Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Vývoj webových aplikací pomocí umělé inteligence

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Autor: Martin Indra
Vedoucí práce: Ing. Richard Antonín Novák, Ph.D.

Praha, 2025

Poděkování

Chtěl bych poděkovat panu Ing. Richardu Antonínu Novákovi, Ph.D. za vstřícné vedení, cenné rady a podporu při zpracování této bakalářské práce. Poděkování patří také mé rodině a blízkým za jejich trpělivost, povzbuzení a podporu během celého studia.

Abstrakt

Cílem práce je prozkoumat oblast systémů založených na umělé inteligenci (AI) použitelných pro vývoj webů a otestování funkčnosti vybraných AI systémů. Práce se zaměřuje na současný trend rozvoje umělé inteligence a dopadu tohoto posunu na oblast vývoje SW a specificky webových aplikací.

Klíčová slova

efektivita, generativní AI, umělá inteligence (AI), vývoj webových stránek, webdesign

Abstract

The aim of this thesis is to explore the field of artificial intelligence (AI) based systems applicable to web development and to test the functionality of selected AI systems. The thesis focuses on the current trend of AI development and the impact of this shift on the field of software development and specifically web applications.

Keywords

artificial intelligence (AI), effectivity, generative AI, web design, web development

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

AI *Artificial Intelligence* - umělá inteligence

AHP *Analytic Hierarchy Process* - analytický hierarchický proces

API *Application Programming Interface* - rozhraní pro komunikaci mezi aplikacemi

CSS *Cascading Style Sheets* - jazyk pro stylování webových stránek

GPT *Generative Pre-trained Transformer* - generativní jazykový model

HTML *HyperText Markup Language* - značkovací jazyk webových stránek

IDE *Integrated Development Environment* - vývojové prostředí

JS *JavaScript* - skriptovací jazyk pro web

LLM *Large Language Model* - velký jazykový model

LWM *Large World Model* - model rozsáhlého virtuálního prostředí

NLP *Natural Language Processing* - zpracování přirozeného jazyka

PDO *PHP Data Objects* - rozhraní pro práci s databázemi v PHP

PHP *Hypertext Preprocessor* - serverový skriptovací jazyk

SQL *Structured Query Language* - jazyk pro databázové dotazy

SRS *Software Requirements Specification* - specifikace požadavků na software

SDLC *Software Development Life Cycle* - životní cyklus vývoje softwaru

UX *User Experience* - uživatelská zkušenost

XSS *Cross-site Scripting* - webový bezpečnostní útok

Úvod

V posledních letech dochází k významné proměně způsobu vývoje webových aplikací. Klasické programování (tzv. self-code) bylo po dlouhou dobu jedinou možností jak vytvářet aplikace. S rostoucím tlakem na rychlost vývoje, dostupnost technologií a s nástupem moderních nástrojů však začínají nabývat na významu alternativní přístupy, jako jsou no-code platformy či využívání umělé inteligence (AI) při vývoji.

K vytvoření funkčního řešení dnes už není nutná hluboká znalost programování. Naopak i jednotlivci bez technického zázemí mohou díky nástrojům jako je no-code platforma Bubble nebo rozhraní využívající jazykové modely (např. ChatGPT) vytvářet aplikace. Tento vývoj ale přináší nové otázky - zejména v oblasti kvality, flexibility, udržitelnosti a bezpečnosti takto vytvořených řešení.

Tato bakalářská práce si klade za cíl systematicky porovnat tři přístupy k vývoji webových aplikací - klasický vývoj (self-code), vývoj při použití AI asistenta a tvorbu aplikace pomocí no-code platformy. Všemi přístupy bude na základě stejných požadavků vytvořena webová aplikace. Výsledkem je nejen vytvoření tří funkčních verzí stejné aplikace, ale také jejich zhodnocení s využitím vícekriteriální metody AHP (Analytic Hierarchy Process).

Cíle práce

Hlavním cílem práce je porovnat tři různé přístupy k vývoji webové aplikace - klasické programování, vývoj za pomoci umělé inteligence a no-code řešení - z hlediska jejich kvality, náročnosti na vývoj, flexibility a závislosti na externích službách.

Tento cíl je rozpracován do následujících dílčích úkolů:

- Vytvoření webové aplikace pomocí klasického programování v jazycích PHP, JavaScript, HTML a CSS.
- Vytvoření téže aplikace s využitím umělé inteligence, konkrétně jazykového modelu ChatGPT (o3-mini-high).
- Implementace aplikace pomocí no-code platformy Bubble.
- Dokumentace celého vývojového procesu, včetně časové náročnosti, problémů, způsobu testování a kvality kódu.
- Zhodnocení jednotlivých přístupů na základě definovaných kritérií pomocí metody AHP.

Výsledkem práce je přehledné srovnání výhod a nevýhod každého přístupu s ohledem na praktické aspekty vývoje webových aplikací.

Metodika

V práci byly využity následující metody:

- Rešerše, komparace, implementace.
- Analýza, syntéza, abstrakce, konkretizace.
- Při vývoji byl využit Personal Scrum.
- Výsledky byly porovnány pomocí AHP.

1. Teoretická část

1.1 Softwarové inženýrství

1.1.1 Vymezení pojmu softwarové inženýrství

Softwarové inženýrství byla historicky disciplína zaměřená na vývoj softwaru s dodržováním zásad a osvědčených metodik (Boehm, 1976). Dalším zaměřením je uživatel, který bude výsledný produkt používat. Neméně důležité je testování a podpora vytvořeného softwaru (Boehm, 1976).

Existuje mnoho definic a autor zvolil jako nejvhodnější tuto:

> *"Praktická aplikace vědeckých poznatků při návrhu a konstrukci počítačových programů a související dokumentace potřebných k jejich vývoji, provozu a údržbě (Boehm, 1976)."*

Mezi klíčové vlastnosti dobrého softwaru patří (Bennett, 2024):

- Znovupoužitelnost - jak jednoduché je software použít na něco jiného
- Flexibilita - náročnost na provedení změn v programu
- Srozumitelnost - složitost čtení kódu
- Funkčnost - kód musí dělat, co bylo zadáno
- Efektivita - nedochází ke zbytečnému využití prostředků

1.1.2 Životní cyklus vývoje softwaru (SDLC)

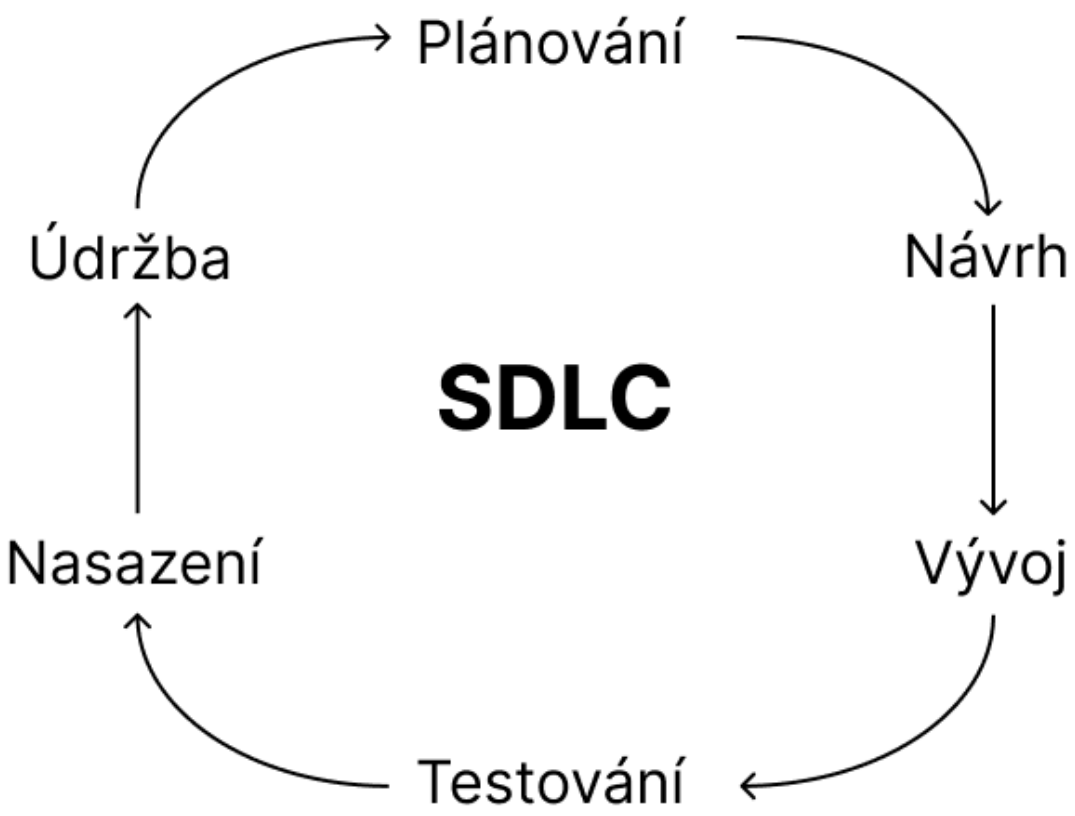
Životní cyklus vývoje softwaru (Software Development Life Cycle, SDLC) je proces, který definuje postupy při vývoji software (Jackson, 2024). Jeho hlavním cílem je zajistit, jak vytvořit produkt tak, aby splňoval požadavky uživatelů a byl vysoké kvality. SDLC poskytuje rámec pro řízení a kontrolu celého procesu vývoje softwaru (Geeksforgeeks, 2024a).

SDLC obvykle zahrnuje následující fáze (Jackson, 2024):

1. **Plánování a analýza**: První fáze stanovuje cíle a rozsah projektu vývoje softwaru. Provádí se analýza požadavků, provádí se průzkum trhu a testování proveditelnosti, hodnotí prototypy. Výstupem této fáze často bývá dokument

specifikace požadavků na software (Software Requirement Specification, SRS), který obsahuje funkce softwaru a časový plán projektu. Díky němu je možné pracovat efektivně a neodchylovat se od zadání (Geeksforgeeks, 2024a; Jackson, 2024).

2. **Návrh**: Před samotným psaním kódu musí vývojáři navrhnout architekturu projektu. Tato fáze zahrnuje navrhování uživatelských rozhraní a návrh databáze. UX designéři pak vytváří prototyp vzhledu aplikace, který slouží pro vizualizaci a konzultaci se zadavatelem (Geeksforgeeks, 2024a; Jackson, 2024).
3. **Vývoj**: Ve třetí fázi vývojáři píší kód podle specifikací a návrhu vytvořeného v předchozích fázích (Geeksforgeeks, 2024a; Jackson, 2024).
4. **Testování**: Po dokončení vývoje je software testován jak na splnění funkčních požadavků, tak na stabilitu a chyby. Testování zahrnuje různé typy testů, jako je jednotkové testování, integrační testování, systémové testování a akceptační testování zadavatelem (Geeksforgeeks, 2024a; Jackson, 2024).
5. **Nasazení**: Pokud je otestovaný produkt přijat, tak dochází k nasazení softwaru do produkčního prostředí. Zahrnuje to instalaci softwaru, migraci dat a školení uživatelů (Geeksforgeeks, 2024a; Jackson, 2024).
6. **Údržba**: V poslední fázi dochází k monitorování a kontrole software, tak aby se bylo zajištěno jeho fungování. Údržba může zahrnovat opravy chyb, aktualizace funkcí a optimalizace výkonu (Geeksforgeeks, 2024a; Jackson, 2024).



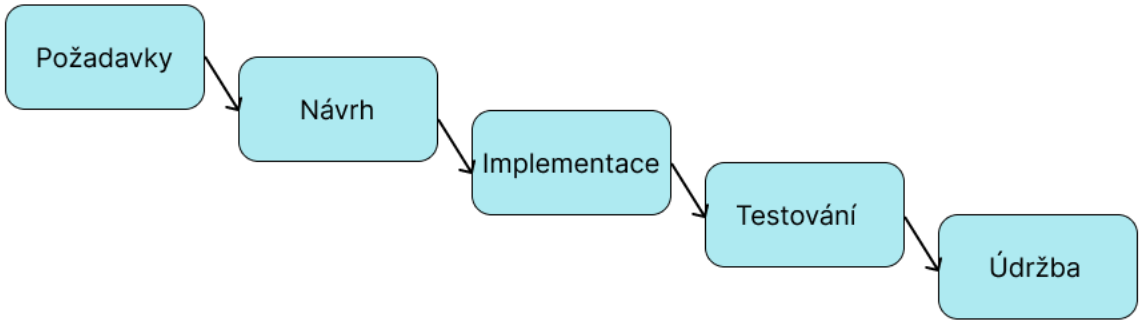
Obrázek 1.1.1: Životní cyklus vývoje softwaru

Zdroj: Autor

Modely SDLC

Existuje několik modelů SDLC, které organizace používají na základě svých potřeb a požadavků projektu. Mezi nejběžnější patří:

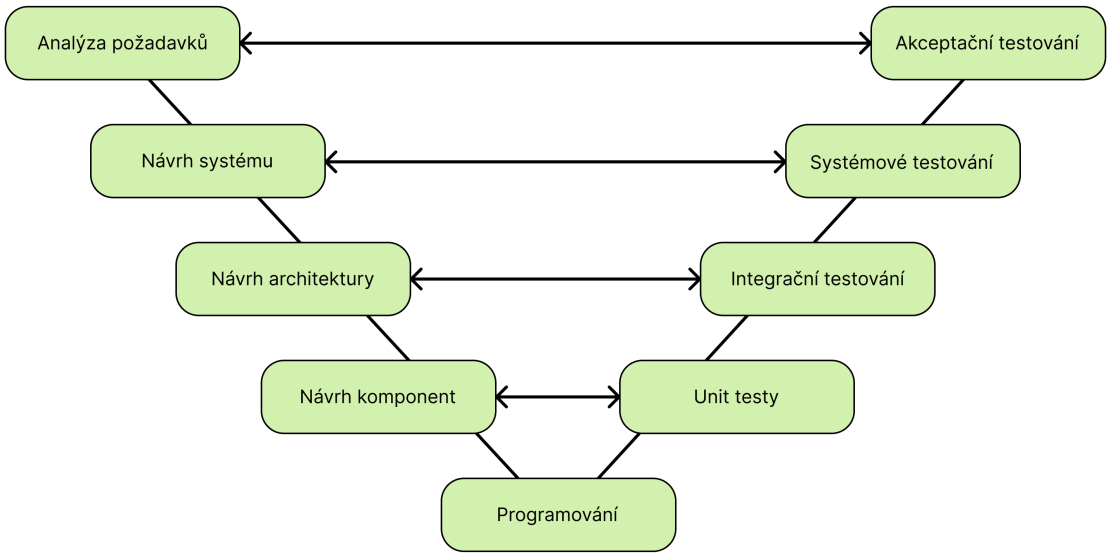
- **Vodopádový model**: vývoj probíhá v přesně definovaných fázích (analýza, návrh, implementace, testování, nasazení, údržba). Vše se provádí postupně, což ztěžuje pozdější úpravy softwaru, a proto je tento model vhodný, pokud jsou požadavky detailně definovány již od počátku. V současné době se již nevyužívá, avšak vychází z něj většina moderních SDLC modelů (Geeksforgeeks, 2024b).



Obrázek 1.1.2: Vodopádový model

Zdroj: Autor

- **V-model**: známý také jako ověřovací a validační model, je rozšířením vodopádového modelu s důrazem na testování v každé fázi vývoje. Struktura modelu připomíná tvar písmene "V", kde levá strana představuje fáze vývoje a pravá strana odpovídající fáze testování (Geeksforgeeks, 2024b).



