Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Realizace webové aplikace Helpdesk pomocí vhodného testovacího nástroje

Bakalářská práce

Autor: Zbyněk Richter
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Filip Malý, Ph.D.

Hradec Králové červenec 2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 9.8.2016 Zbyněk Richter

Poděkování:

Děkuji vedoucímu své bakalářské práce doc. Ing. Filipu Malému, Ph.D. za metodické vedení práce a cenné rady během tvorby této práce.

Anotace

Tato práce se zabývá vývojem webové aplikace Helpdesk pomocí vhodného testovacího nástroje. Teoretická část definuje pojem helpdesk a popisuje systémy, které firma Tender systems s. r. o. používá. Dále vysvětluje důvod vývoje vlastního softwaru sloužícího k zadávání a řešení požadavků. V další kapitole teoretické části se práce zabývá popisem a výběrem nejvhodnějšího testovacího nástroje, který je použit při vývoji aplikace. Praktická část popisuje návrh a realizaci vlastní webové aplikace.

Annotation

The bachelor thesis deals with the development of web application Helpdesk with the help of suitable testing tool. The theoretical part defines the term helpdesk and describes the systems used by company Tender systems s. r. o. It also explains the reasons for development of software used for setting and solving requests. The next chapter of the theoretical part deals with the description and choice of the most suitable testing tool used for the development of the application. The practical part describes the draft and realization of the web application itself.

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam tabulek

1 Úvod

Předmětem bakalářské práce je vhodným způsobem vytvořit webovou aplikaci s názvem Helpdesk na platformě Java EE. Klíčovým prvkem vývoje je najít vhodný testovací nástroj pro nejkomplexnější testování aplikace a zajištění nejvyšší kvality softwaru. V době vysokého hardwarového vývoje narůstá tlak na vysokou kvalitu softwaru. Díky tomu dochází k prudkému nárůstu používání agilních metodik vývoje.

Produkty, které firmy vytvoří, musí být vysoce spolehlivé a stabilní, tj. nemělo by docházet k výpadkům. K vývoji aplikací by měly být použity nejnovější technologie, ale zároveň ty, které fungují na co nejvíce strojích. Z hlediska webové aplikace se jedná především o prohlížeče, které zákazník používá. Aby byl produkt kvalitní, musí být také bezpečný. V době, kdy se takřka denně dozvídáme z nejrůznějších informačních zdrojů o tzv. kyberútocích, musí být aplikace vyvíjena tak, aby se zabránilo jakémukoliv samovolnému proniknutí do aplikace.

Pokud se v aplikaci, kterou zákazník používá, vyskytne problém, informuje provozovatele. Problémy se vkládají do systému, který si provozovatel sám vytvoří nebo si ho pronajme. Vlastník systému garantuje vyřešení problému v určitém časovém úseku, který je uveden v uzavřené smlouvě.

V této práci si ukážeme současné řešení firmy Tender systems s. r. o., které firma používá. V praktické části navrhneme vlastní řešení helpdesku přímo na míru této firmě.

Firma Tender systems s. r. o. doposud ani na jednom svém produktu nepoužila způsob TDD (Test driven development). Cílem této bakalářské práce je najít vhodný testovací nástroj, který by mohla firma používat a také poukázat na jednotlivé výhody tohoto nástroje.

Druhá část bakalářské práce se zabývá praktickou tvorbou webové aplikace Helpdesk. Bude provedena a popsána kompletní analýza požadované oblasti. Budou popsány technologie, které budou použity při implementaci aplikace. K samotné implementaci bude použit nejvhodnější testovací nástroj, který bude vybrán v první části práce. Posledním dílčím krokem implementace bude nasazení do provozu. Na závěr práce budou zmíněny jednotlivé poznatky z vývoje aplikace formou TDD.

2 Popis zkoumané oblasti

Pod pojmem helpdesk se rozumí software, ve kterém pobíhá provoz technické podpory. Uživatel v roli zákazníka, tj. klienta, zadává ve webovém rozhraní svůj požadavek, který obsahuje odhalené problémy v produktu nabízeném provozovatelem. Helpdesk poslouží nespokojenému klientovi, který pracuje se systémem. Do helpdesku vloží klient jednoduchým způsobem svůj problém. Jedná se o komplexní software, který slouží pro podporu napříč celou společností, která klade důraz na kvalitu svého produktu a jeho maximální řešení požadavků. Zjednodušuje práci s požadavky, které jsou evidovány na jednom místě a obvykle řazeny do kategorií, kterými jsou defekty, reklamace, neshody, nebo to bývají pouze dotazy k určité funkcionalitě. Doména popisovaná v této kapitole je převážně vysvětlena na příkladech využití na informačních systémech. [1]

2.1 Princip fungování helpdesku

Na princip fungování systému lze nahlížet ze dvou pohledů, z pohledu klienta, který zadává požadavek, nebo řešitele. Nejprve je třeba říci, co vše systém eviduje a vykonává. Eviduje veškeré požadavky a jejich veškerou komunikaci mezi klientem a řešitelem formou komentářů. Upozorňuje obě strany na různé změny v rámci daného požadavku. Toto upozornění probíhá formu emailu. [2]

Klient si pomocí webového prohlížeče otevře technickou podporu u systému, ke kterému chce vložit svůj požadavek. Podle konkrétní implementace musí být uživatel přihlášen, ale nebývá to pravidlem. Po vložení příspěvku uživatel vidí všechny své zadané požadavky a má přehled o řešení. V rámci každého požadavku lze komunikovat formou komentářů s řešitelem a celá tato komunikace je evidována. [2]

Řešitel přebírá požadavek a řídí jeho kompletní vyřízení. U požadavku je měněn stav podle toho, v jaké fázi řešení zrovna je. Pracovníci v určitých časových intervalech vytváří klientům reporty, kde je uvedeno, zda se v době řešení vešly do tzv. SLA schématu, pokud se jedná o externí použití. Díky reportu zákazník vidí, zda je požadavek splněn ve sjednaném čase. V reportu jsou také uvedeny sankce za pozdní vyřešení požadavku, které může zákazník vymáhat. [2]

2.2 Využití helpdesku

Helpdesk lze využívat dvěma směry. Každý směr má odlišné spektrum využití. Helpdesk lze používat jako pomocníka pro hlášení defektů při vývoji, kdy probíhá interní komunikace mezi členy vývojového týmu. Druhým směrem využití aplikace je technická podpora pro klienty firmy. Firmy si vytváří svá řešení technické podpory na míru, mohou

kombinovat jak interní, tak externí. Zaměstnanci klienta mají přístup k uživatelské dokumentaci, kde je přesně uvedeno, kam mají svůj problém směřovat. [3]

2.2.1 Interní využití

Interní využití je nastaveno tak, jak si ho tým zvolí. Poskytuje užitečného pomocníka malým i komplexním týmům. Tento pomocník přináší velkou úsporu času v komunikaci mezi členy. Do systému se zapisují defekty během testování aplikace. Zprávy hlášené do systému bývají výsledkem kvality práce testera. Ohlašovat defekt však může i jiná osoba než tester např. vedoucí vývoje, manažer, který má přístup k používání aplikace. Aby bylo hlášení o defektu kvalitní, mělo by splňovat tyto pravidla:

- je definované co nejstručněji a přesně,
- uvádí se autor a datum vzniku,
- bývá přiložen soubor, ve kterém je například chyba z logu,
- obsahuje pouze jeden defekt, pokud v rámci tohoto defektu nevznikne další chyba,
- obsahuje informace o tom, jaká data byla v době testu zadávána,
- popisuje informaci o tom, jakým způsobem byl defekt vyvolán,
- je přiložen screen obrazovky s defektem, pokud je to možné. [1]

2.2.2 Externí využití

Externí produkt je cílený na uživatelskou skupinu klient – provozovatel. Jeho funkcí je pomáhat technické podpoře při komunikaci s klientem a řešit jeho požadavky. Na jedné straně tedy stojí klient a na druhé straně řešitel požadavku. [3]

Příkladem takového helpdesku je situace, při které klientovi nastane nestandardní stav aplikace, tj. aplikace spadne a klient nemůže pokračovat ve svém průchodu aplikací. Využije tedy technickou podporu k vyřešení svého problému a popíše svoje zjištění ve webovém formuláři, tzv. požadavek. Obsah požadavku je obdobný jako u interního využívání. Operátor na druhé straně převezme klientovu žádost, požadavek zpracuje a zašle jej do technického oddělení. Programátor (řešitel) problém převezme a zpracovává. Po vyřešení problému zašle řešitel zprávu klientovi o vyřešení požadavku. Klient má možnost komentovat řešení svého požadavku. [4]

2.3 SLA a reportování

Důležitým pojmem v oblasti poskytování služeb je SLA (Service Level Agreement). Tento pojem je používán hlavně v oblasti IT a označuje dohodu o úrovni poskytnutých

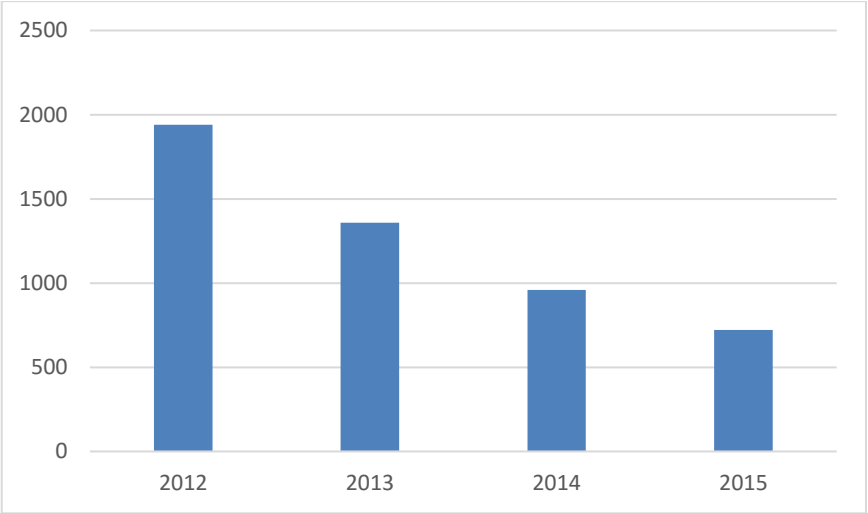
služeb. Jedná se o smlouvu, která popisuje služby poskytované klientovi provozovatelem. Ve smlouvě je definován rozsah, úroveň a kvalita služeb, které produkt musí splňovat. Provozovatel služby garantuje svým klientům rychlost řešení požadavků zadaných klientem od času jejich oznámení. Obvykle se čas na vyřešení požadavku odvíjí od pracovní doby provozovatele, ale vše záleží na dohodě mezi klientem a provozovatelem. Smlouva obsahuje sankce, které nastávají, pokud provozovatel poruší stanovené body. [9]

Smlouva také definuje tvorbu výročních reportů. Report je výstupní zpráva poskytovatele služby o kvalitě a schopnosti plnit garantované podmínky. Na základě tohoto reportu má klient pravomoc vymáhat sankce. Podle něho si může provozovatel vytvářet statistiky o tom, jak jeho zaměstnanci rychle řeší problémy a jakou chybovost má jeho poskytovaný produkt. [10]

2.4 Používaná řešení

V této kapitole bude popsáno, jaká řešení firma Tender systems s. r. o. po dobu své existence na trhu používala a v současné chvíli využívá. Je třeba říci něco o historii této společnosti. Firma vznikla v roce 2012, navazuje na činnost firem Gordion s. r. o. a Ample.cz s. r. o., které fungovaly na trhu od roku 2004. Firma za dobu své existence použila pro společné produkty dvě řešení informačních systémů technické podpory. Jako první se používal software JTrac, později bylo toto řešení ukončeno a přešlo se na TaskPool. V další části této práce budou obě řešení blíže popsána a bude uveden graf počtu požadavků za určité časové období. [11]

Firma poskytuje technickou podporu pro tři své produkty a počet požadavků eviduje od roku 2012 do současnosti. Pracovníci poskytli data k jednomu produktu. Z těchto dat vyplývá, že každým rokem se počet požadavků snižuje. Tento výsledek lze interpretovat ze dvou úhlů pohledu. Z prvního pohledu se dá říci, že systém je takřka dokonalý a zákazníci nemají důvod hlásit defekty. Z druhého úhlu to vypadá že se snižuje počet zákazníků. Podle slov jednatele však firma získává každým rokem nové zákazníky. Graf požadavků v letech 2012 až 2015 vypadá následovně. [11]



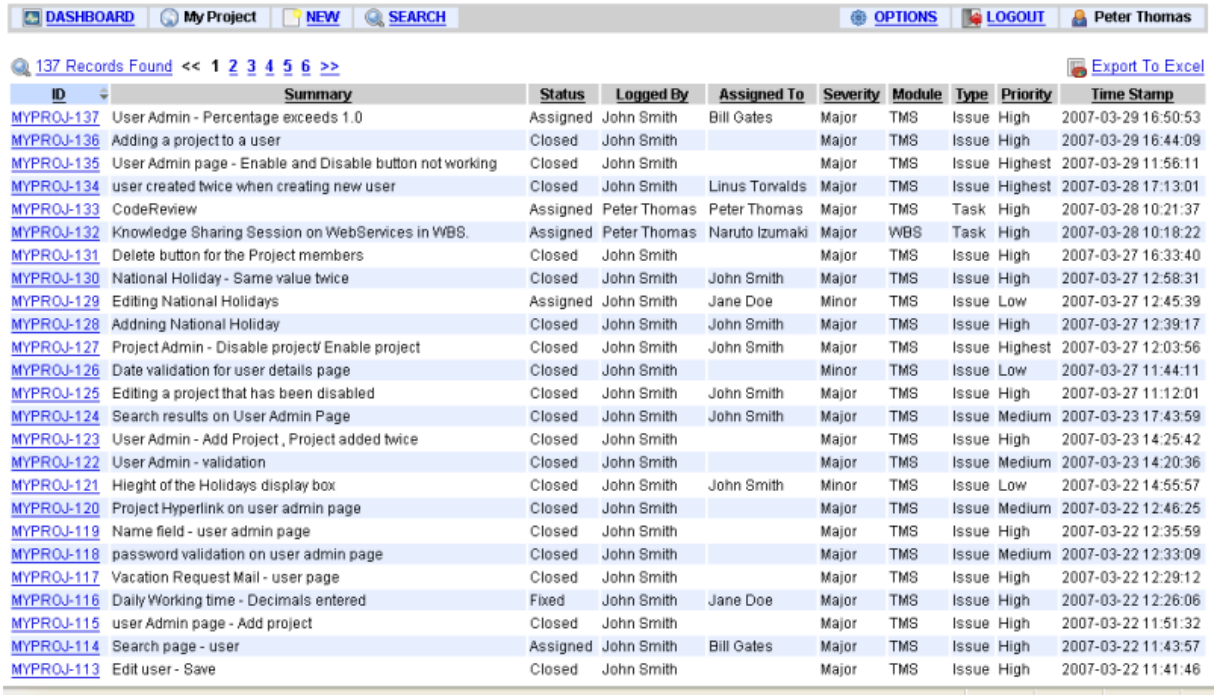
Obrázek 1: Počet požadavků firmy Tender systems s. r. o. [autor]

2.4.1 JTrac

Prvním nástrojem, který firma používala, byl systém JTrac. Tento nástroj byl zaveden v roce 2006 do jejich první komerční aplikace. Toto řešení se používalo do roku 2012 a 6 let tedy posloužilo jako vynikající systém pro technickou podporu. Tento nástroj lze stáhnout na stránkách výrobce ve dvou verzích, buď jako projekt s kompletními zdrojovými kódy, nebo jako už zkompilovaných war soubor. Je vydávaný jako freeware nástroj. JTrac lze využít při zadávání, evidování a generování reportů chyb. První verze nástroje byla vydána 6. 5. 2006 a poslední modifikace byla vydána 31. 8. 2010. Celkem vyšlo 9 verzí, z toho jsou 2 stabilní tj. verze 2.0 a 2.1.0. [13]

U tohoto nástroje si firma kompletně řídila správu nad aplikací. Aplikace byla nahrána na vlastním serveru a byla pro ni zřízena vlastní databáze. Jelikož je aplikace vytvořena v jazyku Java a firma své aplikace vyvíjí v tomto jazyce, mohla si nástroj modifikovat podle svého přesvědčení. Nejdůležitější byla lokalizace do českého jazyka. Jelikož k rozjetí aplikace lze použít jakoukoliv databázi, byla zvolena databáze MySQL. Celá aplikace běžela na webovém kontejneru Apache Tomcat. [12]

S postupem času byl však nástroj nevyhovující. Došlo se k názoru, že aplikace je navržená spíše k internímu použití. Nevýhodou aplikace bylo, že neobsahovala řízení podle SLA schématu, a tudíž firma složitým procesem musela vytvářet reporty pro své zákazníky. Roku 2012 se tedy začalo intenzivně hledat jiné řešení. Ukázka pracovního prostředí vypadá následovně.



ID	Summary	Status	Logged By	Assigned To	Severity	Module	Type	Priority	Time Stamp
MYPROJ-137	User Admin - Percentage exceeds 1.0	Assigned	John Smith	Bill Gates	Major	TMS	Issue	High	2007-03-29 16:50:53
MYPROJ-136	Adding a project to a user	Closed	John Smith		Major	TMS	Issue	High	2007-03-29 16:44:09
MYPROJ-135	User Admin page - Enable and Disable button not working	Closed	John Smith		Major	TMS	Issue	Highest	2007-03-29 11:56:11
MYPROJ-134	user created twice when creating new user	Closed	John Smith	Linus Torvalds	Major	TMS	Issue	Highest	2007-03-28 17:13:01
MYPROJ-133	CodeReview	Assigned	Peter Thomas	Peter Thomas	Major	TMS	Task	High	2007-03-28 10:21:37
MYPROJ-132	Knowledge Sharing Session on WebServices in WBS.	Assigned	Peter Thomas	Naruto Izumaki	Major	WBS	Task	High	2007-03-28 10:18:22
MYPROJ-131	Delete button for the Project members	Closed	John Smith		Major	TMS	Issue	High	2007-03-27 16:33:40
MYPROJ-130	National Holiday - Same value twice	Closed	John Smith	John Smith	Major	TMS	Issue	High	2007-03-27 12:58:31
MYPROJ-129	Editing National Holidays	Assigned	John Smith	Jane Doe	Minor	TMS	Issue	Low	2007-03-27 12:45:39
MYPROJ-128	Addning National Holiday	Closed	John Smith	John Smith	Major	TMS	Issue	High	2007-03-27 12:39:17
MYPROJ-127	Project Admin - Disable project/ Enable project	Closed	John Smith	John Smith	Major	TMS	Issue	Highest	2007-03-27 12:03:56
MYPROJ-126	Date validation for user details page	Closed	John Smith		Minor	TMS	Issue	Low	2007-03-27 11:44:11
MYPROJ-125	Editing a project that has been disabled	Closed	John Smith	John Smith	Major	TMS	Issue	High	2007-03-27 11:12:01
MYPROJ-124	Search results on User Admin Page	Closed	John Smith	John Smith	Major	TMS	Issue	Medium	2007-03-23 17:43:59
MYPROJ-123	User Admin - Add Project , Project added twice	Closed	John Smith		Major	TMS	Issue	High	2007-03-23 14:25:42
MYPROJ-122	User Admin - validation	Closed	John Smith		Major	TMS	Issue	Medium	2007-03-23 14:20:36
MYPROJ-121	Hieght of the Holidays display box	Closed	John Smith	John Smith	Minor	TMS	Issue	Low	2007-03-22 14:55:57
MYPROJ-120	Project Hyperlink on user admin page	Closed	John Smith		Major	TMS	Issue	Medium	2007-03-22 12:46:25
MYPROJ-119	Name field - user admin page	Closed	John Smith		Major	TMS	Issue	High	2007-03-22 12:35:59
MYPROJ-118	password validation on user admin page	Closed	John Smith		Major	TMS	Issue	Medium	2007-03-22 12:33:09
MYPROJ-117	Vacation Request Mail - user page	Closed	John Smith		Major	TMS	Issue	High	2007-03-22 12:29:12
MYPROJ-116	Daily Working time - Decimals entered	Fixed	John Smith	Jane Doe	Major	TMS	Issue	High	2007-03-22 12:26:06
MYPROJ-115	user Admin page - Add project	Closed	John Smith		Major	TMS	Issue	High	2007-03-22 11:51:32
MYPROJ-114	Search page - user	Assigned	John Smith	Bill Gates	Major	TMS	Issue	High	2007-03-22 11:43:57
MYPROJ-113	Edit user - Save	Closed	John Smith		Major	TMS	Issue	High	2007-03-22 11:41:46

Obrázek 2: Pracovní prostředí JTracu [12]

2.4.2 TaskPool

Druhým řešením, které bylo uvedeno do provozu, je TaskPool. Tento systém firma používá od 1. 5. 2012 až do současnosti. Produkt nabízí firma ComArr už 13 let a je nabízen v 5 balíčcích, kterými jsou zákaznický helpdesk, vnitrofiremní helpdesk, firemní úkolování, outsourcing it a helpdesk pro města a obce. Oproti JTracu není TaskPool nabízen jako freeware aplikace. Firma Tender systems s. r. o. má tedy zakoupen balíček zákaznický helpdesk. Na obrázku je vidět pracovní prostředí produktu. [13]

Obrázek 3: Pracovní prostředí TaskPool [autor]

Systém je nabízen v kompletní české lokalizaci. Zadané požadavky se nemažou, takže zůstává jejich kompletní historie. Celkový průběh požadavku je řízen několika komunikačními kanály např. zasíláním emailů. Uživatelské rozhraní je optimalizováno na

většinu současných zařízení. Životní cyklus požadavku je kontrolován podle SLA schématu. Systém nabízí možnost generování až tří druhů reportů. První report obsahuje informace o vytížení řešitelů. Druhý obsahuje informace o požadavcích dle filtračních podmínek např. podle SLA schématu. Třetí report informuje, jaký stav mají aktuálně zadané požadavky. [2]

Požadavek se vkládá pomocí webového formuláře, který si může každý zákazník nechat sestavit, jak potřebuje. Formulář mohou vyplňovat jak přihlášení, tak i nepřihlášeni uživatelé. Zadavatel si může nechat vytvořit formulář na svojí aplikaci, tak jak potřebuje, tj. se svou grafikou a poli, které chce zadávat. Každý produkt obsahuje odkaz na tento formulář. Poskytovatel helpdesku nabízí možnost zadavateli, si nechat vytvořit několik formulářů. Každý formulář je primárně určen pouze pouze pro jeden produkt. Na obrázku je vidět formulář, který je vytvořen pro informační systém Tender arena. [2]

Obrázek 4: Webový formulář Helpdesku [autor]

TaskPool na rozdíl od JTracu neběží na vlastním serveru. Zadavatel nemusí řešit technické problémy, tudíž mu odpadají s tím náklady na údržbu helpdesku. Jelikož je to placený software, tak uživatelské rozhraní vypadá na profesionální úrovni než je to u JTracu. Výhodou je snaha vývojářů zdokonalovat tuto aplikaci o zajímavé poznatky uživatelů, kteří aplikaci používají. Během životního cyklu požadavku je hlídána časová hranice podle SLA

schématu. Systém nabízí větší možnost vytváření reportů. Nevýhodou tohoto systému je, že svátky se musí zadávat každý rok znovu. [2]

Firma se rozhodla pro vytvoření vlastního produktu, protože TaskPool už nesplňuje jejich standardy. Jejich uživatelé jsou zvyklí na funkcionality, které jim TaskPool nemůže poskytnout. Dalším kritériem je mít kompletní správu nad systém. V neposlední řadě je menší finanční náročnost na provozování vlastního systému.

3 Testovací nástroje

Bakalářská práce do současné chvíle popisovala doménu, která se bude vytvářet. Nastal čas, aby se vybral nástroj, pomocí kterého se doména vytvoří. V kapitole jsou uvedeny a použity testovací nástroje, podle kterých lze efektivně napsat test v jazyce Java. Kritéria výběru nástroje jsou na závěr kapitoly shrnuta a je zhodnoceno, který nástroj nejlépe vyhovuje požadavkům. Ten je pak použit při implementaci samotné aplikace.

