VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO SPRÁVU UNIVERZITNÍCH PROJEKTŮ

WEB APPLICATION FOR MANAGEMENT OF UNIVERSITY PROJECTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VIKTOR SHAPOCHKIN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Shapochkin Viktor**
Program: Informační technologie
Název: **Webová aplikace pro správu univerzitních projektů**
Web Application for Management of University Projects
Kategorie: Informační systémy

Zadání:

1. Prostudujte problematiku správy univerzitních projektů. Prozkoumejte existující nástroje určené pro tento účel (např. Ganttův diagram) a aplikace implementující dané nástroje.
2. Prostudujte existující typy vizualizací vhodné pro návrh přehledových obrazovek typu *dashboard.* Prostudujte principy tvorby informačních systémů, webových uživatelských rozhraní a vizualizací dat.
3. Definujte požadavky aplikace řešící problematiku z bodu 1 a srovnejte je s existujícími aplikacemi a službami. Zaměřte se na požadavky uživatelů pro snadnou tvorbu Ganttových diagramů.
4. Dle požadavků z bodu 3 navrhněte webovou aplikaci pro správu univerzitních projektů.
5. Navrženou aplikaci implementujte.
6. Otestujte řešení.

Literatura:

- Johnson, J.: *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier, 2010, ISBN: 978-0-12-375030-3.
- Full stack open 2020: *Deep Dive Into Modern Web Development* [online]. 2020 [cit. 2020-10-02]. Dostupné z: https://fullstackopen.com/en/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

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

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Tato bakalařská práce se zabývá návrhem, implementací a testováním webové aplikace pro správu a řízení univerzitních projektů. Aplikace umožnuje uživatelům vytvářet, sledovat a upravovat jednotlivé projekty. Běh a stav projektů je reprezentován pomocí Ganttova diagramu. Aplikace je implementována v jazyce JavaScript a je rozdělená na dvě časti – server a klient. Server běží na platformě Node.js s použitím rozhrání Express.js a databáze MongoDB. Klientská část aplikace je implementována pomocí JavaScript knihovny React.

Abstract

This bachelor thesis deals with the design, implementation and testing of a web application for the administration and management of university projects. The application allows users to create, monitor and edit the projects. The progress and status of projects is represented by a Gantt Chart. The application is implemented in JavaScript and is divided into two parts—server and client. The server runs on the Node.js runtime environment using the Express.js interface and the MongoDB database. The client part of the application is implemented using the React JavaScript library.

Klíčová slova

informační systém, webová aplikace, správa projektů, server, klient, JavaScript, React, MongoDB, Node.js, Express.js, MERN, Ganttův diagram

Keywords

information stystem, web aplication, project management, server, client, JavaScript, React, MongoDB, Node.js, Express.js, MERN, Gantt Chart

Citace

SHAPOCHKIN, Viktor. *Webová aplikace pro správu univerzitních projektů.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Webová aplikace pro správu univerzitních projektů

Prohlášení

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

.....
Viktor Shapochkin
11. května 2021

Poděkování

Rád bych poděkoval panu Ing. Jiřímu Hynkovi, Ph.D. za odborné vedení mé bakalařské práce, trpělivost a čas strávený konzultacemi.

Obsah

Kapitola 1

Úvod

Řízení a správa projektu jsou vždy spojeny s problémem efektivního využití času. Vždy je nutné správně naplánovat časové intervaly pro jejich vypracování, odhadnout konečné termíny, a hlavně mít před sebou vizuální přehled všech událostí co se v rámci projektu uskuteční. Často se ale stává, že v rámci větších společností a institucí vzniká nutnost najednou řídit sadu několika různých projektů. V tomto případě se jedná o programové řízení. Projektové a programové řízení jsou dvě metodiky řízení projektů, které na sebe navazují, ale každá z nich má svůj význam, a proto bude v této bakalářské práci také popsán jejich rozdíl.

Správa a řízení projektů je aktualní téma v různých oblastech a oborech a týká se také univerzit. Velké množství univerzit má zájem o spolupráci se společnostmi a organizacemi, které působí ve stejném nebo podobném oboru. V rámci této spolupráce se uskutečňuje velký počet různých projektů a akcí, a vzniká tak požadavek na jejich správu, řízení a plánování. Ve výsledku taková spolupráce je přínosná pro obě strany, které tak dosahují svých strategických cílů.

Cílem této bakalářské práce je vytvořit aplikaci pro plánovaní, správu a řízení univerzitních projektů pomocí Ganttova diagramu. Výsledkem by měla být jednoduchá, srozumitelná a uživatelsky přívětivá aplikace, která by umožňovala programovému manažeru dohlížet na jednotlivé projekty a patřící k nim událost. Práce je implementována jako webová aplikace, aby bylo možné se k ní dostat z jakéhokoliv zařízení včetně mobilu a tabletu.

Tato bakalářská práce je rozdělená do dvou časti – teoretická a praktická. Teoretická část obsahuje několik kapitol. Kapitola 2 popisuje základní principy a techniky, které se používají při vývoji a řízení projektů. Popis technologií, včetně těch, které byly použity při implementaci této práce, je uveden v kapitole 3. Kapitola 4 obsahuje analýzu požadavků a popis již existujících řešení. Návrh celého systému je obsažen v kapitole 5. Zde je představen první grafický návrh aplikace, popis jednotlivých částí uživatelského rozhraní a návrh struktury ukládání dat v databázi. Praktické části se věnuje kapitola 6, která obsahuje detailní popis implementace důležitějších části aplikace, jimiž jsou klient a server. Následující kapitola 7 se zabývá testováním výsledné aplikace a je zde popsán proces a výsledky uživatelského a automatického testování.

Kapitola 2

Řízení a vizualizace projektů

Řízení, plánování a správa projektů hrají v dnešní době velkou roli pro různé organizace a instituce, které chtějí dosahovat svých strategických, finančních a taktických cílů. Projekt jako celek je dostatečně složitá a dynamická úloha, kterou ovlivňuje mnoho faktorů, je ovšem nutné, aby byla úspěšně splněna a dokončena v termínu. Úspěšnost jejího řešení předpokládá využití správných postupů, technologií, metodik a nástrojů.

2.1 Projektové a programové řízení

Projekt

Projekt je klíčovou jednotkou projektového řízení. V literatuře lze najít velký počet definic tohoto pojmu.

Dle IPMA (International Project Management Association) standartu ICB v3.1: *„Projekt je jedinečný časově, nákladově a zdojově omezený proces realizovaný za učelem vytvoření definovaných vstupů (rozsah naplnění projektových cílů) v požadované kvalitě a v souladu s platnými standarty a odsouhlasenými požadavky“* [7]. Svozilová charakterizuje projekt jako řízený proces, který má svůj počátek, konec, přesná pravidla řízení a regulace, čímž se odlišuje od obyčejného sledu úkolů, jejíhž výsledek se nemusí v závěru setkat s očekáváním [20].

Projektvé řízení

Řízením a koordinaci projektů se zabývá *projektový management* (projektové řízení). Projektový management je nezbytný pro společnosti, které realizují své výstupy formou projektů. Cílem projektového řízení je co nejrychlejší a nejefektivnější dosažení plánovaného cíle projektu a to v rámci disponibilních zdrojů a stanovených termínů.

Podle Svozilové [20] může být úspěšný projektový management definován jako dosažení zamýšleného cíle projektu, a to při dodržení časového a nákladového limitu nebo jiných čerpaných zdrojů, s dosažením požadovaného cílového výkonu a s akceptací zákazníka projektu.

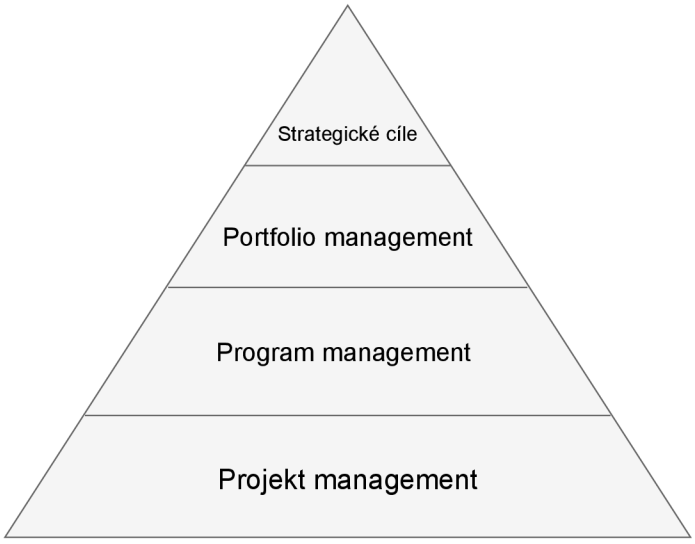
Programové řízení

Ze systémového úhlu pohledu lze najít mezi řízením projektu a programu mnohé paralely a obdobné postupy, oba „útvary“ se však nacházejí na odlišné úrovní detailu (vertikálně – projekt je subsystémem programu, obrázek 2.1) a řeší kvalitativně odlišné záležitosti [7].

Podobně jako projekt i program má určité fáze řízení, minimálně tyto: příprava, zahájení, průběh, ukončení a vyhodnocení. Podle PMI životní cyklus programu se definuje [7] takto:

- definice programu,
- dadávání přínosů programu,
- uzavření programu.

Organizační struktura programu může být podle velikosti a komplexnosti programu také velmi různorodá. V jednodušších případech stačí manažer programu. Je důležité ale říct, že role manažera programu je poněkud odlišná od role projektového manažera. Programový manažer usiluje o to, aby celá programová struktura a její procesy umožnily programu jako celku i jednotlivým komponentám úspěšnou realizaci a společné dodaly organizaci očekávané přínosy [7].



Obrázek 2.1: Organizace dosážení strategických cílů pomocí projektového, programového a portfolio managementu. Zdroj: [15]

2.2 Životní cyklus projektu

„Životní cyklus projektu je souborem obecně následných fází projektů, jejichž názvy a počet jsou určeny potřebami kontroly organizace, která je v projektu angažována.“ [20]

Životní cyklus projektu je nezbytný pro definování počátku a konce projektu a jeho jednotlivých fází, které na sebe postupně navazují. Formy životního cyklu projektu se mohou lišit podle odvětví, ve kterém daný podnik nebo organizace působí a také podle velikosti. Počet fází, na které lze projekt rozčlenit, závisí na rozsáhlostí a složitosti samotného projektu.

Doležal [6] přináší obecné rozdělení fázového modelu projektu na 3 základní fáze:

1. **Předprojektová fáze** – v této fázi organizace zkoumá příležitosti projektu a posuzuje jeho proveditelnost pomocí různých analýz a studií, zahrnuje se také formulování základních myšlenek a námětů na projekt.

   Hlavním výstupem předprojektové fáze by měla být odpověď na otázku, zdali má vůbec smysl projekt realizovat či nikoliv.

2. **Projektová fáze** – tato fáze bývá nejrozsáhlejší a zaměřuje se řízení a realizaci samotného projektu, skládá se zejména z:

   - zahájení projektu – ověření nebo definice cíle projektu, jeho účelu, požadovaných výstupů, sestavení projektového týmu apod.;
   - plánování projektu – definice rozsahu projektu, vytvoření plánu plánu řízení projektu a harmonogramu;
   - realizace projektu – řízení jednotlivých projektových činností, sledování průběhu projektu a jeho porovnání s plánem, v případě zjištění odchylek provádění korelačních opatření;
   - ukončení projektu – projekt je fyzicky i protokolárně ukončen, dochází k předání výstupů, uskutečnění fakturace a dalších náležitostí.

3. **Poprojektová fáze** – táto fáze má za účel analyzovat a vyhodnotit průběh projektu a naplnění jeho cílů. V případě nalezených chyb navrhnout opatření ke zlepšení pro příští projekty.

2.3 Vizualizace projektu

Vizualizace je nezbytnou částí pro uspěné plánování projektu. U rozsáhlých projektů může existovat velké množství jednotlivých úkolů a vazeb mezi nimi, a proto je pro lepší přehled vhodné to mít všechno reprezentované v grafické formě. Vizuální reprezentace projektu by měla splňovat určité požadavky, jako jsou například přehlednost a srozumitelnost. Existuje řada způsobů, jak lze pomocí různých vizualizačních nástrojů poskytnout jednotlivé informace o projektu a sledovat stav jejich vývoje. Takovými nástoroji jsou dashboardy a Ganttův diagram.

Dashboard

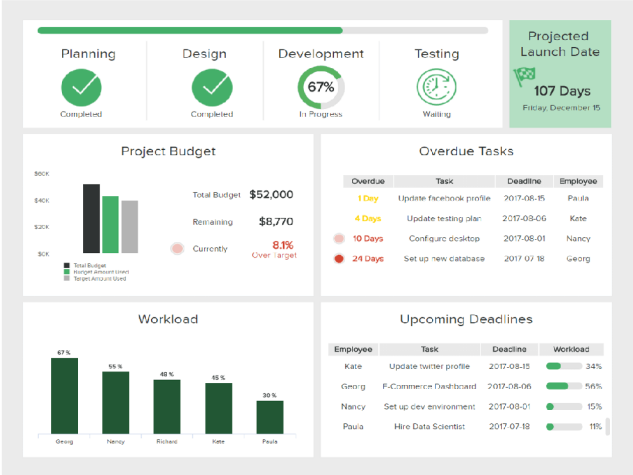
Dashboard, z anglického překladu „nástěnka“, je vizualizační *nástroj*, ve kterém jsou sdružený důležité metriky pro řízení a vykonaní jednotlivých operací nebo procesů. Především je kladen důraz na vhodnou grafickou reprezentaci dat. Správný dashboard by ale měl splňovat několik pravidel. Než ty pravidla budou zdůrazněny, je dobré se podívat do odborné literatury a najít vhodnou definici dashboardu. Jednu z takových vytvořil Stephen Few: *„Dashboard je vizuální zobrazení nejdůležitějších informací, které jsou potřeba pro dosažení jednoho nebo více cílů; sloučeny a uspořádány na jedné obrazovce tak informace lze sledovat na první pohled.“* [8]

Takovou definicí Stephen Few [8] odlišuje informační dashboardy od jiných forem prezentace dat a vyznačuje několik pravidel, které správný dashboard musí splňovat:

- Zobrazování všech dat tak, aby se vešly na jednu obrazovku.
- Důraz na grafickou reprezentaci při zobrazování dat.

- Poskytování nejnovějších dostupných dat.
- Srozumitelná forma.

Z hlediska vývoje a správy projektů dashboard může obsahovat informace například o celkovém počtu projektů, počtu aktuálně běžících projektů, rozpočtu a koncových termínech.



Obrázek 2.2: Příklad dashboardu, který obsahuje nejdůležitější informace o projektu. Zdroj[2]

Ganttův diagram

Dalším vizualizačním nástrojem, který se často využívá pro řízení a správu projektů je *Ganttův diagram*. Diagram byl pojmenován podle Henryho L. Gantta (1861-1919), průmyslového inženýra, který v roce 1910 navrhnul první verzi diagramu [5].

„Ganttův diagram srovnává, co se dělá, s tím, co se stalo – udržuje vedení informované o progresu dosaženém při provádění jeho plánu, a pokud progres není uspokojivý, uvede důvod proč.“ [5]

Smyslem Ganttova diagramu je znázornit návaznost jednotlivých částí a činnosti projektu pomocí malých obrázků, neboli obdélníků (pruhů), na horizontální časové ose, které mohou být propojené mezi sebou pomocí spojnic. Levá strana obdélníku označuje plánovaný začátek projektu a pravá strana plánované ukončení. Plánovanou délku trvání projektu zjistíme na základě délky celého pruhu.

Při plánováni projektu je důležitě mít po ruce kontrolní bod, ve kterém je možné ověřit, zda činnost nebo projekt probíhá podle plánu. Tímto bodem je *milník* (angl. *milestone*).

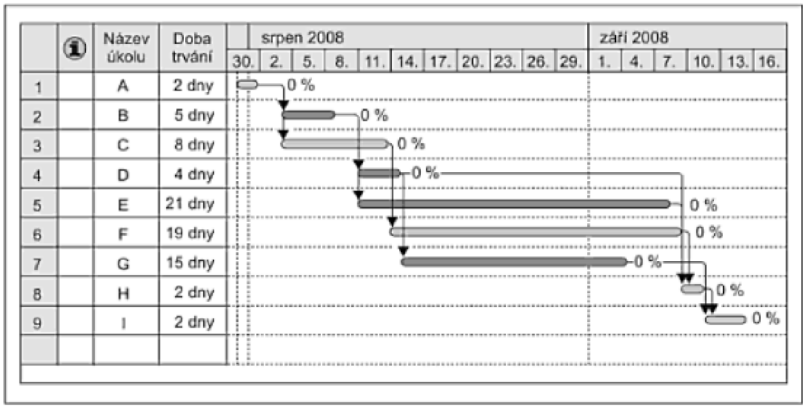
„Všechny úkoly s nulovou dobou trvání jsou označeny jako milníky. Za milník je rovněž možné označit jakýkoliv jiný úkol s libovolnou dobou trvání. Milníky lze v projektu vytvářet jako zástupce pro vnější úkoly.“ [4] V Ganttově diagramu se většinou zobrazuje jako kosočtverec.

