmě, že s možností toto workflow v aplikaci změnit, nemusí být v samotné implementaci tento diagram aktuální.

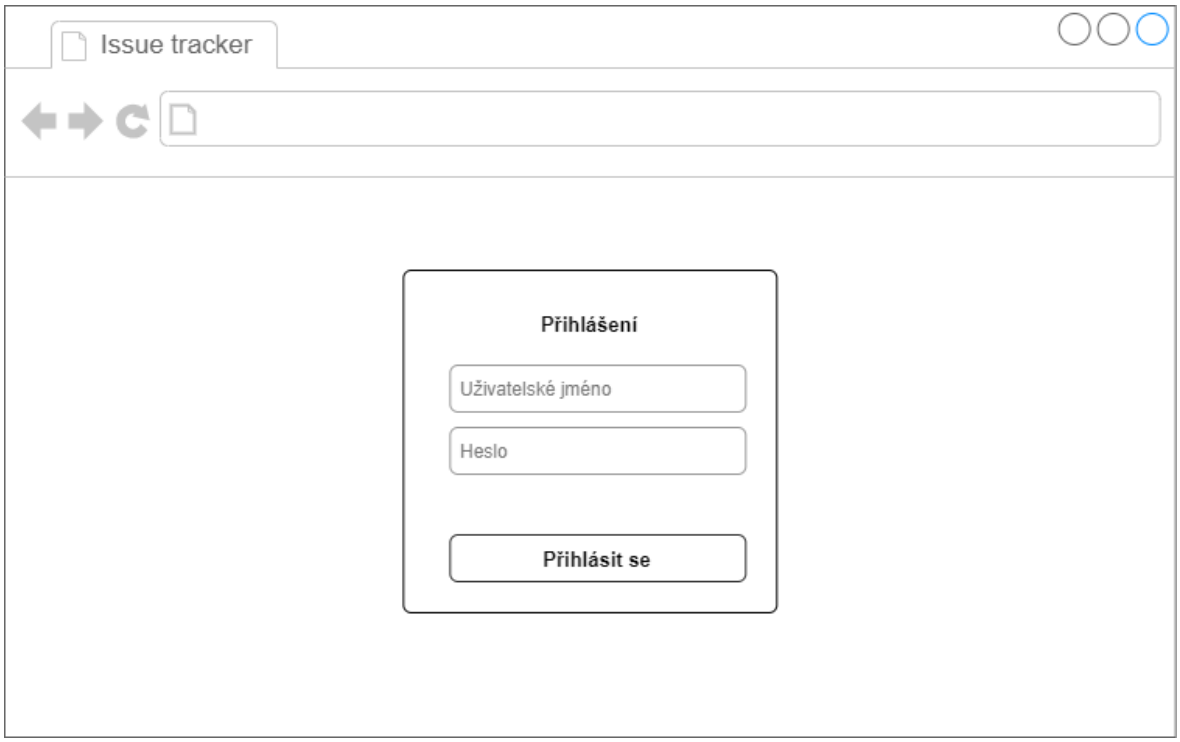
Průběh diagramu je následovný. Při vytvoření issue uživatelem je stav nastaven na Nový. V tomto stavu se čeká, až je issue někomu přiřazeno, aby se na něm mohlo začít pracovat. Jakmile se tak stane, je možné ho přepnout do stavu Otevřený. Poté, co je práce na požadavku hotová, ho lze přepnout do stavu Vyřešený. V tomto stavu čeká na potvrzení, že byl požadavek splněn. Pokud tomu tak je, je issue přepnuto do stavu Zavřený, což je finální stav diagramu. Pokud ovšem nelze prohlásit issue za splněné, je znovu otevřeno, což ho posune do stavu Znovu otevřený. Zde se rozhodne, zda je znovuotevření oprávněné či nikoliv. Pokud ano a na issue je třeba práci dodělat, je stav změněn na Otevřený. Pokud znovuotevření oprávněné není, je issue přesunuto znovu do stavu Zavřený.



Obrázek 8 Stavový diagram

4.2.5 Wireframy

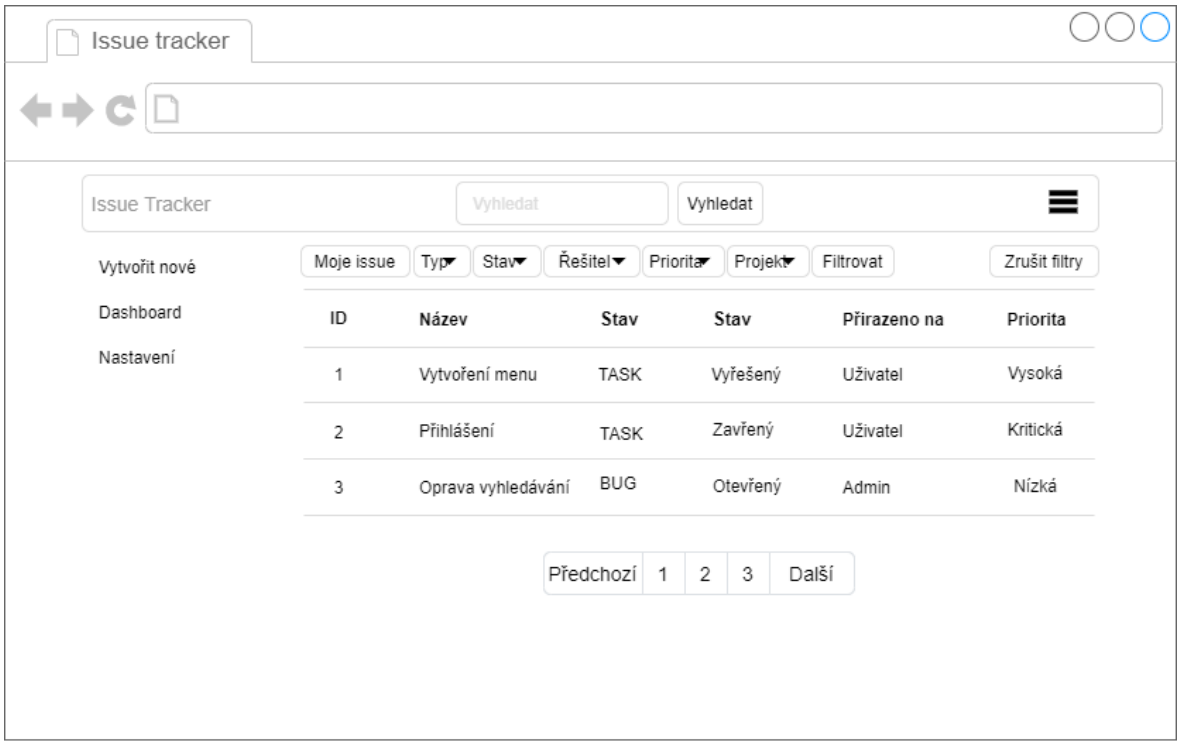
Wireframy slouží k zobrazení, jaké položky se budou na jednotlivých stránkách nacházet a jaká bude jejich poloha. Jelikož je pro vstup do aplikace vyžádáno přihlášení, jako první wireframe byla vytvořena obrazovka přihlášení. Na té se nachází standardní formulář. Uživatel se může přihlásit pomocí zadání uživatelského jména a hesla a kliknutím na tlačítko Přihlásit se. Při stisku tlačítka Přihlásit se probíhá validace na obě pole, která musí být vyplněná. Pokud se přihlášení nezdaří, je na to uživatel upozorněn.



