VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

CENTRÁLNÍ PORTÁL PRO ŘÍZENÍ VÝVOJE PROJEKTŮ V IT FIRMĚ

CENTRAL PORTAL FOR PROJECT DEVELOPMENT MANAGEMENT IN AN IT COMPANY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MAREK CIGÁNIK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÍTĚZSLAV BERAN, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Cigánik Marek**
Program: Informační technologie
Název: **Centrální portál pro řízení vývoje projektů v IT firmě**
Central Portal for Project Development in IT Company
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte nástroje pro tvorbu moderních webových aplikací a techniky UX. Seznamte se s API často využívaných IT služeb pro vývoj projektů (Github/Gitlab, Asana, Jira, Toggle, Clockify a pod.).
2. Navrhněte systém pro společnou správu a synchronizaci s externími IT službami s cílem zvýšení efektivity řízení práce na IT projektech. Zaměřte se na časté procesy a GUI.
3. Implementujte navržený systém s cílem testovat klíčové uživatelské procesy s využitím relevantních dostupných technologií.
4. Vyhodnoťte vlastnosti výsledného systému na základě experimentů s reálnými uživateli.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Brian Burke. *Gamify: how gamification motivates people to do extraordinary things.* Brookline, MA: Garthner, Inc., 2014. ISBN 9781937134853.
- Joel Marsh. *UX pro začátečníky*. Zoner Press, 2019.
- Steve Krug. *Don't make me think, revisited: a common sense approach to web usability.* San Francisco: New Riders, ISBN 978-0321965516.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem práce je vytvořit systém pro podporu vývoje týmových IT projektů odbouráním nutnosti některých akcí pomocí automatizace a poskytování informací z různých zdrojů na jednom místě. Tyto zdroje informací a zároveň automatizace jsou nad vývojářskými nástroji Git, správcem úkolů a časovačem pro měření času. Příklady takových nástrojů jsou ve stejném pořadí Gitlab, Jira a Clockify. Podstatou vytvořeného systému je komunikace s těmito nástroji pomocí jejich API, provádění akcí za vývojáře a získávání potřebných informací. Vytvořený systém poskytuje dvě implementace repozitáře, dvě implementace správce úkolů a dvě implementace časovače. Poskytuje možnost jednoduchého rozšíření o další nástroje. Přínosem této práce jsou malé aspekty systému, které dovolují vývojáři ušetřit čas na repetitivních a formálních záležitostech a dovoluje mu více se věnovat samotnému vývoji software.

Abstract

The goal of the thesis is to create a system to support development of team IT projects by eliminating the need of various actions through automation and providing information from numerous sources in one place. These sources of information, as well as automation, are Git's development tools, issue managing tools, and timers. Examples of such tools in the same order are Gitlab, Jira and Clockify. The core of the created system is communication with these tools using their API, performing actions on behalf of the developer and obtaining the necessary informations. The created system provides two repository implementations, two issue managing tools implementations, and two timer implementations. The application provides possibility to be extended by implementing other tools. The benefits of this work are small aspects of the system that allow the developer to save time on repetitive and formal issues and allows him to focus more on the software development itself.

Klíčová slova

vývoj software, týmový projekt, webová aplikace, užívatelské rozhraní, testování rozhraní

Keywords

software development, team project, web application, user interface, interface testing

Citace

CIGÁNIK, Marek. *Centrální portál pro řízení vývoje projektů v IT firmě.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vítězslav Beran, Ph.D.

Centrální portál pro řízení vývoje projektů v IT firmě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Marek Cigánik
10. května 2021

Poděkování

Chtěl bych se poděkovat vedoucímu mé práce, pánovi Ing. Vítězslavovi Beranovi Ph.D, za pomoc a nasměrování při tvůrčích procesech. Také za to, že mi ukázal, co je na software důležité, aby byl použitelný a jak si tyto informace zjistit v budoucnosti. Dále chci poděkovat všem lidem, kteří mi poskytli odbornou konzultaci, podíleli se na průzkumu nebo testování. Děkuji své rodině a přátelům za podporu, měl jsem díky nim pocit, že dokážu udělat cokoliv.

Obsah

Kapitola 1

Úvod

Cílem práce je vytvořit systém pro automatizaci a přehlednou správu pracovních, školních nebo osobních IT projektů. Projekty uživatel bude spravovat pomocí pracovních nástrojů. Systém bude moct velkou část práce s nástroji automatizovat a poskytovat informace z nich abstrahované. Bude podporovat práci v týmu, sdílené projekty, jednu ze zásadních funkcionalit, které by aplikace pro vývojáře měly obsahovat. Ve výchozí formě systém bude poskytovat základní balíček nástrojů, který může uživatel rozšířit o své oblíbené. Nástrojem je myšlený systém, který ulehčuje práci na projektu a projektové řízení, budou rozděleny do 3 skupin podle funkcionalit:

- Repozitář
- Časovač
- Správce úkolů

kde:

Repozitář je nějakou správcovskou službou nad systémem Git. Zde jsou uloženy projektové soubory, zachycena historie vývoje. V základu služba Git umožňuje větvit projekt a následně zpátky větve spojit. Z této funkcionality vznikly normalizované postupy, jak projekt tímto stylem vyvíjet. V systému si člověk bude moci vybrat, která normalizovaná forma bude v projektu použita.

Časovač je služba schopná zaznamenávat čas strávený nad úkoly v projektu ve formě časových záznamů. Záznamy umí přiřazovat projektu, klientovi. Umí vytvářet skupinové projekty, tj. přiřazování časových záznamů projektu od různých uživatelů.

Správce úkolů je systém umožňující rozdělit práci do logické struktury úkolů. Úkoly umí přiřadit jednotlivcům, umí zachytit stav úkolu. Úkoly se dají agregovat do milníků.

V této práci na začátku definuji potencionálního uživatele a jeho úkony, na které je vytvořený systém zaměřen. Dále jsou popsané koncepty a teorie potřebných konkrétních znalostí k vytvoření funkční této aplikace. Na základě znalostí v první kapitole je definovaný problém a návrh vylepšení. Návrh je rozdělen do vícero kategorií, přičemž každá pomáhá uživateli určitým způsobem a spolu tvoří jeden ucelený koncept. Dále je popsán návrh řazení uživatelů podle projektové role, datový model a rozhraní aplikace. V kapitole realizace jsou popsány použité nástroje, optimalizace potřebné k pohodlnému použití systému, způsob delegace akcí na podpůrné nástroje. Mimo technických informací je zde popsán i pilotní průzkum a uživatelské testování.

Kapitola 2

Cílový uživatel a webové aplikace

Z teoretického hlediska existuje mnoho úskalí a znalostí, které je nutné znát pro realizaci práce. Vyžaduje se, aby byla vnitřně konzistentní a tvořila základní kámen pro návrh systému. Zde je zaměřena pozornost na témata, které jsou potřebná pro vypracování MVP (Minimum Viable Product, produkt s nejmenší možnou funkcionalitou, dále pouze MVP).

2.1 Uživatel

Je osoba pracující v oddělení IT malé nebo středně velké firmy (do 30 lidí v IT oddělení). Je vývojářem, má zkušenost s prací v týmu a také v týmu pracuje. Jeho firma má know-how[1], podle kterého řídí proces vývoje systémů. Není ovšem axiomem, právě naopak otevřené novým možnostem. Nebojí se vyzkoušet příležitosti, systémy, služby, které potencionálně mohou zefektivnit mimo jiné právě tyto řídící procesy. Efektivita práce je pro něho klíčová.

Aplikace není směrovaná na uživatele jako jednotlivce, ale celý pracovní tým. Zmíněné nástroje byly vytvořeny pro týmovou spolupráci, tudíž aplikace, která je sdružuje, je také týmově zaměřena.

Běžné pracovní postupy

Tato část bude řešena pilotním průzkumem. Získané znalosti vytvoří předpoklad, že z hlediska použitelnosti bude vytvořen funkční systém. Slovo "běžné" je uvedeno schválně a je mu přikládán důraz, jelikož v průzkumu bude očekáváno mnohé totožné, ale i extrémy, na které se není třeba zaměřovat. Tyto málo vyskytující se případy si mohou firmy doimplementovat.

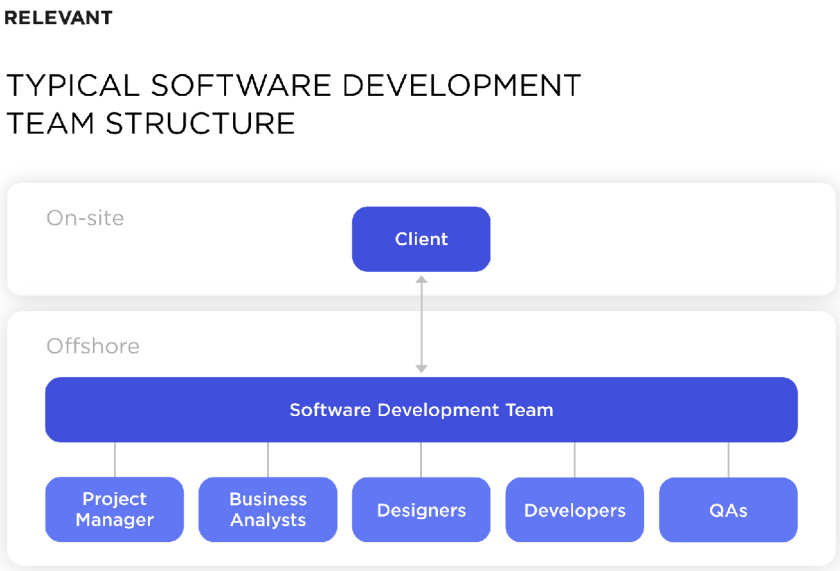
Mezi sledované informace ohledem běžných pracovních postupů patří:

- inicializační páce
- potřebné vedlejší projektové soubory (makety, uživatelské požadavky, diagramy)
- nejčastěji používané nástroje
- struktura týmu
- přísun informací; jaké jsou při vývoji zásadní

[1] Know-how - předpoklady a znalosti pro určitou činnost

- proces zpracování klientských požadavků

Předem se dá usoudit hned několik faktů. Co se týče inicializačních prací, každý projekt potřebuje úložiště a systém dělení práce. Nejoblíbenější úložiště pro týmové projekty je v současné době služba Git. Co se systému dělení práce týče, tady už možnosti nabírají různé variace. Zde se můžou použít Google dokumenty, Trello, Jira, verbální domluva, fyzická nástěnka... Neexistuje žádné konečné nebo ultimátní řešení, které by mělo pouze pozitivní aspekty a žádné negativa. Struktura týmu softwarového vývoje je v nejvíc vyhovujícím stavu následovná:



Obrázek 2.1: Optimální složení týmu [2]

Kde **Projektový manažer** je prostředníkem mezi vyšším vedením a pracovníky. Je zodpovědný za to aby se práce plnila.

Obchodní analytik formuluje cíle, analyzuje a dokumentuje řídící procesy. Hodnotí co funguje, co nefunguje.

Designéři vytváří interakci s konečnými uživateli. Navrhují jak systém bude vypadat a jak se bude používat.

Vývojáři jsou pracovníci, kteří produkt tvoří a píšou kód. Také mají na starosti technickou dokumentaci.

Inženýři zajišťování kvality produkt testují ujišťují, že splňuje klientské požadavky.

Takto vystavěný tým 2.1 je ale příliš nákladný pro malé a středně velké firmy. Mnohé tedy sahají po jiném přístupu a to nemít na jednotlivé odvětví specialisty, ale generalizované pracovníky schopny pojmout různé úkony, což má pozitiva:

- Každý člen týmu má dobrý pohled a pochopení projektu, tudíž se můžou soustředit na zdokonalování produktu jako celku
- Každý člen má dostatečnou kompetenci na dokončení různého typu práce, nejsou tedy závislí jeden na druhém

Ale i negativa:

- Takovému týmu často schází hluboká znalost odvětví. Používá plytké řešení.

2.2 Webové aplikace

Webová aplikace je software předkládaný uživatelům z jiných počítačů, tzv. serverů, prostřednictvím webového prohlížeče jako klienta. Komunikace mezi serverem a klientským prohlížečem probíhá z největší části prostřednictvím HTTP/HTTPS (Hypertext Transfer Protocol - internetový komunikační protokol, dále pouze HTTP/HTTPS) protokolu. Jelikož se jedná o textový protokol, rozhraní i akce aplikace jsou na obou stranách interpretovány. Na straně klienta se zejména jedná o rozbor HTML (Hypertext Markup Language - značkovací jazyk pro tvorbu webových stránek, dále pouze HTML) strukturovaných dat do vizuální podoby - webové stránky [11].

Pro uživatele je jednoduché webové aplikace používat, protože mu stačí prohlížeč, který ve výchozím stavu poskytuje každý běžný operační systém. Je to tedy jedna klientská aplikace pro velké množství software. Pro vývojáře se také jedná o oblíbenou formu distribuce software, jelikož nemusí na klientských počítačích nic instalovat a můžou si být jisti, že jejich produkt bude dostupný kdekoliv, kde je internetové připojení. Také (ne ve všech případech) stačí implementace na jednom stroji - serveru. Pomocí webových aplikací můžou poskytovat služby mnoha druhů, které IT sektor nabízí.

