Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Vývoj webových aplikací pomocí umělé inteligence

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Specializace: Aplikovaná informatika

Autor: Josef Kuník

Vedoucí bakalářské práce: Ing. Richard Antonín Novák, Ph.D.

Praha, květen 2025

Poděkování

Rád bych vyjádřil vděk Ing. Richardu Antonínu Novákovi, Ph.D., za jeho cenné rady a odbornou pomoc při vedení mé práce. Děkuji také Martinu Kučerovi, oponentovi, za jeho podnětné připomínky a inspirativní nápady. Na závěr bych chtěl poděkovat své rodině a přátelům, kteří mě podporovali po celou dobu. Vaše přítomnost a pomoc mi poskytovaly sílu a motivaci k dokončení této práce.

Abstrakt

Bakalářská práce se zaměřuje na zkoumání aplikace umělé inteligence (AI) při vývoji webových aplikací, s důrazem na její vliv na efektivitu, časovou úsporu a kvalitu vývoje, především u menších a středních projektů. Cílem práce bylo analyzovat a porovnat tradiční metody vývoje webových aplikací s přístupy, které využívají generování kódu pomocí AI. Práce se soustředí na porovnání výhod a nevýhod AI generovaného kódu a ručně napsaného kódu, přičemž se hodnotí faktory jako časová náročnost, přístupnost webu, výkonnost a kvalita kódu.

Pro získání relevantních závěrů byly provedeny experimenty, které porovnávaly jednotlivé přístupy při vývoji webových aplikací, a to jak z hlediska efektivity, tak z pohledu kontroly kvality. Výsledky ukázaly, že využívání AI výrazně zrychluje vývojové procesy a usnadňuje automatizaci úkolů, jako je generování kódu nebo strukturování aplikace. Na druhou stranu se ukázalo, že AI přístup, ačkoliv je efektivní pro jednoduché úkoly, může při složitějších projektech vyžadovat další optimalizace a lidský zásah.

Tato práce přináší doporučení pro efektivní nasazení AI při vývoji webových aplikací, zejména v kontextu malých a středních projektů, a upozorňuje na její omezení v komplexních aplikacích, kde je nutná vyšší úroveň přizpůsobení.

Klíčová slova

efektivita, generativní umělá inteligence, optimalizace, umělá inteligence (AI), vývoj webu

Abstract

This bachelor's thesis focuses on exploring the application of artificial intelligence (AI) in web application development, emphasizing its impact on efficiency, time-saving, and quality, particularly for small and medium-sized projects. The aim of the thesis was to analyze and compare traditional methods of web application development with approaches that utilize code generation through AI. The work focuses on comparing the advantages and disadvantages of AI-generated code and manually written code, evaluating factors such as time consumption, web accessibility, performance and code quality.

To obtain relevant conclusions, experiments were conducted that compared the different approaches to web application development, considering both efficiency and quality control aspects. The results showed that the use of AI significantly accelerates development processes and facilitates the automation of tasks such as code generation and application structuring. However, it also became clear that while the AI approach is effective for simple tasks, it may require additional optimization and human intervention in more complex projects.

This thesis provides recommendations for the effective deployment of AI in web application development, particularly in the context of small and medium-sized projects, and highlights its limitations in complex applications, where a higher level of customization.

Keywords

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

Obsah

Úvod....10
1 Vývoj webových aplikací....12
1.1 Modely životního cyklu ve vývoji softwaru....12
1.1.1 Scrum....12
1.1.2 Vodopádový model....13
1.1.3 Etapy vodopádového modelu....13
1.2 Webové technologie....15
1.2.1 HTML....16
1.2.2 CSS....16
1.2.3 JavaScript....16
1.2.4 Framework....17
1.2.5 UI/UX design....18
1.2.6 No-code a Low-code technologie....19
2 Umělá Inteligence (AI)....20
2.1 Historie umělé inteligence....20
2.2 Definice a principy AI....22
2.3 Typy umělé inteligence....23
2.3.1 Strojové učení....23
2.4 Generativní umělá inteligence....25
2.4.1 Hluboké učení....25
2.5 Umělá inteligence ve webovém vývoji....26
2.5.1 Testování a ladění....26
2.5.2 Generování kódu....26
2.5.3 Hrozby....27
2.6 Přehled AI nástrojů pro webový vývoj....27
2.6.1 ChatGPT....27
2.6.2 Microsoft Copilot....27
2.6.3 GitHub Copilot....28
3 Metodika....29
3.1 Multikriteriální analýza metod (Saatyho metoda)....29
3.1.1 Hodnocená kritéria....29
3.1.2 Váhy kritérií....30
4 Praktická část....31

4.1 Nástroje využité v praktické části 31
4.1.1 Vodopádový model 31
4.1.2 Figma 31
4.1.3 Zvolené webové technologie 32
4.1.4 ChatGPT 32
4.2 Struktura a návrh webové aplikace 32
4.3 Tvorba veřejné části 33
4.3.1 Popis a účel 33
4.3.2 Tvoření webu tradičním přístupem 33
4.3.3 Tvoření webu pomocí AI 35
4.3.4 Porovnání přístupů 36
4.4 Tvorba zákaznického portálu 38
4.4.1 Popis a účel 38
4.4.2 Tvoření webu tradičním způsobem 38
4.4.3 Tvoření webu pomocí AI 42
4.4.4 Porovnání přístupů 45
4.5 Porovnání metod pomocí Saatyho metody 47
4.5.1 Srovnání přístupu při tvorbě veřejné části 47
4.5.2 Srovnání přístupu při tvorbě zákaznického portálu 48
4.5.3 Výsledky 49
Závěr 51
Použitá literatura 53

Seznam obrázků

Obrázek 1: Diagram procesu SCRUM modelu (Schwaber and Sutherland, 2017).....13
Obrázek 2: Lidský neuron a MCP neuron (Guy-Evans, 2024).....20
Obrázek 3: Turingův test (andreaminini.com).....22
Obrázek 4: Úvodní obrazovka (Autorské zpracování).....33
Obrázek 5: Úvodní stránka zákaznického portálu (Autorské zpracování).....38
Obrázek 6: Kalkulačka přijatých tekutin (Autorské zpracování).....40
Obrázek 7: Inicializace autorizace uživatele (Autorské zpracování).....42
Obrázek 8: Funkce načtení tekutin z databáze (Autorské zpracování).....44

Seznam tabulek

Tabulka 1: Saatyho škála (Autorské zpracování) 30
Tabulka 2: Párové srovnání a výpočet vah kritérií (Autorské zpracování) 30
Tabulka 3: Naměřené hodnoty výkonnosti (Autorské zpracování) 37
Tabulka 4: Naměřené hodnoty přístupnosti (Autorské zpracování) 37
Tabulka 5: Naměřené hodnoty výkonnosti (Autorské zpracování) 45
Tabulka 6: Naměřené hodnoty přístupnosti (Autorské zpracování) 46
Tabulka 7: Celkové hodnocení kritérií (Autorské zpracování) 48
Tabulka 8: Celkové hodnocení kritérií (Autorské zpracování) 49

Seznam zkratek

AI	umělá inteligence
AJAX	asynchronní JavaScript a XML
AHP	analytický hierarchický proces
CSS	kaskádové styly
DOM	dokumentový objektový model
GENAI	generativní umělá inteligence
HLD	vysoká úroveň designu
HTML	značkovací jazyk pro tvorbu webových stránek
JS	JavaScript
LCNC	nízkódové/ne-kódové platformy
LLD	nízká úroveň designu
LLM	velký jazykový model
SDLC	životní cyklus vývoje softwaru
SPA	jednostránková webová aplikace
UI	uživatelské rozhraní
UX	uživatelská zkušenost
XML	rozšiřitelný značkovací jazyk

Úvod

Vývoj webových aplikací se stále posouvá a přizpůsobuje novým technologiím, které usnadňují práci vývojářů a umožňují vytvářet aplikace s vyšší efektivitou a kvalitou. Jedním z nejnovějších trendů, který ovlivňuje tuto oblast, je využití umělé inteligence (AI). AI přináší nové nástroje a metody, které umožňují automatizovat různé aspekty vývoje aplikací, od generování kódu po optimalizaci uživatelského zážitku. Tato bakalářská práce se zaměřuje na zkoumání, jakým způsobem může AI podpořit vývoj webových aplikací a jaké výhody a omezení s sebou tento přístup přináší.