Obrázek 9 Wireframe Přihlášení

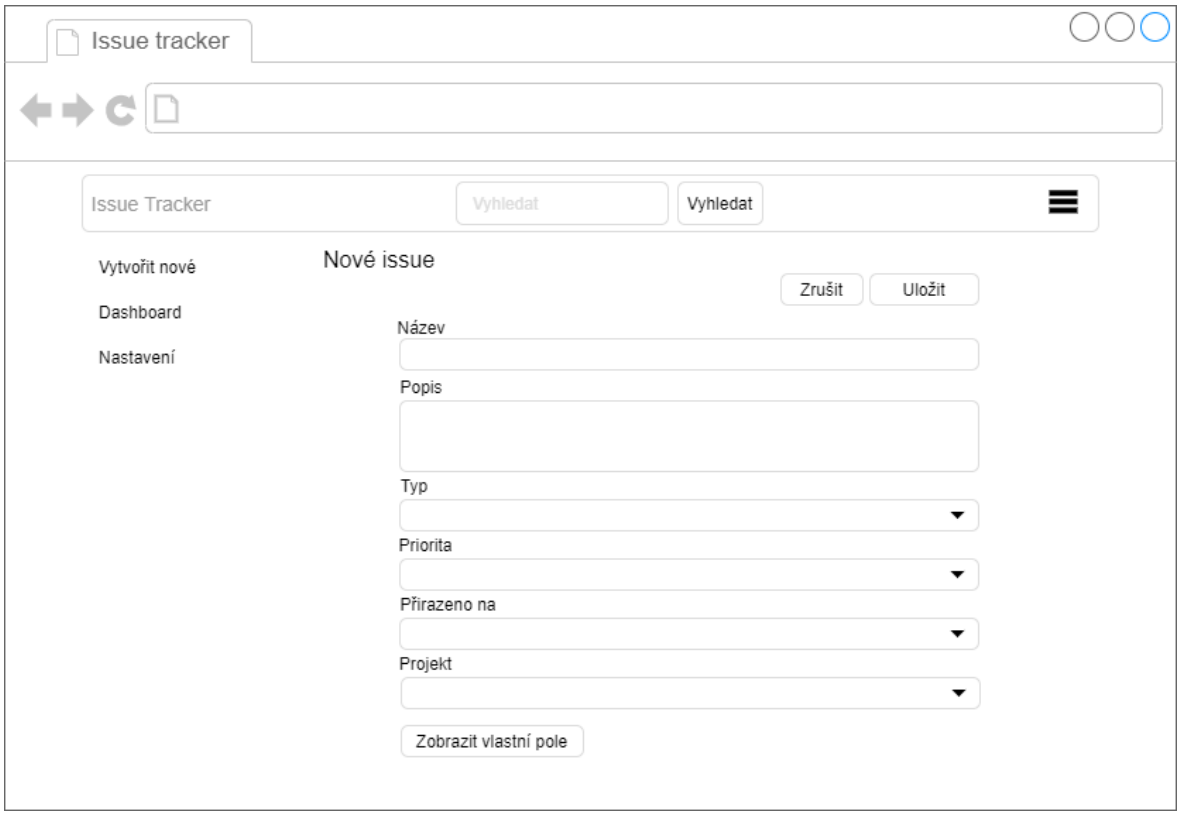
Na všech dalších obrazovkách mimo přihlášení je vždy viditelná hlavička stránky, kde se na levé straně nachází logo aplikace, v prostředku vyhledávácí pole s tlačítkem a na pravé straně uživatelské jméno právě přihlášeného uživatele a odkaz pro odhlášení. Na levé straně stránky je poté možnost využívat menu, kde jsou položky Vytvořit nové, Dashboard a Natavení. Položka Nastavení bude implementovaná jako dropdown a po kliknutí na ni se zobrazí položky Nastavení: Typy, Stavy, Vlastní pole, Workflow, Projekty, Uživatelé.

Při přihlášení se uživateli zobrazí obrazovka dashboard, která obsahuje všechna issue v systému. Na obrazovce je zobrazen pouze omezený počet, zbytek je dostupný pomocí stránkování. Taktéž je k dispozici funkce filtrování výsledků.



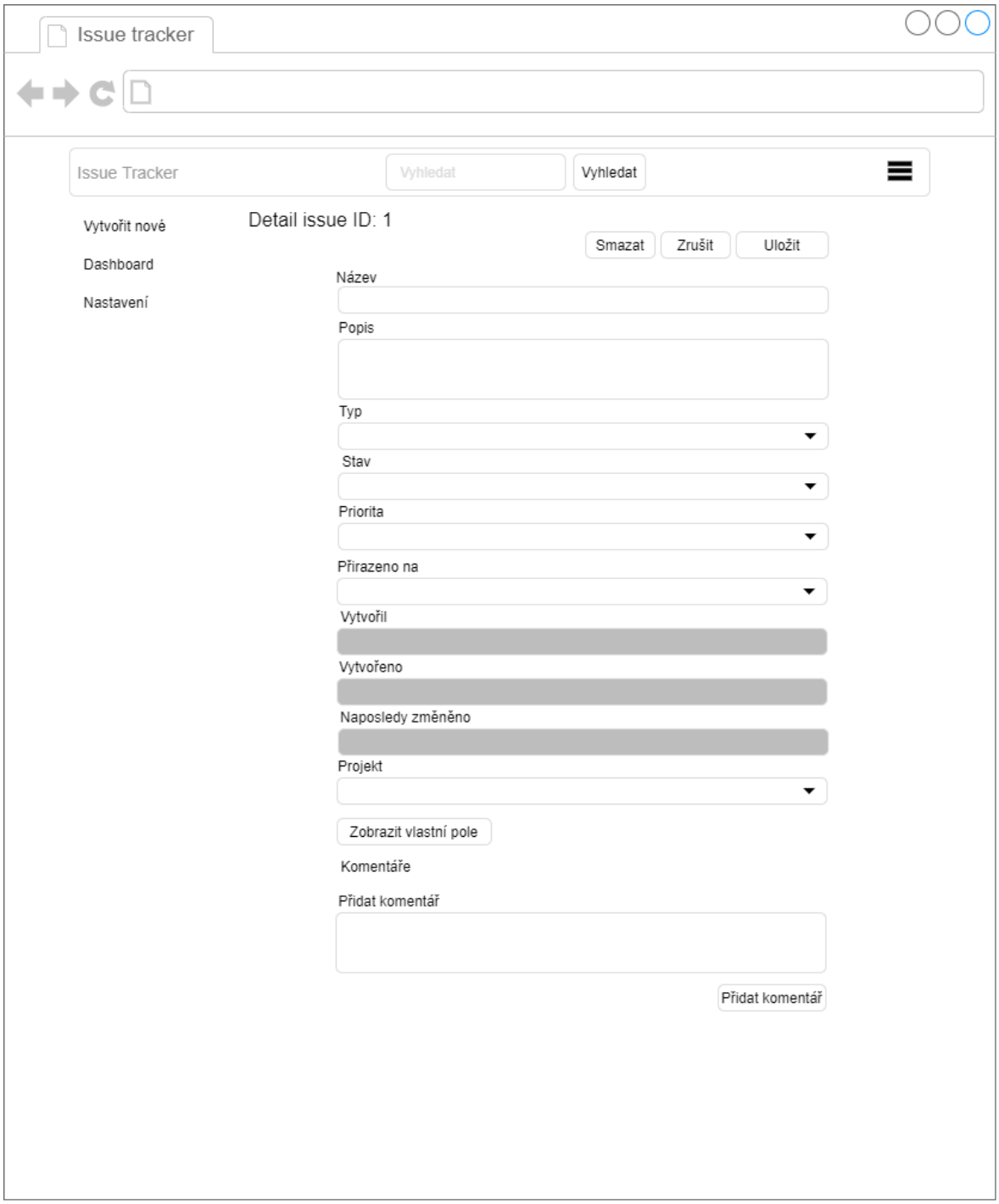
Obrázek 10 Wireframe Dashboard

Další obrazovka slouží pro vytvoření nového issue. Je zde formulář obsahující pole název, popis, stav, priorita a přirazeno na a projekt. Dole na levé straně se nachází tlačítko Zobrazit vlastní pole. Po kliknutí na něj se zobrazí všechny typy vlastních polí, která jsou momentálně v systému a uživatel je může taktéž doplnit. Nahoře na pravé straně se nachází tlačítka Uložit a Zrušit. Po stisku tlačítka Uložit se provede validace polí. Všechna pole na stránce musí být vyplněna validními hodnotami. Pokud některé z nich není, bude na to uživatel upozorněn. Při úspěšně provedené validaci systém přejde na zobrazení detailu issue.