Mezi hlavními důvody, proč používat Ganttův diagram při práci na projektech, patří: [3]

- monitorování a oznámeni pokroku,

[2] www.datapine.com

- možnost úpravy projektů,
- vizualizace plánu vývoje,
- zobrazení vztahů mezi činnostmi.



Obrázek 2.3: Ganttův diagram. Zdroj: [6]

2.4 Správa a řízení výzkumných programů a projektů

Inovace a technologie

V dnešní době nové technologie a inovace hrají velkou roli ve všech oblastech našeho života a nesporným faktem je, že za tím stojí celá budoucnost. Je ovšem důležité říct co pojem „technologická inovace“ znamená. Nejedná se o pouhou ideu nebo nápad, je to především implementace a uvedení nápadů v život. Lze také říct, že technologická inovace je přeměna nápadů a znalostí na nové a komerčně úspěšné produkty, procesy a služby [17]. V závislosti na povaze technologie bylo identifikováno několik různých typů technologických inovací, tedy „Inovace produktů“, „Inovace procesů“, „Inovace služeb“, „Inovace technologií“ apod. [17].

Vývojem a výzkumem se zaobírají různé instituce, které jsou zaměřené na konkrétní oblast výzkumného problému, například vědecké akademie a jednotlivé firmy. Velkou roli pro podporu vývoje, výzkumu a tvorbu inovací hraje stát. Díky tomu, že stát tyto instituce podporuje, může dosáhnout a uskutečnit své strategické cíle a zájmy. Z inovační strategie České republiky [11] zní:

> *„Vláda České republiky se proto rozhodla, že podpora vědy, výzkumu a inovacíse nestane pouhou frází, ale zcela konkrétní aktivitou, která bude řízena ambicí zařadit se během dvanáctilet mezi inovační lídry Evropy a stát se zemí technologické budoucnosti.“*

Podpora výzkumu a inovací

Pro kvalitní rozdělení státních finančních prostředků mezi jednotlivými výzkumnými organizacemi nebo firmami, byla v roce 2009 založena tzv. Technologická agentura České Republiky[3] (TA ČR). Hlavní úlohou a zaměřením TA ČR je připravovat a implemento-

[3] https://www.tacr.cz/

vat programy aplikovaného výzkumu, experimentálního vývoje a inovací, a tím přispívat ke zvyšování konkurenceschopnosti a hospodářskému růstu ČR. V současné době TA ČR připravuje a implementuje hned několik takových programů. Každý z nich má určitý název a předem definovaný záměr. Příkladem může sloužit program *THÉTA*[4], který se zabývá modernizaci energetického sektoru, včetně výzkumu ve veřejném zájmu a energetických strategií. Smyslem těchto programů je propojení výzkumné organizace aplikovaného výzkumu ve firmach i ve státní správě.

Univerzitní programy a projekty

Dalším místem, kde se provádí aktivní vývoj a výzkum inovací a nových technologií, jsou univerzity. Stejně jako stát, i univerzita má své zájmy a strategické cíle, kterých když dosáhne, může se posunout na vyšší úroveň a zároveň s tím, což je také podstatné, vydělat finanční prostředky pro realizace vlastních projektů či plánů. Z tohoto důvodu má dnes většina významných univerzit svá výzkumná centra, která spolupracují s firmami z různých oborů. Součástí této spolupráce je spoluúčast na projektech a smluvní výzkum. Příkladem je Fakulta informačních technologií VUT v Brně, kde v rámci programu smluvního výzkumu probíhá práce hned v několika IT oblastech[5]: „Inteligentní systémy“, „Počítačové systémy“, „Počítačová grafika a multimédia“ a „Informační systémy“. V závislosti na konkrétní oblasti, je proces výzkumu následně rozdělen do jednotlivých projektů, na kterých již pracují výzkumné týmy.

Řízení programu

Zatímco účelem řízení projektu je dosažení konkrétního výsledku, program se skládá z několika projektů, které mají společně splnit komplexnější cíl, většinou na základě strategických cílů určité organizace [15]. Pro to, aby se jakýkoliv velký program uskutečnil je zapotřebí mít člověka, který by měl na starosti řízení a spravu takového programu, tj. *programového manažera*. Úlohou programového manažera je všechny tyto projekty synchronizovat tak, aby v každé fázi bylo dostatek infromací pro správné rozhodování. Mezi funkce programového manažera patří: nacházet souvislosti mezi požadavky, podporovat projektové manažery, realizovat pravidelné schůzky a pomáhat jim držet projektový cyklus ve správném směru, který bude plnit očekávané cíle [15]. To všechno platí pro manažera, který řídí programy v rámci firmy, technologické agentury nebo univerzity. Projekty budou přinášet mnohem větší užitek, pokud budou řízeny na úrovni programu, než jako dílčí projekty [15].

[4] https://www.tacr.cz/program/program-theta/
[5] https://www.fit.vut.cz/cooperation/contract-research/.cs

Kapitola 3

Tvorba informačních systémů

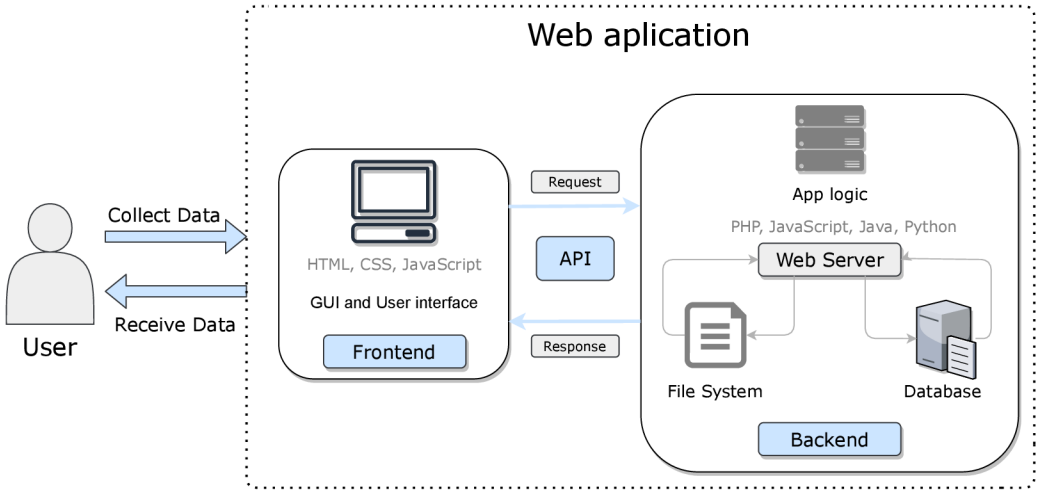
Informační systémy (dále IS) jsou nezbytným systémem pro řešení různých byznys problémů. Jejich cílem je, aby koncový uživatel dostal potřebné informace, v závislosti na své činnosti, které potom může určitým způsobem zpracovat. Bruckner [21] definuje informační systém takto: *„Účelem informačního systému je zajištění správných informací na správném místě ve správný čas. Místem, kam mají být informace dodány, jsou obvykle lidé, kteří jsou součástí byznys systému (uživatelé IS), a kritériem oné správnosti je vhodnost podpory byznys systému v plnění jeho účelu.“* Kvalitní IS by měl být především zaměřen na koncového uživatele. Jinými slovy, systém musí plnit požadavky uživatele na zpracování potřebných informací, jejich doručení, a ve výsledku správnou reprezentaci.

Pro plnění účelu informačních systémů jsou důležité informační a komunikační technologie (ICT). Jsou to hardwarové a softwarové prostředky pro sběr, přenos, ukládání, zpracování a distribuci informace pro vzájemnou komunikaci lidí a technologických komponent informačního systému [21].

V dnešní době existuje mnoho technik a způsobů, jak lze IS implementovat. Mohou to být například aplikace primárně určené pro použití na počítači, tzv. počítačové programy. Nevýhodou takových aplikací je to, že nejsou přenosné. To znamená, že se uživateli neumožní práci, dokud se fyzicky nedostane ke konkrétnímu počítači nebo se připojí vzdáleně. Dalším způsobem je implementace aplikace, která by měla být přístupná přes webové rozhraní, tzv. webová aplikace. Tento způsob má také své nevýhody, jako například menší výkonnost oproti počítačovému programu. S ohledem na teoretické znalosti z kapitoly 2 ovšem lze říct, že lepší volbou pro správu a řízení projektů a programů je aplikace, která umožňuje uživateli mít vždy po ruce možnost sledovat, řídit a spravovat tyto projekty. Proto budou v následujících sekcích popsané technologie tvorby webových aplikací a problémy, které se mají při jejich návrhu a implementaci řešit.

3.1 Webová full stack aplikace

Většina webových aplikací se skládají ze dvou částí – *front-end* (klientská) a *back-end* (serverová), a říká se tomu *full stack* aplikace. Slovo *stack* zde označuje kolekci dílčích modulů – technologií, které se spojují dohromady, aby se dostalo stanovené funkcionality [16]. Pro tvorbu front-end části aplikací se většinou používají technologie jako HTML, CSS, JavaScript. Tvorbu back-endu mají na starosti programovací jazyky PHP, Python, JavaScript a řada dalších. Znázornění schématu webové aplikace je na obrázku 3.1.



Obrázek 3.1: Princip fungování webové aplikace. Uživatel přistupuje k datům skrz tzv. uživatelské rozhraní (GUI) které je součastí front-endu. Poté, pomocí API, probíha komunikace mezi front-endem a back-endem. Server určitým způsobem zpracovává požadavky uživatele a výsledek vrátí zpátky klientské částí. Výsledná data jsou uživateli reprezentovaná pomocí GUI.

Díky se zvyšujícím požadavkům na funkčnost webových aplikací dnes vznikají nové pomocné knihovny a technologie, které umožňují vytvářet robustní aplikace s velkou sadou funkcionality. Zároveň poskytují funkce a nástroje pro usnadnění a zrychlení vývoje. Některé z takových knihoven budou popsané v následujících sekcích.

Z hlediska implementace a návrhu full stack aplikace je potřeba řešit několik problémů [16]:

- návrh architektury aplikace,
- tvorba databáze pro ukládání dat aplikace a tvorba modelu dat,
- implementace serveru, který řeší komunikaci s databází a implementují aplikační vrstvu,
- implementace API[1], které zajišťuje komunikaci mezí platformami,
- implementace klientské části,
- reprezentace dat uživateli.

Možností, jak tyto problemy lze vyřešit, jsou popsané v sekcích 3.3, 3.4 a 3.5.

[1]API – Application Programming Interface (rozhraní pro programování aplikace) je sada definic a protokolů pro vytváření a integraci aplikačního softwaru.

3.2 Architektura aplikace

Jedním z kritérií pro tvorbu webové aplikace je zvolení správné architektury. Architektura popisuje jakým způsobem uživatel s aplikaci komunikuje a jak se předávají data mezi jednotlivými vrstvami aplikace.

Klient-server

Klient–server je jeden z typů architektury informačních systémů. Skládá ze dvou základních komponent: **klient** a **server**. Server je komponenta, která poskytuje určitou službu. Klient je komponenta, která se k serveru připojuje, aby této služby využila. Komunikaci zahajuje klient, který k serveru naváže stabilní spojení a odešle svůj první požadavek. Následně může stejné spojení využít i pro další požadavky. [2]

Třívrstvá architektura

Jedná se o architekturu, která se skládá ze tří komponent:

- prezentační vrstva – vykreslení uživatelského rozhraní (UI) rozhraní,
- aplikační vrstva (někdy se jí také říká byznys vrstva) – výpočetní operace,
- datová vrstva – uchování dat.

Princip komunikace mezi vrstvami je založen na jejich hierarchickém uspořádání. To znamená, že pro to, aby se vstup z prezenční vrstvy dostal do datové, musí nejprve projít přes aplikační vrstvu. Uživatel komunikuje pouze s prezentační vrstvou a dotazuje na vrstvu aplikační, která slouží jako most mezi vyšší a nižší vrstvou. [10]

3.3 Tvorba databáze

Databáze je mechanismus, který se používá pro skladování dat. Uživatel musí být schopen tyto data ukládat a strukturovat. Jakmile jsou data uložena, uživatel by měl mít možnost je lehce získat zpět [18].

Po zvolení vhodné architektury aplikace se musí zvolit způsob, jakým data namodelovat a jak správně ty data následně umístit do databáze. Dnes existují dva typy databázových systému: [16]

- relační databáze,
- nerelační databáze (neboli NoSQL).

Relační databáze je databáze, která pro ukládání dat používá tabulky (*relace*) a vazby mezi nimi. Standardní (*relační*) databáze obsahuje: schéma databáze, schéma tabulek, tabulky, řádky a sloupce, klíče, vztahy a typy dat [18]. Tabulky obsahují v sobě záznamy, sloupce, a atributy, které reprezentují data jako entity datového modelu. Pro správu a řízení SQL databází se používají systémy jako *MySQL*[2], *PostgreSQL*[3], *MS SQL*[4] apod. [16].

[2] https://www.mysql.com/
[3] https://www.postgresql.org/
[4] https://www.mssql.cz/

NoSQL je nerelační, dokumentově orientovaná databáze, která má flexibilní schéma a dotazovací jazyk, který je založen na formátech JSON, BSON nebo XML. To znamená, že vkládaná data nemusí odpovídat vytvořenému schématu. Data v databázi se uchovávají jako kolekce dokumentů v některém z výše uvedených formátů. Nejvíce používané NoSQL databází jsou: *MongoDB*[5], *Cassandra*[6], *Redis*[7].

Volba typu databáze záleží na konkrétních požadavcích uživatele. Už na první pohled je vidět, že struktury relačních a nerelačních databází se liší. Ovšem je možné uvést i další rozdíly mezi těmito typy databází. Díky tomu, že NoSQL má flexibilní schéma, při manipulace s daty se nemusí dodržovat jejich struktura. Z tohoto důvodu návrh modelu nerelační databáze probíhá na základě toho, jak výsledná aplikace bude s daty pracovat. V případě relační databáze je to naopak – nejprve se musí vytvořit schéma pro každou tabulku, protože struktura dat je fixní. Důležitým rozdílem také je, že nerelační databáze nezahrnuje tabulkový model. Místo toho mohou být data uložena v jednom souboru dokumentu, zatímco tabulka relační databáze organizuje datová pole struktury do předem definovaných sloupců.

Jak již bylo řečeno dříve, důležitou částí procesu tvorby databáze je správným způsobem návrhnout její model. *„Konceptuální modelování je součástí řady metodologií vývoje softwarových systémů. Probíhá ve fázi analýzy požadavků. Jeho cílem je vytvoření modelu konceptů aplikační domény, se kterými bude vyvíjený systém pracovat.“* [22] *„Nejznámější a nejčastěji používanou modelovací technikou konceptuálního pro návrh relačních databází je entitně-vztahové modelování, jehož výsledkem je entitně-vztahový diagram (ER diagram nebo ERD z anglického entity-relationship).“* [22]

3.4 Tvorba serverové částí aplikace

Server webové aplikace slouží jako zprostředkovatel mezi uživatelem a databází. Přijímá požadavky od uživatele na získání příslušných dat, zpracuje je a vrátí zpět. Pro tyto účely existuje mnoho různých prostředí a nástrojů, které budou popsané v následujících podsekcích.

REST – Representational State Transfer

Pro zasílání a přijímání požadavků na/z serveru je zapotřebí mít nějaké veřejné rozhraní (API), které by tuto činnost umožnilo. Takovým rozhraním je právě *REST*[8]. REST je architektura, která slouží pro přístup a manipulaci s daty postředstvím protokolu HTTP[9].

REST reprezentuje čtyři základní metody pro přístup k koncovým bodům (tzv. *endpoints*) aplikace, které odpovídají CRUD[10] (*Create, Read, Update* a *Delete*) operacím. Každý takový endpoint je definován pomocí URI (*Uniform Resource Identifier*) ve tvaru */api/operace/data*. Pro práci s těmito endpointy se používají následující metody:

[5] https://www.mongodb.com/
[6] https://cassandra.apache.org/
[7] https://redis.io/
[8] https://restfulapi.net/
[9] https://developer.mozilla.org/en-US/docs/Web/HTTP
[10] https://www.codecademy.com/articles/what-is-crud

- POST (Create,) */api/insert/project* – vytvoří nový projekt.
- GET (Read,) */api/get/project* – vrátí projekt.
- PUT (Update,) */api/update/project* – aktualizuje projekt.
- DELETE (Remove.) */api/delete/project* – smaže projekt.

Vývojové prostředí

Node.js[11] je jedním z nejpopulárnějších běhových prostředí pro implementaci serverové části aplikace. Je postaven na jazyce JavaScript a interpretu V8 od společnosti Google, kde se klade velký důraz na vysokou škálovatelnost, tzn. schopnost obsloužit mnoho připojených klientů naráz. Díky tomu získává podporu vývojářů při tvorbě robustních full stack aplikací [19]. Architektura Node.js stojí na smyčce událostí (angl. *Event Loop*). Funguje to tak, že po vytvoření serveru se čeká na příchozí požadavky uživatele. Jakmile byl nějaký požadavek zachycen, pošle se do smyčky, která jej následně předá kontroleru. Kontroler ten požadavek zpracuje a vrátí výsledek ve tvaru *callback*[12] funkce.