3.1 Popis testování aplikace

Definice pojmu test není zcela jednoznačná. Každý vývojář, který se s tímto pojmem setká, si pod ním představuje trochu něco jiného. Testování lze vysvětlit jako proces, při kterém dochází ke spouštění aplikace s cílem zjistit, zda splňuje vyspecifikované potřeby uživatele. Z širšího pohledu testování zahrnuje oblast dílčích kroků tj. kontrola požadavků, zdrojového kódu, modelu a podobně. U testování je důležité zdůraznit, že se jedná o proces, neboť je vykonáváno vždy s určitým záměrem. Před každým vývojem dochází nejprve k naplánování a navrhnutí procesu. Po vytvoření aplikace se vyhodnotí proces vývoje. Z dat získaných při testování aplikace se získají informaci o její kvalitě. [4]

Na testování se dá nahlížet i jako na nekonečný proces, neboť o jeho ukončení musí být rozhodnuto na základě několika faktorů. Postupem času se exponencionálně snižuje efektivita a zvyšuje se rozpočet na vývoj. Před začátkem procesu je nutné vytyčit všechny klíčové testy a ty provést. Člověk mnohdy chybuje, a tak i testy musí vývojáři kontrolovat. Testování by mělo být prováděno co nejdříve, nejlépe už při samotné analýze, neboť při ní je odstranění problému nejjednodušší. [4]

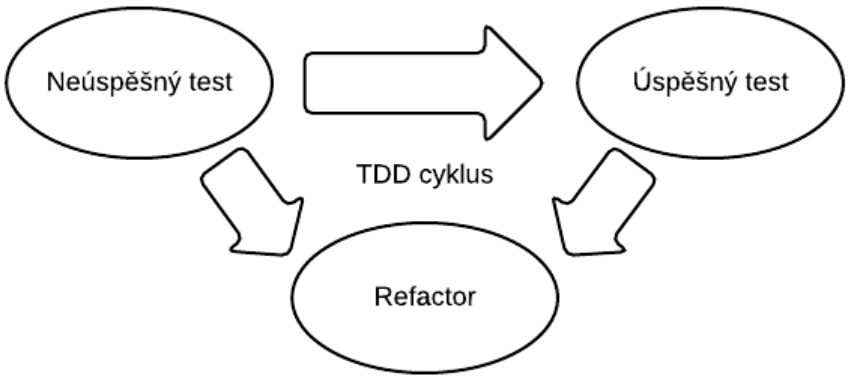
Během testování dochází postupně k hodnocení aplikace, takže se dá už při vývoji říci, jak kvalitně je aplikace navržena. Pokud je testování špatně prováděno, vznikají nedostatečné výstupy. Pokud se nezjistí tento problém na začátku vývoje, nastává nevyhovující proces. Dochází k časovým ztrátám. Z toho vyplývá, že se zvyšuje rozpočet projektu. [4]

Firma může dojít až do kroku, kdy špatný produkt předá zákazníkovi. Výrobce pak utrpí finanční ztrátu způsobenou opravami a pokutami a reputace firmy je velice poškozena. Tyto produkty jsou většinou mediálně sledované, takže každý takový omyl je všude rozebírán a posuzován jako závažný a nedbalý čin. Firma tak přichází o zákazníky: Zaměstnanci firmy jsou s takovou kauzou taktéž spojováni. [4]

3.2 TDD

Pojem TDD (Test-driven development) označuje programování řízené testy ke kontrole jednotlivých částí aplikace. Tento styl programování lze použít napříč všemi programovacími jazyky. Programátor je pouze člověk a není bezchybný. Před spuštěním aplikace v ostrém provozu by se měla alespoň jednou spustit v testovacím prostředí a zkontrolovat, zda funguje tak, jak je popsáno v požadavcích. První kontrolou bývá kompilace, kdy jsou odhaleny některé typy chyb, např. chybějící návaznosti tříd. Při chybě dochází k neočekávánému projevu aplikace. Programátor nepoužívající způsob TDD píše aplikace tímto stylem. Nejprve napíše kód, potom ho zkompiluje a na závěr ho testuje. Objevenou chybu ihned opraví. Tento proces je časově náročný a chyby se mnohdy odhalí pozdě nebo dokonce se neobjeví vůbec. K tomuto problému vznikají automatizované testy, kde si programátor napíše test, ve kterém kontroluje svojí část kódu. Vývojář očekává určitou funkčnost a tato funkce je právě zkontrolována v testu. [5]

Lepší způsob vývoje je použít TDD, a tudíž začít psát testy ještě dříve než samotný kód. Při tomto vývoji existují dvě jednoduchá pravidla. Správně by každý řádek kódu měl být otestován v rámci automatizovaného testu. Duplikace v kódu by se měly maximálně eliminovat. Tento způsob lze použít při psaní jednotkových a integračních testů. Tyto druhy testů jsou detailněji popsány v další kapitole. TDD nabízí 3 stavy, kterých je třeba se držet. Tyto stavy jsou červený, zelený a refactor. Červený znamená, že napsaný test selhává. Pod zeleným stavem si lze představit kód, který testem úspěšně prošel. Při stavu Refactor se zlepšuje kód, aniž by se změnila funkčnost. Tyto tři stavy by se měly používat při psaní každého kódu. Třístavový cyklus lze znázornit následujícím schématem. [7]



Obrázek 5: TDD cyklus [autor]

Při implementaci funkce se nejprve vytvoří test. Napíše se testovací jednotka, ve které jsou metody určující kritéria testu. Při spuštění test spadne na kompilaci, neboť třída, která se testuje, neexistuje. To znamená, že kompilace je v podstatě test. [5]

```
@Test
public void isCorrectResult(){
  Result result = new Result();
  Asser.isNotNull(result);
}
```

Zdrojový kód 1: TDD metoda

Po vytvoření testovací jednotky je třeba napsat třídu Result. Po vytvoření třídy se spustí kompilace a ta projde úspěšně, neboť všechny použité třídy v kódu existuji. Nic nebrání spuštění samotného testu. Test projde úspěšně, neboť konstruktor třídy Result nevrací hodnotu null. [6]

Po úspěšném testu nastává čas pro refactoring, neboli čištění kódu od duplicit, aniž by se měnila funkčnost části kódu. Tyto stavy se dokola mění a třídy se postupně rozšiřují o další kód. [6]

```
public class Result{
  public Result(){
}
```

Zdrojový kód 2: TDD implementace třídy

3.3 Druhy testů

Během tvorby softwaru je prováděno testování v několika úrovních. U jednotlivých testů lze kontrolovat celý systém nebo se může kontrolovat zvolená část funkce. Existuje pět úrovní testování. Každá úroveň nahlíží na testování odlišným způsobem, kdy testuje jiná skupina lidí. Tyto úrovně korespondují s postupem vývoje aplikace. V této práci je úroveň nazývána jako druh. [7]

3.3.1 Testování programátorem

Testování programátorem neboli Assembly tests je prováděno tak, že po napsání zdrojového kódu ho zkontroluje pověřený programátor. Hlavní pravidlem je, že by si ho neměl kontrolovat člověk, který ho vytvořil. Tato kontrola je mnohými vývojáři přehlížena. Mnohdy stačí, když si kód přečte další člověk a ten vidí, zda by šel změnit, nebo zda je správně napsaný. Může nastat situace, kdy by špatný kód byl předán testerovi. On by ho začal testovat a narazil by na chybu. Zdrojový kód by předal zpět autorovi. On by ho opravil a znovu ho předal testerovi. Takový proces je časově a finančně náročný, a proto by měla být prováděna kontrola jiným programátorem.[7]

3.3.2 Testování jednotek

Tento druh testu provádí sám programátor. Testuje jednotlivé části aplikace neboli jednotlivé jednotky. Pod jednotkou si lze představit nejmenší testovatelnou část aplikace. V rámci objektového programování, které je v celé části této práce používáno, to jsou jednotlivé metody, třídy atd. Při psaní jednotkového testu, je třeba mít v paměti, že programátor chce otestovat jednotku nezávisle na jiné jednotce. Prokazuje se správná funkčnost testované části kódu. Jednotkový test nesmí mít ve svém kódu použity další třídy nebo externí funkce. Pokud by programátor použil další funkce, docházelo by ke klamavým výsledkům. Aplikace je mnohdy tak provázána, že tyto zdroje nelze vyřadit. Potom se musí simulovat jednotlivé objekty neboli Mock tak, aby výsledky jednotkového testu byly jednoznačné. [4]

Jak již bylo dříve zmíněno, kód testu se při použití TDD píše jako první. Jednotkové testování je dobrý nástroj pro programátora, protože testy jsou automatizované a opakovatelné. Nutí to programátora se více zamýšlet nad co největším pokrytím možností, které mohou v dané části kódu nastat. Díky tomuto testu se dobře provádí refactoring, neboť se čistí kód od duplicity, ale funkčnost se nemění. Výsledkem testu je přehledný a čistý kód. Nevýhodou je přímá kontrola chování metod, tudíž se komplexnější použití metod nedá otestovat. Na příkladu je ukázáno, jak vypadá jednotkový test. [4]

```
public static int compareTwoNumbers(int a, int b){
  return a >= b ? a : b;
}

@Test
public void compareTwoNumber(){
  int number = 10;
  int result = Number.compareTwoNumber(10,2);
  Asser.assertEquals(number,result);
}
  Asser.assertEquals(number,result);
}
```

Zdrojový kód 3: Jednotkový test

3.3.3 Integrační testování

Integrační testy se píší, když jednotkový test funguje správně a už nedokáže pokrýt funkcionalitu, takže je třeba vytvořit integraci do stabilního celku. Tento druh testu se používá tehdy, když jsou alespoň dva moduly, neboli části kódu, které dohromady poskytují určitou funkcionalitu. Tyto části kódu se považují za samostatný systém. Při integraci dochází k postupnému přidávání jednotkově otestovaných modulů, které musí pracovat

správně a nesmí narušit funkci celku. Při psaní testu je cílem zkontrolovat funkce vznikající spojením jednotlivých částí kódu. Měla by fungovat jak integrace několika částí kódu, tak by i jednotlivé části měly fungovat jako na sobě nezávislé prvky. Příklad takového testu může vypadat následovně. Existuje třída User, ke které se vytvoří třída UserService. V této třídě je metoda, která podle vstupního parametru vrátí uživatele. Během průchodu metodou dochází k volání několika modulů. Prvním je práce s databází, ve které se hledá objekt podle parametru. Druhá část testu je správné vytvoření objektu User podle konstruktoru. Tento příklad se rozdělí na dva jednotkové testy. První test by mohl kontrolovat, zda funguje propojení s databází. Druhý test by kontroloval, zda třída User je správně vytvořena. [4]

3.3.4 Systémové testování

Systémový test se začíná psát, když existuje několik funkčních integračních testů. Během těchto testů se kontroluje aplikace už jako funkční celek. Tyto testy lze psát za podmínky, že už je implementován větší celek. Aplikace je testovaná z pohledu zákazníka, takže se testuje správný průchod aplikací. Tester pomocí scénářů simuluje kroky, které mohou v běžném provozu nastat. Chyby, které jsou nalezeny, jsou opraveny a znovu otestovány. Systémové testování lze rozdělit ještě do několika pod úrovní. [7]

Mezi tyto pod úrovně patří následující testy. Funkční testy zjišťují, zda aplikace splňuje požadavky specifikované pro funkci systému. Mezi funkční testy patří bezpečnostní testy. U těchto testů se kontroluje, zda je aplikace dostatečně zabezpečená. Robustní testy kontrolují, jak se aplikace dokáže vypořádat s nečekanými stavy aplikace, tj. chybami. Testy komfortnosti jsou vytvářeny z důvodu, aby aplikace byla jednoduchá a srozumitelná pro uživatele. Testy interoperability poslouží při testu kompatibility s ostatními systémy. Test spolehlivosti se používá tehdy, když je třeba zkontrolovat stabilitu systému. V neposlední řadě existují výkonové testy, které sledují aplikaci v provozu podle předem stanovených ukazatelů. [7]

3.3.5 Akceptační testování

Akceptační testy jsou prováděny vždy na straně zákazníka. V ideálním případě k těmto testům dochází tehdy, pokud předešlé druhy testů proběhly správně. V tu chvíli se aplikace předá zákazníkovi. Zákazník má složenou skupinu pověřených lidí, která provede kontrolu podle připravených scénářů. Tyto scénáře neboli akceptační kritéria jsou předem stanoveny vývojovým týmem a zákazníkem. Nesrovnalosti mohou vést až k vrácení aplikace k předělání, ale i k celému odmítnutí a odstoupení od smlouvy. Chyby nalezené zákazníkem jsou reportovány zpět vývojářům. [7]

Uživatelé nekontrolují během těchto testů pouze samotnou implementaci aplikace, ale i její součásti. Například to může být technická dokumentace, uživatelská dokumentace atd. Toto testování je prováděno jako jediné na straně zákazníka. Uživatele nehledají defekty, ale kontrolují průchod aplikací během běžného provozu. [7]

3.4 Vybrané nástroje

Nástroj pro tvorbu testů bude vybrán podle několika parametrů. V nástrojích, které jsou zde uvedeny, se používá programovací jazyk Java. U každého je uveden popis a jednoduchý příklad, jak ho používat. Na závěr je každý zhodnocen, zda se může při vývoji aplikace použít. Po shrnutí všech nástrojů je vybrán takový, který je nejlepší pro vytvoření aplikace helpdesk.

Nejvhodnější nástroj je vybrán podle několika kvalifikačních kritérií. Vývoj celé aplikace bude probíhat způsobem TDD, takže je třeba zjistit, zda se dá v nástrojích takto programovat. V dalším bodu hodnocení se ukáže, co vše je třeba zavést do projektu, aby se mohly jednotlivé nástroje použít. Pro správu projektu je zvolen Maven. Každý nástroj potřebuje napsat určitou část kódu, než se mohou začít psát jednotlivé testovací metody. Při vývoji aplikace bude použita platforma Java EE, takže se ukáže, zda testovací nástroje dokáží pracovat s touto platformou. Protože aplikace poběží na serveru, je potřeba zjistit, zda nástroje dokáží pracovat s aplikačním kontejnerem. Pro zajímavost bude uvedeno, jak rychle je vykonán test v jednotlivých nástrojích.

Nejefektivnější nástroj se ukáže z výsledků, kdy budou všechny podrobeny zkoušce stejné funkce. Aplikace, která bude v této práci vytvořena, bude postavena na architektuře 3 vrstev. Testy budou zaměřený na business vrstvu a datovou vrstvu. V business vrstvě jsou EJB (Enterprise java beans) třídy, které v podstatě vytváří stavební kámen aplikace. V datové vrstvě jsou pouze třídy, které korespondují s tabulkami v databázi. Z druhů testů se použije integrační test, protože převážná většina EJB tříd pracuje s databází. [8]

Pro kontrolu funkce budou napsány základní třídy, které se v každém nástroji otestují. Napíše se jedna entita tj. třída, která koresponduje s tabulkou v databázi. Další třída, která je potřeba, je EJB třída, tzv. Service. Scénář testů bude jednoduchý. Existuje skupina uživatelů, kterou je třeba uložit a následně načíst. Při načítání se vyhledává podle přihlašovacího jména. Pokud uživatel neexistuje, měla by se vyvolat výjimka. Při ukládání nového uživatele by nemělo dojít k situaci, kdy více uživatelů bude mít stejné přihlašovací jméno. Přihlašovací jméno je povinný údaj, takže musí být vždycky zadáno.

```
@Entity
public classs User{
  @Id
  private String nick;
další atributy, konstruktor, get/set metody
}

@Stateless
@LocalBean
public class UserService{
  @PersistenceContext
  EntityManager em;
  public User findUser(String nick) throws ValidationException{
    User user = em.find(User.class, nick);
    if(user == null){
      throw new ValidationException(“User not exists”);
    }
    return user;
  }
  Další metody
}
```

Zdrojový kód 4: Vzorové třídy

3.4.1 JUnit

JUnit je knihovna používaná k implementaci jednotkových a integračních testů. Tato knihovna obsahuje velkou škálu funkcí pro porovnání a kontrolu různých hodnot. Navíc obsahuje funkci pro kontrolu očekávaných výjimek a můžeme i otestovat časový limit, ve kterém musí být testovací metoda vykonána. Zavedení do projektu je velmi jednoduché, stačí přidat závislost na knihovnu. U závislosti je třeba specifikovat okamžik, kdy je knihovna používána, tj. pouze při testech, k tomu slouží tzv. scope a hodnota test. [14]

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Zdrojový kód 5: JUnit závislost

Každý test je reprezentován testovací jednotkou, kde název je odvozen z názvu testované třídy a je doplněn příponou Test. Třída obsahuje jednotlivé metody, jejichž jednotlivé metody jsou určeny anotacemi z JUnit knihovny. Nejzákladnější anotace je @Test. Ten označuje metody, které jsou při testu spuštěny. Testovací jednotka může

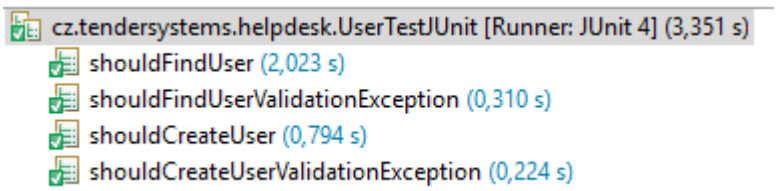
obsahovat následující anotace @Before, @After, které se provedou vždy před nebo po testovací metodě. Anotace @BeforeClass a @AfterClass se provedou před začátkem či po dokončení celé testovací jednotky. Pro testování výsledku existuje třída Assert, která obsahuje metody jako assertTrue() a assertFalse() pro kontrolu logických hodnot. Dále z nejvíce používaných metod to jsou assertNull(), assertNotNull(), assertEquals(). [14]

Po stručném popisu nástroje nastal čas pro praktické vyzkoušení JUnit. Vytvoří se testovací jednotka, která bude mít název UserTestJUnit a pokusí se otestovat třídu UserService. Nejprve je třeba vytvořit metody, nad kterými se použijí anotace @Before a @After, které vytvoří a ukončí transakci. V dalším kroku se načte rozhraní EntityMangaer, které se stará o životní cyklus entit. Po vytvoření těchto dvou metod se mohou psát metody testu. Ukázka těla testovací metody. [15]

```
public void shouldFindUser(){
  User user = userService.findUser("admin");
  Assert.assertNotNull(user);
  Assert.assertEquals(user.getNick(), "admin");
  Assert.assertNotNull(user.getPassword());
}
```

Zdrojový kód 6: JUnit test

Nástrojem JUnit lze programovat způsobem TDD. Zavedení do projektu je jednoduché. Podmínkou je vytvoření testovacích dat v databázi. Nástroj nenabízí jednoduché řešení, jak data pohodlně vložit do databáze, a proto každá testovací jednotka musí mít vytvořeny metody pro vytvoření spojení z databází. Toto není právě automatizované. EJB třídy lze částečné testovat, ale nejdou otestovat jejich životní cykly. Nástroj neumí žádným způsobem pracovat se serverem, takže nejde testovat funkčnost aplikace vůči aplikačnímu kontejneru. Rychlost testu je znázorněna na následujícím obrázku.



Obrázek 6: Výsledek testu v JUnit [autor]

3.4.2 Mockito

Mockito je testovací framework pro vytváření tzv. mocku, neboli náhradních objektů. Tyto objekty jsou používány při psaní integračních testů. Přednost mocku je taková, že při jeho chování se zdá, že se jedná o běžný objekt. To může být ale velký klam, neboť

pomocí mocku se upravuje chování objektu. Tímto způsobem lze nasimulovat daleko větší škálu testovacích případů. Požadavky na napsání zdrojového kódu testu nejsou nikterak veliké. Tento nástroj garantuje plnohodnotnou podporu pro nejnovější verzi Javy 8. K zavedení do projektu je třeba přidat jednu knihovnu. [16]

```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-all</artifactId>
  <version>41.9.5</version>
  <scope>test</scope>
</dependency>
```

Zdrojový kód 7: Mockito závislost

Při psaní testu by se mělo postupovat podle několika pravidel. Nejprve se vytvoří mocky pro všechny objekty, které jsou v testovací třídě potřeba. U mocku je třeba co nejjednodušeji vytvořit takové chování, aby test obsáhl celou funkcionalitu dané jednotky. Když jsou mocky připraveny, tak se zavedou do testovací jednotky. Programátor spustí test a vyhodnotí se, zda je test postačující. [16]

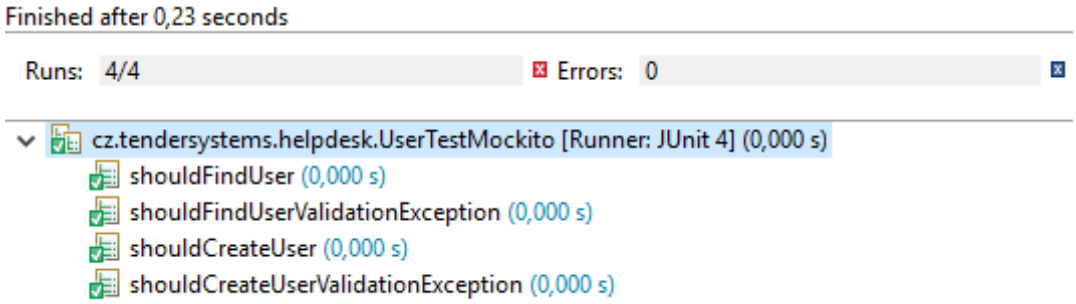
Při tvorbě testovací jednotky je důležité použít anotaci nad definicí třídy @RunWith(MockitoJunitRunner.clas). Takto deklarovaná třída bude pomocí Mockito spouštěče spuštěna. K tvorbě mocku existuje anotace @Mock, která se použije při tvorbě objektu určitého typu. Důležité je říci, že ve výchozím stavu je taková třída prázdná. Pokud její metody definují návratovou hodnotu, tak tato hodnota je null. Obvykle se mocky definují pro každou testovací metodu nové, protože vývojář může chtít při každé metodě kontrolovat jiné chování. Po vytvoření objektu je třeba definovat jeho chování. Programátor píše pravidla pomocí metod any(), isNotNull(), same, které nemají návratovou hodnotu. Pokud je třeba návratových hodnot, použije se např. metoda thenReturn(). [16]

Stejným případem jako v podání JUnit se vytvoří testovací jednotka s názvem UserTestMockito pro otestování třídy UserService. Nad definicí třídy se použije anotace @RunWith(MockitoJunitRunner.clas). K psaní testovacích metod je třeba vytvořit dva mocky, pro třídu UserService a rozhraní EntityManager. Po vykonání všech těchto kroků, přichází na řadu tvorba testovacích metod. Při psaní testovacích metod se používá syntaxe z knihovny JUnit, V každé metodě je třeba nadefinovat chování mocku. Na ukázce je zobrazena metoda z této třídy.

```
@Mock
private EntityManager mockEm@Mock
private UserService userService;

public void shouldFindUser(){
  userService = new UserService;
  userService.setEm(mockEm);
  Vytvoření objektu User
  Mockito.when(mockEm.find(Users.class, "test")).thenReturn(user);
Asser.assertNotNull(userService.find("test"));
}
```

Zdrojový kód 8: Mockito test

Z praktického vyzkoušení nástroje lze usoudit, že jde programovat způsobem TDD. Při zavedení do projektu se vloží pouze jedna závislost na knihovnu. Můžeme testovat celou škálu funkcí EJB tříd. Mockito neumí žádným způsobem během testu pracovat se serverem. Jelikož se nebere v testu reálné chování objektů, musíme vytvořit nové. Po spuštění testovací jednotky byl udělán screen, kde je ukázána rychlost testu. Z obrázku vyplývá, že tyto testy jsou velmi rychlé.



Obrázek 7: Výsledek testu v Mockito [autor]

3.4.3 Arquillian

Arquillian je platforma, pomocí které se vytvářejí především integrační testy v jazyce Java. Nástroj se zaobírá veškerým řízením EJB tříd, nasazováním a inicializováním těchto objektů. Při testu se pracuje s reálnými objekty. Arquillian se snaží minimalizovat psaní zdrojového kódu okolo testů a řídí životní cyklus zásobníku, ve kterém je vykonáván test. Knihovna umí pracovat s injektovanými třídami, tj. EJB a CDI třídy. K práci s tímto nástrojem je třeba si do projektu přidat následující knihovny arquillian-bom, Arquillian-persistence-dbunit, arquillian-persistence-integration-tests, junit, arquillian-container-remote, arquillian-junit-conteiner. [17]

Při tvorbě testovací jednotky je třeba použít anotaci @RunWith a definovat, že chceme spouštěcí řadič Arquillian. První co Arquillian po spuštění testu vyhledá, je statická

metoda, u které je anotace @Deployment. Tato metoda vytváří tzv. mikro deployment. V deploymentu se nachází všechny třídy a potřebné soubory, které si programátor zvolí, aby testovací jednotka neselhala při spuštění na kompilaci. Vytvořený balík se spustí pomocí aplikačního kontejneru. K zabalení slouží rozhraní ShrinkWrap, který umí vytvářet jar i war a mnoho dalších formátů. Příklad takové metody může být následující. [20]

```
@Deployment
public static JavaArchive createTestArchive() {
  return ShrinkWrap.create(JavaArchive.class).addPackages(true,
 "cz.tendersystems");
}
```

Zdrojový kód 9: Metoda pro vytvoření balíku

Po vytvoření deploymentu, přichází na řadu jednotlivé testovací metody, které se píší stejným způsobem jako při psaní testu v nástroji JUnit. Jediný rozdíl mezi Arquillian a JUnit je, že obvykle tyto testy běží za pomoci serveru. Tvůrci Arquillianu garantují funkčnost s těmito webovými kontejnery Weld Embedded, GlassFish, JBoss, Apache TomEE. [20]

Testům, které jsou vytvářeny pro EJB třídy, se říká testy persistence. K takovému testu je nutné přiložit konfigurační soubor persistentní jednotky. Jednotka je popsána v souboru test-persistence.xml. Definuje takové chování, aby při začátku testovací jednotky se vytvořily v databázi nové tabulky a na konci se databáze vyčistila. Stejně jako ostatní nástroje, tak i tento musí vždy nejprve vytvořit testovací data. Pro vytvoření dat existují dva způsoby. Programátor se může vydat cestou metod, které mají anotace @Before a @After, kdy u první metody se vytvoří testovací data a u druhé se smažou. Arquillian vyniká unikátní vlastností. Snaží se krok tvorby testovacích dat vývojáři co nejvíce usnadnit a nabízí anotace @UsingDataSeta @ShouldMatchDataSet. Při použití těchto anotací mají testovací metody defaultně nastaveno, že se databáze při každém zavolání smaže. Nástrojem se může definovat jiné chování práce s databází. Ke změně se použije anotace @Cleanup, nebo @CleanupUsingScrip. U každé anotace je potřeba definovat, kdy se má databáze čistit. @UsingDataSet se použije u metody, kde před provedením testovací metody je třeba vložit data do databáze a ta se v ní poté vyhledávají. Anotace @ShouldMatchDataSet se použije, pokud se testuje metoda, která vkládá nový objekt do databáze, a tento objekt je porovnán s daty v konfiguračním souboru. Anotace dokáží zpracovat soubory ve formátu yml, xml, yaml. V praxi se nejvíce používá formát xml. Takový xml soubor vypadá následovně. [21]

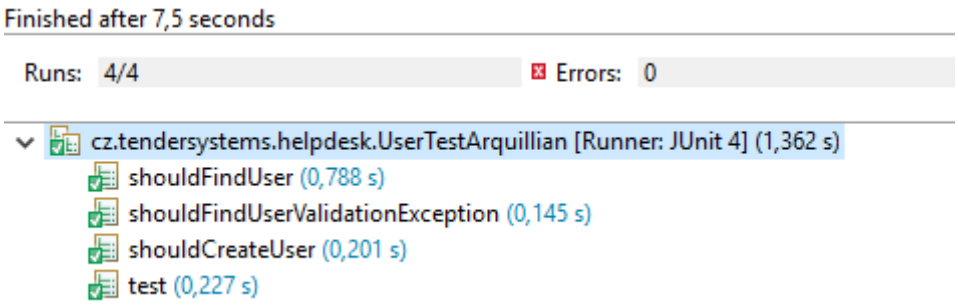
```
<dataset>
  <users nick="test" password="test" firstName="test" lastName="test"/>
</dataset>
```

Zdrojový kód 10: Xml datasets

Tento nástroj je nastaven tak, aby nepoužíval transakce, což je správně. EJB třídy využívají transakce, takže cokoliv, co jde ven z EJB třídy, by nemělo vědět o ní. Pokud testovací metoda potřebuje transakci, tak se musí definovat. [17]

Konkrétní těla testovacích metod vypadají stejně jako v podání nástroje JUnit. U tohoto nástroje stojí za zmínku, že v něm lze jednoduchým způsobem vytvořit funkční test pro webové rozhraní. Tuto funkcionalitu rozšiřuje knihovna Arquillian Drone, která přináší do nástroje velice užitečnou funkci. Arquillian Drone jednoduše komunikuje s webovým prohlížečem. Touto knihovou lze testovat vyplňování formulářových dat, navigaci stránek, nebo ověřování bezpečnosti aplikace. Arquillian Drone umí při spuštění testu pracovat současně ve více prohlížečích a nabízí kompatibilitu s Ajaxem a Javascriptem. [18]

O nástroji Arquillian, lze říci, že pomocí něho lze vyvíjet aplikaci způsobem TDD. Nevýhodou oproti předchozím řešením je náročnější nasazení do projektu. Do Mavenu se musí vložit několik závislostí na knihovny. Arquillian musí oproti ostatním konfigurovat několik xml souborů, protože se chování testovací jednotky jeví jako výsledná aplikace na webovém kontejneru. Nástrojem lze testovat veškerou funkcionalitu EJB tříd. Arquillian se snaží co nejvíce zjednodušovat vývojářům režii testů a nabízí několik užitečných anotací. Výhodou je, že se při testu kontroluje reálné chování objektů. Po vytvoření testovací jednotky byl udělán screen, který ukazuje rychlost testu.



Obrázek 8: Výsledek testu v Arquillian [autor]

3.4.4 Needle

Needle je framework pro testování Java EE komponent, které jsou nasazeny ve webovém kontejneru. Nástroj snižuje psaní testovacího kódu pro automatické injekce objektů. Injekce mock objektů je nastavena ve výchozím stavu, kdy si tyto objekty sám vytváří. Nástroj umí pracovat s injekcí EJB a CDI tříd. Při persistent testu dokáže pracovat

s životní funkcí databáze, tj. dokáže vytvářet a mazat tabulky v databázi. Nástroj zvládá práci s transakcemi. Needle umí zpracovávat mocky vytvořené Mockitem. Při tvorbě testů se používají anotace s knihovny JUnit. Při zavedení nástroje do projektu je třeba přidat závislosti na tyto knihovny easymock, jbosscc-needle. [19]

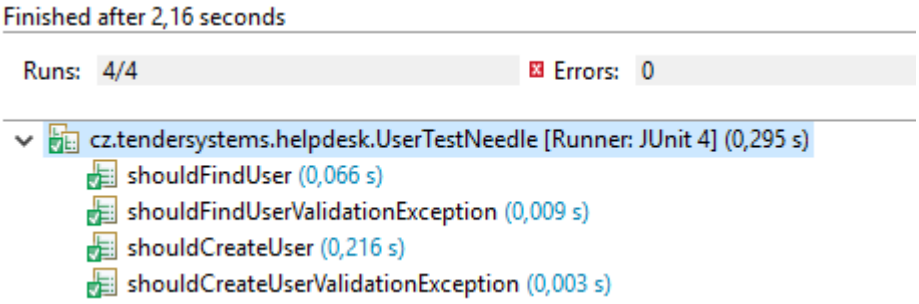
Na začátku testovací jednotky se musí nadefinovat testovací pravidla. Při testu persistence je třeba vytvořit pravidlo pro vytvoření přístupu k databázi. Toto pravidlo se vytvoří s anotací @Rule, která je z knihovny JUnit. Na objekty, které je třeba otestovat, je použita anotace @ObjectUnderTest. Po spuštění testu se takovému objektu vytvoří nová instance a zavede se injekce na definovaná pravidla. Zápis pravidla a testovacího objektu je proveden následujícím způsobem. [19]

```
private static DatabaseRule databaseRule = new DatabaseRule("helpdesk-needle");
private final EntityManager em = databaseRule.getEntityManager();

@Rule
private NeedleRule needleRule = new NeedleRule(databaseRule);

@ObjectUnderTest
UsersService userService;
```

Zdrojový kód 11: Needle pravidlo

U Needle je proveden praktický test, který má prokázat použitelnost při vytvoření testovací jednotky pro předem definovanou třídu. Zdrojový kód testovací jednotky se liší pouze částí kódu, která je uvedena v předchozí ukázce. Testovací metody jsou zcela totožné jako u nástroje JUnit.