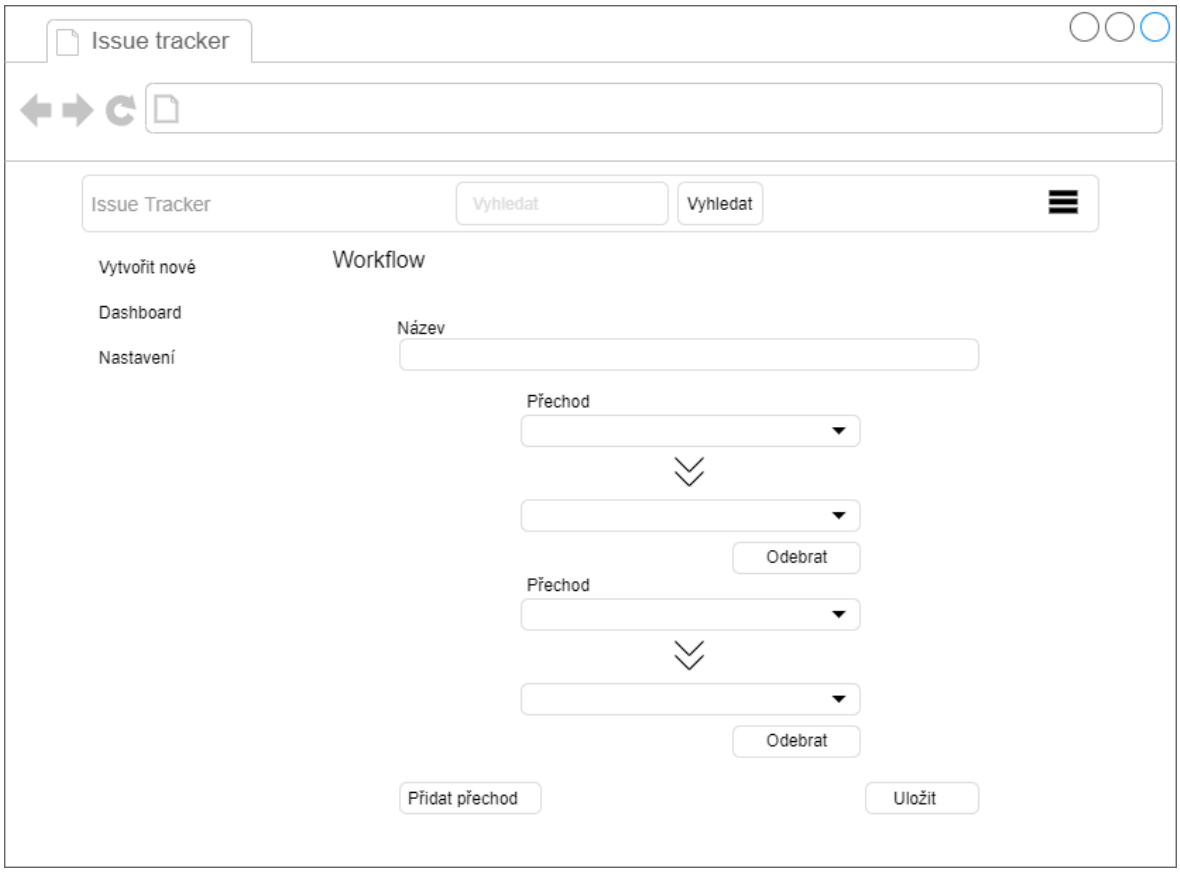
Obrázek 11 Wireframe Nové issue

Následující stránka představuje obrazovku pro zobrazení detailu issue a zároveň pro jeho úpravu. Je téměř totožná jako obrazovka pro vytvoření issue. Oproti ní se zde nachází navíc needitovatelná pole vytvořil, vytvořeno a naposledy změněno. Dole na obrazovce jsou zobrazené komentáře připojené k issue a je taktéž možné přidat nový komentář po vyplnění pole pro text a stisknutí tlačítka Přidat komentář. Nahoře na pravé straně je oproti vytvoření issue i tlačítko Smazat, které je viditelné pouze pro uživatele s rolí Admin.



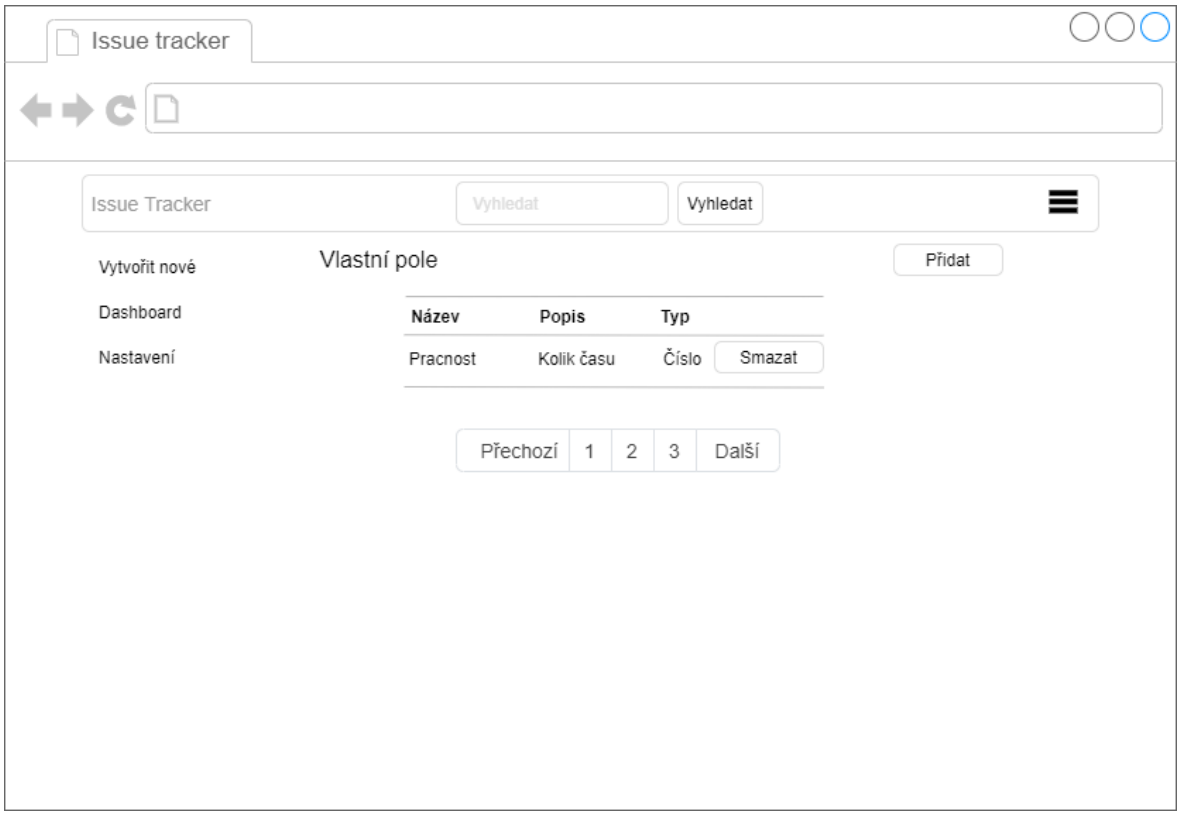
Obrázek 12 Wireframe Detail issue

V nastavení se nachází obrazovka pro správu workflow, jež je ukázána na dalším wireframu. Na obrazovce se nachází formulář pro nastavení přechodu mezi stavy ve workflow. Jedna část formuláře je vždy jeden přechod, kde je zvolen počáteční stav a konečný stav. Přechod se dá odebrat, nebo lze přes tlačítko Přidat další vložit nový přechod.