Struktura práce je rozdělena do teoretické a praktické části. Teoretická část se zaměřuje na vývoj webových aplikací, historický přehled vývoje webových technologií a roli umělé inteligence v současném vývoji webových nástrojů. Dále se práce věnuje klíčovým technologiím a nástrojům, které AI nabízí pro vývoj aplikací, a to především v oblasti automatizace generování kódu, optimalizace výkonu a přístupnosti aplikací.

Praktická část práce zahrnuje komparativní analýzu dvou přístupů k vývoji webových aplikací: tradičního vývoje a vývoje podporovaného AI. V rámci této části byly vytvořeny dvě verze aplikace – veřejná část webu a zákaznický portál. Obě verze byly vyvinuty jak tradičními metodami, tak s využitím generování kódu pomocí AI. Tyto verze byly následně porovnány z hlediska časové náročnosti, výkonnosti, kvality kódu a přístupnosti.

Závěr práce shrnuje hlavní poznatky z teoretické a praktické části, včetně porovnání tradičních metod a využití AI při vývoji webových aplikací. Práce zdůrazňuje přínosy AI, jako je úspora času a zrychlení vývoje, ale také upozorňuje na její omezení v oblasti kvality kódu a výkonnosti. Na základě výsledků práce jsou navržena doporučení pro efektivní využívání AI, zejména u menších a středních projektů, a diskutován její budoucí potenciál pro širší aplikace v oblasti webového vývoje.

Cíl práce

Hlavním cílem této bakalářské práce je zkoumat a porovnat, jak umělá inteligence ovlivňuje vývoj webových aplikací, přičemž se zaměřuje na menší a střední projekty. Práce se soustředí na analýzu rozdílů mezi tradičními metodami vývoje a generováním kódu pomocí AI, přičemž hodnotí faktory, jako je časová náročnost, výkonnost, kvalita kódu a přístupnost. Na základě těchto zjištění práce poskytuje doporučení pro efektivní implementaci AI do vývoje webových aplikací.

Metody práce

V této práci byly použity následující metody vědecké práce:

- Komparativní metoda – byla provedena analýza rozdílů mezi kódem generovaným umělou inteligencí a kódem napsaným tradičními metodami. Práce byla na porovnání silných a slabých stránek obou přístupů.

- Experimentální metoda – každý zkoumaný aspekt vývoje (např. generování kódu, strukturování aplikace) byl proveden dvěma způsoby – ručně a pomocí AI. Výsledky těchto experimentů byly následně porovnány, aby se zjistily rozdíly v efektivitě a kvalitě.
- Saatyho metoda – byla použita pro vícekriteriální hodnocení jednotlivých přístupů, zahrnující faktory jako časová náročnost, výkonnost a kvalita kódu.
- Měření času – pro každý přístup (tradiční a AI) byl měřen čas potřebný k vytvoření jednotlivých komponent webových aplikací, abychom zjistili efektivitu a rychlost vývoje.
- Měření výkonnosti kódu – byl proveden test výkonnosti kódu, zaměřující se na optimalizaci a efektivitu generovaného kódu, aby bylo možné posoudit jeho vliv na výkon aplikace.

1 Vývoj webových aplikací

1.1 Modely životního cyklu ve vývoji softwaru

Životní cyklus vývoje softwaru (SDLC) představuje strukturovaný postup, který sjednocuje proces návrhu, vývoje a nasazení softwaru. Začíná pochopením obchodních potřeb a jejich převodem do konkrétních funkcí a vlastností aplikace, které následně prochází implementací a provozem. Modely SDLC lze chápat jako užitečné nástroje pro efektivní řízení vývoje softwaru. Aby bylo možné vybrat nejvhodnější přístup, je důležité porozumět jednotlivým modelům, jejich využití v praxi a posoudit jejich silné a slabé stránky. Správná volba SDLC modelu zajišťuje, že bude projekt řízen v souladu s jeho požadavky a specifickým kontextem (Pargaonkar, 2023).

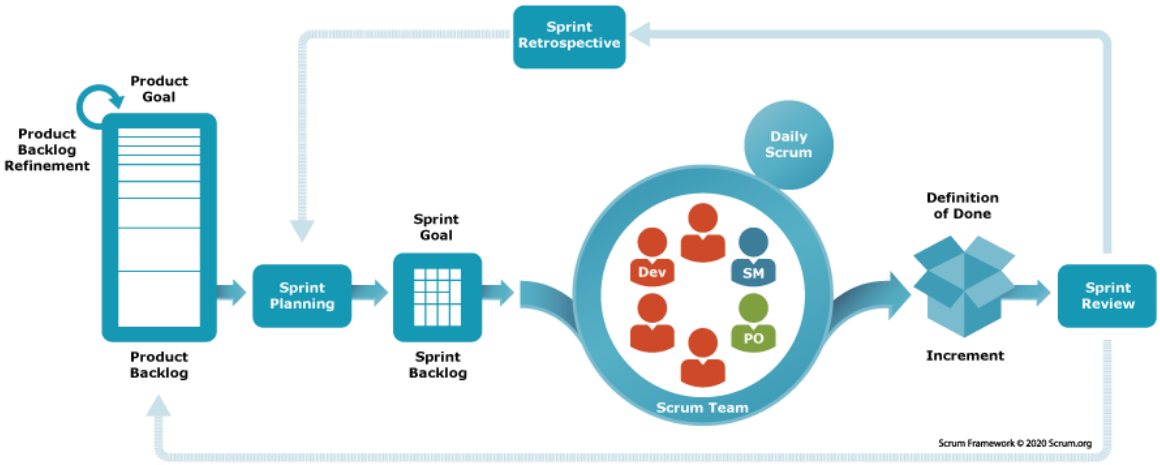
Modely SDLC lze rozdělit do tří hlavních skupin: lineární, iterativní a kombinované (spojující prvky obou přístupů). Lineární model je sekvenční, což znamená, že dokončení jedné fáze nevratně vede k zahájení další. Iterativní model naopak umožňuje návrat ke všem fázím během vývoje, čímž zajišťuje neustálé zlepšování softwaru po celou dobu jeho životního cyklu. Kombinovaný model uznává, že iterativní proces lze v určité fázi zastavit, pokud je dosaženo požadovaného výsledku (Ruparelia, 2010).

Ačkoli existuje mnoho různých SDLC modelů, tento text se zaměřuje na nejvýznamnější a nejrozšířenější přístupy, mezi které patří vodopádový model a SCRUM.

1.1.1 Scrum

Scrum je pravděpodobně nejpoužívanějším agilním modelem. Proces vývoje probíhá v krátkých iteracích zvaných sprinty, které trvají 2 až 4 týdny. Na začátku každého sprintu tým plánuje úkoly a cíle, které mají být splněny, a na konci probíhá retrospektiva sprintu, kde se tým zaměřuje na zlepšení procesu vývoje. Po zahájení sprintu nelze provádět žádné změny v jeho obsahu, což umožňuje soustředit se na splnění stanovených úkolů. Součástí je také sprint review, kde se prezentují dosažené výsledky zákazníkovi a získává se zpětná vazba pro další vývoj (Shiklo, 2019).