Obrázek 1.1.3: V-Model

Zdroj: Autor

- **Agilní model**: jedná se o přístup, kdy se produkt vyvíjí v iteracích. Tyto cykly bývají krátké, což umožňuje flexibilní úpravy softwaru dle aktuálních požadavků zákazníka (Jackson, 2024). Má mnoho forem:
  - **Scrum**: metodika, která rozděluje vývoj do tzv. sprintů, během nichž se produkt postupně naplánuje, naimplementuje, otestuje a zhodnotí (Bechný, 2010). Tento cyklus se opakuje, dokud není program hotov.

  - **Kanban**: na rozdíl od scrumu zde nejsou pevně stanovené intervaly. Veškerá práce se vizualizuje na tabuli, kde je jasně vidět, co se momentálně dělá a co je potřeba udělat.
  - **Personal Scrum**: upravuje metodiku scrum pro individuální potřeby např. pro osobní produktivitu a řízení malých projektů. Tuto metodologii autor využije ve své vlastní implementaci aplikace(Zin et al., 2024).
- **Iterativní a inkrementální model**: Model je založen na postupném vylepšování software, který začíná jako malý projekt, na kterém se postupně v jednotlivých iteracích staví dál. Software je dokončen až je jak dodavatel, tak zadavatel spokojen s výsledkem (Geeksforgeeks, 2024b; Jackson, 2024).
- **Spirálový model**: Je kombinací vodopádového i agilního modelu. Je charakteristický opakovanými cykly, kde dochází postupně k plánování, analýze rizik, vývoji a testování. Všechny tyto fáze jsou zopakovány několikrát, a tak dochází k eliminaci rizik a chyb (Jackson, 2024).

Význam SDLC

Implementace SDLC poskytuje strukturovaný přístup k vývoji softwaru, který pomáhá týmům efektivně plánovat, navrhovat, vyvíjet, testovat a udržovat software. To vede k vyšší kvalitě softwaru, lepší spokojenosti zákazníků a efektivnějšímu využití zdrojů. SDLC také pomáhá identifikovat a minimalizovat rizika v raných fázích vývoje, což může vést k úsporám nákladů a času (Geeksforgeeks, 2024a; Jackson, 2024).

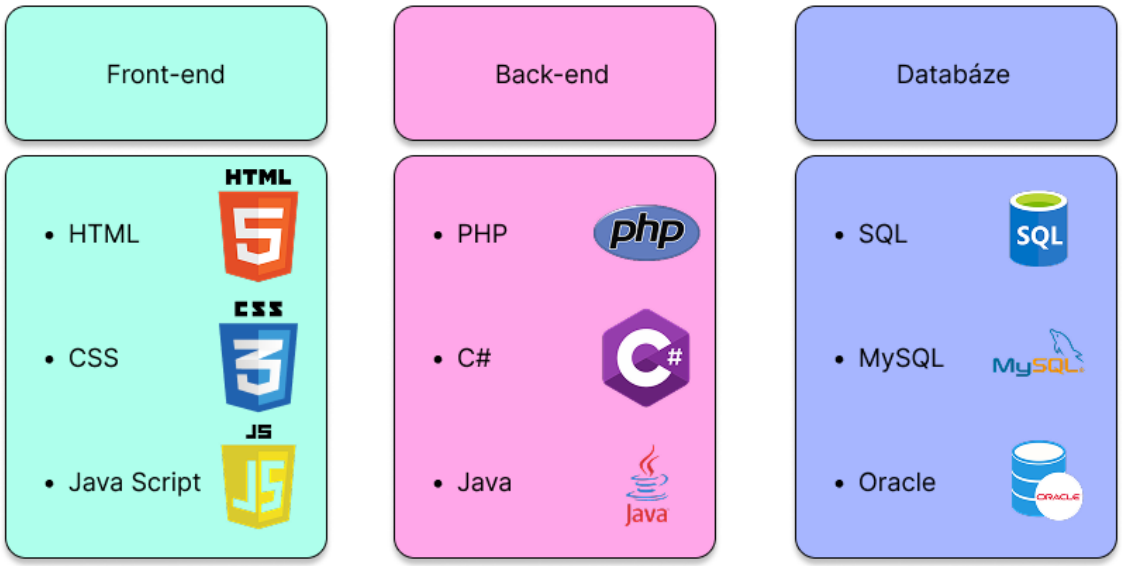
1.1.3 Architektura aplikace

Většina moderních webových aplikací je založena na vícevrstvém přístupu k architektuře (Salas-Zárate et al., 2015). Základem tohoto způsobu je rozdělení aplikace do tzv. vrstev, které jsou od sebe oddělené. Vrstvy mají své specifické funkce, což činí aplikaci modulárnější, snadněji škálovatelnou a udržitelnou.

Nejčastěji vrstvy rozdělujeme na:

- **Frontend**: Slouží jako komunikační vrstva s uživatelem. U webových aplikací se používají technologie jako HTML, CSS a JavaScript, které umožňují vytvořit přehledné, dynamické a intuitivní uživatelské rozhraní. Existuje řada frameworků např. React nebo Vue.js, které tvorbu rozhraní dále usnadňují (Bapna et al., 2024).

- **Backend**: Primárním úkolem backendu je zpracování požadavků od uživatele například vyplnění formuláře, odeslání objednávky či přihlášení (Welling, 2017). Nejčastěji používanou backendovou technologií je PHP.
- **Databáze**: Tato vrstva slouží k bezpečnému uchování dat na dedikovaném serveru (Silberschatz et al., 2011). Obvykle se využívají relační databáze, jako jsou MySQL nebo Postgre a případně i NoSQL řešení (např. MongoDB).



Obrázek 1.1.4: Architektura webových aplikací

Zdroj: Autor

Kromě vícevrstvé architektury se při návrhu webových systémů často využívají i obecnější architektonické vzory, které dále strukturují aplikaci:

- **Model-View-Controller (MVC)**: Rozděluje aplikaci na model (data a logika), view (uživatelské rozhraní) a controller (zpracování vstupů), čímž se dále aplikace funkčně odděluje. Je tak možné snadněji provádět změny v konkretních částech aplikaci bez ovlivňování jiných (Necula, 2024).
- **Mikroslužby**: Aplikace je rozdělena na malé, samostatné služby zaměřené na konkrétní části systému. Služby běží nezávisle a mezi sebou komunikují přes jednoduchá rozhraní (např. HTTP/REST). Díky tomuto přístupu je možné snadno využít různé technologie pro části systému (Kozon, 2024).
- **REST**: Odděluje klienta a server, což umožňuje jejich nezávislý vývoj. Komunikace probíhá přes standardní HTTP operace (např. GET, POST) a je jednoduchá, přehledná a dobře škálovatelná (Isles, 2023).

1.1.4 Technologie pro vývoj webových aplikací

Autor pro realizaci navrhované webové aplikace použil stack PHP a MySQL, a to jak z důvodu své odborné znalosti, tak z důvodu jejich širokého rozšíření ve světě vývoje webových aplikací.

HTML a CSS

HTML (HyperText Markup Language) je značkovací jazyk sloužící k definování struktury webových stránek a aplikací. Funguje na principu takzvaných značek, kdy každá část stránky musí být z obou stran ohraničena tímto způsobem od nadpisů a odstavců až po formuláře a odkazy. Díky tomu prohlížeč přesně ví, jak co a kde má zobrazovat. Také to vyhledávačům umožňuje stránky indexovat, a tím nám je pomoci najít (Contributors, 2023b). HTML začalo vznikat na počátku 90. let, kdy jej standardizovala organizace W3C. Tento fakt změnil vývoj celého internetu (W3C, 2023).

CSS (Cascading Style Sheets) slouží k úpravě vizuální části obsahu definovaného v HTML. CSS nám umožňuje definovat barvy, rozvržení, písmo či animace. Tyto takzvané styly mohou být aplikovány jak na jednotlivé specifické značky s vlastní id, tak na stránku jako celek. Díky oddělení vizuálu od HTML je možné jednoduše měnit vzhled stránky bez zasahování do samotné struktury (Contributors, 2023a). V dnešní době existuje spoustu trendů a moderních přístupů k vytvoření webu jako například responzivní design či Mobile-first.

Kombinace HTML a CSS je základem moderního webdesignu. V praxi se často využívají preprocesory (např. SASS či LESS) a frameworky (např. Bootstrap), díky kterým stránky vypadají moderně a odpadá nutnost vše tvořit od nuly (Contributors, 2023a, 2023b; W3C, 2023).

PHP

PHP (Hypertext Preprocessor) je skriptovací jazyk, který se primárně využívá na straně serveru. Má schopnost dynamicky generovat HTML na základě vstupů od uživatele a ten je následně odeslán do prohlížeče (Group, 2023).

PHP začalo vznikat v polovině 90. let, kdy původně existovalo jako nástroj pro sledování návštěvnosti osobních stránek. V průběhu let se z něj stal plnohodnotný programovací jazyk s podporou objektově orientovaného programování, širokou paletou vestavěných funkcí a rozsáhlou komunitou. Integrace s databázemi, zejména s MySQL, umožňuje efektivní správu a načítání dat, což je klíčové pro mnoho webových aplikací (Kosek, 1999).

Díky neustálému vývoji a pravidelným aktualizacím se PHP přizpůsobuje moderním požadavkům a zůstává konkurenceschopným nástrojem ve světě webového vývoje (Group, 2023).

MySQL

MySQL je open source relační databázový systém, který využívá Structured Query Language (SQL). Jeho přednosti jsou spolehlivost, výkon a škálovatelnost, díky čemuž je druhým nejpopulárnějším databázovým systémem. MySQL se používá jak pro malé osobní projety, tak pro obrovské podnikové systémy (Erickson, 2025).

Je důležité rozlišit od sebe MySQL a SQL. SQL je jen jazyk, který program či programátor využívá pro komunikaci s databází, kterou je v tomto případě MySQL (Erickson, 2025). SQL tak může být využito ke komunikaci se spoustou dalších databázových systémů.

Další technologie pro webový vývoj

Java

Java je objektově orientovaný programovací jazyk. Díky silnému typování, což znamená, že jednotlivým proměnným musí být přiřazen jeden typ, je Java robustním jazykem. Zároveň je velmi optimalizovaná a škálovatelná, čímž se stává vhodnou pro využití ve velkých enterprise aplikacích. Pro vývoj webových aplikací také v Javě existuje bohatý ekosystém frameworků a nástrojů jako je Spring. Na druhou stranu může být Java pro začátečníky složitější kvůli své komplexní syntaxi (Tran, 2024).

Golang (Go)

Golang, vyvinutý společností Google, je kompilovaný programovací jazyk známý svou jednoduchostí a vysokým výkonem. Go se dostává do povědomí v posledních letech a nabírá na stále větší popularitě. Díky své jednoduché syntaxi a vysokému výkonu je ideální pro vývoj webových serverů, mikroservisních architektur a síťových aplikací. Mezi klíčové vlastnosti Go patří vestavěná podpora paralelismu prostřednictvím gorutin, rychlá kompilace a rozsáhlá standardní knihovna (Jebavý, 2024; Tran, 2024).

Python

Python je univerzální programovací jazyk, který se využívá jak pro vývoj webových aplikací, tak i pro spoustu dalších věcí jako je machine learning a statistika. Díky své čitelnosti a jednoduché syntaxi umožňuje jednoduše začít s vývojem webů. Mezi oblíbené frameworky pro webový vývoj v Pythonu patří Django a Flask. Django je robustní framework nabízející mnoho vestavěných funkcí, jako je správa databází a autentizace uživatelů. Flask je naopak lehký mikroframework, který poskytuje větší

flexibilitu a je vhodný pro menší projekty nebo pro vývojáře preferující modulární přístup. Výhodou Pythonu je také jeho aktivní komunita, která poskytuje množství knihoven a nástrojů podporujících webový vývoj (Patkar et al., 2022).

C++

C++ je vysoce výkonný kompilovaný programovací jazyk, který se tradičně využívá pro vývoj náročných aplikací. Přestože není primárně navržen pro webový vývoj, existují nástroje a frameworky, které umožňují tvorbu webových aplikací v C++. Jedním z nich je C++ Web Framework (CWF), který kombinuje výkon jazyka C++ s flexibilitou Qt frameworku. CWF poskytuje vývojářům nástroje pro efektivní vývoj webových aplikací s nízkou spotřebou systémových prostředků, což je ideální pro aplikace vyžadující vysoký výkon a efektivitu (Lima & Eler, 2021).

1.2 Umělá inteligence

1.2.1 Obecná charakteristika umělé inteligence

Umělá inteligence neboli AI je obor informatiky, jež se zaměřuje na vývoj systémů, které se snaží imitovat lidskou inteligenci, a díky tomu zpracovávat nejrůznější úlohy, které dříve nemohly být automatizované. Mezi tyto úlohy patří například rozpoznávání řeči, vizuální vnímání nebo rozhodování (Biswas, 2023). Něco jako AI začalo vznikat už v polovině 20. století, kdy se začaly vytvářet jednoduché algoritmy a teorie, které se postupně rozvíjely a postupně dospěly až ke vzniku strojového učení a neuronových sítí (Biswas, 2023).

1.2.2 Strojové učení

Strojové učení je klíčovou součástí AI, která se zaměřuje na vývoj modelů umožňujících systémům se "učit" a zlepšovat na základě dat. Existují tři hlavní typy strojového učení:

- **Supervised learning**: Model je trénován na označených datech, kde každému vstupu odpovídá známý výstup. To umožňuje modelu naučit se vztah mezi vstupy a výstupy, což je užitečné pro úlohy jako je klasifikace a regresní analýza (Berkeley, 2020).
- **Unsupervised learning**: Model pracuje s neoznačenými daty a snaží se odhalit skryté vzory nebo struktury jako je shlukování nebo asociační pravidla. Tento přístup je užitečný pro objevování skrytých struktur v datech (Berkeley, 2020).
- **Semi-supervised learning**: Kombinace dvou předchozích typů, kde jedna část dat je označená a druhá ne. Postupně si tedy umělá inteligence dokáže i neklasifikovaná data zařadit (Berkeley, 2020).
- **Reinforcement learning**: Model se učí prostřednictvím interakce s prostředím, kdy získává odměny nebo tresty na základě svých akcí. Cílem je naučit se strategii, která maximalizuje kumulativní odměnu. Tento přístup je často používán v robotice a hrách (Berkeley, 2020).

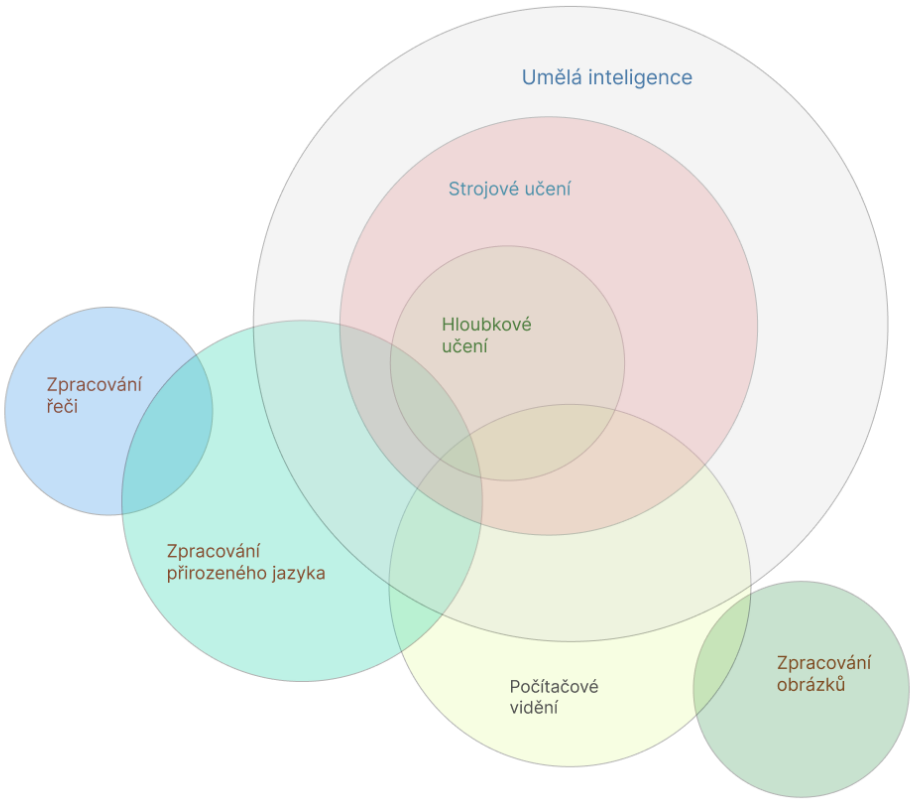
1.2.3 Oblasti umělé inteligence

AI lze dále rozdělit do specifických oblastí zaměřených na různé typy dat a úloh:

- **Natural Language Processing (Zpracování přirozeného jazyka)**: Tato oblast umožňuje počítačům porozumět, interpretovat a generovat lidský jazyk.