Podstatnou výhodou prostředí Node.js je balíčkový systém *npm* (*Node Package Manager*)[13]. Npm umožňuje instalovat a spravovat jednotlivé závislosti a moduly v rámci projektu. Aktuálně tento nástroj obsahuje více než 800 000 různých balíčků, které lze procházet a zkoumat skrz webové rozhraní.

Existují i jiné možností, jak lze implementovat serverové prostředí, například pomocí jazyka Python[14] nebo PHP[15]. Dle statistiky W3tech[16], Python běží pouze na 1,4% webových stránek. Zároveň se PHP používá na backendu 79% webových stránek, tudíž lze říct, že tento jazyk je stále ještě nejpopulárnějším programovacím jazykem pro tvorbu webových serverů.

Serverové aplikační rámce

Pro usnadnění vývoje a údržby serveru se používají aplikační rámce (angl. *framework*). Slouží především pro poskytování různých nástrojů a funkcí, které mohou výrazně zjednodušit vývoj aplikace, konfiguraci serveru a implementaci zabezpečení.

Express.js[17] je rychlý, flexibilní a minimalistický aplikační rámec pro Node.js. Poskytuje robustní sadu funkcí pro webové a mobilní aplikace a umožňuje pohodlnou práci s HTTP metodami a middleware[18]. Umožňuje také jednoduché nastavení serveru, čísel portů a koncových bodů aplikace. [9]

Dalším velmi populárním aplikačním rámcem je **Laravel**[19], který je založen na jazyce PHP. Podporuje, a tím usnadňuje, proces autentifikace uživatele, nastavení routování a poskytuje nástroje pro práci s databází.

[11] https://nodejs.org/
[12] https://developer.mozilla.org/en-US/docs/Glossary/Callback_function
[13] https://www.npmjs.com/
[14] https://www.python.org/
[15] https://www.php.net/
[16] https://w3techs.com/technologies/overview/programming_language
[17] https://expressjs.com/
[18] Middleware je software, který je mezi operačním systémem a aplikacemi, které jsou v něm spuštěné.
[19] https://laravel.com/

Django[20] je aplikačním rámcem, který využívá jazyk Python. Cílem Django je usnadnit vývoj komplexních webových aplikací. Podobně jako Express.js i Laravel poskytuje robustní sadu funkcí pro nastavení routování, middlewarů a autentizace.

3.5 Tvorba klientské částí aplikace

Klientská část má za úkol zobrazovat data uživateli a poskytnou mu možnost s těmito daty dále pracovat. Komunikace uživatele a webové aplikace probíhá v prohlížeči, kde běží kód klientské částí. Implementaci této části aplikace je možné provést pomocí několika různých technologií, které budou popsané v následujících podsekcích.

Základní technologie klientské částí

HTML (*Hypertext Markup Language*) je značkovací jazyk, který definuje strukturu obsahu webové stránky a popisuje semantiku. HTML je tvořeno řadou prvků, které se používájí k zabalení nebo označení různých částí obsahu. Pro oddělení jednotlivých elementů HTML se využívají značky (angl. *tag*). Tyto značky mohou změnit část obsahu na hypertextový odkaz. Dělí se na párové a nepárové. Nepárové se skládají pouze z jedné značky. Avšak párové naopak, mají navíc koncovou značku, do které se přidává lomítko před samotným názvem značky. Využívají se totiž k tomu, že právě pomocí značek přiřazuje jednotlivým slovům či blokům textu různou váhu a dává tak zprávu vyhledávačům, jak mají s takto označenými slovy pracovat [13].

Dále je nutné, aby webová stránka kromě základní kostry, kterou tvoří HTML, měla také definovaný vzhled. Tuto úlohu řeší jazyk kaskádových stylů *CSS*[21] (Cascading Style Sheets). CSS je jazyk pro specifikaci různých způsobů, jakými jsou HTML dokumenty se zobrazují uživateli – jak jsou stylováné, rozmístěné apod. Lze ho použít například pro změnu barvy textu nebo velikosti nadpisu. Jedná se o jazyk, který je založen na pravidlech – to znamená, že vývojář definuje určitá pravidla, které by pak měli být aplikované na konkrétní elementy, nebo skupinu elementů webové stránky. CSS pravidla lze definovat přímo v rámci HTML dokumentu nebo v externím souboru, což se dělá ve většině případů [12].

CSS knihovny

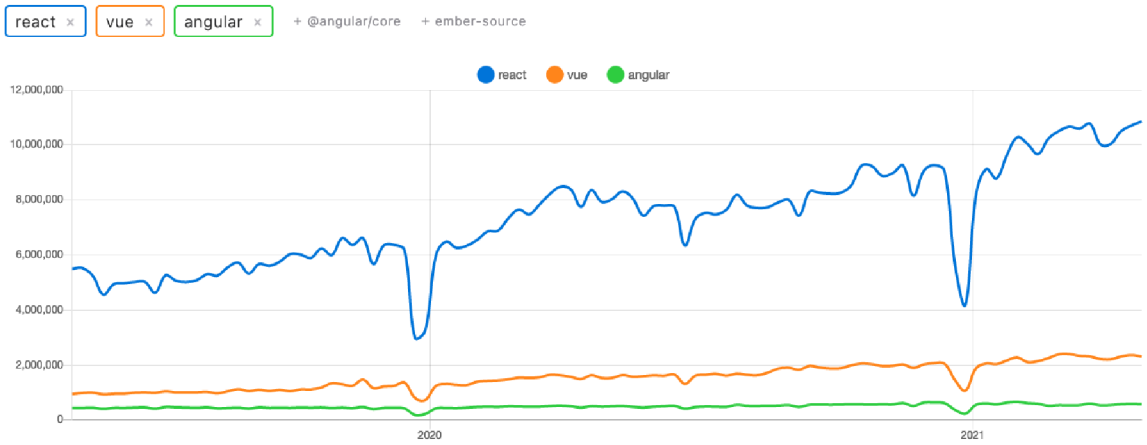
V dnešní době existuje řada robustních CSS frameworků, které se ovšem mezi sebou v určité míře liší. Lze uvést dva typy takových frameworků: ty, které jsou založené na UI komponentách (tzv. *Component-based frameworks*), a ty, které jsou založené na psaní kódu (respektive stylů) pomocí určitých tříd (tzv. *Utility-first frameworks*). UI komponenty umožňují implementovat již předem definované kusy kódů s nastavenými mezerami, odstupy, styly apod. Naopak Utility-first knihovna umožňuje vytvářet vlastní komponenty podle konkretních požadavků. Neexistují pravidla, která by říkaly jaká knihovna se má použít a v jakém případě, ovšem při vývoj full stack aplikací se většinou volí knihovny založené na komponentách. Některým vývojářům, kteří by chtěli tvořit vlastní design, Component-based knihovny mohou přijít příliš omezující, a v takovém případě Utility-first knihovný jsou lepší volbou. Přikladem je *TailwindCSS*[22].

[20] https://www.djangoproject.com/
[21] https://www.w3.org/Style/CSS/Overview.en.html
[22] https://tailwindcss.com/

Jednou z nejpopulárnějších Component-based knihoven je *Bootstrap*. Jedná se o volně dostupnou knihovnu, která obsahuje sadu nástrojů pro tvorbu a implementaci uživatelského rozhraní webových aplikaci. Díky velké řadě předem definovaných pravidel, umožňuje stylovat aplikace pomocí přidávání tříd (angl. *class*) a různých identifikátorů k jednotlivým elementum HTML dokumentu. Tím výrazně zjednodušuje, a hlavně rozšiřuje možností implementace kvalitního uživatelského rozhraní. Podobnou funkcionalitu nabízí knihovny *Materialize*[23], *UIkit*[24], *Pure.css*[25] a mnoho dalších.

Klientské aplikační rámce

Pro implementaci funkcionality klientské částí aplikace je zapotřebí vytvořit prostředí pro obsluhu událostí, možnost denamicky měnícího obsahu a správu akcí uživatele. Pro tyto účely existuje skriptovací jazyk *JavaScript*[26], který se spouští a běží v prohlížeči. Ačkoliv pro tvorbu frontendu lze použít HTML, CSS a JavaScript, v dnešní době je to nepraktický přístup, který zbytečné zpomaluje a omezuje vývoj. Důvodem jsou stále zvyšující se požadavky na funkčnost a vzhled webových aplikací. Proto vznikla velká řada nových, tzv. *front-end frameworků* (neboli klientský aplikační rámec), které jsou založené na jazyce JavaScript, a které usnadňují proces vývoje moderních webových aplikací. V dnešní době existuje velké množství takových frameworků, nicméně lze vyčlenit tři nejpoužívanější: *React*[27], *Angular*[28] *a Vue.js*[29], ovšem React není pravým frameworkem, protože pro implementaci pokročilé funkcionality vyžaduje knihovny třetích stran. Tyto tři nástroje mohou být použity téměř zaměnitelně k vytváření front-endu webové aplikace, ale nejsou stoprocentně stejné. React a Vue.js jsou založené na znovu použitelných komponentech, Angular však tvořen moduly. Na obrázku 3.2 lze vidět popularitu jednotlivých frameworků.



Obrázek 3.2: Graf popularity jednotlivých JavaScript frameworků podle počtu stažení během posledních dvou let[31]

[23] https://materializecss.com/
[24] https://geekflare.com/best-css-frameworks/
[25] https://purecss.io/
[26] https://www.javascript.com/
[27] https://reactjs.org/
[28] https://angular.io/
[29] https://vuejs.org/
[31] https://www.npmtrends.com/react-vs-vue-vs-angular

Pro čtení a změnu obsahu webové stránky nebo reakci na akci uživatele je nutné mít přístup k elementům HTML dokumentu. Tyto elementy jsou součástí DOM[32] (*Document Object Model*), který reprezentován ve tvaru stromu. Vue.js a React vytvoří virtuální kopii modelu DOM, zpracují jej a výsledek se poté porovná s původní verzí. V výslednem dokumentu, tj. na obrazovce uživatele, budou nahrazeny pouze ty části stránky, které se liší od výsledků zpracování. Díky tomu se výrazně zvětšuje *performance* (neboli výkon) aplikace. Přístup ke zpracování DOM pomocí Angular je zásadně odlišný. Zde je rozdělení na dvě vlákna, kde prohlížeč odpovídá za vykreslování (neboli *rendering*) DOM, ale načítání kódu a služeb má na starosti serverová část. Hlavní úlohy, které tyto aplikační rámce plní jsou: ovládání uživatelského rozhraní, reakcí na vstup uživatele, ověřování vstupu uživatele ve formulářích, směrování, správa stavů a zpracování *ajax*[33] požadavků pro komunikaci se serverem. [14]

[32] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
[33] https://developer.mozilla.org/ru/docs/Web/Guide/AJAX

Kapitola 4

Analýza požadavků a existujících řešení

Pro kvalitní a funkční aplikaci je především nezbytné specifikovat a určit uživatelské požadavky, které tato aplikace musí splňovat. Díky tomu lze následně správně navrhnout celý systém a zamezit vzniku nových chyb.

V první části této kapitoly budou popsané základní funkční a nefunkční požadavky na aplikaci. Druhá část kapitoly obsahuje popis již existujících řešení. Na základě tohoto popisu lze pak ukázat výhody a nevýhody jednotlivých aplikací.

4.1 Analýza požadavků

Jak již bylo zmíněno v úvodu, aplikace by měla sloužit pro správu a řízení projektů. To znamená, že požadavky na aplikaci by měly být konzultovány s lidmi, kteří mají určité zkušenosti se správou projektů a rozumí celému procesu řízení. Proto byly původní návrh a požadavky konzultovány s ústavem VCIT na FIT VUT Brno, kde se se mnou ochotně podělili o zkušenosti z oblasti řízení projektů a programů. Dále bylo přesně stanoveno, co systém musí splňovat a jakým způsobem uživatel (v případě mé aplikace programový manažer nebo řešitel projektu) s aplikací pracuje.

Aplikace by měla být zaměřena na správce projektů nebo programů z různých oborů a oblasti. To znamena, že by neměla obsahovat konkrétní speficikaci jen pro řízení projektů s technickým zaměřením nebo naopak, pro projekty z oblastí financí apod. Především se jedná o projekty, které budou řešit jednotlivci nebo menší týmy, a proto potřebují mít systém co nejefektivnější a nejjednodušší pro použití bez zbytečného zatížení. S ohledem na tyto vlastnosti byly specifikovány požadavky, které jsou popsané v následujících sekcích.

4.2 Funkční požadavky

Funkční požadavky definují konkrétní požadavky na funkcionalitu celého systému. Zejména se jedná o zabezpečení nebo grafickou prezentaci dat. Na základě funkčních požadavků bylo následně možné provést návrh technické části aplikace a implementovat systém, který bude tyto požadavky co nejlépe splňovat.

Přihlašování a správa účtu

Prvním a zároveň jedním z klíčových požadavků je autorizovaný přístup do aplikace, který následně zaručí důvěrný přístup k datům. Důvodů pro autorizovaný přístup je několik. Prvním z nich je bezpečnost. Projekty mohou obsahovat citlivé informace jako je například rozpočet, koncové termíny jednotlivých úkolů (neboli *tasků*) nebo dokumenty, které se tykají daného projektu. Dalším důvodem je možnost editace nebo odstranění jednotlivých projektů z plánu, což je schopen provést pouze přihlášený uživatel. Je také důležité, aby přístup do aplikace nemohl získat někdo, kdo nemá žádný vztah k řešení jednotlivých projektů, protože tím pádem se ztrácí význam autorizace jako taková. Proto musí být přístup do aplikace podmíněn přidáním nového uživatele skrz správce systému. Uživatel musí mít možnost upravovat osobní údaje a v případě potřeby změnit heslo.

Správa projektů a událostí

Po úspěšném přihlášení do aplikace bude možné vytvářet, upravovat a odstraňovat jednotlivé projekty a události (mohou to být úkoly, milníky, fakturace a další typy událostí). Jednotlivé události mohou mít jednu nebo více vazeb (provázanost) mezi sebou. Je možné tyto události klasifikovat podle typu, proto by aplikace měla umožňovat tyto typy vytvářet. Těchto typů může být několik, závisí to na konkrétních požadavcích uživatele. Projekt nebo událost musí mít stanovené termíny začátku a konce. Musí existovat možnost přidat název projektu, krátký popis, termín začátku a konce, aktuální stav vývoje, nahrát soubor nebo obrázek v jakémkoli formátu, přidat jméno klienta a rozpočet. Projekt musí mít určitý status, který zjevně označí, v jakém stavu je v daném okamžiku proces vývoje: *in plan, in progress, done.*

Grafická reprezentace dat

Důležitým požadavkem je rovněž grafická reprezentace dat. Po úspěšném přihlášení do aplikace se uživateli zobrazí **dashboard**, který v sobě obsahuje nejdůležitější informace o běžících projektech. Hlavním cílem tohoto dashboardu je hned na úvodní stránce zobrazit co nejvíc informací o aktuálním stavu projektů, aby uživatel nemusel zbytečně prohledávat jednotlivé stránky a získávat tyto informace z různých míst. Musí to být jednoduchý, ale zároveň informativní dashboard, který neobsahuje příliš mnoho zbytečných detailů. V první řádě se musí zobrazit celkový počet projektů, počet aktuálně běžících projektů (které mají stav *in progress*), událostí v aktuálním týdnu a měsíci. Více o grafickém a funkcionálním návrhu dashboardu je popsano v kapitole 6.

Dashboard pouze zobrazuje informace o projektech, čímž nepokrývá všechny potřeby projektového řízení. Jak již bylo zmíněno v kapitole 2, jedním z nejefektivnějších způsobů grafické reprezentace stavu projektů je **Ganttův diagram**. Musí nicméně splňovat určité požadavky na systém, aby byl použitelný a přehledný pro uživatele. Diagram musí mít možnost horizontálního posouvání (*scrollbar*) v závislosti na časové ose, aby bylo možné se podívat do již ukončených projektů v minulosti nebo plánovaných projektů do budoucna. Dalším požadavkem úzce souvisejícím s časovou osou je možnost přibližování času (tzv. *zoom*). To znamená poskytnout uživateli možnost zvětšit nebo zmenšit časový interval diagramu pro lepší přehled projektů nebo úkolů. Na časové ose nesmí chybět aktuální měsíc a datum. Diagram by měl umožnit interaktivní práce s jednotlivými grafickými prvky ve tvaru obdélníků, čímž jsou projekty nebo úkoly. Příkladem je případ, kdy chce uživatel změnit datum začátku projektu. Aby to nemusel upravovat manuálně, stačí přetáhnout le-

vou část obdélníku na nové datum. V případě, že uživatel bude chtít zacházet do detailů projektu, tak musí mít možnost se podívat do jednotlivých událostí, které patří tomuto projektu. Jednotlivé typy událostí se musí odlišovat barvou. Buď je to milník nebo úkol, fakturace nebo deadline – mezi udalostmi musí být vidět rozdíl.