Zapojení zákazníka je klíčové – aktivně přispívá návrhy nových funkcí a pravidelně schvaluje dokončené části projektu. K řízení vývoje Scrum používá měřitelné nástroje, jako je burn-down graf, který ukazuje, jak rychle tým plní úkoly. Vývojáři odhaduji časovou náročnost úloh pomocí story points a rychlost dokončování úkolů se sleduje jako velocity, což pomáhá v plánování budoucích sprintů. Podobně jako všechny agilní modely Scrum je navržen pro menší týmy s různorodými dovednostmi, které pracují společně na jednom místě (Hron & Obwegeser, 2022).



Obrázek 1: Diagram procesu SCRUM modelu (Schwaber and Sutherland, 2017)

1.1.2 Vodopádový model

Vodopádový model představuje striktně lineární přístup k vývoji softwaru, kde se jednotlivé fáze dokončují v přesně daném pořadí, aniž by bylo možné se vracet zpět k předchozím krokům pro úpravy požadavků. Proces se pohybuje shora dolů, podobně jako tok vody ve vodopádu, což znamená, že každá fáze musí být dokončena dříve, než začne další. Tento model je jedním z nejstarších a nejznámějších přístupů k softwarovému vývoji (Pargaonkar, 2023).

Ačkoli vodopádový model postupně ustoupil do pozadí ve prospěch agilních metodik, stále může být vhodnou volbou pro menší projekty, kde jsou požadavky jasně definované a neměnné. Typickým příkladem jeho využití může být vývoj webových stránek pro malé firmy, kde je rozsah projektu omezený a není potřeba častých úprav během vývoje (Shiklo, 2019).

1.1.3 Etapy vodopádového modelu

Specifikace a analýza požadavků

Tato fáze detailně popisuje, jak by měl vyvíjený software fungovat. Cílem je jasně definovat funkční i nefunkční požadavky. Funkční požadavky se zaměřují na konkrétní chování softwaru a jsou často popsány pomocí use casů, které znázorňují, jak budou uživatelé se systémem pracovat. Mezi hlavní aspekty patří účel a rozsah softwaru, jeho funkcionality, požadavky na uživatelské rozhraní a databázi (Senarath, 2021).

Nefunkční požadavky se soustředí na technické vlastnosti a omezení systému, jako je spolehlivost, škálovatelnost, testovatelnost nebo výkon, a určují, jak bude software navržen a provozován, aniž by specifikovaly konkrétní uživatelské interakce (Senarath, 2021).

Návrh

Během fáze návrhu softwaru tým definuje technické požadavky projektu, včetně výběru hardwaru, programovacích jazyků, testovacích metod a uživatelského rozhraní. Tato fáze je klíčová zejména ve vodopádovém modelu, protože zajišťuje, že výsledný software bude splňovat požadované funkční a výkonnostní standardy (Laoyan, 2024).

Návrhová fáze je rozdělena na dvě úrovně (Laoyan, 2024):

- High-Level Design (HLD) – tým vytváří základní strukturu systému, definuje klíčové komponenty a způsob, jakým bude software zpracovávat a ukládat data.
- Low-Level Design (LLD) – zaměřuje se na detailní návrh jednotlivých modulů a jejich propojení, což odpovídá konkrétní implementaci softwarových funkcí. Pokud lze HLD přirovnat ke kostře projektu, pak LLD představuje jeho orgány, které dávají systému funkčnost.

Implementace

Tato fáze vývoje zahrnuje praktickou realizaci obchodních a technických požadavků tím, že se návrh softwaru převede do funkční podoby. To znamená psaní kódu, jeho kompilaci a sestavení aplikace, stejně jako vytvoření databází a dalších souvisejících komponent. Jinými slovy, jde o přeměnu specifikací a návrhových plánů na reálný software připravený k nasazení a provozu (Senarath, 2021).

Testování

Během této fáze se kontroluje celková kvalita výsledného produktu. Dobře provedené testování zajišťuje vyšší spokojenost zákazníků, nižší náklady na údržbu a spolehlivější výsledky. Testování zahrnuje tři hlavní úrovně (Model, 2015):

- Alpha testing – provádí vývojáři k internímu ověření systému.
- Beta testing – testování provádí skupina zákazníků, kteří pomáhají identifikovat reálné problémy v provozu.
- Akceptační testování – zákazník posuzuje software a rozhoduje, zda jej schválí k použití nebo zamítne.

Kromě těchto testů definuje Saravanos a Curing (2023) i další důležité testy zaměřené na konkrétní aspekty systému:

- Unit Testing (Testování jednotek) – testování jednotlivých komponent systému, prováděné vývojáři, zajišťující správnou funkčnost jednotlivých částí kódu.
- Performance Testing (Testování výkonu) – zkoumá, jak systém zvládá vysoké zatížení a zda funguje efektivně i při vysokém počtu uživatelů.
- Integration Testing (Testování integrace) – testuje, jak jednotlivé moduly spolupracují a předávají si data.
- Security Testing (Testování bezpečnosti) – identifikuje bezpečnostní slabiny, jako jsou zranitelnosti proti útokům.

Nasazení

Po úspěšném testování následuje nasazení systému do produkčního prostředí. Tento proces zahrnuje několik klíčových kroků, které definuje Laoyan (2024):

- **Release Management:**
  - Plánování, koordinace a implementace nových verzí softwaru.
  - Testování před vydáním a zajištění hladkého přechodu mezi verzemi.
  - Koordinace nasazení verzí bez výpadků a narušení provozu.
- **DevOps:**
  - Integrace vývoje a operací pro rychlé a stabilní nasazení.
  - Automatizace procesu nasazení a používání nástrojů pro CI (kontinuální integrace) a CD (kontinuální nasazování).
  - Sledování výkonu systému a sběr zpětné vazby od uživatelů pro detekci problémů.
- **Post-deploy monitoring a zálohování:**
  - Monitoring výkonu systému po nasazení pomocí APM a log management nástrojů.
  - Rychlá identifikace problémů a optimalizace výkonu.
  - Zajištění zálohování a obnovy dat pro minimalizaci rizika výpadků.

Údržba

Tato fáze zahrnuje úpravy a vylepšení softwaru po jeho nasazení, s cílem optimalizovat výstupy, odstranit chyby a zvýšit celkový výkon a kvalitu. Přibližně 60 % celkových nákladů na vývoj softwaru je vynaloženo na jeho následnou údržbu a zlepšování. Rozlišujeme tři základní typy údržby (Model, 2015):

- Korektivní údržba – zaměřuje se na opravu chyb, které nebyly objeveny během vývoje.
- Perfekcionistická údržba – slouží ke zlepšení funkcí softwaru na základě zpětné vazby zákazníků.
- Adaptivní údržba – zajišťuje kompatibilitu softwaru s novými platformami a operačními systémy.

1.2 Webové technologie

Moderní webové aplikace se dělí do dvou hlavních částí: frontend a backend. Každá z těchto částí má své specifické úkoly a vyžaduje odlišné technologie, přesto spolu úzce spolupracují na vytvoření kvalitního softwaru.

Frontend vývojáři vytvářejí uživatelské rozhraní a vizuální prvky, které vidí a používají uživatelé. Na druhé straně backend vývojáři se starají o serverovou logiku, databáze a výkon systému (Jalolov, 2024).

1.2.1 HTML

HTML (HyperText Markup Language) je značkovací jazyk používaný pro tvorbu a strukturování webových stránek. Pomocí HTML prvků, jako jsou tagy a atributy, lze definovat sekce, odstavce a odkazy, které tvoří obsah webu (Astari, 2022).

