UNIVERZITA PARDUBICE

NÁZEV FAKULTY

BAKALÁŘSKÁ / DIPLOMOVÁ PRÁCE

2022 Jméno a příjmení autora práce

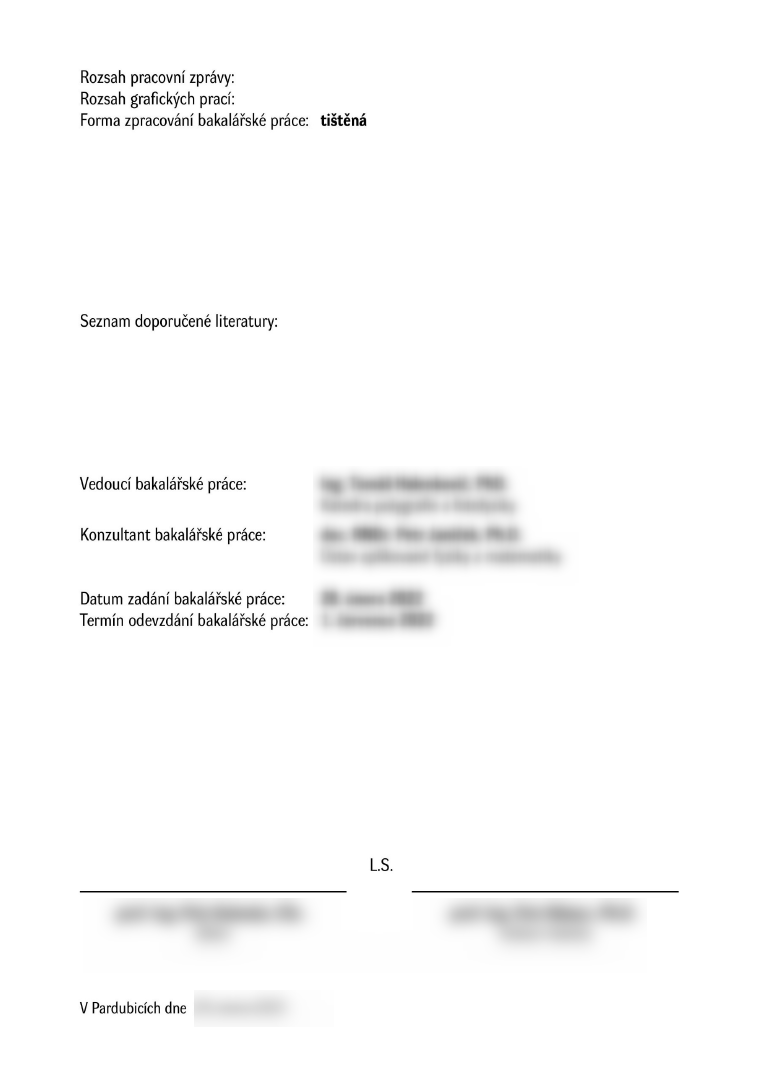
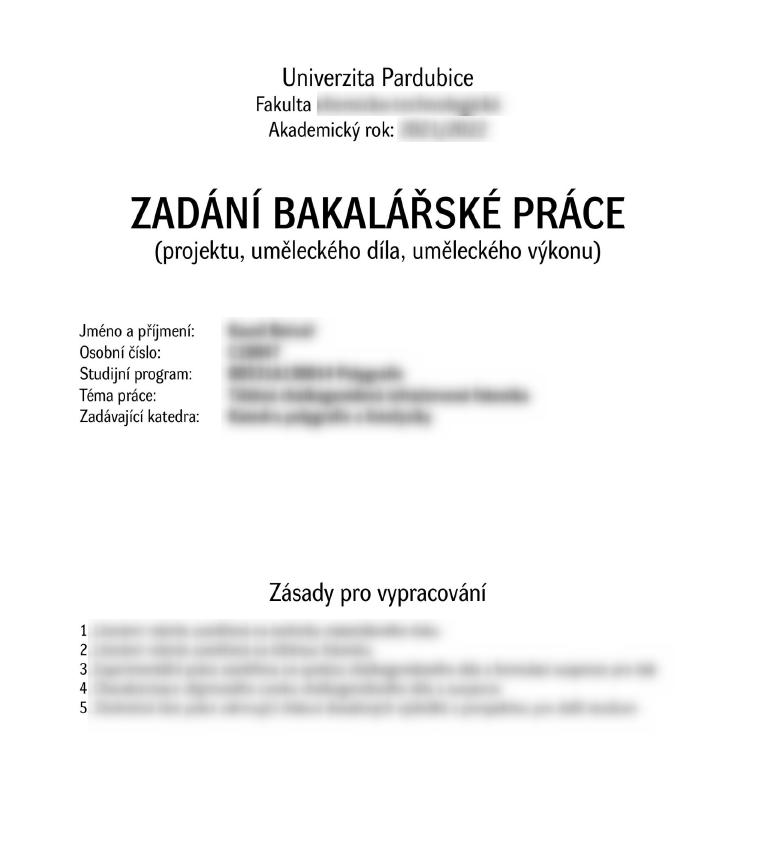
Univerzita Pardubice

Název fakulty

Název práce

Bakalářská / Diplomová práce

2022 Jméno a příjmení autora práce



Prohlašuji:

Práci s názvem ………………………………………………………………………………… jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne XX. XX. 20XX

Jméno a příjmení autora v.r.

**PODĚKOVÁNÍ**

**ANOTACE**

Červený se vedle modrého a zeleného zdá trochu menší, ale to je nejspíš jen optický klam, a i kdyby byl skutečně o něco málo menší, tak vážně jen o trošičku. Vítr skoro nefouká, a tak by se na první pohled mohlo zdát, že se balónky snad vůbec nepohybují. Jenom tak klidně levitují ve vzduchu. Jelikož slunce jasně září a na obloze byste od východu k západu hledali mráček marně, balónky působí jako jakási fata morgána uprostřed pouště.

**KLÍČOVÁ SLOVA**

slunce, vítr, balónky, barvy, pohyb, automatický generátor textu

**TITLE**

Red seems to be a little smaller next to blue and green

**ANNOTATION**

Red seems to be a little smaller next to blue and green, but that's probably just an optical illusion, and even if it's actually a little smaller, it's really just a tiny bit. The wind is barely blowing, and at first glance it might seem that the balloons are not moving at all. They just levitate in the air. As the sun shines brightly and you would be looking for a cloudless in the sky from east to west, the balloons act like a sort of mirage in the middle of the desert.

**KEYWORDS**

sun, wind, balloons, colors, motion, automatic text generator

**OBSAH**

[SEZNAM ILUSTRACÍ A TABULEK 11](#_Toc193744533)

[SEZNAM ZKRATEK A ZNAČEK 12](#_Toc193744534)

[TERMINOLOGIE 13](#_Toc193744535)

[ÚVOD 14](#_Toc193744536)

[1 Webová aplikace 15](#_Toc193744537)

[1.1. Architektura řešení při vývoji webových aplikací 15](#_Toc193744538)

[1.1.1 Architektonický vzor MVC 16](#_Toc193744539)

[1.1.2 Model-View-Presenter 17](#_Toc193744540)

[1.1.3 Model-View-ViewModel (MVVM) 18](#_Toc193744541)

[1.1.4 Model-View-ViewModel-Coordinator (MVVM-C) 18](#_Toc193744542)

[1.1.5 Monolitická architektura 19](#_Toc193744543)

[1.1.6 Mikroslužbová architektura 19](#_Toc193744544)

[1.1.7 Zhodnocení architektur 20](#_Toc193744545)

[2 Technologie používané při vývoji webových aplikací 20](#_Toc193744546)

[2.1 Backendové technologie 21](#_Toc193744547)

[2.1.1 Srovnání technologií pro backend 22](#_Toc193744548)

[2.2 Frontendové technologie 23](#_Toc193744549)

[2.2.1 Srovnání technologií pro frontend 24](#_Toc193744550)

[2.3 Relační databáze 25](#_Toc193744551)

[2.3.1 PostgreSQL 25](#_Toc193744552)

[2.4. Docker a Docker Compose 26](#_Toc193744553)

[2.5 Java Spring Boot 26](#_Toc193744554)

[2.5.1 Architektura a principy Spring Boot 26](#_Toc193744555)

[2.5.2 Výhody a přínosy použití Spring Boot 27](#_Toc193744556)

[2.5.3 Srovnání se standardním Spring frameworkem 27](#_Toc193744557)

[2.5.4 Omezení a perspektivy budoucího vývoje 27](#_Toc193744558)

[2.5.5 Z čeho se skládá Spring Boot projekt 27](#_Toc193744559)

[2.6 Vue.js 28](#_Toc193744560)

[2.6.1 Architektura a principy Vue.js 28](#_Toc193744561)

[2.6.2 Výhody a přínosy použití Vue.js 28](#_Toc193744562)

[2.6.3 Nedostatky Vue.js 29](#_Toc193744563)

[2.6.4 Z čeho se skládá Vue projekt 29](#_Toc193744564)

[EXPERIMENTÁLNÍ ČÁST 30](#_Toc193744565)

[3. Vybrané komunitní platformy a intranety 30](#_Toc193744566)

[3.1 Facebook 30](#_Toc193744567)

[3.2 Reddit 30](#_Toc193744568)

[3.3 Telegram 30](#_Toc193744569)

[3.4 LinkedIn 31](#_Toc193744570)

[3.5 Instagram 31](#_Toc193744571)

[4. Požadavky 31](#_Toc193744572)

[4.1 Funkční požadavky 32](#_Toc193744573)

[4.2 Nefunkční požadavky 32](#_Toc193744574)

[4.3 Databáze 33](#_Toc193744575)

[4.4 Diagramy aktivit 34](#_Toc193744576)

[4.4.1 Hlavní diagram 34](#_Toc193744577)

[4.4.2 Diagram autorizace 35](#_Toc193744578)

[4.4.3 Diagram fóra 36](#_Toc193744579)

[4.4.4 Diagram bazaru 37](#_Toc193744580)

[4.4.5 Diagram skupin 38](#_Toc193744581)

[1.1.3 Obhajoba volby modelu Web 2.0 38](#_Toc193744582)

[ZÁVĚR 42](#_Toc193744583)

[POUŽITÁ LITERATURA 43](#_Toc193744584)

# SEZNAM ILUSTRACÍ A TABULEK

[Obrázek 1: Balonky 14](#_Toc24459062)

[Tabulka 1: Příklad tabulky 15](#_Toc116377311)

# SEZNAM ZKRATEK A ZNAČEK

# TERMINOLOGIE

# ÚVOD

Očividně je uvázaná dostatečně pevně, protože balónky skutečně neucházejí. To ale není nic zvláštního. Překvapit by však mohl fakt, že nikdo, snad krom toho, kdo balónky k obloze vypustil, netuší, jakou má ona stuha barvu. Je totiž tak lesklá, že za světla se v ní odráží nebe a za tmy zase není vidět vůbec. Když svítí slunce tak silně jako nyní, tak se stuha třpytí jako kapka rosy a jen málokdo vydrží dívat se na ni přímo déle než pár chvil. Jak vlastně vypadají ony balónky? Ptají se často lidé.