NLP je základem pro systémy jako je ChatGPT, které díky velkým jazykovým modelům dokáží interagovat s uživatelem běžnou řečí (Biswas, 2023).

- **Computer Vision (Počítačové vidění)**: Zaměřuje se na umožnění počítačům interpretovat a zpracovávat vizuální informace z reálného světa například z obrázků a videí. Hlavní schopnosti jsou rozpoznávání objektů, sledování pohybu a analýza obrazu (Moin, 2023).
- **Generativní AI**: Hlavním cílem generativní AI je vytváření nového obsahu - textů, obrázků či hudby. Modely pracují s obrovským množstvím dat, která pomocí neuronových sítí zpracovávají, a tím dokáží vytvářet nové a originální výstupy (Khan, 2023).



Obrázek 1.2.1: Rozdělení AI

Zdroj: Autor

1.2.4 Integrace oblastí AI

Kombinace různých oblastí AI jako je NLP a počítačové vidění, vede k vývoji systémů schopných komplexního porozumění a interakce s uživateli. Například integrace těchto technologií umožňuje strojům nejen vidět obraz, ale také rozumět textu v něm

obsaženém, což vede k pokročilým aplikacím v oblasti analýzy obrazu a generování popisů.

1.2.5 Současné trendy

V roce 2025 se očekává vývoj v těchto oblastech:

Generativní virtuální světy: Společnosti jako Google DeepMind představily modely, které dokáží z jediného obrázku vytvořit interaktivní 2D platformovou hru. Startupy jako World Labs pracují na tzv. "large world models" (LWM). Tyto modely mají potenciál nejen v herním průmyslu, ale i při tréninku robotů ve virtuálních prostředích (James O'Donnell, 2025).

Velké jazykové modely (LLM) s "uvažováním": Nové modely jako je OpenAI o3, jsou navrženy tak, aby pracovaly krok za krokem při řešení složitých problémů, čímž napodobují lidský způsob uvažování. Starší modely pouze předaly první věc, co je napadla bez toho, aby se "zamyslely" jestli to vlastně dává smysl a odpovídá to na dotaz (James O'Donnell, 2025).

Generativní AI vyhledávání: Je velice pravděpodobné, že tradiční vyhledávače, tak jak je známe se změní. Nově se očekává, že by vyhledávače na otázky uživatelů rovnou odpovídaly pomocí generativní AI. Tím se zjednoduší a zrychlí proces vyhledávání. Nese to sebou samozřejmě i spoustu rizik, jako například validitu dat (James O'Donnell, 2025).

AI agenti transformující pracovní prostředí: AI agenti představují pokročilou úroveň generativní AI. Agenti jsou navrženi tak, aby měli specifické odborné znalosti, což jim umožňuje řešit složité nebo vícekrokové úkoly, jako je zpracování zákaznických dotazů, správa objednávek nebo optimalizace a pracovních procesů. Tímto by se mělo docílit velké automatizace rutinních procesů, a tím uvolnit lidskou kapacitu na složitější úkony (Ray, 2024).

1.2.6 AI systémy a jejich využití ve vývoji webových aplikací

Při vývoji webových aplikací se AI používá stále více, protože dokáží výrazně zjednodušit a zefektivnit procesy jak vývoje, tak testování. Mezi hlavní funkce patří:

- **Generování kódu**: Moderní generativní AI modely, jako je například ChatGPT, dokáží vymýšlet kód v jakémkoliv programovacím jazyce. Dobře zvládají také kód analyzovat a na základě toho uživateli vysvětlit, co daný úsek dělá. Nejlépe se

dá AI využít na generování metod, které se podobají metodám již vytvořeným uživatelem. Další způsob jak AI využít, je pro generování nápadů, jak daný úkol udělat. AI může navrhnout spoustu myšlenek jak problém vyřešit a programátor si jednu z nich vybere a naprogramuje (Biswas, 2023).

- **Návrh databáze**: Pokud předáme AI dostatek informací o projektu, styl jakým chceme aplikaci implementovat, dokáže nám navrhnout celou strukturu databáze. Na základě toho návrh pak dokáže sestavit SQL příkazy k vytvoření dané databáze. V případě potřeby nám AI může poskytnou i testovací data pro vyzkoušení funkčnosti.
- **Optimalizace výkonu aplikací**: Díky schopnosti umělé inteligence pochopit uživatelem poskytnutý kód, je pro ní jednoduché nalézt nedostatky a navrhnout jak je může programátor opravit. Případně pokud má dostatek podkladů může poskytnou celý nový optimalizovaný kód.
- **Generování komentářů a dokumentace**: AI dokáže díky svým schopnostem vytvořit k poskytnutému kódu smysluplné komentáře v jakémkoliv jazyce, tak aby je dokázal pochopit další člověk, co kód čte.

Mezi konkrétní nástroje, které se v této oblasti využívají, patří:

- **ChatGPT**: ChatGPT je nástroj vyvinutý společností OpenAI, který byl poprvé uveden na trh v listopadu 2022 (OpenAI, 2022). Využívá techniky zpracování přirozeného jazyka (NLP) a modely typu Generative Pre-trained Transformer, z čehož plyne název GPT. Díky tomuto je AI schopna komunikovat s uživatelem lidskou řečí. V dnešní době se ChatGPT nepoužívá jen na psaní kódu, ale také ke spoustě dalších úkonů jako například sumarizace textu či odpovídání na běžné dotazy (OpenAI, 2022).
- **GitHub Copilot**: GitHub Copilot je AI asistent zaměřený přímo na psaní kódu. Jeho nespornou výhodou je přímá integrace do spousty populárních vývojových prostředí, jako je Visual Studio či Visual Studio Code. Má tak rovnou přístup k celému projektu a díky tomu dokáže doplňovat části kódu či dokonce celé třídy či metody (GitHub, 2025).

1.2.7 Nocode/Lowcode

No-code a low-code platformy jsou způsob, jak vyvíjet software s minimálním nebo žádným psaním kódu. Umožňují uživatelům bez programátorských znalostí vytvářet aplikace pomocí vizuálních nástrojů a drag-and-drop rozhraní. Low-code platformy kombinují tento přístup s psaním minimálního množství kódu pro pokročilejší funkce (IBM, 2025; SAP, 2025).

Mezi populární no-code a low-code platformy patří například Bubble, která poskytuje vizuální editor umožňující design uživatelského rozhraní, správu databáze a definici obchodní logiky prostřednictvím workflow. Dalšími významnými platformami jsou lovable.dev, Mendix, OutSystems a Microsoft Power Apps (Bubble, 2025; Mendix, 2025).

1.3 Saatyho metoda (AHP)

1.3.1 Úvod do AHP

Analytický hierarchický proces (AHP) je metoda pro podporu vícekriteriálního rozhodování, kterou v 70. letech 20. století vyvinul matematik Thomas L. Saaty (Fiala et al., 1997). Tato metoda pomáhá činit složitá rozhodnutí tím, že strukturuje problém do hierarchie cílů, kritérií a alternativ, což umožňuje systematické porovnávání a hodnocení různých možností (Fiala et al., 1997).

1.3.2 Hierarchická struktura rozhodování

AHP bývá rozložen do hierarchické struktury, která se obvykle skládá ze tří úrovní (Fiala et al., 1997):

1. **Cíl rozhodování**: Hlavní účel nebo cíl, kterého chceme dosáhnout.
2. **Kritéria a subkritéria**: Faktory které ovlivňují dosažení cíle.
3. **Alternativy**: Možnosti, mezi kterými se rozhoduje (Fiala et al., 1997).

Díky této struktuře je možné se důkladně zaměřit na jednotlivé části problému, a tím zajistit, že se na nic nezapomene.

Párové porovnávání a stanovení vah

Klíčovou součástí AHP je tzv. párové porovnávání. Dvojice prvků dají vedle sebe a určuje se jejich relativní význam pomocí škály, která obsahuje hodnoty od 1 (stejná důležitost) do 9 (extrémní důležitost jednoho prvku nad druhým). Díky těmto porovnáním lze určit, jakou váhu mají jednotlivé kritéria (T. Saaty, 1994).

Syntéza priorit

Po stanovení vah pro jednotlivá kritéria a alternativy následuje syntéza, při níž jsou kombinovány váhy z různých úrovní hierarchie. Toto vede k určení celkových priorit alternativ, což umožňuje jejich seřazení od nejvýhodnější po nejméně výhodnou (T. L. Saaty & Vargas, 2012).

Využití metody AHP v praxi

Díky obrovské flexibilitě a možnosti metody ji lze využít na víceméně jakýkoliv rozhodovací problém. Je hojně využívána manažery a analytiky. Saatyho metoda je široce využívána v oblastech, jako jsou strategické plánování, řízení rizik či hodnocení projektů (Fiala et al., 1997).

1.3.3 Výhody a limity Saatyho metody

Výhody:

- **Strukturovaný přístup**: Nespornou výhodou je jasný způsob, jak rozložit složité rozhodovací problémy na menší, a tím usnadnit jejich analýzu (Fiala et al., 1997).
- **Kvantifikace subjektivních hodnocení**: Metoda umožňuje převést subjektivní názory rozhodovatelů do kvantitativní podoby, díky čemuž je většina subjektivity eliminována (T. Saaty, 1994).
- **Flexibilita**: AHP lze jednoduše přizpůsobit specifickým potřebám různých odvětví (T. L. Saaty & Vargas, 2012).

Limity:

- **Konzistence porovnání**: U většího počtu kritérií může být náročné udržet konzistenci (kritérium A je důležitější než kritérium B a kritérium B je důležitější než kritérium C, logicky by mělo být A důležitější než C), což může snížit spolehlivost výsledků (T. Saaty, 1994).
- **Časová náročnost**: Proces párového porovnávání a výpočtu vah může být u rozsáhlých rozhodovacích problémů časově náročný (T. L. Saaty & Vargas, 2012).

1.3.4 Využití AHP v práci

Pro porovnání jednotlivých způsobů vývoje webové aplikace autor zvolil následující kritéria:

- **Rychlost implementace**: Čas potřebný k vytvoření funkční verze aplikace od začátku do konce.
- **Kvalita výsledného kódu/aplikace**: Zaměřuje se na čistotu kódu, efektivitu, bezpečnost a celkovou kvalitu implementace.
- **Flexibilita a udržovatelnost**: Hodnotí, jak snadno lze aplikaci dodatečně upravit a rozšiřovat.
- **Náročnost osvojení technologie**: Reflektuje, jak složité je naučit se pracovat s daným přístupem či technologií.
- **Závislost na externích nástrojích/službách**: Udává míru rizika, že při změně či výpadku externích služeb přestane aplikace fungovat.

Pro určení vah byla použita metoda AHP s párovým porovnáváním. Níže uvedená tabulka obsahuje matici, kde jsou kritéria vzájemně porovnána dle Saatyho škály (1

- stejná důležitost, 3 - mírná převaha, 5 - výrazná převaha, 7 - velmi silná převaha, 9 - absolutní převaha; hodnoty mezi jsou mezistupně). Hodnoty pod diagonálou jsou převrácené hodnoty z horní části.

Pro každé kritérium se vypočítá geometrický průměr (GM) řádku:

$$\mathrm{GM}_i = \left(\prod_{j=1}^{5} a_{ij} \right)^{\frac{1}{5}},$$

kde a_{ij} je hodnota z matice porovnání a 5 je počet kritérií. Po výpočtu všech geometrických průměrů se provede normalizace, tedy každé kritérium obdrží váhu jako podíl svého GM na součtu všech GM.

Kritéria	**Rychl.**	**Kvalita**	**Flexib.**	**Osvoj.**	**Závisl.**	**GM**
Rychlost implementace	1	2	3	2	3	2.047
Kvalita výsledku	1/2	1	2	3	3	1.569
Flexibilita	1/3	1/2	1	1	3	1.423
Náročnost osvojení	1/2	1/3	1	1	1	1.308
Závislost na ext. službách	1/3	1/3	1/3	1	1	1.246

Tabulka 1.3.1: Párové porovnání kritérií podle AHP

Na základě zvolených hodnot vychází přibližné výsledné váhy:

$$\text{Celkový součet GM} = 2.047 + 1.569 + 1.423 + 1.308 + 1.246 \approx 7.593.$$

$$\begin{aligned}
\text{Rychlost implementace} &= \frac{2.047}{7.593} \approx 0.270 \quad (27.0\%), \\
\text{Kvalita výsledku} &= \frac{1.569}{7.593} \approx 0.207 \quad (20.7\%), \\
\text{Flexibilita a udržovatelnost} &= \frac{1.423}{7.593} \approx 0.187 \quad (18.7\%), \\
\text{Náročnost osvojení} &= \frac{1.308}{7.593} \approx 0.172 \quad (17.2\%), \\
\text{Závislost na externích službách} &= \frac{1.246}{7.593} \approx 0.164 \quad (16.4\%).
\end{aligned}$$

2. Praktická část

2.1 Vývoj aplikace vlastní implementací

2.1.1 Analýza požadavků a návrh aplikace

Autor zvolil aplikaci pro správu výdajů, protože představuje ideální kompromis mezi jednoduchostí a komplexností. Typ aplikace je známý a dobře uchopitelný, což umožňuje efektivně porovnávat různé metody implementace (self-code, AI code, low-code) na základě zvolených kritérií. Navíc aplikace přináší unikátní kolaborativní funkci, kterou autor sám využije v praxi, čímž projekt získává přidanou hodnotu nejen z akademického, ale i praktického hlediska.

Aplikace bude zaměřena jak na využití na osobním počítači, tak i na mobilním zařízení. Proto při vývoji bude autor klást důraz na responzivitu aplikace, a tím i její použitelnost na různých typech zařízení. Autor se také bude snažit dodržovat moderní principy návrhu webových aplikací.

Pro realizaci webové aplikace zaměřené na plánování výdajů a společné finanční cíle byla provedena analýza požadavků, na základě níž autor sestavil návrh funkcionalit:

Uživatelé

Aplikace bude mít možnost vytvářet uživatelské účty, které budou ukládány v databázi. Jednotlivé účty budou mít uživatelské jméno, email a heslo. Registrovaní uživatelé budou mít přístup do celé aplikace, která bude v databázi ukládat jejich finance.

Evidence výdajů a sledování rozpočtu

Uživatelé aplikace budou mít možnost evidovat své příjmy a výdaje. Při zadávání transakce uživatel zvolí typ (příjem nebo výdaj), zadá částku, datum transakce a případně krátký popis. Aplikace průběžně vypočítává aktuální finanční zůstatek, což umožňuje uživateli neustále sledovat stav svého rozpočtu. Tato funkce je základem správy financí a pomáhá uživatelům v efektivním plánování a kontrole jejich výdajů.

Vytváření a správa finančních cílů

Další klíčovou funkcí bude možnost vytváření finančních cílů. Uživatel si stanoví konkrétní cíl například našetřit určitou částku na dovolenou, nový automobil nebo vybavení domácnosti. Při nastavování cíle uživatel určí cílovou částku a procentuální podíl příjmu, který se má automaticky přidávat do tohoto cíle. Aplikace následně

při každém zaznamenání příjmu automaticky rozdělí částku dle nastavených pravidel a příslušnou část připíše k finančnímu cíli.