HTML používá vnořené tagy, což znamená, že celá stránka musí být obsažena v <html></html>. Uvnitř se nachází sekce <head></head>, kde se definuje například titulek prohlížeče, a sekce <body></body>, která obsahuje viditelný obsah webu. Tagy mohou mít atributy pro formátování textu, změnu barvy nebo stylu. HTML umožňuje nejen formátování textu, ale i vkládání obrázků a dalších prvků (Telg et al., 2015).

1.2.2 CSS

Dříve se k formátování HTML dokumentů používaly značky a jejich atributy, například <font>, <b> nebo , k rozvržení stránek se využívaly tabulky. Postupem času však s rostoucí složitostí webových stránek vznikaly problémy – objem formátovacích značek často převyšoval samotný obsah, což vedlo ke zpomalení načítání stránek a k vyšším nárokům na servery i jejich provozní náklady. Používání tabulek pro formátování navíc komplikovalo přístupnost stránek pro jiná zařízení než běžné prohlížeče (Cyroň, 2006).

CSS umožňuje oddělit vizuální styl od samotného HTML kódu, což znamená, že veškeré styly se ukládají do samostatného souboru s názvem stylopis (stylesheet) a následně se propojí s HTML stránkou. Díky tomu je HTML kód přehlednější a snáze udržovatelný. Další výhodou CSS je, že není nutné opakovaně nastavovat vzhled jednotlivých prvků, což zjednodušuje práci, zkracuje kód a snižuje riziko chyb. Navíc CSS umožňuje aplikovat různé styly na jedinou HTML stránku, čímž poskytuje široké možnosti přizpůsobení. V dnešní době se tak oddělené stylování stává nezbytným standardem při tvorbě moderních webových stránek (Domantas, 2023).

Stylopis obsahuje pravidla formátování, která se skládají alespoň ze tří částí(Cyroň, 2006):

- Selektor, který určuje, na které prvky se pravidlo vztahuje.
- Vlastnost, která definuje konkrétní aspekt vzhledu.
- Hodnota, která specifikuje konkrétní nastavení vlastnosti.

Například pravidlo níže způsobí, že všechny odstavce (<p>) budou zobrazeny červenou barvou:

```
p { color: red; }
```

1.2.3 JavaScript

JavaScript je klíčový programovací jazyk pro webové aplikace. Většina moderních webů a prohlížečů, včetně těch na počítačích, mobilních telefonech a herních konzolích, obsahuje JavaScriptové interprety, což z něj činí jeden z nejpoužívanějších jazyků vůbec. Spolu s

HTML (pro strukturu obsahu) a CSS (pro vizuální styl) tvoří základní pilíře webového vývoje (Jordana, 2024).

JavaScript je vysokoúrovňový, dynamický jazyk, který podporuje objektově orientované i funkcionální programování. Jeho syntaxe částečně vychází z Javy, ale zároveň využívá prvky ze Scheme a Self, například prototypovou dědičnost. Pro jeho zvládnutí není nutná předchozí znalost těchto jazyků (Flanagan, 2011).

Přestože název „JavaScript" naznačuje souvislost s Javou, ve skutečnosti jde o zcela odlišný jazyk. JavaScript se v průběhu let rozvinul v výkonný a efektivní nástroj pro široké spektrum aplikací, přičemž novější verze přinášejí pokročilé funkce pro velké softwarové projekty (Flanagan, 2011).

1.2.4 Framework

Framework je soubor nástrojů a knihoven, které poskytují předem napsaný kód pro běžné úkoly ve vývoji aplikací. Umožňuje vývojářům soustředit se na specifické funkce aplikace a neřešit základní problémy, jako je správa souborů nebo ovládání událostí. Frameworky často definují strukturální rámec aplikace, který usměrňuje práci vývojářů a umožňuje snadné přidávání nových funkcí (Shukla, 2023).

JavaScript frameworky mají zásadní význam pro moderní webový vývoj. Poskytují standardizovanou strukturu, která usnadňuje vývoj webových aplikací a zajišťuje, že kód bude čitelný, udržovatelný a dobře organizovaný. Použití frameworků znamená, že vývojáři nemusí začínat kódování aplikace od nuly, ale mohou využít hotové komponenty a knihovny, které zjednodušují a urychlují celý proces (Dubey et al., 2024).

Mezi hlavní výhody používání JavaScript frameworků podle Shukly (2023) patří:

- Modularita – umožňuje vývoj aplikací pomocí znovupoužitelných komponent.
- Efektivita – zrychluje vývoj díky opakovanému použití kódu a integraci běžně používaných funkcí.
- Údržba a škálovatelnost – snižuje složitost kódu a usnadňuje budoucí úpravy nebo rozšíření aplikace.

Dubey et al. (2024) jmenují jako nejznámější JavaScript Frameworky:

- React – knihovna pro vytváření interaktivních uživatelských rozhraní, vyvinutá Facebookem. Využívá virtuální DOM, což zajišťuje efektivní renderování aplikace. React je známý pro svůj komponentově orientovaný přístup, který umožňuje opakované použití kódu.
- Angular – komplexní front-endový framework vyvinutý Googlem. Umožňuje vytváření jednostránkových aplikací (SPA) a zahrnuje nástroje pro routing, dependency injection a testování. Angular je známý svou silnou strukturou a vhodností pro velké projekty.
- Vue.js – progresivní framework, který vyniká svou jednoduchostí a flexibilitou. Je vhodný pro malé i velké projekty a umožňuje postupné přidávání funkcí, což je ideální pro vývoj aplikací s nízkou až střední komplexitou.

- Node.js – runtime prostředíro JavaScript, které umožňuje použití JavaScriptu pro backend vývoj. Postaveno na V8 engine, Node.js je skvělé pro real-time aplikace, které vyžadují vysokou rychlost a asynchronní operace.
- jQuery – knihovna, která zjednodušuje práci s DOM a AJAX. Umožňuje rychlou manipulaci s HTML prvky a je stále oblíbená pro menší projekty nebo pro starší webové aplikace.

1.2.5 UI/UX design

UI/UX design je klíčovým procesem při vytváření kvalitních a efektivních digitálních produktů. UI (User Interface) design se zaměřuje na vizuální a interaktivní prvky aplikace, jako jsou tlačítka, menu, ikony a celkové rozvržení, které ovlivňují uživatelskou interakci. Na druhé straně UX (User Experience) design se soustředí na celkový zážitek uživatele při používání produktu, zahrnující jeho pohodlí, intuitivnost a efektivitu. Dobrý UI/UX design má zásadní vliv na spokojenost uživatelů, zvyšuje použitelnost a může přispět k úspěchu produktu na trhu (Calonaci, 2021).

Sharma a Tiwari (2021) zmiňují několik oblíbených nástrojů pro UI/UX design, které pomáhají návrhářům vytvářet efektivní a atraktivní uživatelská rozhraní a zajišťovat skvělé uživatelské zkušenosti:

- Figma
- Adobe XD
- Sketch
- InVision
- Balsamiq

Figma

Figma je moderní nástroj pro návrh uživatelských rozhraní (UI), který umožňuje kolaborativní práci v reálném čase. Figma je cloudová platforma, která nabízí širokou škálu funkcí pro tvorbu interaktivních prototypů, vizuálních návrhů, drátových modelů (wireframů) a dalších prvků designu. Tento nástroj je oblíbený mezi designéry a vývojáři díky své flexibilitě, intuitivnímu rozhraní a možnosti spolupráce (Staiano, 2022).

Figma umožňuje snadnou integraci s dalšími nástroji a poskytuje přehledné pracovní prostředí pro tvorbu a správu designových projektů. Uživatelé mohou navrhovat webové stránky, mobilní aplikace a komplexní interaktivní rozhraní, přičemž mají k dispozici širokou knihovnu komponent a nástrojů pro prototypování. Díky cloudové synchronizaci mohou více uživatelů pracovat na stejném projektu současně, což zjednodušuje komunikaci mezi členy týmu (Calonaci, 2021).