Inu, jak by vypadaly – jako běžné pouťové balónky střední velikosti, tak akorát nafouknuté. Červený se vedle modrého a zeleného zdá trochu menší, ale to je nejspíš jen optický klam, a i kdyby byl skutečně o něco málo menší, tak vážně jen o trošičku. Vítr skoro nefouká, a tak by se na první pohled mohlo zdát, že se balónky snad vůbec nepohybují. Jenom tak klidně levitují ve vzduchu. Jelikož slunce jasně září a na obloze byste od východu k západu hledali mráček marně, balónky působí jako jakási fata morgána uprostřed pouště. Zkrátka široko daleko nikde nic, jen zelenkavá tráva, jasně modrá obloha a tři křiklavě barevné pouťové balónky, které se téměř nepozorovatelně pohupují ani ne moc vysoko, ani moc nízko nad zemí. Kdyby pod balónky nebyla sytě zelenkavá tráva, ale třeba suchá silnice či beton, možná by bylo vidět jejich barevné stíny – to, jak přes poloprůsvitné barevné balónky prochází ostré sluneční paprsky. Jenže kvůli všudy přítomné trávě jsou stíny balónků sotva vidět, natož aby šlo rozeznat, jakou barvu tyto stíny mají. Uvidět tak balónky náhodný kolemjdoucí, jistě by si pomyslel, že už tu takhle poletují snad tisíc let. Stále si víceméně drží výšku a ani do stran se příliš nepohybují.

Proti slunci to vypadá, že se slunce pohybuje k západu rychleji než balónky, a možná to tak skutečně je. Nejeden filozof by mohl tvrdit, že balónky se sluncem závodí, ale fyzikové by to jistě vyvrátili. Z fyzikálního pohledu totiž balónky působí zcela nezajímavě. Nejvíc bezpochyby zaujmou děti – jedna malá holčička zrovna včera div nebrečela, že by snad balónky mohly prasknout. A co teprve ta stuha. Stuha, kterou je každý z trojice balónků uvázán, aby se nevypustil. Očividně je uvázaná dostatečně pevně, protože balónky skutečně neucházejí. To ale není nic zvláštního.

# 1 Webová aplikace

Webová databázová aplikace představuje moderní softwarové řešení, které umožňuje uživatelům přistupovat k centralizované databázi prostřednictvím webového rozhraní. Tyto aplikace integrují správu dat a webovou prezentaci, což usnadňuje aktualizaci, údržbu i zabezpečení informací. V současnosti jsou webové databázové aplikace klíčovým prvkem při správě informací ve vzdělávacích institucích i firmách, protože umožňují dynamické a interaktivní služby založené na standardních protokolech, jako jsou HTTP/HTTPS, a využívají moderní technologie, například AJAX a RESTful API, pro efektivní komunikaci mezi klientem a serverem [1].

Moderní webové aplikace fungují na principu modelu „klient-server“ a využívají standardní protokoly HTTP/HTTPS, čímž zaručují spolehlivou a škálovatelnou komunikaci mezi uživatelem a serverem. Vývoj těchto aplikací je charakterizován přechodem od statických webových stránek k dynamickým systémům, které jsou postaveny na modulární architektuře umožňující snadnou integraci nových funkcí a efektivní škálovatelnost. Technologie AJAX umožňují aktualizaci částí stránky bez nutnosti jejího kompletního načtení, zatímco RESTful API standardizují výměnu dat mezi klientem a serverem, což usnadňuje vývoj a údržbu aplikací. Historický vývoj webových technologií dokládá postupný přechod od Web 1.0 k interaktivním řešením Web 2.0 a dále k experimentálním přístupům Web 3.0, přičemž nejrozšířenější a nejspolehlivější jsou právě aplikace založené na modelu Web 2.0 [3]. Moderní frameworky pro frontend, jako jsou Vue.js, React nebo Angular, v kombinaci s backendovými technologiemi, například Spring Boot nebo Laravel, umožňují vytvářet flexibilní, rychlé a spolehlivé aplikace, které splňují požadavky současného uživatele [1].

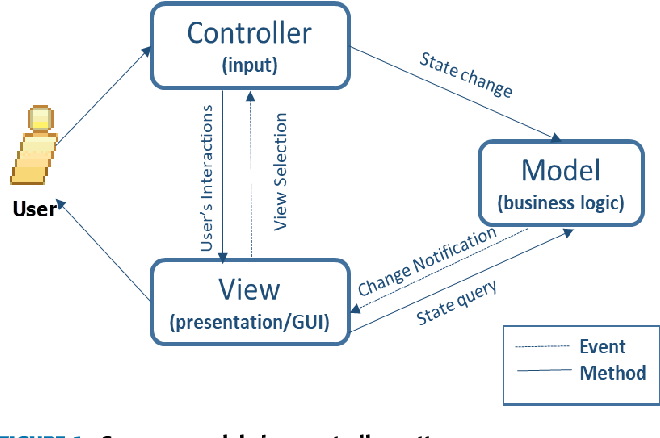
## 1.1. Architektura řešení při vývoji webových aplikací

Architektonická řešení ve vývoji webových aplikací hrají klíčovou roli při formování celkové struktury a funkčnosti systému. Správná volba architektury určuje nejen výkon a škálovatelnost aplikace, ale také její odolnost vůči změnám, snadnost údržby a možnost integrace nových komponent v budoucnu. V moderním světě, kde se software neustále vyvíjí, se architektonická rozhodnutí stávají základem pro rychlou reakci na nové obchodní požadavky a technologické výzvy. Je důležité si uvědomit, že architektura webové aplikace není jen souborem komponent, ale souhrnem podložených návrhových rozhodnutí, která ovlivňují celý životní cyklus produktu. Tento přístup umožňuje snížit pravděpodobnost vzniku chyb, minimalizovat náklady na údržbu a zajistit vysokou úroveň bezpečnosti dat, což je obzvláště aktuální pro systémy využívané ve vzdělávacím i korporátním prostředí [1].

## 1.1.1 Architektonický vzor MVC

Model-View-Controller (MVC) je základní architektonický vzor, který se stal nedílnou součástí vývoje moderních webových aplikací. Tento vzor rozděluje aplikaci do tří hlavních komponent – modelu, view a controlleru – přičemž každá má jasně vymezenou roli. Model představuje datovou a business logiku; stará se o správu, validaci a ukládání dat, a zároveň definuje pravidla, která s těmito daty souvisejí. View, tedy prezentační vrstva, zodpovídá za vizuální zobrazení informací, které jsou získány z modelu, a to ve formě uživatelského rozhraní (Guided User Interface, GUI). Controller slouží jako prostředník mezi modelem a view; přijímá vstupy od uživatele, interpretuje je a na jejich základě volá odpovídající metody modelu, přičemž následně vyvolává aktualizace ve view.

Toto oddělení odpovědností přináší řadu výhod. Díky jasnému rozdělení funkcí lze jednotlivé části aplikace vyvíjet a testovat nezávisle na sobě, což výrazně usnadňuje údržbu a škálovatelnost systému. Například změny v uživatelském rozhraní (view) nevyžadují zásahy do business logiky (model), a naopak. Tento přístup také umožňuje týmům pracovat efektivněji, neboť jednotliví vývojáři mohou soustředit svou pozornost na konkrétní komponenty bez nutnosti hlubokého porozumění celkové implementaci.



Obrázek 1*:* Common model-view-controller pattern [4]

Výzkumy, jako například studie klasifikace MVC softwarových aplikací pomocí samoorganizujících map (Self-Organizing Maps, SOM) [4], ukazují, že správná implementace MVC vzoru výrazně ovlivňuje kvalitu softwaru. Analýza MVC aplikací pomocí metrik jako počet řádků kódu, duplicity, složitost a designová kvalita poskytuje cenné informace o tom, jak různé implementační strategie ovlivňují udržovatelnost a robustnost systému. Tato analýza umožňuje identifikovat potenciální problémy (tzv. "code smells") a optimalizovat architektonická rozhodnutí, což je zásadní pro dlouhodobou úspěšnost a efektivitu softwarových řešení.

Z praktického hlediska je MVC vzor obzvláště cenný v prostředích, kde je potřeba zajistit vysokou míru interaktivity a dynamickou aktualizaci obsahu, což je typické pro moderní webové aplikace. Ve spojení s technologiemi jako AJAX pro asynchronní komunikaci a RESTful API pro standardizovanou výměnu dat poskytuje MVC pevný a flexibilní základ pro vývoj systémů, které jsou uživatelsky přívětivé a snadno udržovatelné [4].