Tímto frameworkem se může pohodlně vyvíjet způsobem TDD. Do projektu se musí zavést více závislostí. Není třeba žádná další konfigurace xml souborů ani samotného nástroje. Needle směřuje svůj vývoj k co největšímu zjednodušení zápisu přebytečného kódu. Dokáže zpracovávat veškeré injekce EJB a CDI tříd. Nástroj však neumí pracovat se serverem, takže nelze testovat funkcionality s ním spojené. Myšlenka nástroje je snižovat rychlost testů na minimum. Výsledek Testovací jednotky je následující.



Obrázek 9: Výsledek testu v Needle [autor]

3.4.5 Vyhodnocení nástrojů

Po praktickém vyzkoušení všech nástrojů přichází výsledné zhodnocení. Při hodnocení jsou uvedeny důvody, proč se použijí konkrétní nástroje pro implementaci helpdesku. První kritérium, které bylo řečeno, že má nástroj splňovat, je možnost vývoje způsobem TDD. Toto kritérium splňují všechny nástroje. Jediný rozdíl je v tom, že Arquillian nabízí možnost vývoje pouze integračních testů, zatímco u ostatních lze psát jak jednotkové, tak i integrační testy. Při zavádění do projektu byly nejlépe hodnoceny JUnit a Mockito, kterým stačí přidat pouze jednu knihovnu. O něco hůře dopadl Needle, který potřebuje přidat 2 knihovny. Nejhůře dopadl Arquillian, kde je třeba přidat 5 knihoven. Kvůli komplexnosti Arquillianu se musí dále konfigurovat jednotlivé funkce např. konfigurace aplikačního serveru. Při psaní testů ve všech nástrojích je určitá režie, kterou je třeba nejprve vytvořit, aby se vůbec testovací jednotka správně spustila a byla úspěšně provedena. U tohoto kritéria jsou všechny nástroje na stejné úrovni, neboť každý potřebuje implementovat určitou funkcionalitu. Platformu Java EE nejlépe otestuje Arquillian a částečně ji zvládne zpracovat Needle. JUnit neumí pracovat s životními cykly EJB tříd a neumí používat injekce těchto tříd. Mockito je postaveno na testu simulujících objektů, takže se netestuje reálné chování EJB tříd. S webovým kontejnerem umí pracovat pouze Arquillian. Nejrychlejší provedení testovací jednotky bylo vykonáno v nástroji Mockito a nejpomalejší v Arquillian.

Z výsledků vychází nejlépe Arquillian, který testuje výhradně reálné chování objektů, výborně pracuje s platformou Java EE a veškeré funkce dokáže spolehlivě otestovat při použití webového kontejneru. Nástroj nabízí nejlepší řešení tvorby testovacích dat do databáze, neboť se vytváří soubory v různých formátech, s daty, která jsou přidána, jako nové řádky do databáze. Soubory s testovacími daty se mohou používat nezávisle na sobě, a to v libovolném počtu testovacích jednotek.

4 Analýza

Vývoj aplikací je v současné době posunut do takové šíře, že každý den vzniká nová webová, desktopová či mobilní aplikace. Dnes také existuje vysoký tlak zákazníků na rychlost vývoje aplikací, který dříve nebyl. Problém je v tom, že na úkor rychlosti nasazení do provozu, vzniká hluché místo v podobě nezvládnuté analýzy. Před samotným vývojem je nutné stanovit jasné požadavky, co se bude vytvářet. Je třeba jít cestou maximální kvality a toho dosáhneme formou dokonalé analýzy. Ze samotné definice slova analýza vyplývá, že je třeba rozložit komplexní oblast na jednotlivé dílčí celky, podle kterých bude probíhat samotná implementace. Pokud není daný problém rozebrán do detailu, tak odhad nejde určit, protože je to jenom časový odhad, kdy bude aplikace hotová. [22]

Na internetu existují články o tom, že při agilním programování, není třeba tvořit analýzu. Tento způsob se dnes velice používá. Samotná architektura aplikace se sama ukáže za pochodu. Jednoduchým příkladem je návrh vzhledu stránky. Vzhled stránky je mnohdy kámen úrazu celé aplikace, neboť zákazník a analytik mají mnohdy rozdílné představy. Příkladem rozporu může být jednoduchý webový formulář, ve kterém obě strany mohou mít rozdílný pohled na rozložení polí a dalších grafických prvků. Je vhodnější navrhnout vzhled aplikace v analýze, než přijít na rozpor až při implementaci. Analýza poslouží jako výborný seznamovací nástroj s výrobkem v jakékoli fázi vývoje. [23]

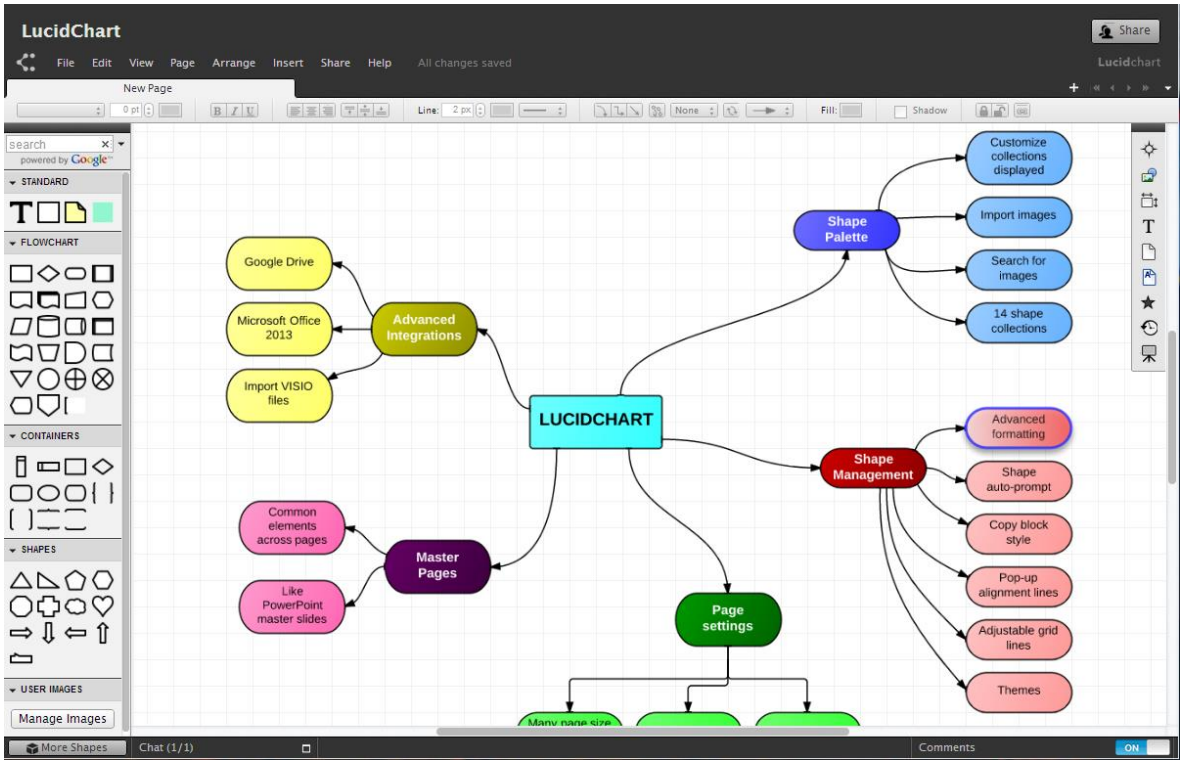
Před samotným návrhem systému helpdesk je nutné si nejdříve něco říci o grafickém jazyku UML (Unified Modeling Language), podle kterého jsou vytvořeny jednotlivé diagramy. V první části analýzy budou rozebrány klíčové požadavky na systém a poté budou nakresleny jednotlivé diagramy. Pro dostačující návrh budou vytvořeny tyto diagramy: Use Case diagram, Class diagram, Aktivity diagram, Wireframe. Na závěr analýzy je podrobně rozebráno navrhované zabezpečení celé aplikace. Název aplikace po dohodě s firmou Tender systems s. r. o. je zvolen Tender desk.

4.1 Nástroj pro tvorbu diagramů

Diagramy jsou kresleny v nástroji Lucidchart. Tento nástroj je výhradně používán jako online software. K používání tedy poslouží jakýkoliv webový prohlížeč, čímž nabízí velkou výhodu, protože uživatel neprovádí žádnou instalaci do svého počítače. Program lze používat také na tabletech a mobilních zařízeních napříč všemi operačními systémy. Nástroj má širokou škálu využití, např. kreslení myšlenkových map a diagramů. Je také vhodný pro tvorbu Wirefaremu, neboli návrhu webových stránek. Všechny dokumenty lze exportovat

do mnoha formátů např. PDF, PNG. Weboví fanoušci zde najdou funkci pro publikování diagramů do sociálních sítí pomocí HTML kódu. [24]

Lucidchart nabízí firma Lucid Software Inc. ve čtyřech různých verzích – free, basic, pro, team. Každá verze nabízí rozdílné funkce. Pro tvorbu diagramů je používán účet poskytnutý od firmy Tender systems s. r. o., která má zakoupenou verzi team. Na následujícím obrázku je ukázán hlavní pracovní sešit, ve kterém se tvoří diagramy. [25]



Obrázek 10: Nástroj LucidChart [26]

Diagramy lze kreslit podle standardu UML. Tento obecný jazyk je tvořený notacemi. Jazyk je určen pro návrh a dokumentaci informačních systémů. Je vynikající komunikační nástroj, podle kterého analytik s klientem dávají dohromady, jak aplikace bude vypadat. Jednotlivé diagramy jsou blíže popsány při tvorbě návrhu aplikace. [27]

Tento nástroj byl použit z důvodu, že v Lucidchartu lze vytvořit kompletní analýzu a firma Tender systems s. r. o. k němu poskytla přístup, protože chtěla mít dohled nad návrhem.

4.2 Požadavky

Požadavky na informační systém jsou nedílnou součástí analýzy. Přesné a srozumitelné vysvětlení definice požadavku zní takto: Pomocí požadavků se vyklíčují a eliminují problémy, které by se mohly objevit při tvorbě projektu, zcela v zárodku vývoje. V pozdějších fázích vývoje se požadavky využívají pro následnou kontrolu, zda vše, co bylo navrženo, bylo implementováno. Požadavky se dělí na funkční a nefunkční. [28]

4.2.1 Funkční požadavky

Funkční požadavky definují soubor funkcionalit a bezpečnostních pravidel, které musí budoucí systém splňovat. Tyto požadavky spolu vytváří analytik se zákazníkem. Seznam funkčních požadavků na systém helpdesk je níže uveden. [28]

- V systému budou uživatelé rozděleni do těchto rolí – administrator, řešitel, pozorovatel, zadavatel.
- Systém umožní vkládat požadavek k vybranému produktu. Ten bude moci vkládat přihlášený i nepřihlášený uživatel. Uživatel bude vybírat z produktů, ke kterým bude mít přístup. K požadavku se budou přikládat soubory. Každý požadavek bude mít přiřazeno SLA schéma, které vybere uživatel. Systém vypočítá čas na reakci a čas na vyřešení pomocí SLA schématu a pošle email s potvrzením o přidání požadavku.
- Nepřihlášený uživatel si zobrazí požadavek pouze z odkazu v emailu. Přihlášený uživatel zadavatele bude vidět seznam svých požadavků a z něj se dostane na detail požadavku.
- Uživatelé provozovatele uvidí požadavky, ke kterým budou mít přístup. Každý uživatel bude mít přiřazen produkt, který bude řešit. Administrátor systému vidí všechny požadavky.
- Ke každému požadavku se bude vkládat komentář. Ale to za podmínky, že k požadavku bude mít uživatel přístup. Ke každému komentáři se budou přikládat soubory.
- Přihlášení uživatelé budou vyhledávat požadavky pomocí filtračních podmínek. Generovaná data půjdou exportovat do formátu PDF a XLS.
- Řešitelé budou řešit požadavky ve stanoveném čase, který bude vypočítaný z SLA Schématu?. Hodnota času reakce a času na vyřešení bude maximálně 24 hodin.

- U požadavku bude řešitel měnit SLA schéma, když ho autor požadavku nevhodně zvolí. Časy pro reakci a pro vyřešení se při takové změně znovu přepočítají.
- Pozorovatelé si budou pouze zobrazovat požadavky.
- Administrátor bude mít správu nad uživateli, zadavateli, produkty a SLA schématy, požadavky.
- Systém umožní administrátorovi generovat reporty požadavků podle zadaných parametrů. Tyto reporty se budou exportovat do PDF a XLS.
- Systém bude archivovat ukončené požadavky po dobu 1 roku.
- Systém umožní uživateli si resetovat heslo.

4.2.2 Nefunkční požadavky

Nefunkční požadavky definují věci, které jsou technického nebo časového rázu. Jsou to například technologie, které se využívají pří vývoji systému. Seznam nefunkčních požadavků na systém je následující: [28]

- třívrstvá architektura, za použití standardů jazyka Java EE,
- názvy tříd, metod atd. budou psány v českém jazyce kvůli správě systému firmou Tender systems s. r. o.,
- datová vrstva na platformě PostgreSQL,
- optimalizace pro různé prohlížeče, tj. systém by měl běžet na nejrozšířenějších prohlížečích současnosti,
- autorizace přes uživatelské účty,
- kontrola pomocí CAPTCHA.

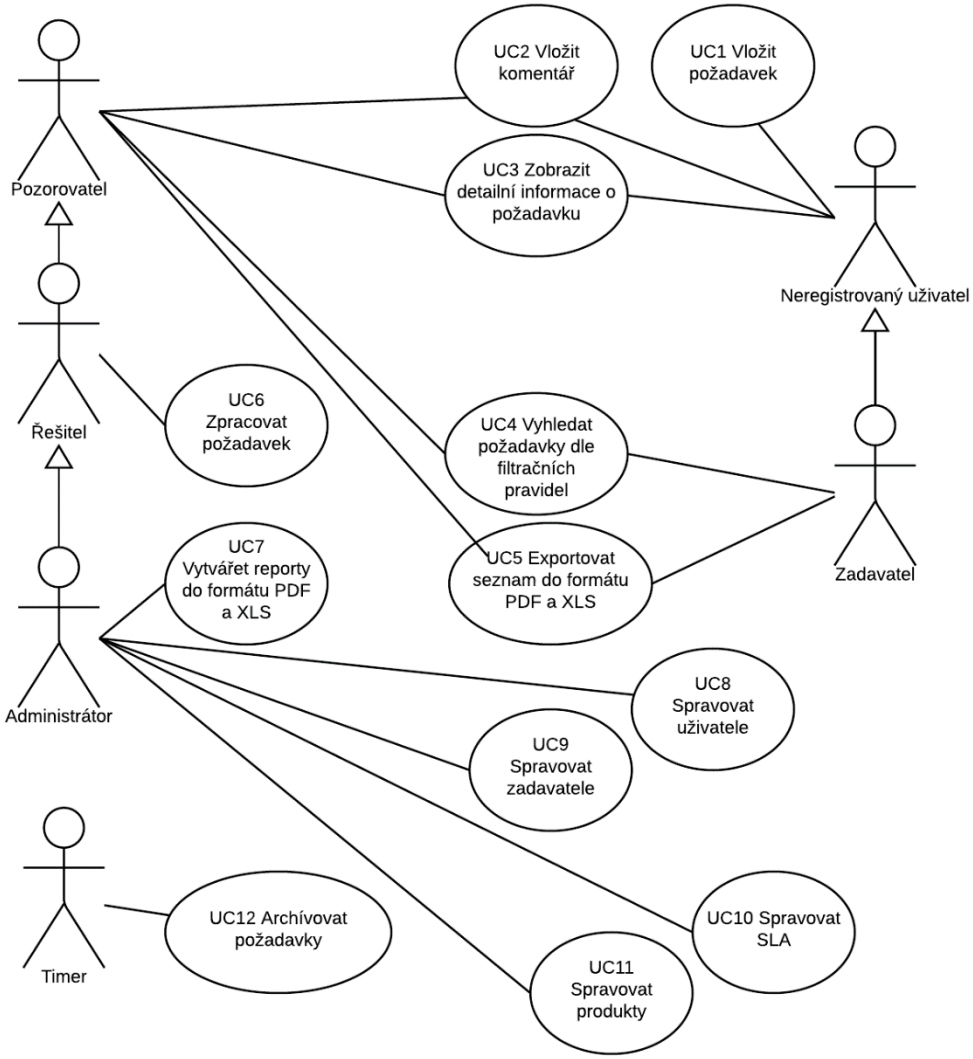
4.3 Diagram případů užití a scénáře

Diagram případů užití je Use Case Diagram, který znázorňuje, jak se systém chová z pohledu uživatele. Diagram ukazuje, co si zákazník představuje pod funkcí systému. Při analýze je třeba nejprve vědět, co se bude vytvářet. Tento diagram bývá kreslen jako první a obsahuje 2 základní prvky. [29]

Prvním prvkem je aktér neboli role v systému. Každý aktér pracuje s jednotlivým případem užití. Pod aktérem si lze představit fyzickou osobu, ale může to být čas. Mezi jednotlivými aktéry může být zavedena generalizace. To znamená, že některý aktér může dědit funkcionalitu od jiného aktéra. [29]

Akci jako takové se říká případ užití neboli UC. Každý případ užití znázorňuje nějaký cíl. Jeden případ užití se rovná jedné funkcionalitě systému. Každý případ užití znázorňuje

co má systém umět. Podle UML specifikace se případy užití kreslí elipsou. Po vytvoření aktérů a případů užití je třeba tyto prvky spojit vazbou. Na diagramu je vidět jednoduchá vazba neboli asociace. Práce probíhá vždy od pokynu aktéra k případu užití. [29]



Obrázek 11: Use Case Diagram [autor]

Z obrázku lze vyčíst, jaké funkce bude systém vykonávat, avšak ne bližší specifikaci problému. K tomu vzniká dokument, ve kterém jsou popsány případy užití. Každý případ užití má v dokumentu svojí specifikaci definovanou několika částmi. Nejdříve by měl být uveden krátký popis případu užití podle toho, jakou funkci má pro uživatele. Dále jsou uvedeni aktéři, kteří zasahují do případu užití. Poslední část je scénář. Jedná se o sérii několika bodů, v nichž se střídá aktér a systém. Scénář je maximálně zaměřen na funkčnost případu užití. Nevysvětluje grafické rozložení prvků v systému. Bývá rozdělen na dva toky. První je základní scénář. Při něm nastává průchod systémem bez chyb. Druhý scénář přichází na řadu, když se v základním toku očekává nějaká alternativa či chyba. Tento tok je spojen

s bodem základního průběhu. Zpravidla je alternativní tok odkázán na bod v základním toku, ve kterém se pokračuje. Pro ukázku byl zvolen jeden případ užití. Zbytek scénářů je uveden v příloze A. Specifikace případů užití. [30]

Název	UC1 Vložit požadavek
Krátký popis	Use Case umožňuje uživateli vytvořit nový požadavek do systému.
Aktéři	Neregistrovaný uživatel, zadavatel
Základní tok	1. Uživatel inicializuje vložení nového požadavku. 2. Systém zobrazí prázdný formulář pro vložení požadavku. 3. Uživatel vyplní povinná pole (název, popis, kategorii požadavků) a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží požadavek a odešle notifikační email. 6. Systém zobrazí stránku potvrzující přijetí požadavku.
Alternativní tok 1	2a. V případě, že je uživatel přihlášený, systém zobrazí předvyplněný formulář pro vložení požadavku.
Alternativní tok 2	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí požadavek vytvořit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

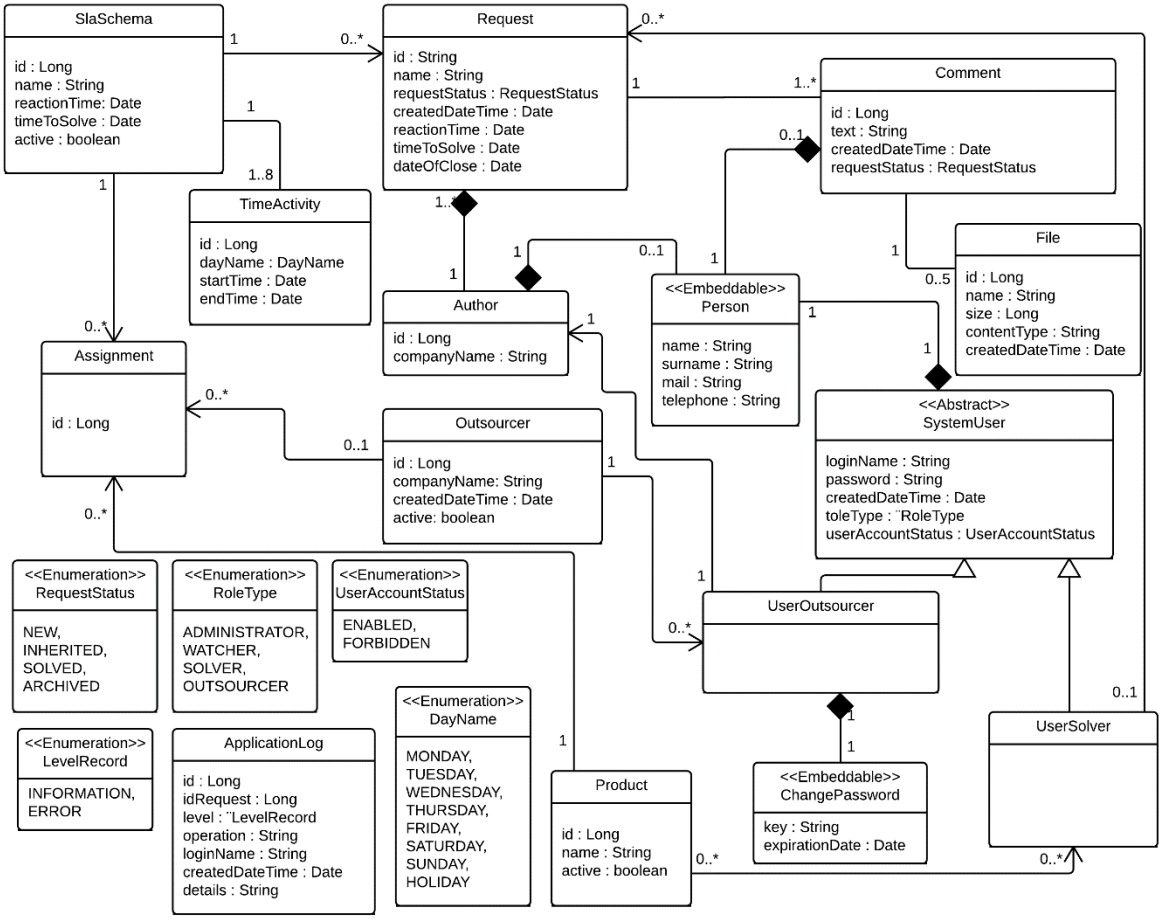
Tabulka 1: Specifikace případu užití

4.4 Class diagram

Class diagram je diagram tříd, které systém obsahuje. Prvky diagramu přepisují programátoři do programovacího jazyka aplikace. V diagramu musí být nakresleny všechny třídy, které jsou potřeba k implementaci. Základním stavebním kamenem diagramu je třída. Třída se skládá z názvu, atributů a metod. Názvy prvků ve třídě nesmí obsahovat diakritiku. [31]

Třídy se spojují určitým typem vztahu. V diagramu, který bude vytvořen k doméně helpdesk, se použije několik vztahů. První vztah je Asociace. Asociace znamená, že každá třída existuje nezávisle na jiné, se kterou je spojena. Druhým vztahem je kompozice. Jedna třída je označena jako část, kdy bez druhé nemůže existovat. Třetí použitou vazbou je generalizace. Generalizace je dědičnost, kdy jedna třída dědí funkcionalitu jiné třídy. [32]

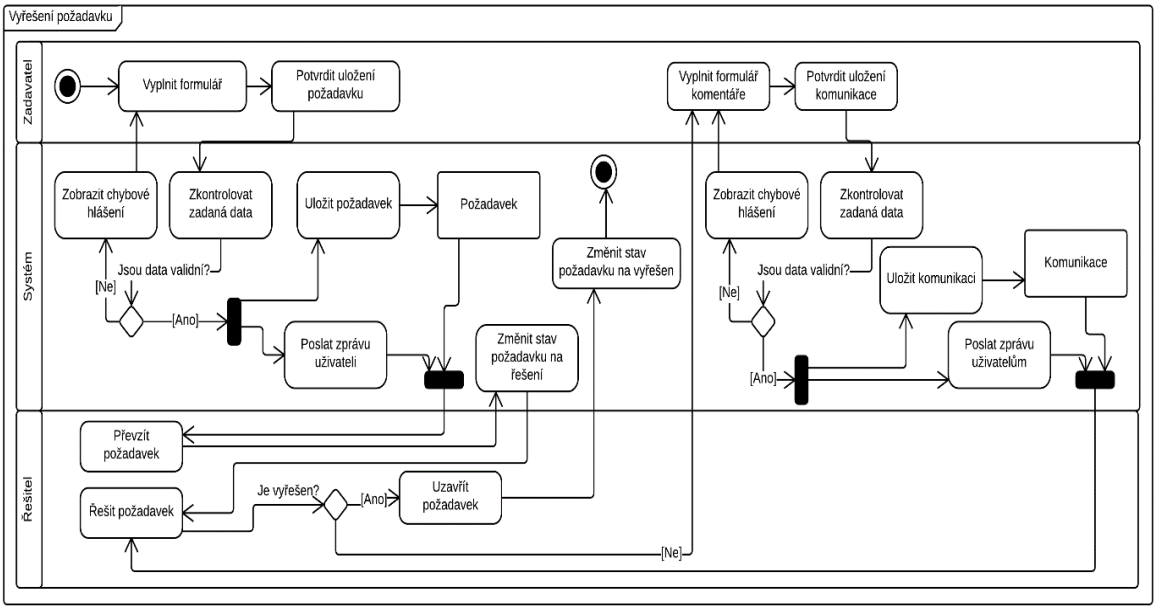
Class diagram vytvořený pro doménu obsahuje třídy, které korespondují s datovým modelem. Třídy obsahují pouze atributy. Metody nebyly uvedeny, protože tyto třídy obsahují pouze metody get a set.



Obrázek 12: Class diagram [autor]

4.5 Aktivity diagram

Dalším krokem analýzy je tvorba aktivity diagramu. *„Diagram aktivit (Activity Diagram) je typem diagramu interakcí, který se používá pro popis procedurální logiky, byznys procesů či pracovních postupů.“* [33] Na diagramu je vidět průchod procesu, jak se postupně vykonává. Základním prvkem je aktivita. Celá aktivita je rozdělena na několik oddílů. Každý oddíl znázorňuje typ uživatele, který spravuje určitou část aktivity. Aktivita musí nějak začínat a nějak končit. K tomuto jevu slouží uzly. Nejzákladnějším prvkem aktivity je akce. Jedná se o primitivní jednotku, která se nedá dále dělit. Jednotlivé akce jsou spojovány hranou. Ta ukazuje průchod z jedné akce na další. Uzly použité na obrázku jsou rozhodovací a větvící uzly.