Další výhodou Figmy je její schopnost exportovat návrhy do formátů vhodných pro implementaci, což vývojářům umožňuje přímé převzetí designu do kódu. Tento nástroj je obzvláště ceněný pro designové procesy v agilních týmech, kde je rychlé iterování návrhů a snadná kooperace klíčová (Staiano, 2022).

1.2.6 No-code a Low-code technologie

Low-Code/No-Code (LCNC) platformy jsou nástroje pro vývoj softwaru, které umožňují uživatelům vytvářet aplikace bez nutnosti psaní rozsáhlého kódu. Tyto platformy jsou navrženy tak, aby umožnily i neprogramátorům (např. podnikatelům, analytikům nebo marketérům) vytvářet aplikace, a to pomocí vizuálního rozhraní a drag-and-drop nástrojů. Tento přístup umožňuje rychlý vývoj aplikací, což zkracuje dobu potřebnou pro tvorbu a nasazení produktů (Picek, 2023).

Low-Code platformy obvykle umožňují uživatelům psát kód pro specifické funkce, pokud je to potřeba, ale zároveň poskytují širokou škálu předpřipravených komponent a šablon, které urychlují vývoj. No-Code platformy nevyžadují žádné programování a jsou plně zaměřeny na vizuální návrh aplikací, kde uživatelé pouze skládají předdefinované bloky (Bhattacharyya & Kumar, 2023).

Výhody a omezení LCNC

Výhody LCNC platforem spočívají především v jejich rychlosti vývoje a nízkých nákladech. Díky vizuálnímu rozhraní a předpřipraveným komponentám mohou uživatelé vytvářet aplikace mnohem rychleji než při tradičním programování. To umožňuje i neprogramátorům snadno vytvářet webové stránky nebo aplikace, což zjednodušuje celý vývojový proces. LCNC platformy také snižují náklady na vývoj, protože není potřeba odborných programátorských dovedností, což umožňuje firmám demokratizovat proces vývoje a podpořit inovace i bez rozsáhlých technických týmů (Bhattacharyya & Kumar, 2023).

Na druhou stranu mají LCNC platformy i svá omezení. I když jsou ideální pro jednoduché až středně složité aplikace, mohou být méně vhodné pro komplexní projekty s pokročilými technickými požadavky. Omezují flexibilitu vývojářů, protože některé platformy neposkytují úplnou kontrolu nad kódem, což může být problém pro aplikace vyžadující specifické funkce nebo přizpůsobení. Dalším omezením je údržba aplikací vytvořených na těchto platformách, protože pokud se v budoucnu objeví potřeba přidat složitější funkce nebo provést velké změny, může být složité přejít na tradiční vývojové prostředí (Rokis & Kirikova, 2022).

Nejznámější LCNC platformy

Picek (2023) zmiňuje několik světoznámých LCNC platforem. Mezi nejznámější řadí:

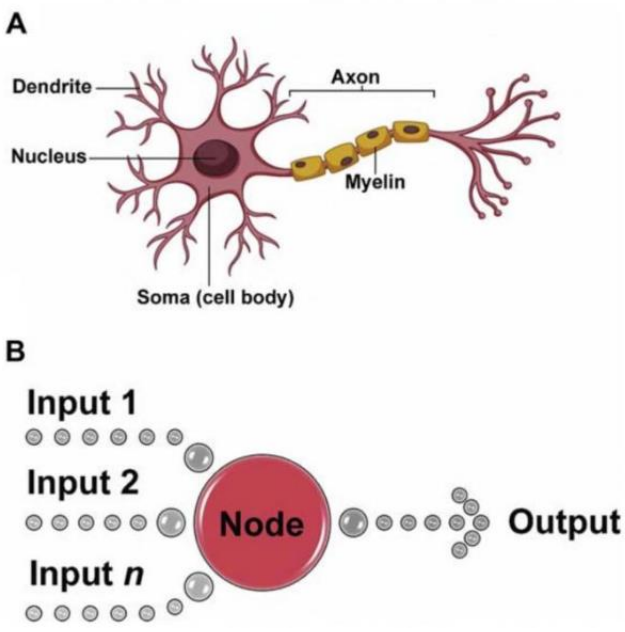
- Bubble – jedna z nejpopulárnějších No-Code platforem, která umožňuje vytvářet webové aplikace s komplexní logikou bez psaní kódu.
- OutSystems – low-code platforma určená pro vývoj podnikových aplikací. Umožňuje vysoce přizpůsobené aplikace s nízkým množstvím kódu.
- Appgyver – no-code platforma, která je zaměřena na vývoj mobilních a webových aplikací s jednoduchým drag-and-drop rozhraním.
- Webflow – no-code platforma pro vývoj vizuálně atraktivních webových stránek, která umožňuje i pokročilou animaci a interaktivitu.

2 Umělá Inteligence (AI)

Umělá inteligence (AI) často evokuje představy o humanoidních robotech známých ze sci-fi literatury, což může být jedním z důvodů, proč již desítky let fascinuje a rozděluje veřejnost, vědce i akademiky. Ve skutečnosti však AI nezahrnuje pouze robotiku, ale svým dosahem zasahuje do oblastí informatiky, psychologie, filozofie a dalších vědních disciplín. Inteligence, definována jako obecná mentální schopnost k usuzování, řešení problémů a učení se (Colom et al., 2010), se zde kombinuje s přívlastkem „umělá“, což evokuje něco vytvořeného člověkem, nepřirozeného či technického původu. Přesto toto spojení nachází logické opodstatnění. „Umělá“ část v tomto kontextu odkazuje na počítače, stroje či software, zatímco „inteligence“ – ač bez přesné definice – zahrnuje dvě klíčové schopnosti, které sdílí lidé i některé technologie: schopnost učit se a řešit problémy (Colom et al., 2010; Manning, 2020). I když by bylo možné diskutovat o tom, zda stroje skutečně dokážou usuzovat, přizpůsobovat se či aktivně ovlivňovat své prostředí, právě učení a řešení problémů tvoří hlavní průsečíky jejich definice.

2.1 Historie umělé inteligence

První modely umělé inteligence (AI) se zaměřovaly na napodobení činnosti neuronů. McCulloch a Pitts (1943) vytvořili MCP neuron, který pracoval s Booleovskými hodnotami (pravda/nepravda), ale neuměl se učit ani přizpůsobovat nové informace. Tento model byl sice užitečný pro základní logické operace, ale jeho omezené možnosti vedly k nutnosti hledat složitější přístupy (Muthukrishnan et al., 2020).



Obrázek 2: Lidský neuron a MCP neuron (Guy-Evans, 2024)

Ve stejné době Alan Turing navrhl The Bombe, stroj na dešifrování německé šifry Enigma, čímž významně přispěl k ukončení druhé světové války. Tento úspěch jej přivedl k zamyšlení nad tím, zda mohou stroje napodobovat lidské myšlení. V roce 1950 publikoval článek Computing Machinery and Intelligence, ve kterém formuloval Turingův test. Tento test měl ověřit, zda je možné, aby stroj komunikoval takovým způsobem, že by jej člověk nerozpoznal od skutečné osoby (Haenlein & Kaplan, 2019).

V roce 1956 se konala konference Dartmouth, kde John McCarthy poprvé použil termín "umělá inteligence". Na této konferenci byl představen Logic Theorist, program vyvinutý Allenem Newellem, Cliffem Shawem a Herbertem Simonem, který dokázal simulovat lidské myšlení při řešení matematických problémů. Tento program je považován za první skutečnou AI aplikaci (Anyoha, 2017).