1.1.2 Model-View-Presenter

Model-View-Presenter (MVP) je evoluční variantou klasického MVC, která se vyvinula s cílem snížit úzké propojení mezi prezentační vrstvou a obchodní logikou. V MVP je View definováno jako pasivní komponenta, která nezpracovává žádnou prezentační logiku, ale pouze zobrazuje data a předává uživatelské akce do Presenter. Presenter funguje jako prostředník, který přijímá vstupy z View, komunikuje s Modelem a následně aktualizuje View. Tímto způsobem dochází k jasnému oddělení odpovědností, což umožňuje lepší testovatelnost a údržbu aplikace.

Empirické studie zaměřené na výkon nativních Android aplikací ukazují, že MVP může mít výhodu v oblasti správy zdrojů – například některé experimenty uvádějí, že MVP architektura vykazuje nižší spotřebu paměti v porovnání s alternativními vzory, přičemž její jednoduchost implementace a testovatelnost patří mezi hlavní přednosti [8]. Nicméně, v praxi může dojít ke zvýšení složitosti kódu v důsledku hromadění obchodní logiky v Presenter komponentě, což může ztížit další rozvoj a údržbu [5].

## 1.1.3 Model-View-ViewModel (MVVM)

Model-View-ViewModel (MVVM) je architektonický vzor, který se stal oblíbeným zejména díky podpoře obousměrné datové vazby a reaktivního programování. V MVVM se aplikace dělí na tři hlavní komponenty: Model, který spravuje data a obchodní logiku; View, jež představuje uživatelské rozhraní; a ViewModel, který funguje jako prostředník mezi Modelem a View a transformuje data do formy vhodné pro zobrazení. Díky datové vazbě se změny v Modelu automaticky promítají do View a změny v uživatelském rozhraní jsou reflektovány ve ViewModelu bez nutnosti explicitní synchronizace.

Rozsáhlé literární přehledy a empirické studie, jako je například práce "Evaluating Android Architectural Patterns: A Deep Dive into MVP, MVVM, and MVI" [7], potvrzují, že MVVM zvyšuje testovatelnost a udržovatelnost aplikací. Použití reaktivní datové vazby přispívá k rychlejší odezvě a efektivnějšímu využití CPU, přičemž zároveň však může vést k mírně vyšší spotřebě paměti kvůli implementaci dodatečné knihovny pro datovou vazbu [8].

## 1.1.4 Model-View-ViewModel-Coordinator (MVVM-C)

MVVM-C představuje rozšíření standardního MVVM vzoru o dodatečnou komponentu – koordinátora (Coordinator), která se zaměřuje na správu navigačních toků a řízení přechodů mezi obrazovkami. V této architektuře zůstávají základní principy MVVM zachovány – Model spravuje data, View zobrazuje uživatelské rozhraní a ViewModel obsahuje prezentační logiku – avšak koordinátor zajišťuje centralizovanou správu navigace. Díky tomu může ViewModel zůstat čistý a zaměřený pouze na správu dat a obchodní logiku, aniž by se musel starat o implementaci přechodů mezi různými částmi uživatelského rozhraní.

Studie a disertační práce [5], poukazují na to, že integrace koordinátora může významně zlepšit modularitu a udržovatelnost aplikace. MVVM-C umožňuje centralizovat logiku navigace, čímž se snižuje duplicitní kód a minimalizuje riziko chyb při řízení toků mezi různými moduly. Tato architektura se ukazuje jako zvláště vhodná pro rozsáhlé projekty, kde je potřeba efektivně řídit interakce mezi komponentami a zároveň podporovat agilní vývoj a škálovatelnost [7].

## 1.1.5 Monolitická architektura

Monolitická architektura představuje tradiční přístup k vývoji webových aplikací, kdy je celá obchodní logika, správa dat a prezentace uživatelského rozhraní implementována v jediné kódové bázi. V tomto přístupu nejsou aplikace rozděleny do nezávislých služeb – všechny komponenty běží v rámci jednoho procesu a využívají společnou databázi. Mezi hlavní výhody monolitu patří jeho relativní jednoduchost při vývoji, testování, ladění a nasazení, protože není třeba implementovat složité mechanismy mezi-procesové komunikace. Studie uvedená v [9] dokládá, že v jedno-serverových nasazeních může monolit vykazovat vyšší výkon díky minimálním režijním nákladům spojeným s vnitřním voláním metod. Nicméně, s rostoucí velikostí a složitostí aplikace vznikají problémy se škálovatelností – změna v jednom modulu může ovlivnit celý systém a horizontální škálování často vyžaduje kompletní replikaci aplikace, což vede k neefektivnímu využití zdrojů [9]. Navíc, udržování modulární struktury a správa závislostí se stává náročnějším, což komplikuje adaptaci systému na měnící se požadavky [1], [3].

## 1.1.6 Mikroslužbová architektura

Na rozdíl od monolitické architektury mikroslužbový přístup rozděluje aplikaci na sadu nezávislých, samostatně provozovaných služeb, přičemž každá služba zodpovídá za konkrétní funkci. Každá mikroslužba může mít vlastní obchodní logiku, vlastní databázi a je vyvíjena a nasazována nezávisle na ostatních službách. Tento přístup umožňuje flexibilní škálování – je možné zvýšit počet instancí pouze u těch služeb, které jsou zatíženy nejvyšším počtem požadavků, což je klíčové pro rozsáhlé systémy s vysokou mírou paralelních dotazů [9]. Mikroslužby také zvyšují odolnost systému, protože selhání jedné služby neznamená ztrátu funkčnosti celé aplikace. Nicméně, distribuovaná povaha mikroslužeb přináší další složitosti, zejména pokud jde o komunikaci mezi službami, která může zvýšit režii a komplikovat monitorování, zejména při vysokém zatížení [9]. Použití standardních protokolů, jako jsou HTTP/HTTPS a RESTful API, pomáhá standardizovat výměnu dat mezi službami, avšak zároveň vyžaduje dodatečnou infrastrukturu pro orchestraci a správu služeb, například prostřednictvím API bran nebo systémů pro objevování služeb [2], [9].

## 1.1.7 Zhodnocení architektur

Výběr mezi monolitickou a mikroslužbovou architekturou závisí na mnoha faktorech, včetně velikosti a složitosti aplikace, požadavků na škálovatelnost a odolnost systému, a schopnosti týmu efektivně spravovat a vyvíjet systém. Jak uvádí [9], monolitická architektura může v jednoserverových prostředích vykazovat lepší výkon díky absenci nákladů na meziprocesovou komunikaci. Na druhé straně, u rozsáhlých systémů s dynamicky se měnícím zatížením nabízí mikroslužbový přístup značné výhody v oblasti škálovatelnosti a nezávislého řízení jednotlivých částí systému. Použití RESTful API a moderních technologií pro automatizaci a orchestraci umožňuje minimalizovat režijní náklady spojené s distribuovanou povahou mikroslužeb [3], [9].

Celkově lze tedy konstatovat, že volba architektury by měla být založena na specifických potřebách projektu, přičemž pro menší a jednodušší systémy může být vhodnější monolit, zatímco pro rozsáhlejší aplikace s vysokými nároky na škálovatelnost a odolnost se doporučuje mikroslužbový přístup.

Na základě přehledu výhod a nevýhod jednotlivých architektonických řešení vyplývá, že volba optimální architektury pro konkrétní projekt musí vycházet z jeho specifických požadavků, jako je velikost systému, očekávané zatížení a potřeba rychlé iterace. Pro projekt, který se zaměřuje na relativně malý až středně velký rozsah funkcí a nepočítá s masivním nárůstem uživatelské základny, se jako nejvhodnější volba jeví monolitická architektura. Díky své jednoduchosti, nižší režii interních volání a snadnější implementaci umožňuje rychlejší vývoj, testování a nasazení aplikace, což je klíčové pro dosažení stabilního a efektivního řešení. I když mikroslužby nabízejí výhody v oblasti škálovatelnosti a nezávislého nasazení jednotlivých komponent, pro náš případ představuje monolit pevný a udržitelný základ, na němž lze případně později stavět rozšíření nebo případnou migraci, pokud by se projekt rozrostl [3], [9].

# 2 Technologie používané při vývoji webových aplikací

Vývoj moderních webových aplikací je komplexní proces, který vyžaduje integraci řady technologií, přičemž klíčovým prvkem je správná volba technologického stacku. Tento stack zahrnuje frontendové technologie, které zajišťují interaktivní a responzivní uživatelské rozhraní, a backendové technologie, kde je implementována obchodní logika, správa dat a zabezpečení. Kromě toho hraje roli i integrace s databázemi a externími službami prostřednictvím RESTful API nebo jiných komunikačních protokolů. Taková řešení umožňují dynamickou výměnu informací mezi klientem a serverem a představují základ pro škálovatelné a udržitelné systémy [3] [9] [10].

## 2.1 Backendové technologie

Backendové technologie představují "motor" webové aplikace a mezi nejpoužívanější patří Node.js, PHP s webovými frameworky (např. Django, Flask) a robustní enterprise platformy. Node.js, postavený na jazyce JavaScript, využívá asynchronní, event-driven architekturu, která umožňuje efektivní zpracování velkého množství paralelních požadavků [10]. Díky svému neblokujícímu I/O modelu je Node.js ideální pro aplikace, které jsou náročné na vstupně-výstupní operace, a je často preferován v prostředích, kde je třeba rychlá odezva při vysoké konkurenci uživatelů. PHP je osvědčený skriptovací jazyk využívaný především u menších a středně velkých aplikací, neboť jeho jednoduchost a široká komunita umožňují rychlý vývoj základních funkcí. Nicméně, při vysokém zatížení může synchronní povaha PHP představovat omezení z hlediska škálovatelnosti a efektivity [1] [10]. Python se vyznačuje čitelností a rychlým vývojem, přičemž frameworky jako Django nebo Flask poskytují robustní strukturu pro tvorbu rozsáhlých aplikací, což usnadňuje integraci nových funkcí a umožňuje efektivní rozšiřitelnost díky bohaté knihovně a silné komunitní podpoře [1] [10]. Pro rozsáhlé podnikové aplikace jsou často využívány robustní platformy, jako jsou Spring Boot (Java) a MS.NET Core (Microsoft), které nabízejí integrovanou podporu pro ORM, správu databázových operací a RESTful API, což zajišťuje vysokou úroveň bezpečnosti, škálovatelnosti a výkonu i za podmínek extrémního zatížení [9] [12] [13].