Jednoduché odvození proč jsou webové aplikace používány a proč jsou oblíbené bylo poskytnuto. Existují ale další výhody použití:
Použití standardizovaného komunikačního protokolu - jelikož distribuce software klientům probíhá přes internetovou síť, komunikace používá známé standardy a komunikační protokoly (HTTP, HTTPS, WebSocket...). Tato skutečnost znamená, že všichni vývojáři webových aplikací tvoří jednu velkou komunitu, kde se znalosti ohledem vývoje sdílí a šíří. Také existují knihovny pro snadnou práci při řešení často řešených problémů.
Klientská aplikace není na poskytovaném software závislá ani naopak - nejsou k sobě vázány. Pokud se webová aplikace změní, neovlivní to metodu jakou klientská aplikace volá a komunikuje. Poskytovaná aplikace se stává lépe spravovatelnou[7].
Použití XML (Extensible Markup Language - značkovací jazyk, dále pouze XML) - aplikace na prezentační vrstvě používají XML pro dokumenty (nebo JSON (JavaScript Object Notation - způsob zápisu dat, dále pouze JSON) pro data), které jsou stejně jako samotný komunikační protokol standardizované známé jazyky.
Poskytování API (Application Programming Interface - rozhraní pro programování aplikace, dále pouze API) **pro klienty různé od webových prohlížečů** - klienti webové aplikace nemusí být pouze webové prohlížeče. Software může poskytovat API prezentovaný standardizovanými textovými formáty (JSON, XML, YAML (Ain't Markup Language - formát pro serializaci strukturovaných dat, dále YAML)...). Uživatel si může vytvořit vlastní klientskou aplikaci přizpůsobenou jeho potřebám.

Práce se přímo nabízí býti webovou aplikací z několika důvodů.

- **Práce v týmu** - předpokladem je stejné prostředí, jeden systém. Webová aplikace ve výchozím stavu nabízí tuto možnost. Jelikož instance aplikace je nainstalována pouze na jednom serveru a ta je klientům servírována přes internetovou síť, reálně existuje pouze 1 prostředí, které uživatelé sdílí. Prostředím se myslí jak aplikační tak datová bublina.

- **Přenositelnost a bezpečnost** - Firma používající tuto aplikaci by ráda měla vlastní privátní instanci (chránila svoje projekty a data). Je pohodlné nainstalovat aplikaci na svůj vlastní server ze zdrojových souborů a touto jednou akcí ji zpřístupnit pro všechny své zaměstnance a nikoho jiného. Přístup k firemní síti může být omezen firemní síťovou politikou.
- **Integrace** - Nástroje, se kterými bude aplikace pracovat jsou taktéž webovými aplikacemi. Klient tedy zůstává stejný, na jakého jsou uživatelé zvyklí.

Architektura Model-View-Controller

Je softwarový architektonický vzor [4] rozdělující aplikaci do tří základních na sobě nezávyslých vrstev Model (modely), View (pohledy) a Controller (kontrolery). Každá vrstva má udělený záměr, kterého rámec striktně dodržuje a nepřesahuje. Vzniká logicky členěná, přehledná aplikace s jednodušší rozšiřitelností. Ve webových aplikacích jej používá množství frameworků (platforma pro vývoj aplikací, dále pouze framework) v různých jazycích, stala se totiž oblíbenou architekturou hned z několika důvodů:

- vývoj aplikace probíhá rychle
- je jednoduché pro jednotlivé vývojáře kolaborovat na jednom projektu
- jednoduchá rozšiřitelnost
- jednoduché hledání chyb díky rozdělení logiky do vícero vrstev
- párování pohledů a kontrolerů
- oddělení modelu od prezentační vrstvy - pohledů

Kontroler

je část aplikace, která se stará o logiku a obsluhuje požadavky uživatele [4]. Pracuje s vrstvou modelu a vrací pohledy. Jedná se tedy o prostředník. Uživatel požádá o data a kontroler tento požadavek zachytí. Může také validovat vstupní uživatelská data nebo uživatele autorizovat. Požadované informace získá z modelu a vrací pohled, ve kterém tázané informace zobrazuje. Ve webových aplikacích se jedná o komunikaci funkce programovacího jazyka (třeba PHP nebo Python), která představuje kontroler s databází, která představuje model. Jako pohled pak vrací HTML soubor obsahující data z databáze.

Model

představuje datové úložiště, procedury úzce s daty spojené a rozhraní pro manipulaci s daty [4]. Spolupracuje pouze s vrstvou kontroler, která jej obsluhuje.

Pohled

je ta část aplikace, se kterou uživatel interaguje. Zde se definuje zobrazení dat, poskytování akcí a vzhled rozhraní [4]. Pomocí pohledů obsluhuje aplikaci. Znamená to vytváření dalších požadavků na kontroler, který posléze vrátí další pohled nebo vykoná akci v pozadí bez změny pohledu. K této části aplikace se také někdy referuje jako front-end, čili přední část aplikace, okno do software.

Webové API

Webové API je způsob jak poskytovat rozhraní pro širokou škálu klientů zahrnujících webové prohlížeče, chytré telefony, tablety, hodinky... Jedná se o okno do softwarového programu umožňující jiným programům interagovat s ním bez nutnosti sdílet jakýkoliv kód. Komunikační protokol je HTTP/HTTPS. Přenášena jsou tedy textová data zpravidla naformátována jako JSON, HTML, XML nebo YAML [12].

V praxi práce s API nějaké služby znamená volání URL adres s různými hlavičkami nebo GET/POST parametry. Či už kvůli autorizaci nebo specifikování požadavku. V HTTP odpovědi uživatel dostane požadované informace v případě 2XX status kódu značící, že komunikace proběhla úspěšně a nenastala chyba ani na jedné z komunikujících stran.

Práce s API

Pro integraci s nástroji je nutné znát práci s internetovou komunikací pomocí protokolů HTTP a HTTPS. Nejen teorii ale i použití ve zvolené technologii (Python, PHP, Java...). Taktéž je potřeba znát konkrétní API rozhraní použitých nástrojů. Tohle je klíčová část, jádro projektu, proto jsou tyto znalosti NUTNOSTÍ. Také součástí této znalosti je vědomost o ER modelu (Entity-relation model - entitně vztahový model - znázornění dat, dále pouze ER model) služeb, se kterými se bude komunikovat. Ta se dá odhadnout praxí se službami, nebo pomocí jejich API dokumentace:

Základem modelu v nástroji *Gitlab* je projekt. Ten má své členy s různými rolemi a tudíž různými právy. V projektu se nachází větve, kde větev *master* je základní, vytvořená už na začátku. Větve mají svého předka (mimo *master* větve) a své potomky. Ke větvi se vážou *commity* (*commit* - příspěvek kódu do repozitáře, dále pouze *commit*). *Commit* má svého autora, větev a také komentář. Větve se dají slučovat. To je proces, při kterém proběhne příspěvek kódu jedné větve do druhé. Takové sloučení může být plánované tzv. *merge request* entitou, která má svého zakladatele, komentáře, popis a uživatele, který tento *merge request* schvaluje[1].
Služba Github má základ modelu totožný[2] se službou *Gitlab*.

Z hlediska služby *Gitlab* jako správce úkolů, projekt obsahuje tzv. *issues*, které představují úkolové jednotky. Ty mají svého zadavatele, uživatele, který má úkol splnit, popis a stav. Stav je obecně udáván tím, zda-li je úloha nová, na testování, dokončená... *Issues* se agregují do milníků, které představují skupinu úkolů, po kterých dokončení se projekt posune do nové fáze, bude vydaná nová verze nebo započne nový sprint[3]. Dle poměru splněných úkolů k nesplněným se dá číst hodnota splnění milníku jako celku v procentech. Milníky jsou důležité zejména pro týmového manažera nebo klienta.
Github má základ modelu z hlediska služby jako správce úkolů se službou *Gitlab* totožný.

Časovač *Toggl*[4] a *Clockify*[5] rozdělují projekty a klienty do tzv. *workspaces* (pracovní prostředí), které představují pracovní prostředí. V jednom *workspace* může být více uživatelů, tudíž můžou projekty a klienty sdílet. Časové záznamy se v službách skládají z data a času počátku, data a času ukončení, vypočteného času, autora a projektu, kterému záznam náleží.

[1] Gitlab API - https://docs.gitlab.com/ee/api/
[2] Github API - https://docs.github.com/en/rest
[3] Sprinty - krátké, opakovatelné fáze vývoje software
[4] Toggl API - https://github.com/toggl/toggl_api_docs
[5] Clockify API - https://clockify.me/developers-api

Další části nebo detaily modelů nejsou pro systém důležité.

Cache

Je systém uchovávání dat za účelem rychlejšího přístupu k nim u dalšího požadavku. Filozofie cache je uchovat malé a často dotazované data "blíž" k příjemci. Zmenšuje to časovou odezvu a výpočetní náročnost. V cache se může uschovat celý webový dokument, jeho části běžně tázané z databáze nebo výpočetní výsledky/mezivýsledky. Cache data se ukládají pomocí klíče jako ukazatele a expirační doby. Po expiraci se data považují za nevalidní. Způsob uložení se liší podle implementace - do paměti RAM nebo do souborů na disku. Hlavní výhodou je rychlejší přístup k datům. Pokud systém používá externí datové zdroje nebo používá náročné výpočty k jejich finální transformaci na informace, použití cache se stává žádaným aspektem aplikace [1].

Nevýhodou je neaktuálnost informací. Pokud u požadavku bude použit cache jako zdroj dat, vždy je šance, že data budou neaktuální. Mezi prvním požadavkem - uložením dat do systému cache a druhým požadavkem můžou být data v původním zdroji změněna. Tudíž v požadavku druhém uživatel dostane data z cache, která už nejsou aktuální.

2.3 Uživatelská zkušenost

Uživatelská zkušenost je termín známý pod zkratkou UX (User Experience). Je to široký pojem, který vysvětluje všechny aspekty lidské zkušenosti se systémem. Zahrnuje rozhraní, grafiku, design, fyzickou interakci a manuál. Dále se tento termín vykládá jako snaha o vytvoření soudržného, prediktivního a především žádoucího designu.

UX je často spojován se termínem „usability", tedy použitelnost - pro UX je důležité, aby věc, kterou vyprodukuje byla použitelná. Občas je tento pojem zaměňován se zkratkou UI - User Interface, které s pojmem souvisí (protože UI je na UX navázáno), ale není to to samé [6].

Přehlednost a dostupnost informací/akcí je v aplikacích klíčová. Uživatel bude očekávat, že systém, kterého efektivita je jeden z pilířů, nebude náročný na použití. Design a GUI (Graphical User Interface - grafické uživatelské rozhraní, obsahuje interaktivní grafické ovládací prvky) se tomuto předpokladu musí přizpůsobit. Jak zobrazovat informace? V jakém pořadí, na jakých podstránkách? Jak definovat akce? Jak je uživateli interpretovat?

Pravidlo s maximálním počtem kliknutí pro dosažení jakékoliv akce/informace není tak důležité. Existuje jiný, doporučenější postup a to nenutit uživatele přemýšlet nad akcemi a navigací po stránce. Je jedno kolik kliknutí uživateli zabere, než se někam dostane, hlavní je to, aby si cestu nemusel pamatovat a aby byla intuitivní [5].

Uživatel stránku nečte, on jí skenuje. Už navštívil mnoho stránek a tak podvědomě ví, kde co hledat a pokud rozložení nesedí s všeobecným standardem, bude nucen začít věnovat velkou pozornost a to ho vyčerpává, ztrácí motivaci se systémem pracovat [5].

I proto je důležité používat hodně nadpisů a formátovat text tak, aby toto skenování podporoval. Nepoužívat žádné zbytečné "happy talks", udělat vizuálně jasné, které prvky jsou interaktivní. Velkou částí toho, co uživatel na stránce dělá, je hledání další věci, na kterou kliknout. Vizuální separace interaktivních prvků mu s navigací pomůže. Vizualizační separací rozumíme změnu barvy, kurzoru, tvaru objektu (třeba u tlačítka) [5].

Důležitým aspektem je také poskytovat odrazové můstky, tzn. dát uživateli místa, kde začít. Typicky totiž nezkoumá, jak systém funguje, ale brodí se. Stačí aby našel CTA (Call to Action - výzva k akci, dále pouze CTA), který málo napovídá tomu, co hledá, a už na

něj klikne. Odrazové můstky musí být tedy přesně definovány. Je potřebné ale dát si pozor na vizuální ruch. Každý uživatel má nějakou toleranci vůči komplexitě a distrakcím [5].

Dalším zdrojem nadbytečných slov jsou instrukce. Pokud nejsou velice krátké a výstižné, nikdo je ani číst nebude. Uživatel je čte pouze tehdy, když se mu akce nepodaří vykonat. Instrukce musí být stručné, nejlépe nepoužívány vůbec. Samotné jejich použití znamená, že stránka není intuitivní [5].

Navigace by měla být jasná, vždy dostupná. Hlavní 2 funkcionality navigace jsou dostat uživatele kam potřebuje a ukázat, kde se právě nachází. Správně by měla vždy obsahovat komponenty: *Site ID*, sekce a *utility*. Stejně jako očekáváme název budovy nad předním vstupem, očekáváme *Site ID* vlevo nahoře webové stránky. Jedná se o logo nebo název. Sekce by měly obsahovat pouze top level hierarchické struktury celkového obsahu. *Utility* jsou důležité prvky, které nejsou přímo součástí hierarchie stránky (přihlášení, registrace...). Osvědčené je také přidat sekundární navigaci typu breadcrumb - seznam odkazů, přes které se uživatel dostal tam, kde se nachází [5].

2.4 Testování a pilotní průzkum

Software je potřebné otestovat aby bylo zajištěno jeho správné chování a aby splňoval požadavky, které se na něj kladou. Procesy zajišťování kvality se staly nedílnou součástí vývoje a to z dobrých důvodů. Vyplynou na povrch problémy, které si vývojář nebo kterákoliv zainteresovaná osoba mimo konečného uživatele nevšimne. Zainteresovaná osoba bývá zaujatá vůči svému produktu v mnoha ohledech, i použitelnosti. To znamená, že ověřování kvality musí probíhat za pomoci jiných osob, nejlépe do projektu nezaujatých, potencionálních koncových uživatelů.

Pilotní průzkum

Mimo testování jsou i jiné způsoby, jak zajistit to, že produkt bude kvalitní. Jedním z nich je pilotní průzkum. Je to způsob jak kalibrovat cíle produktu ještě před započetím vývoje, tudíž i před možností testování samotného. Sekundární význam je validace premis, ze kterých se vychází při návrhu. U nevalidních výchozích částech úsudků hrozí špatně navržený systém a časově i finančně náročné opravy. Pokud není důvod k poskytování produktu v plném rozsahu, pilotní průzkum má potenciál tuto skutečnost odhalit. Pokud se jedná o průzkum s uživateli, může mít podobu dotazníků, osobních pohovorů, analýzy chování a podobně [10].