Další významný průlom nastal v roce 1966, kdy Joseph Weizenbaum vyvinul chatbot ELIZA. Tento systém simuloval lidskou konverzaci pomocí jednoduchých pravidel založených na klíčových slovech. I když nedokázal skutečně porozumět významu vět, jeho schopnost vést dialog vzbudila velký zájem o vývoj chatbotů. ELIZA ukázala, že i jednoduchá AI dokáže vyvolat iluzi inteligentní interakce (Al-Amin et al., 2024).

V 70. letech se AI potýkala s problémy. Rostoucí očekávání vedla k předpokladu, že během několika let budou vyvinuty plně inteligentní stroje. V praxi však chyběl dostatečný výpočetní výkon a efektivní algoritmy, což vedlo ke zklamání. Výzkum byl kritizován za vysoké náklady a nízké výsledky, došlo tedy ke snížení financování. Tento útlum, známý jako „zima AI“, trval téměř dvě desetiletí (Haenlein & Kaplan, 2019).

V polovině 90. let došlo k obnovení zájmu o AI, a to především díky technologickému pokroku. Moorův zákon a rostoucí výkon počítačů umožnily efektivnější trénování neuronových sítí. Významný průlom nastal v roce 1997, kdy IBM Deep Blue porazil šachového velmistra Garryho Kasparova. Tento úspěch ukázal, že AI dokáže překonat lidské schopnosti i v oblastech, které vyžadují strategické myšlení (Muthukrishnan et al., 2020).

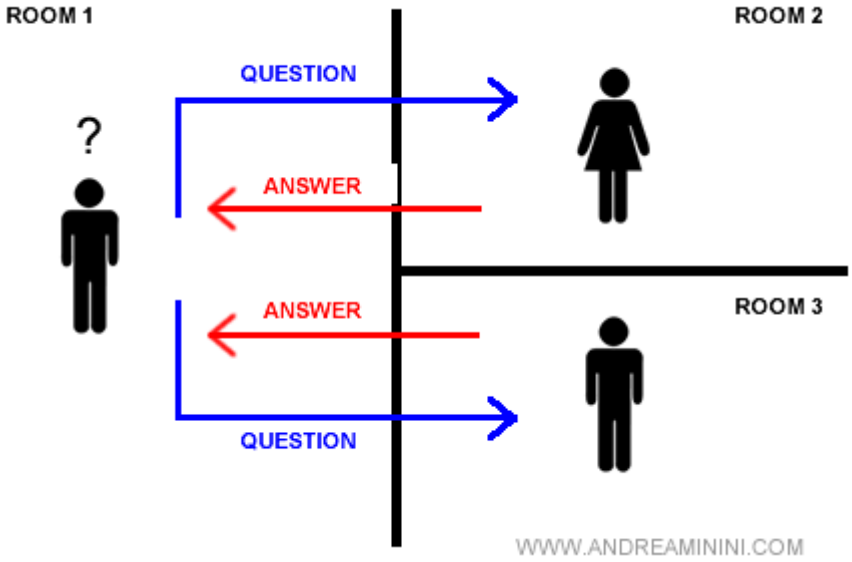
Další velký posun přinesl rok 2015, kdy AI poprvé porazila člověka ve hře Go. Program AlphaGo, vyvinutý Google DeepMind, využíval Deep Learning a zesílené učení, což mu umožnilo analyzovat obrovské množství herních strategií a přizpůsobovat se novým situacím. Go je mnohem složitější než šachy a dlouho se předpokládalo, že počítače v této hře nebudou schopny překonat lidské hráče. Výhra AlphaGo znamenala revoluci v AI, protože ukázala sílu hlubokých neuronových sítí.

Dnes jsou neuronové sítě a Deep Learning klíčovou součástí moderní AI. Tyto technologie umožňují pokročilé rozpoznávání obrazu, řeči a autonomní rozhodování, což vedlo k jejich širokému využití v průmyslu. AI je dnes základem pro autonomní vozidla, hlasové asistenty, analýzu dat a mnoho dalších aplikací. Jsme svědky největšího rozmachu AI v historii, kdy se umělá inteligence stává nedílnou součástí každodenního života (Haenlein & Kaplan, 2019).

2.2 Definice a principy AI

Ačkoli AI v současném výzkumu a technologickém pokroku stále roste na významu, zůstává přesto těžko definovatelná. I když existovaly pokusy o její jasnou definici, podobně jako u lidské inteligence, doposud neexistuje všeobecně uznávaná definice umělé inteligence. Tato nejasnost je ještě umocněna neustálým vývojem nových technologií, teoretických přístupů a praktických aplikací.

Jedním z prvních teoretických vymezení umělé inteligence je Turingův test, navržený Alanem Turingem. Počítač testem projde, pokud lidský tazatel po položení několika písemných otázek, nedokáže rozlišit, zda odpovědi pocházejí od člověka, nebo od počítače. Tento test se zaměřuje na schopnost stroje zpracovávat přirozený jazyk, reprezentovat znalosti, uvažovat a učit se, což jsou klíčové schopnosti pro jeho úspěšné vykonání. Nicméně je otázkou, zda Turingův test není spíše zaměřen na chování a imitaci lidského jednání, než na skutečný projev inteligence (Graham et al., 2021).



Obrázek 3: Turingův test (andreaminini.com)

Lepší koncepci porovnání s lidmi definovala Elaine Rich(Rich, 1985): *„Umělá inteligence (AI) je studium toho, jak naučit počítače vykonávat činnosti, ve kterých jsou lidé lepší.“* Ačkoliv existuje spousta oblastí, ve kterých je počítač lepší, rozhodně tomu tak není ve všech oblastech. Jako například řešení problémů, porozumění lidskému jazyku, vnímání okolního světa a provádění vědeckého výzkumu. Jako další příklad může být situace, kdy při vstupu do neznámé místnosti člověk během chvíle rozpozná své okolí, dokáže se rychle rozhodovat a plánovat další kroky, což by pro autonomní roboty představovalo velmi složitý úkol. Z této definice tedy vyplývá, že právě takové schopnosti jsou cílem snah v oblasti umělé inteligence. Formulace Elaine Rich tak zůstává aktuální, protože její závěr zajišťuje, že jakmile AI dosáhne lidské úrovně nebo ji překoná, stane se tato oblast irelevantní a zaměří se na nové výzvy (Ertel, 2024).

2.3 Typy umělé inteligence

Umělá inteligence zahrnuje širokou škálu technologií a přístupů, které umožňují strojům vykonávat úkoly vyžadující lidskou inteligenci. Mezi nejvýznamnější patří symbolická AI, která se zaměřuje na logické operace a pravidla, a strojové učení, kde se modely učí na základě dat. Pokročilejší formou je hluboké učení, využívající neuronové sítě pro složité úkoly, jako je rozpoznávání obrazu či generování textu.

2.3.1 Strojové učení

Strojové učení (Machine Learning) se zabývá vytvářením algoritmů umožňujících systémům automaticky se učit z dat a zlepšovat své výstupy bez explicitního programování. Tento přístup se stal klíčovým díky rostoucímu množství dostupných dat, výpočetnímu výkonu a pokrokům v algoritmech. Strojové učení lze definovat jako studium počítačových algoritmů, které se zlepšují automaticky na základě zkušeností (Mitchell & Mitchell, 1997).

Základní princip strojového učení spočívá v tom, že algoritmy analyzují data, rozpoznávají v nich vzory a následně je využívají k predikci nebo rozhodování. Podle Russella a Norviga „strojové učení představuje jeden z klíčových přístupů k budování inteligentních systémů“ (Russell & Norvig, 2016). Různé typy učení se liší svým přístupem k datům:

- Učení s učitelem.
- Učení bez učitele.
- Posilované učení.

Učení s učitelem

K analogii učení s učitelem lze použít následující příklad: Učitel studentům vysvětlí, jak funguje sčítání, a ti si ho procvičují na různých příkladech. Jakmile jsou schopni správně spočítat i nové příklady, které dosud neviděli, znamená to, že látku skutečně pochopili. Například po vyřešení padesáti příkladů už mohou sčítání aplikovat na nekonečné množství dalších situací (Ertel, 2024).