### 2.1.1 Srovnání technologií pro backend

Tabulka 1: Srovnání backendových technologií. Zdroj:[1] [9] [10] [12] [13]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kritéria | Express (Node.js) | Laravel (PHP) | Spring Boot (Java) | Django (Python) |
| Výkon a škálovatelnost | Asynchronní architektura umožňuje efektivní využití zdrojů a nižší latence při vysoké konkurenci [10]. | Vhodné pro menší projekty, optimalizační výhody pro náročné I/O operace chybí [1]. | Zajišťují stabilní a konzistentní výkon i při extrémním zatížení [9] [12] [13]. | Při rostoucí složitosti může být omezená škálovatelnost [1] [10]. |
| Vývojářská přívětivost a údržba | Podporuje modulární struktury a udržitelnost rozsáhlých systémů [1] [10]. | Rychlý vývoj a jednoduchá údržba usnadňují tvorbu prototypů, avšak limity se projevují při rozrůstání projektu [1] [10]. | Nabízí pokročilé bezpečnostní mechanismy a dlouhodobou údržbu komplexních systémů [9] [12] [13]. | Ideální pro prototypování, ale složitější projekty vyžadují robustnější řešení [1] [10]. |
| Komplexnost a integrace | Vhodné zejména pro I/O náročné aplikace [10]. | Omezená flexibilita při integraci více databází a externích API [1]. | Disponují pokročilými mechanismy pro správu transakcí a efektivní komunikací mezi komponentami [9] [12] [13]. | Jednoduché implementační řešení, ale pro komplexní integrace je často třeba doplňkových nástrojů [1] [10]. |
| Náklady a dostupnost odborných znalostí | Vyžaduje specifické znalosti, nabízí vysokou flexibilitu a potenciál pro budoucí rozšiřitelnost [1] [9]. | Díky dlouhé historii a široké komunitní podpoře je snadnější získat kvalifikované odborníky [1] [9]. | I když vyžadují specializované znalosti, jsou považovány za investici do dlouhodobého rozvoje [1] [9]. | Dobře zavedené technologie s velkou komunitní podporou, které však mohou mít omezení u extrémně náročných systémů [1] [9]. |

Celkově lze konstatovat, že volba optimální backendové technologie závisí na specifických požadavcích daného projektu. Pro aplikace s vysokým počtem souběžných uživatelů a náročnými I/O operacemi je vhodné využít asynchronních platforem, jako je Express (Node.js), které zajišťují nízkou latenci a efektivní využití zdrojů, zatímco rozsáhlé podnikové systémy s přísnými požadavky na bezpečnost, robustnost a integraci se nejlépe realizují pomocí enterprise řešení, například Spring Boot. Menší a jednodušší projekty pak mohou být efektivně vyvíjeny s využitím frameworku Laravel (PHP), který podporuje rychlý vývoj, nebo Django (Python), jenž je vhodný zejména pro prototypování a méně náročné aplikace [1] [9] [10] [12] [13].

## 2.2 Frontendové technologie

Frontendové technologie představují klíčovou složku vývoje moderních webových aplikací, neboť zajišťují interaktivní a uživatelsky přívětivé rozhraní, které reaguje na dynamické změny v reálném čase. Mezi nejrozšířenější patří JavaScriptové frameworky, jako jsou React.js, Angular.js a Vue.js, které umožňují vytváření komplexních a škálovatelných single-page aplikací (SPA). Tyto technologie využívají pokročilé techniky, například virtuální DOM pro efektivní aktualizaci uživatelského rozhraní, obousměrnou datovou vazbu, která zajišťuje synchronizaci mezi modelem a zobrazením, a komponentový přístup, jenž podporuje opětovné použití kódu. Například Vue.js je založen na architektuře MVVM, která výrazně zjednodušuje správu stavu aplikace a umožňuje plynulé oddělení prezentační logiky od obchodní logiky [15]. Díky těmto vlastnostem se frontendové technologie staly nezbytným nástrojem pro zajištění kvalitní uživatelské zkušenosti a rychlého načítání obsahu, což je zvláště důležité v době, kdy očekávání uživatelů ohledně rychlosti a interaktivity webových aplikací neustále rostou [14].

### 2.2.1 Srovnání technologií pro frontend

Tabulka 2: Srovnání frontendových technologií. Zdroj: [14] [15] [19]

|  |  |  |  |
| --- | --- | --- | --- |
| Kritéria | Angular.js | React.js | Vue.js |
| Robustnost | Robustní a komplexní rámec vhodný pro rozsáhlé podnikové aplikace | Flexibilní řešení pro rychlý vývoj modulárního kódu | Lehké a přehledné řešení ideální pro menší až středně velké projekty |
| Učební křivka | Strmější učební křivka s vyššími nároky na konfiguraci | Nízká učební křivka, což usnadňuje rychlý start | Velmi přívětivá pro začátečníky díky jednoduchosti a architektuře MVVM |
| Správa stavu | Podporuje komplexní správu stavu vhodnou pro náročnou logiku | Efektivní správa stavu umožňuje rychlou aktualizaci a údržbu kódu | Reaktivní datová vazba usnadňuje synchronizaci dat mezi komponentami |
| Komunita a podpora | Silná komunita a rozsáhlá podpora, ideální pro velké projekty | Velká a aktivní komunita | Rostoucí a aktivní komunita |
| Rychlost vývoje | Vyžaduje více konfigurace, což může zpomalit vývoj, zejména u menších projektů | Velmi agilní, vhodné pro prototypování a rychlý vývoj | Rychlý vývoj díky nízké složitosti kódu a jednoduché konfiguraci |
| Dokumentace | Podrobná a komplexní dokumentace, usnadňuje pochopení složitých konceptů, ale může být náročná na studium | Dobrá dokumentace s rozsáhlým ekosystémem knihoven, avšak některé aspekty se spoléhají na komunitní zdroje | Výjimečně přehledná a srozumitelná dokumentace, usnadňuje učení a implementaci, |

Srovnání jednotlivých frontendových technologií odhaluje, že každý framework má své specifické přednosti i omezení, které ovlivňují volbu vhodného řešení pro daný projekt. Rozsáhlé a komplexní aplikace často těží z robustního rámce Angular.js, zatímco agilní vývoj modulárního kódu je efektivně řešen pomocí React.js. Vue.js pak kombinuje jednoduchost a přehlednost s moderními prvky, jako je reaktivní datová vazba, což jej činí ideální volbou pro menší a středně velké projekty, kde je kladen důraz na rychlost vývoje a nízkou složitost kódu [14] [15] [19].

## 2.3 Relační databáze

Relační databáze představují základ moderní správy dat, jejichž koncepty se ustálily již více než padesát let od zásadního příspěvku E.F. Codda, který v roce 1970 definoval relační model, založený na reprezentaci dat v řádcích a sloupcích. Tento model umožnil oddělit logickou strukturu dat od jejich fyzické reprezentace a otevřel cestu pro deklarativní dotazovací jazyk SQL, který si dodnes udržuje dominantní postavení díky své jednoduchosti a efektivitě při zpracování dotazů [16]. Historický vývoj databázových systémů zaznamenal postupný přechod od hierarchických a síťových modelů, které byly omezeny svou flexibilitou, k relačnímu přístupu, jenž umožnil optimalizaci vyhledávání i manipulaci s daty pomocí sofistikovaných optimalizátorů, které překládají dotazy do efektivních exekučních plánů [16]. Navzdory kritikám, například kvůli absenci úplné ortogonality či problémům s null hodnotami, se relační databáze staly osvědčeným a široce používaným řešením díky své robustnosti, konzistenci a možnosti správy velkých objemů dat ve vysoce konkurenčním prostředí [16]. S nástupem nových požadavků na horizontální škálovatelnost a dynamickou správu dat s flexibilním schématem se na trhu objevují i systémy NoSQL, které částečně uvolňují některá omezení relačních databází, avšak díky dlouhodobému využívání a široké podpoře se relační databáze, jako MySQL, PostgreSQL, Microsoft SQL Server, SQLite či Oracle Database, stále řadí mezi nejpoužívanější systémy pro stěžejní aplikace [16].

### 2.3.1 PostgreSQL

PostgreSQL je open-source relační databázový systém, který je známý svou robustností, vysokou úrovní standardizace a rozšířenými funkcemi, jež umožňují efektivní správu komplexních datových operací [20]. Tento systém podporuje pokročilé SQL funkce, jako jsou rekurzivní dotazy, window funkce či možnost definice vlastních datových typů, což jej činí ideálním řešením pro aplikace vyžadující vysokou datovou integritu a flexibilitu [20]. Ve srovnání s MySQL se PostgreSQL vyznačuje lepší schopností zvládat složitější datové struktury a náročné transakční operace, což potvrzují i srovnávací studie zaměřené na výkonnost a charakteristiky jednotlivých databázových systémů [21]. Díky aktivní komunitě, rozsáhlé dokumentaci a pravidelným aktualizacím se PostgreSQL stává stabilní volbou pro aplikace, kde je klíčové zajistit konzistenci a bezpečnost dat. Tento systém tak představuje komplexní řešení, které kombinuje vysoký výkon s flexibilitou a rozšiřitelností, a je proto preferovaným nástrojem pro webové aplikace.

## 2.4. Docker a Docker Compose

Docker je open-source platforma, která umožňuje vývoj, nasazení a správu aplikací pomocí kontejnerů. Kontejnery představují přenosná a izolovaná prostředí, ve kterých aplikace běží konzistentně napříč různými systémy a infrastrukturami, což výrazně zjednodušuje migraci a škálování software [17] Docker umožňuje vývojářům plně se soustředit na psaní kódu, protože eliminuje nutnost řešení složitých konfigurací operačního systému, což výrazně zkracuje dobu vývoje a nasazení aplikací [17].