Přehledová statistika a grafy

Pro přehlednost a lepší informovanost uživatelů bude aplikace obsahovat statistické přehledy a vizualizace ve formě grafů a tabulek.

Spolupráce s přáteli na společných cílech

Unikátní funkcionalitou této aplikace bude možnost spolupracovat na finančních cílech s dalšími uživateli. Uživatel může vytvořený cíl sdílet se svými přáteli či rodinou. Každý přizvaný uživatel může sledovat společný pokrok a přispívat k plnění cíle. Každý uživatel má vlastní nastavení pro výši svého příspěvku, což umožňuje spravedlivé rozdělení podílu na společném cíli. Tato kolaborativní funkce je zvláště užitečná při plánování společných akcí jako jsou dovolené, nákup společného dárku či spoření na velké rodinné investice.

2.1.2 Návrh databáze a její implementace

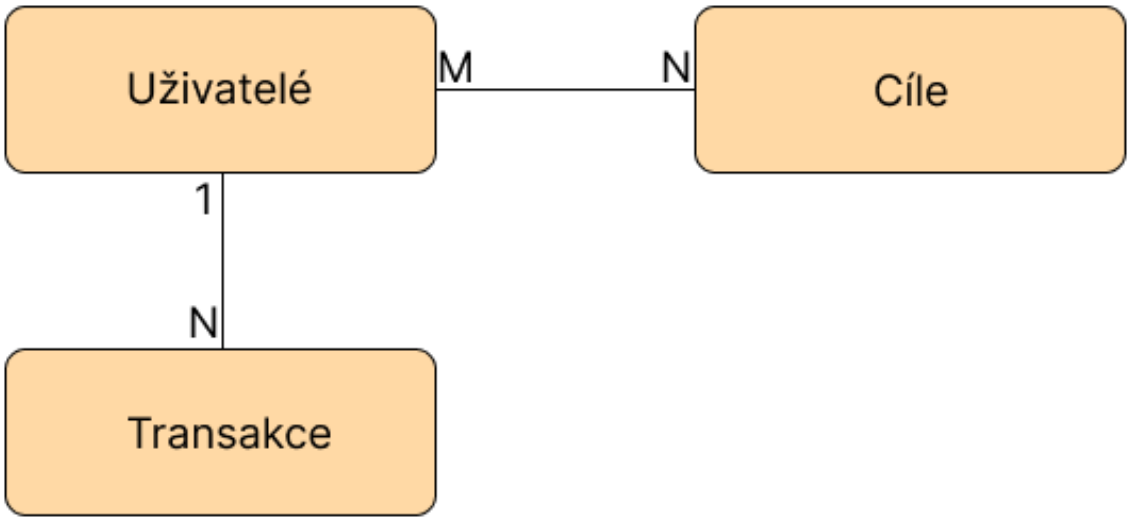
Konceptuální model

Prvním krokem implementace aplikace bylo vytvoření konceptuálního diagramu, který slouží k vizuálnímu znázornění základních entit a jejich vzájemných vztahů. V konceptuálním modelu autor zvolil následující entity a vztahy:

Uživatelé: hlavní entita reprezentující uživatele aplikace. Tato entita je klíčová, protože uchovává údaje o uživatelích a slouží jako spojovací článek pro další entity.

Transakce: reprezentuje jednotlivé finanční transakce (výdaje či příjmy), které uživatel zaznamenává. Vztah mezi uživatelem a transakcemi je typu 1:N, protože jeden uživatel může zaznamenat mnoho transakcí, ale každá transakce patří vždy jen jednomu uživateli.

Cíle: reprezentuje finanční cíle uživatelů. Zvolený vztah mezi uživateli a cíli je typu M:N, protože jeden cíl může být sdílen více uživateli a zároveň jeden uživatel může spravovat nebo spolupracovat na více cílech současně. Tato volba vychází z klíčové kolaborativní funkce aplikace, která umožňuje sdílení a spolupráci na společných finančních cílech.



Obrázek 2.1.1: Konceptuální model

Zdroj: Autor

Databáze

Druhým krokem bylo vytvoření modelu databáze na základě konceptuálního diagramu. V tomto kroku autor konkrétně definoval tabulky a jejich atributy:

Tabulka „users“: ukládá informace o uživatelích. Hlavní atributy jsou:

- **user id (int)**: jednoznačný identifikátor uživatele (primární klíč)
- **username (varchar)**: uživatelské jméno pro přihlášení
- **email (varchar)**: emailová adresa uživatele pro komunikaci a obnovu hesla
- **password (varchar)**: zašifrované heslo uživatele pro bezpečný přístup

Tabulka „transactions“: uchovává jednotlivé finanční transakce. Hlavní atributy jsou:

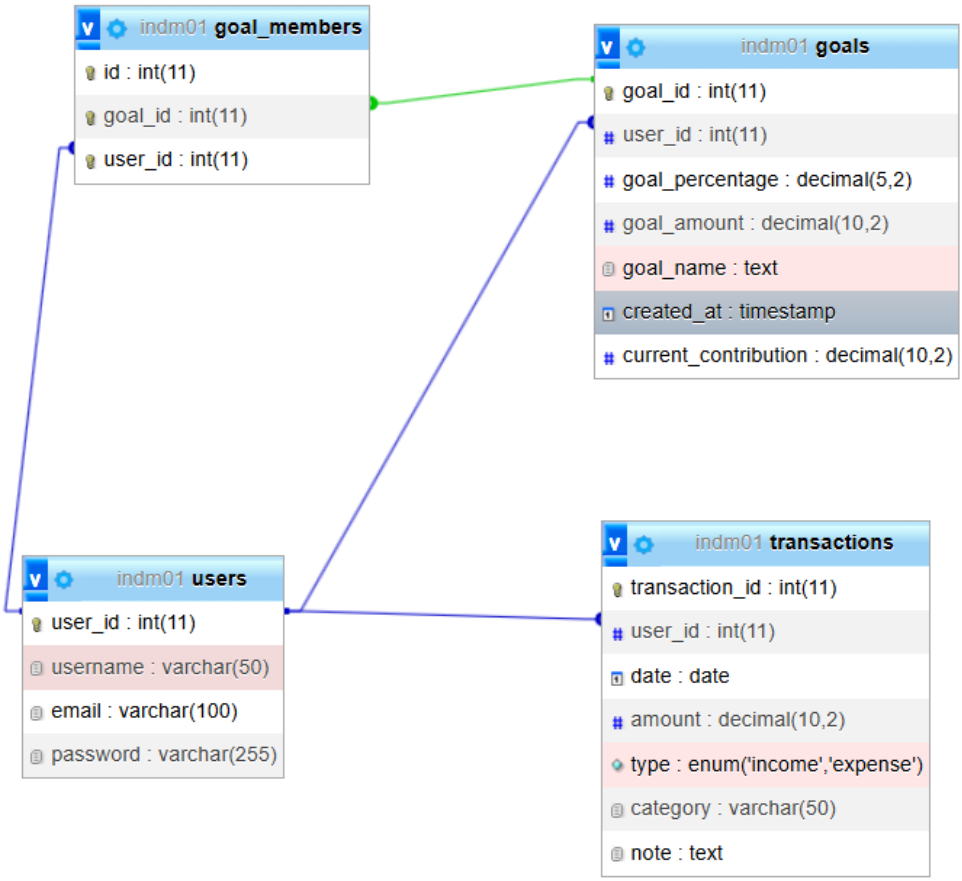
- **transaction id (int)**: jednoznačný identifikátor transakce (primární klíč)
- **user id (int)**: identifikátor uživatele, který transakci vytvořil (cizí klíč)
- **date (date)**: datum provedení transakce
- **amount (decimal)**: částka transakce
- **type (enum)**: typ transakce (příjem nebo výdaj)
- **category (varchar)**: kategorie transakce pro lepší organizaci a statistiky
- **note (text)**: volitelná poznámka k transakci

Tabulka „goals“: obsahuje informace o finančních cílech uživatelů. Hlavní atributy jsou:

- **goal id (int)**: jednoznačný identifikátor cíle (primární klíč)
- **user id (int)**: identifikátor tvůrce cíle (cizí klíč)
- **goal name (text)**: popis cíle
- **goal amount (decimal)**: cílová částka, které chce uživatel dosáhnout
- **goal percentage (decimal)**: procentuální část příjmu, která je automaticky přidělována cíli
- **current contribution (decimal)**: aktuální stav naspořené částky
- **created at (timestamp)**: datum vytvoření cíle

Tabulka „goal members“: propojuje uživatele s cíli v rámci kolaborace (řeší vztah M:N). Atributy jsou:

- **id (int)**: jednoznačný identifikátor propojení (primární klíč)
- **goal id (int)**: identifikátor sdíleného cíle (cizí klíč)
- **user id (int)**: identifikátor uživatele, který je členem cíle (cizí klíč)



Obrázek 2.1.2: Databáze

Zdroj: Autor

2.1.3 Vývoj backendu v PHP

Pro vývoj serverové části aplikace byl autorem zvolen jazyk PHP, a to zejména kvůli osobní zkušenosti autora.

Je důležité zmínit, že autor při tvoření backendu paralelně tvořil i frontend. Nebylo to samozřejmě ve finální podobě, ale pro testování jednotlivých funkcionalit bylo nutné vždy vytvořit alespoň základní kostru HTML a do ní přidávat potřebné prvky jako tlačítka a formuláře.

Práce na backendu byla organizována pomocí jednoduchého osobního systému inspirovaného metodikou Scrum. Autor si celý vývojový proces rozdělil do několika dílčích logických celků („sprintů“), přičemž na každý sprint byly vyhrazeny přibližně dva dny. Tato organizace umožnila flexibilně reagovat na výzvy při vývoji, rychle testovat a upravovat jednotlivé části bez narušení celkové struktury systému.

Registrace a přihlášení uživatele

Jako první byl vytvořen formulář, který přijímá jméno, e-mail a heslo. Formát e-mailu a minimální délka hesla je validována. Hesla jsou ukládána pomocí funkce password_hash(). Také byl vytvořen formulář pro přihlášení, kde je nutné vyplnit e-mail a heslo. Přihlášení probíhá porovnáním zadaného hesla s hashem uloženým v databázi pomocí password_verify(). Po úspěšném přihlášení je uživateli přiřazena session.

Po úspěšné implementaci této základní logiky přihlašování se autor přesunul k jádru aplikace. Za toto jádro autor považuje sledování příjmů a výdajů.

Přidávání a správa výdajů

Autor vytvořil formulář, který zpracovává zadávání příjmů a výdajů. Každá transakce obsahuje částku, datum, typ (příjem/výdaj), kategorii a volitelnou poznámku. Dále byly přidány metody na získání jednotlivých transakcí pro aktuálně přihlášeného uživatele.

Jako další logický krok autor považoval vytváření finančních cílů.

Vytváření a správa finančních cílů

Byl vytvořen další formulář, kde uživatel zadá název cíle, částku, které chce dosáhnout, a procento z příjmu, které se bude do daného cíle dávat. Backend při každém příjmu zkontroluje aktivní cíle a automaticky provede aktualizaci příspěvku. Bylo nutné zamezit plnění cílů na více než 100 %. Dále byla autorem implementována možnost každý cíl smazat. Zde vznikal problém, kdy se našetřené peníze po smazání cíle

vymazaly z databáze, a proto byl implementován systém vrácení peněz. Při každém smazání je tak našetřená částka vrácena na hlavní účet.

Statistiky a grafy

V backendové části bylo potřeba poskytnout agregovaná data pro grafy zobrazené ve frontendové části - rozdělení výdajů dle kategorií, rozdělení příjmů a výdajů, průběh plnění cílů.

Základní funkcionalita tak byla kompletní. Autor se nyní pustil do speciální funkce aplikace, a to možnosti sdílení jednotlivých cílů mezi uživateli.

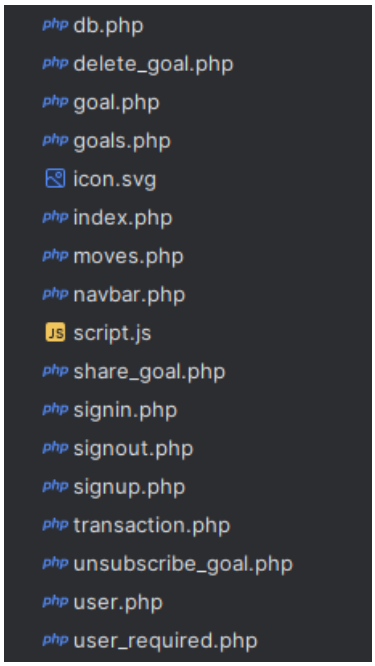
Sdílení cílů

Ve správě cílů autor přidal možnost kromě mazání i daný cíl sdílet. Pomocí tabulky goal_members autor implementoval možnost pozvat jiného uživatele ke spolupráci na cíli. Bylo nutné předělat většinu logiky zpracování finančních cílů, protože bylo potřeba se dotazovat i na výše zmíněnou tabulku a nejen na tabulku goals.

Během celého procesu vývoje backendu se autor snažil ošetřovat jednotlivé vstupy a také používat takzvané "prepared statements" kvůli bezpečnosti aplikace a bránit se tak proti útokům jako Cross-site scripting (XSS).

Backend aplikace je strukturován do několika souborů, které mají jasně definované role:

- **index.php, navbar.php**: základní struktura aplikace a navigační panel
- **db.php**: zajišťuje připojení k databázi.
- **signup.php, signin.php, signout.php**: zpracování registrace, přihlášení a odhlášení uživatele.
- **transaction.php, moves.php**: správa transakcí uživatele.
- **goal.php, goals.php, delete_goal.php, share_goal.php, unsubscribe_goal.php**: správa finančních cílů, včetně sdílení.
- **user.php, user_required.php**: správa údajů uživatelů a zabezpečení přístupu.
- **script.js**: grafy a další JavaScript funkcionalita



Obrázek 2.1.3: Struktura aplikace

Zdroj: Autor

Časová náročnost vývoje backendu:

Vývoj backendové části byl realizován během několika iterací. Autor si jednotlivé části zaznamenával s odhadem času:

- Registrace a autentizace uživatele - *2 hodiny*
- Implementace transakcí - *3 hodiny*
- Finanční cíle - *2 hodiny*
- Implementace sdílení cílů, spolupráce - *5 hodin*

Celkový čas věnovaný vývoji backendové části tedy činil přibližně 12-13 hodin. Autorovy se tak podařilo vytvořit funkční a bezpečný backend, který splňuje všechny požadavky definované v návrhu aplikace a databáze.

2.1.4 Vývoj frontendu v JavaScriptu a CSS

Při vývoji frontendu autor používal kombinaci HTML, JavaScriptu a CSS, přičemž styling aplikace probíhal pomocí frameworku Bootstrap. To umožnilo rychlou tvorbu responzivního uživatelského rozhraní, které je snadno použitelné jak na desktopu, tak i na mobilních zařízeních.

Autor postupoval podle navržených funkcionalit, které implementoval krok za krokem. Rozhraní bylo rozděleno na následující části.

Styling pomocí Bootstrapu

Všechen styling probíhal přímo v HTML dokumentech za pomoci Bootstrap tříd, což zjednodušilo a urychlilo vývoj. Použití Bootstrapu zajistilo konzistentní a responzivní vzhled aplikace na různých typech zařízení bez potřeby detailního vlastního CSS.

Jako první část se autor rozhodl nadesignovat hlavní dashboard aplikace.