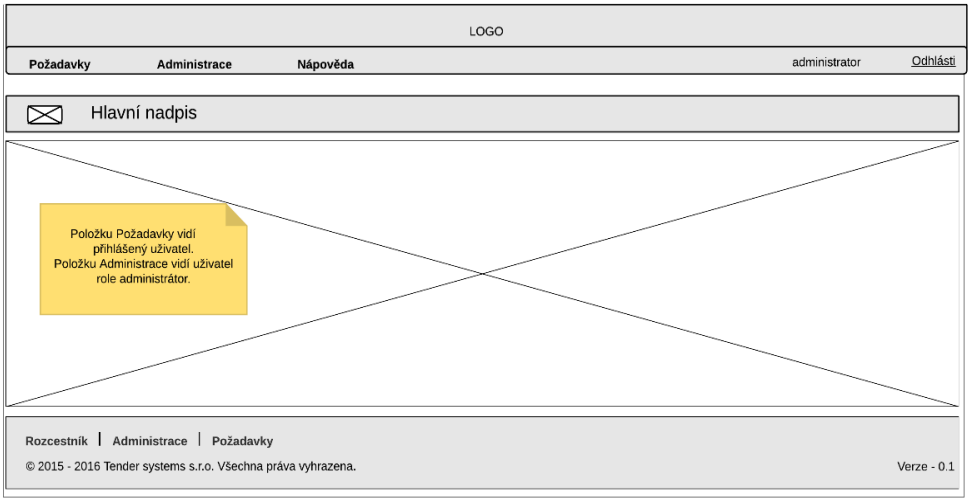


Obrázek 13: Aktivity diagram [autor]

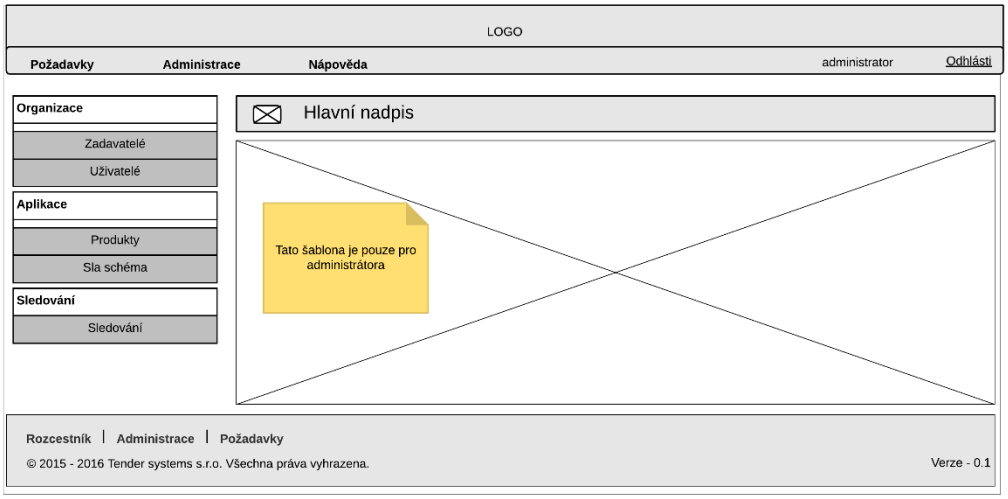
4.6 Wireframe

Při tvorbě aplikace je dobré vytvořit návrh vzhledu základních stránek. Analytik kreslí tzv. wireframe. „*Wireframe definuje rozmístění funkčních prvků na stránce. Nejedná se v žádném případě o grafický návrh, neobsahuje obrázky a je tvořen pouze pomocí čar a textu. Nedoporučuje se ani použití barev, až na výjimky, které je potřeba odlišit.“* [34] Tento návrh definuje obsah a rozložení prvků na stránce. Při projektových schůzkách mezi analytikem a zákazníkem se na těchto nákresech ukazuje, jak systém bude vypadat. V praxi to znamená, že když s nimi obě strany souhlasí, tak už nemohou být provedeny velké změny bez finanční náhrady zákazníkem.

Aplikace helpdesk je navržena do dvou základních šablon. Šablony se liší především v návrhu menu. První šablonou je šablona bez panelu. To znamená, že je použito pouze horizontální menu. Tato šablona je navržena pro stránky, kam mají přístup všichni uživatelé. V horizontálním menu se nachází položky Požadavky, Administrace, Nápověda a přihlašovací jméno s odkazem pro odhlášení. Položku Požadavky vidí pouze přihlášení uživatelé. Administrace se zobrazuje uživatelům s rolí administrátor. Nápověda je přístupná všem uživatelům. Druhá šablona je použita pro administrátorskou část. Do administrace má přístup pouze uživatel s rolí administrátor. Tato šablona je rozdělena do tří sekcí. V první sekci je správa zadavatelů a uživatelů. V druhé je správa produktů a SLA schéma. Ve třetí sekci je položka pro zobrazení auditní stopy uživatelů.

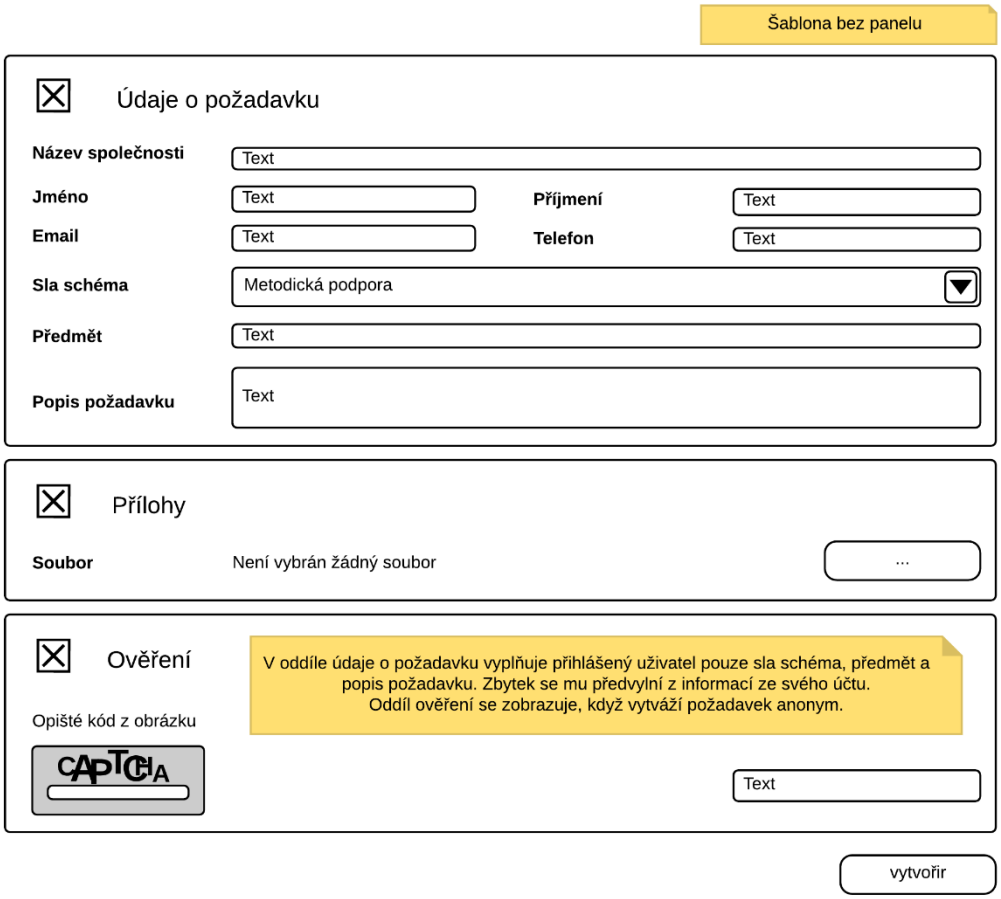


Obrázek 14: Šablona bez panelu [autor]



Obrázek 15: Šablona s panelem [autor]

Hlavní funkcí helpdesku je poskytnout uživateli možnost vytvořit požadavek na jeho problém. Při tvorbě wireframu bylo cíleno na co nejefektivnější způsob zadání požadavku do systému. Systém musí nabídnout uživatelsky přívětivou interakci. Požadavek zadávají dva druhy uživatelů. V návrhu formuláře je vidět, co jaký uživatel vyplňuje.



Obrázek 16: Šablona vytvoření požadavku [autor]

4.7 Bezpečnost

V současnosti prakticky každá organizace používá nějaký informační systém. Přesto, nebo možná právě proto, se snaží, aby riziko úniku citlivých dat bylo minimální. Na bezpečnost je proto nutné se zaměřit už při vývoji informačních systémů. Bezpečnostní architektura informačního systému by měla proto být postavena na principu „vrstvené ochrany“ tak, aby dopad neúčinnosti jednotlivého bezpečnostního prvku byl minimalizován dalšími opatřeními, případně k reálnému negativnímu dopadu ani nedošlo. [35]

Aplikace bude rozdělena do 3 vrstev. Z toho vyplývá, že toto rozdělení bude bezpečnostní opatření. Žádná vrstva nemůže obsluhovat více jak jednu vrstvu. V aplikaci bude použita prezenční vrstva. V ní budou umístěny html šablony a CDI beany. Další bude business vrstva, která se postará o celou logiku aplikace. Nejnižší vrstvou bude datová vrstva, která se bude starat o reprezentování objektů z databáze. [36]

Dalším bezpečnostním prvkem budou třídy, které budou obrazem entitních tříd. Výhoda bude, že do webové vrstvy se entity vůbec nedostanou, takže v nich nedojde k úmyslnému změnění dat.

Nejdůležitějšími prvky ve vytváření aplikace jsou pojmy autorizace a autentizace. „*Autentizace je operace, při které zjišťujeme totožnost subjektu. Autorizace je operace, při které zjišťujeme, jestli je subjekt oprávněn k nějaké činnosti, např. přístup k objektu.“* [37]

Autentizace bude tvořena přihlašovacím formulářem. Celá správa autentizace bude probíhat vůči aplikačnímu serveru. V něm bude definována security doména a přihlašovací modul. Uživatel bude porovnáván s uživateli uloženými v databázi. Úspěšně autentizovanému uživateli bude vždy přiřazena role, která bude korespondovat s hodnotou výčtového typu. Stejným způsobem jako přihlášení bude probíhat odhlášení. Po kliknutí na tlačítko odhlásit ve formuláři ztratí uživatel sezení a má anonymní identitu.

Autorizace se bude kontrolovat na dvou místech. Bezpečnostní omezení se budou kontrolovat v prezenční vrstvě. Jedná se o to, že na určitá místa v aplikaci se mohou dostat pouze vybraní uživatelé. Umístěním jsou definovány povolené role, které si mohou stránku zobrazit. Pouze toto omezení je nedostačující. Aplikace by měla mít kontrolu i na ostatních vrstvách. Důvodem zabezpečení na více vrstvách je ten, že programátor je pouze člověk. Stačí neuvést jedno webové omezení a už nastává hluché místo. Je dobré mít jednu sérii kontrol na několika vrstvách. Při vývoji helpdesku se bude kontrolovat bezpečnost na business logice. [37]

Formulář pro vytvoření požadavku, který bude vyplňovat nepřihlášený uživatel, je třeba zabezpečit nějakou technologií. Aplikace musí být schopna ověřit, zda formulář vyplňuje člověk či stroj. K tomu by měla být vhodná implementace Turingova testu. „*Cílem počítačového programu je přesvědčit člověka o svém "lidství" - počítač je úspěšný v případě, že bude na základě konverzace pokládán za člověka.“* [38]

Uživatelé se budou ověřovat vůči technologii Captcha. Tato technologie pracuje na principu generování textu do obrázku. Tento text je pro robota zkreslený a nedokáže ho přečíst. Ve formuláři bude generován matematický součet, kdy uživatel zapíše výsledek do textového pole. [39]

5 Implementace

Při implementaci webové aplikace musí vývojový tým překonávat různá úskalí. Největším problémem, který každý tým musí řešit, je internetové připojení zákazníka. Webová stránka, kterou si uživatel načítá, musí mít co nejmenší velikost, aby se za jakýchkoliv podmínek rychle načetla.

Firma vytvářející webovou aplikaci si musí dát pozor na širokou škálu zařízení, operačních systémů a webových prohlížečů, které zákazníci používají. Dříve se aplikace optimalizovaly pouze pro stolní počítače a notebooky. Dnes, kdy domácnost obsahuje mnoho chytrých zařízení, musí aplikace být čitelná na mobilních telefonech, tabletech a dalších zařízeních. Postupem času vznikají nové instance nebo verze webových prohlížečů na různých platformách. Vývojový tým by měl garantovat použitelnost aplikace na co nejvyšším počtu prohlížečů.

Vývoj aplikace je komplexní záležitost, takže mnohdy nezůstane u původní analýzy. Postupem času, kdy přibývá funkcionalita, se naráží na problémy, které se v prvním návrhu neobjevily. Tyto změny zpravidla zlepšují aplikaci, ale nemusí to tak být nutně. Tým musí uvážit, zda tyto změny opravdu vylepší aplikaci. Pokud dojde ke změně systému, tak je nutné, aby nebyla příliš změněna funkce aplikace.

5.1 Použité technologie

Aplikace je tvořena jazykem Java ve verzi 8. Tato verze je nejnovější, její použití je tedy výhodné díky novým funkcím oproti předchozím verzím. Při vývoji jsou používány nové funkce času. Pomocí nových tříd se počítá čas na reakci a čas na vyřešení. Další důvod pro tuto verzi je takový, že v každé nové verzi je bezpečnost na lepší úrovni. Díky použití Javy se použije Java EE ve verzi 7. Obě technologie garantují oboustrannou podporu. Testy jsou psány technologií Arquillian ve verzi 1.0.0.Alpha v kombinaci s JUnit ve verzi 4.12.

Aplikace musí být zabalena tak, aby s ní dokázal pracovat aplikační server. Pro tvorbu buildu se použije nástroj Maven. Neudělá nic jiného, než že vytvoří EAR soubor. Do souboru jsou přidány zkompilované Java třídy, knihovny, HTML, CSS a XML soubory. [40]

Aplikace běží na aplikačním serveru WildFly ve verzi 9. Tato verze aplikačního serveru byla zvolena, protože běží na serverech firmy Tender systems s. r. o. S Wildfly souvisí mnoho použitých knihoven. Ty nejsou součástí projektu, ale jsou importovány z databáze knihoven aplikačního serveru. Pro konverzi dat mezi relační databází a objektově programovacím jazykem, tzv. ORM, byla použita technologie Hibernate ve verzi 4.3.10.Final. Na serverech firmy běží databázový systém PostgreSQLve verzi 9.5.

Dále je použit itext. Itext nabízí službu, která dokáže v jazyce Java vygenerovat soubory ve formátu PDF. V našem případě se generují veškeré seznamy v aplikaci. Webová vrstva je napsána v jazyce JSF ve verzi 2.2. Tento jazyk je vytvořen na základě xml tagů a dokáže pracovat s třídami napsanými v Javě. Další technologií použitou na webové vrstvě je RichFaces ve verzi 4.5.1.Final. Jde o nadstavbu jazyka JSF. Z tohoto frameworku je použita komponenta pro realizaci kalendáře. Celý vzhled stránek je implementován v jazyku CSS.

5.2 Verzování

Je to proces, kdy na projektu většího rozměru pracuje více lidí a vznikají různé verze. Každý člověk v týmu vykoná změnu kódu. Takový kód je třeba propojit s jiným, který používají všichni vývojáři. Každé takové změně se říká nová verze, tzv. revize. Vývojář si synchronizuje svoji lokální verzi vůči centrální. [41]

Pro vývoj aplikace je použito uložiště, které poskytla firma Tender systems s. r. o. Toto uložiště pracuje na principu SVN. „*Subversion je centralizovaný systém pro správu verzí*.“ [41] SVN je uložen na serveru a z něho si každý uživatel stahuje soubory na svůj lokální stroj. Je vhodné se každý den synchronizovat vůči repozitáři. Změně, která je stažena ze serveru se říká „update“. Opakem stahování je nahrávání. To je prováděno, když existuje změna, se kterou by měli pracovat všichni uživatelé. Tomuto jevu se říká „commit“. Subversion nabízí kompletní historii s číslem revize. Výhodou je, že nedovolí bezhlavé nahrávání změn. Pokud jsou soubory v konfliktu s jinými, musí se určit, která verze je vhodnější.

5.3 Nasazení

Aplikace je nahrána na server, který poskytla firma. Na serveru běží operační systém CentOS. Pro aplikaci je přiděleno 8 GB RAM paměti. Velikost disku je 5 GB. Na server se připojuje pomocí zabezpečeného komunikačního protokolu SSH. Díky romuro protokolu můžeme pracovat v příkazové řádce našeho počítače. [42]

Na serveru byla připravena čistá verze Wildfly. Do složky standalone byl překopírován konfigurační soubor standalone.xml. Tento soubor se nachází v příloze C. Obsah kompaktního disku. Dále byla vytvořena databáze. Po připravení serveru se vložil build aplikace do složky deployment ve WildFly. Aplikační server byl spuštěn a po kontrole funkčnosti byla aplikace předána k testování.

```
Create database helpdesk;
/etc/init.d/wildfy start
```

Zdrojový kód 12: Přikaz pro vytvoření databáze a spuštění serveru

5.4 Testování

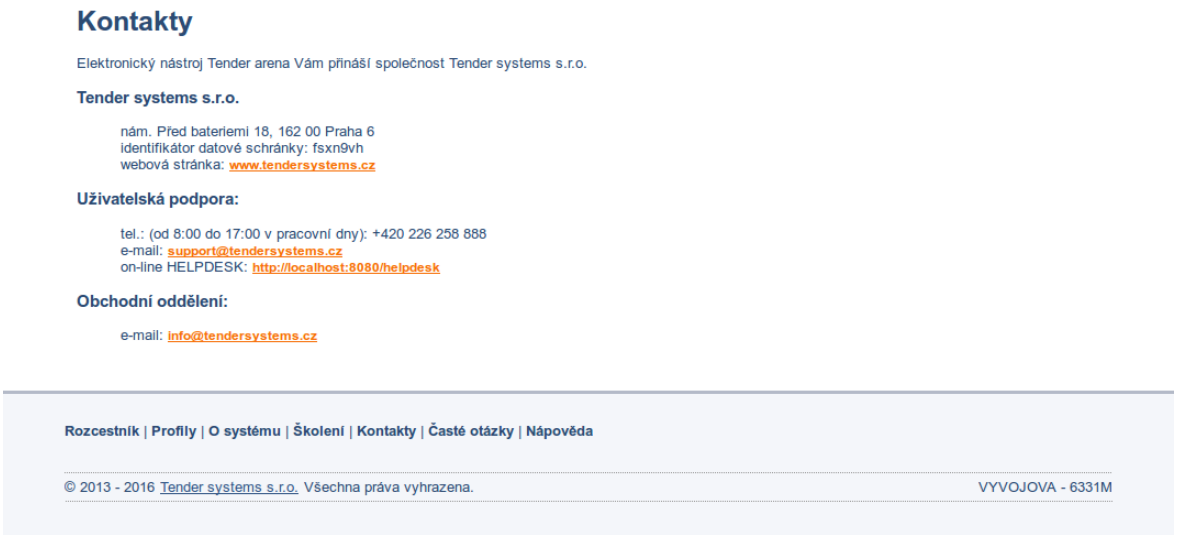
Vývoj aplikace helpdesk je proveden formou TDD, které bylo popsáno v kapitole Testovací nástroje. Od samého začátku vývoje byly psány jednotkové testy a integrační testy. Pro rychlost a efektivitu vývoje bylo zvoleno, že se testuje pouze EJB modul. Důvod tohoto řešení je, že v tomto modulu se provádí prakticky veškerá logika aplikace. Jednotkové testy jsou psány nástrojem JUnit u tříd, které neprovádí komunikaci s databází. Integrační testy jsou vytvořeny nástrojem Arquillian. Tyto testy jsou pro EJB. Po vytvoření většího celku aplikace, se prováděly funkční testy. Při funkčním testu se zkoušelo použití na co nejvíce prohlížečích. Velikost testovací funkce se odvíjela od jejich složitosti. Do polí webových formulářů byla vyplňována celá škála správných i nesprávných hodnot a kontrolovalo se, jak se aplikace chová při takovém zadávání dat. Při testování se sledovalo i to, jak aplikace působí na uživatele, kteří jí budou používat. Po konci vývoje byla aplikace nasazena na server. V tu chvíli nastalo uživatelské testování pracovníků firmy Tender systems s. r. o.

5.5 Funkcionalita

V této kapitole jsou popsány nejzajímavější části aplikace. Všechny stránky uživatelského prostředí jsou umístěny v příloze B. Snímky uživatelského prostředí. Po spuštění aplikace se na úvodní stránce objeví odkaz na přihlašovací formulář. Stránka je ukázána na snímku s titulkem Obrázek příloha 1: Úvodní stránka [autor]. Uživatel s vytvořeným účtem se může přihlásit do aplikace. Po přihlášení nabízí aplikace několik úkonů. Tyto úkony může provádět pouze uživatel, který k nim má přístup. Základním kamenem helpdesku je úkon zadání a řešení požadavku. Celý vývoj se točil okolo této funkcionality. Do formuláře pro vytvoření požadavku se přímo nedostaneme z žádného místa v aplikaci. Aplikace byla vytvořena tak, že odkaz na tento formulář musí být vložen do konkrétního produktu. V našem případě byla upravena Tender arena o následující tag.

```
<h:link value="#{texts['button.helpdesk']}
outcome="http://localhost:8090/helpdesk/request/create.jsf?idProduct=2" />
```

Zdrojový kód 13: Odkaz na helpdesk



Obrázek 17: Odkaz na helpdesk [autor]

Po kliknutí na odkaz se zobrazí stránka s výběrem, zda chceme požadavek vložit jako přihlášený, nebo anonymní uživatel, viz Obrázek příloha 9: Stránka pro vytvoření požadavku [autor]. Část xhtml šablony, která je na následující ukázece obsahuje dva jsf tagy. Jsou to odkazy pro otevření formuláře přihlášeným uživatelem a anonymem.

```
<h:panelGroup styleClass="areaActionButtons formAction">
  <h:link value="#{texts['button.createLoggedRequest']}"
    outcome="createLoggedRequest" rendered="#{not securityAction.userLogged ||
    securityAction.isRoleType('OUTSOURCER')}"
    styleClass="formButton actionButton"/>
  <h:link value="#{texts['button.createAnonymousRequest']}"
    outcome="createAnonymousRequest" rendered="#{not securityAction.userLogged
    || not securityAction.isRoleType('OUTSOURCER')}"
    styleClass="formButton actionButton"/>
</h:panelGroup>
```

Zdrojový kód 14: Šablona vytvoření požadavku

Po vybrání a kliknutí na odkaz způsobu zadání se nám zobrazí formulář. V případě anonymního uživatele je fomulář znázorněn na Obrázku příloha 10: Stránka pro vložení požadavku anonymem [autor]. Při druhé variantě vypadá formulář jako Obrázek 11: Stránka pro vložení přihlášeným uživatelem [autor].

Formulář je rozdělen do tří oddílů. V prvním s názvem Údaje o požadavku se vyplňují pole o uživateli a SLA schématu. Přihlášenému uživateli se některé pole z jeho účtu

předvyplní. Všechna pole, která se nachází v tomto oddíle jsou znázorněna v následující tabulce.

Název pole	**Povolená hodnota**	**Předvyplnění**
Název společnosti	Řetězec o velikosti 1 až 255	Ano
Jméno	Řetězec o velikosti 1 až 50	Ano
Příjmení	Řetězec o velikosti 1 až 50	Ano
Email	Řetězec o velikosti 1 až 50	Ano
Telefon	Řetězec ve formátu [+420xxxxxxxxx]	Ano
Sla schéma	Hodnota z roletky	Ne
Předmět	Řetězec o velikosti 1 až 255	Ne
Popis požadavku	Řetězec o velikosti 1 až n znaků	Ne

Tabulka 2: Tabulka polí údajů o požadavku

Ve druhém oddíle s názvem přílohy se vkládají soubory. Uživatel přikládá soubory, které souvisí s požadavkem. Přilohy jsou vkládány následujícím tagem.

```
<h:inputFile value="#{fileUploadAction.file}">
  <f:ajax render="fileUploadContainer
   attachmentTable" onevent="setFileName(this)"
    listener="#{actionFile.addFileData()}" />
   <rich:validator />
</h:inputFile>
```

Zdrojový kód 15: Vložení souboru

Současně může být k požadavku vloženo více souborů. Neukládají se fyzicky do databáze. V databázi vznikne pouze jejich záznam. Binární data přílohy se ukládají do datového úložiště serveru. Cesta k souboru se dohledá ze záznamu v databázi. Pattern cesty k souboru vypadá následovně.

```
/rok/měsíc/den/
```

Zdrojový kód 16: : Pattern souborové struktury

Třetí sekce s názvem Ověření se zobrazí pouze anonymnímu uživateli. Při anonymním zadávání se ověřuje, zda formulář nevyplňuje robot. Do formuláře byl implementován jednoduchý matematický součet. Pro tento test vznikla CDI beana CaptchaAction, která obsahuje metodu validate. Metoda kontroluje zadanou hodnotu s očekávanou. Pokud se hodnoty shodují, dovolí aplikace uložit formulář. Pokud se neshodují, objeví se validační zpráva.

```
FacesContext context = FacesContext.getCurrentInstance();
UIComponent component = event.getComponent();
UIInput enteredValueArray = (UIInput) component.findComponent("checkCaptcha");
String enteredValue = enteredValueArray.getLocalValue() == null ? "" :
enteredValueArray.getLocalValue().toString();

if (!enteredValue.equals("") &&
    Integer.parseInt(enteredValue) != expectedValue) {
  context.addMessage(enteredValueArray.getClientId(), new
  FacesMessage(FacesMessage.SEVERITY_ERROR, "",
  Tools.getText("error.valueIsBad")));
  context.renderResponse();
}
```

Zdrojový kód 17: Metoda pro captchu

Po vyplnění formuláře se klikne na tlačítko vytvořit. Pokud je formulář validní, nastává proces uložení do databáze. Před samotným uložením se musí vypočítat čas na reakci a čas na vyřešení z vybraného SLA schématu. Při implementaci výpočtu času se musela vzít v potaz následující kritéria. Prvním krokem je hlídat, zda název dne, kdy je požadavek zadán, je ve zvoleném SLA schématu povolen. Druhým krokem se kontroluje, jestli čas vytvoření spadá do pracovní doby, která je určena SLA schématem. Třetím krokem je kontrola, zda den vytvoření není svátek. Pokud je svátek, kontroluje se, zda je v SLA schématu určen jako pracovní den. Pokud není, musí se přeskočit. Nejprve vznikl test pro výpočet.

```
public void shouldEquelsDates1() {
  ToolsForRequestDates tools = new ToolsForRequestDates();
  tools.calculateRequestDates(prepareRequest("12.06.2016 13:00:00"));
  Assert.assertEquals("Thu Jun 16 09:00:00 CEST 2016",
  tools.getReactionTime().toString());
  Assert.assertEquals("Thu Jun 16 11:00:00 CEST 2016",
  tools.getTimeToSolve().toString());
}
```

Zdrojový kód 18: Test kontroly výpočtu času požadavku

Pro výpočet vznikla třída ToolsForRequestDates, která obsahuje jednu veřejnou metodu calculateRequestDate. V metodě se počítají oba dva časy. Nejprve se kontroluje, zda SLA schéma má zadané pracovní dny, k tomu slouží metoda prepareTimeActivityForHoliday.

```
for (TimeActivity timeActivity : timesActivity) {
  if (DayName.HOLIDAY.equals(timeActivity.getDayName())) {
    return timeActivity;
  }
}
```

Zdrojový kód 19: Metoda pro zjištění zda sla schéma obsahuje svátek

Po provedení kontroly existence svátku v SLA schématu, přichází na řadu iterace přes všechny časy aktivity SLA schématu. Při každém iteračním kroku se zjišťuje, zda aktuální den výpočtu je nastaven v SLA schématu. Pokud jsou všechny podmínky splněny, přechází se k výpočtu času.

```
if
(actualDate.toLocalTime().compareTo(convertDateToLocalTime(timeActivity.getSta
rtTime())) == -1) {
  LocalTime tempTime = convertDateToLocalTime(timeActivity.getStartTime());
  actualDate = LocalDateTime.of(actualDate.getYear(), actualDate.getMonth(),
  actualDate.getDayOfMonth(), tempTime.getHour(), tempTime.getMinute(),
  tempTime.getSecond());
}
actualDate                                                                =
actualDate.plusHours(updateTime.getHour()).plusMinutes(updateTime.getMinute())
.plusSeconds(updateTime.getSecond());

if
(actualDate.toLocalTime().compareTo(convertDateToLocalTime(timeActivity.getEnd
Time()))==                                1                                ||
actualDate.toLocalTime().compareTo(convertDateToLocalTime(timeActivity.getStar
tTime())) == -1) {
   updateTime = calculateDivideTime(timeActivity.getEndTime());
} else {
   updateTime = LocalTime.of(0, 0, 0);
}
```

Zdrojový kód 20: Výpočet času požadavku

V této kapitole byla ukázána funkce vložení požadavku. Aplikace se skládá z dalších funkcí, viz analýza. Všechny snímky aplikace jsou znázorněny v příloze B. Snímky uživatelského prostředí. Všechny zdrojové kódy, tj. třídy, testy, xhtml soubory, css soubory, jsou přiloženy v příloze C. Obsah kompaktního disku. Při vývoji byly použity třídy, které byly firmou poskytnuty, jako její standardy. Seznam tříd je uveden v tabulce v příloze D. Seznam použitých tříd.