Učení s učitelem (supervised learning) je metoda strojového učení, při níž se model učí na základě tréninkové sady obsahující vstupní data spárovaná se správnými výstupy. Hlavní funkcí je naučit model zobecnit vztah mezi vstupními a výstupními daty tak, aby byl schopen správně predikovat výstupy i pro nová, dosud neviděná data. Tento přístup se využívá zejména pro klasifikaci (přiřazování vstupů do předdefinovaných kategorií) a regresi (predikci spojitých hodnot). Cílem je minimalizovat chybu predikce a zajistit efektivní generalizaci modelu (Jiang et al., 2020).

Supervised learning se nejčastěji využívá pro klasifikaci, kdy model přiřazuje vstupy do předem definovaných kategorií (například detekce spamu v e-mailové komunikaci), a pro regresi, kde model predikuje spojité číselné hodnoty (například předpověď cen nemovitostí na základě různých parametrů) (Ertel, 2024).

Při tréninku modelu je klíčové zajistit správnou rovnováhu mezi přeučením (overfitting) a nedostatečným učením (underfitting). Přeučení nastává, když se model příliš přizpůsobí tréninkovým datům a nedokáže dobře zobecňovat na nová data, zatímco nedostatečné učení znamená, že model je příliš jednoduchý a nerozpoznává důležité vzory. Abychom těmto problémům předešli, využívá se křížová validace (cross-validation), která testuje model na různých částech dat, čímž pomáhá vybrat optimální nastavení a zajistit lepší schopnost modelu generalizovat (Jiang et al., 2020).

Učení bez učitele

Učení bez učitele (unsupervised learning) je metoda strojového učení, při níž model pracuje s neoznačenými daty, tedy bez předem daných správných výstupů. Hlavním cílem není predikce, ale hledání skrytých vzorů, struktur a vztahů v datech. Model analyzuje vstupní proměnné a snaží se v nich objevit souvislosti, které mohou pomoci s vizualizací dat, jejich zjednodušením nebo rozdělením do skupin (James et al., 2023).

Tento přístup se nejčastěji využívá pro shlukování (clustering), kde se model snaží rozdělit data do skupin podle podobnosti. Například při analýze zákaznického chování může model najít skupiny zákazníků s podobnými nákupními vzory, což pomáhá s cíleným marketingem. Další častou metodou je analýza hlavních komponent (PCA), která se používá ke zjednodušení složitých dat tím, že zachová co nejvíce důležitých informací, ale sníží počet proměnných. To usnadňuje vizualizaci a další analýzu dat (Brownlee, 2016).

Učení bez učitele se hodí tam, kde nemáme přesně definované výstupy, ale chceme pochopit strukturu dat, identifikovat vzory, detekovat anomálie (například podvodné transakce) nebo připravit data pro další modely (James et al., 2023).

Posilované učení

Posilované učení (Reinforcement Learning) je metoda strojového učení, kde se agent učí optimálnímu chování prostřednictvím interakce s prostředím. Místo toho, aby měl k dispozici pevně dané správné odpovědi jako při učení s učitelem, získává zpětnou vazbu ve formě odměn nebo trestů na základě svých akcí. Hlavním cílem je, aby se agent naučil takovou strategii rozhodování, která mu přinese co nejvyšší dlouhodobou odměnu (Wiering & Van Otterlo, 2012).

Agent na začátku nezná pravidla prostředí a učí se metodou pokus-omyl. Zkouší různé akce, sleduje jejich důsledky a postupně vylepšuje svůj přístup, aby dosáhl co nejlepších výsledků. Klíčovou výzvou je nalezení rovnováhy mezi průzkumem nových možností a využíváním již osvědčených strategií. Dalším problémem je, že některé akce mohou mít okamžitou odměnu, zatímco jiné přinášejí výhody až v dlouhodobém horizontu, což agent musí správně vyhodnotit (Ernst & Louette, 2024).

Posilované učení se využívá v různých oblastech, například při vývoji autonomních systémů, kde se algoritmy učí řídit vozidla, v herní umělé inteligenci, která se učí hrát hry jako šachy, v robotice pro optimalizaci pohybu robotů nebo ve finančním sektoru pro tvorbu investičních strategií (Wiering & Van Otterlo, 2012).

2.4 Generativní umělá inteligence

Generativní umělá inteligence (GAI) je oblast strojového učení, která umožňuje vytvářet nový obsah na základě vzorů naučených z existujících dat. Na rozdíl od tradiční AI, která se soustředí na analýzu a klasifikaci, generativní modely dokáží syntetizovat texty, obrázky, hudbu, videa nebo kód, čímž napodobují lidskou kreativitu a otevírají nové možnosti v oblasti automatizované tvorby obsahu. Tato technologie nejen usnadňuje a urychluje kreativní procesy, ale také rozšiřuje hranice inovace, protože umožňuje vytvářet originální materiály, které by jinak vyžadovaly lidskou invenci. Generativní AI tak nachází uplatnění ve výtvarném umění, literatuře, hudební tvorbě i filmové produkci, kde pomáhá s generováním nápadů, stylizací obsahu nebo dokonce kompletní tvorbou uměleckých děl. Její využití sahá i do průmyslového designu, vědeckého výzkumu a softwarového inženýrství, kde napomáhá optimalizaci procesů a urychluje inovace (Banh & Strobel, 2023).

2.4.1 Hluboké učení

Hluboké učení (Deep Learning) je oblast strojového učení založená na umělých neuronových sítích s mnoha vrstvami, které dokáží modelovat složité vztahy v datech. Tyto hluboké sítě se skládají z propojených neuronů, které zpracovávají informace prostřednictvím vážených spojení a nelineárních transformací. Proces učení spočívá v postupné úpravě vah pomocí algoritmů, jako je zpětná propagace, která minimalizuje chybu mezi predikovaným a skutečným výstupem. Hluboké učení se liší od klasických neuronových sítí svou schopností automaticky extrahovat hierarchické reprezentace dat. Nižší vrstvy se zaměřují na základní rysy (např. hrany v obrazech), zatímco vyšší vrstvy dokážou rozpoznávat složitější struktury, jako jsou objekty nebo vzory v datech (Schmidhuber, 2015).

Vše začíná inicializací parametrů, při kterém se neuronu v neuronové síti na začátku trénování přiřazují počáteční hodnoty vah. Váha určuje důležitost konkrétního vstupu, tedy jak moc ovlivní výstup neuronu – vyšší váha znamená silnější vliv vstupního signálu, zatímco nižší váha ho potlačuje. Správná inicializace je zásadní, protože pokud by byly váhy například příliš malé, signál v síti by se postupně zmenšoval a neuronová síť by se učila velmi pomalu nebo by se nenaučila nic. Naopak příliš velké váhy by mohly vést k nestabilním výpočtům a explozi gradientů, což by způsobilo špatnou konvergenci modelu. Během trénování se váhy postupně upravují pomocí optimalizačních algoritmů, aby se síť naučila správné vzory a vztahy mezi vstupy a výstupy (Razavi, 2021).

Proces zpětné propagace začíná tím, že síť provede výpočet dopředným směrem – vstupní data projdou všemi vrstvami až k výstupní vrstvě, kde se vytvoří předpověď. Následně se porovná s očekávaným výstupem a vypočítá se chyba modelu. Zpětná propagace poté šíří tuto chybu zpět skrze vrstvy sítě, přičemž upravuje váhy jednotlivých neuronů tak, aby snížila celkovou chybu. Tento proces probíhá pomocí gradientního sestupu, což je optimalizační metoda, která hledá správné nastavení vah tak, aby model postupně zlepšoval své výsledky. Tímto iterativním učením se síť přizpůsobuje datům a zlepšuje svou schopnost přesněji předpovídat výsledky (Janiesch et al., 2021).