Testování

Je výzkum jehož cílem je získat informace o testovaném produktu z hlediska použitelnosti. Testování můžeme definovat jako proces zajišťující kvalitu a ověřující to, zdali software splňuje požadavky a vlastnosti definované v jeho návrhu [1].

Pro splnění požadavků na systém z hlediska použitelnosti existuje odvětví UX testování. Je to metoda na měření toho, jak jednoduchá (nebo složitá) a uživatelsky přívětivá aplikace je. Malý výběr z cílové skupiny uživatelů používá systém podle předem vytvořeného scénáře za účelem odhalení vad v použitelnosti [9].

Pokud se aplikace zveřejní bez jakéhokoliv UX testování, je možné, že selže na některých z následujících bodů:

[1] Testování - https://www.geeksforgeeks.org/what-is-web-api-and-why-we-use-it/

- Kam mám kliknout dál?
- Které stránky potřebují navigaci?
- Která ikona reprezentuje co?
- Chybové hlášky nejsou konzistentní nebo efektivně zobrazeny
- Jsou obsahy stránek dobře formátovány?

Testování pomůže identifikovat vady v těchto nebo podobných případech ještě před zveřejněním.

Mezi známé druhy UX testování patří [8]:

- **Nemoderované UX testování** - Nemoderovaný test je relace, kde účastník dokončí úkoly sám, bez přítomnosti moderátora. Tento typ metody testování použitelnosti UX je rychlý a levný způsob, jak shromažďovat údaje o cílové skupině nebo testovaném produktu. Často bývá pořizovaný záznam aby se testování mohlo prohlédnout později. Pokud je potřeba více informací od více testovaných osob v kratším čase, nemoderovaný test je k tomuto účelu nejvhodnější. Protože nemoderovaný test je ale bez vnějších vstupů zadavatele, musí být zajištěno, aby byly všechny pokyny k testu jasné. Nejasné úkoly znamenají, že výsledky testů nebudou sledovat vlastnosti, které měly.
- **Moderované UX testování** - Moderované testování je typ testovací metody, který zahrnuje moderátora. Ten pozoruje a vede účastníky při jejich dokončení osobně nebo na dálku online. Moderovaná sezení umožní uspořádat živou relaci s účastníky, což vede k lepšímu porozumění chování a umožňuje hlouběji se zabývat kritickými body a problémy s použitelností. Moderovaný test může být užitečný, pokud se použije na začátku vývojové nebo prototypové fáze v počáteční fázi produktového cyklu. Sledují se reakce na nápady a systémové vlastnosti pro lepší nasměrování budoucích návrhů a vývoje. Pomocí moderovaných relací se testuje komplikovaný produkt, protože moderátor může účastníky procházet kontextem a jeho příslušnými ovládacími prvky. V nemoderovaném testovacím prostředí nemusí testovaná osoba pochopit složitost produktu.
- **A/B testování** - Tato metoda testování použitelnosti UX zahrnuje porovnání dvou verzí aplikace nebo webu proti sobě za účelem posouzení, která verze funguje lépe. Statistickou analýzou se určuje, která verze funguje lépe. Pomocí testování A/B se dá zjistit, jak ze stávajícího provozu získat lepší návratnost investic, snižuje míru okamžitého opuštění a také provádět drobné úpravy na vašem webu, například změna CTA tlačítka. Čím více lidí je otestováno, tím spolehlivější budou výsledky. Vyplývá ze statistického posuzování a vyvozování závěrů. Pokud se testuje s menším počtem lidí, existuje šance, že výsledky nebudou mít žádnou váhu. Pokud se testuje větší skupina účastníků, větší skupina lidí bude reagovat na obě verze návrhu. Tento výsledek je statisticky významnější a je nepravděpodobné, že by se změnil, pokud by byl proveden podobný test ještě jednou.

Kapitola 3

Návrh řešení

Zde je opsána analýza problému a koncepce systému poskytující vylepšení pro řešenou problematiku. Vychází ze zjištěné teorie. Velikým faktorem jsou zjištění z rozhovorů s Ludom Jambrichom, člověkem s dlouholetou praxí v projektovém řízení.

3.1 Analýza problému

Uživatel jako programátor tráví poměrně hodně času prací, která se dá automatizovat. V mnoha oblastech se automatizační nástroje uplatnili (file watchers, auto kompilátory, našeptávání ve vývojových prostředích). Tyto snahy potlačit čas vynaložený práci ve vývojovém okolí odráží skutečnost, že uživatel potřebuje snížit dobu investovanou do repetitivních úkolů a soustředit se na úkoly, které může vyřešit jen on jako specialista, to jest vymýšlet algoritmy a psát kód. Když se čas zabraný formalitami a repetitivními úkoly zredukuje, zbývá více času na reálnou práci. Tento fakt si uvědomují firmy a pracovní týmy velmi dobře.

Je faktem, že vývoj software je v dnešní době podporován různými nástroji, které poskytují různé funkcionality a různé informace. Pro získání konkrétních informací musí pracovník procházet konkrétní nástroje jednotlivě. Mnoho informací koreluje s informacemi nástroje jiného a tyto spojení musí zjišťovat a analyzovat pracovník sám. Zpřístupnění nejdůležitějších informací o projektu na jednom místě také pomáhá zkrátit čas vynaložený na řízení a každodenní práci.

Níže popsané činnosti vychází z osobních zkušeností a nastudované teorie. S každým nástrojem se zachází specificky podle jeho typu. Má primární úkony a méně používané sekundární, na které ale projekt není zaměřen. Primární úkony pracovníka jsou rozděleny do 3 skupin podle typu nástroje:

Repozitář

- Vytvoření projektu
- Přidání spolupracovníků
- Inicializační commit
- Vytvoření větvícího systému (master, develop, hot fixes, feat/..., fix/...)

v těle životního cyklu:

- Vytvoření větve
- **Práce na úkolu, commit, push**
- Vytvoření merge requestu (komentář, název, přidání uživatelů na revizi...)
- Akceptování změn nebo vrácení úkolu do TODO
- Merge

Správce úkolů

- Vytvoření úkolu, přiřazení vývojáři
- Přidání tagů; přesun úkolu do patřičného sloupce (TODO, vysoká priorita, nízká priorita...)
- **Práce na úkolu**
- Přesun na revizi, aktualizace tagů
- **Revize**
- Přesun do vyřízených úkolů nebo vrácení do TODO

Časovač

- Zapnutí stopek, přiřazení projektu a klienta časovému záznamu
- **Práce na úkolu**
- Zastavení stopek

Na konci měsíce (nebo jiné zúčtovací časové jednotky) vytvoření sumarizace odpracovaného času.

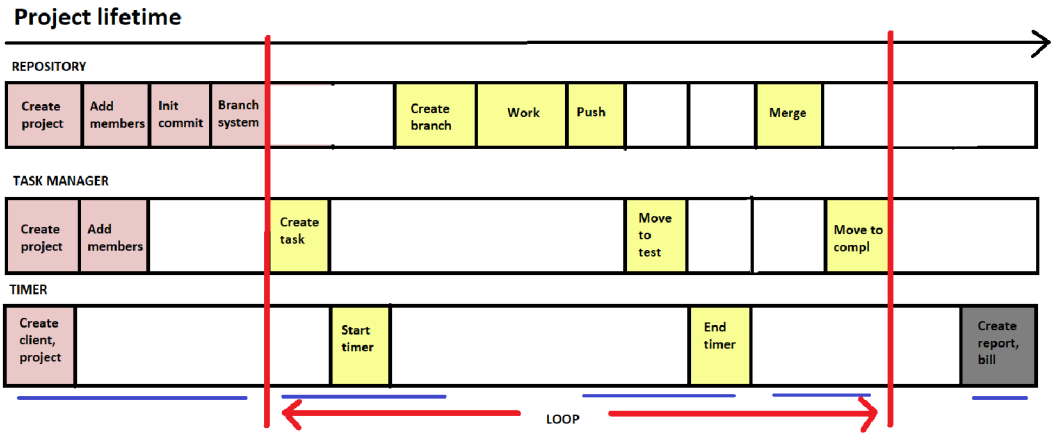
Tyto kroky pomáhají udržovat projektové řízení v stabilním stavu. Většinou se jedná pouze o důsledky potřeby zanechávat za sebou informace, která vznikla kvůli týmové práci. Kdyby člověk pracoval na projektu sám, nemusel by vytvářet nové větve pro každý přírůstek do kódu, veřejně zaznamenávat čas nebo tvořit nástěnky s úkoly. Úplně by stačily poznámky. Tučně jsou znázorněny ty části, které jsou důležité pro projekt samotný, zbytek je správa.

Další problém nastává u výběru nástrojů. Za roky vývoje těchto specializovaných systému na ně vznikly generalizované požadavky, které by měly jednotlivé implementace splňovat a proto jsou si v mnoha ohledech totožné. V dnešní době jsou k dispozici početné aplikace pro každou ze tří popsaných typů. Problém není ve výběru samotném, ale v možnosti vybrat různé kombinace pro jednotlivé projekty. Reálně volba jednoho nástroje vůči druhému nepřináší velké pozitiva, jedná se spíše o osobní preference nebo zvyk. Sto lidí, sto chutí; stává se, že pro i projekty jedné firmy používají různé nástroje. V extrémním případě může nastat to, že u vývoje dvou projektů bude každý používat jiné implementace nástrojů každého z typů, tedy dva projekty a šest podpůrných systémů. Zde začíná být správa projektů nepříjemná a roztříštěná.

3.2 Řešení

Je řazeno do tří kategorií, kde první kategorií je automatizace nástrojů nebo částí v různých nástrojích, které s sebou logicky souvisí, druhou kategorií je poskytování důležitých informací pro pracovníky sesbírané z vícero zdrojů - nástrojů, prezentovány jako celek a třetí kategorií je adaptérový přístup k práci s různými nástroji. Tyto řazení byly vytvořeny implicitně a validovány pilotním průzkumem.

Automatizace



Obrázek 3.1: Část životního cyklu vývoje software z pohledu vývojářů. Na grafu výstřižku životního cyklu projektu je znázorněn běžný postup vývoje projektu v IT:
Červená barva: inicializační postup - provádí se pouze jednou a to na začátku vývoje.
Žlutá barva: opakující se akce, tělo životního cyklu, také naznačeno červenými čarami
Šedá barva: opakující se jednou za zúčtovací období

Z grafu lze vypozorovat několik logicky propojených akcí, které se dají sloučit do jedné (naznačeno **modrou**)
Jedná se o:

- Inicializační práce
- Vytvoření úkolu, zapnutí časovače, vytvoření větve
- Push, přesun k revizi, ukončení časovače
- Merge, označení úkolu jako hotový
- Vytvoření časového reportu, zúčtování

Automatizace je jeden z primárních účelů aplikace. Odbourává nutné formality a pomáhá vývojářovi tím, že se místo toho může zaměřit na jinou práci. Návrhů pro automatizaci je hned několik.

V inicializačních pracích se vytvoří entity projektu ve vybraných nástrojích. Obyčejně zdlouhavá formální práce, která se přímo nabízí býti automatizována. V repozitáři se vytvoří

projekt, přidají se členové týmu, vytvoří se inicializační commit a zvolí se systém větvení, který bude projekt následovat a nuceně dodržovat. Vzniknou tedy mantinely, které ovšem nemají účel uživatele omezovat, ale pomáhat dodržovat normalizované pracovní postupy. Dodržování zamezuje konfliktům v repozitáři a chybám při vývoji.

V správci úkolů stačí v této fázi vytvořit projekt a přidat do něj členy. Nijaké úkoly se zde nespecifikují, projekt se ale může vytvořit s předem definovanou strukturalizací úkolů a milníků. Tím je myšleno jejich řazení typu TODO, Backlog, In progress, To test, Completed a podobně.

V časovači se vytvoří entita projektu a klienta, pokud ještě nebyla definována. Také se ale může vytvořit pracovní prostor čistě pro projekt nebo skupinu vývojářů.

Dále se v těle životního cyklu projektu dá automatizovat vytvoření úkolu do správce úkolů a nové větve v repozitáři. Zde nastává další propojení nástrojů. U každého nového úkolu, ať už jde o feature, bug-fix, hot-fix..., se vždy vytvoří entita v správci úkolů a nová větev v repozitáři, tudíž tato část je dobrým kandidátem na automatizaci - provedení obou kroků v jednom. Zase se odbourá činnost navíc pro uživatele. Započetí časovače záleží na tom, zda bude úkol vytvořen na okamžité zpracování (TODO) nebo pozdější (Backlog).

Po dokončení práce běžně vývojář přispívá svým kódem do repozitáře, označí úkol jako hotový, nebo na test a ukončí časovač. Označit úkol jako hotový nebo na test a vypnutí stopek se může vykonat automaticky po commitu nebo manuálně, pořád ale odpadne nutnost aktualizovat stavy v různých nástrojích.

Jako další je validování kódu (ta se provádí manuálně), označení úkolu jako hotové nebo vrácení do TODO v případě nesplnění zadání a spojení větve úkolu s výchozí větví (samozřejmě pouze v případě splnění zadání). Mimo samotné validace se opět dá spojit nevyhnutelně po sobě následující úkony.

Vytvoření časového reportu za zúčtovací období se dá jednoduše agregací časů strávených na úkolech nebo jinou prací. Tyto časy budou vypláceny vývojářům a účtovány klientům.

Poskytované informace, centralizace

Celá automatizace popsaná v předešlé podkapitole se děje na pozadí. Dalším důležitým aspektem systému jsou informace, které bude uživateli poskytovat o stavu projektu. Tyto informace budou zobrazovány a řazeny na základě role, kterou uživatel v projektu představuje. Důraz bude tvořen GUI praktikami k vizuálním rozlišením a strukturalizaci obsahu.