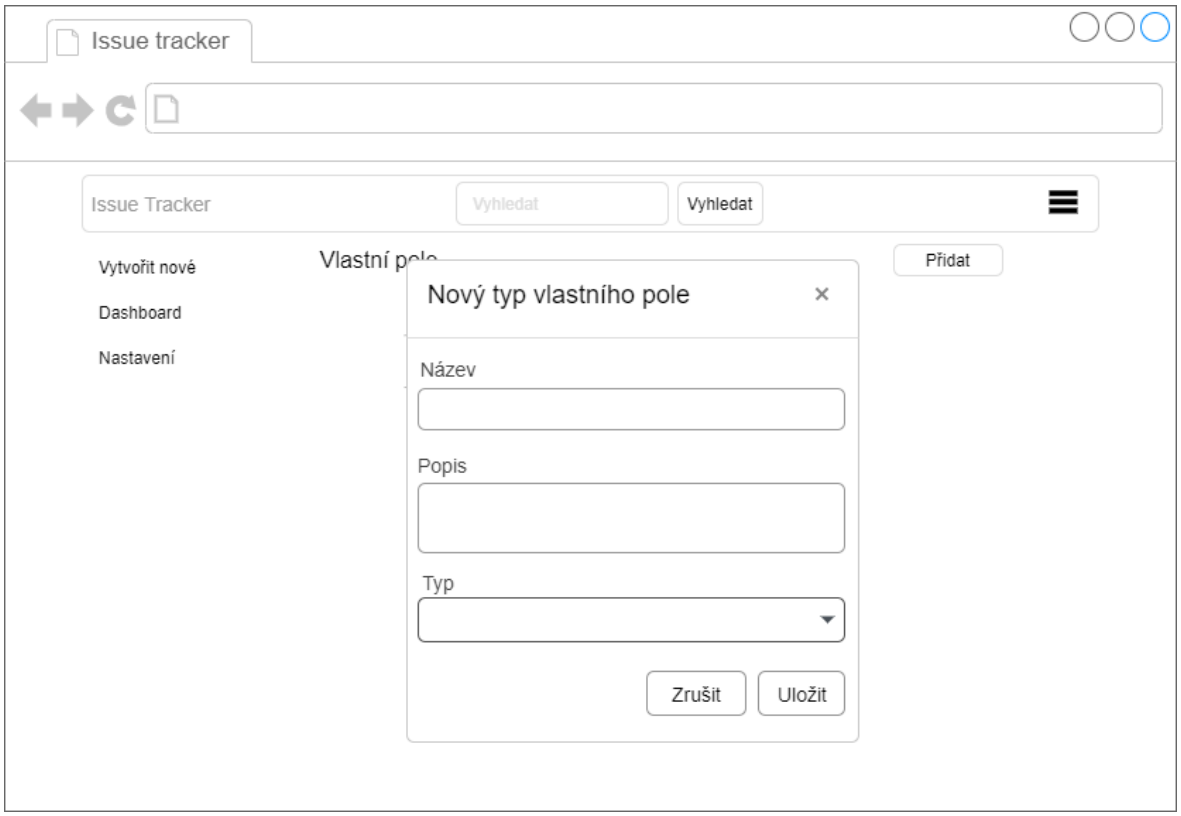


Obrázek 13 Wireframe Správa workflow

Další wireframe ukazuje návrh stránky pro správu typů vlastních polí. Zde se bude nacházet tabulka s již existujícími typy. U každého typu je popsán název pole, jeho popis a typ a na konci řádku je zobrazeno tlačítko pro smazání. Nahoře v pravém rohu stránky se nachází tlačítko Přidat. Toto tlačítko otevře modální okno pro přidání nového typu vlastního pole. Zbylé obrazovky v nastavení aplikace jsou velmi podobné této a jejich wireframy tedy nebudou v této práci zahrnuty.



Obrázek 14 Wireframe Správa vlastních polí



Obrázek 15 Wireframe Nový typ vlastního pole

4.3 Implementace

Jako IDE neboli vývojové prostředí, bude použita IntelliJ IDEA, která byla vybrána především z důvodu osobní zkušenosti autora s touto aplikací. IDEA poskytuje perfektní podporu pro vývoj Java aplikací s integrací na Spring framework a zároveň s ní lze po stažení pluginu pohodlně vyvíjet i Angular aplikaci. Mimo to je k dispozici mnoho jiných pluginů, které je možné si doinstalovat pro ještě pohodlnější vývoj.[29]

4.3.1 Vytvoření projektu

Projekt vytvoříme přímo ve vybraném IDE. V samotné Intellij IDEA je zabudovaný Spring Initializr, přes který se Spring Boot aplikace vytváří velmi snadno. Stačí zadat základní informace, jako název projektu, jaký sestavovací systém bude použit, na jaké verzi Javy aplikace poběží atd., a poté je již možné vybrat si závislosti, které pro vývoj bude potřebovat, nicméně ty se dají doplnit i později během samotného vývoje. Poté již stačí samotný projekt vygenerovat a můžeme začít pracovat.

Aby bylo možné vytvořit Angular projekt, musí být na pracovní stanici nainstalován Node.js, což je platforma, díky které je možné spouštět kód napsaný v JavaScriptu mimo klasický webový prohlížeč. Následně si můžeme v IDE vytvořit nový projekt přes zabudovaný generátor Angular CLI.

4.3.2 Struktura projektu

Jelikož byl pro projekt využit sestavovací systém Maven, struktura Spring aplikace z toho vychází. Stejně tak jako v případě Angular aplikace, struktura složek je z velké části dána samotným frameworkem. Při vytváření aplikace bylo rozhodnuto, že se backend a frontend budou nacházet v jednom projektu. V IDE byl nejdříve vytvořen Spring projekt a v jeho složce webapp poté Angular aplikace.

```
+---main
|   +---java
|   |   \---cz
|   |       \---czu
|   |           \---pef
|   |               \---issuetracker
|   |                   +---config
|   |                   +---controller
|   |                   |   \---vm
|   |                   +---domain
|   |                   +---repository
|   |                   |   \---specifications
|   |                   +---security
|   |                   |   \---jwt
|   |                   \---service
|   |                       +---dto
|   |                       +---impl
|   |                       \---mapper
|   +---resources
|   |   +---static
|   |   \---templates
\---webapp
    +---e2e
    |   \---src
    \---src
        +---app
        |   +---guard
        |   +---header
        |   +---issue
        |   |   +---dashboard
        |   |   +---issue-create
        |   |   +---issue-detail
        |   |   \---search
        |   +---login
        |   +---logout
        |   +---model
        |   +---service
        |   +---settings
        |   |   +---create-user
        |   |   +---custom-field-settings
        |   |   +---project-settings
        |   |   +---state-settings
        |   |   +---type-settings
        |   |   +---user-settings
        |   |   +---user-settings-update
        |   |   \---workflow-settings
        |   \---sidebar
        +---assets
                    \---environments
```

Nejdříve si projdeme strukturu backendové Spring aplikace. V balíčku cz.czu.pef.issuetracker.config se nachází konfigurační soubory aplikace, konkrétně SecurityConfiguration.java, který se stará o zabezpečení backendové aplikace a bude více popsán v kapitole 4.3.4 Zabezpečení aplikace. Další balíček cz.czu.pef.issuetracker.controller obsahuje controllery, které se starají o obsluhování HTTP požadavků od frontendu aplikace. V balíčku cz.czu.pef.issuetracker.controller.vm se nachází virtuální modely JwtRequest.java a JwtResponse.java, určené pro reprezentaci

objektu požadavku a odpovědí pro JWT token, taktéž bude více popsáno v kapitole o zabezpečení aplikace. Balíček cz.czu.pef.issuetracker.domain obsahuje třídy reprezentující databázové entity. V cz.czu.pef.issuetracker.repository jsou rozhraní sloužící pro komunikaci s datovou vrstvou, v našem případě s databází MySQL. V cz.czu.pef.issuetracker.repository.specifications se poté nachází třídy sloužící pro podporu filtrování při vyhledávání dat v databázi. Balíček cz.czu.pef.issuetracker.security slouží pro zabezpečení a bude mu věnována pozornost v kapitole týkající se zabezpečení aplikace. Dalším balíčkem je cz.czu.pef.issuetracker.service, který reprezentuje servisní vrstvu. Zde se nachází rozhraní služeb aplikace, která poskytují metody pro práci s daty bez závislosti na datové vrstvě. V balíčku cz.czu.pef.issuetracker.service.dto jsou třídy využívající návrhový vzor Data Transfer Object. Ten slouží pro zapouzdření dat, která se chystáme odesílat ke klientovi. Na to navazuje balíček cz.czu.pef.issuetracker.service.mapper, obsahující mapovací třídy, které poskytují metody pro mapování entitních tříd do DTO tříd a naopak. Posledním balíčkem Spring aplikace je cz.czu.pef.issuetracker.service.impl, kde se nachází implementace jednotlivých servisních rozhraní. Zde je využit návrhový vzor Strategy, kde máme jedno rozhraní, které může mít několik implementací a o výběr konkrétní implementaci, která bude použita, se rozhoduje až při samotném běhu aplikace.