5.6 Uživatelská dokumentace

K aplikaci byla vytvořena uživatelská dokumentace. „*Dokumentaci lze chápat jako interaktivní nápovědu aplikace, zároveň je však i strukturovaným manuálem*“. [43] Uživatelskou příručku si může stáhnout každý návštěvník aplikace. V dokumentaci jsou popsány postupy, jak má uživatel pracovat se systémem. U každého postupu je uveden popis a obrázek. Uživatelská příručka je přiložena na kompaktním disku v příloze C. Obsah kompaktního disku.

Obrázek 18: Stránka uživatelské dokumentace [autor]

6 Shrnutí výsledků

Cílem bakalářské práce bylo prozkoumat a rozebrat jednotlivé testovací nástroje k vývoji webové aplikace v jazyku Java. Nástroje byly teoreticky popsány a prakticky vyzkoušeny. Po prozkoumání možností byl vybrán ten nejvhodnější pro vytvoření aplikace helpdesk. Veškerý postup vývoje aplikace byl rozebrán.

Webová aplikace byla vytvořena pomocí dvou testovacích nástrojů. Pro standardní Java třídy byl použit nástroj JUnit. Pro EJB beany byly psány testy pomocí nástroje Arquillian. Vývoj vycházel z předem definovaného návrhu. Datový model koresponduje s class diagramem. Funkce, které aplikace musí splňovat, byly sepsány do požadavků. Grafický vzhled aplikace byl vytvořen podle nakreslených Wireframů. Grafika byla optimalizovaná tak, aby byla správně zobrazena na co nejvíce webových prohlížečích. Helpdesk byl navržen, aby splňoval co nejvíce bezpečnostních pravidel.

7 Závěry a doporučení

Vývoj informačního systému může být prováděn v různých programovacích jazycích. Tvorba aplikace je mravenčí práce. Dochází ke složitému procesu, kde každý díl skládačky musí zapadat do celku. V tomto oboru se točí nemalé peníze, protože je potřeba mít kvalitní pracovníky týmu. Výsledek aplikace musí vždy splňovat požadavky zákazníka.

Webové aplikace dosáhly v dnešních dnech na velké popularity. Díky vysokému pokrytí internetovým připojením je možné s aplikací pracovat kdekoliv a na jakémkoliv zařízení. Nevýhodou webových aplikací je vysoka internetová kriminalita. Doba přináší nové technologie, které aplikace posouvají o krok dále k dokonalosti. Vývojář musí stále sledovat nové trendy a zajímat se o ně.

S nástupem moderních technologií se změnil pohled na používání počítačů. Toto odvětví má vysoký podíl na obchodním trhu. Firmy se snaží nabízet jak aplikace na míru, tak i balíčky funkcionalit, které využije více zákazníků. To, co se před deseti lety evidovalo v papírové formě, se dnes převádí do elektronické podoby. Příkladem takových systémů jsou bankovní portály nebo evidence veřejných zakázek.

Při vývoji se tým vývojářů snaží používat testy, které vedou k dokonalosti aplikace. Testy kontrolují jak nové funkce, tak i upravené funkcionality. Pokud se testy píší správným způsobem, tak ulehčí týmu objevení bagů v aplikaci ještě před odevzdáním zákazníkovi.

Tato bakalářská práce by mohla být dobrým podkladem pro rozšíření aplikace. Uživatele používající aplikaci by v budoucnu uvítali správu svého uživatelského účtu. Zajímavou funkcí by bylo ukládání jejich filtrů pro seznamy, aby uživatel mohl filtrovat podle podmínek, které často používá. Do budoucna by bylo vhodné vkládání požadavku realizovat webovou službou. To by mělo za výhodu, že by si každá aplikace sama definovala vzhled formuláře podle svého standardu.

8 Seznam použité literatury

[1] FIALA, Petr. Helpdek- porovnání systému a návrh implementace vybraného. Praha, 2014. Bakalářské práce. Vysoká škola ekonomická v Prace, Fakulta informatiky a statistiky. Vedoucí práce Ing. Tomáš Bruckner, Ph.D.

[2] Webový helpdeskový systém pro efektivní správu požadavků a řízení firemních úkolů. taskpool.cz [online]. ©2015 [cit. 2016-02-01]. Dostupné z: http://taskpool.cz/data/dokumenty/produktovy-list-1398347541.pdf

[3] Možnosti a využití. helpdesk-software.cz. [online]. ©2010 [cit. 2016-02-01]. Dostupné z: http://helpdesk-software.cz/o-helpdesku/moznosti-vyuziti/

[4] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. 1. vyd. Brno: Computer Press, 2013, 208 s. ISBN 978-80-251-3816-8.

[5] Pospíchal, Vlastimil. Test-driven development v jazyce Java. Root. [online]. 2013-01-10 [cit. 2016-01-10]. Dostupné z: http://root.cz/clanky/test-driven-development-v-jazyce-java/

[6] Sirkovský, Marek. Testováním řízený vývoj. Programujte. [online]. 2006-05-21 [cit. 2016-01-10]. Dostupné z: http://programujte.com/clanek/2006051006-testovanim-rizeny-vyvoj/

[7] Hlava, Tomáš. Fáze a úrovně provádění testů. Testování softwaru. [online]. 2011-08-21 [cit. 2016-01-12]. Dostupné z: http://testovanisoftwaru.cz/tag/systemove-testovani/#developer

[8] Crume, Jimm a kolektiv. EJB fundamentals and session beans: Begin exploring the world of EJBs. Javaworld. [online]. 2006-02-13[cit. 2016-01-12]. Dostupné z: http://javaworld.com/article/2071724/java-web-development/ejb-fundamentals-and-session-beans.html

[9] SLA (Service Level Agreement). Management Mania. [online]. 2015-12-02 [cit. 2016-01-14]. Dostupné z: https://managementmania.com/cs/service-level-agreement

[10] Dokument: Report o průběhu plnění SLA [online]. [cit. 2016-01-15]. Dostupné z: http://mbi.vse.cz/public/cs/obj/DOCUMENT-27

[11] O společnosti. Tendersystems. [online]. ©2012 [cit. 2016-01-15]. Dostupné z: http://tendersystems.cz/o-spolecnosti-327.htm

[12] Thomas, Peter. Chapter 2.Features. [online]. ©2008 [cit. 2016-01-17]. Dostupné z: http://jtrac.info/

[13] Taskpool. [online]. ©2015 [cit. 2016-01-19]. Dostupné z: http://taskpool.cz/

[14] Hordějčuk, Vojtěch. JUnit4. [online]. ©2008-2016 [cit. 2016-01-20]. Dostupné z: http://voho.cz/wiki/java-junit/

[15] Hordějčuk, Vojtěch. JPA (Java Persistence API) [online]. ©2008-2016 [cit. 2016-01-20]. Dostupné z: http://voho.cz/wiki/java-jpa/

[16] Hordějčuk, Vojtěch. Mockito [online]. ©2008-2016 [cit. 2016-01-23]. Dostupné z: http://voho.cz/wiki/mockito/

[17] Arquillian tutorial. Mastertheboss. [online]. 2013-09-29 [cit. 2016-01-23]. Dostupné z: http://mastertheboss.com/jboss-frameworks/arquillian/arquillian-tutorial

[18] Knutsen, Aslak. Drone. docs.jboss.org. [online]. 2014-06-15 [cit. 2016-01-24]. Dostupné z: https://docs.jboss.org/author/display/ARQ/Drone

[19] Needle for JAVA EE. needle.spree.de. [online]. ©2010-2012 [cit. 2016-01-24]. Dostupné z: http://needle.spree.de/public/documentation/2.2/html/NeedleReference.html

[20] Allen, Dan. Getting Started. Arquillian. [online]. 2015-08-02 [cit. 2016-01-25]. Dostupné z: http://arquillian.org/guides/getting_started/

[21] Majsak, Bartosz. Persistence. [online]. docs.jboss.org. 2015-03-06 [cit. 2016-02-01]. Dostupné z: https://docs.jboss.org/author/display/ARQ/Persistence

[22] Hanák, Drahomír. Jak rychle a kvalitně vytvořit (nejen) webovou aplikaci. ITnetwork. [online]. 2013-01-28 [cit. 2016-02-01]. Dostupné z: http://itnetwork.cz/aktivita/article/jak-rychle-a-kvalitne-vytvorit-webovou-aplikaci

[23] Knesl, Jiří. Je analýza software mrtvá?. knesl.com. [online]. 2011-11-01 [cit. 2016-02-01]. Dostupné z: http://knesl.com/articles/view/je-analyza-software-mrtva

[24] Žegklitz,Martin. Lucidchart- diagramy online. Metodický portál. [online]. 2014-01-09 [cit. 2016-02-01]. Dostupné z: http://spomocnik.rvp.cz/clanek/18235/LUCIDCHART---DIAGRAMY-ONLINE.html

[25] Subscription Level. lucidchart.com. [online]. ©2016 [cit. 2016-02-01]. Dostupné z: https://lucidchart.com/users/level

[26] Frey, Chuck. LucidChart raises the bar for diagramming applications. The mind mapping software blog. [online]. 2013-02-22 [cit. 2016-02-01]. Dostupné z: http://mindmappingsoftwareblog.com/lucidchart-review/

[27] Čápka, David. 1.díl – Úvod do UML. ITnetwork. [online]. ©2016 [cit. 2016-02-01]. Dostupné z: http://itnetwork.cz/navrhove-vzory/uml/uml-uvod-historie-vyznam-a-diagramy/

[28] Kajzar, Dušan. C) Specifikace požadavků na system. Projektování informačních systémů II. [online]. [cit. 2016-02-01]. Dostupné z: http://zdenek2.euweb.cz/doc3/prois7c.html

[29] Čápka, David. 2. díl - UML - Use Case Diagram. ITnetwork. [online]. ©2016 [cit. 2016-02-04]. Dostupné z: http://itnetwork.cz/aktivita/article/uml-use-case-diagram

[30] Čápka, David. 3. díl - UML - Use Case Specifikace. ITnetwork. [online]. ©2016 [cit. 2016-02-04]. Dostupné z: http://itnetwork.cz/navrhove-vzory/uml/uml-use-case-specifikace-diagram

[31] Čápka, David. 5. díl - UML - Class diagram. ITnetwork. [online]. ©2016 [cit. 2016-02-07]. Dostupné z: http://itnetwork.cz/navrhove-vzory/uml/uml-class-diagram-tridni-model/

[32] Němec, Miloš. UML: Diagramy tříd. milosnemec.cz. [online]. 2010-05-15 [cit. 2016-02-07]. Dostupné z: http://milosnemec.cz/clanek.php?id=199

[33] Rejnková Petra. Diagram aktivit. uml.czweb.org. [online]. ©2009 [cit. 2016-07-12]. Dostupné z: http://uml.czweb.org/diagram_aktivit.htm

[34] Webové stránky pro školy. www.proskoly.cz. [online]. ©2009 [cit. 2016-02-08]. Dostupné z: http://wwwproskoly.cz/co-je-to-wireframe-webu/

[35] Čermák, Miroslav. Bezpečnost webových aplikací a vícevrstvá. cleverandsmart.cz. [online]. 2013-05-09 [cit. 2016-02-09]. Dostupné z: http://cleverandsmart.cz/bezpecnost-webovych-aplikaci-a-vicevrstva-architektura/

[36] Čermák, Miroslav. Vícevrstvá architektura: popis vrstev. cleverandsmart.cz. [online]. 2010-04-05 [cit. 2016-02-09]. Dostupné z: http://cleverandsmart.cz/vicevrstva-architektura-popis-vrstev/

[37] Kučera, František. Java na webovém serveru: autorizace a autentizace. zdrojak.cz. [online]. 2010-02-26 [cit. 2016-02-13]. Dostupné z: https://zdrojak.cz/clanky/java-na-webovem-serveru-autorizace-a-autentizace/

[38] Počítač přesvědčivě napodobil člověka. Hrál si na puberťáka. Technet.cz. [online]. 2014-05-09 [cit. 2016-04-25]. Dostupné z: http://technet.idnes.cz/uspesny-turinguv-test-07g-/software.aspx?c=A140609_181110_software

[39] CAPTCHA: Telling Humans and Computers Apart Automatically. Captcha.net. [online]. ©2009- 2010 [cit. 2016-07-12]. Dostupné z: http://captcha.net

[40] Webová aplikace v Javě od A do Z – Maven plugin pro Eclipse a přidání Struts2 do projektu (2. díl). nullpointer.cz. [online]. 2012 [cit. 2016-07-12]. Dostupné z: http://nullpointer.cz/webova-aplikace-v-jave-od-a-do-z-maven-plugin-pro-eclipse-a-pridani-struts2-do-projektu-2-dil

[41] Verzovací systémy – svn, git, hg – svatá válka?. blog.frantovo.cz. [online]. 2015-06-29 [cit. 2016-07-13]. Dostupné z: https://blog.frantovo.cz/c/69/Verzovac%C3%AD%20syst%C3%A9my%20%E2%80%93%20svn%2C%20git%2C%20hg%20%E2%80%93%20svat%C3%A1%20v%C3%A1lka%3F

[42] SSH – bezpečné používání vzdáleného počítače a kopírování dat. Dsl.cz. [online]. ©2015 [cit. 2016-07-12]. Dostupné z: http://dsl.cz/jak-na-to/jak-na-ssh

[43] Roun Jiří. Tvorba softwarové dokumentace. swdokumentace.cz. [online]. 2016-05-03 [cit. 2016-07-12]. Dostupné z: http://swdokumentace.cz/index.html?sw_nabizene_typy_dokumentace.htm

Seznam příloh

Seznam obrázků v přílohách

Seznam tabulek v přílohách

A. Specifikace případů užití

Název	UC2 Vložit komentář
Stručný popis	Use case umožňuje komentovat požadavek a tím zasílá zprávu všem zúčastněným stranám.
Aktéři	Pozorovatel, řešitel, administrátor, neregistrovaný uživatel, zadavatel
Základní tok	1. Uživatel inicializuje vložení komentáře. 2. Systém zobrazí formulář pro vytvoření komentáře. 3. Uživatel vyplní formulářová data a formulář odešle. 4. Systém zkontroluje data od uživatele. 5. Systém uloží požadavek a odešle notifikační email účastníkům požadavku.
Alternativní tok 1	5a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí požadavek vytvořit. 5b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 1: UC2 Vložit komentář

Název	UC3 Zobrazit detailní informace o požadavku
Stručný popis	Use case umožňuje zobrazit detail požadavku.
Aktéři	Pozorovatel, řešitel, administrátor, neregistrovaný uživatel, zadavatel
Základní tok	1. Uživatel inicializuje zobrazení detailu požadavku. 2. Systém zobrazí detail požadavku.
Alternativní tok 1	1a. Nepřihlášený uživatel inicializuje zobrazení detailu požadavku z odkazu v emailu.

Tabulka příloha 2: UC3 Zobrazení detailní informace o požadavku

Název	UC4 Vyhledat požadavky dle filtračních pravidel
Stručný popis	Use case umoňuje zobrazit seznam požadavků a v něm filtrovat podle filtračních pravidel.
Aktéři	Pozorovatel, řešitel, administrátor, zadavatel
Základní tok	1. Uživatel inicializuje zobrazení seznamu požadavků. 2. Systém zobrazí seznam požadavků a formulář pro filtraci, ke kterým má uživatel přístup. 3. Uživatel vyplní filtrační podmínky a potvrdí. 4. Systém filtrační podmínky zpracuje a provede vyhledání.
Alternativní tok 1	3a. Uživatel klikne na tlačítko rozšířit filtr. 3b. Systém zobrazí panel s filtračními údaji. 3c. Uživatel vybere, jaká filtrační pravidla chce přidat, a potvrdí. 3d. Systém data zpracuje a zavře panel.

Tabulka příloha 3: UC4 Vyhledat požadavky dle filtračních pravidel

Název	UC5 Exportovat seznam do formátu PDF a XLS
Stručný popis	Use case umožňuje seznam požadavků vyexportovat do formátu PDF a XLS.
Aktéři	Pozorovatel, řešitel, administrátor, zadavatel
Základní tok	1. Uživatel inicializuje export seznamu požadavků do PDF a XLS. 2. Uživatel klikne na tlačítko vytvořit PDF, nebo XLS. 3. Systém vygeneruje data. 4. Systém zobrazí okno pro stažení dokumentu. 5. Uživatel potvrdí a dokument stáhne.

Tabulka příloha 4: UC5 Exportovat seznam do formátu PDF a XLS

Název	UC6 Zpracovat požadavek
Stručný popis	Use case umožňuje uživateli řešit požadavek.
Aktéři	Řešitel, administrátor
Základní tok	1. Uživatel inicializuje řešení požadavku. 2. Systém zobrazí seznam požadavků. 3. Systém spustí UC Zobrazit detailní informace o požadavku. 4. Uživatel převezme požadavek. 5. Systém zobrazí formulář pro převzetí požadavku. 6. Uživatel vyplní formulářová data a formulář odešle. 7. Systém zkontroluje data od uživatele. 8. Systém uloží požadavek a odešle notifikační email účastníkům požadavku.
Alternativní tok 1	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí požadavek upravit. 4b. Uživatel opraví neplatná data a pokračuje bodem 6.

Tabulka příloha 5: UC6 Zpracovat požadavek

Název	UC7 Vytvářet reporty do formátu PDF a XLS
Stručný popis	Use case umožňuje uživateli vytvářet reporty ze zadaných podmínek a exportovat je do PDF a XLS.
Aktéři	Administrátor
Základní tok	1. Uživatel inicializuje vytvoření reportu. 2. Systém zobrazí seznam požadavků nebo aplikačních logů. 3. Uživatel vyplní filtrační podmínky a klikne na tlačítko pro exportování do zvoleného formátu. 4. Systém zkontroluje a vygeneruje data. 5. Systém zobrazí okno pro stažení dokumentu. 6. Uživatel potvrdí a dokument stáhne.
Alternativní tok 1	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí report vytvořit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 6: UC7 Vytvářet reporty do formátu PDF a XLS

Název	UC8 Spravovat uživatele
Krátký popis	Use Case umožňuje uživateli vykonávat CRUD operace nad uživateli.
Aktéři	Administrátor
Základní tok	**Vytvoření** 1. Uživatel inicializuje vytvoření uživatele. 2. Systém zobrazí prázdný formulář pro vytvoření uživatele. 3. Uživatel vyplní povinná pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží uživatele. 6. Systém vygeneruje přihlašovací heslo a odešle ho na email. **Čtení** 1. Uživatel inicializuje zobrazení detailu požadavku. 2. Systém zobrazí detail uživatele. **Upravení** 1. Uživatel inicializuje upravení uživatele. 2. Systém zobrazí formulář pro upravení uživatele. 3. Uživatel upraví pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží uživatele. **Smazání** 1. Uživatel inicializuje smazání uživatele. 2. Systém zobrazí formulář pro smazání uživatele. 3. Uživatel klikne na tlačítko smazat. 4. Systém smaže uživatele.
Alternativní tok 1-Vytvoření	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí uživatele vytvořit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.
Alternativní tok 1-Upravení	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí uživatele upravit.

	4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 7: UC8 Spravovat uživatele

Název	UC9 Spravovat zadavatele
Krátký popis	Use Case umožňuje uživateli vykonávat CRUD operace nad zadavateli.
Aktéři	Administrátor
Základní tok	**Vytvoření** 1. Uživatel inicializuje vytvoření zadavatele. 2. Systém zobrazí prázdný formulář pro vytvoření zadavatele. 3. Uživatel vyplní povinná pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží zadavatele. **Čtení** 1. Uživatel inicializuje zobrazení detailu zadavatele. 2. Systém zobrazí detail zadavatele. **Upravení** 1. Uživatel inicializuje upravení zadavatele. 2. Systém zobrazí formulář pro upravení zadavatele. 3. Uživatel upraví pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží zadavatele. **Smazání** 1. Uživatel inicializuje smazání zadavatele. 2. Systém zobrazí formulář pro smazání zadavatele. 3. Uživatel klikne na tlačítko smazat. 4. Systém smaže zadavatele.
Alternativní tok 1-Vytvoření	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí zadavatele vytvořit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Alternativní tok 1-Upravení	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí zadavatele upravit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 8: UC9 Spravovat zadavatele

Název	UC10 Spravovat SLA
Krátký popis	Use Case umožňuje uživateli vykonávat CRUD operace nad SLA.
Aktéři	Administrátor
Základní tok	**Vytvoření** 1. Uživatel inicializuje vytvoření SLA. 2. Systém zobrazí prázdný formulář pro vytvoření SLA. 3. Uživatel vyplní povinná pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží SLA. **Čtení** 1. Uživatel inicializuje zobrazení detailu SLA. 2. Systém zobrazí detail SLA. **Upravení** 1. Uživatel inicializuje upravení SLA. 2. Systém zobrazí formulář pro upravení SLA. 3. Uživatel upraví pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží SLA. **Smazání** 1. Uživatel inicializuje smazání SLA. 2. Systém zobrazí formulář pro smazání SLA. 3. Uživatel klikne na tlačítko smazat. 4. Systém smaže SLA.
Alternativní tok 1-Vytvoření	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí SLA vytvořit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Alternativní tok 1-Upravení	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí SLA upravit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 9: UC10 Spravovat SLA

Název	UC11 Spravovat produkty
Krátký popis	Use Case umožňuje uživateli vykonávat CRUD operace nad produkty.
Aktéři	Administrátor
Základní tok	**Vytvoření** 1. Uživatel inicializuje vytvoření produktu. 2. Systém zobrazí prázdný formulář pro vytvoření produktu. 3. Uživatel vyplní povinná pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží produktu. **Čtení** 1. Uživatel inicializuje zobrazení detailu produktu. 2. Systém zobrazí detail produktu. **Upravení** 1. Uživatel inicializuje upravení produktu. 2. Systém zobrazí formulář pro upravení produktu. 3. Uživatel upraví pole a odešle formulář. 4. Systém zkontroluje data od uživatele. 5. Systém uloží produkt. **Smazání** 1. Uživatel inicializuje smazání produktu. 2. Systém zobrazí formulář pro smazání produktu. 3. Uživatel klikne na tlačítko smazat. 4. Systém smaže produkt.
Alternativní tok 1-Vytvoření	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí produkt vytvořit.

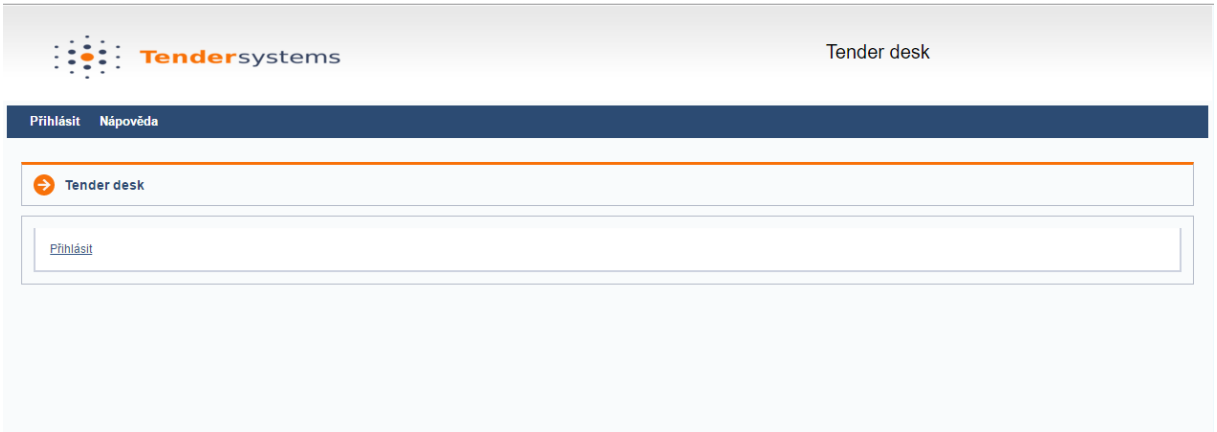
	4b. Uživatel opraví neplatná data a pokračuje bodem 3.
Alternativní tok 1-Upravení	4a. Pokud uživatel zadal neplatná data, systém upozorní uživatele a nedovolí produkt upravit. 4b. Uživatel opraví neplatná data a pokračuje bodem 3.

Tabulka příloha 10: UC11 Spravovat produkty

Název	UC12 Archivovat požadavky
Krátký popis	Use Case umožňuje uživateli archivovat ukončené požadavky.
Aktéři	Administrátor
Základní tok	1. Uživatel inicializuje archivování požadavků. 2. Systém vyhledá požadavky, které jsou už ukončeny. 3. Systém nalezené požadavky archivuje.