Přehled rozpočtu a výdajů

Na hlavní stránce bylo implementováno shrnutí aktuálního zůstatku a kolik uživatel našetřil. Také zde byly vloženy grafy, vytvořené díky knihovně Charts.js, pro vizualizaci dat, jako jsou rozdělení výdajů a vývoj finančních cílů. Součástí stránky je také formulář pro zaznamenávání příjmů a výdajů. Použitím karet v Bootstrapu vzniklo přehledné rozhraní, které poskytuje uživateli přehled o jeho financích. Nachází se zde také tlačítka na zvolení, zdali chce uživatel vidět statistiky za posledních 30 dní či za celou dobu.

Obrázek 2.1.4: Počítačové zobrazení dashboardu

Zdroj: Autor

Následně bylo potřeba vytvořit způsob, jak se po aplikaci pohybovat, a tak jako další si autor vytvořil komponentu pro navigaci.

Navigační lišta

Navigační lišta byla vytvořena pomocí komponenty Bootstrapu. V levé části se nachází název aplikace a odkazy na hlavní stránky aplikace jako správa cílů a pohyby. Pravá část lišty obsahuje dynamické prvky, kde se přihlášenému uživateli zobrazí jeho uživatelské jméno, které slouží také pro změnu hesla a možnost odhlášení. Nepřihlášeným uživatelům se zobrazují odkazy pro přihlášení a registraci.

Pohyby

Autor vytvořil stránku, která v tabulce zobrazuje jednotlivé transakce na účtu od nejnovějších po nejstarší. Příjmy a výdaje jsou od sebe odděleny barevně.

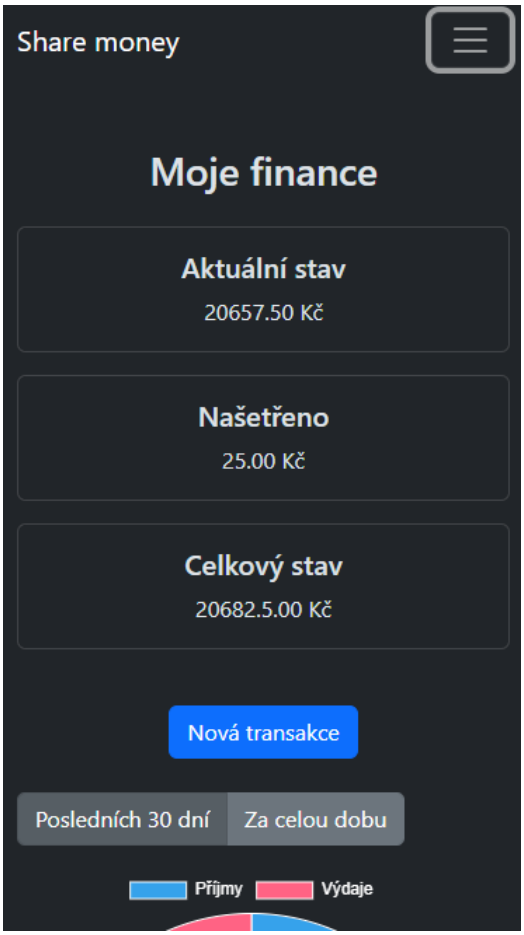
Uživatel

Na stránce uživatele autor vytvořil jednoduchý formulář na změnu hesla.

Tvorba finančních cílů a sdílení

V této části byl vytvořen formulář, kde uživatel definuje název cíle, požadovanou částku a procento z příjmů, které chce k cíli přidělovat. Každý cíl je také zobrazen v tabulce a je možnost jej smazat či sdílet. Při sdílení je uživatel vyzván k zadání e-mailu toho, s kým chce cíl plnit.

Autor se tímto přístupem snažil vytvořit moderně vypadající a velmi responzivní rozhraní, které je použitelné na jakémkoliv zařízení. Implementace vizuální stránky aplikace zabrala autorovi 9 hodin.



Obrázek 2.1.5: Mobilní rozhraní

Zdroj: Autor

2.1.5 Testování a ladění aplikace

Autor během vývoje aplikace průběžně prováděl testování a ladění jednotlivých implementovaných funkcionalit. Testování probíhalo vždy po dokončení již zmíněného mini sprintu a jeho cílem bylo ověřit, jak se aplikace chová v různých scénářích. Díky nacházení a následným opravám se autor snažil zajistit stabilitu a funkčnost aplikace.

Po dokončení aplikace byl autorem proveden ještě finální test, kdy byla postupně procházena celá aplikace, od registrace a přihlášení uživatelů přes správu transakcí až po finanční cíle a jejich sdílení. Autor narazil na několik problémů, které bylo nutné řešit:

- Při prvním přihlášení nově vytvořeného účtu byly problémy se zobrazováním hodnot, protože databáze ještě neobsahovala žádné transakce nebo cíle. Autor

musel implementovat kontrolu na prázdná pole, aby byla aplikace funkční i bez dat.
- Značná část ladění se týkala funkcionality sdílení finančních cílů mezi uživateli. Bylo nutné provést podrobnější ladění logiky sdílení a přístupových práv, aby se zajistilo správné zobrazení a aktualizace sdílených cílů pro všechny přizvané uživatele.

Autor testoval jednotlivé fáze po tuto dobu:

- Registrace a autentizace uživatele - *30 minut*
- Transakce - *1 hodina*
- Finanční cíle - *1 hodina*
- Sdílení cílů, spolupráce - *2 hodiny*

Finální testování pak zabralo další 2 hodiny. Celkem tedy 7-8 hodin.

Díky tomuto systematickému přístupu k testování a ladění autor zajistil, že výsledná aplikace je stabilní, spolehlivá a odpovídá původním požadavkům a specifikacím.

2.1.6 Zhodnocení vlastního přístupu

Aplikace splňuje všechny zadané požadavky. Byla implementována jak základní funkcionalita (registrace, přihlášení, správa výdajů a cílů), tak i pokročilejší kolaborativní funkce sdílení finančních cílů mezi uživateli.

Organizace práce, která byla inspirovaná metodikou Scrum, se ukázala jako velmi praktická i pro tento menší projekt. Rozdělení vývoje do krátkých iterací („sprintů“) umožnilo autorovi identifikovat problémové části systému. Jediným nedostatkem bylo ne vždy úplné dodržení stanovených plánů, což občas vedlo k prodloužení některých částí práce.

Celková časová zátěž vyrobení funkční aplikace byla v součtu tedy přibližně 30 hodin. Čas se zkrátil znalostí použitých technologií, čímž odpadla nutnost učit se nové věci. Samozřejmě bylo potřeba dohledávat určité funkce jak v PHP, tak nejvíce v Bootstrap, se kterým autor neměl tolik zkušeností. Vše ale bylo velmi dobře dokumentováno na oficiálních stránkách, a tím pádem nebyl problém cokoliv najít.

Aplikace určitě není dokonalá a má spoustu možností, jak by mohla být vylepšena a doplněna. Pokud by služba měla být veřejně nasazena, tak si autor myslí, že díky použití známých technologií by neměl být problém službu rozšířit a zdokonalit. Kód je přehledně rozdělen do jednotlivých souborů a je psán anglicky pro větší čitelnost.

V případě budoucího rozšíření aplikace by se autor zaměřil především na přidání notifikačního systému, který by uživatele informoval o důležitých událostech a změnách a integraci bankovních API pro automatickou synchronizaci transakcí.

Celkově autor hodnotí svůj přístup jako úspěšný, přičemž prostor ke zlepšení vidí především v lepším plánování jednotlivých sprintů a rozšíření bezpečnostních a uživatelských funkcí aplikace.

2.2 Vývoj aplikace s využitím AI asistenta

2.2.1 Analýza požadavků a výběr AI modelu

Požadavky na aplikaci jsou identické jako v předchozí části práce. Autor se tedy pomocí umělé inteligence snažil vytvořit aplikaci, která splňuje stejná kritéria.

Pro realizaci aplikace pomocí AI přístupu autor zvolil nástroj ChatGPT, konkrétně model o3-mini-high. Volba tohoto modelu vzešla z několika důvodů. Tím hlavním byla autorova předchozí pozitivní zkušenost s tímto modelem z praxe. Autor tohoto AI asistenta používá pro vytváření jednoduchých metod v jazyce C# v jeho zaměstnání.

Dalšími důvody byla jednoduchost použití. Model je schopný generovat celé třídy a ne pouze úryvky kódu. Také se jedná o tzv. „reasoning model“, tedy model pro řešení složitějších problémů. Díky tomu byl schopen nejen generovat kód, ale také reagovat na autorovy požadavky, analyzovat problémy a navrhovat vhodná řešení během celého vývojového cyklu.

2.2.2 Úvodní prompt a analýza požadavků

Autor zahájil vývoj aplikace zadáním následujícího úvodního promptu modelu o3-mini-high:

"Pojďme vytvořit webovou aplikaci na základě těchto požadavků, které jsem vytvořil. Použij základní stack HTML, PHP, CSS a JavaScript. Můžeš používat knihovny a frameworky. Bude potřeba vytvořit vše, takže postupně piš, čím bys začal a následně konkrétní implementaci pro tuto aplikaci:"

Za tímto promptem autor vložil seznam požadavků definovaných v kapitole 2.1.1. Následně umělá inteligence nejprve analyzovala všechny uvedené požadavky a navrhla vhodnou strukturu databáze i celkovou strukturu projektu:

```
/expense-manager
├── index.php          -> vstupní bod (možná přesměrování na dashboard)
├── login.php          -> přihlašovací formulář a~logika
├── register.php       -> registrační formulář a~logika
├── dashboard.php      -> hlavní obrazovka pro evidenci transakcí, cílů
├── goal.php           -> stránka pro vytváření a~správu finančních cílů
├── collaborate.php    -> stránka pro správu spolupráce u~cílů
├── inc/
```

```
│   ├── db.php       -> připojení k~databázi (PDO) a~konfigurace
│   ├── header.php   -> společný header (HTML hlavička s~Bootstrapem)
│   └── footer.php   -> společná patička
├── css/
│   └── style.css    -> vlastní styly (doplněk k~Bootstrapu)
└── js/
└── main.js      -> vlastní skripty, inicializace Chart.js apod.
```

Poté AI model začal postupně generovat první návrhy kódů jednotlivých částí aplikace, čímž byl zahájen samotný vývoj.

2.2.3 Generování a implementace databáze s AI

Autor následně zadal modelu požadavek na vytvoření SQL kódu pro implementaci databáze. Prompt zněl následovně:

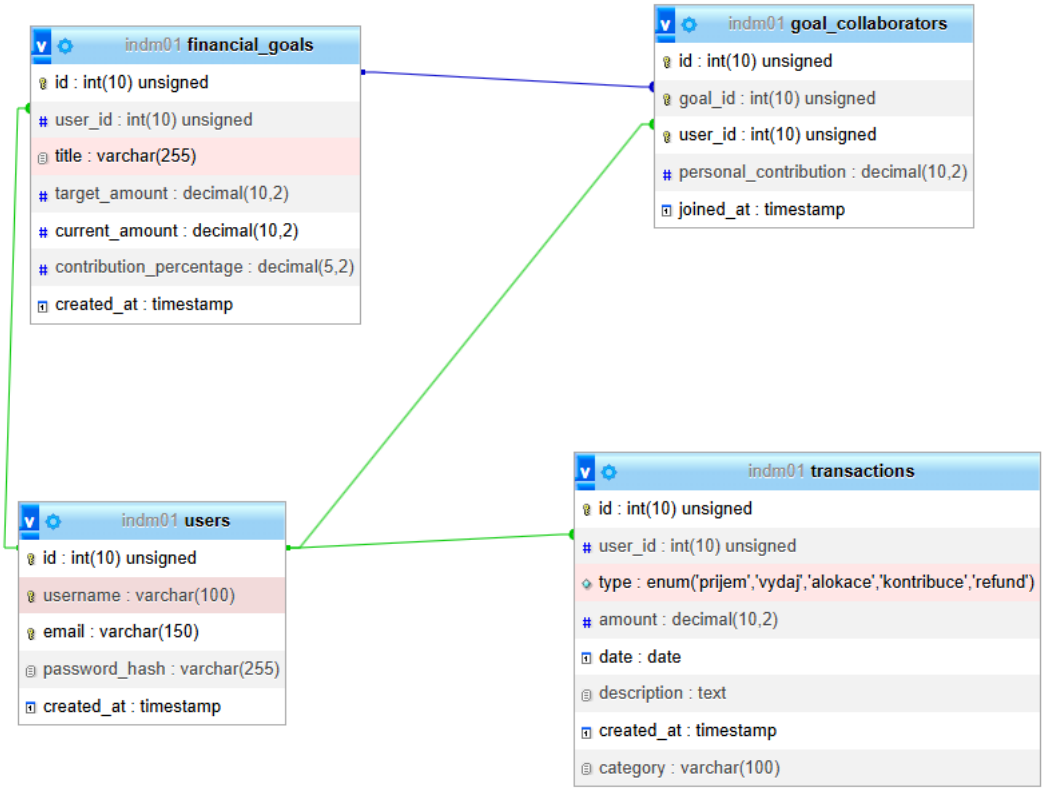
"Napiš mi tedy ještě SQL příkazy pro vytvoření databáze a piš vše jako by to byla reálná implementace (jak jsi psal věnovat pozornost bezpečnosti (ochrana proti SQL injection, validace vstupů, zabezpečení session), dále optimalizovat kód pro lepší škálovatelnost atd.)."

AI model vygeneroval komplexní a robustní SQL skript, který autor vložil do databázového systému. Databáze byla úspěšně vytvořena hned napoprvé. Vygenerovaná databáze se velmi podobá databázi vytvořené autorem v self-code části práce. Struktura databáze je následující:

- **users** - informace o uživatelích (id, username, email, password_hash, created_at)
- **transactions** - záznamy o příjmech a výdajích (id, user_id, type, amount, date, description, category, created_at)
- **financial_goals** - finanční cíle uživatelů (id, user_id, title, target_amount, current_amount, contribution_percentage, created_at)
- **goal_collaborators** - správa spolupráce na finančních cílech (id, goal_id, user_id, personal_contribution, joined_at)

Databáze byla z velké části již od začátku kompletní. Během vývoje však autor dvakrát narazil na nedostatek v databázi. Poprvé to byla chybějící možnost zvolit kategorii výdaje, a tak bylo potřeba vytvořit nový sloupec v tabulce transactions. Dále také díky tomu, že AI u typu transakce zvolila jako datový typ enum, tak v průběhu vývoje bylo třeba tento enum rozšířit z původních dvou hodnot na pět. Umělá inteligence si s těmito problémy uměla jednoduše poradit a velmi rychle vyřešila problémy jednoduchými SQL příkazy:

```
ALTER TABLE transactions ADD COLUMN category VARCHAR(100) DEFAULT NULL;
ALTER TABLE transactions MODIFY COLUMN type ENUM('prijem', 'vydaj',
'alokace', 'kontribuce', 'refund') NOT NULL;
```



Obrázek 2.2.1: Databáze vytvořená AI

Zdroj: Autor

2.2.4 Generování backendu pomocí AI

Autor postupně aplikaci generoval společně s AI krok za krokem přes jednotlivé doporučené třídy. Veškerý generovaný kód byl na první pokus funkční a nebylo nutné řešit větší množství syntaktických či technických chyb.

Backendová část aplikace generovaná modelem o3-mini-high byla na dobré úrovni co se týče bezpečnosti. AI používala bezpečnostní opatření, jako důsledné validace uživatelských vstupů, využívání funkce htmlspecialchars() pro ochranu proti XSS útokům a použití parametrizovaných dotazů (prepared statements) proti SQL injection. Díky tomu byla základní bezpečnost backendu dobře zajištěna již v úvodní fázi vývoje.

Naopak hlavním problémem se ukázala být logická provázanost jednotlivých částí aplikace. AI model sice dokázal efektivně generovat jednotlivé samostatné komponenty backendu (například přidávání transakcí či vytváření finančních cílů), ale opakovaně nedokázal pochopit a implementovat logické souvislosti a interakce mezi nimi.