Co se týče frontendové aplikce, ta se nachází ve složce webapp a nás primárně zajímá složka src/app/. Jako první se zde nachází složka guard, kde se nachází guardy aplikace. Ty slouží při požadavku na přístup na jednotlivé stránky pro rozhodnutí, zda aktuální uživatel má právo na stránku přistoupit. Vice bude popsáno v kapitole týkající se zabezpečení aplikace. Ve složce header se nachází komponenta pro hlavičku stránek. Ve složce issue se nachází složky komponent týkající se issue, konkrétně dashboard, vytvoření issue, detail issue a vyhledávání. Složka login obsahuje komponentu pro přihlašovací obrazovku aplikace a ve složce je komponenta pro odhlášení. V modelu se nachází všechny entity. Složka service obsahuje servisní třídy, což jsou třídy, které obsahují části kódu, které je možné opakovaně využívat napříč aplikací. Ve složce settings nalezneme složky pro komponenty, které jsou pro sekci nastavení. A nakonec složka sidebar, kde se nachází komponenta levého menu.

4.3.3 Připojení k databázi

Jak již bylo zmíněno, pro databázi aplikace bude použito MySQL. K této databázi je nutné připojit backendovou část vyvíjené aplikace. Aby připojení bylo možné, je potřeba mít správný konektor. V našem konkrétním případě se jedná o MySQL Connector, který do projektu lze přidat jako Maven závislost. Stačí tedy do POM souboru přidat následující kód:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Obrázek 16 Maven závislost MySQL Connector

Po jeho naimportováni je Spring framework již schopný se k databázi připojit a my mu musíme dodat informace k jaké konkrétní databázi se připojit. Toto nastavení se provádí v souboru application.properties ve složce resources. Zde je primárně podstatné nastavit na jaké url se databázový server nachází a uživatelské jméno a heslo, kterým se k němu lze připojit. Jelikož je pro komunikaci s datovou vrstvou využitý Spring Data JPA, který v základu poskytuje konfiguraci pro Hibernate, musíme nastavit správný dialekt pro MySQL. Vlastnost ddl-auto nastavíme na none, protože nechceme, aby nám Hibernate upravoval strukturu databáze. Poté se již můžeme zkusit k databázi připojit, a pokud jsme zadali správné přístupové údaje, měla by se aplikace bez problému připojit.

```
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
spring.datasource.url=jdbc:mysql://localhost:3306/issue_tracker?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=admin
```

Obrázek 17 Soubor application.properties

4.3.4 Zabezpečení aplikace

Jelikož jeden z požadavků na aplikaci byl, že je možné s ní pracovat pouze po přihlášení a bez něho do ní nemá uživatel přístup, je nutné celou aplikace zabezpečit proti neoprávněnému přístupu. Toto zabezpečení je potřeba udělat jak na backendové části, tak zároveň i na straně frontendu.

Nejdříve si ukážeme zabezpečení na straně serveru. Backend vystavuje endpointy, které jsou konzumované klienty. Aby endpointy mohli využívat pouze ověření klienti, je třeba je zabezpečit. Použitý Spring framework poskytuje modul týkající se zabezpečení nazvaný Spring Security. Aby bylo možné ho využívat, je nutné ho přidat jako Maven závislost do POM souboru projektu.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Obrázek 18 Maven závislost Spring Security

Jako další krok je potřeba vytvořit konfigurační soubor, který bude o zabezpečení rozhodovat. V našem případě je to konkrétně SecurityConfiguration.java. Zde se zaměříme na metodu configure, který nastavuje třídu HttpSecurity. V této metodě nastavujeme třídu zpracovávající výjimky, v našem případě vlastní implementaci rozhraní AuthenticationEntryPoint, která odmítne všechny neautorizované požadavky. Dále je potřeba nastavit parametr pro Cross-origin resource sharing, díky čemuž je možné se na endpointy připojit z jiné domény. To je potřeba z důvodu, že Angular aplikace neběží na stejné doméně jako server. Jelikož se jedná o aplikaci postavenou na RESTu, natavujeme zde bezstavovost. Jako další věc máme nastavení přístupu k endpointům. Adresa /auth slouží pro přihlášení a dostane se na ni každý. Pro přístup k jakémukoliv jinému zdroji na serveru je již nutné být přihlášený. Na závěr nastavujeme filtr, který bude pro každý příchozí požadavek ověřovat validitu tokenu, kterým se klient prokazuje. Filtr je vlastní implementací třídy OncePerRequestFilter.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
            .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .authorizeRequests().antMatchers( ...antPatterns: "/auth").permitAll()
            .anyRequest().authenticated();

    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
}
```

Obrázek 19 Metoda nastavující zabezpečení aplikace

Dále máme třídu JwtTokenUtils, která se stará o generování a validací tokenu. Jak je z názvu třídy patrné, jedná se o JWT token, což je zkratka z JSON Web Token, který

se používá jako standard pro přístupové tokeny. Token má tři části, header, payload a signature, které jsou rozdělené tečkami, a je zakódovaný pomocí Base64Url. Token může vypadat následovně:

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyIiwicm9sZXMiOlt7ImF1dGhvcml0eSI6IlJPTEVfVVNF
UiJ9XSwiaWF0IjoxNTg0NTY0NDM4LCJleHAiOjE1ODQ5MjQ0Mzh9.zJBRZwNLeRMsUynAwUpgrDHA2gBE
33FpyfZkANfOyJs7oUbhcURahBtpK_yUhkNjbPuVvUgoE3wZ3U96jt04hA
```

Pro vytvoření tokenu je určená metoda generateJwtToken. Do tokenu uložíme uživatelské jméno, role uživatele, nastavíme čas vytvoření tokenu a čas expirace, a poté provedeme zakódování pomocí algoritmu HS512 a přidáme jwtSecret, což je tajný klíč, díky tomu můžeme ověřit, že příchozí token byl skutečně vytvořen naší aplikací.

```
public String generateJwtToken(Authentication authentication) {

    UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();
    Claims claims = Jwts.claims().setSubject(userPrincipal.getUsername());
    claims.put("roles", userPrincipal.getAuthorities());

    return Jwts.builder()
            .setSubject((userPrincipal.getUsername()))
            .setClaims(claims)
            .setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
}
```

Obrázek 20 Metoda pro generování JWT

Jako další krok vytvoříme controller pro příjmení požadavků pro přihlášení na endpointu /auth, který bude přijímat JwtRequest obsahující uživatelské jméno a heslo, které pomocí AuthenticationManager zkontroluje, a pokud se jedná o validně zadané údaje, vrátí jako odpověď vygenerovaný token.

```
@PostMapping("/auth")
public ResponseEntity<?> createAuthenticationToken(@Valid @RequestBody JwtRequest loginRequest) {

    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));

    final String token = jwtTokenUtils.generateJwtToken(authentication);

    return ResponseEntity.ok(new JwtResponse(token));
}
```