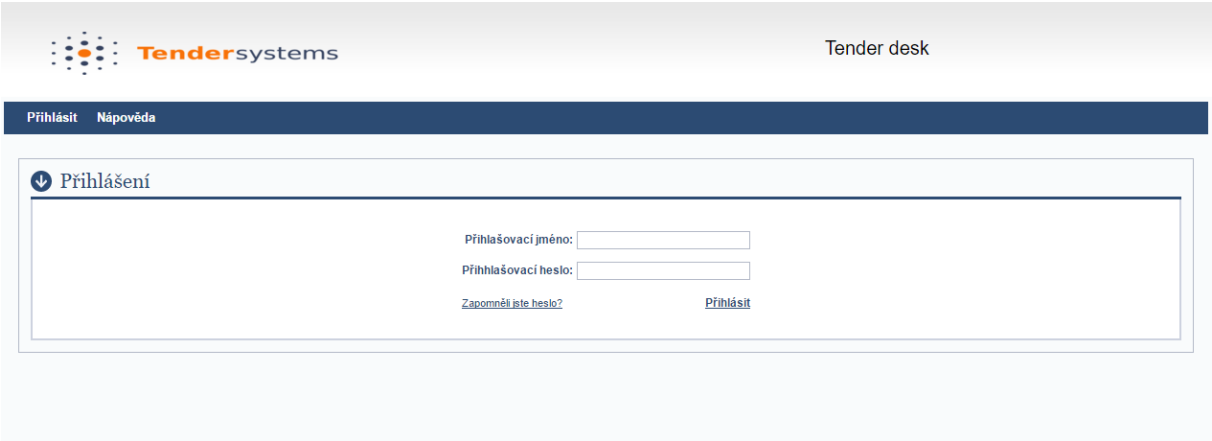
Tabulka příloha 11: UC12 Archivovat požadavky

B. Snímky uživatelského prostředí

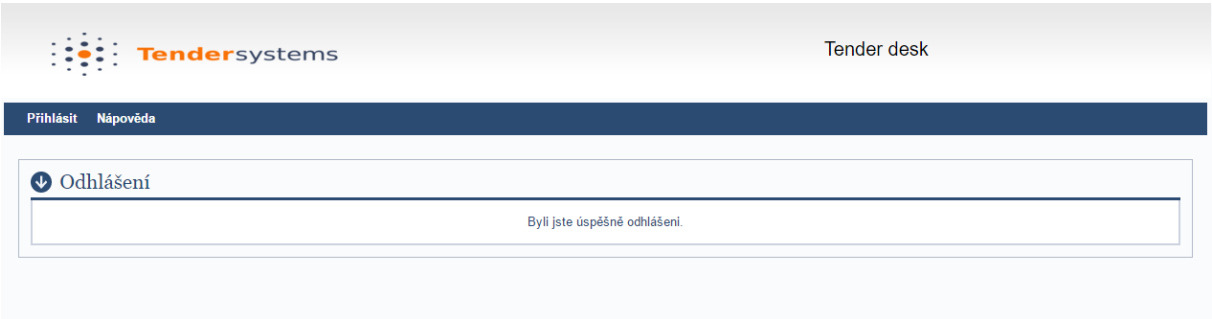


Obrázek příloha 1: Úvodní stránka [autor]

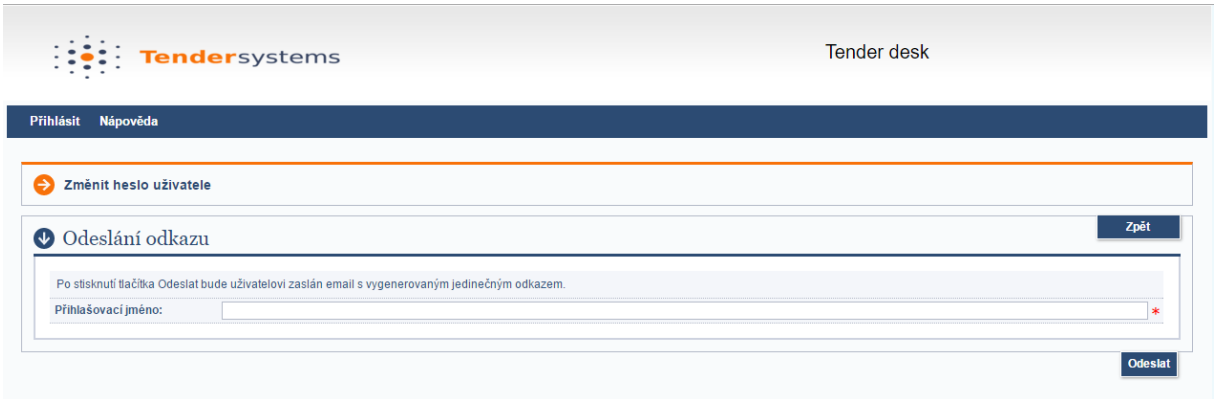
Obrázek příloha 2: Patička [autor]



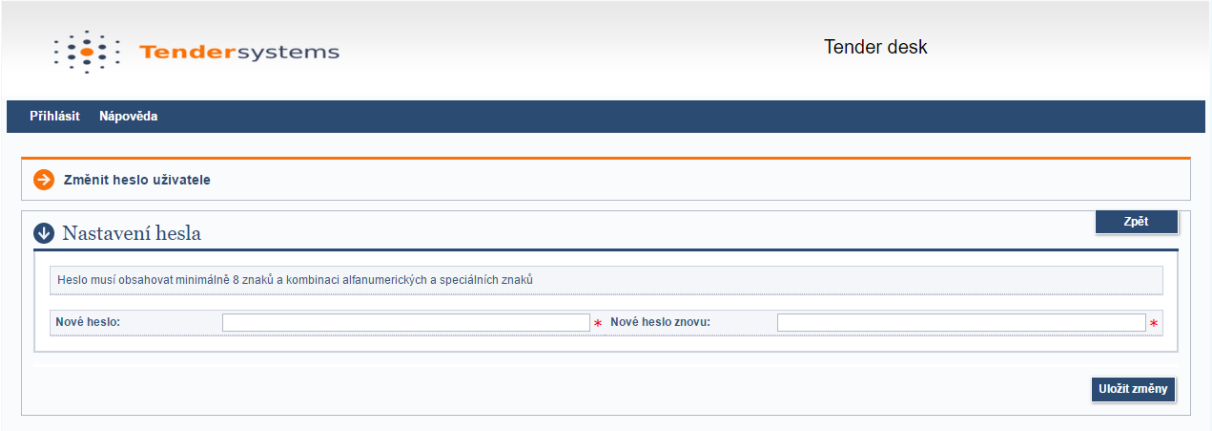
Obrázek příloha 3: Stránka pro přihlášení [autor]



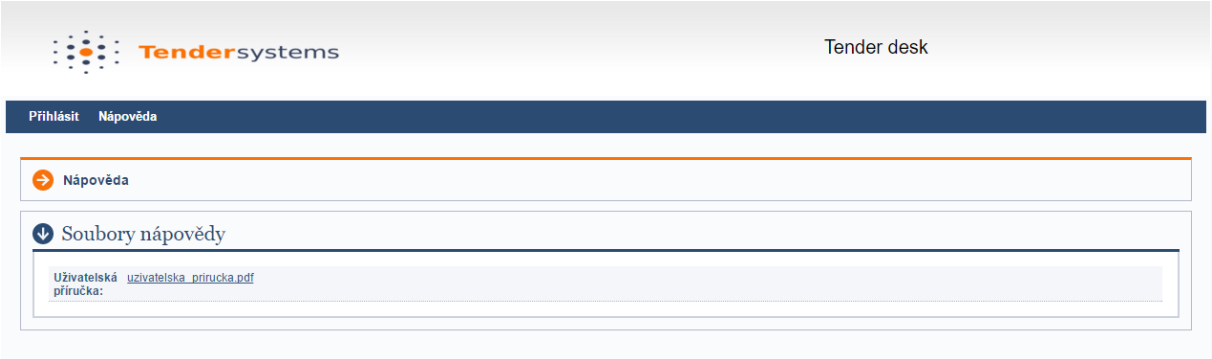
Obrázek příloha 4: Stránka pro odhlášení [autor]



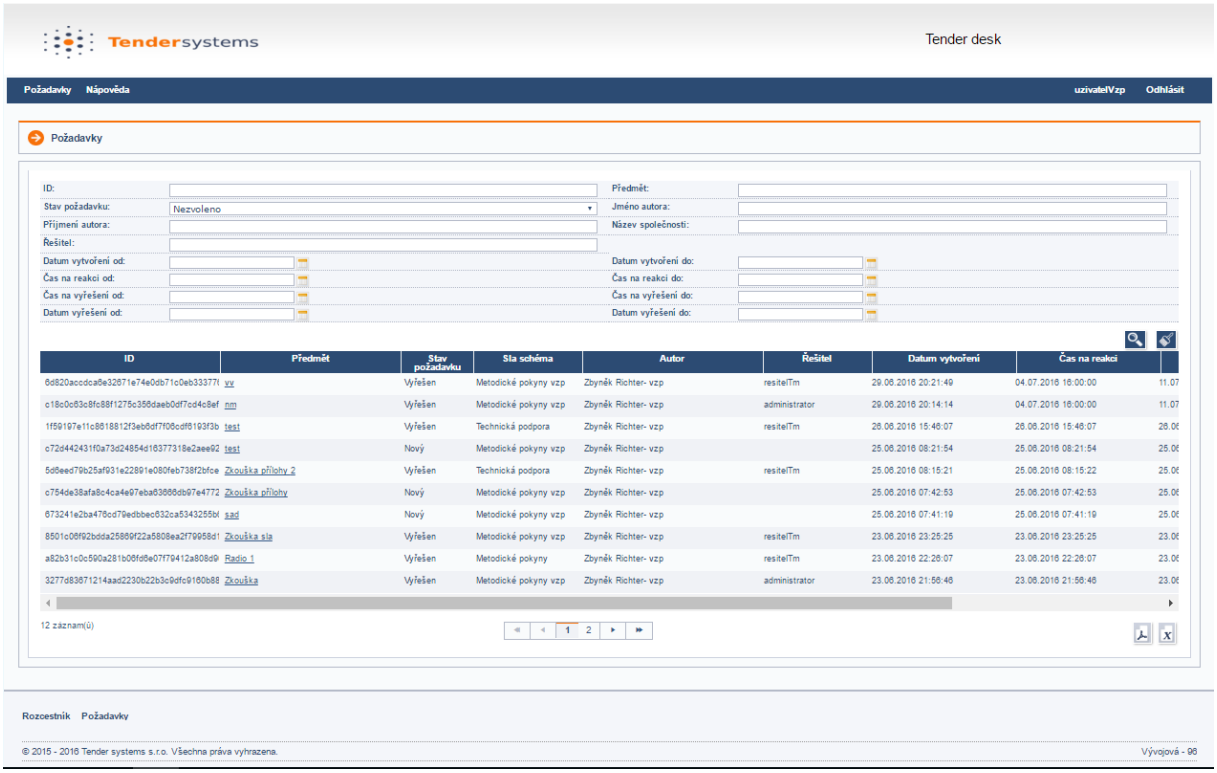
Obrázek příloha 5: Stránka pro změnu hesla [autor]



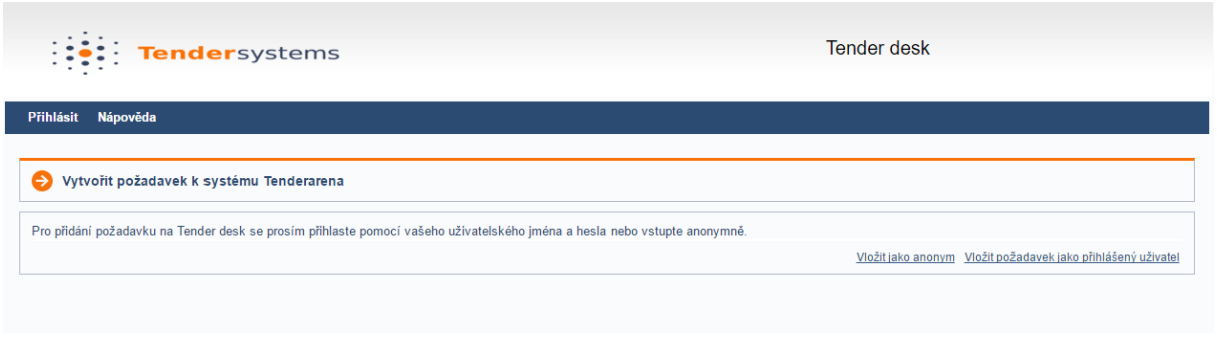
Obrázek příloha 6: Stránka pro nastavení hesla [autor]



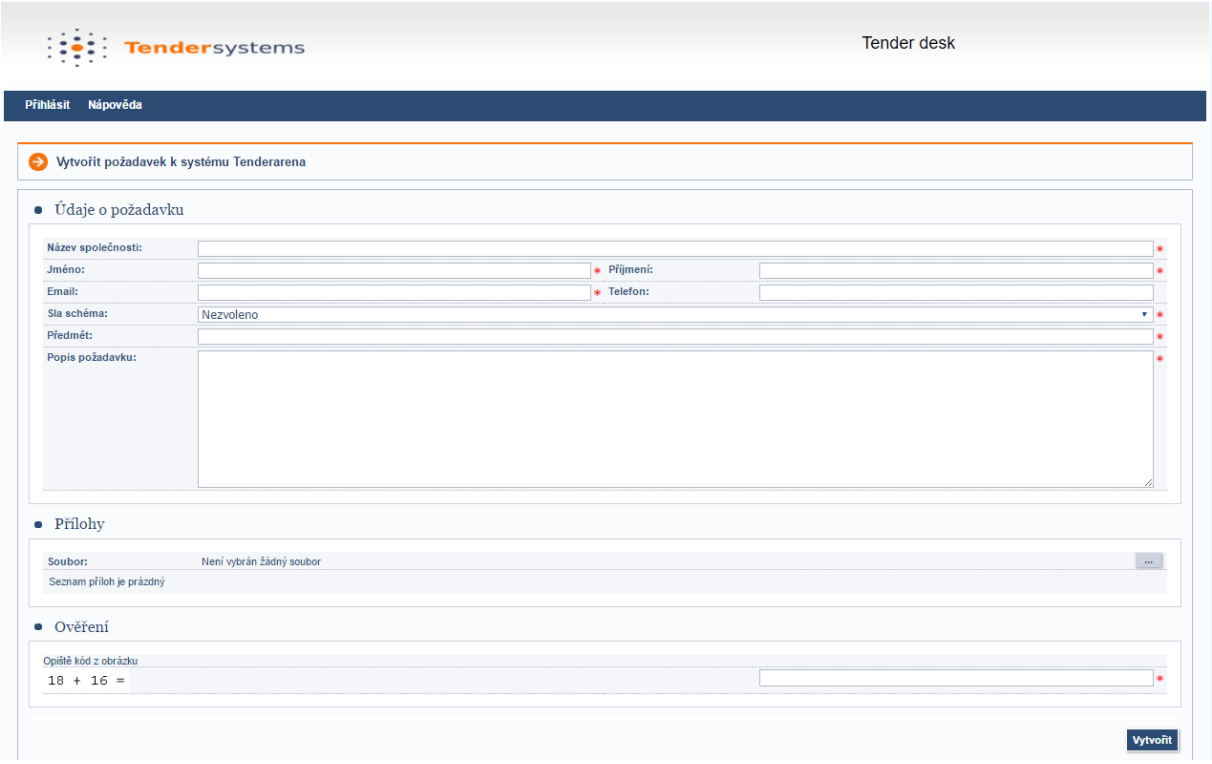
Obrázek příloha 7: Stránka s nápovědou [autor]



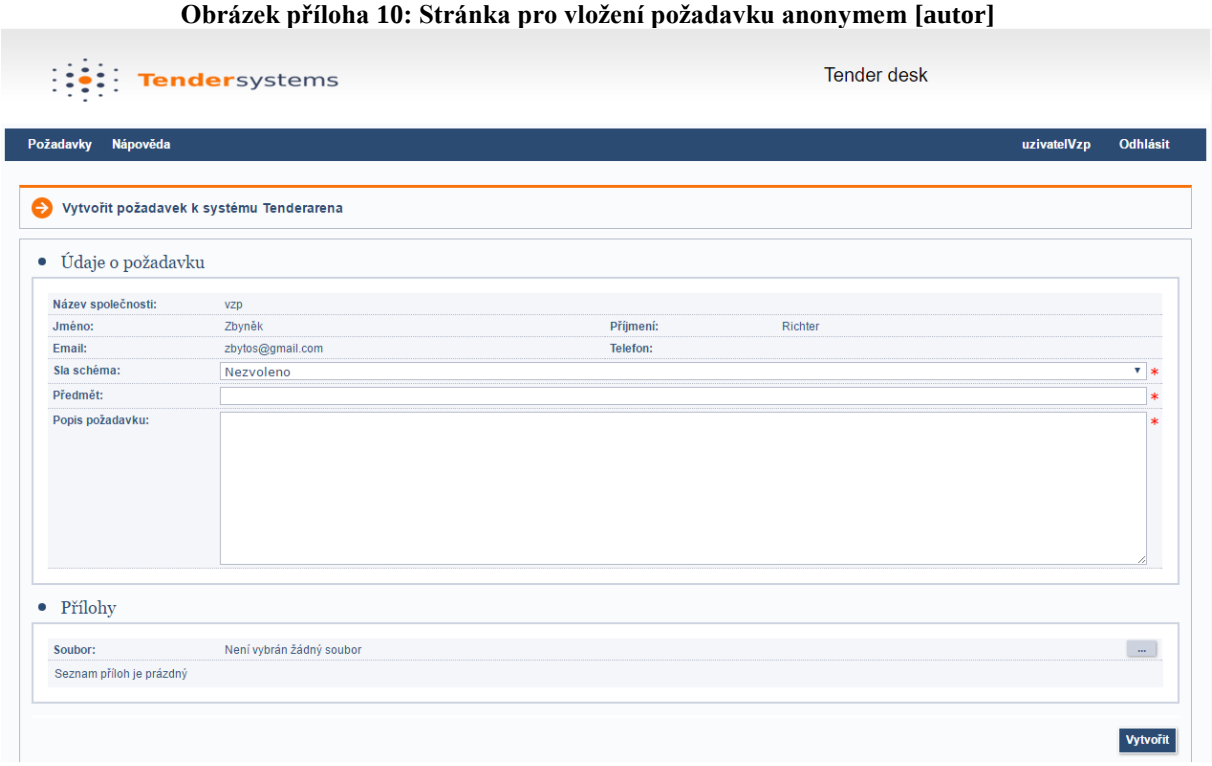
Obrázek příloha 8: Stránka se seznamem požadavků [autor]



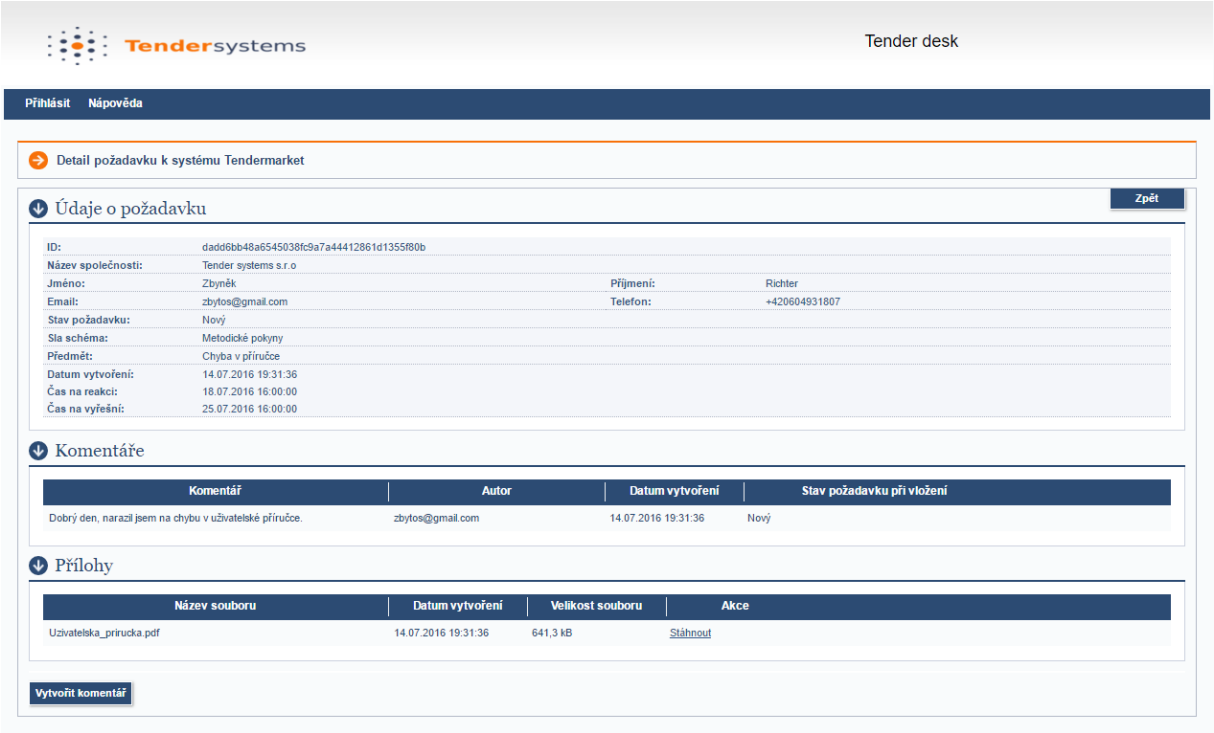
Obrázek příloha 9: Stránka pro vytvoření požadavku [autor]



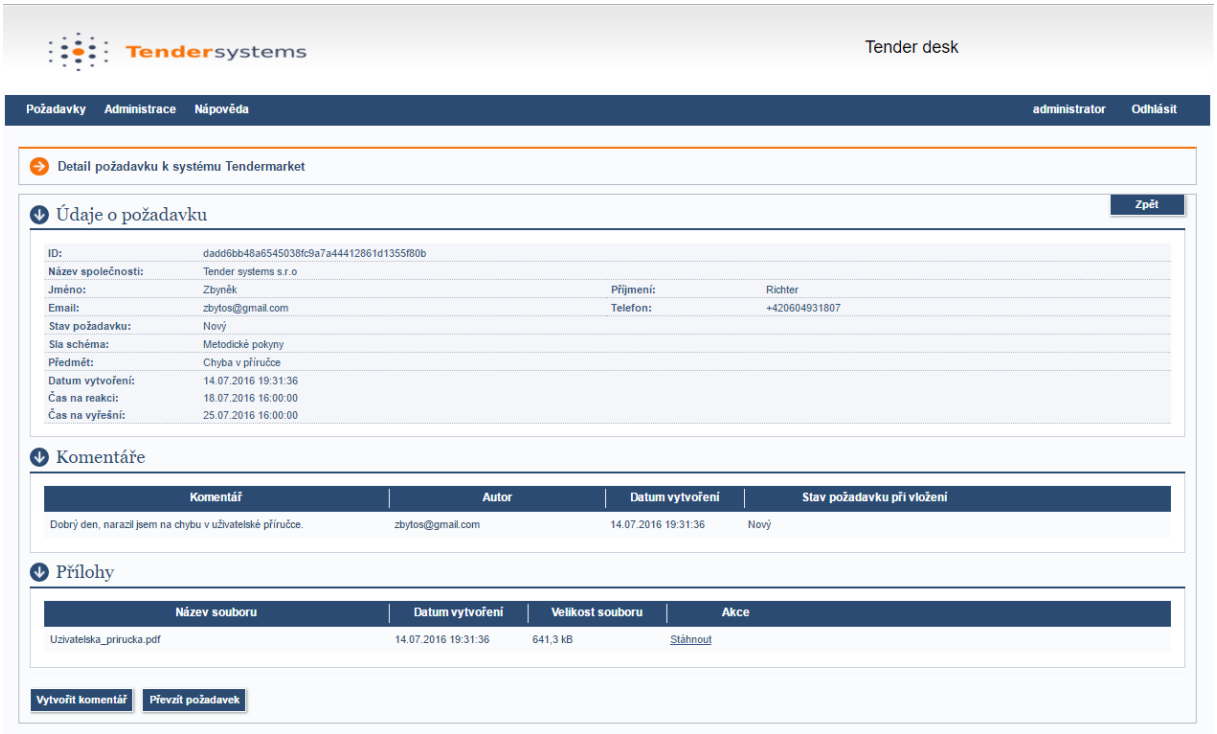
Obrázek příloha 10: Stránka pro vložení požadavku anonymem [autor]



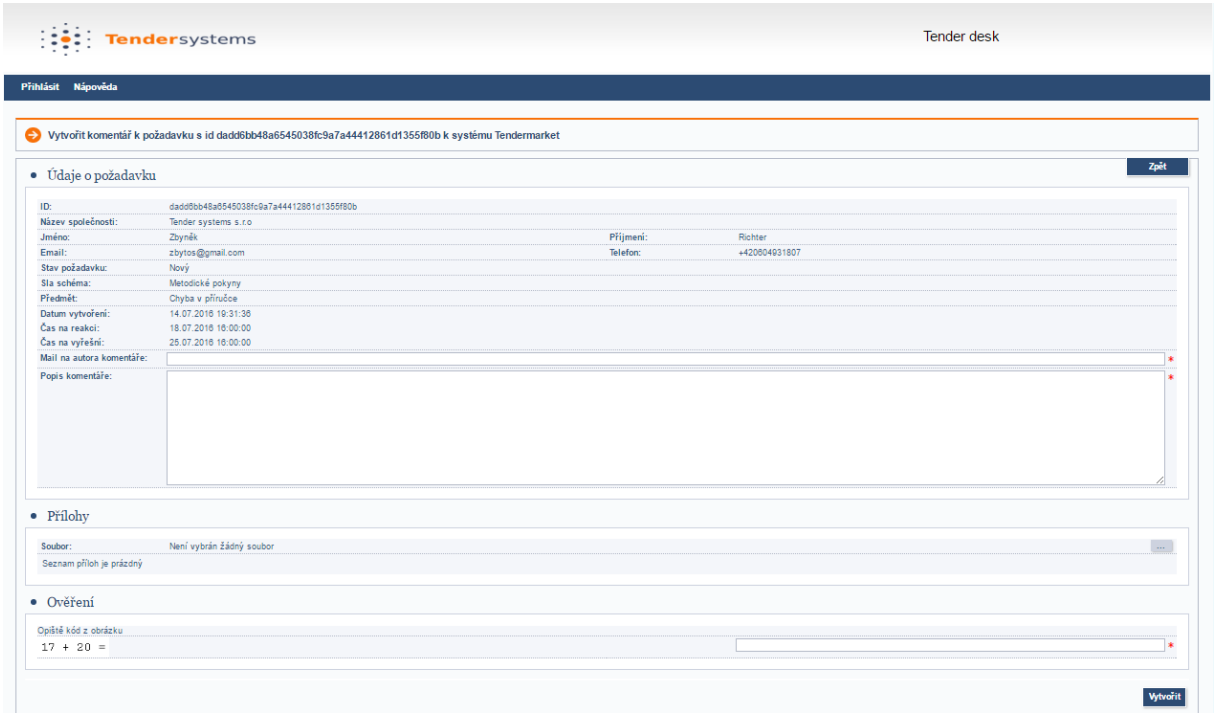
Obrázek příloha 11: Stránka pro vložení požadavku přihlášeným uživatelem [autor]



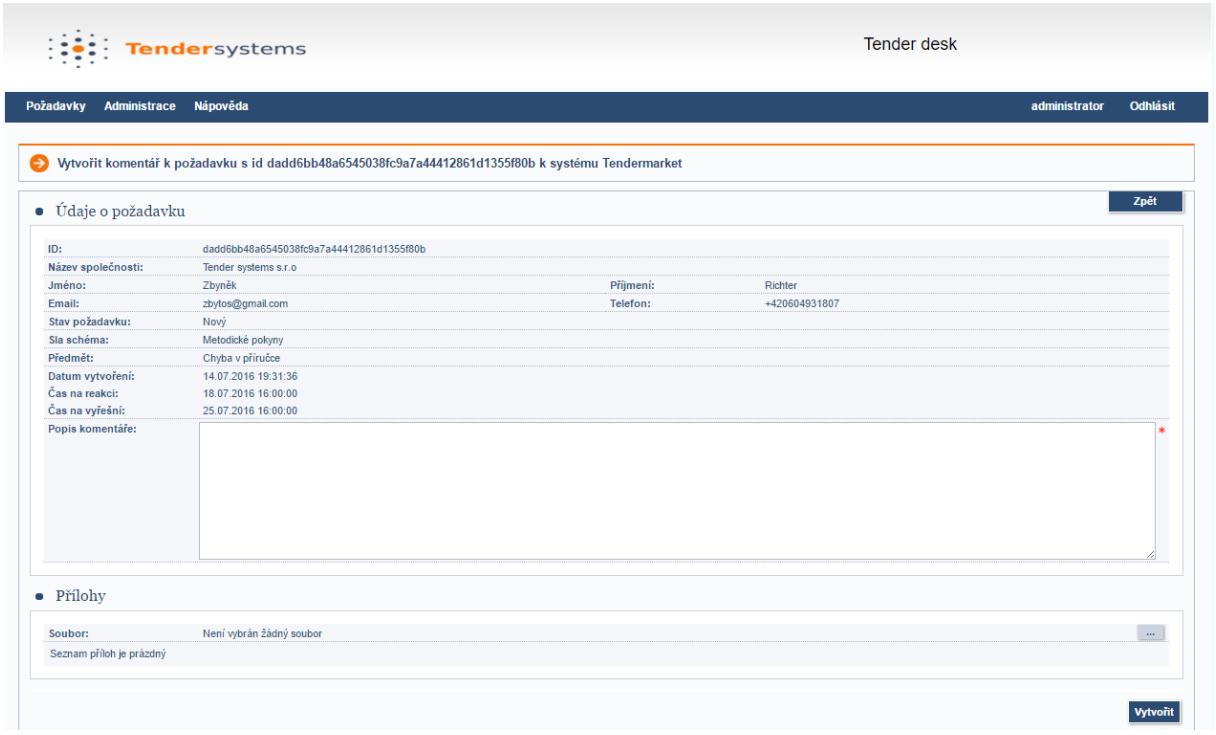
Obrázek příloha 12: Stránka s detailem požadavku uživatele bez role pro řešení [autor]