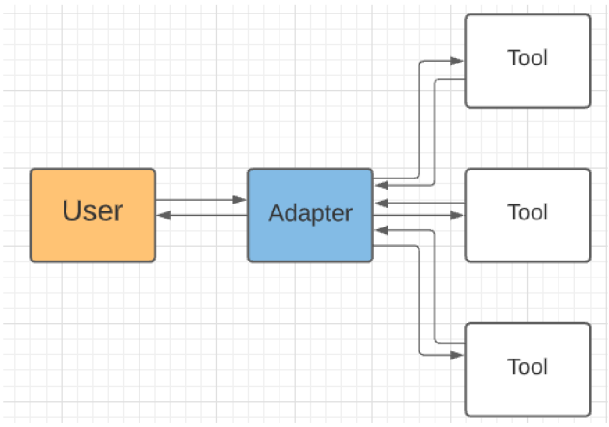
Z hlediska poskytovaných informací o stavu projektu by měla aplikace poskytovat následující:

- Výpis aktivit na projektu
- Přehled stavu projektu
- Detailní popis aktivity: který úkol byl splněn, odkaz na repozitář, čas práce, odkazy na commit a správce úkolů, revizní zprávu
- Výpis projektů a klientů k nim přidělených. Také agregovaný čas, který se bude účtovat
- Celkové zúčtovací rozhraní. To bude ale dostupné pouze pro uživatele se speciální rolí odrážející skutečnost, že je to člověk zodpovědný za finanční procesy firmy nebo je s nimi úzce spojen

- Aplikace bude rozlišovat jakou roli má člověk v projektu. Podle toho mu bude zobrazovat adekvátní informace; ty, které jsou pro něj důležité.
- Osobní revize svého postupu, své odvedené práce
- Přehled úkolů, které jsou po konečném termínu nebo se mu blíží

Adaptér

Problém použití různých nástrojů u řízení vícero projektů se dá řešit adaptérovým přístupem [3]:



Obrázek 3.2: Adaptérový přístup

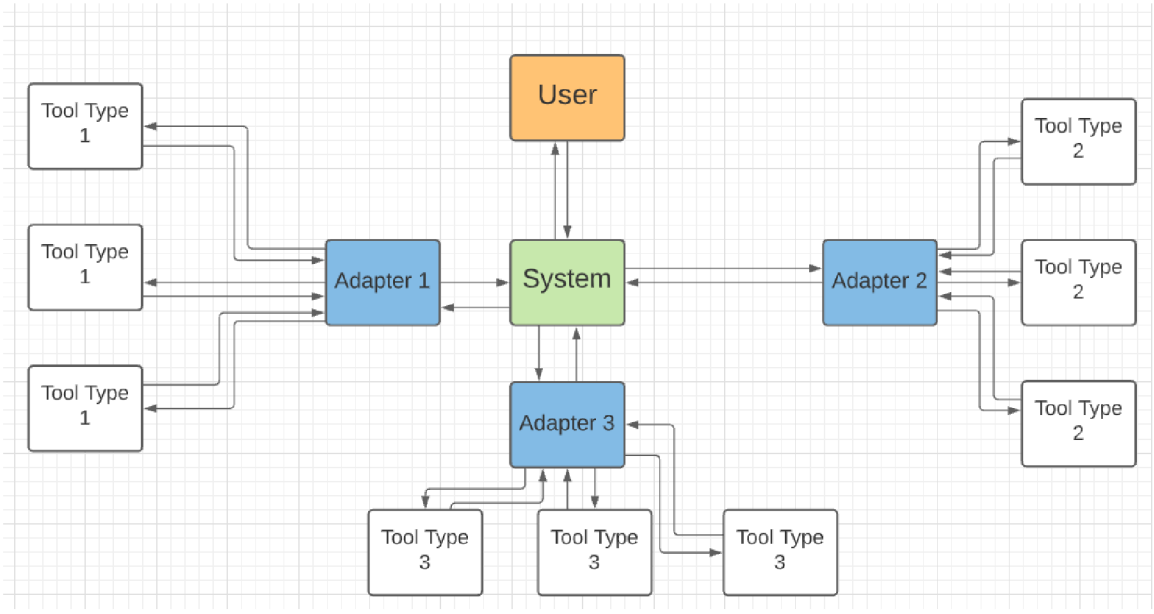
Kde uživatel nekomunikuje s nástrojem přímo, ale pomocí prostředníka, který je schopen komunikovat s vícero nástroji stejného typu, třeba repozitářem. Takový přístup když se uplatní u každého typu nástroje, vznikne jeden systém pro správu všech tří kategorií nástroje s různými implementacemi:

3.3 Role

Struktura týmu popsána v kapitole 2.1 obsahuje mnoho rolí. Je ovšem zmíněno, že v malých a středně velkých firmách, které jsou cílové, se takový tým těžce seskupí. Nejmenší reálně schopná pracovní jednotka se skládá z vývojářů a týmového manažera. Tyto role mají různý přístup a užití zmíněných tří nástrojů podle úkonů, které v nich mají na starosti. K těmto dvěma aktivním rolím je přidána ještě jedna role pasivní - pozorovatel. Jedná se o zainteresovanou osobu v projektu, která ovšem, jako z názvu vyplývá, pouze sleduje průběh. Tato role sdružuje klienta a vyšší management ve firmě, třeba ředitele.

Týmový manažer

Týmový manažer je prostředník mezi vyšším řízením (případně přímo klientem) a zaměstnanci. On je zodpovědný za interpretaci vize do technického řešení. Ve vytvořeném systému bude moci projekt řídit a přidělovat práci. Z tohoto hlediska je nezbytné, aby mohl vytvářet nové úkoly, určovat jejich prioritu a přiřazovat je pracovníkům. Také samozřejmě tyto úkoly dále spravovat. Kromě úkolů už nemusí do projektů přispívat ničím, nýbrž je pro něj



Obrázek 3.3: Adaptérový přístup pro různé typy nástrojů

důležitější sledovat průběh, hlídat termíny a získávat jiné informace pro něj primární a to jest:

- Jaké úkoly jsou plánované
- Na kterých se právě pracuje
- Které byly nedávno splněny
- Které nebyly splněny včas
- Hlídat projektový rozpočet

Pro jeho práci mu stačí správce úkolů a časovač. Správce úkolů z důvodu jejich zadávání a editaci a časovač z důvodu hlídání projektového rozpočtu. Odpracované hodiny jsou totiž základem pro zhotovení faktur.

Vývojář

Vývojář je člověk, který používá aktivně všechny tři nástroje. V časovači vytváří časové záznamy o své práci a získává informace o výši své mzdy. Do repozitáře přispívá svým kódem nebo jiným způsobem tvůrčí práce a ze správce úkolů čerpá informace o tom, co má dělat. Úkony v jednotlivých nástrojích se prolínají a pro započetí práce potřebuje použít každý z nich (platí i pro ukončení). Zapne časovač, zjistí úkol a označí ho jako rozpracovaný, vytvoří novou větev v repozitáři a až pak začíná pracovat. Reverzně použije každý z nástrojů u ukončení práce. Empiricky se jedná o jeden úkon, je však rozdělen jako důsledek použití specializovaných nástrojů.

Pracovník by měl mít rychlý přehled o svých úkolech, případně úkolech, na kterých pracují kolegové, aby nedošlo k nedorozumění nebo desynchronizaci. Své úkoly má řazeny podle typu (feature, bug-fix…), priority nebo konečného termínu. Vybere si jeden a začne pracovat. Měl by mít přehled o rozpracovaném úkolu.

Jeho vlastní *merge requesty*, tj. větve, které má on zkontrolovat, musí mít po ruce a hned jasně viditelné, aby nový kód dlouho nečekal na schválení. Důležitým prvkem je také informace o svém bilančním stavu za právě probíhající zúčtovací období. Ten je počítán z jeho času stráveným nad úkoly. Čas probíhající by měl vědět pozastavit, aby si mohl odskočit třeba na oběd.

Pozorovatel

Ten nevytváří nic, pouze sleduje průběh vývoje projektu. Pro něj jsou nejdůležitější milníky a projektový rozpočet.

Milník je skupina úkolů, po jejich dokončení projekt přechází do nového stavu. Nový stav může být třeba vydání verze. V základu je charakterizován počtem úkolů, které jsou splněny, počtem ještě nesplněných úkolů a konečným termínem. Postup v milnících je dobrým ukazatelem celkového postupu vývoje projektu.

Projektový rozpočet by se neměl nafukovat a pokud by takováto situace hrozila, pozorovatel o tom musí být zpraven. Měl by proto přehledně vidět kolik peněz se účtuje za právě probíhající zúčtovací období ale i za celkový čas vývoje projektu.

Z informací o milnících a projektovém rozpočtu může plánovat další postup a delegovat ho tým manažerovi, který tyto pokyny zpracuje do nového technického řešení.

3.4 Datový Model

V databázi aplikace bude reálně uložen pouze zlomek dat, které se budou používat. Většina bude získávána z externích zdrojů (zaznamenané časy, úkoly). Je potřeba uložit pouze reference na tyto objekty. Celý model pozůstává z následujících entitních množin:

- Activity - prováděná aktivita v projektu
- Client - seznam klientů
- Group - skupina uživatelů pracujících na projektu se stejnou rolí
- Invitation - pozvánky k projektům
- Membership - členství uživatele v projektu
- Project - seznam projektů
- User - uživatelé
- UserRepository - uživatelovi repozitáře
- UserTaskManager - uživatelovi správci úkolů
- UserTimer - uživatelovi časovače

Entitní množiny UserRepository, UserTaskManager a UserTimer

Se skládají z **url**, **access tokenu**, **typu nástroje** a **názvu** 3.4. Jsou to základní komunikační prostředky k práci s API různých nástrojů. Každý záznam v tabulce *uživatel* má N nástrojů, které se rozdělují do tří skupin - repozitář, časovač nebo správce úkolů. Díky názvu může rozlišit více registrovaných nástrojů na stejné adrese, třeba pracovní a osobní *gitlab* na `https://gitlab.com`.

Entitný množiny group, membership a invitation

Propojují záznamy *uživatel* se záznamy *projekt*. Nesou sebou informace o **nástrojích**, které uživatelé chtějí použít pro práci v projektu a uživatelovu **roli** v projektu nasledovným způsobem:

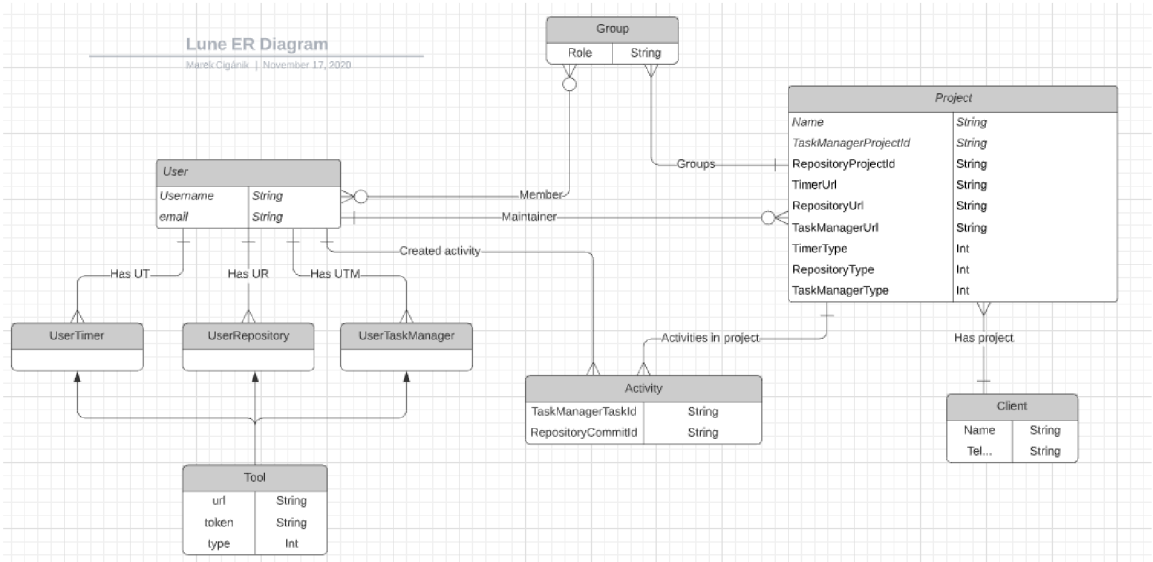
Uživatel má členství v *group* pomocí entity *membership*. V ní se specifikuje, s jakými nástroji do *group* vstupuje. V základě má každý záznam typu *projekt* vazbu na tři entity typu *group*, přičemž každá *group* sdružuje uživatele podle **role** 3.4. Tímto způsobem se dají uživatelé jednoduše kategorizovat.

Invitation tabulka uchovává záznamy o pozvánkách k spolupráci na projektech 3.4. Uživatel, který entitu projektu vytvoří může pozvat další uživatele k spolupráci. Po přijetí pozvánky se vytvoří *membership* záznam ke příslušné *group*, jelikož *invitation* záznam sebou nese i informaci, k jaké **roli** byl uživatel do projektu pozván. Je tu ale nutná podmínka, že nový uživatel má registrované nástroje, které jsou v projektu použity.

Entitní množiny project a activity

Projekt je základní jednotkou systému představující jednu pracovní instanci. Obsahuje **reference na projekty** v nástrojích, nad kterými je vytvořen. Přes *group* a *membership* tabulky má vazby na *uživatele*, kteří se na *projektu* podílí. Kromě toho obsahuje základní informace jako **název projektu**, **příslušící klient**, **typ větvení** v repozitáři a **projektový rozpočet** 3.4.

Také sdružuje záznamy v tabulce *aktivity*. Takový záznam uchovává práci. Obsahuje **odkazy na úkol**, který byl řešen, **záznamy v časovači**, spolu tvořící celkový čas strávený na aktivitě a **větev v repozitáři**, aby se tato větev mohla automatizovaně obsloužit po ukončení práce 3.4.