Obrázek 21 Endpoint pro přihlášení uživatele

V implementované aplikaci jsou některé oblasti přístupné pouze pro roli Admin. Je samozřejmé, že na frontendu by mělo být omezení takové, že se uživatel bez této role na chráněné funkčnosti nedostane, ale na to bychom neměli v rámci backendu spoléhat, a i tak omezit přístupy k těmto funkčnostem. Toto lze provést velmi jednoduše pomocí anotace @PreAuthorize. V obrázku je ukázáno použití anotace v ProjectController na metodě pro vytvoření projektu.

```
@PreAuthorize("hasRole('ADMIN')")
@PostMapping(URI)
public ResponseEntity<ProjectDTO> createProject(@RequestBody ProjectDTO projectDto) throws URISyntaxException {
    log.info("Projects - request to create project: {}", projectDto);
    ProjectDTO createdProject = projectService.create(projectDto);
    return ResponseEntity.created(new URI( str: URI + createdProject.getIdProject())).body(createdProject);
}
```

Obrázek 22 Zabezpečení endpointu pomocí rolí

Tímto můžeme považovat zabezpečení Spring aplikace za hotové a je potřeba vyřešit i Angular, který na to již ve svém základu poskytuje nástroje. Jako první vytvoříme službu pro autentizaci. Zde máme implementovanou metodu, která pošle požadavek na endpoint serveru /auth s uživatelským jménem a heslem. Při úspěšném požadavku si musíme uložit token, který jsme od backendu získali. Zde se nám nabízejí dvě možnosti, sessionStorage a localStorage. SessionStorage ukládá informace jen po dobu otevření daného prohlížeče a poté se smažou. LocalStorage ukládá informace natrvalo, dokud nedojde k jejich smazání. Z tohoto pohledu se sessionStorage jeví jako bezpečnější varianta, ale jelikož u implementované aplikace nechceme, aby se musel uživatel při každém přístupu na stránku přihlašovat, bylo zvoleno uložení do localStorage. Do localStorage uložíme příchozí token a taktéž uživatelské jméno.

```
authenticate(username: string, password: string) {
  return this.httpClient.post<any>( url: 'http://localhost:8080/auth', body: {username, password})
    .pipe(
      map(
        project: user => {
          let token = 'Bearer '+ user.token;
          localStorage.setItem('token', token);
          localStorage.setItem('username', username);
          this.loggedIn.next( value: true);
          return user;
        }
      )
    )
}
```

Obrázek 23 Metoda pro získání tokenu

Takto uložený token budeme poté používat při volání dalších REST služeb a ověříme se jím tak serveru. K tomu úkonu slouží v Angularu rozhraní HttpInterceptor, díky kterému můžeme ke každému odesílanému požadavku přiřadit dodatečné informace. To se primárně využívá pro přidání tokenu do požadavků na server. Při implementaci tohoto rozhraní musíme implementovat metodu intercept. V ní zkontrolujeme, zda máme v localStorage uložený token a pokud ano, přidáme ho do hlavičky požadavku.

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  if (localStorage.getItem( key: 'token')) {
    req = req.clone( update: {
      setHeaders: {
        Authorization: localStorage.getItem( key: 'token')
      }
    })
  }

  return next.handle(req);
}
```

Obrázek 24 Metoda pro vkládání tokenu do hlavičky požadavku

Jako poslední musíme omezit přístupy na určité stránky jen pro vybrané uživatelské role. Na to využijeme tzv. guardy. Jako první implementuje guard, který bude omezovat přístup na všechny stránky, pokud uživatel nebude přihlášen. Takový uživatel se dostane pouze na přihlašovací obrazovku. Všechny guardy musí implementovat rozhraní CanActivate a jeho metodu canActivate. V této metodě ověříme, zda je uživatel přihlášený. Pokud ne, redirectujeme ho na obrazovku přihlášení. Pokud přihlášený je, vpustíme ho na stránku, kterou požadoval navštívit.

```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
  return this.authService.isLoggedIn()
    .pipe(
      take( count: 1),
      map( project: (isLoggedIn: boolean) => {
        if (!isLoggedIn) {
          this.router.navigate( commands: ['/login']);
          return false;
        }
        return true;
      })
    )
}
```

Obrázek 25 Implementace metody canActivate pro AuthGuard

Mimo tento základní guard vytvoříme ještě jeden, který bude přístup omezovat dle role, která je pro danou obrazovku vyžadována. Zde z tokenu uživatele získáme informace o jeho rolích. Pokud má oprávnění, které je vyžadováno pro přístup, je mu umožněn, v opačném případě je redirectován na základní stránku aplikace

```
canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
  const user = this.authService.decode();
  let authorities = [];

  user.roles.map( callbackfn: role => authorities.push(role.authority));
  let role = authorities.filter( callbackfn: authority => next.data.role.includes(authority));
  if (role.length > 0) {
    return true;
  }

  this.router.navigate( commands: ['/']);
  return false;
}
```

Obrázek 26 Implementace metody canActivate pro RoleGuard

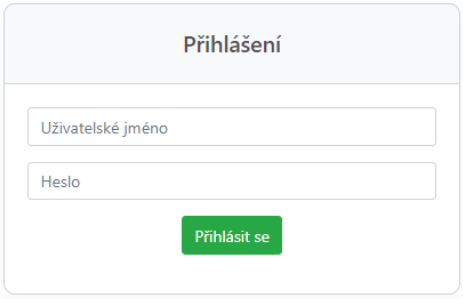
Poté co máme guardy vytvořené, je třeba specifikovat, na jaké cesty mají být použity. To provedeme v souboru pro routing aplikace. Na obrázku vidíme, že na cestu login není použít žádný guard, a tím pádem se na ni dostane kdokoliv. Na další cesty jako například dashboard, new-issue atd. už je použit pomocí parametru canActivate AuthGuard, který zabrání nepřihlášenému uživateli se na tyto komponenty dostat. Jelikož je sekce nastavení přístupná jen pro uživatele s rolí Admin, jsou všechny dané cesty chráněny pomocí RoleGuard, a v parametru data jsou specifikované role, které musí uživatel mít. V našem případě ROLE_ADMIN.