Docker Compose pak rozšiřuje funkčnost Dockeru tím, že umožňuje definovat a koordinovat vícekontejnerové aplikace prostřednictvím jediného konfiguračního souboru. Tento nástroj usnadňuje správu závislostí mezi jednotlivými službami, jako jsou databáze, webové servery a další komponenty, a automatizuje jejich nasazení. Díky vzájemné komunikaci mezi kontejnery lze komplexní systémy spravovat mnohem efektivněji, aniž by bylo třeba každou službu nastavovat ručně [18].

Celkově Docker a Docker Compose představují moderní přístup k vývoji a správě softwaru, který podporuje agilní metody, kontinuální integraci a nasazení (CI/CD) pro DevOps flow a umožňuje rychlou adaptaci na měnící se požadavky trhu. Platforma Docker usnadňuje efektivní využití hardwarových zdrojů a zvyšuje spolehlivost a konzistenci běhu aplikací, zatímco Docker Compose umožňuje koordinovat více kontejnerových služeb do jednotného, snadno spravovatelného celku [17] [18]. Tímto způsobem jsou tyto nástroje flexibilní a umožňují vysokou škálovatelnost.

## 2.5 Java Spring Boot

Spring Boot je framework vyvinutý společností Pivotal, který usnadňuje vývoj a nasazení aplikací založených na platformě Spring. Díky své auto-konfigurační povaze a přednastaveným „starterům“ umožňuje vývojářům rychle vytvářet samostatné, produkčně připravené aplikace s minimálním množstvím konfigurace.

### 2.5.1 Architektura a principy Spring Boot

Spring Boot staví na robustní architektuře tradičního Spring frameworku, avšak výrazně zjednodušuje konfiguraci díky mechanismu automatické konfigurace. Ten na základě dostupných knihoven a aktuálního kontextu prostředí volí vhodné nastavení, čímž zajišťuje optimální běh aplikace. Tato architektura podporuje vývoj mikroslužeb a modulárních systémů, přičemž umožňuje jejich snadnou správu a škálování. Auto-konfigurační moduly, tzv. „starters“, a vestavěný server (např. Tomcat) eliminují potřebu složitého manuálního nastavování, což výrazně urychluje celý proces vývoje a nasazení aplikací [22].

### 2.5.2 Výhody a přínosy použití Spring Boot

Díky auto-konfiguraci a přednastaveným konfiguracím se vývojáři mohou soustředit přímo na psaní obchodní logiky, což značně rychlost vývoje. Samostatné aplikace, které lze spustit jako běžné Java programy, usnadňují nasazení a škálování aplikací v cloudovém prostředí. Spring Boot taky umožňuje snadnou integraci s dalšími technologiemi, jako jsou databáze, messaging systémy a další enterprise systémy, což z něj činí vhodné řešení pro komplexní aplikace. Díky rozsáhlé dokumentaci, četným tutoriálům a aktivní komunitě je řešení Spring Boot dobře podporováno a aktualizováno, což zajišťuje dlouhodobou udržitelnost projektů [23].

### 2.5.3 Srovnání se standardním Spring frameworkem

Spring Boot představuje rozšíření standardního Spring frameworku, přičemž hlavní rozdíl spočívá v míře automatizace konfigurace a jednodušším nasazení aplikací. Zatímco tradiční Spring aplikace vyžadují podrobné XML konfigurace nebo rozsáhlé konfigurace pomocí anotací, Spring Boot tyto komplikace odstraňuje díky svému auto-konfiguračnímu mechanismu. To umožňuje rychlejší začátek vývoje a snížení rizika chyb při konfiguraci. Navíc, díky vestavěnému serveru a přednastaveným závislostem (starters) se vývojáři nemusí starat o kompatibilitu verzí a konfiguraci externích knihoven, což usnadňuje jak vývoj, tak i následnou údržbu aplikací [22] [24].

### 2.5.4 Omezení a perspektivy budoucího vývoje

Ačkoliv Spring Boot výrazně zjednodušuje vývoj a nasazení aplikací, existují určité oblasti, kde může být náročnější přizpůsobit framework specifickým potřebám. Například u velmi komplexních aplikací může být auto-konfigurace omezující, pokud je třeba provést jemné ladění konfigurace nebo specifické optimalizace. Nicméně, díky otevřenému zdrojovému kódu a aktivní komunitě se tyto problémy řeší pravidelnými aktualizacemi a vylepšeními. Perspektivy budoucího vývoje jsou slibné, neboť Spring Boot neustále rozšiřuje své možnosti integrace s moderními cloudovými platformami a nástroji pro mikroslužby, což z něj činí klíčovou technologii pro enterprise vývoj i nadále [22] [24].

2.5.5 Z čeho se skládá Spring Boot projekt  
Typická struktura projektu ve Spring Boot je navržena tak, aby podporovala logické rozdělení kódu a usnadňovala jeho správu. Hlavním vstupním bodem je třída obsahující metodu main, která se nachází v hlavním balíčku a inicializuje Spring kontext. Zdrojový kód se obvykle umisťuje do složky ***src/main/java***, zatímco konfigurační soubory, jako je application.properties nebo application.yml, jsou umístěny v adresáři ***src/main/resources***. Součástí projektu jsou také složky pro statické zdroje, jako jsou obrázky a styly, a pro šablony, pokud aplikace využívá serverové renderování. Projekt je obvykle organizován do logických vrstev, kde jsou odděleny kontrolery, služby, repozitáře a modely, což přispívá k modularitě a udržitelnosti kódu. Testy se pak nacházejí v adresáři ***src/test/java***, což umožňuje automatizované ověřování funkčnosti aplikace. Tato struktura je často generována pomocí nástroje Spring Initializr, který poskytuje vývojářům správně nakonfigurovaný základ pro jejich projekty, a je podporována rozsáhlou dokumentací a aktivní komunitou, což zvyšuje produktivitu a kvalitu vyvíjeného software [22] [23] [24].

## 2.6 Vue.js

Vue představuje moderní přístup k vývoji uživatelských rozhraní, který klade důraz na jednoduchost, modularitu a vysokou výkonnost díky svým inovativním principům a architektonickým řešením [19].

### 2.6.1 Architektura a principy Vue.js

Vue je postaveno na architektuře MVVM, která zajišťuje oddělení datové logiky od prezentační vrstvy prostřednictvím reaktivní datové vazby, což umožňuje automatickou aktualizaci uživatelského rozhraní při změnách v datech. Framework využívá virtuální DOM, který optimalizuje manipulaci se skutečným DOM a tím zvyšuje výkon aplikací. Dále je Vue založeno na komponentově orientovaném přístupu, kdy každá komponenta obsahuje šablonu, skript a styly v jednom souboru, což podporuje modularitu a opakované použití kódu. Vue také podporuje TypeScript, což přináší statickou kontrolu a robustnější vývoj, a lze jej využít i pro tvorbu mobilních aplikací prostřednictvím platformy Weex [25].

### 2.6.2 Výhody a přínosy použití Vue.js

Vue nabízí vývojářům značnou flexibilitu a rychlost vývoje díky intuitivní syntaxi a přehledné dokumentaci, která usnadňuje rychlý start i pro začátečníky. Reaktivní systém frameworku zajišťuje, že změny v datech se okamžitě projeví v uživatelském rozhraní, což vede k hladkému a efektivnímu chodu aplikací. Díky své komponentové struktuře lze snadno spravovat i rozsáhlé projekty, aniž by se zvyšovala složitost kódu, což je velkou výhodou při vývoji menších i středně velkých aplikací [26].

### 2.6.3 Nedostatky Vue.js

I přes řadu výhod má Vue i své omezení. Ve srovnání s některými konkurenty může být ekosystém a komunita menší, což může vést k omezené dostupnosti specializovaných knihoven a rozšíření. Další výzvou je rychlý vývoj frameworku, který vede k častým aktualizacím a nutnosti přizpůsobovat stávající projekty novým verzím, což může komplikovat dlouhodobou údržbu. Tyto faktory mohou představovat určité potíže zejména v komplexnějších enterprise projektech [26].

### 2.6.4 Z čeho se skládá Vue projekt

Typický Vue projekt je strukturován tak, aby podporoval modulární vývoj a efektivní správu kódu. Hlavním vstupním bodem je soubor ***main.js*** (nebo ***main.ts***, pokud je využit TypeScript), kde se inicializuje instance Vue a nastavují se globální konfigurace, včetně integrace routeru a správy stavu, například pomocí Vuex. Složka komponent obsahuje jednotlivé .vue soubory, kde každá komponenta zahrnuje šablonu, logiku a styly, což umožňuje opakované použití a izolaci jednotlivých částí uživatelského rozhraní. Dále projekt obvykle obsahuje složku pro definici tras, která zajišťuje navigaci mezi komponentami, a složku s assety, kde jsou umístěny statické zdroje jako obrázky, styly a další soubory [25].

# EXPERIMENTÁLNÍ ČÁST

# 3. Vybrané komunitní platformy a intranety

Intranet představuje vnitřní síť, určenou pro výměnu informací, spolupráci a automatizaci procesů uvnitř organizace nebo vzdělávací instituce. Díky omezenému přístupu k těmto systémům je zajištěna vysoká úroveň bezpečnosti a centralizovaného řízení dat, což umožňuje snadnou moderaci obsahu a rychlé aktualizace informací. To je zvláště důležité pro vzdělávací instituce, kde má ochrana osobních údajů studentů a učitelů zásadní význam. Možnosti a výhody intranetových systémů jsou široce reflektovány v odborné literatuře a standardech, což potvrzuje jejich význam v moderních informačních infrastrukturách [1]

## 3.1 Facebook

Je globální sociální síť, kterou využívají jednotlivci, rodiny, firmy, neziskové organizace i vládní instituce. Uživatelé si zde vytvářejí osobní profily, sdílejí fotografie, videa a příspěvky, komunikují prostřednictvím soukromých zpráv a mohou se zapojovat do skupin, které podporují komunitní interakci. Kromě toho Facebook nabízí funkci Marketplace, která umožňuje nákup a prodej zboží přímo na platformě, a řadu dalších nástrojů pro event management, reklamu a analytiku. Přestože platforma poskytuje rozsáhlé možnosti, čelí kritice zejména z hlediska ochrany osobních údajů a moderace obsahu, což může představovat omezení pro některé uživatele [27].