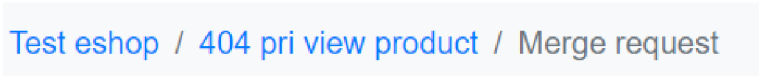


Obrázek 3.4: ER diagram datového modelu

3.5 Rozložení webu

Uživatel má po přihlášení přístup k projektům, ve kterých kolaboruje, pozvánkám ke kolaboraci, databázi klientů, svým registrovaným nástrojům, svému profilu a administraci pro uživatele s pravomocemi administrátora. Tyto top level stránky v hierarchii obsahu jsou dostupné z menu. Menu samotné indikuje barvami místo, kde se uživatel nachází.

V administraci může upravovat všechny informace entit z databáze, jedná se v podstatě o databázovou fasádu s aplikačními omezeními pro pohodlné úpravy jakékoliv informace. Pokud má uživatel pravomoc administrátora, v menu se mu poskytne k administraci přístup. V profilu může upravit svůj email, jméno, heslo a jazyk. Jelikož k použití systému nejsou od něho potřebné jiné informace, profil je pouze strohá a malá část aplikace. Své registrované nástroje má rozdělené v menu do tří kategorií podle typu. Po zvolení jedné ze tří možností se mu zobrazí seznam jeho nástrojů a možnost přidat nový. V menu položce s přístupem k projektům má zobrazeny pozvánky, seznam projektů a možnost přidat nový projekt. Po zvolení jednoho z projektů ze seznamu se dostává do kontextu dané entity a zobrazí se mu nové, projektové menu a breadcrumb navigace nad vnořenými pohledy v projektu.



Obrázek 3.5: Breadcrumb navigace

Zde v zobrazení projektu už je obsah personalizován podle uživatelovi role, jemu konkrétně přidělené práce a stavu projektu. Obsah se dělí do sekcí nebo takzvaných widgetů:

Pro proli Developer jsou k dispozici widgety:

- Úkoly na revizi
- Moje úkoly
- Výplata, odpracované hodiny

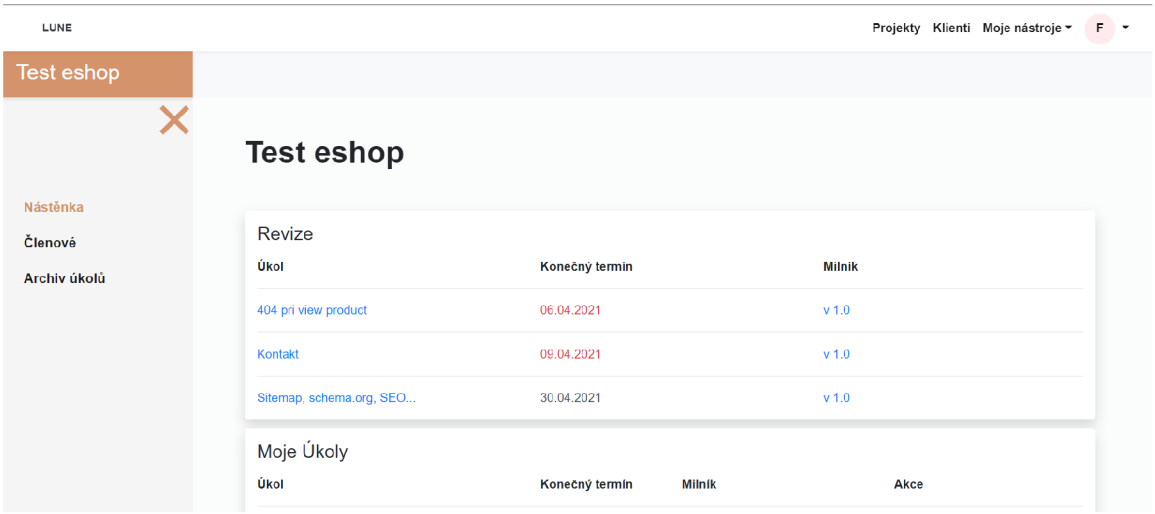
Pro roli Manažer jsou k dispozici:

- Úkoly po deadline
- Widget povýšení připravených úkolů do TODO, vytváření nových úkolů
- Milníky, vytváření milníků
- Seznam úkolů, na kterých se aktuálně pracuje
- Projektový rozpočet, odpracované hodiny jednotlivých pracovníků

Pro roli Observer:

- Milníky
- Projektový rozpočet, odpracované hodiny jednotlivých pracovníků

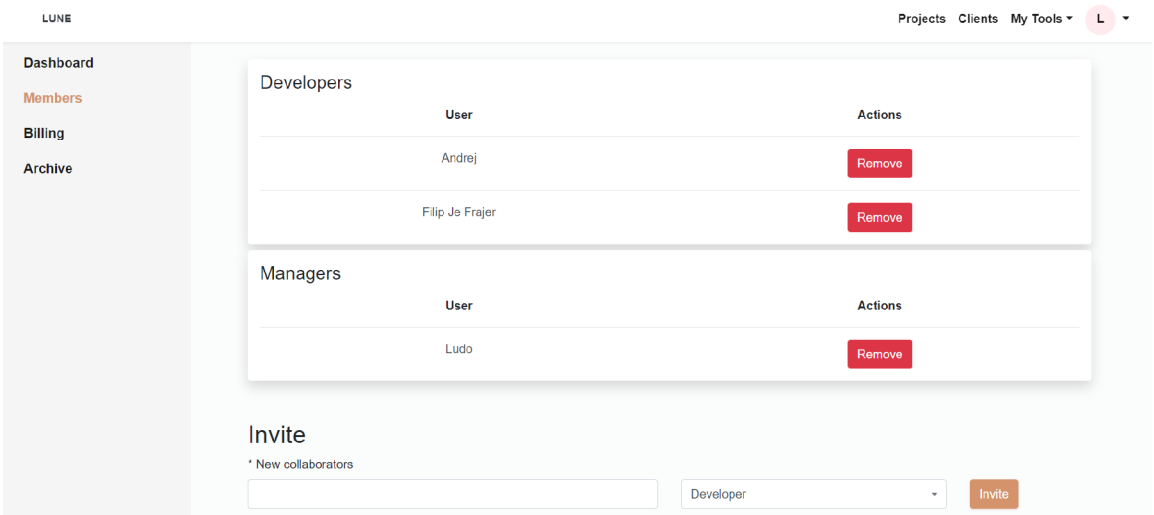
Tyto informace a akce s nimi spojené jsou dostupné z projektové nástěnky:



Obrázek 3.6: Náhled na projekt

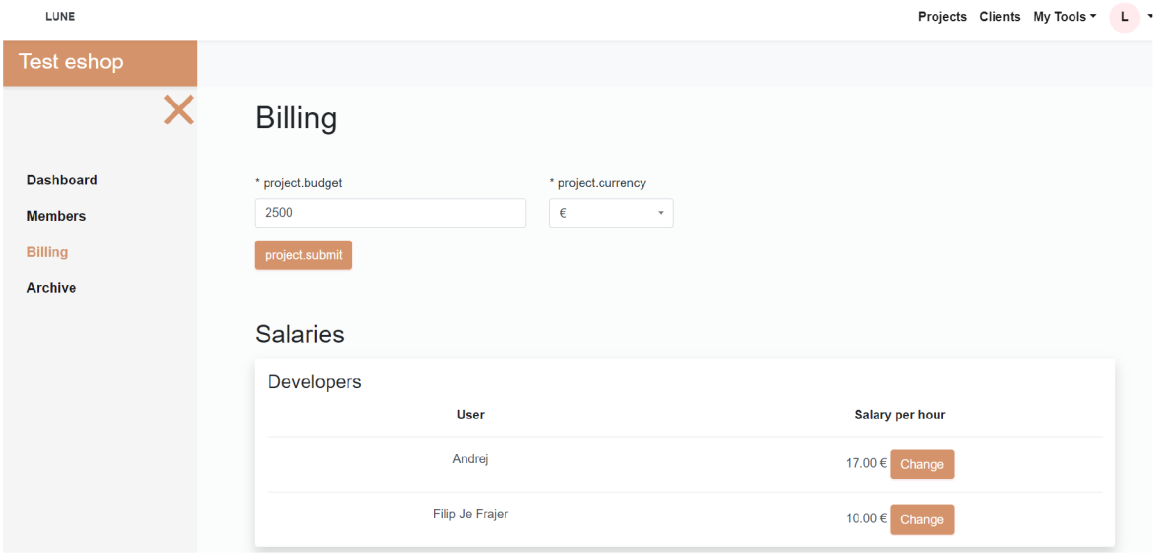
Jedná se totiž o nejdůležitější nebo nejčastěji čtené informace a prováděné akce. Vychází z návrhu, maket pohledů a poznatků získaných z pilotního průzkumu o míře důležitosti poskytovaných informací.

Dále se přes projektové menu uživatel dokáže dostat ke správě kolaborantů projektu 3.7. Zde jsou seznamy uživatelů podle role v projektu, možnost přizvání nových uživatelů a seznam pozvánek. Pozvánky i kolaboranty může z projektu odebrat.



Obrázek 3.7: Členové a pozvánka

V záložce fakturace 3.8 v projektovém menu může měnit projektový rozpočet, měnu, ve které se peníze evidují a hodinovou sazbu jednotlivým pracovníkům. K tomuto rozhraní má ale přístup pouze uživatel s rolí Manažer.



Obrázek 3.8: Správa peněžní stránky projektu

Jako poslední je v menu přístup k archivu úkolů v projektu, čistě pro informativní účely. Po rozkliknutí se uživateli zobrazí dvě tabulky s úkoly. První obsahuje nedávno dokončené úkoly a ta druhá všechny ostatní dokončené úkoly. Tabulky obsahují i informace o tom, kdy byly úkoly ukončeny a kým.

Kapitola 4

Realizace a testování

4.1 Pilotní průzkum

Pro pilotní průzkum jsem si našel tyto osoby:

Radovan Hanták - Programátor firmy GFXPulse v Považskej Bystrici, v IT pracuje již mnoho let. Má zkušenosti v různých odvětvích, s prací v týmu i spoluprací napříč firmami.

Filip Markovič - Junior programátor firmy Dgstudio, vystudoval IT obor střední školy, momentálně v 1. ročníku na VŠ Tomáše Bati ve Zlíně. Pracuje už skoro rok, práce v týmu je pro něj v zásadě nová.

Patrik Melicherík Pracuje v IT 8 let, nejdříve jako programátor, nově projektový manažer ve firmě Hyperia v Žilině. Pracoval ve více firmách, vyzkoušel různé pracovní postupy.

Samuel Mořkovský - Paralelně se studiem na FIT si našel práci u jednotlivce, jsou tedy teď 2 v relaci podřízený - nadřízený (neznám jméno, jen to, že je živnostník). Práce na dálku, také mají určité pracovní postupy, využívají nástroje pro práci v týmu.

Zjištění

Zásadní zjištění bylo od Radovana Hantáka, které upravuje cílovou skupinu uživatelů. Doposud byla aplikace mířená na malé/středně velké firmy, malé/středně velké IT týmy. To je důležitý faktor, výběr se ještě zúžil podle produktu, který tým tvoří. Radovan kategorizoval produkt

- 1. typ produktu, Webové stránky/eshopy a jiné produkty se šablónovým přístupem
- 2. typ produktu, vlastní produkt (třeba firma Microsoft vyvíjí balíček Miscrosoft Office)
- 3. typ produktu, zakázky a unikátní řešení

Popsal míru přidané hodnoty, kterou by systém poskytoval firmě na základě této kategorizace na stupnici od 0 (nejhorší) do 5 (nejlepší).

5 bodů dal pro firmy pracující s 1. případem produktu. Tady se totiž opakovaně provádí základní úkony, vývojové procesy a celý proces tvorby má mnohem plynulejší vývoj oproti dalším 2 variantám. Toto chování vyhovuje systému (nebo spíš systém svým chováním

vyhoví vývojářům). Systém postaví mantinely, určí základní pracovní procesy a projekty se jím budou řídit.

1 bod dal pro firmy pracující s 2. případem produktu, kdy vzniká málo projektů a většina z nich se ve vývoji odlišuje. Pro tento typ firmy se více hodí volnější a komplexnější přístup k metodikám při vývoji, než by systém mohl poskytovat.

0 bodů pro firmy pracující s 3. případem produktu. Zde projekty nabírají různé variace a komplexita řízení projektu je velmi vysoká. Radovan mi ukázal jeden takový projekt. Jednalo se o propojování pevných linek a práci s VoIP napříč pracovištěm pro Holandskou firmu. Řízení projektu bylo natolik komplexní, že se člověk mohl rovnou vzdát vize systému, který by tyto procesy generalizovaně zastřešoval. Zdůrazňuji - tohle byl jediný projekt a firma pracující s 3. případem produktu jich má víc.

Co se použití šablon pro nové projekty týče, pouze Dgstudio, firma zaměstnávající Filipa Markoviče, používala šablony pro nové projekty. Není to ovšem překvapení nahlédne-li na fakt, že jako jediná nepracuje na dlouhodobých projektech nebo unikátních zakázkách. Pro ostatní firmy by si šablonami pouze přivodily starosti, protože je málo pravděpodobné, že by je použili a také kvůli faktu, že šablony je potřebné udržovat aktualizované. Protože ale tento projekt má největší smysl právě pro firmy podobné Dgstudio, implementace šablon má smysl.

Žádná firma ale nepoužívá nástroje podobné tomuto ve smyslu vytvoření pracovního prostředí. Všemi formalitami při zakládání projektu prochází ručně znovu u každé instance. Vylepšení v tomto aspektu by mohlo být přínosnou a vítanou změnou.

U vzhledu těla životního cyklu došlo ke shodě s předpovězenými činnostmi mezi všemi tázanými subjekty, tudíž se dá považovat za validní. Jediný problém grafu je, že zachycuje utopický stav, kdy vývoj probíhá bez těžkostí. V reálném světě úkoly často neprochází přes proces validace vyšší autoritou a vrací se zpátky do fronty nevyřízených požadavků.