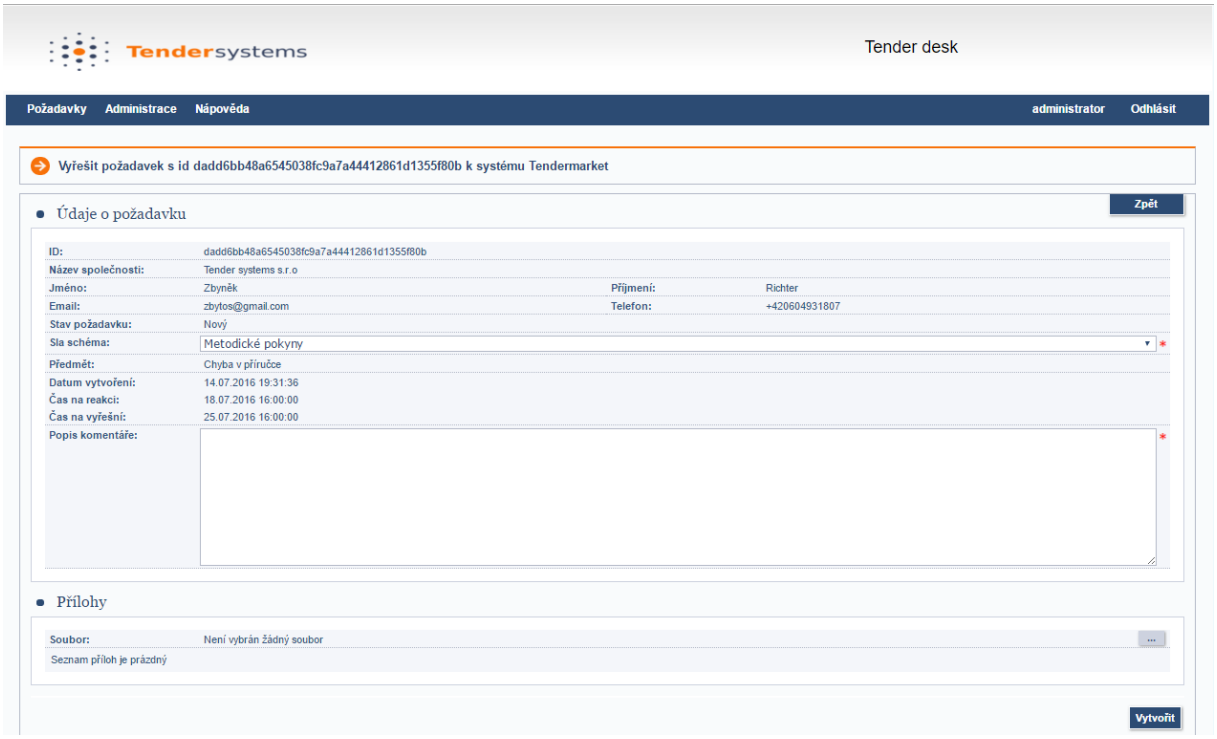
Obrázek příloha 13: Stránka s detailem požadavku uživatele bez role pro řešení [autor]



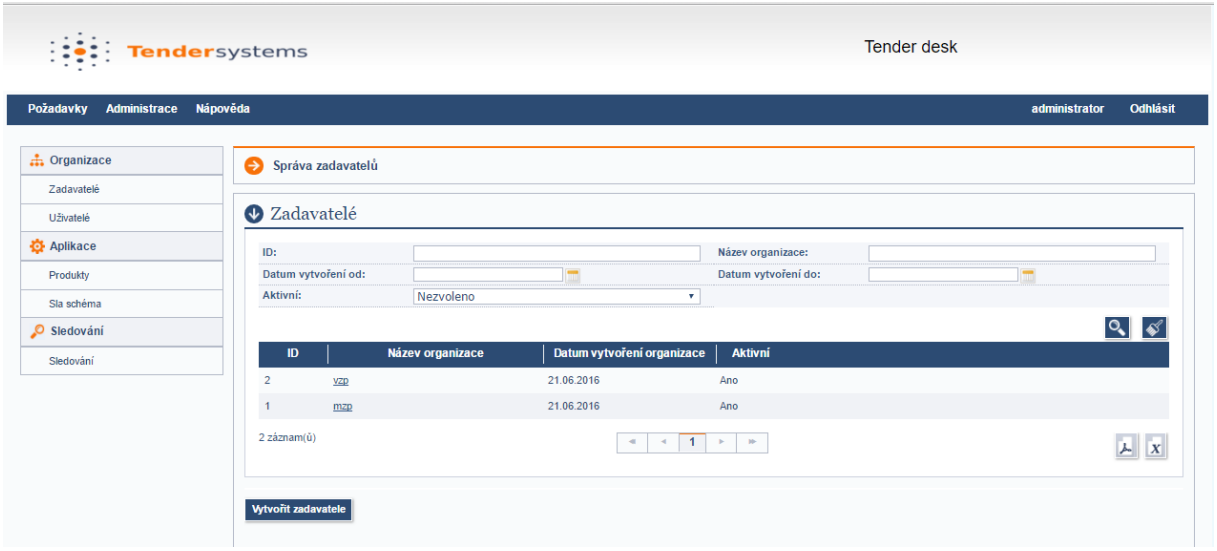
Obrázek příloha 14: Stránka pro vytvoření komentáře nepřihlášeným uživatelem [autor]



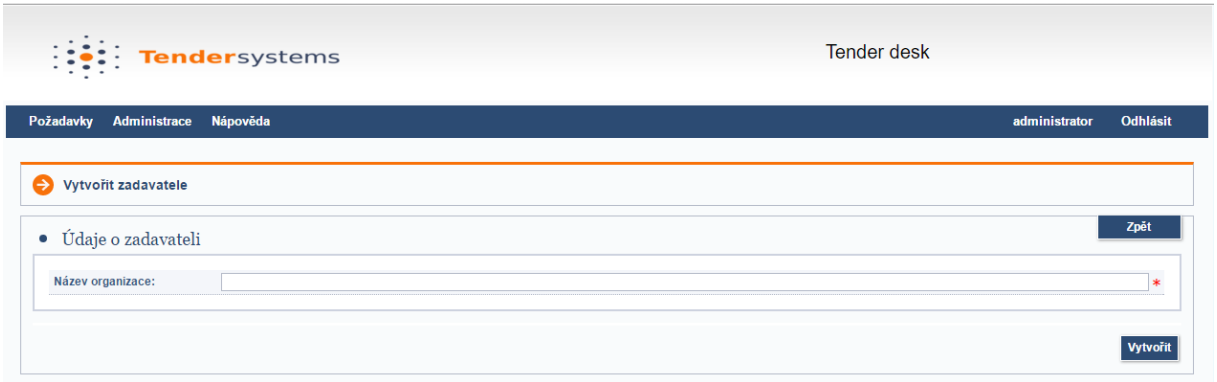
Obrázek příloha 15: Stránka pro vytvoření komentáře přihlášeným uživatelem [autor]



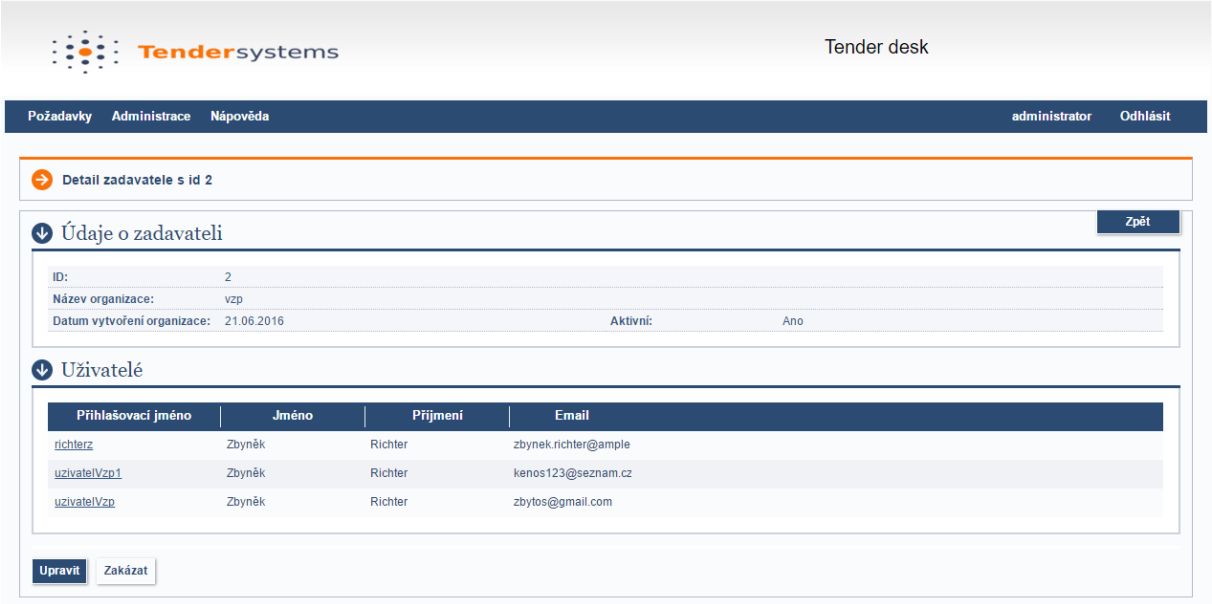
Obrázek příloha 16: Stránka pro řešení požadavku [autor]



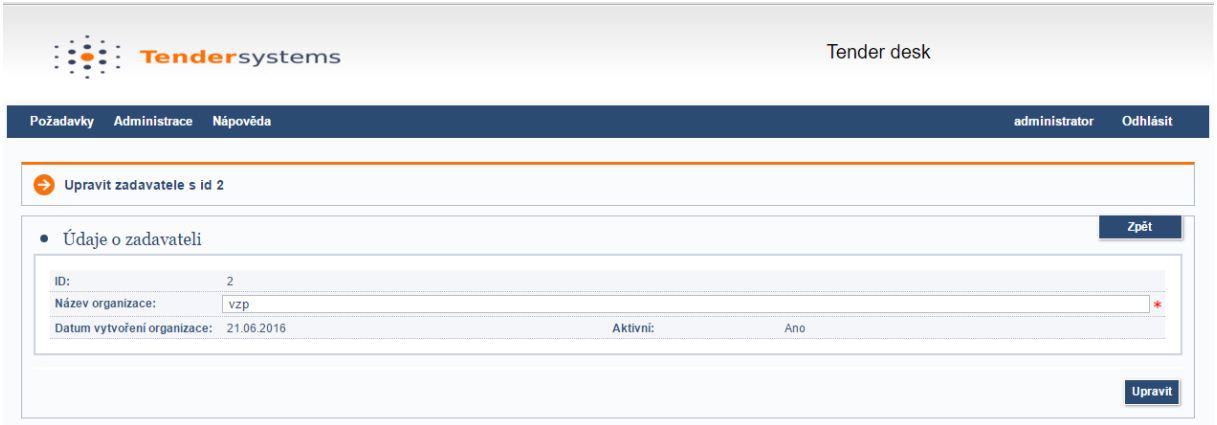
Obrázek příloha 17: Stránka se seznamem zadavatelů [autor]



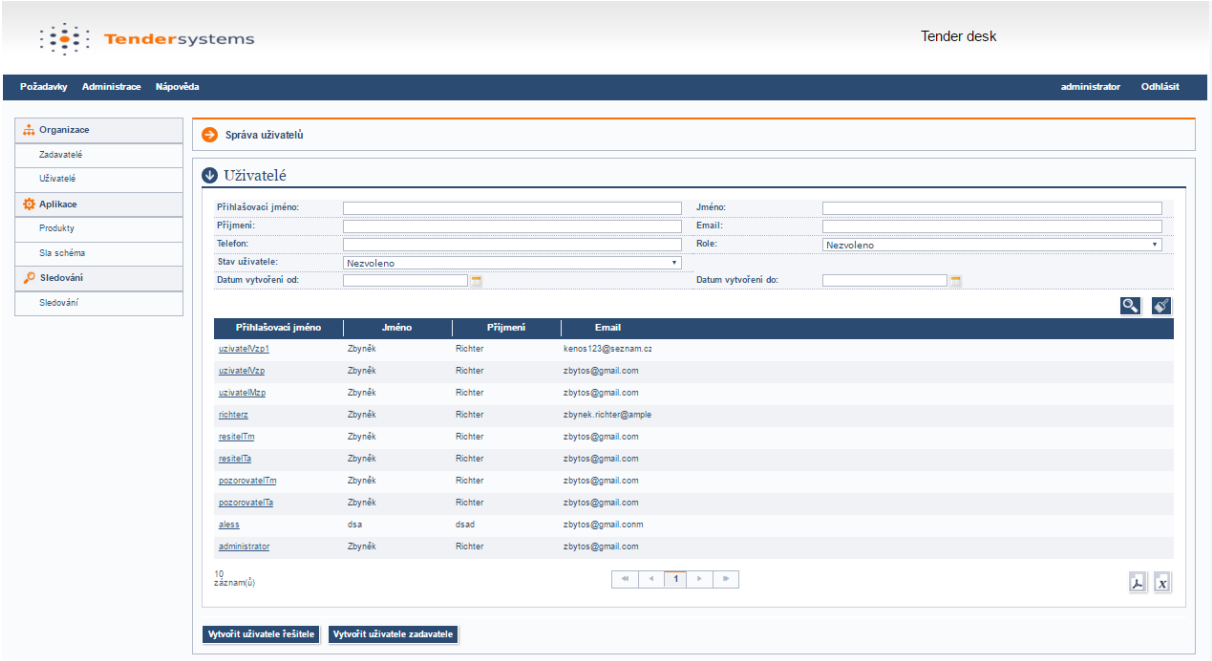
Obrázek příloha 18: Stránka pro vytvoření zadavatele [autor]



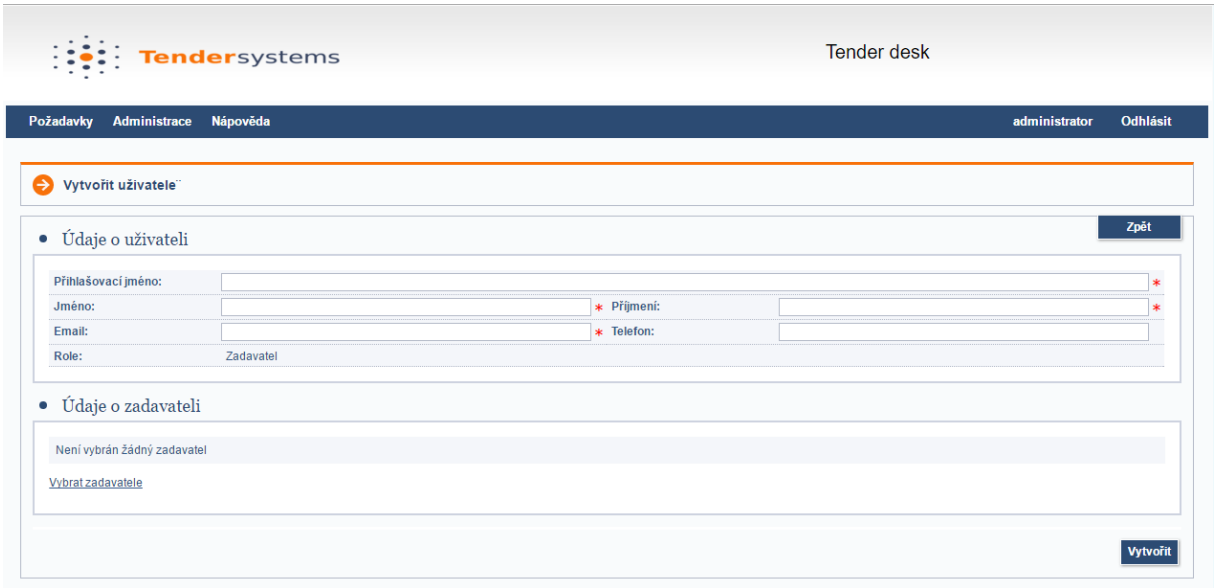
Obrázek příloha 19: Stránka detailu zadavatele [autor]



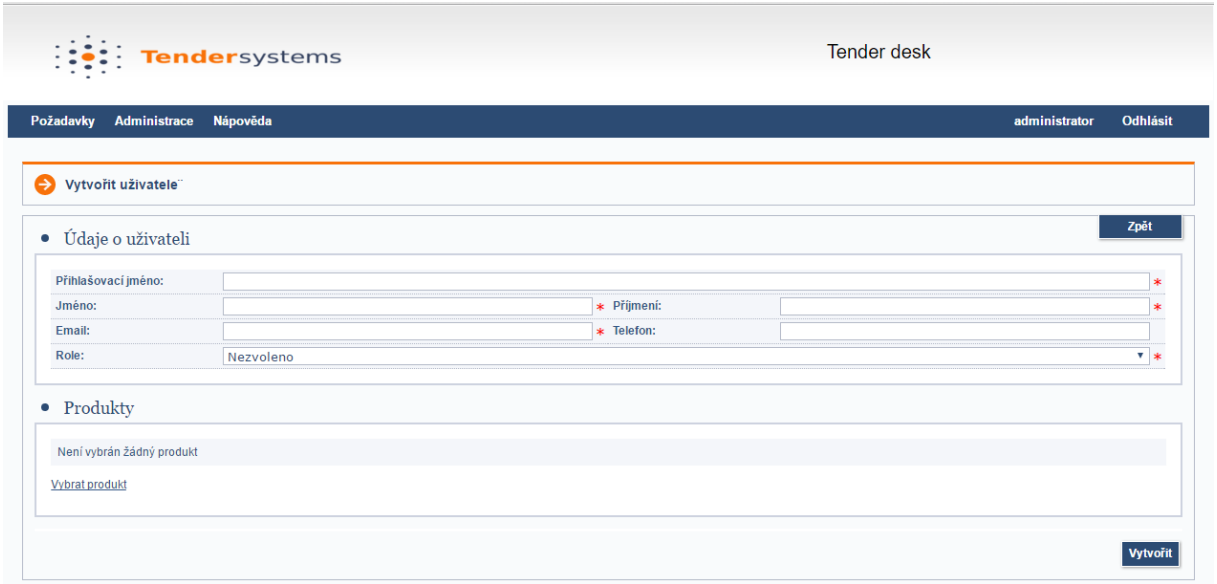
Obrázek příloha 20: Stránka pro upravení zadavatele [autor]



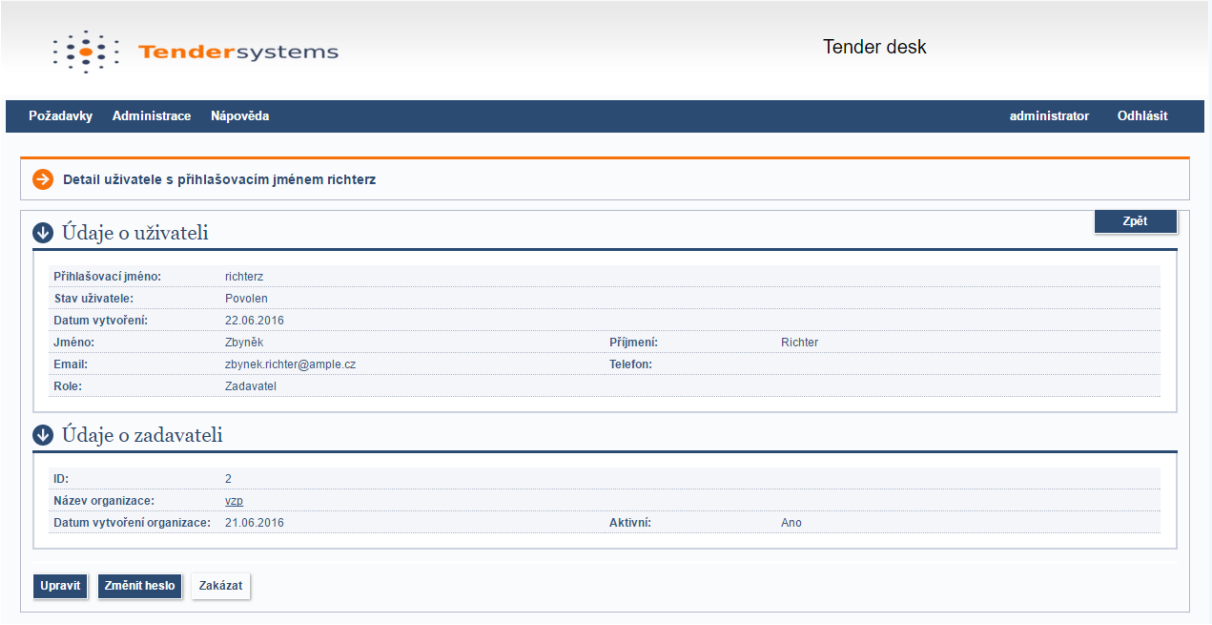
Obrázek příloha 21: Stránka se seznamem uživatelů [autor]



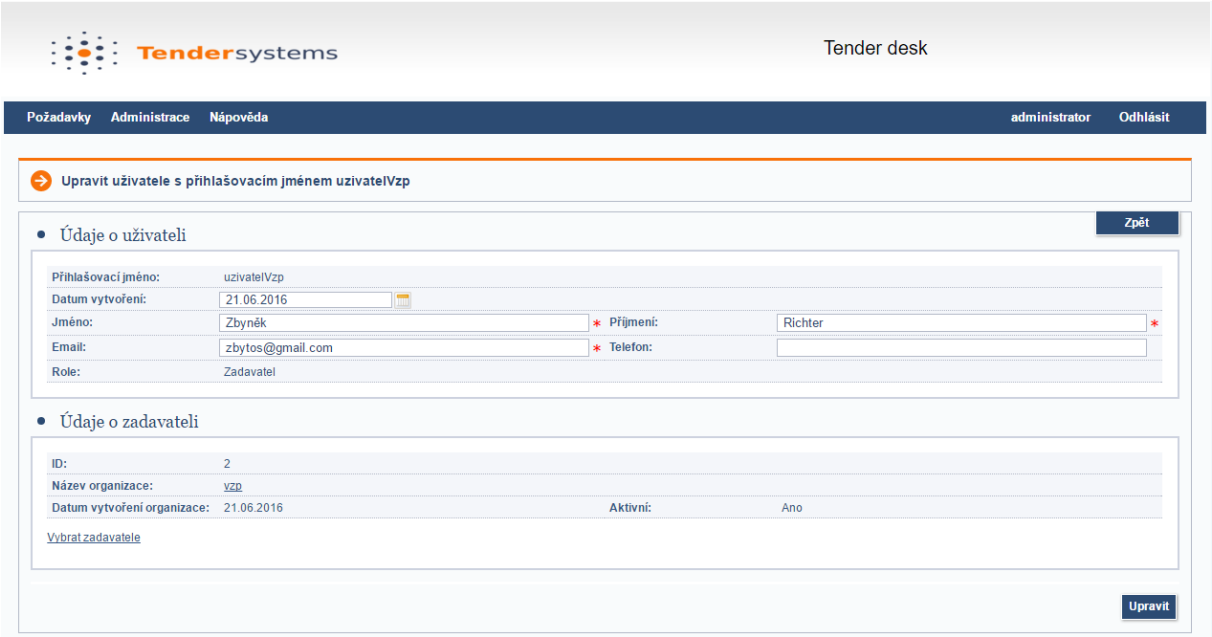
Obrázek příloha 22: Stránka pro vytvoření uživatele zadavatele [autor]



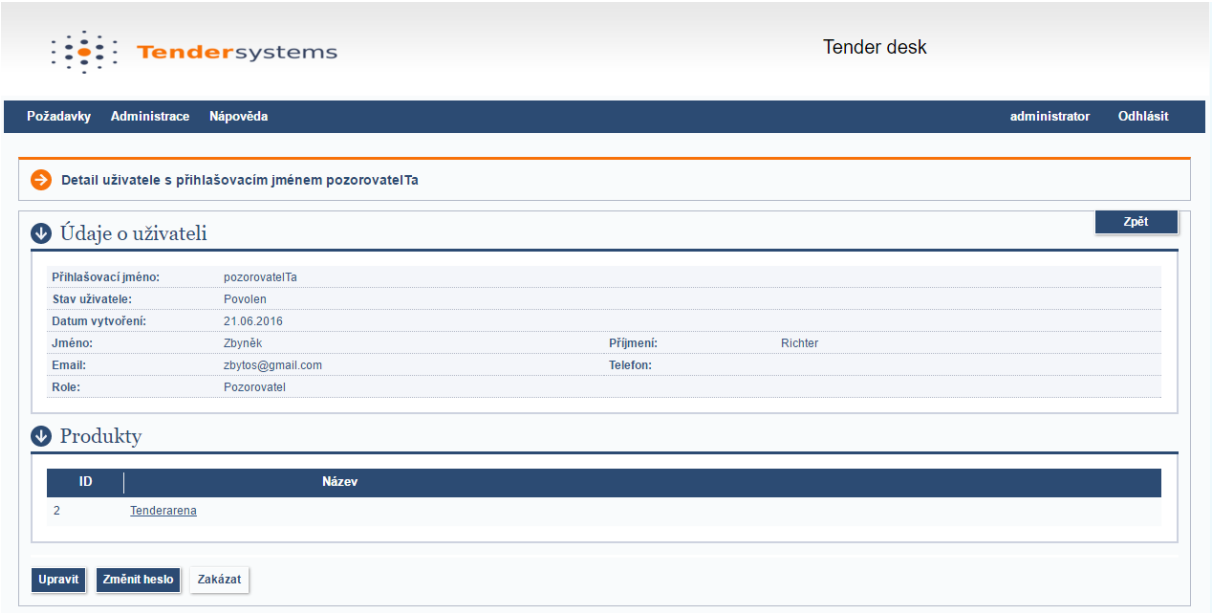
Obrázek příloha 23: Stránka pro vytvoření řešitele [autor]



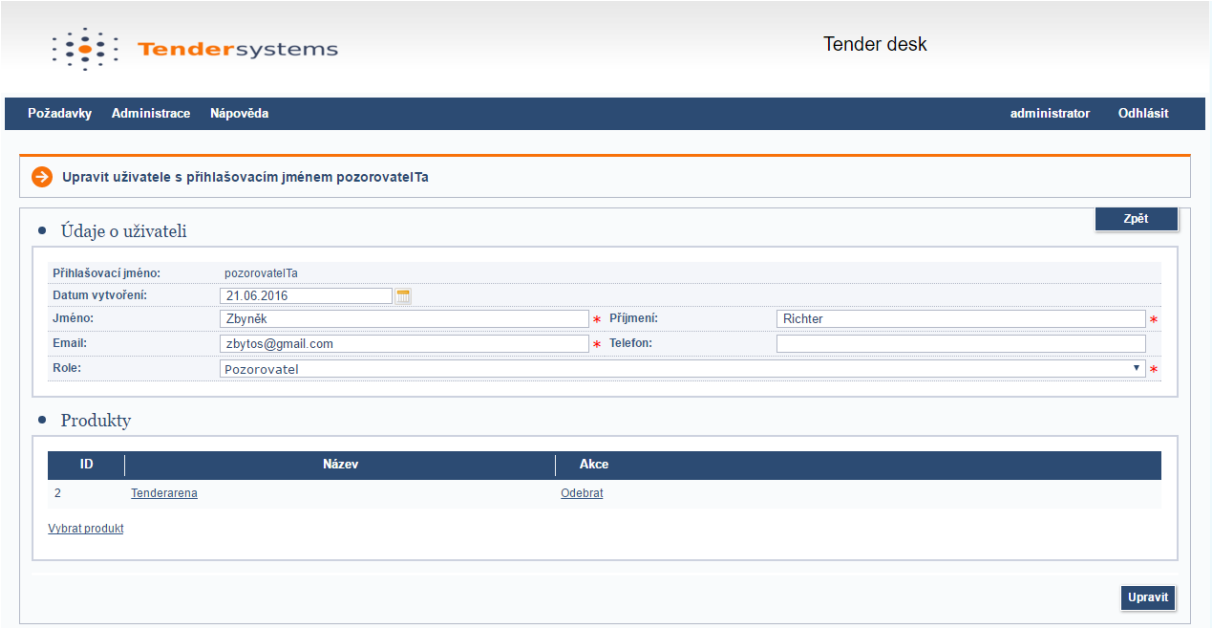
Obrázek příloha 24: Stránka detailu uživatele zadavatele [autor]