```
const routes: Routes = [
  {path: '', component: DashboardComponent, canActivate: [AuthGuard]},
  {path: 'login', component: LoginComponent},
  {path: 'logout', component: LogoutComponent, canActivate: [AuthGuard]},
  {path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard]},
  {path: 'new-issue', component: IssueCreateComponent, canActivate: [AuthGuard]},
  {path: 'issue/:id', component: IssueDetailComponent, canActivate: [AuthGuard]},
  {path: 'search/:summary', component: SearchComponent, canActivate: [AuthGuard]},
  {path: 'settings/types', component: TypeSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/states', component: StateSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/custom-fields', component: CustomFieldSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/workflow', component: WorkflowSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/projects', component: ProjectSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/users', component: UserSettingsComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/users/edit/:id', component: UserSettingsUpdateComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}},
  {path: 'settings/users/create', component: CreateUserComponent, canActivate: [RoleGuard], data: {role: ['ROLE_ADMIN']}}
];
```

Obrázek 27 Nastavení cest a přístupů v Angular aplikaci

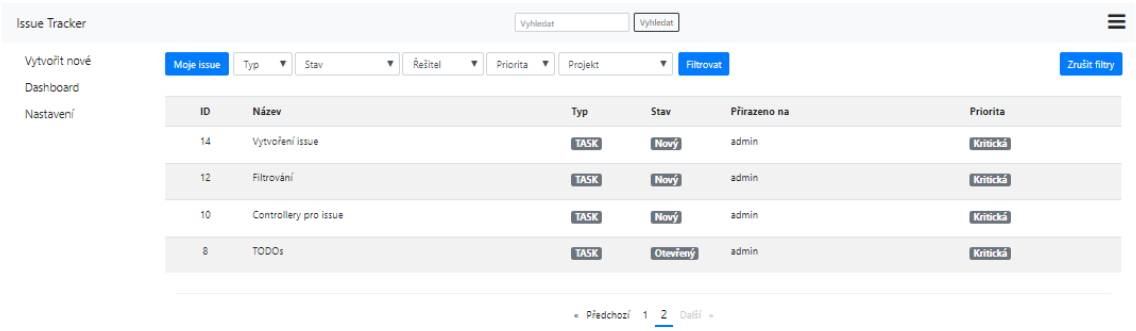
4.3.5 Realizace obrazovek

Pro vytvoření stylu obrazovek byl použit Boostrap, což je sada open source nástrojů ulehčující práci s HTML a CSS. V této kapitole bude ukázáno pár implementovaných obrazovek, aby bylo možné provést porovnání s navrhnutými wireframy z kapitoly 4.2.5. Jelikož je do aplikace nutné se přihlásit, jako první uživatel vidí obrazovku přihlášení.

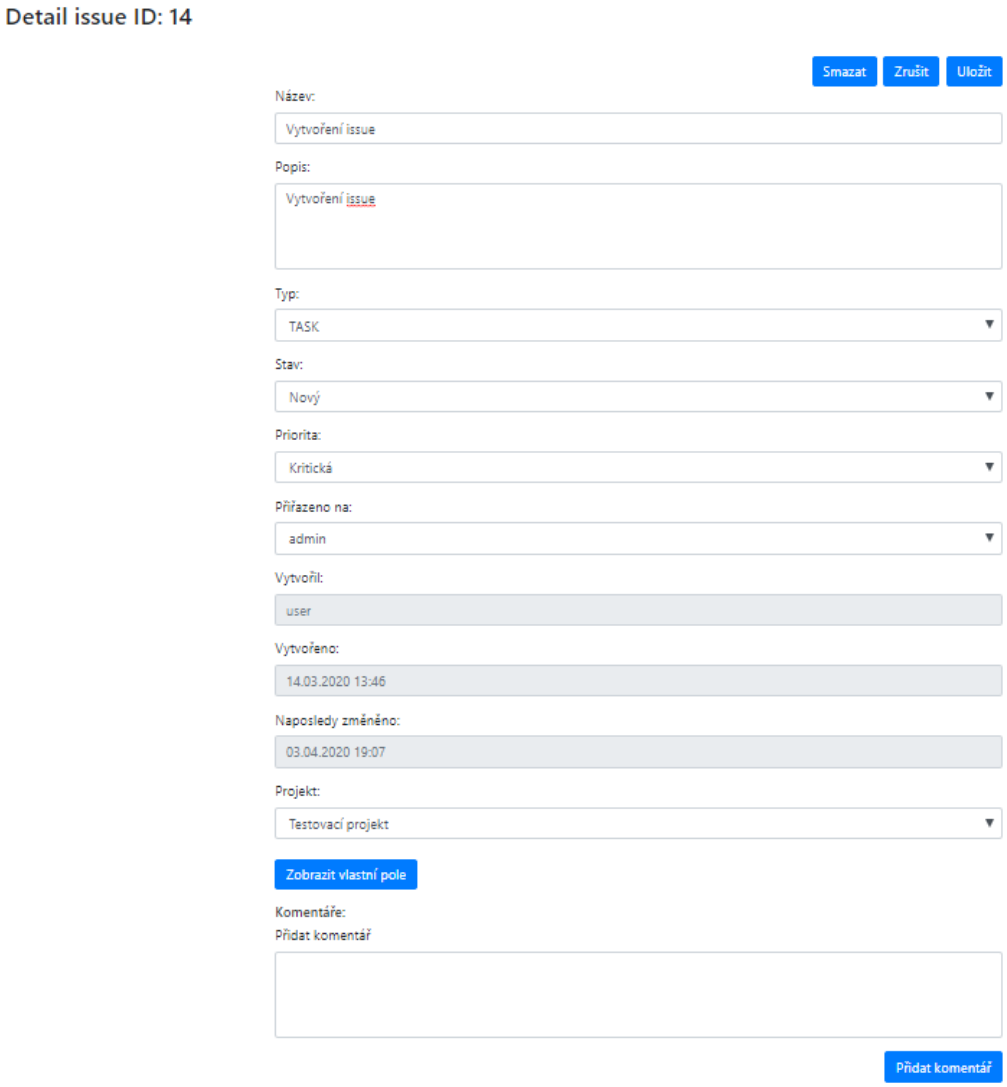


Obrázek 28 Obrazovka přihlášení

Po úspěšném přihlášení, se uživatel ocitne na obrazovce dashboard, kde vidí všechny zadané issue, mezi kterými může filtrovat. Při kliku na konkrétní issue se uživatel dostane na obrazovku detail.



Obrázek 29 Obrazovka dashboard



Obrázek 30 Obrazovka detail issue

4.4 Testování

Testování aplikace bylo prováděno po celou dobu implementace aplikace autorem pro ověření správné funkčnosti implementovaných částí. Po dokončení implementace byl systém kompletně manuálně otestován. Pro testování nebyly vytvořené konkrétní testovací scénáře. Namísto toho byl použit use case diagram a jeho scénáře, podle kterých byl prováděn samotný vývoj, a poté taktéž testování.

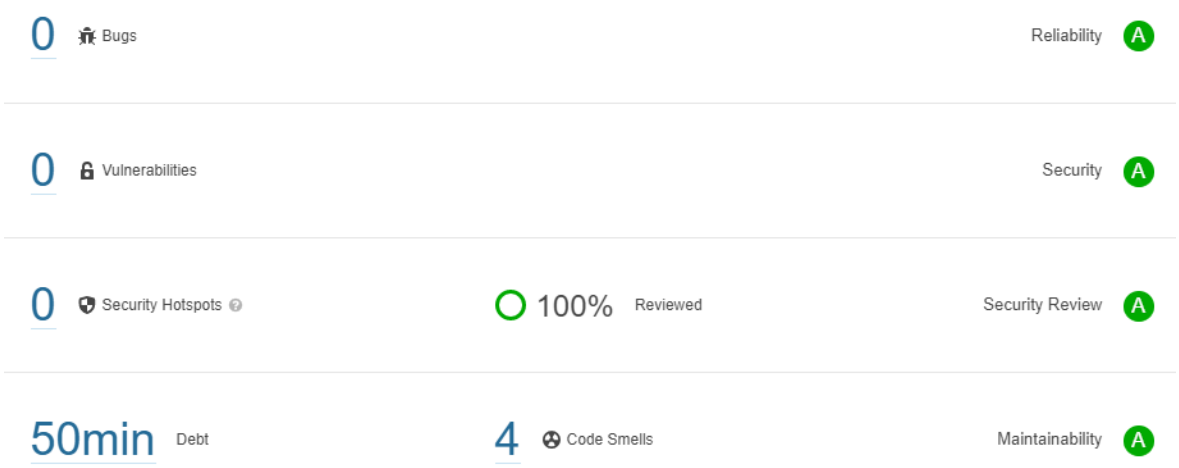
Dále byla provedena statická analýza kódu pomocí nástroje SonarQube verze 8.2. Tento nástroj provádí automatickou kontrolu kódu, který umí detekovat chyby nebo třeba

zranitelnosti systému. Aby bylo možné v aplikaci SonarQube využít, je nutné ho vložit do POM souboru jako plugin. [28]