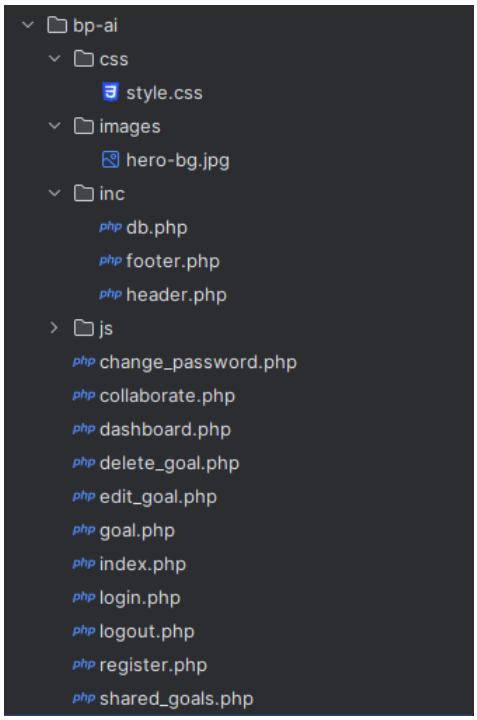
Typickým příkladem tohoto nedostatku byla situace, kdy uživatel přidával peníze k finančnímu cíli. AI automaticky generovala kód, který správně aktualizoval částku v daném cíli, avšak neřešil již odečet příslušné částky z celkové bilance uživatele. Takových případů bylo velké množství. Autor se opakovaně snažil AI navigovat a specifikovat, aby prošla všechny možné logické souvislosti a chybějící interakce mezi komponentami aplikace.

Bohužel umělá inteligence nebyla schopná tyto souvislosti samostatně odhalit. Autor musel vždy dodat konkrétní zadání problému. Jakmile však autor identifikoval a jasně formuloval konkrétní problém, AI již dokázala vytvořit správné řešení a doplnit potřebnou logiku.

AI tedy není schopná najít všechny logické propojení v backendu a není tak schopná samostatně doručit kompletní aplikaci. Toto je problém zejména pro uživatele bez programátorských znalostí. Bez dostatečného pochopení logických souvislostí a správné definice zadání může výsledná aplikace obsahovat vážné nedostatky a chyby, které autor bez programovacích znalostí nemusí být schopen odhalit. Pokud jste však schopen AI správně vést a hledat chyby a nedostatky, které AI vytvoří, tak pak není problém udělat funkční a bezpečný backend.

Struktura projektu navržená AI byla přehledná. AI doporučila základní strukturu složek a souborů, kterou postupně doplňovala a rozšiřovala dle aktuálních potřeb vývoje. Výsledná struktura aplikace byla následující:

- **css/** - soubor stylů aplikace (style.css)
- **images/** - obrázky použité v aplikaci
- **inc/** - obsahující soubory pro opakované použití:
  - db.php (připojení k databázi)
  - header.php (společná hlavička)
  - footer.php (společná patička)
- **js/** - složka byla původně určena pro JavaScript, avšak veškerý JavaScript byl nakonec AI vložen přímo do PHP souborů
- **PHP soubory** - konkrétní stránky aplikace: index.php, login.php, register.php, dashboard.php, goal.php, collaborate.php, edit_goal.php, delete_goal.php, shared_goals.php, change_password.php, logout.php



Obrázek 2.2.2: Struktura navržená AI

Zdroj: Autor

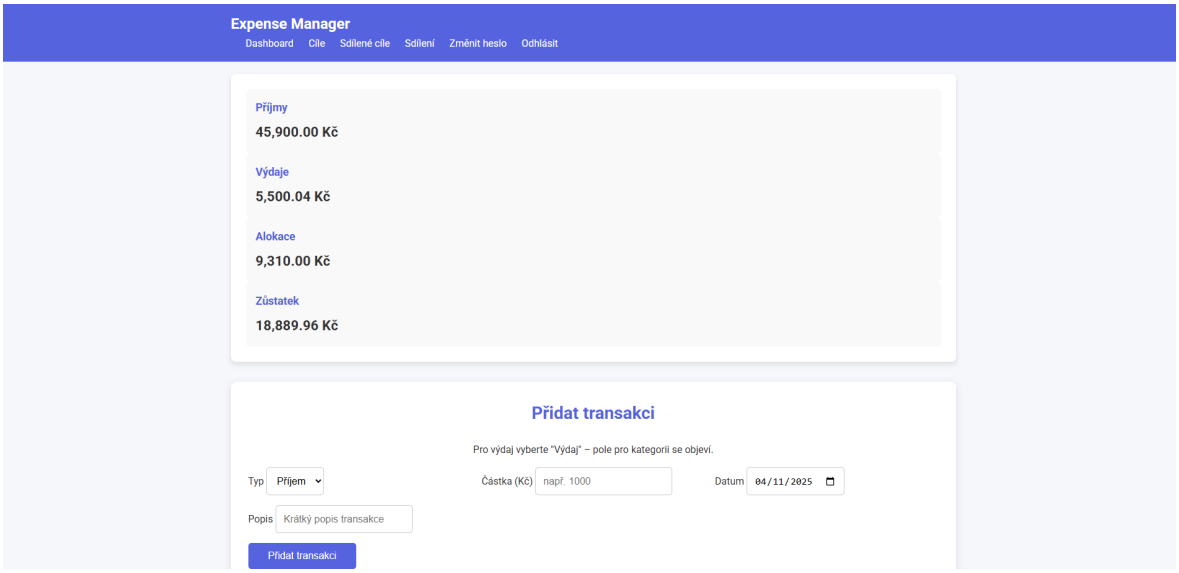
Celkově však AI navrhla backend, který byl velmi podobný tomu, jaký autor vytvořil v části self-code. Hlídala bezpečnostní problémy a dodržovala standardy programování. Co by autor backendové části vytknul, je nízké rozdělení mezi soubory. Určitě by se AI dala nasměrovat tak, aby soubory více rozdělovala, ale ve své základní podobě je kód například dashboardu hodně dlouhý a namíchaný s JavaScriptem. Laik, který by programoval bez programátorských znalostí, by tak mohl na konci skončit s většinou aplikace v jednom souboru.

2.2.5 Generování frontendu s AI

Frontendová část aplikace, kterou AI původně vygenerovala společně s backendem, byla velmi jednoduchá. AI využívala framework Bootstrap, který si sama naimportovala, avšak nebyla to nejaktuálnější verze. Funkčně by tento design stačil, ale byl velmi generický.

Po dokončení backendu se tedy autor rozhodl společně s AI frontend přepracovat a vylepšit pomocí CSS. Úpravy již existujícího kódu byly však složitější, než autor původně očekával. AI často dělala chyby, měnila vzhled částí aplikace, které neměla ovlivňovat, nebo naopak zcela ignorovala některé požadované úpravy. Autor postupně

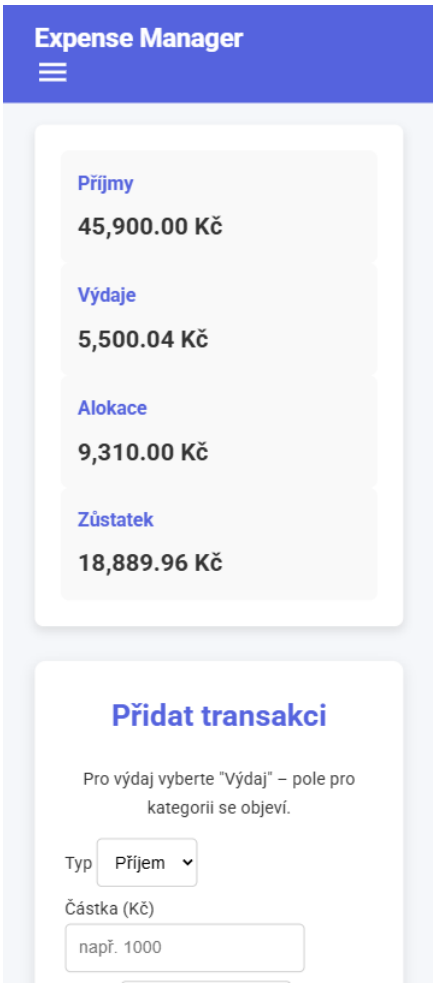
AI posílal jednotlivé části aplikace na úpravu. Jednotlivé úpravy si vždy vyžádaly několik iterací, než byl vizuál dostatečný.



Obrázek 2.2.3: Počítačové zobrazení vytvořené AI

Zdroj: Autor

Dalším problémem byla responzivita, kterou aplikace na začátku díky využití bootstrapu měla, ale postupným přestylováním se tato funkcionalita ztrácela. Bylo proto nutné umělou inteligenci několikrát upozornit, aby doplnila funkčnost pro responzivitu. Poradila si s tím obstojně.



Obrázek 2.2.4: Mobilní zobrazení vytvořené AI

Zdroj: Autor

Autor využil AI také pro vygenerování obrázku na landing page, což fungovalo velmi dobře a vylepšilo to celkový dojem aplikace.

Při práci na frontendu se ukázalo, že AI je schopna rychle navrhnout základní struktury. Bez konkrétních příkazů není však stejně jako v backendu schopna sama vymyslet vzhled aplikace a zachovat jeho konzistentnost. Autor tak musel vždy důkladně zkontrolovat celou aplikaci, protože AI občas změnila i prvky v úplně jiných částech systému.

2.2.6 Testování aplikace generované AI

Testování aplikace generované pomocí modelu o3-mini-high autor provedl manuálně na základě instrukcí, které poskytla umělá inteligence. AI sama o sobě totiž nebyla schopna aplikaci spustit, provést její testování a ani analyzovat její chování přímo v provozu.

Autor proto požádal AI, aby navrhla testovací plán - tedy které konkrétní části aplikace a jakým způsobem by měly být otestovány. Tento přístup byl efektivní v tom, že byly jasně definované body, které bylo potřeba prověřit. Mezi testované oblasti patřily například registrační a přihlašovací formuláře, zadávání transakcí, vytváření a sdílení finančních cílů a další interakce v uživatelském rozhraní.

Během tohoto testování autor narazil na několik chyb. Všechny tyto problémy byly umělou inteligencí opraveny.

Autor by určitě v budoucnu uvítal, kdyby byla umělá inteligence samostatně schopna aplikaci spustit a otestovat. Avšak ani tento přístup nebyl špatný a mít rámec, podle kterého testovat nebylo špatné.

2.2.7 Časová náročnost

Celkově autor vyvíjel aplikaci s pomocí AI necelých 22 hodin. Pro jednotlivé vrstvy aplikace to je:

- Frontend - *8 hodin*
- Databáze - *30 minut*
- Backend - *9 hodin*
- Testování - *4 hodiny*

2.2.8 Zhodnocení využití AI při vývoji

Aplikace vytvořená za pomoci ChatGPT modelu o3-mini-high naplnila všechny definované cíle projektu. Během vývoje byly úspěšně pokryty jak základní požadavky, tak i pokročilé prvky jako je sdílení finančních cílů mezi uživateli.

Zásadní výhodou tohoto přístupu byla schopnost rychle navrhnout technický základ projektu. AI dokázala vygenerovat ucelenou strukturu databáze, navrhnout projektové složky a vytvořit první funkční prototyp backendu i frontendu. AI navíc průběžně generovala bezpečný kód - využívala například připravené SQL dotazy a validaci vstupů.

Hlavní problémem tohoto způsobu vývoje byla neschopnost umělé inteligence pochopit hlubší logické souvislosti mezi jednotlivými částmi aplikace. Model často vytvořil izolované komponenty, které sice fungovaly samostatně, ale nezohledňovaly jejich interakce mezi sebou. Autor tak musel hledat tyto souvislosti sám a vždy je AI podstrčit.

Celkově autor hodnotí práci s AI jako přínosnou, zejména v počátečních fázích návrhu aplikace, kdy je potřeba rychle získat funkční kostru systému. Vývoj však vyžaduje aktivní dohled, zkušenosti a schopnost správně navést model, aby byl výsledek opravdu funkční a kvalitní. V budoucnu by autor doporučil využít AI především jako chytrého asistenta v kombinaci s vlastním vývojem, nikoliv jako plně autonomní nástroj. Tím by se ještě více dokázal zkrátit čas pro vývoj, protože by nebylo nutné AI přesvědčovat o tom, co a jak dělat, ale prostě si části aplikace dopsat podle sebe.

2.3 Vývoj aplikace pomocí no-code platformy

2.3.1 Analýza požadavků a výběr platformy

Požadavky na aplikaci vytvořenou pomocí no-code platformy jsou totožné jako v případě aplikace vytvářené klasickým programováním (self-code). Podrobně jsou tedy uvedeny v předchozích částech práce. Hlavními funkcemi jsou evidence příjmů a výdajů a vytváření a možnost sdílení cílů.

Pro realizaci aplikace pomocí no-code přístupu byly zvažovány různé platformy, například Bubble, Webflow a Wix. Po porovnání dostupných možností byla zvolena platforma Bubble, a to z několika důvodů:

- **Silná integrace s AI**: Bubble nabízí pokročilé možnosti integrace s umělou inteligencí. Bubble umožňuje zadat úvodní příkaz pro AI, dle kterého následně vygeneruje velkou část aplikace. Dále je pak možné generovat i dílčí části jako jednotlivé stránky.
- **Popularita a komunita**: Platforma Bubble má přes 5 milionů uživatelů a 8 milionů vytvořených aplikací. Tudíž existuje spoustu komunitních návodů i dobrá dokumentace.
- **Bezplatná vývojová verze**: Bubble umožňuje kompletní vývoj aplikace v bezplatném režimu, což autorovi umožnilo aplikaci plnohodnotně otestovat a vytvořit bez dodatečných nákladů.

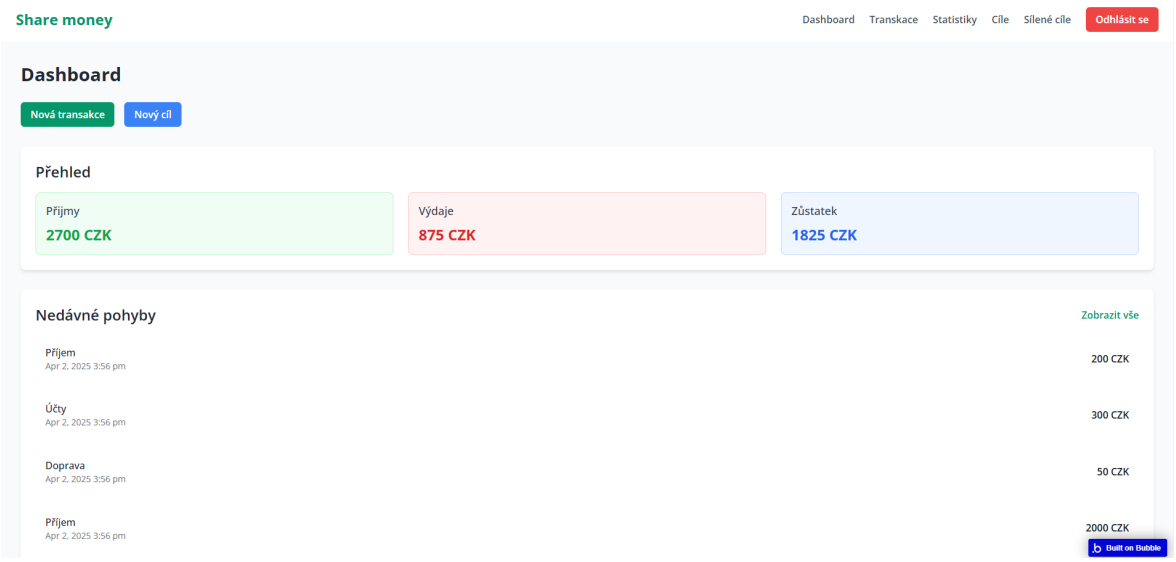
Na základě těchto kritérií byla platforma Bubble vyhodnocena jako nejvhodnější pro realizaci tohoto projektu.