## 3.2 Reddit

představuje komunitně orientovanou diskusní platformu, kterou využívají studenti, nadšenci, profesionálové a experti z různých oborů. Uživatelé Redditu se zapojují do diskusí ve specializovaných fórech, tzv. subredditech, kde mohou publikovat příspěvky, hlasovat a komentovat obsah. Tato platforma umožňuje relativní anonymitu a podporuje živé a dynamické diskuse, což je výhodné pro sdílení znalostí a názorů. Na druhou stranu však může být Reddit náchylný k problémům s moderací a občas nese riziko šíření kontroverzního nebo polarizujícího obsahu [28].

## 3.3 Telegram

Komunikační aplikace, kterou využívají jednotlivci, skupiny, firmy, a dokonce i vládní agentury, a je známá především svou vysokou úrovní zabezpečení a šifrování. Uživatelé Telegramu mohou využívat nejen standardní funkce pro posílání zpráv, ale také skupinové chaty, kanály pro vysílání informací a integrované miniaplikace (tg Mini Apps), které umožňují provádět například nákupy nebo rezervace přímo v rámci aplikace. Tato rozšířená funkčnost podporuje inovativní využití platformy, ale může také čelit regulacím v určitých zemích a potenciálním bezpečnostním rizikům, zejména v případě veřejně přístupných kanálů [29].

## 3.4 LinkedIn

Profesionální sociální síť, kterou používají pracovníci, personalisté, odborníci a firmy pro budování kariérních kontaktů a profesní komunikaci. Platforma umožňuje vytváření detailních profilů, sdílení odborných článků a informací o pracovních příležitostech, a také podporuje účast ve skupinách zaměřených na konkrétní odvětví. Mezi klíčové funkce patří možnost vyhledávání pracovních nabídek, navazování kontaktů a budování profesionálních sítí, což značně usnadňuje nábor a kariérní rozvoj. Omezení LinkedInu spočívají zejména ve formálním prostředí, které může omezovat osobní interakce ve srovnání s volnějšími sociálními sítěmi [30].

## 3.5 Instagram

Instagram je vizuálně orientovaná sociální síť, kterou využívají jednotlivci, firmy a influenceři pro sdílení fotografií, videí a příběhů. Uživatelé zde vytvářejí atraktivní profily a prostřednictvím funkcí jako Stories, Reels nebo živých přenosů mohou efektivně komunikovat se svým publikem. Platforma nabízí také řadu nástrojů pro podniky, jako jsou analytika, placené reklamy a možnost správy kampaní, což umožňuje značkám lépe cílit na své zákazníky a monitorovat interakce s obsahem. Přestože Instagram podporuje budování silné vizuální identity a komunity, čelí kritikám ohledně ochrany osobních údajů a moderace obsahu, což může představovat určitá omezení pro uživatele i firmy [31].

# 4. Požadavky

Tato kapitola shrnuje hlavní požadavky, které jsou kladeny na navrhovanou aplikaci. Nejprve jsou definovány základní funkce a vlastnosti systému – popisuje se, co by aplikace měla umět a jaké chování se od ní očekává, včetně bezpečnosti, rychlosti a intuitivnosti uživatelského rozhraní. Dále je představena struktura databáze, která vysvětluje, jak budou data organizována, včetně přehledu jednotlivých tabulek a jejich vzájemných vztahů. Následuje část zaměřená na vizuální modelování procesů a interakcí v systému. Diagram aktivit ukazuje tok činností a dynamiku jednotlivých operací, zatímco diagram případů užití ilustruje, jakým způsobem uživatelé a další aktéři s aplikací interagují a jaké konkrétní úkoly může systém vykonávat. Celkově tato kapitola poskytuje ucelený pohled na to, co systém má dělat, jaké má vlastnosti a jak je navržena jeho vnitřní struktura.

## 4.1 Funkční požadavky

Tabulka 3: Funkční požadavky webové aplikace (zdroj vlastní)

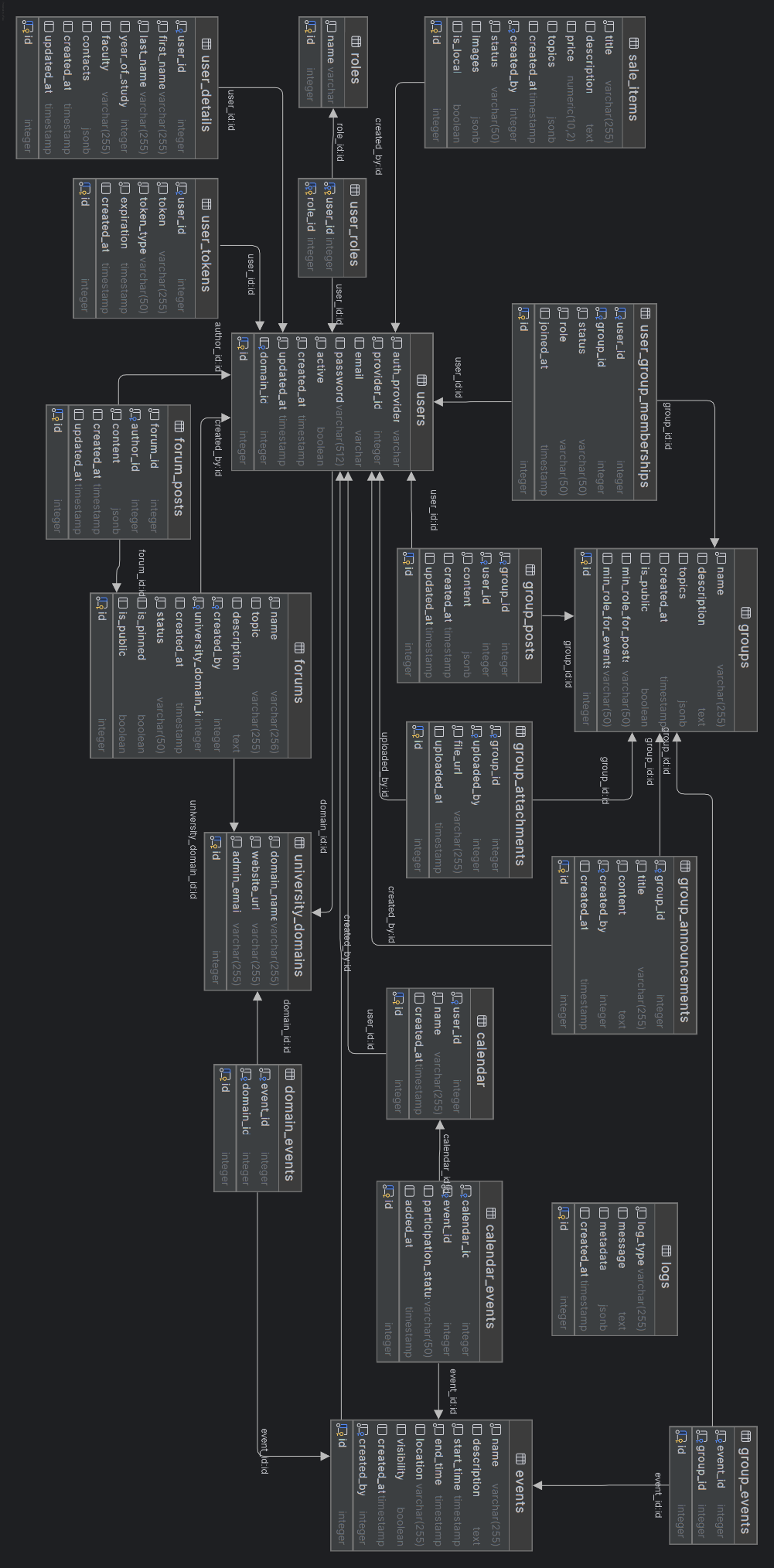
|  |  |  |
| --- | --- | --- |
| F1 | Registrace uživatelů | Aplikace musí umožňovat registraci uživatelů pomocí univerzitního e-mailu či ověřené identity (např. LDAP). |
| F2 | Přihlášení | Aplikace musí zprostředkovat bezpečné přihlášení (login/heslo, případně 2FA) do systému. |
| F3 | Správa profilů | Aplikace musí umožnit úpravu uživatelského profilu (jméno, foto, studijní skupina) a zobrazení profilu. |
| F4 | Fórum a diskuse | Aplikace musí umožňovat vytváření diskusních témat, odpovědí a zobrazování vláken ve fóru. |
| F5 | Správa skupin | Aplikace musí umožňovat zakládat a řídit skupiny (přidávat a odebírat členy, řídit obsah). |
| F6 | Kalendář akcí | Aplikace musí poskytovat modul pro plánování a sledování událostí (přednášky, setkání, akce) s možností registrace. |
| F7 | Kalendář akcí | Aplikace musí **podporovat načítání rozvrhu studenta** (např. z univerzitního systému) pro zobrazení v kalendáři. |
| F8 | Bazar | Aplikace musí umožňovat vložení a správu inzerátů (prodej, výměna, hledání), a to výhradně přihlášenými uživateli. |
| F9 | Systém notifikací | Aplikace musí umět zasílat notifikace (např. o nových příspěvcích, odpovědích, akcích) buď e-mailem, nebo jinou formou (push, interní upozornění), dle preferencí uživatele. |