4.2 Použité nástroje

V této sekci jsou popsány vybrané technologie vhodné pro vývoj systému.

Framework Symfony

Využití už hotových generických řešení je dobrá věc z pohledu ušetření času a většího pohodlí pro vývojáře. Symfony[1] webový aplikační framework pro PHP. Používá MVC architekturu, zajišťuje přehledné rozdělení souborů do logických struktur, usnadnění implementace základních funkcionalit webu, které jsou pro všechny weby společné, třeba routing. Ve výchozím stavu pracuje s databází pomocí frameworku Doctrine, který umožňuje automatické převedení záznamů databáze na entity aplikace. Pro vykreslování HTML kódu využívá modulu Twig, pomocí kterého se k HTML souboru může přistupovat jako k šabloně, může se rozšiřovat nebo může zahrnovat další HTML soubory. Základem je dobrá konfigurace frameworku přes konfigurační soubory. Zde je potřebná dobrá znalost Symfony. Všechny základní konfigurace se nachází v souborech v adresáři **config/packages**. V základu má projekt následující adresářovou strukturu:
assets - obrázky, Javascript, CSS
config - konfigurační soubory
templates - twig soubory

[1] Symfony - https://symfony.com/explained-to-my-boss

migrations - databázové migrace, verzování databáze
public - soubory přístupné klientovi, jako robots.txt nebo sitemap
src - zdrojové soubory aplikace, entity, kontrolery...
vendor - PHP závislosti
var - cache a log
translations - translační soubory

která se dále větví.
Momentálně je v LTS verze 4.4, tato verze tedy bude použita pro vývoj aplikace[1].

Framework Doctrine a Object-Relational Mapping

Je PHP framework využívající ORM (Object-relational Mapping - objektově relační mapování, způsob konverze dat relační databáze na objekty, dále pouze ORM). Má nízkou náročnost na konfiguraci a definování databázového modelu, kde využívá entity jako základní jednotku záznamu v databázi. Framework Doctrine je rozdělen do tří vrstev[2]:

- **Common** - Definuje základní obecná rozhraní, třídy a knihovny. Například nástroje pro práci s kolekcemi, anotacemi, cachováním, událostmi apod. Ty jsou pak využívány oběma vyššími vrstvami.
- **DBAL (DataBase Abstraction Layer)** - Zajišťuje databázovou nezávislost. Dokáže tudíž pracovat s vícero druhy databází, záleží pouze na konfiguraci.
- **ORM** - Jedná se o nejvyšší vrstvu a zajišťuje mapování objektů do relační databáze, jejich ukládání a načítání. Mapování je provedeno pomocí XML, YAML nebo anotací.

4.3 Potřebné optimalizace

Jelikož aplikace využívá externích zdrojů, kterých se musí doptávat HTTP komunikací, vzniká zpoždění na straně serveru a odpověď na uživatelský požadavek trvá déle. Toto zpoždění se zvyšuje počtem požadavků, které je přímo úměrné počtu projektů a počtu uživatelů v projektu. Také časem narůstá objem požadovaných dat. Dalším problémem, který se musí řešit je omezení ze strany nástrojů, se kterými aplikace komunikuje, na počet požadavků za určitou časovou jednotku. Pro příklad *Github* omezuje počet požadavků na 5000 na uživatele za hodinu.

Redis

Je úložiště datových struktur v paměti, které se používá jako distribuovaná databáze klíč → hodnota s volitelnou expirační dobou. *Redis* podporuje různé druhy abstraktních datových struktur, jako jsou řetězce, seznamy, mapy, sady...[3].

Použití v kontextu se Symfony frameworkem tvoří dobrý a jednoduše použitelný systém pro ukládání cache dat do paměti. Konfiguračně se dá nastavit url adresa a port, pod kterým *Redis* server běží. Pro Windows operační systém existuje pro *Redis* alternativa s názvem *Memurai*.

[1] Symfony roadmap - https://symfony.com/releases
[2] Doctrine - https://zdrojak.cz/clanky/doctrine-2-uvod-do-systemu/
[3] Redis - https://redis.io/

Asynchronous JavaScript and XML

V zkratce AJAX[1] je dobrým řešením pro práci s nadměrným množstvím dat. Základ stránky se v odpovědi serveru vrací bez zpoždění a data se následně získávají po tom, co klient obdrží odpověď. Pocitově tento přístup zkracuje čekací dobu.

4.4 Frontend

Jedná se zpravidla o HTML obohacené Javascriptem a CSS. Jak už bylo řečeno, HTML se vygeneruje přes Twig šablony:

```
{% extends 'base.html.twig' %}
{% block title %}{{ 'workspace'|trans({}, 'titles') }}{% endblock %}
{% block body %}
    <div class="container mt-5 mb-3">
        <div class="row">
            <div class="col"><h1 class="title">{{ 'projects'|trans({}, 'table-titles') }}</h1></div>
            <div class="col text-right">
                <a class="btn btn-primary" href="{{ path('project_create') }}">{{ 'add_project'|trans({}, 'actions') }}</a>
            </div>
        </div>
    </div>
    <section>
        {% include 'workspace/invitations_table.html.twig' %}
    </section>
    <section class="py-1">
        {% include 'workspace/projects_table.html.twig' with {'projects' : projects, 'tableTitle' : 'projects'|trans({}, 'table-titles')} %}
    </section>
{% endblock %}
{% block footer %}{% endblock %}
```

Obrázek 4.1: Vytváření HTML souboru pomocí Twigu

Můžeme vidět rozšíření šablony *base.html.twig*, konkrétně se jedná o dashboard, ve kterém uživatel vidí projekty a pozvánky ke spolupráci na projektech. Šablona *base.html.twig* obsahuje základní rozložení stránky jako menu nebo vykreslování systémových hlášek, zahrnuje potřebný CSS a Javascript kód. Dále pomocí klíčového slova *include* se zde zahrnují tabulky projektů a pozvánek.

Javascript a CSS jsou uloženy ve složce v kořenovém adresáři *assets*. Zde je kód do modulů a ty pak spolu tvoří potřebný celek pro konkrétní stránku. Moduly jsou propojeny a je vygenerován kód podporován i staršími prohlížeči pomocí nástroje *Webpack*:

```
const projectForm = require('./js/project-form');
const dates = require('./js/dates');
const taskPromotionFilter = require('./js/task-promotion-filter');
const language = require('./js/language');

$(document).ready(() => {

    selectize.init();
    tooltips.init();
    tool.init();
    invitation.init();
    datepicker.init();
    taskNote.init();
    taskPromotion.init();
    startTask.init();
    timerInfo.init();
    removeTask.init();
```

Obrázek 4.2: Javascript soubor sdružující menší moduly

Je tedy možné psát logicky rozdělený přehledný kód nejnovější syntaxí, který se až následně překládá. Takto přeložený kód se následně vkládá do HTML:

[1] AJAX - https://www.w3schools.com/js/js_ajax_intro.asp

```
{% block javascripts %}{{ encore_entry_script_tags('app') }}{% endblock %}
```

Obrázek 4.3: Vložení Javascriptu do HTML pomocí Twig syntaxe

Jednotlivé kontrolery generují z šablon HTML tak, že je nejdříve naplní potřebnými daty:

```
public function initInvite(Project $project): Response {

    $form = $this->createForm( type: InvitationType::class,  data: null, ['with_rows' => false]);
    return $this->render( view: 'workspace\project\create_invite.html.twig',
        [
            'form' => $form->createView(),
            'project' => $project
        ]
    );
}
```

Obrázek 4.4: Jednoduchý kontroler

Zde kontroler vrací stránku, na které uživatel může pozvat další uživatele ke spolupráci na projektu. K vygenerování správného HTML šablona potřebuje informace o projektu a formulář pozvánky.

4.5 Komunikace s nástroji

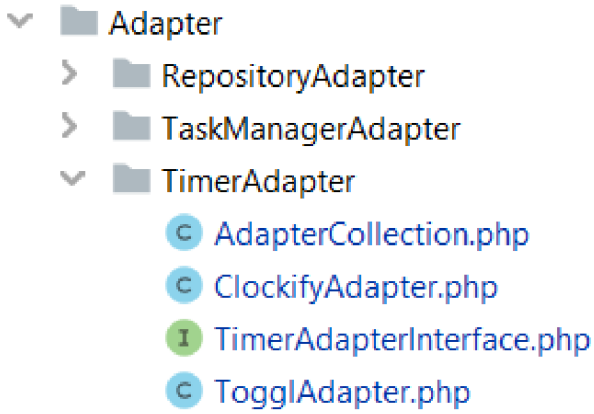
Při vytváření komunikačního rozhraní se muselo myslet na velkou dávku abstrakce, aby se pak mohlo jednoduše pracovat s konkrétními implementacemi. Nejlépe vysvětleno na příkladu:

```
$repositoryHandler = $this->repositoryHandlerCollection->getHandler($userRepository);
$repositoryHandler->createProject($userRepository,  &: $project);
```

Obrázek 4.5: Vytvoření projektu na nástroji repozitáře

Zde je kód zodpovědný za vytvoření projektu ve zvoleném repozitáři při vytváření entity projektu. Nezáleží na konkrétní implementaci, zda-li si uživatel přeje využívat *Github* nebo *Gitlab*. Pro každý podporovaný nástroj je vytvořena služba, která obsahuje konkrétní implementaci. Tato skutečnost je ale skryta. Jediné, co je vidět je to, že se z kolekce takovýchto implementací vybere ta správná a v ní se zavolá konkrétní funkce. Toto je zrovna příklad pro vytvoření projektu, jiná komunikace je ale řešena úplně stejně.

V konkrétní implementaci je už řešena logika přes HTTP komunikaci, kdy se z nástroje vezme *uri* a *access token* a volá se aplikační *endpoint*. Také se zde zachytávají chybové stavy.

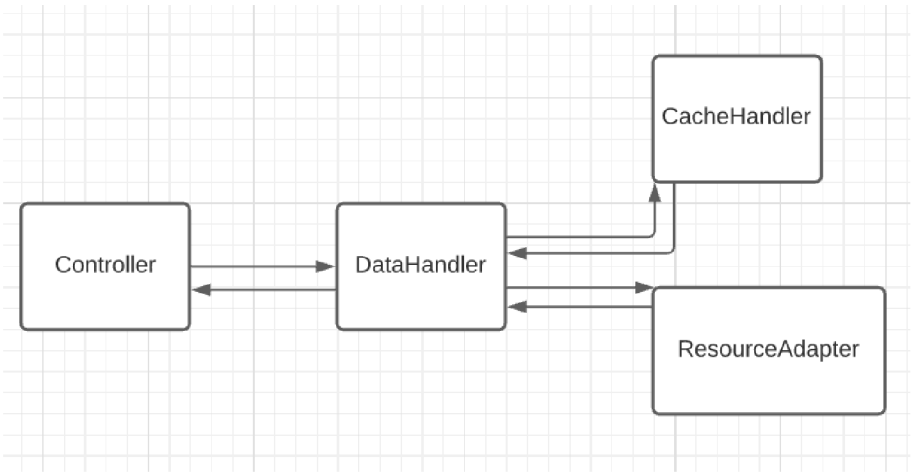


Obrázek 4.6: Adaptéry

ClockifyAdapter a *TogglAdapter* jsou konkrétní implementace, *TimerAdapterInterface* je rozhraní diktující vzor konkrétních implementací a *AdapterCollection* je služba pro vyhledávání konkrétních implementací. Nástroje jsou plně rozšiřitelné, firma používající tento software si může definovat vlastní konkrétní implementace. Stačí, aby nová třída vycházela z rozhraní *TimerAdapterInterface*. Celý systém bude s novým časovačem plně kompatibilní.

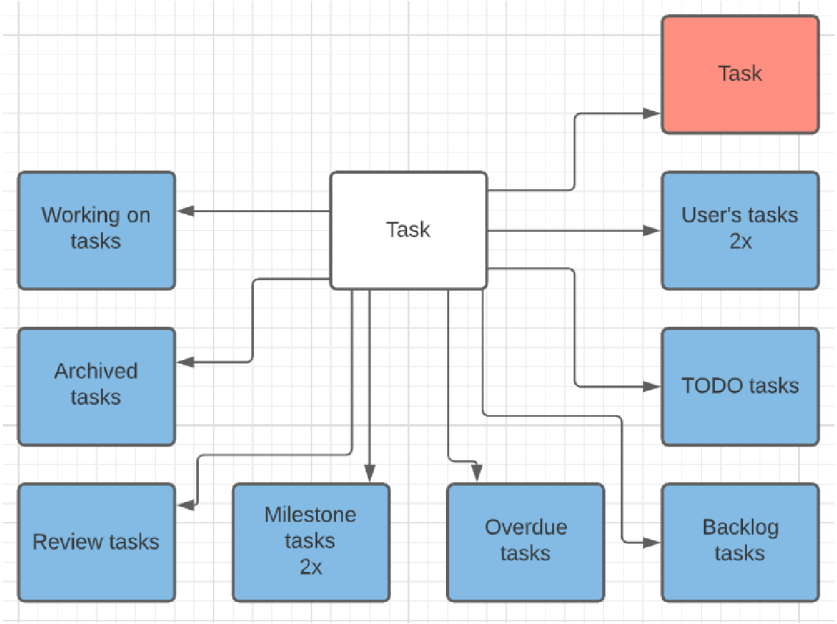
4.6 Cache

Ukládání informací z HTTP odpovědí do cache je brilantní způsob zkrácení čekací doby. Z několika sekundové akce se může stát záležitost pár milisekund. Pro implementaci práce s cache je v projektu zařazen nástroj *Redis*. Aplikace očekává *Redis* připojení na localhostu, tudíž stejném serveru a portu 6379. Tyto parametry jsou však nastavitelné. Umění práce s cache avšak není o tom která data ukládat, jak často nebo v jakém objemu, ale v mazání cache. Jednoduše se totiž může stát to, že uživateli budou posílána neaktuální data. Byl tedy vypracován způsob ukládání a mazání dat a práci cache přes oddělené komponenty. Ty kontrolují datové části, ukládají načtené informace z externích zdrojů do cache a při vytváření nebo editaci dat se maže jen nezbytně potřebná část cache paměti:



Obrázek 4.7: Komunikace služeb obsluhy dat

Kontroler získává data ze služby *DataHandler*, který obsluhuje cache a externí zdroje. Nejdříve se pokusí najít požadované data v cache paměti. Po selhání si je vyžádá z externího zdroje a následně data zapíše do cache. Ty pak v příštím požadavku v paměti najde.