2.3.2 Vývoj frontendu pomocí no-code

Po výběru platformy autor zahájil vývoj aplikace zadáním základního promptu do AI generátoru přímo v prostředí Bubble: *"Make web application for tracking finances, add a functionality of tracking goals and sharing them"*. AI generátor na základě tohoto zadání vytvořil prvotní verzi aplikace, která obsahovala již poměrně kvalitní uživatelské rozhraní a základní strukturu databáze.

Uživatelské rozhraní (frontend) bylo generováno poměrně kvalitně a vyžadovalo pouze drobné úpravy jako doplnění tlačítek a zajištění responzivity. Díky jednoduchému uživatelskému rozhraní na úpravu frontendu, který se podobá například známému prototypovacímu nástroji Figma, se kterým má autor zkušenosti, se většina věcí dala

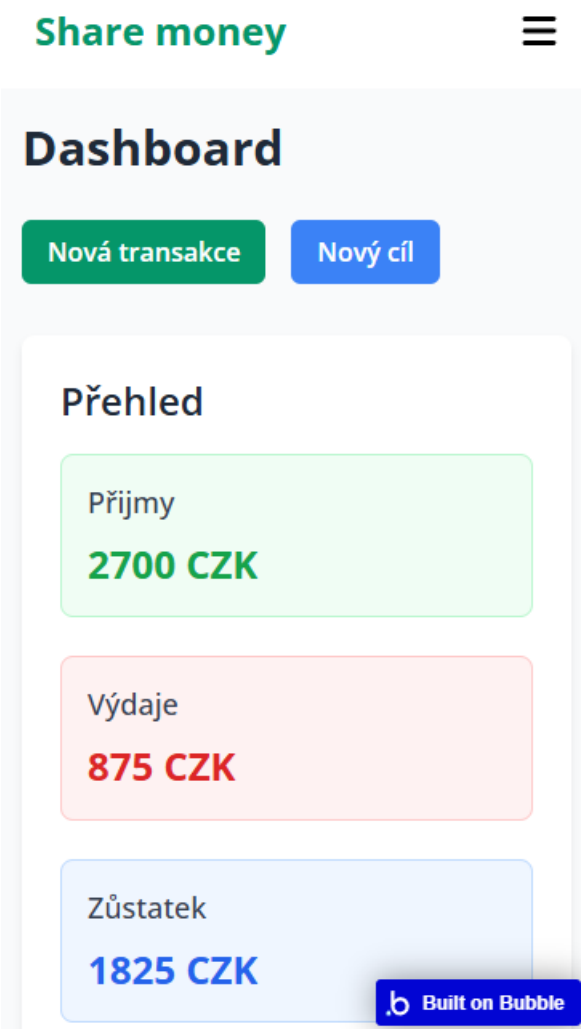
intuitivně vytvořit. Díky využití AI na vytvoření velké části UI, se autor mohl více věnovat dalším částem vývoje.



Obrázek 2.3.1: Dashboard v no-code aplikaci na PC

Zdroj: Autor

Autor celou aplikaci přeložil do češtiny, protože AI ji vygenerovala anglicky. Dále bylo potřeba vytvořit stránku po přihlášení a registraci, která úplně chyběla. Autor se snažil napodobovat styl, který už vytvořila umělá inteligence ve zbytku aplikace. Dále bylo potřeba udělat stránku se statistikami a grafy. Pro vytvoření grafů bylo nutno využít plugin dostupný zdarma v Bubble. Také bylo potřeba vytvořit responzivní navigační panel, jelikož AI vytvořila pouze statický, který by na mobilních zařízeních nefungoval. Autor vytvořil ještě mnoho vyskakovacích oken, AI se některé podařilo vytvořit, ale většinu bylo nutné dodělat.



Obrázek 2.3.2: Dashboard v no-code aplikaci na telefonu

Zdroj: Autor

2.3.3 Vytvoření databáze pomocí no-code

Databáze vygenerovaná pomocí AI také vyžadovala jen menší úpravy, jako byla manuální konfigurace přístupových práv k datům, což bylo nezbytné zejména pro správnou funkčnost sdílení finančních cílů mezi uživateli. Celková struktura, tak jak ji navrhla AI, vlastně odpovídá tomu, jak to autor udělal v self-code části.

Tabulka „User“: slouží pro ukládání uživatelů. Je naprosto stejná jako ji vytvořil autor. Skládá se s:

- **username (text)**: uživatelské jméno
- **email (text)**: emailová adresa
- **password (text)**: heslo

Tabulka „Transaction“: zde jsou uložené transakce, na rozdíl od self-code tu chybí typ transakce (výdaj/příjem) a vše je řešeno přes kategorii. Atributy jsou:

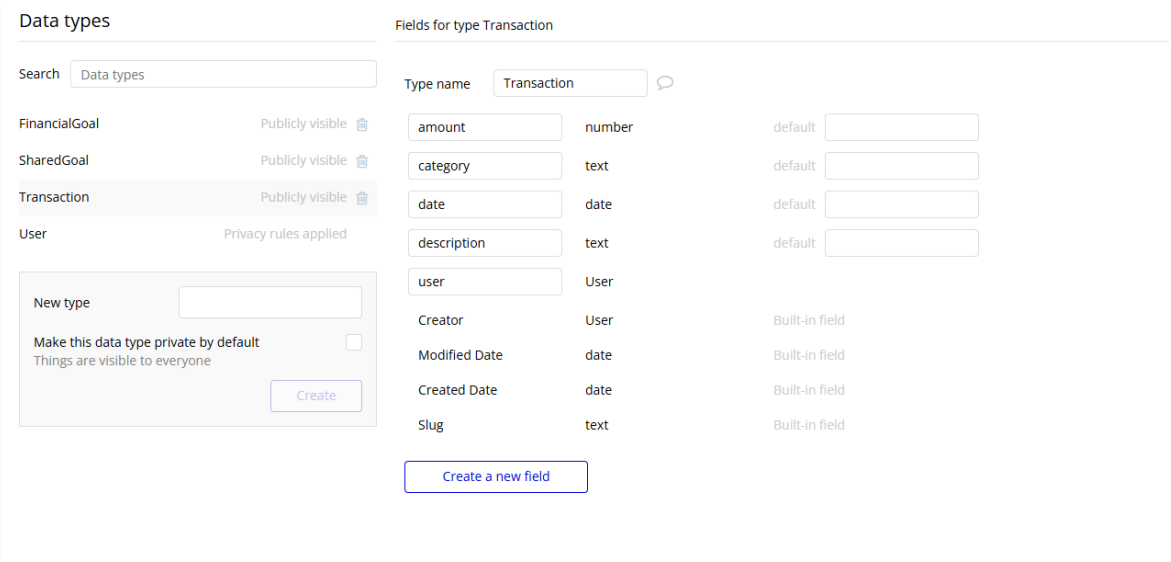
- **amount (number)**: částka transakce
- **category (text)**: kategorie transakce
- **date (date)**: datum transakce
- **description (text)**: popis transakce
- **user (User)**: reference na uživatele, kterému transakce patří

Tabulka „FinancialGoal“: opět se velmi podobá jako ji vytvořil v předchozí části autor. Chybí tu procentuální příspěvek z příjmu. Hlavní atributy jsou:

- **current_amount (number)**: aktuálně naspořená částka
- **goal_name (text)**: název finančního cíle
- **target_amount (number)**: cílová částka
- **user (User)**: reference na uživatele, který cíl vytvořil

Tabulka „SharedGoal“: stejně jako ve vlastní implementaci i tady slouží k řešení M:N vztahu. Atributy jsou:

- **goal (FinancialGoal)**: reference na sdílený finanční cíl
- **shared_with (User)**: reference na uživatele, se kterým je cíl sdílen



Obrázek 2.3.3: Databázová tabulka v no-code prostředí

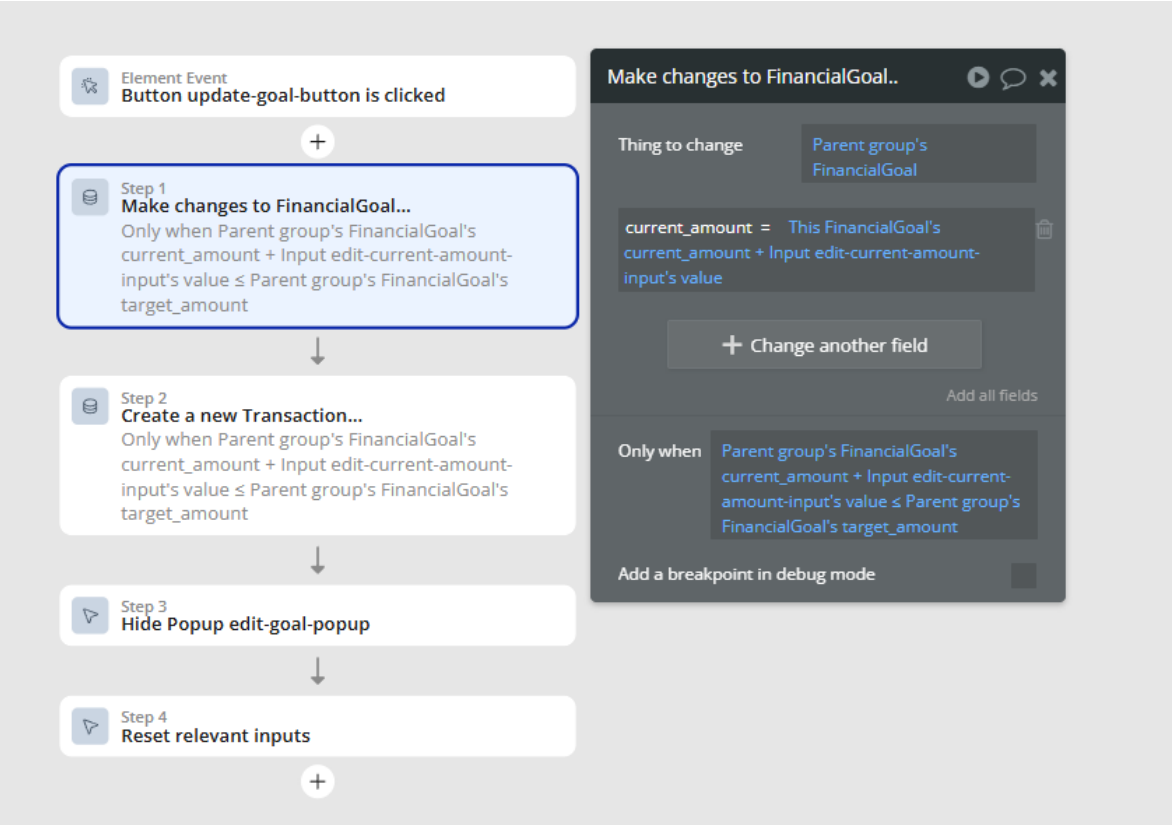
Zdroj: Autor

2.3.4 Vývoj backendu pomocí no-code

Naopak backendová logika byla oblastí, kde AI generátor nedokázal vytvořit funkční řešení. Propojení mezi frontendem a databází prostřednictvím workflows bylo z větší části nutné vytvořit manuálně. Autor tak musel jednotlivé akce, ať už to bylo zobrazení vyskakovacího okna po kliknutí na tlačítko, či celá logika přidání do záznamu do databáze, definovat prostřednictvím Bubble editoru workflows.

Jednodušší operace byly relativně intuitivní a snadno realizovatelné. Existuje zde spoustu užitečných funkcí na prokliky mezi stránkami, zobrazování a skrývání elementů na stránce.

Nicméně složitější backendové operace, jako například přidávání peněz k cíli, bylo ze startu velmi náročné tvořit. Autor strávil několik hodin snahou pochopit a vytvořit první komplexnější workflow. Autor by u takovéto služby, zaměřené spíše na uživatele bez programátorských znalostí, očekával, že bude nabízet intuitivnější a jednodušší způsob tvorby workflow. Vše totiž bylo psáno textově za použití jednoduchých programovacích funkcí, které vám sice editor napovídal, ale neřekl vám, co jednotlivé funkce dělají a proč jsou někdy dostupné a jindy využít nejdou. Autor by zde uvítal nějaký grafický editor, kde by si člověk jednoduše pospojoval třeba části formuláře s atributy v databázi. Zbytek logiky by si pak dodělala služba sama. Přestože autor postupně workflow logiku zvládl, tak se v průběhu implementace objevily i problémy s nekonzistentním chováním platformy - občas něco fungovalo bez problémů v jedné části aplikace, zatímco v jiné části vůbec. Vytváření backendu tak bylo velmi frustrující a zdlouhavé.



Obrázek 2.3.4: Workflow pro přidání peněz k finančnímu cíli

Zdroj: Autor

Dalším problémem bylo rozdělení workflows mezi jednotlivé stránky aplikace, což výrazně komplikovalo orientaci a navigaci v rámci backendové logiky. Autor se často v tomto rozdělení ztrácel, což vedlo ke značné časové náročnosti při hledání a úpravách konkrétních workflows.

2.3.5 Testování v no-code

Testování aplikace probíhalo průběžně během celého vývoje. Vzhledem k tomu, že autor neměl předchozí zkušenosti s platformou Bubble, bylo nutné každé workflow důkladně testovat ihned po vytvoření, aby autor pochopil jeho chování a funkčnost.

Bubble nabízí debug režim, který umožňuje jednoduše krokovat jednotlivé workflow a sledovat jeho chování. Díky tomu mohl autor snadno hledat problémy a následně je opravovat. Velmi užitečné bylo také zobrazení proměnných a jeho obsah využitých v jednotlivých krocích. Toto autorovi velmi pomohlo při vytvoření a testování jednotlivých workflows.

Během vývoje autor zjistil, že existuje plugin pro vypisování proměnných či celých příkazů přímo do konzole prohlížeče. Díky tomuto nástroji mohl autor snadněji kontrolovat výstupy různých operací a měl jasnější představu o tom, jak aplikace pracuje s daty.

Na závěr autor provedl ještě celkové testování aplikace, při kterém ověřil funkčnost jednotlivých částí, stabilitu aplikace a funkčnost sdílení cílů mezi uživateli. Celkově autor hodnotí testovací prostředí Bubble velmi pozitivně, neboť umožnilo jednoduché ladění a rychlé řešení vznikajících problémů během vývoje.

2.3.6 Časová náročnost

Celkově autor vytvářením aplikace strávil přibližně 38 hodin. Většina vývoje pomocí no-code přístupu byla úzce provázaná, a proto nebylo snadné časově oddělit jednotlivé části.

Autor tedy uvádí pouze časové odhady jednotlivých častí:

- Frontend - *12 hodin*
- Databáze - *1 hodina*
- Backend - *20 hodin*
- Testování - *5 hodin*

Pokud by autor znal způsob vytváření backendu již z minulých zkušeností, tak by se pravděpodobně dala část logiky stihnout rychleji. Bohužel díky nekonzistentním výstupům si autor nemyslí, že by množství ušetřeného času bylo nějak velké. Odhaduje tak 5 hodin.

2.3.7 Zhodnocení no-code přístupu

Aplikace vytvořená pomocí platformy Bubble splňuje všechny stanovené požadavky. Všechny funkce včetně složitější funkcionality sdílení cílů mezi uživateli byly implementovány.

Výhodou tohoto přístupu byla možnost rychlého návrhu uživatelského rozhraní a databáze, zejména díky integraci s AI, která poskytla funkční základ. Autor tak mohl značnou část času věnovat backendové logice, která vyžadovala ruční konfiguraci. Výhodou použití platformy Bubble je také možnost snadného nasazení výsledného řešení do produkčního prostředí bez nutnosti vlastního serveru.

Největší slabinou celého přístupu se ukázal být právě vývoj backendu - především komplexnější workflows, které bylo nutné tvořit ručně a často ne zcela intuitivně. Práce s nimi byla časově náročná a někdy i frustrující, zejména nekonzistentnímu chování editoru.