## 4.2 Nefunkční požadavky

Tabulka 4: Nefunkční požadavky webové aplikace (zdroj vlastní

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | N1 | | Bezpečnost | Aplikace musí implementovat šifrovanou komunikaci (HTTPS), chránit se proti útokům (XSS, CSRF, SQL injection) a umožnit řízení přístupových práv (role). |
| N2 | Dostupnost | |  | | --- | | Aplikace by měla být dostupná 24/7 s definovanou maximální dobou výpadku. Je nutné zavést pravidelné zálohy databáze a případné hot-standby řešení. | |
| N3 | Použitelnost (UX/UI) | Rozhraní musí být responzivní, přehledné a konzistentní. Orientace v aplikaci má být intuitivní pro studenty i správce, s ohledem na zásady UX designu. |
| N4 | Provozní prostředí | Aplikaci musí být možné spustit a nasadit na běžných serverech s OS Linux či Windows (případně kontejnerizovaně), s minimálními softwarovými závislostmi. |
| N5 | Lokalizace a jazyk | Aplikace by měla umožnit vícejazyčné rozhraní (primárně CZ, případně EN) podle nastavení uživatele. |
| N6 | API rozhraní | Aplikace by měla nabízet REST API pro mobilní klienty či jiné systémy (autentizovaný přístup k datům). |
| N7 | Rozšiřitelnost a modularita | Architektura aplikace (např. vícevrstvá MVC) má být navržena tak, aby usnadňovala přidávání nových modulů (např. integrace messengeru, rozhraní pro mobilní aplikace). |

## 4.3 Databáze



## 4.4 Diagramy aktivit

V této části jsou představeny hlavní aktivity uživatele při práci s navrženou komunitní platformou, a to včetně autorizace, správy fóra, práce s bazarem a fungování skupin. Jednotlivé diagramy aktivit poskytují přehled klíčových kroků a rozhodovacích bodů, které uživatel nebo systém provádí v rámci konkrétních funkcí.

### 4.4.1 Hlavní diagram

Изображение выглядит как диаграмма, текст, План, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

Obrázek 2: Hlavní diagram aktivit [vlastní zdroj]

Hlavní diagram aktivit znázorňuje obecný postup uživatele po spuštění aplikace. Po případné autorizaci (pokud je vyžadována) může uživatel přistupovat k jednotlivým funkcím, jako je fórum, bazar, skupiny, osobní údaje či správa obsahu. Rozhodovací body umožňují rozlišit, zda je uživatel autorizován, a podle toho mu nabídnout příslušné možnosti (například editaci profilu či vytváření nových příspěvků). Diagram tak poskytuje komplexní pohled na to, jak se uživatel v aplikaci pohybuje a jaké funkce jsou pro něj dostupné v závislosti na úrovni přístupu.

### 4.4.2 Diagram autorizace

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Obrázek 3: Diagram autorizace [vlastní zdroj]

Tento diagram se zaměřuje na proces vytváření uživatelského účtu a následné přihlášení. Nejprve uživatel rozhoduje, zda si chce založit nový účet, nebo se přihlásí pomocí stávajících přihlašovacích údajů či externí služby (např. OAuth). Po úspěšné registraci či autentizaci může pokračovat v plném využívání aplikace. Diagram zobrazuje rozhodovací uzly, kdy se ověřuje správnost zadaných údajů, a v případě neúspěchu umožňuje vrátit se k zadávání přihlašovacích či registračních dat.

### 4.4.3 Diagram fóra

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Obrázek 4: Diagram fóra [vlastní zdroj]

V diagramu aktivit pro fórum jsou popsány možnosti, které má uživatel k dispozici v rámci diskusních vláken. Může si prohlédnout již existující fórum, vytvářet nová fóra nebo uzavírat stávající (typicky pouze s příslušnými oprávněními). Dále je zde popsána možnost přispívat do fóra, což zahrnuje vkládání příspěvků či komentářů k daným tématům. Výsledkem je ukončení práce ve fóru, ať už formou odhlášení se z diskuse, nebo přechodem k jiné funkci aplikace.

### 4.4.4 Diagram bazaru

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Obrázek 5: Diagram bazaru [vlastní zdroj]

Aktivity související s bazarem se zaměřují na práci s inzeráty. Uživatel s příslušnými oprávněními může inzeráty vytvářet nebo je naopak odstraňovat. Všichni uživatelé pak mají možnost prohlížet si aktuální inzeráty a kontaktovat prodávajícího, pokud je zaujme nabízené zboží či služba. Stejně jako u fóra je v závěru aktivity ukončení práce v bazaru, po němž se uživatel může vrátit k jiným sekcím aplikace nebo aplikaci opustit.

### 4.4.5 Diagram skupin

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Obrázek 6: Diagram skupin [vlastní zdroj]

Diagram aktivit pro skupiny zobrazuje proces správy a účasti ve skupinách. Uživatel může skupinu prohlédnout, vstoupit do ní či z ní odejít, vytvářet nové skupiny nebo v případě vyšších oprávnění skupinu spravovat (například přidávat další účastníky, plánovat události či měnit nastavení skupiny). Tato funkce je užitečná pro organizaci a komunikaci v rámci menších i větších komunit. Po ukončení práce ve skupinách se uživatel může přesunout do dalších sekcí aplikace nebo zcela ukončit svou činnost.

- UCD

- dotazník

### 1.1.3 Obhajoba volby modelu Web 2.0

Pro vzdělávací instituce je optimálním řešením využití modelu Web 2.0, neboť poskytuje potřebnou rovnováhu mezi interaktivitou, bezpečností a centralizovaným řízením. V rámci Web 2.0 jsou implementovány ověřené mechanismy autentizace a autorizace, které spolehlivě chrání osobní data, což je zásadní pro vzdělávací platformy. Centralizovaný systém umožňuje efektivní moderaci a rychlé aktualizace obsahu, čímž se snižuje riziko šíření nesprávných informací. Navíc ekonomická a technická opodstatněnost využívaných technologií snižuje náklady na vývoj a údržbu aplikace, což kontrastuje s experimentálními řešeními Web 3.0, jež vyžadují značné zdroje pro zajištění decentralizace. Tento přístup tak odpovídá požadavkům na stabilitu, bezpečnost a jednoduchou správu, které jsou nezbytné pro úspěšnou implementaci vzdělávací platformy [2].

Jak vlastně vypadají ony balónky? Ptají se často lidé. Inu, jak by vypadaly – jako běžné pouťové balónky střední velikosti, tak akorát nafouknuté. Červený se vedle modrého a zeleného zdá trochu menší, ale to je nejspíš jen optický klam, a i kdyby byl skutečně o něco málo menší, tak vážně jen o trošičku. Vítr skoro nefouká, a tak by se na první pohled mohlo zdát, že se balónky snad vůbec nepohybují. Jenom tak klidně levitují ve vzduchu. Jelikož slunce jasně září a na obloze byste od východu k západu hledali mráček marně, balónky působí jako jakási fata morgána uprostřed pouště. Zkrátka široko daleko nikde nic, jen zelenkavá tráva, jasně modrá obloha a tři křiklavě barevné pouťové balónky, které se téměř nepozorovatelně pohupují ani ne moc vysoko, ani moc nízko nad zemí. Kdyby pod balónky nebyla sytě zelenkavá tráva, ale třeba suchá silnice či beton, možná by bylo vidět jejich barevné stíny - to jak přes poloprůsvitné barevné balónky prochází ostré sluneční paprsky. Jenže kvůli všudy přítomné trávě jsou stíny balónků sotva vidět, natož aby šlo rozeznat, jakou barvu tyto stíny mají.



Obrázek 1: Balonky (Min, 2016)

Uvidět tak balónky náhodný kolemjdoucí, jistě by si pomyslel, že už tu takhle poletují snad tisíc let. Stále si víceméně drží výšku a ani do stran se příliš nepohybují. Proti slunci to vypadá, že se slunce pohybuje k západu rychleji než balónky, a možná to tak skutečně je. Nejeden filozof by mohl tvrdit, že balónky se sluncem závodí, ale fyzikové by to jistě vyvrátili. Z fyzikálního pohledu totiž balónky působí zcela nezajímavě. Nejvíc bezpochyby zaujmou děti – jedna malá holčička zrovna včera div nebrečela, že by snad balónky mohly prasknout. A co teprve ta stuha. Stuha, kterou je každý z trojice balónků uvázán, aby se nevypustil. Očividně je uvázaná dostatečně pevně, protože balónky skutečně neucházejí. To ale není nic zvláštního. Překvapit by však mohl fakt, že nikdo, snad krom toho, kdo balónky k obloze vypustil, netuší, jakou má ona stuha barvu. Je totiž tak lesklá, že za světla se v ní odráží nebe a za tmy zase není vidět vůbec.

Tabulka 1: Příklad tabulky

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Zdroj: (Min, 2016)

Když svítí slunce tak silně jako nyní, tak se stuha třpytí jako kapka rosy a jen málokdo vydrží dívat se na ni přímo déle než pár chvil. Jak vlastně vypadají ony balónky? Ptají se často lidé. Inu, jak by vypadaly – jako běžné pouťové balónky střední velikosti, tak akorát nafouknuté. Červený se vedle modrého a zeleného zdá trochu menší, ale to je nejspíš jen optický klam, a i kdyby byl skutečně o něco málo menší, tak vážně jen o trošičku. Vítr skoro nefouká, a tak by se na první pohled mohlo zdát, že se balónky snad vůbec nepohybují. Jenom tak klidně levitují ve vzduchu. Jelikož slunce jasně září a na obloze byste od východu k západu hledali mráček marně, balónky působí jako jakási fata morgána uprostřed pouště. Zkrátka široko daleko nikde nic, jen zelenkavá tráva, jasně modrá obloha a tři křiklavě barevné pouťové balónky, které se téměř nepozorovatelně pohupují ani ne moc vysoko, ani moc nízko nad zemí. Kdyby pod balónky nebyla sytě zelenkavá tráva, ale třeba suchá silnice či beton, možná by bylo vidět jejich barevné stíny - to jak přes poloprůsvitné barevné balónky prochází ostré sluneční paprsky.