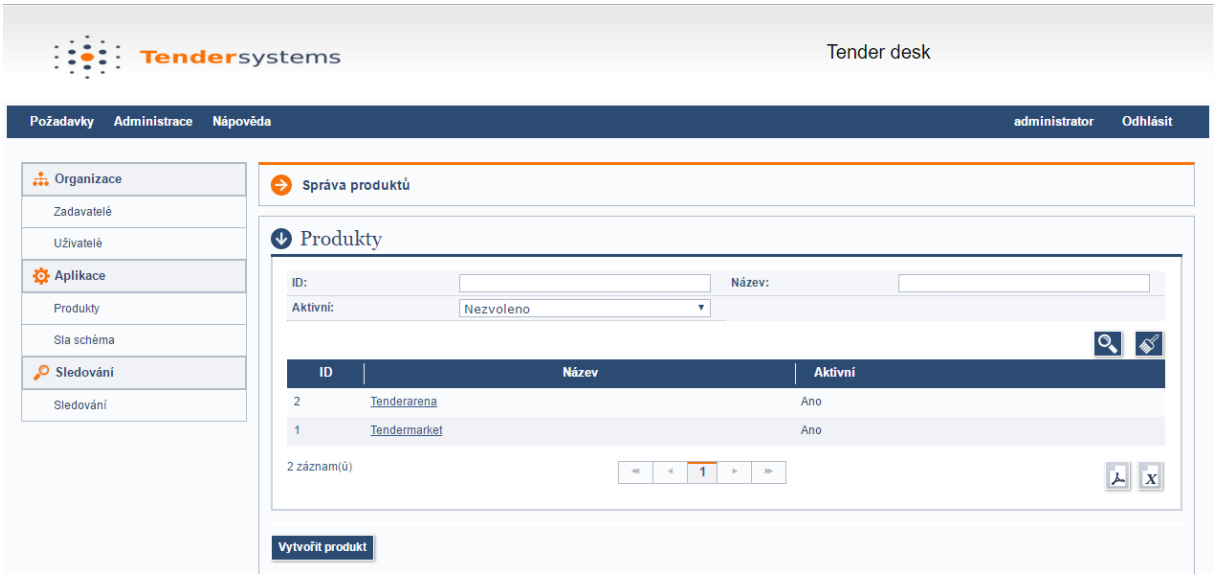
Obrázek příloha 25: Stránka detailu řešitele [autor]



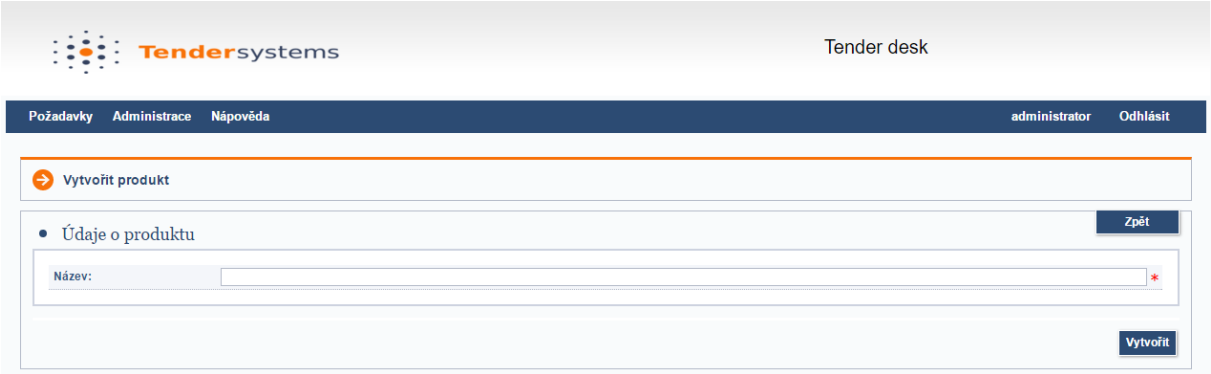
Obrázek příloha 26: Stránka pro upravení uživatele zadavatele [autor]



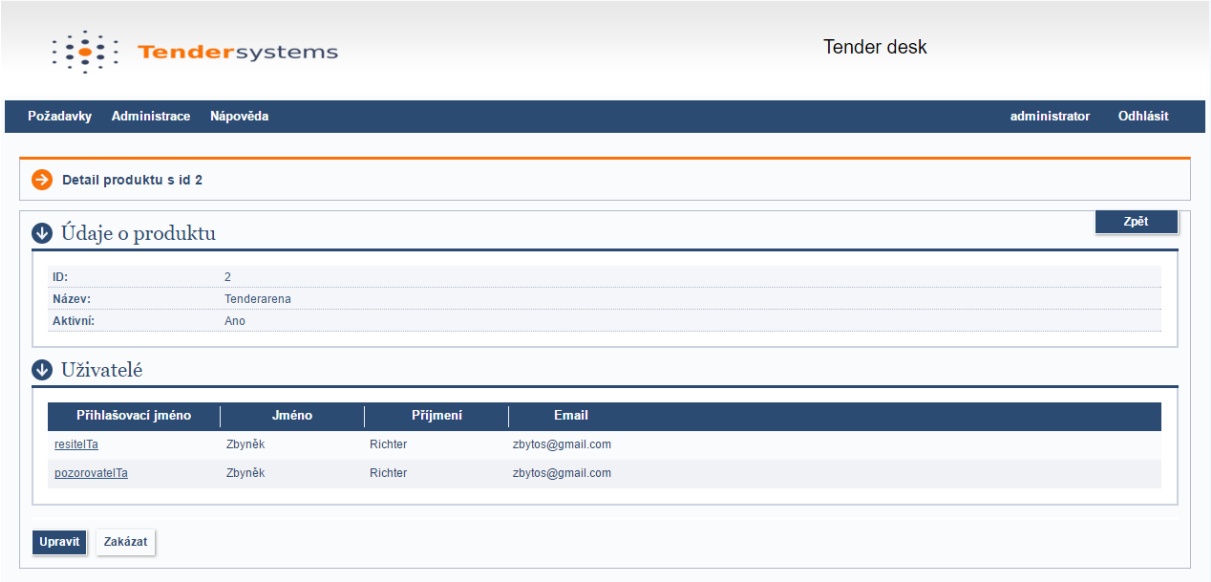
Obrázek příloha 27: Stránka pro upravení řešitele [autor]



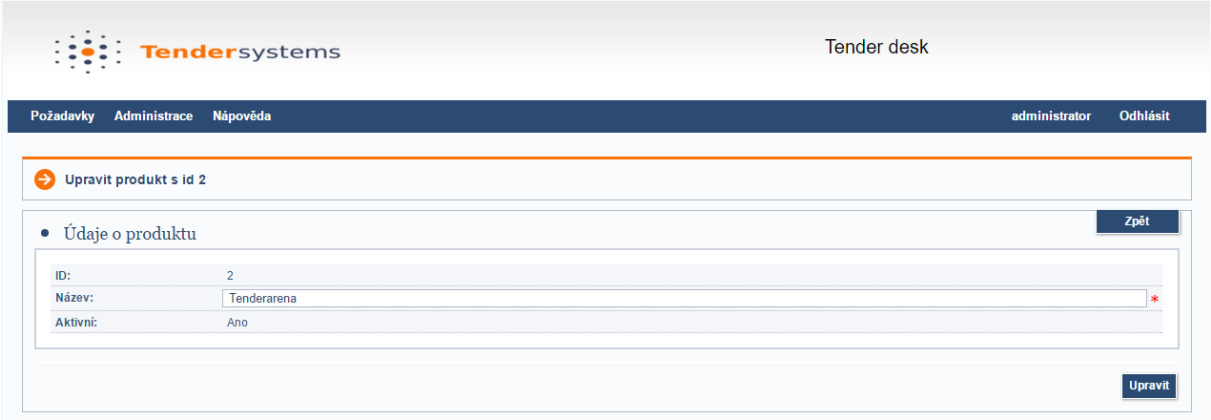
Obrázek příloha 28: Stránka se seznamem produktů [autor]



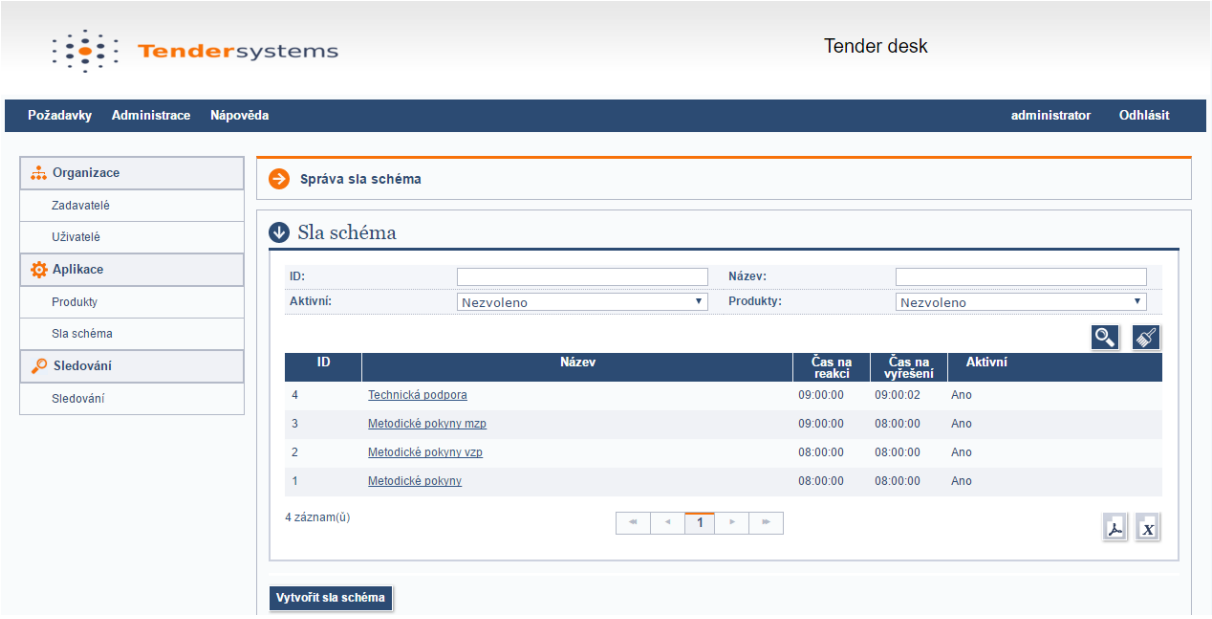
Obrázek příloha 29: Stránka pro vytvoření produktu [autor]



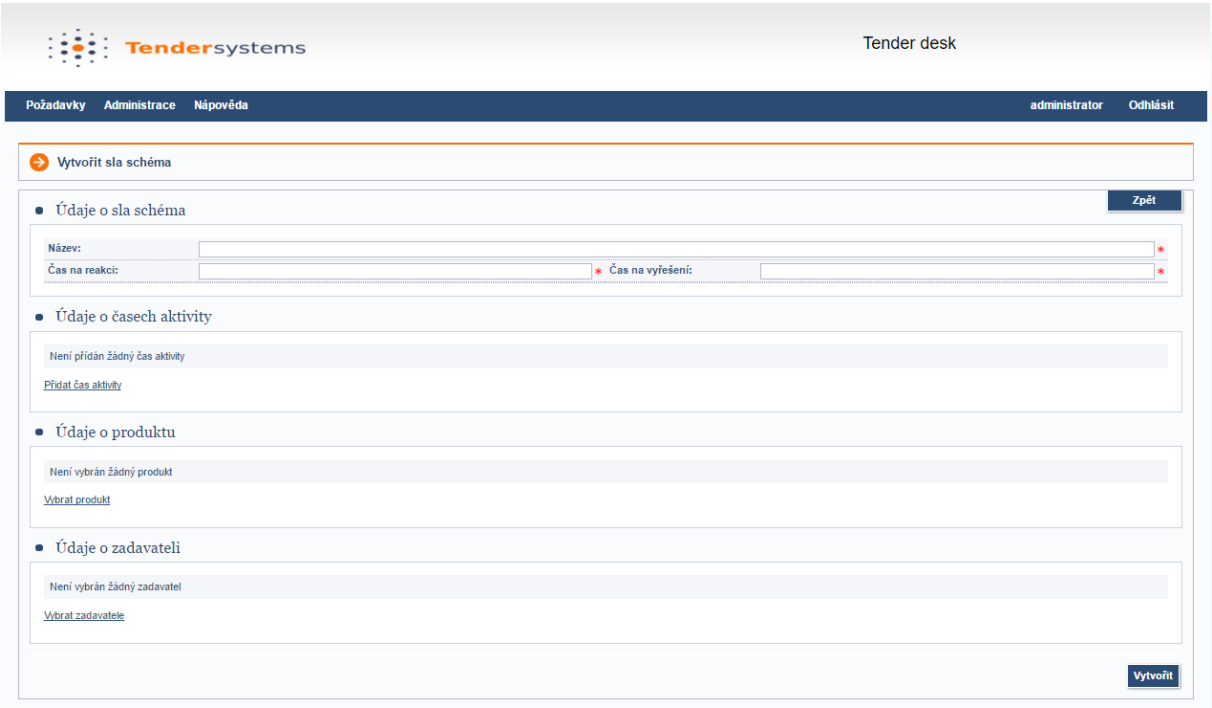
Obrázek příloha 30: Stránka detailu produktu [autor]



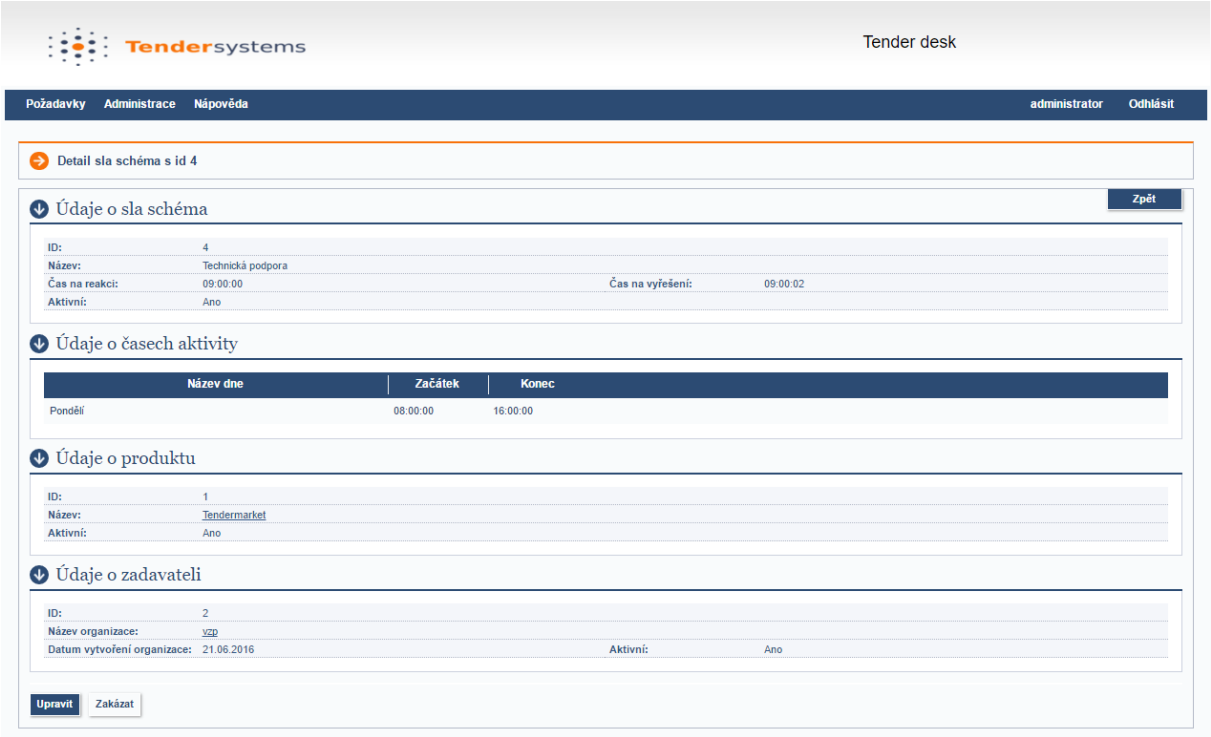
Obrázek příloha 31: Stránka pro upravení produktu [autor]



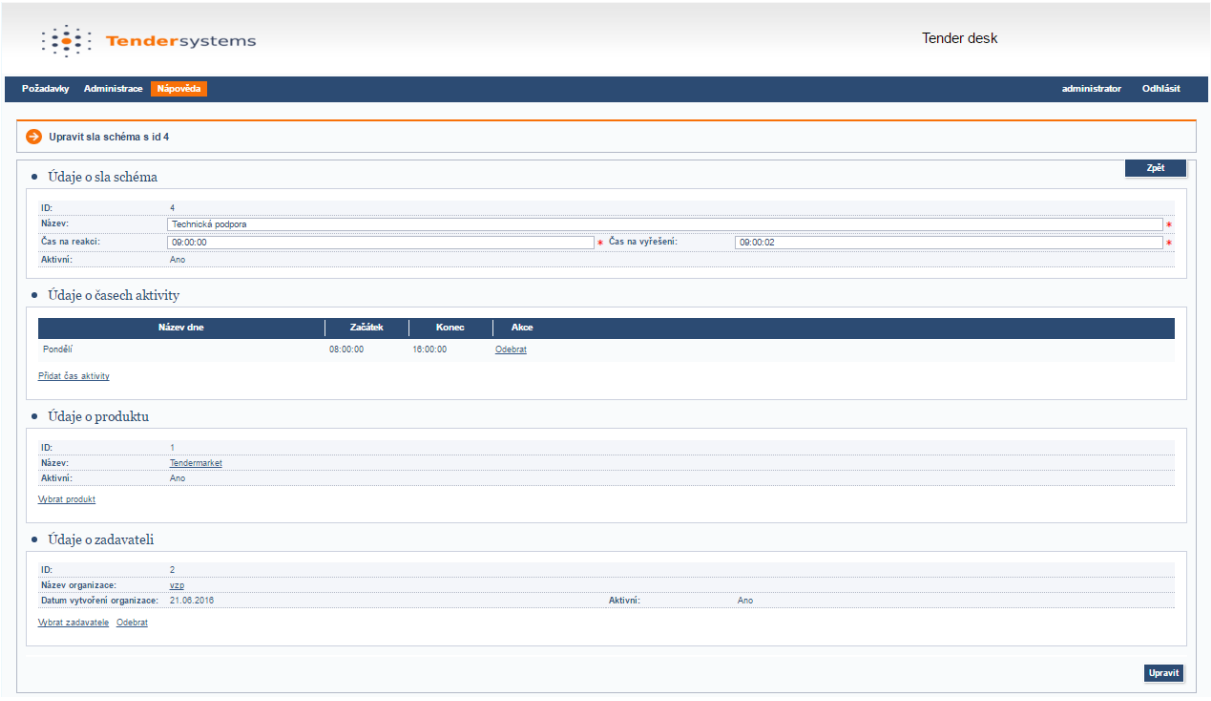
Obrázek příloha 32: Stránka se seznamem SLA schémat [autor]



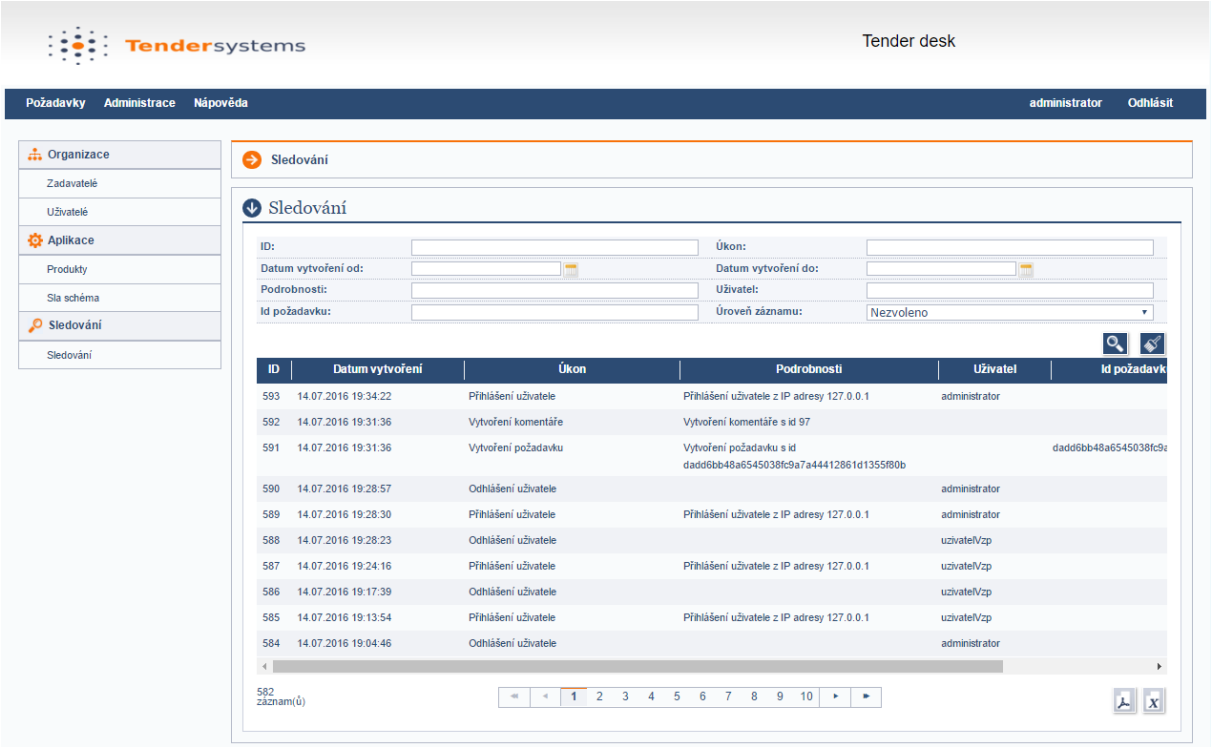
Obrázek příloha 33: Stránka pro vytvoření SLA schémat [autor]



Obrázek příloha 34: Stránka detailu SLA schémat [autor]



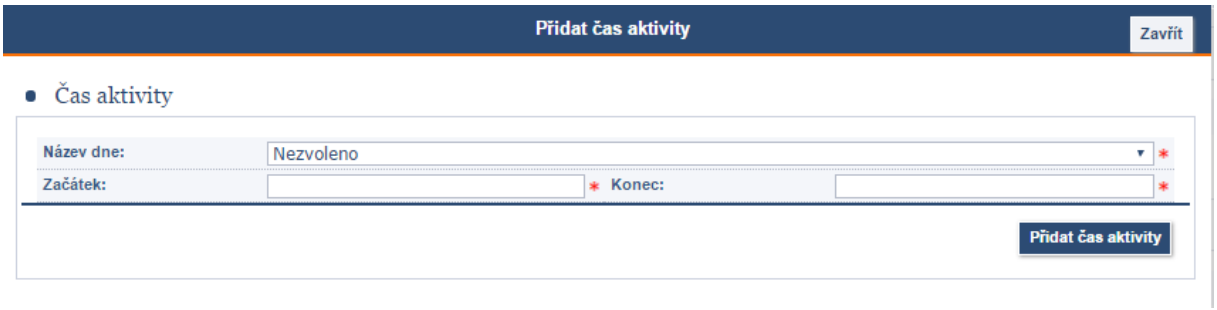
Obrázek příloha 35: Stránka pro upravení SLA schémat [autor]



Obrázek příloha 36: Stránka se seznamem auditní stopy [autor]

Obrázek příloha 37: Modální panel pro výběr produktu [autor]

Obrázek příloha 38: Modální panel pro výběr zadavatele [autor]



Obrázek příloha 39: Modální panel pro přidání času aktivity [autor]

C. Obsah kompaktního disku

K bakalářské práci je přiložen kompaktní disk, na kterém jsou uloženy všechny zdrojové soubory aplikace s uživatelskou příručkou. Na disku se nachází nakonfigurovaný aplikační server.

D. Seznam použitých tříd

Název třídy	Umístění
HelpdeskNamingStrategy	cz.tendersystems.helpdesk.entity.general
GeneralException	cz.tendersystems.helpdesk.exception
HelpdeskException	cz.tendersystems.helpdesk.exception
SecurityException	cz.tendersystems.helpdesk.exception
ValidationException	cz.tendersystems.helpdesk.exception
BackNavigationAction	cz.tendersystems.helpdesk.general
HelpdeskExceptionHandler	cz.tendersystems.helpdesk.general.jsf.exception
HelpdeskExceptionHandlerFactory	cz.tendersystems.helpdesk.general.jsf.exception
ViewContext	cz.tendersystems.helpdesk.general.jsf.extensions
ViewContextExtension	cz.tendersystems.helpdesk.general.jsf.extensions
ViewScope	cz.tendersystems.helpdesk.general.jsf.extensions
FacesMessagesPhaseListener	cz.tendersystems.helpdesk.general.jsf.messages
GeneralFacesMessage	cz.tendersystems.helpdesk.general.jsf.messages
Parameter	cz.tendersystems.helpdesk.general.jsf.parameter
ParameterAction	cz.tendersystems.helpdesk.general.jsf.parameter
AbstractResource	cz.tendersystems.helpdesk.general.jsf.resources
CustomResourceHandler	cz.tendersystems.helpdesk.general.jsf.resources
ImageResource	cz.tendersystems.helpdesk.general.jsf.resources
LayoutCustomization	cz.tendersystems.helpdesk.general.jsf.resources
StyleResource	cz.tendersystems.helpdesk.general.jsf.resources
AuthenticationAction	cz.tendersystems.helpdesk.security
FileDownloadServlet	cz.tendersystems.helpdesk.servlet
ToolsForBinaryDownload	cz.tendersystems.helpdesk.general

Tabulka příloha 12: Seznam použitých tříd

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Richter Zbyněk	Rokytno 152, Rokytno	I1100930

TÉMA ČESKY:

Realizace webové aplikace Helpdesk pomocí vhodného testovacího nástroje

NÁZEV ANGLICKY:

Implementation of web application Helpdesk using appropriate test tools

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce: Cílem práce je vytvořit webovou aplikaci helpdesk. Bude popsáno současné řešení ve firmě Tendersystems a navrženo a realizováno nové řešení. Při realizaci bude kladen velký důraz na psaní testů. Pro tento účel budou uvedeny nástroje, ze kterých bude vybrán nejvhodnější k samotné realizaci aplikace.

1.Úvod
2.Popis zkoumané oblasti
3.Testovací nástroje
4.Analýza
5.Implementace
6.Závěr
7.Seznam použitých zdrojů

SEZNAM DOPORUČENÉ LITERATURY:

bude upresneno

Podpis studenta: Richter Datum: 26.8.2015

Podpis vedoucího práce: Malý Datum: 26.8.2015