Posledním požadavkem na grafickou reprezentaci dat je možnost otevřít aplikaci na mobilním zařízení. Z tohoto požadavku vyplývá, že aplikace musí být responzivní, to znamená, že se musí přizpůsobit jakémukoliv rozlišení a vhodným způsobem zobrazit informace v dashboardu a Ganttově diagramu.

4.3 Nefunkční požadavky

Mezi nefunkční požadavky jsou zařazeny obecné žádosti na systém – jak se aplikace musí chovat vůči uživateli, jaké vlastnosí musí splňovat a jaké omezení může mít. Základními nefunkčními požadavky na tuto aplikaci, které byly stanoveny, jsou:

- Rozšiřitelnost – aplikace musí mít možnost přidavat novou nebo modifikovat jíž existující funkcionalitu systému bez ovlivnění dálších částí systému.
- Dostupnost – to znamená, že všechny fuknkce aplikace musí být přistupné vždy po celou dobu běhu systému.
- Udržitelnost – v případě, že aplikace má v sobě nedostatky, musí existovat způsob provedení úprav bez porušení základní funkcionality systému.
- Bezpečnost – aplikace musí zaručit spolehlivý přistup k datům, důvěrný přistup a integritu.

Systém by měl být ve výsledku jednoduchý, přehledný a intuitivní pro uživatele. Návrh a implementace aplikace pro splnění uvedených požadavků jsou popsané v kapitolách 5 a 6.

4.4 Analýza existujících řešení

V dnešní době je správa a řízení projektů aktuálním tématem. Proto existuje řada aplikací umožňujících různým společnostem a jednotlivým manažerům řídit své projekty. V této kapitole bude popsáno několik aplikací, u kterých se zaměřím na popis důležitých vlastností, předností a nedostatků.

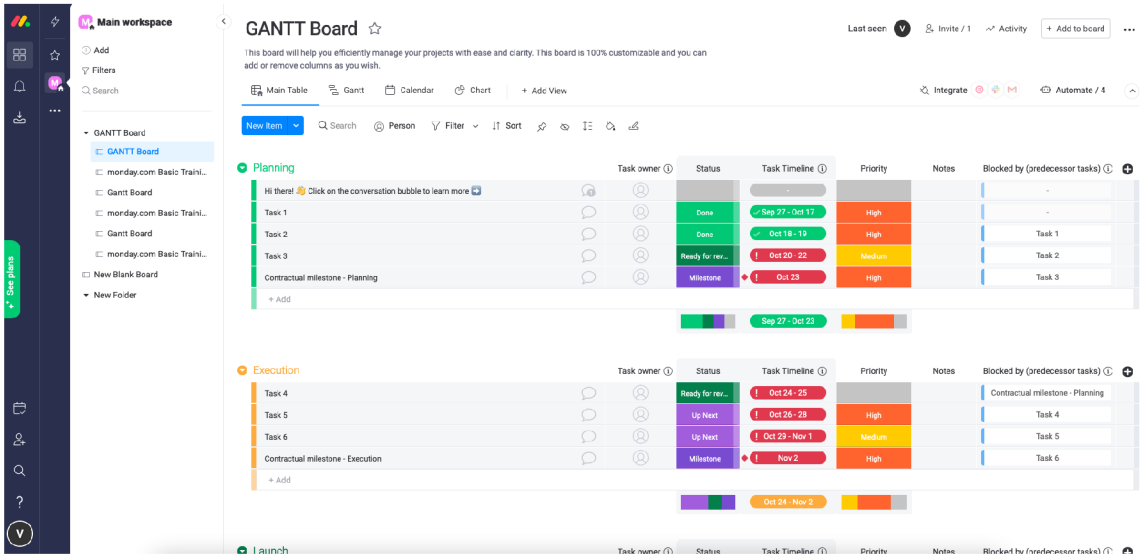
Monday.com

Jedná se o jednu z nejpopulárnějších aplikací na trhu. Má především příjemné uživatelské prostředí, čímž hned upoutá potenciálního zákazníka. Platforma je vyvinuta takovým způsobem, aby vyhovovala širokému spektru požadavků včetně požadavků klienta například na vývoj a výzkum, marketing nebo IT služby.

Mezi hlavní výhody aplikace oproti ostatním projektovým nástrojům patří více než 200 předpřipravených šablon pro plánování marketingových a sales kampaní, správu faktur, produktové roadmapy a mnohé další. Důležitou vlastností, kterou ocení jak malé tak i velké firmy, je propojení aplikace s externími službami. Mezi ně patří Slack, Excel, Gmail, Google, Outlook, Zoom, Dropbox a další.

Na obrázku 4.1 je vidět hlavní stránku aplikace, která se zobrazí uživateli při práci na projektech. Lze vidět seznam jednotlivých projektů a možnost přecházet mezi jednotlivými dashboardy. Monday.com se hodí do malých i do větších firem. Především se zalíbí majitelům různých agentur, kteří potřebují efektivně spravovat projekty a úkoly.

Ovšem je to placená aplikace, kde cena závisí na předplatném. Existuje také bezplatná verze, která ale má určitá omezení. Nevyhodou je příliš mnoho funkci, které nejsou důležité pro menší projekty a ve kterých se lze lehce ztratit a zbytečně to komplikuje používaní aplikace.



Obrázek 4.1: Řidicí panel aplikace Monday.com. Zdroj[1]

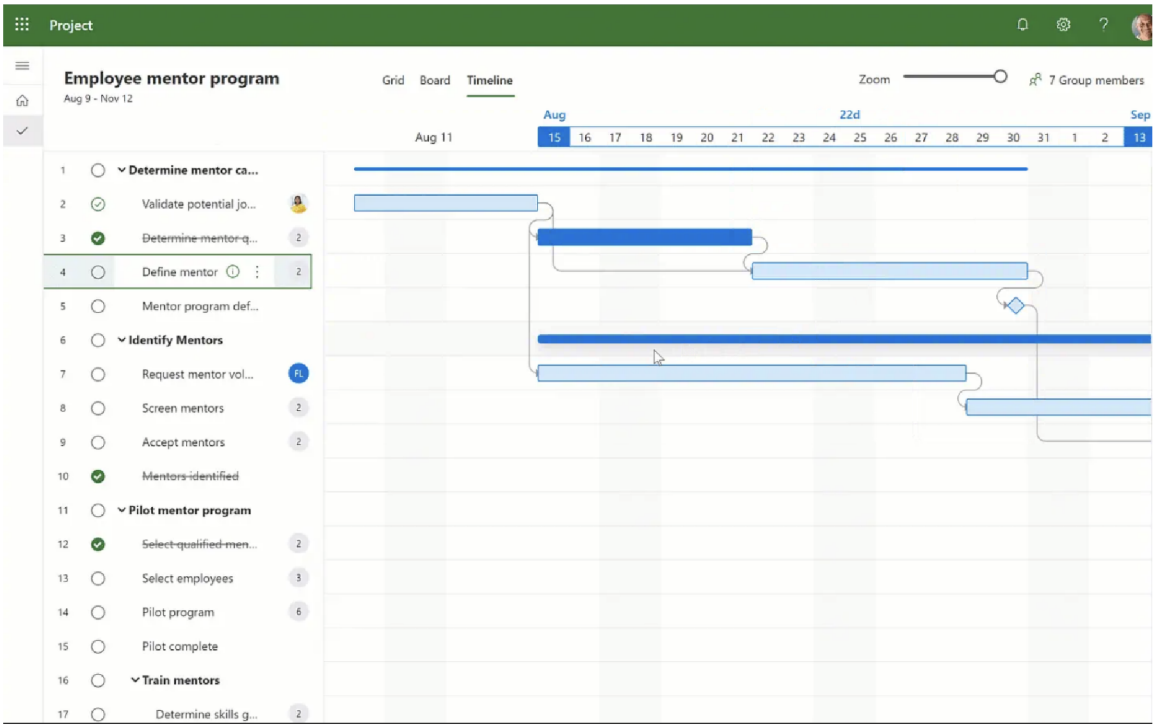
Microsoft Project

Microsoft Project je jednou z pokročilejších platforem řízení projektů, které jsou dnes k dispozici. Stejně jako každé kvalitní řešení pro správu projektů je do této aplikace zahrnuta možnost práce s Ganttovým diagramem. Jednou z hlavních vlastností Microsoft Project je tzv. automatické plánování projektů. Aplikace očekává, že při plánování projektu uživatel určitým způsobem definuje úkoly a milníky. Jakmile se zadá rozsah celého projektu, do hry vstoupí automatický plánovač, který uloží do kalendáře jednotlivé úkoly a díky tomu vytvoří lepe zvladatelný tok projektů na časové ose ve tvaru Ganttova diagramu. Nabízí také řadu dalších funkcí, které lze najít v podobných aplikacích, mezi které patří možnost nahrát příslušné soubory, vytvořit provázané podúkoly atd.

Podobně jako Monday.com je to placená aplikace. Slouží především pro pokročilého uživatele. Mezi nevýhody lze také uvést, že aplikace není přenosná. Klient se nedostane k datům, pokud nemá nainstalovánou aplikaci na svém počítači.

Na obrazku 4.2 je vidět příklad pracovní plochy pro práci s Ganttovým diagramem.

[1] https://monday.com



Obrázek 4.2: Řidicí panel aplikace Microsoft Project. Zdroj[2]

Wrike

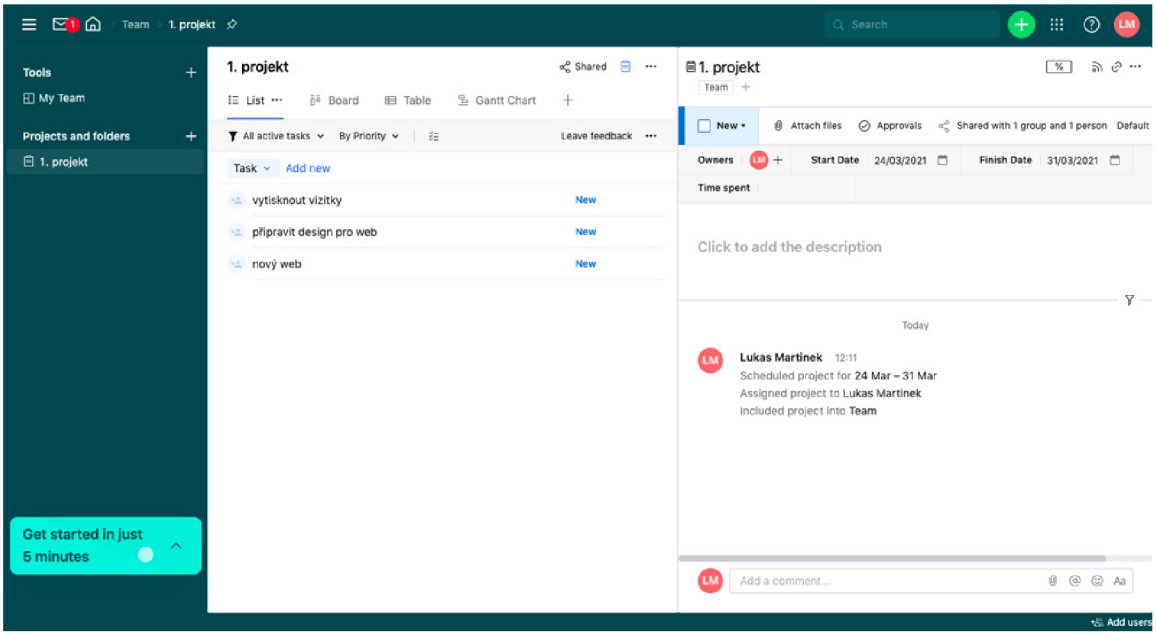
Aplikace Wrike je jedna z nejstarších aplikaci, která působí na trhu od roku 2006, čímž získává velkou důvěru ze strany zákazníků. Wrike v současné době patří mezi nejpopulárnější aplikace v oblasti projektového managementu. K těmto účelům platforma nabízí velké množství funkcí, mezi kterými jsou Ganttův diagram, reporty v reálném čase, sdílení souborů, práci s dashboardy a řadu dalších funkcí. Aplikace slouží primárně pro velké firmy, kde se vyplatí mít velké množství pomocných funkcí, ale pro malé firmy by byla příliš rozsáhlá a drahá. Pro nového uživatele, který neměl dříve zkušenosti s Wrike, by aplikace byla dost náročná a neintuitivní kvůli velkému množství, občas zbytečné, funkcionality. Wrike je profesionální nástroj pro správu a řízeni projektů, a proto je také zpoplatněn.

Na obrázku 4.3 je zobrazen řídicí panel aplikace.

Zhodnocení současného stavu

V dnešní době lze pozorovat jak na trhu vzniká velké množství aplikací, které slouží pro správu a řízení projektů. Pro účely své analýzy jsem vybral ty nejpopulárnější, které už mají dobrou pověst a působí na trhu jako kvalitní a profesionální nástroje. Uvedené aplikace nesplňují zcela přesně zvolené požadavky. Mezi výhody ale patří příjemné vizuální rozhraní, robustní sada funkcí a možnost propojení aplikace s jinými nástroji. Cena, nutnost školení a občas příliš náročné prostředí lze považovat za nevýhody aplikace. Dle nabízené funkcionality v uvedených aplikacích se jeví nejvíce vhodná aplikace Wrike. Z hlediská vizualního rozhraní by na prvním místě mohla být aplikace Monday.com. Ve výsledku jsou to velké ko-

[2] https://www.microsoft.com/microsoft-365/project/project-management



Obrázek 4.3: Řidicí panel aplikace Wrike. Zdroj[3]

merční aplikace, které se hodí především pro profesionální správu projektů, a proto nejsou vhodné pro individuálního zákazníka, který nemá zkušenosti s aplikacemi stejného typu.

Mnou navržená aplikace slouží pro individuální uživatele s úzce zaměřenými požadavky na správu a řízení projektů. Důležitou vlastností aplikace je, že oproti již existujícím aplikacím uvedeným vyše, tato aplikace je vyvinutá na míru na základě konkrétních požadavků, což znamená, že řešení nebude obsahovat zbytečnou a hlavně příliš náročnou funkcionalitu. Ve výsledku aplikace bude jednoduchá a intuitivní pro koncového uživatele. V případě vzniku nových potřeb a požadavků bude existovat možnost již funkční řešení jednoduše rozšířit, přizpůsobit a modifikovat. Další vlastností, a zároveň i výhodou, je také to, že aplikace nebude zpoplatněná.

[3] https://wrike.com

Kapitola 5

Návrh a architektura aplikace

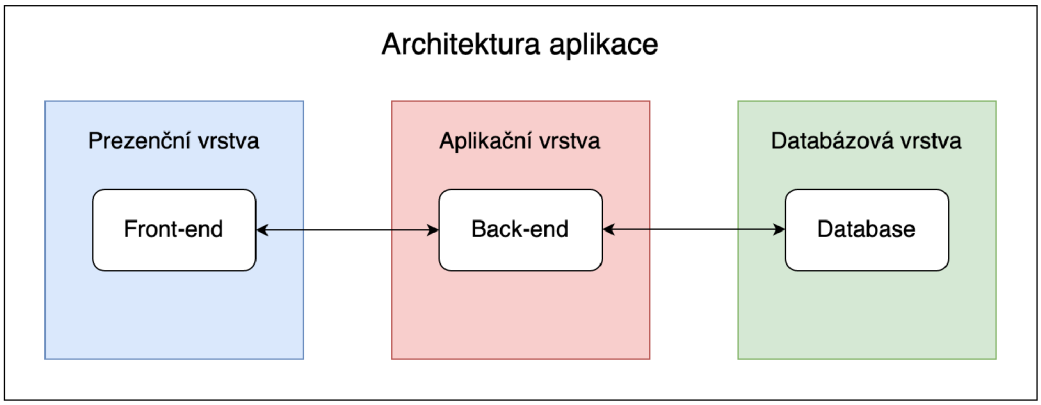
5.1 Architektura aplikace

Na základě analýzy požadavků jsem rozhodl navrhnout webovou aplikaci, která se bude skládat ze tří částí: front-end, back-end a databáze. Z principu webová aplikace není vázána na konkrétní typ zařízení, tudíž bude přístupná jak z počítače, tak i z mobilu nebo tabletu.

Na obrázku 5.1 je uveden diagram popisující architekturu aplikace. Jedná se o tradiční třívrstvou architekturu, která se skládá z:

- prezenční vrstvy (*front-end*), která popisuje uživatelské rozhraní a vzhled aplikace.
- aplikační vrstvy (*back-end*), která tvoří prostředí aplikačních funkcí (neboli obchodní logiku).
- databázové vrstvy: poskytuje práce s datovým modelem aplikace a řízení databázových operaci.