Obrázek 4.8: Závislá cache

Zde je zobrazeno možné ovlivnění paměti cache na základě editace úkolu. Červenou je vyznačena cache, kterou je nutné vymazat. Modrou je znázorněna ta část paměti, které vymazání záleží na obsahu editovaného objektu. Je patrné, že mazání takového množství pokaždé, když se změní objekt úkolu není žádoucí. Veliká část může být ušetřena a také je díky rozpoznávání změn nad objekty.

Pro příklad, pokud se jedná o akci editování zdrojů, kontroler zavolá akci editace v službě *DataHandler*. Ten z editovaného objektu (třeba úkolu) extrahuje potřebné klíče do cache, které pošle do služby *CacheHandler* a ten je označí jako nevalidní. Pokud třeba úkolu bylo změněno přiřazení pracovníka a byl přeřazen do jiného milníku, z cache se vymaže list úkolů pracovníka, list úkolů daného milníku a data o úkolu samotném. Byla tedy označena jako nevalidní pouze ta část paměti, která byla reálně ovlivněna změnou objektu.

Kontroler ale zůstává celé této logiky ušetřen, zůstává tedy přehledný:

```
$this->dataHandler->edit( type: DataHandler::TASK, $membership, $task, $cloned);
```

Obrázek 4.9: Zde je volání editace úkolu, kde první parametr je zadaný typ objektu, který se edituje, druhý je objekt typu *Membership*, ze kterého *DataHandler* čerpá informace o externím zdroji, *$task* je zeditovaný objekt a *$cloned* je objekt před editací pro porovnání.

Získávání dat vypadá v kontroleru následovně:

```
$milestones = $this->dataHandler->getData( key: null, $membership, type: DataHandler::MILESTONES);
```

Obrázek 4.10: Akce získávání milníků v kontroleru

Klíč je parametr sloužící pro identifikaci konkrétního objektu, třeba ID úkolu. Zde ale získáváme data o všech milnících, klíč tedy není potřeba. V pozadí se děje celý proces s integrací cache a voláním externích zdrojů. Zase platí, že kontroler je od této logiky odstíněn a tudíž zůstává přehledný.

4.7 Základní obsluha systému

Registrace nástroje

Registrovaným nástrojem se myslí uložená reference obsahující *access token* pro API komunikaci. Uživatel si vybere, který typ nástroje chce registrovat. Následně zadá *url* a *token*. Před uložením je vyzván na otestování nástroje. Jedná se o pokus spojení se službou. Pokud vrátí HTTP status 200, považují se poskytnuté informace za validní a uživatel si může nástroj registrovat. Takový nástroj bude mít ve svém seznamu použitelných nástrojů a může ho použít při vytváření nových projektů nebo pomocí něj přijmout pozvánku.

Vytvoření projektu

Probíhá přes rozhraní aplikace, deleguje se na nástroje, které se uživatel rozhodl použít. Pomocí API komunikačního rozhraní se pošle požadavek na vytvoření projektu nesoucí název, který uživatel zadal do formuláře. Z odpovědí se získávají potřebné informace pro zpětné adresování projektu, jako ID nebo url safe název v případě služby *Github*.

V konečném důsledku je tedy projekt vytvořen přes delegaci na všech zvolených nástrojích a v aplikaci se uchovávají pouze referenční data. Musí se ale ohlížet na množství krajních případů, kdy taková delegace selže, třeba projekt se stejným názvem už existuje nebo *access token* přestal být platný.

Takový projekt se zobrazí v seznamu projektů a uživatel může začít vývoj.

Pozvánky

U vytvořeného projektu může správce pozvat další uživatele ke kolaboraci. Vytvoří pozvánku, která nese relaci na pozvaného, jeho roli a odkaz na projekt. Pozvaný na dashboardu aplikace pozvánky vidí a může je přijmout nebo odmítnout. U přijetí platí podmínka, že musí mít registrované nástroje, které projekt používá. Pokud je nemá, je vyzván na jejich vytvoření.

Po přijetí je přidán ve všech službách, které projekt používá jako kolaborant. Přes API správce se zašlou pozvánky a hned se i přijmou pomocí API pozvaného. Nový uživatel je zařazen do projektové *group* podle role.

Administrace

Řešena pomocí balíku *Sonata admin*. Umožňuje jednoduchou implementaci administrace pro systémového administrátora. Není proto pro něj nutné Chodit do databáze, pokud

vznikne potřeba změnit nějakou hodnotu, která z běžného uživatelského rozhraní není dostupná. Je implementována v podobě správy jednotlivých objektů entity a jejich hodnot. Tudíž administrátor bude mít přístup ke každému záznamu databáze ve formě objektu. Hlavním přínosem je implementace aplikačních omezení, která by se v databázovém systému těžko implementovala.

4.8 Uživatelské Testování

Po vytvoření prvních reálných pohledů webové aplikace bylo nutné validovat jejich UX řešení. Pohledy vycházely z předem vytvořených mockupů na papíře. Byly uzpůsobeny uživateli v jeho roli, tudíž existuje více verzí. Bylo potřebné sehnat více lidí pro role, ke kterým mají nejblíže. Kritické jsou role týmový manažer a vývojář.

Experimenty

Pomocí nemoderovaného testování byly testované následující osoby:

Ludo Jambrich pro roli týmový manažer
Ukázal se jako vynikající volbou, jelikož sám UX testování prováděl už mnohokrát - jako tester i jako testovaný. Uměl vyjadřovat své myšlenky a tím poskytl kvalitní zpětnou vazbu.
Andrej Zbín pro roli vývojář
Programátor s dloholetou praxí s podobnými systémy a se spoluprací na projektových řízeních. Zná obecně uznávané postupy, dalo se tedy očekávat, že rychle zjistí co, jak a proč v systému funguje. Neměl zkušenosti s uživatelským testováním, musel být oznámen o aspektech jeho části vstupu.
Filip Markovič pro roli vývojář
Junior programátor, pracoval na projektech toho typu, pro který je systém vytvořen. To byl důvod mojí volby pro něho jako testované osoby. Pracoval se všemi typy nástrojů, které aplikace sdružuje. Také ale neměl zkušenosti s uživatelským testováním.

Každá testovaná osoba byla přizvána 3 krát na online schůzku, u které plnila připravené úkoly. Ze schůze byly pořizovány záznamy, ze kterých se posléze vyhodnocovaly závěry. Úkoly byly navrženy tak, aby sledovali konkrétní vlastnosti systému - srozumitelnost, dostupnost akcí a informací, vypěstování návyku na systém a přehlednost. Úkoly sledovaly scénář, ve kterém jsou osoby v jednom týmu a mají rozpracovaný projekt stránky online obchodu. Všechny testované osoby také vědí, co takový projekt obnáší. Tím je myšleno projekt krátkodobý a šablonovitý. Každá schůzka pojímala 2 až 3 úkoly.

Časy trvání plnění úkolů byly zpětně změřeny. Výsledné časy jsou zaznamenané v následující tabulce:

	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
Jambrich	58	60	25	18	45		15	36
Markovič	33	23	30	21	20	16	14	21
Zbín	57	12	15	14	45	29	14	14

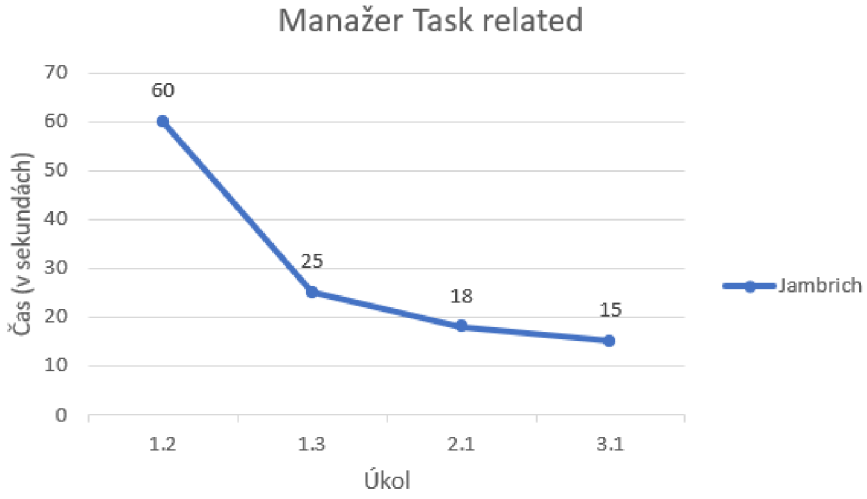
Kde řádky značí testovanou osobu a sloupce označují číslo úkolu ve tvaru X.Y, kde X je číslo schůzky a Y je číslo úkolu na schůzce. Data v tabulce jsou normalizované časy trvání splnění úkolu v sekundách:

$$\text{Normalizovaný čas} = \frac{\text{Čas splnění úkolu}}{\text{Počet akcí}}$$

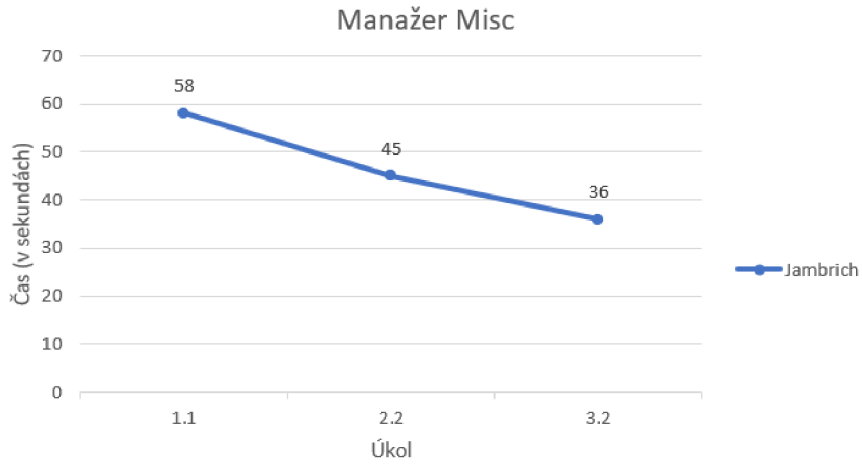
Dále jsou barevně rozlišeny data podle typu úkolu, kde žlutou barvou jsou označeny úkoly související s prací na projektových povinnostech (tzv. Task related) a modrou všeobecná správa projektu (Misc), například přidání nového pracovníka do týmu u Manažera, nebo čtení tabulky o platu u Vývojáře. Zelené jméno je Developer, Šedé Manažer. Každý z úkolů disponoval několika akcemi, které musel uživatel provést, aby úkol splnil:

Z takových dat u těchto 2 kategorií (Task related a Misc) se dá vysledovat zvyk uživatele na systém, nebo jeho schopnost pohybovat se po systému a vyznat se. Sledované vlastnosti jsou klíčové pro použitelnost systému. Pozvolné klesání normalizovaného času u úkolů by naznačovalo uživatelovu tendenci zvykat si na aplikaci. Podobné časy by naopak signalizovaly neschopnost vidět v systému pořádek nebo řád. Zvyšování časů by znamenalo pouhé štěstí, že uživatel dokončil úkol prvně v kratším čase. Poté se mu to už nepodařilo - projev zmatku a absence řádu v aplikaci.

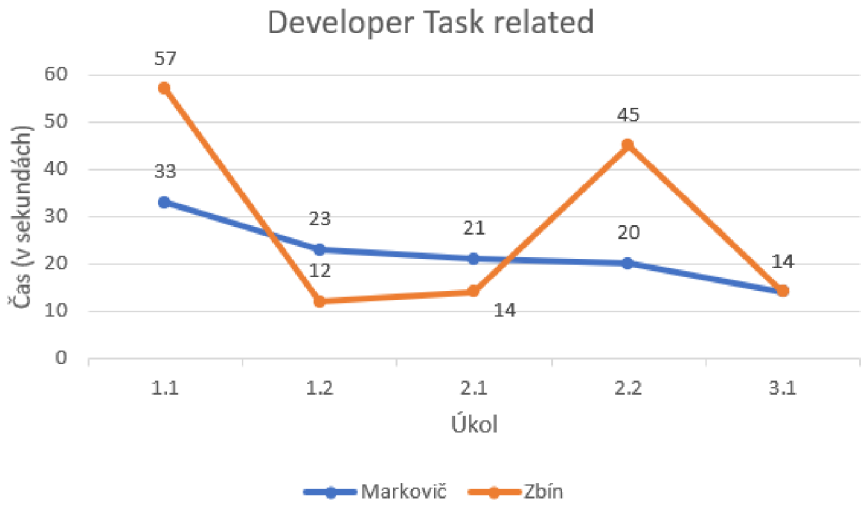
Normalizované data byly rozděleny do následujících grafů dle role a kategorie úkolů:



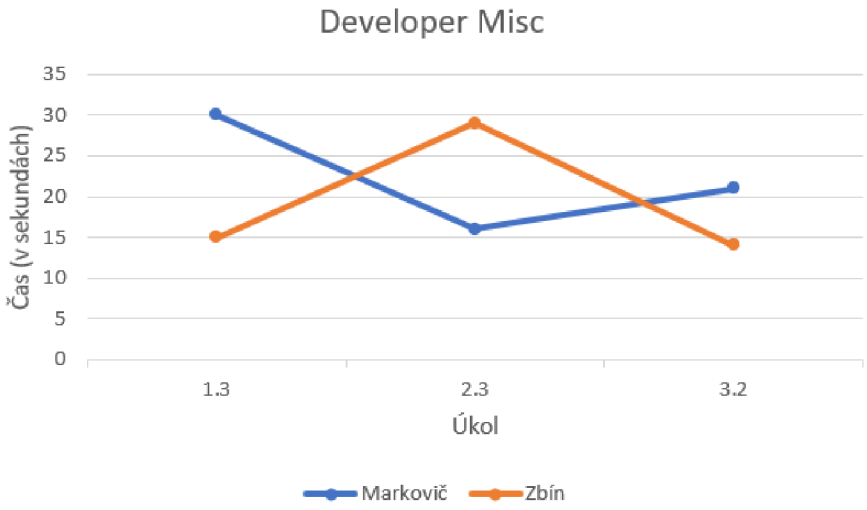
Obrázek 4.11: Výsledky role týmový manažer v kategorii Task related



Obrázek 4.12: Výsledky role týmový manažer v kategorii Misc



Obrázek 4.13: Výsledky role vývojář v kategorii Task related



Obrázek 4.14: Výsledky role vývojář v kategorii Misc

Vyhodnocení

U Manažera (Jambrich) je značný chronologický pokles potřebného času mezi jednotlivými akcemi k splnění úkolu. V grafu Task related je vidět jasný postup, hodnota posledního měření je 4 krát nižší, než hodnota měření prvního. Uživatel si za 3 sezení dokázal vypěstovat návyk na systém, dokázal v něm najít řád. Ve videozáznamech je patrný jeho menší potřebný myšlenkový vstup. Ze začátku komentoval, co vidí, kde si myslí, že hledanou akci nebo informaci najde. Později tyto komentáře přeskakoval, protože systém už znal. Stačilo mu s aplikací menší seznámení. Tuto skutečnost odráží i graf Manažer Misc, kde jsou záznamy časů úkolů, které se znalostí systému počítaly.