```
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
</plugin>
```

Obrázek 31 Maven závislost SonarQube

Poté je již možné spustit sken celého kódu a nechat si zobrazit výsledky. Ve výsledcích je poté možné sledovat, kolik chyb SonarQube zachytil, kolik zranitelností a bezpečnostních rizik se v systému nachází a taktéž kolik je v kódu tzv. „code smells“, což jsou indikátory toho, že by v kódu mohl být nějaký hlubší problém



Obrázek 32 Výsledky v SonarQube

5 Výsledky a diskuse

V následující kapitole si shrneme výsledky dosažené v rámci této diplomové práce a možnosti rozšíření aplikace. Primárním úkolem byl návrh a implementace webové aplikace pro issue tracking. Aplikace měla být určena především pro použití pro malé a střední firmy či týmy a měla obsahovat základní sadu funkcí, které jsou pro issue tracking potřeba. To se v rámci této práce povedlo a výsledkem je plně funkční webová aplikace. V aplikaci je možná práce s issue, která odpovídá zadaným požadavkům. Lze je vytvářet a upravovat, přiřazovat na uživatele v systému, přepínat jejich stavy atd. V aplikaci je taktéž možnost pro přidávání nových stavů a též úprava workflow. Dále je možné přidávat nové typy, projekty a vlastní pole na issue. Tyto funkce umožnují týmu, který aplikaci používá, si do značné míry modifikovat proces issue trackingu tak, aby mu co nejvíce vyhovoval. Taktéž je dle požadavku vstup do aplikace možný pouze pro přihlášené uživatele.

Aplikace jako taková je momentálně ve své první verzi a šla by samozřejmě rozšiřovat o další funkce. Jedná z funkcí, která by se velmi hodila, je notifikace v případě přirazení issue na uživatele. V takovém případě by uživateli mohl přijít e-mail, aby o tom byl informován. Funkci, která by jistě našla uplatnění, by byl i export dat z aplikace, pravděpodobně ve formátu pro Excel. Taktéž by bylo možné zařazovat funkce podporující agilní způsob vývoje softwaru, který je v posledních letech velmi používaný, ale tím už by se aplikace dostávala více do projektového řízení a nejednalo by se o aplikaci čistě na issue tracking. Další možnou úpravou by mohly být jazykové mutace. Momentálně je možné využívat pouze českou verzi. S využitím některé z knihoven, které jsou pro Angular dostupné, by bylo možné aplikaci nabízet ve více jazykových variantách a tím rozšířit potencionální klientelu.

6 Závěr

Hlavním cílem diplomové práce byl návrh a implementace webové aplikace pro issue tracking. Dílčím cílem bylo popsat proces issue trackingu, představit vybrané existující aplikace spadající do dané kategorie a přiblížení technologií, které byly pro implementaci aplikace využity.

V teoretické části práce byl představen proces issue trackingu a jeho podstata při vývoji softwaru. Bylo popsáno, co je issue a workflow. Poté byli vybráni čtyři nejrozšířenější zástupci toho softwaru na trhu. Při vybírání se též přihlíželo k faktu, aby každý z nich měl trochu jiné zaměření. Jednalo se konkrétně o aplikace Jira, Azure DevOps, YouTrack a Bugzilla. Nástroje byly detailně popsány, byly představeny jejich funkce, výhody a nevýhody jejich používání. Další částí v teoretických východiskách bylo popsání technologií a postupů které byly poté použité v praktické části práce při implementaci výsledného řešení. V těchto kapitolách byly např. představeny programovací jazyky Java a TypeScript a frameworky Spring a Angular, návrhové vzory, vývojové principy nebo třeba architektonický styl REST.

Praktická část již obsahovala samotný návrh a implementaci aplikace. Zde byla nejdříve provedená analýza, při které byly specifikovány funkční a nefunkční požadavky, které by měla výsledná implementace splňovat. Následovala část návrhu systému, kde byl vytvořen diagram případů užití a scénáře případů užití. Následovalo navrhnutí datového modelu se všemi tabulkami, které se v databázi budou nacházet. Částí návrhu je také stavový diagram, který představuje základní workflow v systému. Dále byly v této části představeny wireframy obrazovek, podle kterých byly vytvořeny obrazovky v aplikaci. Poté již přišla na řadu samotná implementace aplikace. Ta pro backendovou část využívá jazyk Java se Spring frameworkem a frontend byl implementován ve frameworku Angular s použitím jazyka TypeScript. Frontendová a backendová část spolu komunikují pomocí REST rozhraní. Aplikace byla nakonec otestována, byly zhodnoceny výsledky práce a byla navržena možná rozšíření.

Výstupem této diplomové práce je plně funkční webová aplikace, která splňuje zadané požadavky a poskytuje základní funkce pro potřeby issue trackingu.

7 Seznam použitých zdrojů

[1] BERTRAM, Dane, Amy VOIDA, Saul GREENBERG a Robert WALKER. Communication, collaboration, and bugs. *Proceedings of the 2010 ACM conference on Computer supported cooperative work - CSCW '10* [online]. New York, New York, USA: ACM Press, 2010, 2010, , 291- [cit. 2020-04-02]. DOI: 10.1145/1718918.1718972. ISBN 9781605587950. Dostupné z: http://portal.acm.org/citation.cfm?doid=1718918.1718972

[2] *Atlassian Documentation* [online]. [cit. 2020-04-02]. Dostupné z: https://confluence.atlassian.com/

[3] Atlassian [online]. [cit. 2020-04-02]. Dostupné z: https://www.atlassian.com/

[4] *Bugzilla* [online]. [cit. 2020-04-02]. Dostupné z: https://www.bugzilla.org/

[5] *JetBrains* [online]. [cit. 2020-04-02]. Dostupné z: https://www.jetbrains.com/

[6] *JetBrains Blog* [online]. [cit. 2020-04-02]. Dostupné z: https://blog.jetbrains.com/

[7] *Microsoft Azure* [online]. [cit. 2020-04-02]. Dostupné z: https://azure.microsoft.com/

[8] *Microsoft Docs* [online]. [cit. 2020-04-02]. Dostupné z: https://docs.microsoft.com/

[9] *Visual Studio Marketplace* [online]. [cit. 2020-04-02]. Dostupné z: https://marketplace.visualstudio.com/

[10] *Spring* [online]. [cit. 2020-04-02]. Dostupné z: https://spring.io/

[11] *Apache Maven Project* [online]. [cit. 2020-04-02]. Dostupné z: http://maven.apache.org/

[12] *Oracle MySQL* [online]. [cit. 2020-04-02]. Dostupné z: https://www.oracle.com/mysql/

[13] *Angular* [online]. [cit. 2020-04-02]. Dostupné z: https://angular.io/

[14] SCHILDT, Herbert. *Java 7: výukový kurz*. 2nd ed. Brno: Computer Press, 2012. Moderní programování. ISBN 978-80-251-3748-2.

[15] *Stack Overflow: Developer Survey Results 2019* [online]. [cit. 2020-04-02]. Dostupné z: https://insights.stackoverflow.com/survey/2019

[16] *TypeScript* [online]. [cit. 2020-04-02]. Dostupné z: https://www.typescriptlang.org/

[17] *Oracle: Java SE Subscription* [online]. [cit. 2020-04-02]. Dostupné z: https://www.oracle.com/technetwork/java/javaseproducts/overview/index.html

[18] *Oracle: Java Documentation* [online]. [cit. 2020-04-02]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/

[19] FENTON, Steve. *Pro TypeScript: application-scale JavaScript development*. New York, N.Y.: Apress, 2014. ISBN 978-148-4232-484.

[20] BEAULIEU, Alan. *Learning SQL: stavební kameny objektově orientovaných programů*. 2nd ed. Sebastopol: O'Reilly, 2009. Moderní programování. ISBN 978-059-6520-830.

[21] *REST API* [online]. [cit. 2020-04-02]. Dostupné z: https://restfulapi.net/

[22] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000

[23] *Spring Framework Reference Documentation* [online]. [cit. 2020-04-02]. Dostupné z: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/

[24] PECINOVSKÝ, Rudolf. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[25] GAMMA, Erich. *Návrh programů pomocí vzorů: stavební kameny objektově orientovaných programů*. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0302-5.

[26] *HTTP/1.1: Method Definitions* [online]. [cit. 2020-04-02]. Dostupné z: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[27] *MySQL: Workbench* [online]. [cit. 2020-04-02]. Dostupné z: https://www.mysql.com/products/workbench/

[28] *SonarQube* [online]. [cit. 2020-04-02]. Dostupné z: https://www.sonarqube.org/

[29] *IntelliJ IDEA* [online]. [cit. 2020-04-02]. Dostupné z: https://www.jetbrains.com/idea/