Tyto vrstvy jsou tvořené speciálními nástroji, které budou popsané v následujících kapitolách. Smyslem třívrstvé architektury je, aby každá z těchto vrstev představovala samostatný nezávislý celek a měla vlastní funkcionalitu.

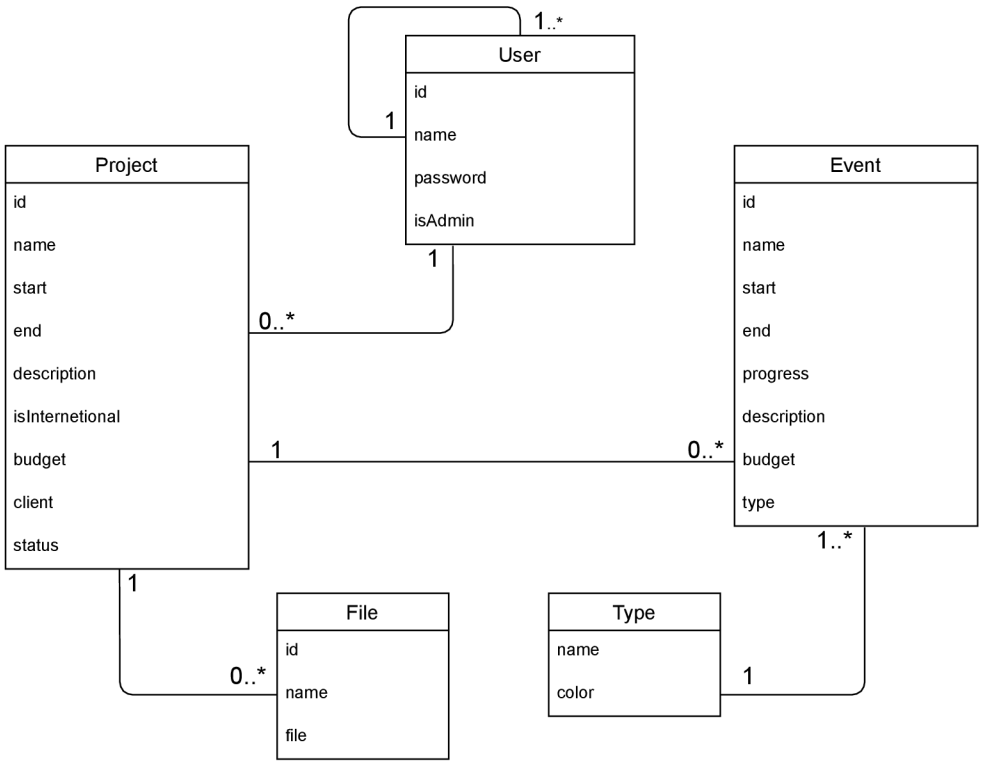


Obrázek 5.1: Třívrstvá architektura aplikace

5.2 Návrh databáze a datového modelu

Ukládání dat je důležitým aspektem při vývoji aplikace. Cílem návrhu databáze je určit, jak budou data uložena v databázi a jak lze definovat datový model aplikace. Databáze musí uchovávat informace o uživatelích aplikace, jednotlivých projektech čí událostech. Pro tyto účely bude využita NoSQL (nerelační) databáze, což znamená, že její model není založen na relačních vztazích mezi tabulkami. Podrobný rozdíl mezi relační a nerelační databází je popsán v kapitole 3.

Databáze musí ukládat informace o jednotlivých uživatelích systému, projektech a událostech, které tito uživatelé mohou spravovat, modifikovat a mazat. Popis jednotlivých případů užití lze nalezt na obrázku 5.3. Pro lepší pochopení a přehlednost byl návrh struktury databáze modelován pomocí ER diagramu uvedeného na obrázku 5.2.



Obrázek 5.2: ER diagram popisující návrh struktury databáze

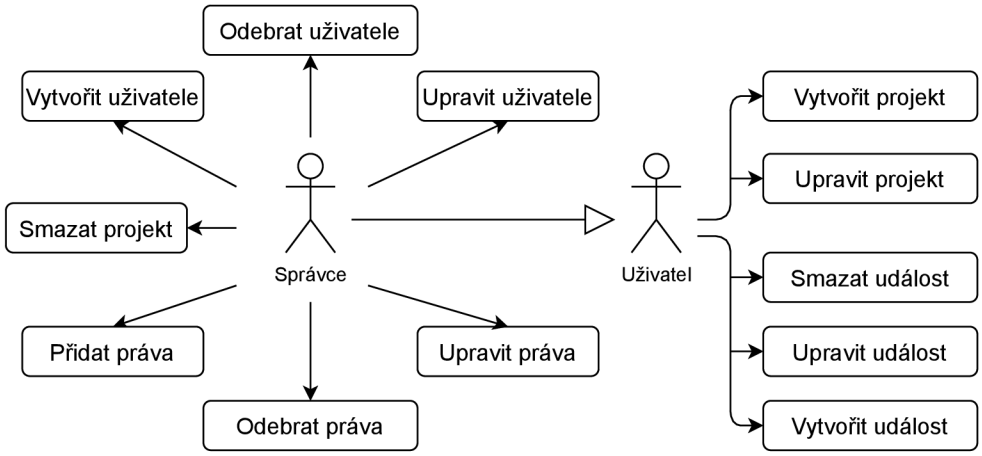
Z diagramu jsou patrné vazby mezi entitami. Následně by tento diagram měl byt transformován do podoby jednotlivých dokumentů dle NoSql databáze. Návrh diagramu vychází ze specifikovaných požadavků. Uživatel by měl mít možnost vytvořit jeden nebo více projektů. Po přihlášení by měl mít možnost do projektů pouze nahlížet a nevytvářet žádný nový. Jelikož dle požadavků je potřeba zajistit bezpečnost systému, je nutné, aby měl každý uživatel přidělenou roli. Správce systému by tyto role měl nastavit během registrace uživatele.

Na základě teoretických znalostí o projektovém řízení získaných z kapitoly 2, lze stanovit, že každý projekt může mít několik událostí. Každá událost musí mít určitý typ, ovšem události stejného typu může být několik. Další vlastnosti jednotlivých dokumentů jsou popsané v následujících sekcích.

Uživatel

Uživatel v aplikaci hraje podstatnou roli. Pro vykonání jakékoliv činnosti se do aplikace musí přihlásit prosřednictvím přihlašovacího formuláře, kde zadá svůj email a heslo. Pokud uživatel v systému ještě neexistuje, musí požádat správce o vytvoření nového účtu. Z hlediska práce na projektech se role běžného uživatele se mohou lišit. Stejně tak správce bude mít možnost uživatele ze systému smazat a tím odebrat oprávnění s aplikaci pracovat. Každý uživatel systému by měl mít uvedené jméno, email, heslo a přidělenou roli.

Jednotlivé role a činnosti uživatele jsou reprezentovány v diagramu případů užití (neboli use-case diagram) na obrázku 5.3, který je popsán v jazyce UML[1]. Tento diagram popisuje chování systému z pohledu uživatele. Skladá se z aktérů a případů užití – jsou to činnosti, které uživatelé mohou vykonat a rámci aplikace. V systému se pracuje se dvěma uživatelskými rolemi: správce (*admin*) a uživatel (*user*). V následujícím diagramu jsou uvedeny role a jejich vztahy s případy užití.



Obrázek 5.3: Diagram případů užití

Projekt

Projekt je důležitou částí datového modelu. Může ho vytvořit, upravit, pouze přihlášený uživatel, který má nastavenou příslušnou roli. Ovšem pouze správce systému má oprávnění ke smazání projektů. Jak již bylo zmíněno, nepřihlášený uživatel nemá přístup k funkcím aplikace, a proto nemůže provádět žádnou interakci s projekty.

Mezi povinné atributy patří: název projektu, termín začátku a konce, jmeno řešitele projektu, jméno klienta, pro kterého se daný projekt vykonává, rozpočet, stav projektu, informace, zdali se jedná o mezinárodní projekt. Nepovinné jsou atributy, které obsahují popis projektu a možnost nahrát soubor.

Událost

Ke každému projektu bude mít uživatel možnost vytvořit jednu nebo více událostí. Díky tomu mohou v systému existovat stejné události patřící různým projektům. Události v systému musí mít: název, termín začátku a konce, stav, ve kterém se daná událost aktuálně

[1] https://www.uml.org

nachází (vyjádřený v procentech), typ (milník, fakturace a další), závislosti, kde jsou vypsané provázané události, a identifikátor projektu, ke kterému tato událost patří.

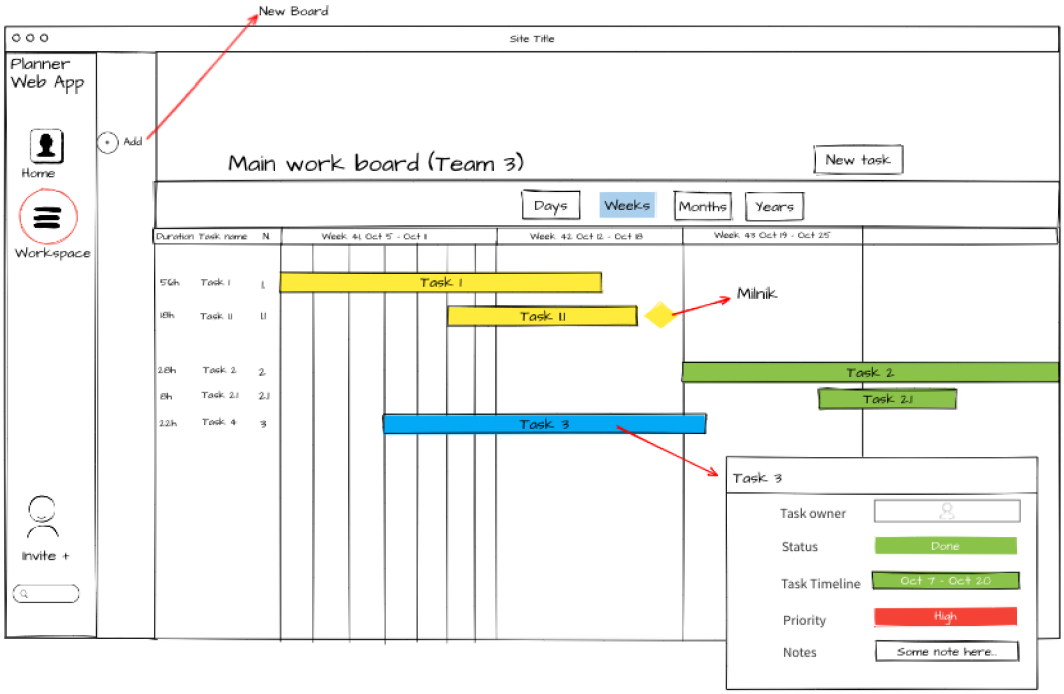
Tvorba uživatelského účtu

Jedna se o akci, která má za úkol vytvořit nového uživatele v systému. Tato možnost bude dostupná pouze pro správce. Stejně tak správce může upravovat informace o uživateli a smazat ho z databáze.

Po vytvoření učtu nový uživatel dostane přístup do aplikace a bude schopný provádět interakce s projekty. Nedostane ale možnost přidat nového uživatele do systému.

5.3 Návrh grafického uživatelského rozhraní

V této kapitole bude popsán návrh grafického uživatelského rozhraní aplikace. Tento návrh je postaven na předchozí analýze (kapitola 4.2), ve které byly definovány požadavky na aplikaci. Bude zde popsán obecný grafický návrh systému a návrh jednotlivých funkčních částí aplikace. Mezi požadavky na grafický návrh byl zařazen bod, aby aplikace měla přehledný a moderní vzhled. První náčrt aplikace jsem vytvořil již po první konzultaci, na které byly probrány základní požadavky na systém (obrázek 5.4). Během implementace a následujících konzultací se tento návrh několikrát měnil.



Obrázek 5.4: První návrh uživatelského rozhraní

Navigační lišta

Navigační lišta (dale navbar) se bude nacházet v horní částí stránky a bude mít horizontální formu. Vlevo je umístěno logo aplikace. Potom následuje obsah navigace, která obsahuje tři položky. První z nich je „Home“, která vede na úvodní stránku aplikace obsahující dashboard. Druhá položka je „Gantt“, která přesměruje uživatele na stránku obsahující plochu Ganttová diagramu. Třetí položka „User Settings“ se zobrazí pouze správci. Na pravé straně navbaru bude jméno uživatele a tlačítko „Logout“.

V původním návrhu byl navbar implementovan jako postranní panel (angl. *sidebar*) a ten měl možnost se skrývat v případě, kdyby by bylo potřeba mít větší pracovní plochu. Nakonec jsem rozhodl tento navbar přemístit nahoru, aby uživatel měl vždy po ruce všechny odkazy a nemusel ho skývat pokaždé, když potřebuje zvětšit pracovní plochu pro práci s diagramem.

Formulář

Hned po otevření aplikace dojde k přesměrování na přihlašovací formulář, který je svým stylem jednoduchý a obsahuje jenom funkční prvky, které jsou potřebné. Při přihlášení uživatel zadá e-mail a heslo, v případě neúspěchu dostane upozornění pod textovým vstupem.

Aplikace bude mít ještě jeden druh formuláře, který slouží k vytváření projektů a úkolů. Každý formulář bude mít odpovídající počet polí, to ale záleží na počtu atributů odpovídajícího datového modelu.

Dashboard

Po přihlášení se uživateli zobrazí úvodní stránka obsahující jednoduchý a přehledný dashboard. Dashboard obsahuje několik od sebe oddělených sekcí a každá z nich uchovává příslušnou informaci.

V první sekci uživatel najde informace o celkovém počtu projektů. Další sekce bude obsahovat informace o aktuálně běžících projektech (ty, co mají stav *in progress*). Hned pod tím se bude nacházet sekce, ve které bude reprezentován seznam projektů. Je to z toho důvodu, aby uživatel měl možnost hned po otevření aplikace nahlédnout do detailů jednotlivých projektů.

Úplně na začátku stránky bylo umístěno tlačítko „Create Project“. Po kliknutí se uživatel přesměruje na formulář pro vytvoření projektu.

Ganttův diagram

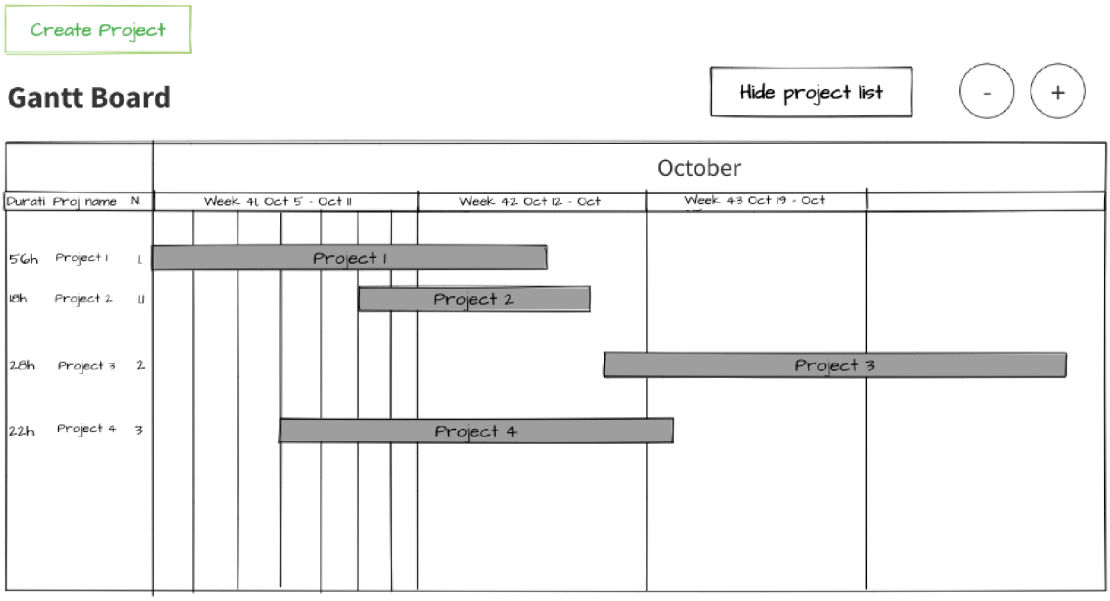
Hlavním cílem grafického návrhu je správným způsobem reprezentovat strukturu projektů a jednotlivých událostí, které danému projektu patří. Pro tyto účely bude použit Ganttův diagram, ke kteremu se lze dostat skrz navbar kliknutím na záložku „Gantt“. Ganttův diagram je hlavním grafickým prvkem aplikace, a proto byl na návrh této části kladen velký důraz. Musí být jednoduchý, interaktivní a přehledný. Návrh diagramu lze vidět na obázku 5.5. Hlavním prvkem je plocha, kde jsou reprezentovány jednotlivé části diagramu. Čas je zde znázorněn pomocí časové osy, která je pro přehlednost umístěná nad samotnými činnostmi. Tato osa bude rozdělena na stejně velké časové úseky, nad kterými bude umístěn název aktuálního měsíce a aktualní rok. Časová osa bude dynamická a bude schopná se přizpůsobovat podle přání uživatele. Z toho důvodu se do diagramu zavede možnost zoomu, čímž umožní nahlédnout na určité činnosti z pohledu různých časových jednotek,

a to od jednoho týdne až po celý rok. Pro tyto účely budou v pravé části nad plochou diagramu implementovány tlačítka umožňující přiblížit nebo oddálit obsah plochy. Uživatel navíc dostane možnost horizontálního scrollingu plochy pomocí posuvníku ve spodní části diagramu.