U role vývojář (Markovič a Zbín) je patrný pokles v grafu Task related mezi záznamy 1.1 a 1.2. Dále tyto časy ještě klesly u třetího záznamu, kde byl představen nový widget pro obsluhu rozdělané práce na projektu. U záznamu 2.2 je patrné váhání uživatele Zbína. Časy se postupně ustalovaly, první úkol zabral mnoho krát více času než úkol poslední.

V grafech Misc jsou normalizované časy v průměru o 5 vteřin delší než v grafech Task related. To je ale dobře, protože Task related grafy zobrazují akce, které se provádí v systému primárně a častěji, tudíž jsou vystaveny více na očích.

Osobám bylo zadáno, aby celý čas mluvily, o čem zrovna přemýšlí, jaký mají dojem a co od akce, kterou provádí, očekávají. V mluveném komentáři myšlenkových pochodů každé osoby byl časem patrný pokles potřeby zkoumat prostředí systému. Jejich pozornost se přenesla z prostředí aplikace do práce, kterou v aplikaci provádí.

Navigace je navržená dobře. Odpovídá uživateli na dvě základní otázky: Jak se kam dostanu a kde se právě nacházím. Testovaní neměli problém se po systému navigovat. Také nízké užití textů a větší důraz na členění obsahu pomocí nadpisů a sekcí vyhovuje brodící taktice uživatelova skenování webu. Tato podpořená taktika mu dovoluje úplně vynechat obsah, na který nepotřebuje poutat pozornost. Například u sezení s Andrejom Zbínom, úkolu 1.1 se rozhlížel po rozhraní a našel i tabulku s se svými odpracovanými hodinami. Její obsah ale ignoroval, jelikož úkol s ní nesouvisí. Až u úkolu 1.3 se na její obsah zaměřil, sekci už znal a zpětně ji jednoduše našel.

V průběhu testování se objevilo několik jasných chyb, které se mohly opravit přímo bez kontinuálních analýz. Způsobovaly zmatek a přispěly k nepřehlednosti systému. Odstranění chyb se provádělo paralelně mezi testovacími schůzemi, mohl být tedy vysledován pozitivní/negativní dopad změny. Jednalo se především o pozitivní zpětnou vazbu:

- Úpravy widgetu pro výběr data. Přeškrtnutí starých, nevalidních dat, zvýraznění dnešního data. Jambrich se chtěl orientovat podle dnešního data, nebylo však zvýrazněno v kalendáři.
- Označení povinných položek ve formuláři.
- Přidán vizuální náznak pro widget s povyšováním úkolů, konkrétně pro možnost drag and drop.
- Widget se zapnutým časovačem pro úkol, na kterém programátor pracuje, byl přesunut z menu položky do nové lišty pod menu. Ani Markovič, ani Zbín si starého widgetu po jeho vytvoření nevšimli.
- Projektové menu nyní nese název projektu. Uživatel tedy ví, zda-li je v správném prostředí
- V archivu, skladišti starých a splněných úkolů, byla změněna tabulka úkolů, aby poskytovala informace, pro které uživatel do archivu v první řadě půjde
- Přidán filtr nad úkoly do widgetu s povyšováním úkolů. Nyní otevřenější škálovatelnosti
- Vylepšeny popisy zvolených nástrojů. Když uživatel vytváří projekt, volí si nástroje, kterými ho chce spravovat. Z voleb ale nebylo jasné, který nástroj volí nebo co vůbec volba nástroje znamená.
- Možnost přidat úkol přímo z detailu milníku, kterému uživatel chce úkol přiřadit
- Možnost přesunu stavu úkolu mezi Backlog a TODO přímo z detailu úkolu
- Přidán prefix “Milník” u detailu milníku. Vývojáři nebylo zřejmé, že se nachází v detailu milníku
- Znefunkčnení tlačítek během provádění akce jako indikace toho, že akce probíhá

Kapitola 5

Závěr

Podle zadání bylo cílem práce vytvořit systém ulehčující práci při vývoji software pomocí automatizací podpůrných nástrojů a poskytování informací z nich extrahovaných.

Na počátku práce byl realizován pilotní průzkum, který pomohl zúžit cílovou skupinu uživatelů. Mimo jiné mi pomohl podchytit generalizovaný postup vývoje software pro rychlé projekty na zakázku, třeba internetové obchody. Podařilo se vytvořit webovou aplikaci, pomocí které uživatel komunikuje s podpůrnými nástroji a spouští několik logicky souvisejících akcí zároveň. Uživatel už nemusí používat nástroje jednotlivě. Už ve fázi vývoje bylo myšleno na další rozšíření, aplikace tedy s nimi komunikuje pomocí adaptérů. Důsledkem je, že je možné jednoduše implementovat další nástroje podle vlastního výběru a tvořit různé kombinace užití při vývoji projektů. Program má v základu balík pro demonstraci implementace, obsahující adaptéry dvou nástrojů repozitáře, dvou nástrojů správce úkolů a dvou nástrojů časovače. Komunikace samotná byla zrychlená užitím dynamického systému cache.

Vytvoření uživatelského prostředí předcházely konzultace a průzkum na uživatelích, kteří pracují v oboru, tedy programátoři a týmoví manažeři. Pro validaci UX bylo provedeno nemoderované testování, které přineslo pozitivní výsledky týkající se adaptace na systém, použitelnosti a také odhalilo relevantní chyby, které se tak podařilo včas opravit.

Přínosem systému je urychlení práce na projektech a zisk pohodlí, kdy uživatel nemusí obsluhovat různé nástroje jednotlivě, ale používá jeden, který je sdružuje. Tento komfort je zaručen vícero implementacemi nástrojů. Kdyby byl implementován pouze jeden nástroj z každé kategorie a uživatel by si nemohl vybrat, práce by byla k ničemu, pokud by preferoval jiný nástroj.

V práci bych chtěl pokračovat tím způsobem, že bych implementoval další nástroje pro rozšíření základního balíku. Rozšiřování základního balíku by bylo možné pomocí instalovatelných rozšíření přes Composer, správce PHP závislostí. Tím by se staly veřejně dostupné a ke každé instanci projektu by si firma mohla vybrat nástroje, které chce používat.

Literatura

[1] *Caching Overview* [online]. Amazon, duben 2021 [cit. 2021-05-03]. Dostupné z: https://aws.amazon.com/caching/.

[2] Feoktistov, I. *What Agile Software Development Team Structure Looks Like* [online]. RELEVANT, březen 2019 [cit. 2021-05-03]. Dostupné z: https://relevant.software/blog/what-agile-software-development-team-structure-looks-like/.

[3] Joshi, R. *Java Design Patterns*. 1. vyd. Java Code Geeks, 2015.

[4] Koďousková, B. *ARCHITEKTURA MVC: DEFINICE, STRUKTURA, FRAMEWORKY* [online]. Rascasone, duben 2021 [cit. 2021-05-03]. Dostupné z: https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky.

[5] Krug, S. *Don't make me think, revisited : a common sense approach to web usability*. 1. vyd. San Francisco : New Riders, 2014. ISBN 978-0321965516.

[6] Kultová, A. *UX design* [online]. WikiKnihovna, březen 2021 [cit. 2021-05-03]. Dostupné z: https://wiki.knihovna.cz/index.php/UX_design.

[7] Picurelli, L. *Advantages and disadvantages of web app development* [online]. LinkedIn, květen 2017 [cit. 2021-05-03]. Dostupné z: https://www.linkedin.com/pulse/advantages-disadvantages-web-app-development-luis-picurelli.

[8] *10 Popular Usability Testing Methods* [online]. Playbook UX, duben 2020 [cit. 2021-05-03]. Dostupné z: https://www.playbookux.com/10-popular-usability-testing-methods/.

[9] Rungta, K. *What is Usability Testing? UX(User Experience) Testing Example* [online]. GeeksforGeeks, březen 2021 [cit. 2021-05-03]. Dostupné z: https://www.geeksforgeeks.org/what-is-web-api-and-why-we-use-it/.

[10] Urban, J. *KROKY PŘI PŘÍPRAVĚ A REALIZACI DOTAZNÍKOVÉHO ŠETŘENÍ* [online]. Centrum pro otázky životního prostředí UK - Univerzita Karlova, 2005 [cit. 2021-05-03]. Dostupné z: https://www.czp.cuni.cz/czp/images/stories/Vystupy/Seminare/2005%20LS%20Ocenovani%20ZP/urban_priprava_dotazniku.pdf.

[11] *Web application* [online]. Tech Target, srpen 2019 [cit. 2021-05-03]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app.

[12] *What is Web API and why we use it?* [online]. GeeksforGeeks, květen 2020 [cit. 2021-05-03]. Dostupné z: https://www.geeksforgeeks.org/what-is-web-api-and-why-we-use-it/.

Příloha A

Otázky pilotního průzkumu

1. Pracujete v týmu? Případně jak velkém?
2. Jaká je vaše pracovní pozice?
3. Měníte pracovní procesy vývoje software? Snažíte se je vylepšovat?

Tyhle první 3 otázky jsou pouze za účelem určení váhy výpovědi

5. Formality kolem zakládání projektu a vytváření pracovního prostředí jsou často zdlouhavé. Recyklujete tyto činnosti nějakým konkrétním způsobem třeba projektovými šablonami, automatizovaným systémem a podobně?
6. Projektové řízení vyžaduje detailní zprávy o vykonané práci kvůli zodpovězení různých otázek (Bylo splněno zadání? Jak teď vypadá projektová struktura? Kolik času se strávilo prací?), což vyžaduje zápisy do vícero systémů. Můj projekt má mimo automatizace za účel tyto informace poskytovat na jednom místě. Jak by jste vnímali tento přínos? Jakou významnost by jste mu připsali od 1 zlý přínos po 10 velmi dobrý přínos?
7. Váš běžný pracovní postup vývoje v celém životním cyklu projektu je... Podívejte se na následující obrázek 3.1. Porovnejte. Co říkáte na navrženou automatizaci? Na jednotlivé bloky? Chybí něco, je něco navíc... ?

Příloha B

Obsah přiloženého paměťového média

- **application** – Složka se zdrojovými kódy aplikace
- **poster** – Složka s plakátem ve formátu PDF
- **projekt.pdf** – text bakalářské práce.
- **video** – Soubor s prezentačním videem.

Příloha C

Úkoly nemoderovaného UX testování

1. Iterace:
   Manažer

   1.1. Andrej má přidáno o 5 €, přidej mu a zkontroluj projektový rozpočet

   1.2. Pokud některý z pracovníků nemá naplánovaný úkol, přidej z Backlogu

   1.3. Vytvoř úkol *Edit profil* do milníku *Profil užívateľa* pokud tam takový úkol ještě není

   Programátor

   1.1. Pokračuj v práci na úkolu *X* a dokonči jej

   1.2. Začni a dodělej úkol *Edit profil*

   1.3. Zjisti, kolik peněz můžeš očekávat na výplatní pásce za tento měsíc

2. Iterace:
   Manažer

   2.1. Pokud je nějaký úkol deadline, uprav její deadline tak, aby měl programátor čas ještě týden

   2.2. Přidej do projektu 3. programátora *Janko*

   Programátor

   2.1. Začni pracovat na novém úkolu *X*, nedodělej jej, pouze rozpracuj, dnes jsi nestihl

   2.2. V práci se probírá úkol *Platobná brána*, na kterém se pracovalo v minulosti a je dokončen. Jsi zvědavý, jaké bylo zadání

   2.3. Vytvoř svůj boční projekt *Škola projekt2*, jsi v něm týmový manažer a máš spolužáka *Janko*, který plní úkol vývojáře

3. Iterace:
   Manažer

   3.1. Podívej se na procentuální stavy všech milníků, přesuň úkol z Backlogu do TODO pro milník, který nejvíc zaostává

3.2. Pokud některý z pracovníků nemá naplánovaný úkol, přidej z Backlogu

3.3. Zjisti, zda-li jsou nějaké nedávno dokončené úkoly, resp. zda-li vývoj probíhá hladce, (3 a víc nedávno dokončených úkolů = vývoj probíhá hladce)

Programátor

3.1. Vyber úkol s podle tebe největší prioritou a dokonči jej, na revizi dej Andreje

3.2. Okomentuj kterýkoliv úkol z milníku *Profil užívateľa*