# ZÁVĚR

Překvapit by však mohl fakt, že nikdo, snad krom toho, kdo balónky k obloze vypustil, netuší, jakou má ona stuha barvu. Je totiž tak lesklá, že za světla se v ní odráží nebe a za tmy zase není vidět vůbec. Když svítí slunce tak silně jako nyní, tak se stuha třpytí jako kapka rosy a jen málokdo vydrží dívat se na ni přímo déle než pár chvil. Jak vlastně vypadají ony balónky? Ptají se často lidé. Inu, jak by vypadaly – jako běžné pouťové balónky střední velikosti, tak akorát nafouknuté. Červený se vedle modrého a zeleného zdá trochu menší, ale to je nejspíš jen optický klam, a i kdyby byl skutečně o něco málo menší, tak vážně jen o trošičku. Vítr skoro nefouká, a tak by se na první pohled mohlo zdát, že se balónky snad vůbec nepohybují. Jenom tak klidně levitují ve vzduchu. Jelikož slunce jasně září a na obloze byste od východu k západu hledali mráček marně, balónky působí jako jakási fata morgána uprostřed pouště. Zkrátka široko daleko nikde nic, jen zelenkavá tráva, jasně modrá obloha a tři křiklavě barevné pouťové balónky, které se téměř nepozorovatelně pohupují ani ne moc vysoko, ani moc nízko nad zemí. Kdyby pod balónky nebyla sytě zelenkavá tráva, ale třeba suchá silnice či beton, možná by bylo vidět jejich barevné stíny – to, jak přes poloprůsvitné barevné balónky prochází ostré sluneční paprsky. Jenže kvůli všudy přítomné trávě jsou stíny balónků sotva vidět, natož aby šlo rozeznat, jakou barvu tyto stíny mají.

Uvidět tak balónky náhodný kolemjdoucí, jistě by si pomyslel, že už tu takhle poletují snad tisíc let. Stále si víceméně drží výšku a ani do stran se příliš nepohybují. Proti slunci to vypadá, že se slunce pohybuje k západu rychleji než balónky, a možná to tak skutečně je. Nejeden filozof by mohl tvrdit, že balónky se sluncem závodí, ale fyzikové by to jistě vyvrátili. Z fyzikálního pohledu totiž balónky působí zcela nezajímavě. Nejvíc bezpochyby zaujmou děti – jedna malá holčička zrovna včera div nebrečela, že by snad balónky mohly prasknout.

A co teprve ta stuha. Stuha, kterou je každý z trojice balónků uvázán, aby se nevypustil. Očividně je uvázaná dostatečně pevně, protože balónky skutečně neucházejí. To ale není nic zvláštního. Překvapit by však mohl fakt, že nikdo, snad krom toho, kdo balónky k obloze vypustil, netuší, jakou má ona stuha barvu. Je totiž tak lesklá, že za světla se v ní odráží nebe a za tmy zase není vidět vůbec. Když svítí slunce tak silně jako nyní, tak se stuha třpytí jako kapka rosy a jen málokdo vydrží dívat se na ni přímo déle než pár chvil.

# POUŽITÁ LITERATURA

[1] ACKERMANN, Philip. *Full Stack Web Development: The Comprehensive Guide.* Rheinwerk Computing, 2023.

[2] FISHER, Derek. *Application Security Program Handbook: A guide for software engineers and team leaders.* Manning, 2022.

[3] JACKSI, Karwan and Shakir M. Abass. “Development History Of The World Wide Web.” *International Journal of Scientific & Technology Research*, 2019. <https://www.semanticscholar.org/paper/Development-History-Of-The-World-Wide-Web-Jacksi-Abass/a6b82f4a6caabd0010343793c834e9d5de6cc09c>

[4] CLASSIFYING MODEL-VIEW-CONTROLLER SOFTWARE APPLICATIONS USING SELF-ORGANIZING MAPS, Guamán et al., 2021. <https://www.semanticscholar.org/paper/Classifying-Model-View-Controller-Software-Using-Guam%C3%A1n-Delgado/cf2875b64995fd753c2be54c3d33ef30f48236e0>

[5] MAMIYEV, Azad. (2021). Comparison of Different Architecture Approaches on Android OS [Master’s thesis, Czech Technical University in Prague, Faculty of Information Technology]. (F8-DP-2021-Mamiyev-Azad-Mamiyev-thesis-text.pdf)

[6] **KURAPATI, Lakshmanarao.** (2024). Different Types of Architectures in Frontend Design and Development. *International Journal for Multidisciplinary Research*, IJFMR240528707, September–October 2024. Dostupné z: <https://www.ijfmr.com/>

[7] **ASIMIYU, Zainab.** (2024, November). Evaluating Android Architectural Patterns: A Deep Dive into MVP, MVVM, and MVI. Dostupné z: <https://www.researchgate.net/publication/385492745_Evaluating_Android_Architectural_Patterns_A_Deep_Dive_into_MVP_MVVM_and_MVI>

[8] **Performance Comparison of Native Android Application on MVP and MVVM**, Bambang Wisnuadhi, Ghifari Munawar, & Ujang Wahyu. (2020).

[9] **BLINOWSKI, G.J., Ojdowska, A., & Przybyłek, A.** (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, 20357–20374. Dostupné z: <https://www.semanticscholar.org/paper/Monolithic-vs.-Microservice-Architecture%3A-A-and-Blinowski-Ojdowska/31a9d5d7286b24d5d2a99af005dca7a814640aec>

[10] Brar, H., Kaur, T. P., & Rajoria, Y. (2021). *The Better Comparison between PHP, Python-web & Node.js*. International Journal of Research in Engineering and Science.

[11] Odeniran, Q. (2023). *Comparative Analysis of Fullstack Development Technologies: Frontend, Backend and Database*. Georgia Southern University.

[12] Godinho, A., Rosado, J., Filipe SA, & Cardoso, F. (2024). *Performance Comparison of RESTful Web APIs using a Test Suite: .NET vs. Java Spring Boot*. Journal of Software & Systems Development.

[13] Dhalla, H. K. (2021). *A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core*. Journal of Physics: Conference Series, 1933, 012041. https://iopscience.iop.org/article/10.1088/1742-6596/1933/1/012041

[14] Odeniran, Qozeem. „Comparative Analysis of Fullstack Development Technologies: Frontend, Backend and Database.“ Georgia Southern University, Fall 2023. Dostupné z: <https://digitalcommons.georgiasouthern.edu/etd/2663>

[15] Li, Nian a Bo Zhang. „The Research on Single Page Application Front-end development Based on Vue.“ Journal of Physics: Conference Series, 1883, 012030, 2021. doi:10.1088/1742-6596/1883/1/012030

[16] CHAMBERLIN, Donald. "50 Years of Queries." Communications of the ACM, vol. 67, no. 8, August 2024, pp. 110–121. doi:10.1145/3649887.

[17] Docker, Inc. "Four Ways Docker Boosts Enterprise Software Development." Dostupné z: <https://www.docker.com/blog/four-ways-docker-boosts-enterprise-software-development/#:~:text=For%20workloads%20targeting%20on,are%20still%20dominant>

[18] BMC Software. "State of Containers." Dostupné z: <https://www.bmc.com/blogs/state-of-containers/#:~:text=The%20primary%20reason%20for%20leveraging,%E2%80%9D>

[19] Zero To Mastery. "Angular vs React vs Vue: Which Framework Should You Choose?" Dostupné z: <https://zerotomastery.io/blog/angular-vs-react-vs-vue/#:~:text=Angular%20vs%20React%20vs%20Vue%3A,Detailed%20Documentation%3A%20Vue%27s>

[20] PostgreSQL. Dostupné z: <https://www.postgresql.org/>

[21] Integrate.io. "PostgreSQL vs MySQL: Which One Is Better for Your Use Case?" Dostupné z: <https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/#:~:text=Here%20are%20some%20of%20the,defining%20characteristics%20of%20PostgreSQL>

[22] Spring Boot. Dostupné z <https://spring.io/projects/spring-boot>

[23] Spring. Dostupné z <https://spring.io/>

[24] IJFMR. (2024). Spring Boot, introduced by Pivotal, enhances productivity in enterprise application development. International Journal for Multidisciplinary Research, IJFMR28930, Dostupné z: <https://www.ijfmr.com/papers/2024/5/28930.pdf>

[25] Vue.js. Dostupné z: <https://vuejs.org/>

[26] Invedus. "Angular vs React vs Vue.js – A Comparative Study." Dostupné z: <https://invedus.com/blog/angular-vs-react-vs-vue-js-a-comparative-study/>

[27] Facebook. Dostupné z: <https://www.facebook.com>

[28] Reddit. Dostupné z: <https://www.reddit.com>

[29] Telegram. Dostupné z: <https://telegram.org>

[30] LinkedIn. Dostupné z: <https://www.linkedin.com>

[31] Instagram. Dostupné z: [https://www.instagram.com](https://www.instagram.com/)

ГОЛ работы на будущее

<https://www.researchgate.net/publication/357842727_Web_Development_and_performance_comparison_of_Web_Development_Technologies_in_Nodejs_and_Python#:~:text=>,

В ожидании полной работы

<https://www.researchgate.net/publication/343421705_JSP_Application_Architectures>

в ожидании полной работы, в описаны JAVA EE JSP

<https://zerotomastery.io/blog/angular-vs-react-vs-vue/#:~:text=Angular%20vs%20React%20vs%20Vue%3A,Detailed%20Documentation%3A%20Vue%27s>

- статья сравнения ангуляр и вью и реакта

Синим – дополнить цитирование

Красным – источник под вопросом

Зеленым – Всё заебись

Желтый – стоит разобраться или хотя бы почитать источник

SEZNAM PŘÍLOH

Příloha A: Název přílohy

**PŘÍLOHA A*:* Název přílohy**

Vložit přílohu, popis přílohy A, zdroj.