Procesní činnosti budou zobrazeny na již zmíněné ploše diagramu ve tvaru obdélníků. Pro přehlednost budou obdélníky obsahovat názvy činností, které reprezentují. Jejich délka závisí na intervalu, který uživatel nastaví během vytváření projektu. Na začátku a konci obdélníku se budou nacházet interaktivní prvky, které ho umožní uživateli buď roztáhnout nebo zmenšit, čímž se ovlivní délka trvání činnosti. Oproti projektům události dostanou možnost propojení mezi sebou. Bude to implementovné ve formě jednosměrné šipky, která vede od rodičovské události k synovské. Těchto vztahů může být 0 až N. Jednotlivé události pak budou mít různé typy. Z toho důvodu se obdélníky budou lišit barvou pro lepší orientaci v diagramu.

V levé části diagramu se bude nacházet tabulka s počtem řádků, který odpovídá počtu činností. Každý z těchto řádků bude obsahovat informace o názvu činnosti, době trvání a termínu začátku a konce. V případě potřeby uživatel bude schopen tuto tabulku skrýt pomocí příslušného tlačítka umístěného nad plochou diagramu.

Události, stejně jako projekty, se zobrazují ve formě obdélníků v Ganttově diagramu. Slouží to především pro lepší přehlednost a komfortní správu.

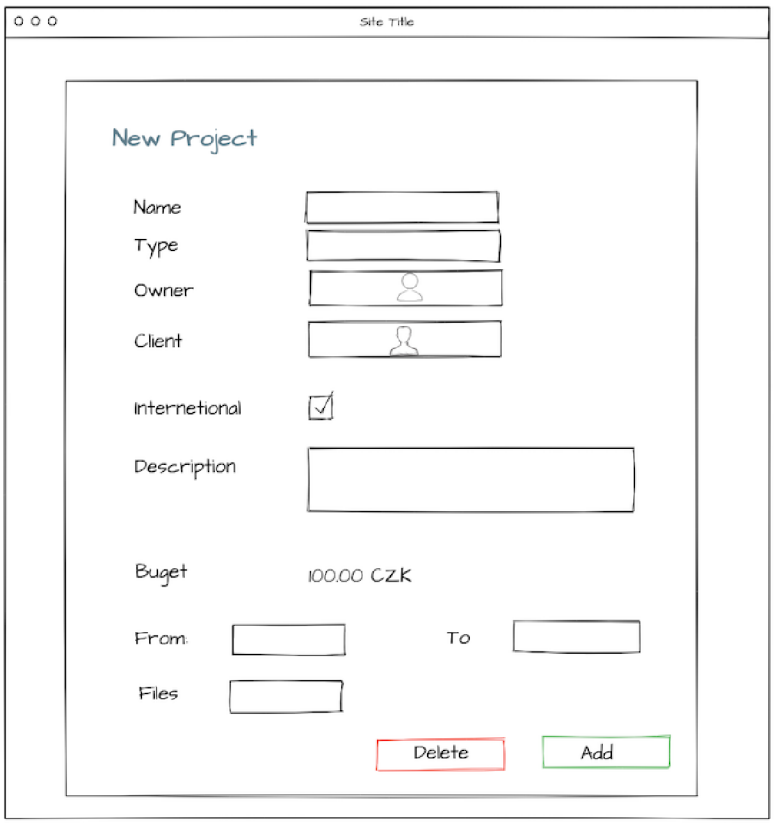


Obrázek 5.5: Návrh Ganttova digramu

Výtvoření projektu

Na základě diagramu případů užití, uvedeného na obrázku 5.3 lze stanovit, že všichni přihlášení uživatelé mají možnost vytvořit nový projekt. Pro provedení této operace se uživatel potřebuje dostat k odpovídajícímu formuláři, který musí vyplnit. Vytvořit projekt lze ze dvou míst aplikace: skrz tlačítko „Create Project“ na úvodní stránce nebo na stránce s Ganttovým diagramem.

Po vyplnění formuláře a kliknutí na tlačítko „Save“ klientská část aplikace odesílá informace o projektu na server. Po úspěšném uložení projektu do databáze se uživateli zobrazí zpráva o úspěšném provedení operace a hned se mu nabídne možnost vytvořit další projekt nebo se vrátit na stránku s Ganttovým diagramem. Klientská část pak na základě úspěšně uložených údajů vygeneruje nový obdélník v diagramu, který bude označovat nově vytvořený projekt. První verzí návrhu tohoto formuláře lze vidět na obázku 5.6.



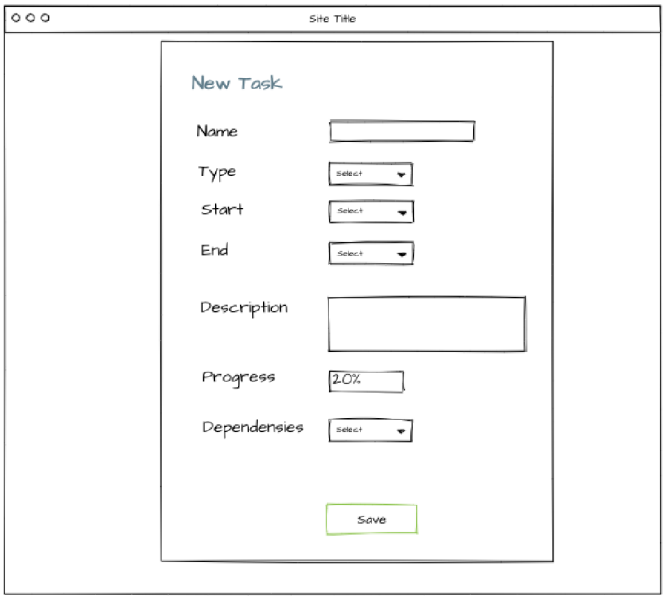
Obrázek 5.6: Návrh formuláře pro vytvoření proejktu

Výtvoření událostí

Uživatel může nahlédnout do seznamu událostí pomocí kliknutí na tlačítko „Show the tasks“ ve vyskakovacím okně projektu. Pokud projekt neobsahuje žádné události, uživateli se místo diagramu zobrazí odpovídající nápověda a tlačítko „Create Task“. Vytváření nové události je ve svém principu podobné vytváření nového projektu, ovšem liší se v několika bodech. Hlavní odlišnost spočívá v tom, že událost může mít vztahy (nebo-li provázanosti) s jinými událostmi a musí mít určitý typ, navíc neobsahuje položku klient a rozpočet.

Pro správu událostí uživatel musí kliknout na odpovídající obdélník Ganttová diagramu, kde se mu pak zobrazí vyskakovací okno s tlačítky „Edit“, „Delete“ a „Close“. Když zvolí „Edit“, vyvolá tím formulář pro editaci. Tlačítkem „Delete“ vyvolá požadavek klientské části na server pro odstranění události z databáze.

Po kliknutí na tlačítko „Create Task“ se uživateli zobrazí formulář 5.7, který je nutné vyplnit. Jelikož jednotlivé události mohou mít různé typy, vzhled formuláře se může lišit pro



Obrázek 5.7: Návrh formuláře pro vytvoření úkolu

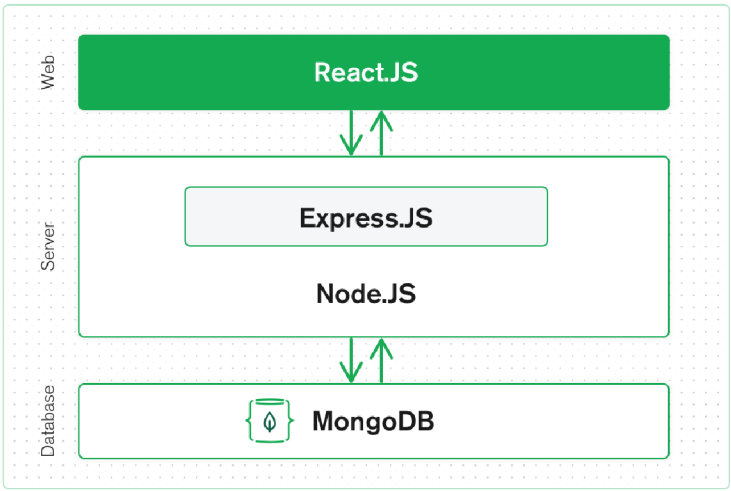
každý z těchto typů. Například, když si uživatel zvolí typ „Milestone“ (milník), nepotřebuje uvádět datum začátku a konce, popis a stav. Stačí mu uvést jenom datum, kam ten milník chce umístit, protože se jedná o jednoznačnou událost, která má nulovou délku. Kliknutím na talčítko „Save“ klientská část odešle požadavek na server pro uložení informace do databáze. Touto činností se zároveň přiřadí úkol k odpovídajícímu projektu. Typ události ovlivní barvu obdélníku, čímž zjednoduší uživateli pohled na celý diagram.

Kapitola 6

Implementace

V této kapitole bude popsána implementace aplikace. Implementace vychází z analýzy a specifikace požadavků v kapitole 4 a návrhu aplikace v kapitole 5. Budou zde rovněž popsané problémy, jež bylo nutno během vyvoje vyřešit. Mezi nejzajímavější patří autentizace, ukládání dat do databáze a implementace grafického rozhraní.

Aplikace je implementována s využitím technologického balíčku MERN (obrázek 6.1), kde je back-end reprezentován prostředstvim prostředí Node.js a frameworku Express.js. Pro manipulaci s daty je použitá NoSQL databáze MongoDB, která používá knihovnu mongoose. Klientská část je napsaná pomocí frameworku React společně s jazyky HTML, CSS a knihovnou Bootstrap. Adresářová struktura aplikace je tedy rozdělena do dvou hlavních složek: server a client. Strukturu adresáře se zdrojovými soubory lze nalézt v příloze A. Příloha B obsahuje seznam koncových bodů API (angl. *endpoint*).



Obrázek 6.1: Struktura technologického balíčku MERN. Zdroj[1]

[1] https://www.mongodb.com/mern-stack

6.1 Back-end

Hlavním účelem back-endu je zpracování příchozích požadavků z front-endu a případné poskytování/vkládání dat z/do databáze. Serverová část obdrží požadavky ze strany klienta a nasměruje je pomoci aplikačního rámce Express.js na příslušnou obslužnou rutinu na základě definovaných koncových bodů REST API. Server také umožnuje nastavení tzv. *middlewares*, přes které procházejí jednotlivé požadavky než se dostanou do jednotlivých koncových bodů.

Konfigurace serveru

Než se začne práce na implementaci funkcionality a uživatelského rozhraní aplikace, je zapotřebí nakonfigurovat samotný server. V první řadě se musí nainstalovat a importovat všechny pomocné moduly a knihovny. Instalace modulů proběhne pomoci správce baličků **npm** příkazem *npm install název_balíčku*.

Nejdůležitější knihovnou na straně serveru je Express.js, pomocí které pak lze celý server nastavit a spustit. Na začátku se nastaví směrování, které řeší komunikaci s klientem, tudíž příchozí a odchozí požadavky. Dalším krokem je potřeba spustit naslouchání na zvolenem portu a propojit databázi a server.

Směrování

Směrování (neboli angl. *routing*) v zakladu specifikuje, jak server odpoví na požadavek od uživatele na konkrétním endpoint, který je reprezentován ve tvaru URI. Endpoint se implementuje pro každý datový model zvlášť. Definici všech směrovačů (angl. *routes*) lze nalézt ve složce */routes*. Příkladem může sloužit implementace směrovače pro načtení projektů z databáze. Implementace obslužní rutiny se nachází v souboru *routes/projects.js*. V první řadě je nutné nastavit objekt směrovače pomocí metody `Router()`, která je instancí knihovny Express.js. Dalším krokem je vytvořit samotný směrovač. Jelikož se jedná o načítání projektů z databáze, je potřeba vytvořit HTTP požadavek GET, který lze implementovat pomocí knihovny *mongoose*[2] a její metody `get()` nad objektem *router*: `router.get('/', getProjects)`, kde `getProject` je kontroler, který implementuje obslužnou rutinu směrovače. Účelem kontroleru je zpracovávat příchozí požadavky od uživatele. Implementace jednotlivých kontrolerů lze nalezt ve zložce */controllers*. Po zpracování se odešle klientské části odpověď ve formátu JSON s nastavením příslušného stavového kódu[3].

Po implementaci všech potřebných směrovačů je zapotřebí je importovat do hlavního serverového souboru *server.js*. Pomocí Express.js middlewaru `app` a metody `use()` se provede propojení směrovačů a aplikace: `app.use('/projects', projectRoutes)`.

Pro další HTTP požadavky POST, DELETE a UPDATE se tedy nad objektem *router* volají metody `post()`, `delete()` a `patch()`.

Napojení databáze

Jako hlavní uložiště sysému jsem zvolil NoSQL databázi MongoDB. Tato databáze slouží pro práci se systémem postaveným na jazyce JavaScript, neboť ukládá data ve formátu JSON (nebo-li BSON podle MongoDB), se kterým se pak snadno pracuje jako s JavaScript objekty, které obsahují páry klíč-hodnota.

[2] https://mongoosejs.com/

[3] Seznam jednotlivých stavových kódů: https://www.restapitutorial.com/httpstatuscodes.html

Pro usnadnění práce s databází MongoDB a přístupu k jejím objektům byl použit již zmíněný modul mongoose. Pro vytvoření spojení mezi serverem a databází je nutne zavolat metodu `connect()` nad modulem mongoose, kde prvnímu parametru se zadá připojovací řetězec (URL databáze). Výhodou modulu mongoose je, že není nutné definovat schémata kolekce v databázi, ale stačí to vytvořit přímo ve zdrojovém kódu aplikace. Ve schématu je potřeba určit atributy jednotlivých objektů a poté datové typy těchto atributů.

Po specifikování schématu je nutné udělat poslední krok a tím je vytvoření modelu. Jedná se o konstruktor vytvořeny na základě definované schématu. Tento model je zprostředkovatelem mezi samotnou aplikací a databází, který může obsahovat metody, které reprezentují jednotlivé operace v databázi. Příkladem mohou být metody `create()` nebo `save()`, které se poté používají ve funkcích kontroleru. Implementace jednotlivých modelů je rozdělena do různých souborů, které se nachází ve složce */models*. Celkem databáze obsahuje čtyři modely: project, task, user a file. Každý z nich je implementován na základě návrhu uvedeného v kapitole 5, kde jsou popsané struktury jednotlivých datových modelů.

Při komunikace klientu a serveru se data předávají ve formátu JSON. Slouží to především pro zjednodušeni práce s daty jak na straně serveru tak i na straně klienta.

Autentizace na straně serveru

Účelem autentizace je jasným způsobem identifikovat uživatele, který přistoupil k aplikaci. Systém musí vědět, že se jedná o přihlášeného uživatele, v opačném případě se mu nesmí zobrazit žádný obsah.

Z principu je proces autentizace relativně jednoduchý. Server obdrží požadavek ze strany klienta a na základě toho požadavku vygeneruje unikátní *jsonwebtoken*[4] (dál JWT). Tento token se poté posílá společně s každým požadavkem, který odesílá klient na server. Slouží to pak jako unikátní elektronický podpis uživatele, který tím potvrzuje svou identitu a následně může se serverem komunikovat pomocí volání jednotlivých endopointů.

Implementace procesu autentizace na straně serveru je popsana v souboru */user.js* ve složce *controllers*. V první řadě se zkontroluje přítomnost uživatele v databázi, tedy se zavolá metoda `findOne()` nad modelem User. Pokud takový uživatel v databázi existuje, tak je zapotřebí zkontrolovat jeho heslo. Pro tyto účely lze použít knihovnu bcrypt.js[5]. Jedná se o knihovnu, která umožňuje vytvořit hash hesla pomocí speciálního hashovacího algoritmu založeného na šifře Blowfish. Kontrola probíhá pomocí volání `bcrypt.compare(password, existingUser.password)`. Hlavní myšlenka metody `compare()` spočívá v tom, aby porovnala již uložené v databázi zahashované heslo příslušného uživatele, a heslo, které ten uživatel zadal při přihlášení do aplikace.

Po uspěšné validaci se následně vytvoří JWT. Pomocí metody `jwt.sign()` lze definovat – na základě jakých atributů se ten token vygeneruje a jakou dobu platnosti bude mít. Mezi atributy byly zvolené email a userID a doba platnosti tokenu byla nastavena na 2 hodiny. Vytvořený token se potom pošle zpět uživateli a tím proces přihlašování na straně serveru končí.