Celkově autor hodnotí tento přístup jako úspěšný, zejména z hlediska rychlého vytvoření funkčního prototypu, využití AI a získání praktické zkušenosti s no-code vývojem. V případě podobného projektu v budoucnu by autor zvažoval kombinaci no-code nástrojů s ručním doprogramováním složitější logiky.

2.4 Vyhodnocení přístupů pomocí Saatyho metody

2.4.1 Přehled porovnaných přístupů

V praktické části byly analyzovány tři různé způsoby vývoje webové aplikace:

- **Self-code**: klasický vývoj pomocí PHP, JavaScriptu, HTML a CSS, kde autor sám navrhl a naprogramoval celou aplikaci.
- **AI-asistovaný vývoj**: aplikace vygenerovaná ve spolupráci s jazykovým modelem ChatGPT (o3-mini-high).
- **No-code vývoj**: tvorba aplikace v platformě Bubble s využitím AI a vizuálního rozhraní.

Každý přístup byl hodnocen podle stejných kritérií a výsledků dosažených v praxi.

2.4.2 Použitá kritéria a jejich váhy

Na základě metody AHP bylo v teoretické části definováno 5 kritérií a jejich váhy:

- **Rychlost implementace** (27.0%)
- **Kvalita výsledného řešení** (20.7%)
- **Flexibilita a udržovatelnost** (18.7%)
- **Náročnost osvojení technologie** (17.2%)
- **Závislost na externích službách** (16.4%)

2.4.3 Bodové hodnocení přístupů podle kritérií

Tabulka ukazuje subjektivní hodnocení přístupů autorem dle praktické zkušenosti v jednotlivých oblastech na stupnici 1-9:

Kritérium	**Self-code**	**AI-asistent**	**No-code**
Rychlost implementace	4	8	6
Kvalita výsledku	8	7	7
Flexibilita a udržovatelnost	9	6	2
Náročnost osvojení	5	6	4
Závislost na externích službách	9	7	3

Tabulka 2.4.1: Subjektivní hodnocení přístupů (1–9)

U rychlosti implementace autor vycházel z časů, které byly zaznamenávány při jednotlivých způsobech implementace. Kvalitu výsledku autor hodnotil dle kompletnosti a celkového dojmu z aplikace. Flexibilita a udržovatelnost byla hodnocena z pohledu čitelnosti kódu a možnosti aplikaci dále rozvíjet. Náročnost bral autor jako obtížnost, se kterou mu při jednotlivých způsobech pracovalo. Závislost je provázanost aplikace s daným prostředkem.

2.4.4 Celkové vážené hodnocení

- Self-code: $0.27 \cdot 4 + 0.207 \cdot 8 + 0.187 \cdot 9 + 0.172 \cdot 5 + 0.164 \cdot 9 = \mathbf{6{,}75}$
- AI-asistent: $0.27 \cdot 8 + 0.207 \cdot 7 + 0.187 \cdot 6 + 0.172 \cdot 6 + 0.164 \cdot 7 = \mathbf{6{,}911}$
- No-code: $0.27 \cdot 6 + 0.207 \cdot 7 + 0.187 \cdot 2 + 0.172 \cdot 4 + 0.164 \cdot 3 = \mathbf{4{,}623}$

2.4.5 Závěrečné zhodnocení

Nejlepšího výsledku dosáhl přístup s využitím AI, který nabídl dobrou rovnováhu mezi rychlostí, kvalitou a nižší náročností na osvojení.

Klasický self-code byl lepší v kvalitě kódu a flexibilitě. Tento přístup je však časově nejnáročnější a vyžaduje plnou znalost všech použitých technologií.

No-code byl pro autora nejhorší variantou jak po stránce náročnosti na osvojení, tak i finálního výsledku. Je závislý na platformě a to sebou nese velká rizika jako je ztráta aplikace či navýšení ceny. Hodí se zejména pro rychlé prototypy nebo jednoduché aplikace bez složité logiky.

Volba ideálního přístupu tak závisí na požadavcích projektu a schopnostech vývojáře. Kdyby se využil přístup kombinování AI s vlastním kódem mohli bychom dojít k velmi podobným výsledkům jako self-code za výrazně nižší čas.

Diskuse

Kapitola je zaměřena na nalezení limitů práce. Autor si uvědomuje, že získaná zjištění vycházejí z jedné ukázkové aplikace, využití jediného no-code nástroje a konkrétní verze AI modelu. Diskuse proto upozorňuje na omezenou zobecnitelnost závěrů a na oblasti, kde je nutné další ověření.

Jedna aplikace

Veškerý vývoj byl směřován pouze na jeden typ konkrétní webové aplikace splňující předem definovanou sadu funkčních požadavků. Aplikace sice byla vybírána s ohledem na její univerzálnost a komplexnost i tak bohužel není možné výsledky zobecnit na všechny typy aplikací. Pro projekty které by řešili úplně jiný problém, měli jinou škálu dat či požadavky mohou jednotlivé přístupy (tradiční programování, AI vývoj, no-code) dopadnout jinak.

Výběr pouze jednoho nocode nástroje

Jako reprezentanta no-code platforem autor využil Bubble. Na trhu se však nachází celá řada alternativ například lovable.dev, Mendix, Microsoft Power Apps. Každá z nich nabízí odlišný rozsah vestavěných komponent, cenu či škálovatelnost. Pokud by byl použit jiný nástroj, výsledné skóre AHP by se zcela určitě změnilo - například platforma lovable umožňuje export kódu a tím by se velmi snížila závislost na externích službách.

Využití pouze jednoho AI asistenta

Při vývoji s využitím umělé inteligence byl autorem použit pouze jeden model o3-mini-high. V dnešní době se AI každým dnem výrazně zlepšuje a vznikají nové modely, které mohou být lepší v rychlosti, přesnosti a celkově ve vývoji aplikací. Během psaní práce OpenAI přišlo s novými modely o4-mini a o3, které v testech dopadají výrazně lépe než o3-mini (OpenAI, 2025).

Subjektivní nastavení kritérií AHP

Metoda AHP vyžaduje ohodnocení významu jednotlivých kritérií. Přestože byly váhy stanoveny na základě rešerše zůstávají z částí subjektivní. Odlišné preference různých zadavatelů by jistě vedly k jinému pořadí kritérií.

Krátkodobé testování

Funkčnost vyvinutých aplikací byla ověřena pouze autorem ve vývojovém prostředí bez dlouhodobého provozu pod reálnou zátěží. Autor si tedy uvědomuje že oblasti jako škálovatelnost a bezpečnost nebyly ověřeny v plném provozu reálnými uživateli.

Směry dalšího výzkumu

Na zmíněné limity přímo navazují možnosti pokračování práce:

1. Rozšířit srovnání o více no-code platforem a novější AI modely.
2. Otestovat aplikace v reálném nasazení.
3. Validovat subjektivní váhy AHP.

Závěr

Cílem této bakalářské práce bylo porovnat tři různé přístupy k vývoji webové aplikace - klasický self-code přístup, vývoj za pomoci AI asistenta a no-code řešení. Výsledkem bylo vytvoření tří verzí stejné aplikace pro správu osobních financí a sdílených cílů, které byly následně porovnány z hlediska kvality, rychlosti, udržovatelnosti i technické náročnosti.

Teoretická část práce se věnovala přehledu technologií používaných při vývoji webových aplikací. Dále byla rozebrána problematika umělé inteligence a její rostoucí využití ve vývoji softwaru, včetně přístupů jako no-code a AI-asistovaný vývoj. V neposlední řadě byla popsána vícekriteriální metoda AHP, která sloužila pro závěrečné porovnání jednotlivých přístupů.

V praktické části byly všechny tři přístupy aplikovány na stejné zadání. Autor postupoval vždy od návrhu přes implementaci až po testování aplikace, přičemž zaznamenával časovou náročnost, problémy i subjektivní hodnocení. V závěru byla pomocí metody AHP provedena analýza, která ukázala, že nejlépe hodnoceným přístupem byl AI-asistovaný vývoj. Ten nabídl dobrý kompromis mezi kvalitou, rychlostí a nároky na osvojení technologie.

Tato práce ukazuje, že různé přístupy mají své výhody i limity a jejich vhodnost závisí na konkrétním kontextu použití. Autor se domnívá, že kombinace AI asistenta a vlastního kódování může být v současné době jedním z nejefektivnějších způsobů vývoje, zejména pro menší projekty nebo jednotlivce.

Použitá literatura

Bapna, S., Shrivastava, V., Pandey, A., & Sharma, A. (2024-03). Analyzing Front-End Web Development Technologies Based on their Interactive Nature, Optimization & Characteristics. *International Journal of Research Publication and Reviews*, *5*(3), 1706–1709.

Bechný, O. (2010). *Návrh metodiky vývoje softwaru z pohledu samostatného vývojáře* [dipl. pr., MASARYKOVA UNIVERZITA].

Bennett, L. (2024-11). *What is Software Engineering? Definition, Basics, Characteristics*. https://www.guru99.com/cs/what-is-software-engineering.html

Berkeley. (2020-06). *What Is Machine Learning (ML)?* https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/

Biswas, S. (2023-01). Role of ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science*, 8–16. https://doi.org/10.58496/mjcsc/2023/002

Boehm, B. W. (1976-12). Software Engineering. *IEEE TRANSACTIONS ON COMPUTERS, C-25*(12).

Bruckner, T. (2012). *Tvorba Informačních Systémů*. Grada Publishing A.s.

Bubble. (2025). https://bubble.io/

Contributors, M. (2023a). *CSS: Cascading Style Sheets*. https://developer.mozilla.org/en-US/docs/Web/CSS

Contributors, M. (2023b). *HTML: HyperText Markup Language*. https://developer.mozilla.org/en-US/docs/Web/HTML

Erickson, J. (2025). *MySQL: Understanding What It Is and How It's Used*. Oracle. https://www.oracle.com/mysql/what-is-mysql/

Erich Gamma, R. J. a. J. V., Richard Helm. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software: Elements of reusable object-oriented software*. Addison-Wesley.

Fiala, P., Josef, J., & Maňas, M. (1997). *Vícekriteriální rozhodování*. Vysoká škola ekonomická v Praze Fakulta informatiky a statistiky.

Geeksforgeeks. (2024a). *Software Development Life Cycle (SDLC)*. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

Geeksforgeeks. (2024b). *Top 8 Software Development Life Cycle (SDLC) Models used in Industry*. https://www.geeksforgeeks.org/top-8-software-development-models-used-in-industry/

GitHub. (2025). *What is GitHub Copilot?* https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot

Group, T. P. (2023). *PHP: Hypertext Preprocessor*. https://www.php.net/

IBM. (2025). *What is Low-Code/No-Code?* IBM. https://www.ibm.com/think/topics/low-code

Isles, A. (2023). *REST - A Deep Dive into REST APIs*. https://aptori.dev/blog/rest-apis-a-deep-dive

Jackson, G. (2024). *What is the software development life cycle (SDLC)?* https://www.ibm.com/think/topics/sdlc

James O'Donnell, M. H., Will Douglas Heaven. (2025). *What's next for AI in 2025*. https://www.technologyreview.com/2025/01/08/1109188/whats-next-for-ai-in-2025/

Jebavý, J. (2024). *Programovací jazyk Go*. https://blog.josefjebavy.cz/cs/programovani/programovaci-jazyk-go

Khan, S. (2023). Role of Generative AI for Developing Personalized Content Based Websites. https://doi.org/10.5281/ZENODO.8328205

Kosek, J. (1999). *PHP tvorba interaktivních internetových aplikací*. Grada Publishing.

Kozon, T. (2024). *Harnessing the Power of Microservices Architecture in Web Development*. https://boringowl.io/en/blog/harnessing-the-power-of-microservices-architecture-in-web-development

Lima, H., & Eler, M. (2021). C++ Web Framework: A Web Framework for Web Development using C++ and Qt. *Proceedings of the 23rd International Conference on Enterprise Information Systems*, 76–87. https://doi.org/10.5220/0010457700760087

Mendix. (2025). https://www.mendix.com/

Moin, T. S. (2023). Overview of Computer Vision. https://doi.org/10.13140/RG.2.2.13989.68327

Muller, M., Chilton, L. B., Kantosalo, A., Martin, C. P., & Walsh, G. (2022-04). GenAICHI: Generative AI and HCI. *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. https://doi.org/10.1145/3491101.3503719

Muna Abu Jaber, A. A. A., Adna Beganović. (2023). Methods and Applications of ChatGPT in Software Development: A Literature Review. *Southeast Europe Journal of Soft Computing*.

Necula, S. (2024-04). Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications. https://doi.org/10.20944/preprints202404.1860.v1

OpenAI. (2022). *Introducing ChatGPT*. https://openai.com/blog/chatgpt

OpenAI. (2025). *Introducing OpenAI o3 and o4-mini*. https://openai.com/index/introducing-o3-and-o4-mini/

Patkar, U., Singh, P., Panse, H., Bhavsar, S., & Pandey, C. (2022-04). PYTHON FOR WEB DEVELOPMENT. *International Journal of Computer Science and Mobile Computing*, *11*(4), 36–48. https://doi.org/10.47760/ijcsmc.2022.v11i04.006

Pereyra, I. (2023). *Universal Principles of UX: 100 Timeless Strategies to Create Positive Interactions between People and Technology*.

Ray, S. (2024). *AI agents — what they are, and how they'll change the way we work*. https://news.microsoft.com/source/features/ai/ai-agents-what-they-are-and-how-theyll-change-the-way-we-work/

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, *1*(1), 83. https://doi.org/10.1504/ijssci.2008.017590

Saaty, T. L., & Vargas, L. G. (2012). *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Springer US. https://doi.org/10.1007/978-1-4614-3597-6

Saaty, T. (1994). How to Make a Decision: The Analytic Hierarchy Process. *Aestimum*, Atti del XXIV Incontro di Studio (1994). https://doi.org/10.13128/AESTIMUM-7138

Salas-Zárate, M. d. P., Alor-Hernández, G., Valencia-García, R., Rodríguez-Mazahua, L., Rodríguez-González, A., & López Cuadrado, J. L. (2015-05). Analyzing best practices on Web development frameworks: The lift approach. *Science of Computer Programming*, *102*, 1–19. https://doi.org/10.1016/j.scico.2014.12.004

SAP. (2025). *What is low-code/no-code application development?* https://www.sap.com/products/technology-platform/build/what-is-low-code-no-code.html

Silberschatz, A., Korth, H., & Sudarshan, S. (2011). *Database System Concepts*. McGraw-Hill Education.

Tran, T. (2024). *Golang Vs. Java: Choose the Right Programming Language for Web Development*. https://www.orientsoftware.com/blog/golang-vs-java/

Voříšek, J. (2015). *Principy a modely řízení podnikové informatiky*. Vysoká škola ekonomická v Praze Fakulta informatiky a statistiky.

W3C. (2023). *HTML and CSS Standards*. https://www.w3.org/standards/webdesign/htmlcss

Welling, L. (2017). *PHP and MySQL web development* (L. Thomson, Ed.; Fifth edition) [Previous edition: 2009]. Addison-Wesley.

Zin, N. A. M., Mazlan, A. N., Arifin, N., Rahman, N. A., Ahmad, M., & Manaf, M. (2024-12). Applying Personal Scrum Methodology in System Development for Project-Based Learning Course. *Journal of Mathematics and Computing Science*, *10*(2), 64–73.

Přílohy

Příloha A: Dokumentace aplikace vytvořené klasickým způsobem

Odkaz na dokumentaci (PDF)

Odkaz na aplikaci (není funkční, protože databázi využívá AI implementace)

Příloha B: Dokumentace aplikace vytvořené pomocí AI

Odkaz na dokumentaci (PDF)

Odkaz na aplikaci

Příloha C: Dokumentace aplikace vytvořené v Bubble

Odkaz na dokumentaci (PDF)

Odkaz na aplikaci