Tím ale nekončí proces ověření identity uživatele při zasílání požadavků na jednotlivé endpointy aplikace (neboli *Autorizace*). Nestačí zkontrolovat existenci uživatele, ale důležité je také zkontrolovat oprávnění přistupovat k jednotlivým endointům aplikace. Například je nutné vědět, jestli tento uživatel má povolení vytvořit nový projekt nebo provést jakoukoliv další akci. Příkladem může sloužit načítání projektů v Ganttově diagramu. Uživatel odešle

[4] https://jwt.io/
[5] https://www.npmjs.com/package/bcrypt

požadavek na patřičný endpoint, ve kterém požadá o vracení seznamu projektů (v tomto případě se jedna o endpoint na adrese */projects*, nad kterým se zavolá funkce kontroléru `getProject()`. Než server tento požadavek přesměruje dál do databáze, zavolá se první middleware, který zkontroluje hlavičku `Authorization` v HTTP požadavku, kde se nachází unikátní token přihlášeného uživatele. Pomocí metody `jwt.verify()` se tento token dekóduje a vrátí informace, které jsou v tomto tokenu uložené (jedna se o email a userID). V případě úspěchu se poté zavolá funkce `next()`. Po úspěšném zpracování autentifikačního middleware se zavolá další, který má za úkol má zkontrolovat oprávnění uživatele existující projekty načíst. Kontrola probíhá pomocí ověření role uživatele. V případě úspěchu se spustí funkce kontroléru příslušné akce, čímž aplikace načte data z databáze, které poté vrátí uživateli ve tvaru grafické reprezentace projektů. Schématické znázornění procesu je na obrázku 6.2.

Obrázek 6.2: Zpracování požadavku klienta skrz middleware

Přidání nového uživatele do systému

Jelikož se do aplikace nelze zaregistrovat skrz registrační formulář, tato možnost byla přidělená správci systému. Správce je jediný uživatel, který může zaregistrovat nového uživatele do systému. Zároveň je jediný, kdo může uživatele ze systému smazat. Z hlediska implementace, je proces registrace nového uživatele podobný procesu přihlášení, ale s tím rozdílem, že nad modelem User se volá metoda `save()`, která vloží do databáze údaje o vytvořeném uživateli.

Po registraci nového uživatele na vyplněný ve formuláři email přijde zpráva s aktivačním odkazem a nastaveným heslem. Po aktivaci uživatel toto heslo může změnit. Proces aktivaci zpracovává funkce `activateAccount()`.

6.2 Front-end

Účelem front-endu je odesílání dat na server, zpracování a reprezentace příchozí odpovedi pro uživatele ve tvaru grafického uživatelského rozhraní. Komunikace klientu a serveru je implementovana v souboru */redux/api.js*, který se nachází v klientské části aplikace, v níž probíhá volání metod POST, GET, PUT a DELETE nad přislušnými endpointy aplikace. Tyto metody pak reprezentuji CRUD operace nad modely v databázi. Do implementace front-endu je také zahrnuta práce nad elementy grafického uživatelského rozhráni. Komunikace mezi klientem a serverem je asynchronní – to znamená, že prechody mezi stavy aplikace (načítání a ukládání dat) jsou plynulé pro uživatele a jsou znázorněné pomocí animováného točícího kolečka nebo jiné animace odpovídající přislušné akci.

React

Pro implementaci byla zvolená JavaScript knihovna React, která je součástí MERN stacku. Vyhodou je především možnost implementace jednolivých stranek a funkcionality pomocí

znovupoužitelných komponent, čímž se ušetřil čas při vývoji aplikace a ve výsledku kód je více čitelný. Komponenty jsou uloženy ve složce */components*, ve které jsou dále rozděleny do jednotlivých podsložek dle užití. Toto rozdělení napomáhá lepší orientaci ve struktuře projektu.

V Reactu lze vytvořit komponentu pomoci tříd anebo pomocí funkcí. Ve své práci jsem zvolil druhou variantu a tvořil jsem komponenty pomocí funkcí, jinými slovy *funkcionální komponenty*. Velkou výhodou takových komponent je možnost použití tzv. *Hooks*[6]. Jsou to primitiva, které umožňují jednoduše pracovat se stavem komponenty (*state*) a dalšími tzv. *lifecycle* metodami.

Jedním z nejpoužívanějších Hooks aplikace je `useState()`, který umožňuje pracovat se stavem komponenty. Používá se například při volání modálního okna, které se zobrazí po kliknutí na obdélník reprezentující projekt v Ganttově diagramu. Hook vrátí dvojici, kterou lze získat pomocí destrukturalizace. První parameter získané dvojíce obsahuje určitý stav, v tomto případě hodnoty atributů zvoleného projektu (na začátku mají prázdnou hodnotu). Druhým parametrem je funkce, kterou lze zavolat v jakemkoliv miste v kódu a předat ji nějakou hodnotu, která pak stane novým stavem. Po kliknuti na projekt se vyvolá posluchač (angl. *event listener*) `onClick()` v komponentě *ReactGantt*, který následně vyvolá ovladač (angl. *handler*) `passProject()`. Tento ovladač zpracuje získanou hodnotu aktuálního projektu (project) a předá ji pomocí `setActualProject(project)` jako nový stav do parametru `actualProject`. Získaný stav se poté předá do komponenty modálního okna, kde s nim již lze dal pracovat.

Dalšími nejpoužívanějšími hooks v aplikaci `jsou useEffect(), useSelector()` a `useDispatch()`.

React Bootstrap[7]

Pro tvoření dynamických komponent používám ve své práci framework React–Bootstrap. Jedná se o front-end framework, který byl založen na Bootstrap 4 komponentách, ovšem byla každá komponenta postavena od nuly jako skutečná React komponenta. Příkladem je modální okno */Modals/ModalProject.js*, které se objeví při kliknutí na jednotlivé projekty v diagramu.

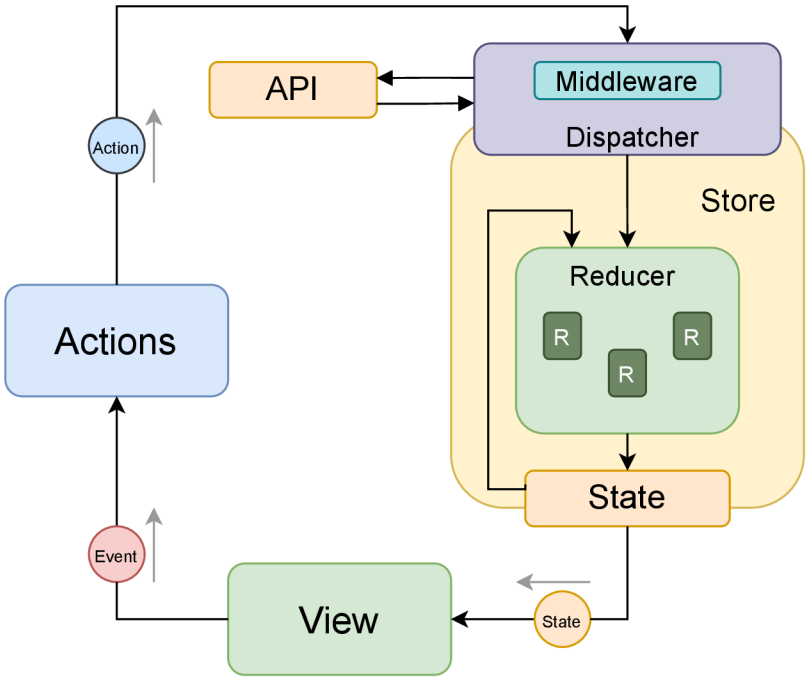
Zpracování stavů aplikace

Při implementaci aplikace bylo nutné nějakým způsobem vyřešit zpracování jednotlivých stavů aplikace. Na základě informací o existujících technologiích získaných z kapitoly 3 jsem rozhodl použít knihovnu Redux[8], která se nejčastěji používá společně s knihovnou React.

Pro správu uživatelského rozhraní je zapotřebí sledovat stav (angl. *state*) aplikace. Příkladem může sloužit okamžik, když uživatel chce vytvořit nový projekt a informovat o tom server, který tento projekt následně uloží do databáze. Jak již bylo zmíněno v kapitole 3, stav aplikace se nachází v jednom místě, které se nazývá *store*. V případě, že uživatel zvolí akce pro vytvoření projektu z formuláře, pomocí speciální metody `dispatch()` se vyvolá příslušná akce (*action*) `createProject(project)` v souboru */actions/projects.js*, která jako vstupní argument obdrži proměnnu `project`, ve které se nachází všechny informace o nově vytvořeném projektu. Následně se zavolá endpoint na serveru, který zpracovává požadavek

[6] https://reactjs.org/docs/hooks-intro.html
[7] https://react-bootstrap.github.io/
[8] https://redux.js.org/

a vrátí do proměnné `data`, do které se uloží informace o vytvořeném projektu. Tato proměnná se poté pomocí metody `dispatch()` a odpovídajícího typu akce `CREATE_PROJECT` předá do reduceru. Reducer, na základě získaného typu akce, provede zpracování informace z proměnné `data` a následně vrátí state do samotné aplikace. Proces zpracování akce je znázorněn na obrázku 6.3. Tímto operace tvoření nového projektu skončí a díky reduxu je zachován konzistentní stav aplikace se kterým lze dále pracovat.



Obrázek 6.3: Popis fungování Redux. Uživatel vyvolá určitý *event* který pak následně vyvolá odpovídající akci (*action*). Tato akce se předá do dispatcheru, kde se provede volání příslušného endpointu API. Po zpracování požadavku, action se dostane do reduceru, kde se provede následující zpracování na základě jeho typu. Výsledkem je nový stav aplikace (*state*). Zdroj [1]

Autentizace na straně klienta

Pro implementaci přihlašovacího systému existuje mnoho způsobů, ovšem lze uvést dva nejrozšířenější:

- pomocí HTTP cookies,
- použitím lokálního úložiště (*localStorage*) – uchování přihlašovacích údajů na zařízení klienta.

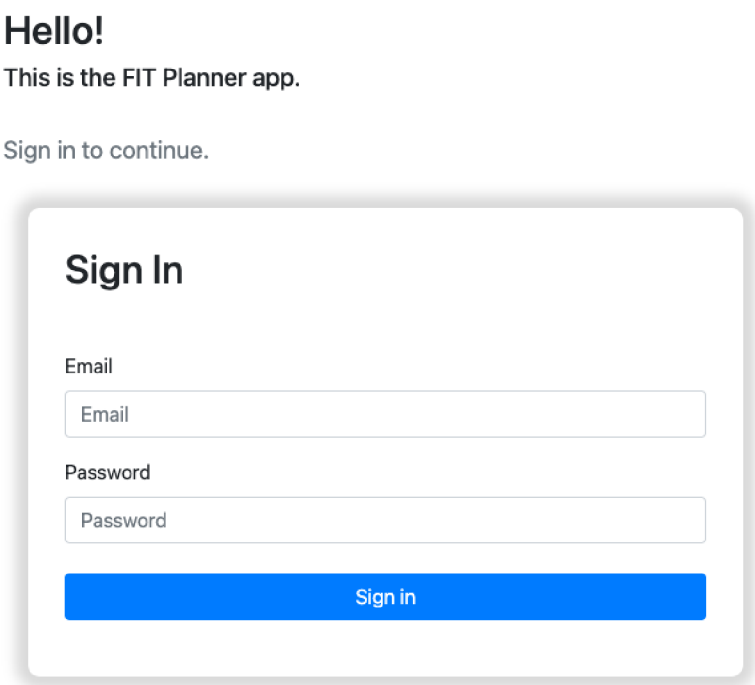
Na základě analýzy obou způsobů jsem zvolil implementaci pomocí localStorage. Aby se uživatel dostal do aplikace, musí vyplnit přihlašovací formulář, kam zadá email a heslo. Po kliknutí na tlačítko „Login“ se odešle požadavek na server, kde se provádí proces autentizace uživatele, který byl popsan v kapitole 6.1. Výsledkem je vygenerovaný JWT, který se vrátí zpatky uživateli a uloží se do lokálního úložiště v prohlížeči (localStorage) tímto

způsobem: `localStorage.setItem('profile', user)`, kde parameter *user* obsahuje informaci o uživateli ve tvaru JWT. Všechny budoucí požadavky klienta na server budou tento token obsahovat. Ovšem doba jeho platnosti je limitovaná na 2 hodiny, což znamená, že po uplynutí této doby se uživatel bude muset přihlásit znovu.

Pro vykonání jakékoliv činnosti v aplikaci, uživatele potřebuje vyvolat příslušnou akci s pomocí knihovny Reduxu (viz sekce 6.2). Funkce reprezentující tuto akci provede volání odpovídajícího endpointu a odešle požadavek na server. Jak již bylo zmíněno v sekci 6.1, společně s požadavkem se na server musí odeslat JWT, který se vytáhne z lokálního úložiště pomocí metody `localStorage.getItem('profile').token`. Pro přenos tokenu bylo použitá schéma Bearer[9]. Pokud při komunikaci dojde k chybě, server vrátí chybovou hlášku, jenž se zobrazí uživateli v aplikaci. Přihlašovací formulář aplikace lze vidět na obrázku 5.7.

Formik[10]

Formuláře jsou důležitou součástí aplikace, které se používají při přihlášení, vytváření projektů, událostí a nových uživatelů. Z toho důvodu by práce s tahovým prvkem měla být jednoduchá a pohodlná. Pro tyto účely byla zvolena knihovna Formik s využitím schématu *Yup* pro jednoduchou verifikaci jednotlivých formulářových polí. Vyhodou knihovny Fromik je to, že umožňuje vytvářet formuláře, aniž by bylo potřeba ručně spravovat hodnoty formulářů nebo psát obslužné rutiny událostí pro vstup. Implementaci formulářů lze nalézt ve složce */Forms*. Na obrázku 6.4 je uveden formulář pro přihlášení do aplikace.

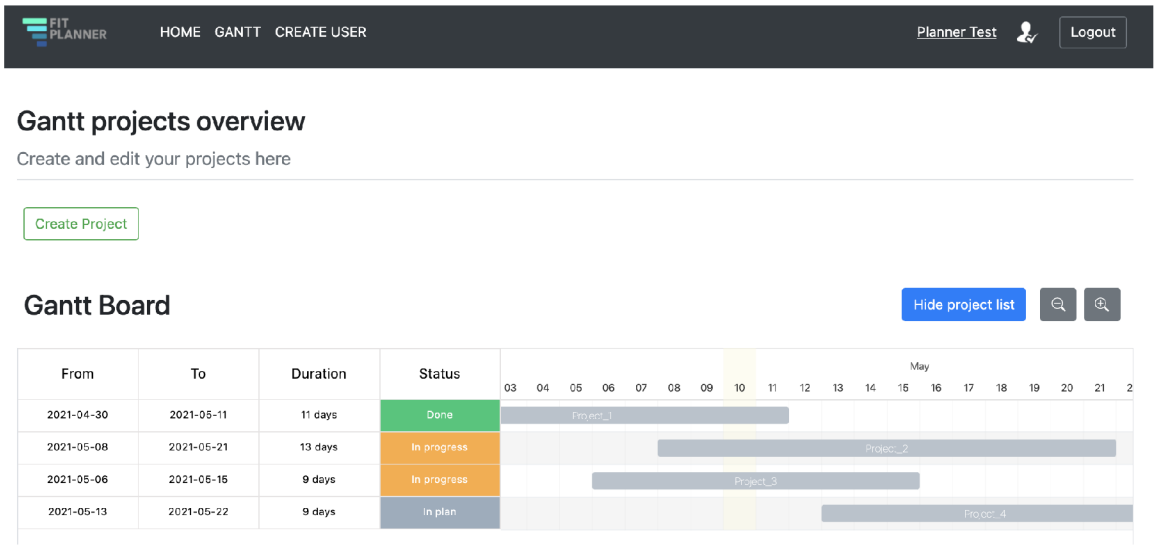


Obrázek 6.4: Přihlašovácí formulář aplikace

[9] https://jwt.io/introduction
[10] https://formik.org/

Ganttův diagram

Ganttův diagram je nejdůležitější částí implementace uživatelského rozhraní. Je to část aplikace, se kterou uživatel pracuje nejvíce. Na základě analýzy řešení knihoven pro implementaci Ganttova diagramu v prostředí React jsem zvolil knihovnu *gantt-for-react*[11], která umožňuje pohodlnou práci s diagramem. Tato knihovna je nadstavbou knihovny *Frappe-Gantt*[12] v jazyce JavaScript. Má otevřený kód, díky čemuž ji lze používat a modifkovat podle potřeb. Mezi hlavní výhody knihovny patří jednoduchost, příjemný vzhled a možnost interakce s jednotlivými prvky diagramu.



Obrázek 6.5: Finální vzhled aplikace

Aplikace obsahuje dva druhy diagramů: první – reprezentuje seznam projektů, druhý – reprezentuje seznam úkolů a události. Z hlediska implementace se tyto dva diagramy liší jen v jednom bodě, a to jsou vazby mezi jednotlivymi obdélníky, které jsou implementované pouze pro diagram obsahující události. Navíc obdélníky reprezentující úkoly mohou mít odlišné barvy na základě zvoleného typu. Pro lepší správu diagramů jsem rozdělil implementaci do dvou různých souborů – *Gantt.js* a *GanttTask.js*.

Zobrazení samotného diagramu zajišťuje komponenta **ReactGantt**, která obsahuje v sobě několik tzv. *props*[13], skrz které se volá API knihovny. Podstatnými propsy z hlediska funkčností diagramu jsou:

- **tasks** – přijímá pole projektů/událostí. Na základě tohoto parametru se vykresluje příslušný počet obdélníků do diagramu
- **viewMode** – přijímá pole hodnot *Quarter Day, Half Day, Day, Week, Month*. Na základě těchto hodnot lze provádět přibližování času (angl. *zoom*)

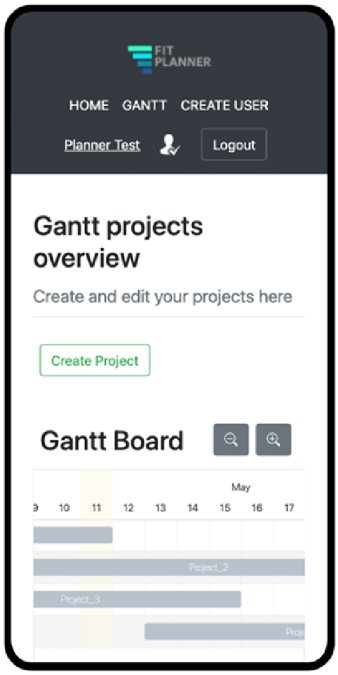
Komponenta ReactGantt umožňuje použít metody (`onClick()`, `onDateChange()`, `onTasksChange()`) pro vývoj a zlepšení funkcionality diagramu. Například v případě, že

[11] https://www.npmjs.com/package/gantt-for-react
[12] https://frappe.io/gantt
[13] Props – slouží pro předávání dat mezi různými komponenty aplikace

přetáhnutím projektu v diagramu uživatel změní termín začátku nebo konce projektu, metoda `onDateChange()` vrátí tyto obnovené informace, které lze odeslat na server, zpracovat a uložit do databáze. Tímto zaručí správné chování diagramu v případě interakce s uživatelem.

Dle požadavků uvedených v kapitole 4 bylo potřeba implementovat pomocnou tabulku vedle Ganttova diagramu, která by obsahovala základní informace o projektech. Jelikož knihovna takovou tabulku neposkytuje, implementoval jsem ji v pomocné komponentě **GanttTable**. Uživatel podle vlastních preferencí tuto tabulku může schovat pomocí tlačítka „Hide project list“ v horní části diagramu. Výsledný diagram lze vidět na obrázku 6.5.

Jedním z požadavků také bylo, aby aplikace byla přístupná z mobilu nebo tabletu. To znamená, že aplikace může být v plné míře použitelná, jak je to umožněné na její desktopové verzi. Tuto možnost jsem implementovat pomocí CSS stylů a knihovny React Bootstrap. Bohužel ale v mobilní verzi aplikace je uživatel schopen pouze posouvat (angl. *scrolling*) diagram, bez možnosti přetahovat jednotlivé projekty. Toto omezení je způsobeno samotnou knihovnou, která nepodporuje velké množství interakcí s diagramem na mobilním zařízení. Responzivní vzhled aplikace je reprezentován na obrázku 6.6.



Obrázek 6.6: Finální vzhled mobilní verze aplikace

Kapitola 7

Testování

Testování je nezbytnou součástí procesu vývoje aplikace. Díky tomu lze odhalit chyby během vývoje, a tím zaručit, že před nasazením a uvedením aplikace do provozu nedojde k neočekávaným výpadkům. Nejedná se o jednoduchý proces, proto jej může v praxi mnoho developerů podcenit, což ve výsledku vede k spoustě zbytečných chyb. Existuje několik různých kategorií testování, ovšem lze uvést ty základní a nejdůležitější: automatické testování a uživatelské testování. Tyto dvě kategorie budou popsané v následujících sekcích.

7.1 Automatické testování

Testy z této kategorie se provádějí s cílem odhalit chyby z hlediska funkcionality jednotlivých prvků aplikace. Jedná se v podstatě o psaní nového kódu, který by měl pomocí vestavěných funkci otestovat již existující, tzv. produkční funkce (neboli třídy). Automatické testy by se většinou měly provádět během vývoje a pokaždé, když se do projektu přidá nová funkcionalita.

Automatické testování se také dělí na několik typů. Jedná se o *Unit testy, integrační testy* a tzv. *End-to-End testy* (neboli E2E). Unit testy se používají převážně pro aplikace, které nemají externí závislosti (práce se soubory, databází apod.), nebo když stačí otestovat pouze určité funkce aplikace. DDalší typ testů, integrační, se na rozdíl od Unit testů používá pro aplikace, které mají externí závislosti. Jsou náročnější na tvorbu a trvají delší dobu. Poslední typ testů, E2E, se používá pro automatické testování uživatelského rozhraní. Pomocí speciálních nástrojů lze nahrávat a sledovat interakci uživatele s aplikací a poté provést analýzu a opravit případné chyby.

V případě mé aplikace jsem většinu automatických testů prováděl v posledních krocích implementace, a proto bylo prioritou otestovat především její nejdůležitější funkce. Podle již uvedeného popisu jednotlivých typů automatických testů jsem pro svou aplikaci vytvářel převážně integrační testy. Hlavním účelem těchto testů bylo zkontrolovat správnost komunikace serveru a databáze. Součástí této komunikace je například proces přihlašování do aplikace nebo proces vytváření a ukládání projektů.

Aplikace vyhověla základním automatickým testům, nicméně objevily se drobné chyby a nepřesnosti při volání některých API endpointů, zajmena v procesu přihlášení do systému. V jednotlivých případech server vracel špatné odpovědi, což vedlo k nesprávnému chování celého procesu autentizace. Z větší částí tyto chyby byly opravené a naslědně znovu otestované.

7.2 Uživatelské testování

Účelem uživatelského testování je to, aby aplikace byla otestována za reálných podmínek. To znamená, že v tento okamžik aplikace už disponuje určitou funkcionalitou a skutečný uživatel si to může vyzkoušet a označit své připomínky nebo nahlasit chyby. Pro to, aby tato fáze testování proběhla co nejlépe a výsledek byl co nejpřesněji, se uživateli většinou poskytné seznam určitých činnosti, které by se měl v rámci testování uskutečnit.

Aplikace byla testovaná ve spolupráci s VCIT na VUT FIT Brno. Uživatelský účet pro správce aplikace již byl předem vytvořen, stejně tak i několik testovacích projektů. Uživatel měl splnit několik jednoduchých kroků. Prvním je přihlášení do aplikace pomocí předem poskytnutého emailu a hesla. Po přihlášení měl uživatel na úvodní stránce zkontrolovat, zda má uvedenou správnou roli (Admin). Dále měl uživatel proklikat jednotlivé záložky a zkontrolovat, jestli obsahují příslušné informace. Dalším krokem bylo vytvoření nového projektu pomocí speciálního formuláře. Po vytvoření by se měl projekt objevit v Ganttově diagramu se zobrazením příslušného stavu. Jakmile byl projekt vytvořen, bylo nutné vytvořit události, které by tomuto projektu patřily. V tomto kroce bylo důležité ověřit, zda uživatel bude schopen jednotlivé události mezi sebou propojit, aby v diagramu bylo vidět vazby. Potom uživatel musel provést editaci projektů a událostí a následně je ze systému smazat. V posledním kroce uživatel musel vytvořit několik nových hostitelských účtů. Po vytvoření hostitelského účtu bylo potřeba se tímto účtem do aplikace přihlásit a zkontrolovat svou roli (Guest). Důležitou vlastností, kterou bylo potřeba ověřit, je zakaz smazání projektů a tvoření nových účtů.

Testování funkčnosti jednotlivých prvků aplikace objevilo několik chyb. Jedná z nich vznikla při ukládání souboru do databáze. Další dílčí chyby jsou byly způsobeny nepřesnostmi při implementování jednotlivých komponent a komunikaci mezi klientem a serverem.

7.3 Výsledky testování

Po dokončení automatických a uživatelských testů bylo zjištěno, že klíčové funkce aplikace fungují správně. Ovšem objevily se nepřesnosti u jednotlivých funkcí, které bylo potřeba z hlediska implementace opravit a doplnit. Interakce s aplikací byla snadná a přehledná, až na výjimku jednotlivých funkcí. Součástí testování byly podněty, které jsem vzal v potaz pro případ dalšího rozvoje a vývoje aplikace.

7.4 Možné zlepšení

Na základě provedených uživatelských a automatických testů lze dospět k závěru, že aplikace potřebuje určitá zlepšení. Především se jedná o funkcionalitu. Jednou z takových chybějících funkcí je možnost přidávat jednotlivé klienty. Současné řešení umožňuje uživateli informaci o klientovi zadat pouze jako parametr projektu. Tyto informace jsou ovšem značně omezené a jedná se pouze o název společnosti, pro kterou byl daný projekt vykonán. Vhodným zlepšením by byla možnost přidávat a uzpůsobovat podle potřeb různé typy projektů. Pro integraci aplikace do každodenních aktivit je vhodné implementovat možnost přidávat události spojené s projektem do vlastního Google kalendáře. Tento typ aplikace umožňuje implementovat velkou sadu dalších nových užitečných funkcí, které by ale měly být předem diskutováne s koncovým uživatelem systému.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat webovou aplikaci pro správu a řízení univerzitních projektů pomocí Ganttova diagramu. Nejprve bylo nutné se seznámit s problematikou správy univerzitních projektů a prozkoumat existující řešení. Požadavky na aplikaci byly konzultovány s ústavem VCIT na FIT VUT Brno. Na základě získaných informací byl proveden návrh architektury aplikace včetně grafického uživatelského rozhraní a modelování dat v databázi.

Po návrhu architektury bylo potřeba zvolit nástroje, pomocí kterých bude aplikace implementována. Z ohledem na třívrstvou architekturu aplikace byl zvolen technologický balíček MERN, do kterého patří nástroje Node.js jako běhové prostředí, Express.js jako aplikační rámec, knihovna React pro implementaci uživatelského rozhraní a nerelační databáze MongoDB.

Důležitým prvkem aplikace je Ganttův diagram, který slouží pro vizualizaci stavů jednotlivých projektů a znázorňuje závislostí mezi událostmi. Po vytvoření nového projektu v diagramu se objeví nový obdélník příslušný tomuto projektu. Na podobném principu je založené tvoření událostí v rámci jednotlivých projektů. Kromě Ganttova diagramu má uživatel (programový manažer) možnost získat základní informace o projektech pomocí dashboardu. Slouží především pro orientaci, poskytuje přehled nejdůležitějších informací bez nutnosti nahlížet do Ganttova diagramu. Velký důraz byl kladen na přehlednost a jednoduchost aplikace, protože uživateli musí být na první pohled jasné, jak lze projekt vytvořit nebo jak získat informace o událostech.

Po dokončení implementační části byla aplikace testována pomocí automatických a uživatelských testů. Systém vyhověl základním automatickým a uživatelským testům, nicméně aplikace měla nepřesnosti a chyby, které byly z větší části opravené.

Do budoucna bych měl zájem na vývoji aplikace pokračovat. S ohledem na výsledky uživatelského testování vidím možnost přidat další zlepšení a rozšíření, které by mohly aplikaci posunout na vyšší úroveň a tim poskytnou uživateli více možností pro práci na řízení projektů, programů a dalších podobných aktivitách.

Literatura

[1] Abramov, D. a spol. *Redux Fundamentals, Part 2: Concepts and Data Flow* [online]. Rudux.org, 2021 [cit. 2021-20-04]. Dostupné z: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow.

[2] apachebooster. *What is client-server architecture and what are its types?* [online]. Apachebooster, květen 2018 [cit. 2021-28-04]. Dostupné z: https://apachebooster.com/blog/what-is-client-server-architecture-and-what-are-its-types/.

[3] APM. *What is Gantt chart?* [online]. Association for project managment [cit. 2021-04-05]. Dostupné z: https://www.apm.org.uk/resources/find-a-resource/gantt-chart/.

[4] Bočková, K. H. *Projektové řízení: Učebnice*. E-knihy jedou. ISBN 9788075124319.

[5] Clark, W. *The Gantt Chart, a working tool of managment*. The Ronald Press Company, 1923. Dostupné z: https://archive.org/details/ganttchartworkin00claruoft/page/n7/mode/2up.

[6] Doležal, J. a kolektiv. *Projektový management podle IPMA - 2., aktualizované a doplněné vydání*. Grada Publishing a.s., 2012. ISBN 9788024742755.

[7] Doležal, J. a kolektiv. *Projektový management: Komplexně, prakticky a podle světových standardů*. Grada Publishing a.s., 2016. ISBN 9788027190669.

[8] Few, S. *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media, Incorporated, 2006. O'Reilly Series. ISBN 9780596100162.

[9] Hoque, S. *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js, 2nd Edition*. Packt Publishing, 2020. ISBN 9781839213113.

[10] IBM. *What is Three-Tier Architecture* [online]. IBM, říjen 2020 [cit. 2021-28-04]. Dostupné z: https://www.ibm.com/cloud/learn/three-tier-architecture.

[11] Havlíček, K. a spol. *Inovační strategie České republiky 2019–2030* [online]. 2019 [cit. 2021-01-05]. Dostupné z: https://www.vlada.cz/assets/urad-vlady/poskytovani-informaci/poskytnute-informace-na-zadost/Priloha_1_Inovacni-strategie.pdf.

[12] MDN contributors. *CSS: Cascading Style Sheets* [online]. 2021 [cit. 2021-29-04]. Dostupné z: https://developer.mozilla.org/docs/Web/CSS.

[13] MDN contributors. *HTML: HyperText Markup Language* [online]. 2021 [cit. 2021-29-04]. Dostupné z: https://developer.mozilla.org/docs/Web/HTML.

[14] MDN CONTRIBUTORS. *Understanding client-side JavaScript frameworks* [online]. 2021 [cit. 2021-29-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks.

[15] PITKA, E. *Řízení projektů vs. programů vs. portfolia* [online]. TAYLLORCOX s.r.o., 2019 [cit. 2021-01-05]. Dostupné z: https://www.tx.cz/blog/jaky-je-rozdil-mezi-projekty-programy-a-portfolio-managementem.

[16] SASIDHARAN, D. a KUMAR, S. *Full Stack Development with JHipster: Build full stack applications and microservices with Spring Boot and modern JavaScript frameworks, 2nd Edition.* Packt Publishing, 2020. ISBN 9781838820244.

[17] SCHRAMM, L. *Technological Innovation: An Introduction.* De Gruyter, 2017. ISBN 9783110429190.

[18] STEPHENS, R. a PLEW, R. *Database Design.* Pearson Education, 2000. Sams White Book. ISBN 9780134306568.

[19] SUBRAMANIAN, V. *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node.* Apress, 2017. ISBN 9781484226537.

[20] SVOZILOVÁ, A. *Projektový management: Systémový přístup k řízení projektů - 2., aktualizované a doplněné vydání.* Grada Publishing a.s., 2011. ISBN 9788024774282.

[21] TOMÁŠ, B., JIŘÍ, V., ALENA, B. a KOLEKTIV. *Tvorba informačních systémů.* Grada, 2012. Management v informační společnosti. ISBN 9788024741536.

[22] ZENDULKA, J. a RUDOLFOVÁ, I. *Databázové systémy IDS.* Brno: FIT VUT v Brně, červenec 2006 [cit. 2021-28-04].

Příloha A

Obsah přiloženého paměťového média

Přiložená SD karta má následující adresářovou strukturu:

- `Server/` – adresář obsahuje zdrojivé kódy serverové částí aplikace
- `Client/` – adresář obsahuje zdrojivé kódy klientské částí aplikace
- `Zpráva/` – adresář obsahuje zdrojivé kódy pro vytvoření technické zprávy
- `bp.pdf` – Technocká zpráva ve formatu PDF.
- `README` – Návod pro napojení a spuštění aplikace

Příloha B

Seznam koncových bodů aplikace

- User
  - POST */api/user/signin* – proces autentizace
  - POST */api/user/createuser* – proces vytvoření nového uživatele
  - GET */api/user/userslist* – vrací seznam uživatelů. kteří nemají Admin prava
  - DELETE */api/user/deleteuser* – smáže uživtele z databáze
- Project
  - GET */api/projects* – vrací seznam projektů
  - POST */api/projects* – vytvoří nový projekt a uloží ho do databáze
  - PATCH */api/projects/:id* – uprava proejktu s příslušným id
  - DELETE */api/projects/:id* – smáže projekt z databáze
- Task
  - GET */api/tasks* – vrací seznam událostí
  - POST */api/tasks* – vytvoří novou událost a uloží ji do databáze
  - PATCH */api/tasks/:id* – uprava události s příslušným id
  - DELETE */api/tasks/:id* – smáže událost z databáze
- File
  - GET */api/files* – vrací seznam souborů
  - POST */api/files* – vloží nový soubor do databáze
  - POST */api/files/:id* – stáženi příslušného souboru