2.5 Umělá inteligence ve webovém vývoji

Umělá inteligence hraje klíčovou roli v modernizaci vývoje webových stránek. Díky AI lze automatizovat rutinní úkoly, například generování kódu, testování a nasazování aplikací, což vede k rychlejšímu vývoji a vyšší efektivitě. Automatizace zároveň zlepšuje přesnost a zajišťuje jednotnost v programovacích standardech. AI také přispívá ke zlepšení uživatelského prostředí tím, že umožňuje personalizovaný obsah, dynamická rozhraní a inteligentní chatboty, což vede k většímu zapojení a spokojenosti uživatelů. Díky těmto možnostem mohou webové stránky lépe reagovat na potřeby uživatelů, být interaktivnější a přizpůsobitelnější. Tento vývoj představuje zásadní posun směrem k inteligentnějším a efektivnějším metodám tvorby webových aplikací (Upadhyaya, 2024).

2.5.1 Testování a ladění

Testovací nástroje využívající umělou inteligenci (AI) umožňují automatizaci různých forem testování, včetně jednotkového, integračního a end-to-end testování. Díky strojovému učení dokážou napodobovat chování uživatelů, detekovat nestandardní situace a efektivněji diagnostikovat problémy než tradiční manuální metody. AI také analyzuje historická data, aby identifikovala vzory a předpovídala možné chyby, což umožňuje rychlejší detekci a opravu problémů. Zapojením umělé inteligence do testovacích procesů mohou vývojáři zvýšit kvalitu kódu, minimalizovat potřebu manuálního testování a zlepšit celkovou spolehlivost softwaru (Upadhyaya, 2024).

2.5.2 Generování kódu

Technologie pro generování kódu poháněné umělou inteligencí zásadně mění způsob, jakým vývojáři vytvářejí software. Platformy jako GitHub Copilot nebo ChatGPT využívají modely strojového učení k asistenci při psaní kódu. Tyto nástroje analyzují kontext kódu, který je právě psán, a nabízejí relevantní úryvky kódu, funkce nebo dokonce celé moduly. Tato funkce výrazně zrychluje proces vývoje tím, že poskytuje okamžité návrhy a automatizuje opakující se programátorské úlohy. Kromě toho umělá inteligence pomáhá sjednotit kvalitu kódu a minimalizovat chyby tím, že nabízí konzistentní a kontextově relevantní doporučení (Upadhyaya, 2024).

Stále důležitější roli hraje AI při tvorbě a vývoji grafických uživatelských rozhraní (GUI). Návrh rozhraní obvykle začíná tím, že designéři vytvářejí koncepty na tabuli a jakmile je návrh hotový, bývá často převeden do obrázkové podoby, kterou vývojový tým následně manuálně transformuje do HTML wireframu, aby mohl zahájit implementaci. Tento postup je však časově náročný a může zpomalit celý proces vývoje. V reakci na tuto výzvu vzniklo několik přístupů využívajících umělou inteligenci, které dokážou automaticky generovat HTML wireframy přímo z ručně kreslených návrhů. Tím se výrazně zrychluje a zefektivňuje proces návrhu uživatelského rozhraní (Stocco, 2019).

2.5.3 Hrozby

Přestože přínosy této technologie jsou značné, je důležité si uvědomit její omezení a potenciální rizika, která mohou ovlivnit kvalitu softwarového vývoje. Jedním z hlavních problémů je možná nadměrná závislost na generovaných výstupech, kdy vývojáři mohou přejímat návrhy ChatGPT bez důkladné kontroly. Tento přístup může vést k implementaci suboptimálního nebo dokonce chybného kódu, jelikož model nemusí vždy poskytovat nejefektivnější či nejbezpečnější řešení (Abu Jaber et al., 2023).

Dalším významným rizikem je zkreslení v trénovacích datech (bias), které může vést k neobjektivním nebo nesprávným doporučením. ChatGPT je trénován na rozsáhlých datových sadách, které mohou obsahovat chyby, zastaralé informace nebo nevhodné vzory kódování. Výsledkem může být replikace a šíření nevhodných praktik, což by mohlo mít negativní dopad na udržitelnost a bezpečnost softwaru. Dále se objevuje problém interpretovatelnosti modelu, jelikož ChatGPT negeneruje odpovědi na základě deterministických pravidel, ale na základě pravděpodobnostního modelování. To znamená, že vývojář nemá přesný vhled do toho, proč model generuje konkrétní odpověď, což může ztížit ladění chyb, optimalizaci kódu a odhalování bezpečnostních hrozeb. Neméně vážným problémem je také bezpečnost softwarových systémů, jelikož generované návrhy nemusí vždy splňovat bezpečnostní standardy a mohou obsahovat zranitelnosti, které mohou být zneužity kybernetickými útočníky (Abu Jaber et al., 2023).

2.6 Přehled AI nástrojů pro webový vývoj

2.6.1 ChatGPT

ChatGPT je pokročilý chatbot, který využívá techniky strojového učení k analýze vstupů od uživatelů a generování smysluplných odpovědí. Byl vyvinut společností OpenAI a uveden na trh v listopadu 2022, přičemž jeho základ tvoří architektura GPT-3.5. Funguje na principu hlubokého učení, konkrétně transformátorových neuronových sítí, které umožňují zpracovávat a generovat text na základě naučených vzorů (Kanade, 2023).

Model byl trénován na rozsáhlé databázi textových zdrojů, včetně knih, článků a online obsahu, což mu umožňuje odpovídat na široké spektrum témat – od technologie a vědy po sport a politiku. Kromě faktických odpovědí dokáže také vytvářet obsah, jako jsou básně a příběhy nebo zdrojový kód (Verma & Lerman, 2022).

ChatGPT je dostupný prostřednictvím webového rozhraní nebo chatovací aplikace. Uživatelé mohou jednoduše napsat dotaz, na který model odpoví. Při generování odpovědí využívá kombinaci rozpoznávání vzorů, statistických analýz a porozumění kontextu, čímž se snaží napodobit lidskou komunikaci (Kanade, 2023).

2.6.2 Microsoft Copilot

Copilot vlastněný Microsoftem využívá GPT-4 od OpenAI, stejnou technologii, na které běží ChatGPT. Na rozdíl od ChatGPT byl Copilot vyvinut pro firemní využití. Je placený a

obsahuje pokročilé bezpečnostní prvky společnosti Microsoft, což zajišťuje vyšší úroveň ochrany dat oproti běžným generativním AI nástrojům (Team, 2024).

2.6.3 GitHub Copilot

GitHub Copilot je nástroj vyvinutý ve spolupráci mezi GitHubem a OpenAI, který využívá pokročilé modely strojového učení k automatickému doplňování kódu. Tento nástroj je navržen tak, aby zefektivnil práci programátorů a zvýšil produktivitu při psaní kódu (Zhang et al., 2023).

GitHub Copilot je postavený na základě modelu GPT-4, což je jeden z nejpokročilejších jazykových modelů pro strojové učení vyvinutý společností OpenAI. Tento model je trénován na obrovském množství veřejně dostupného kódu z repozitářů na GitHubu. Copilot tedy „chápe“ syntaxi a strukturu kódu a může na základě tohoto trénování generovat kód, který je relevantní a syntakticky správný (Zhang et al., 2023).

3 Metodika

K analýze metod vývoje webové aplikace bude využito multikriteriální hodnocení založené na Saatyho metodě (AHP), které umožňuje systematické srovnání jednotlivých přístupů na základě předem definovaných kritérií, jako jsou rychlost vývoje, kvalita kódu, či bezpečnost. Tento postup zajistí objektivní a přehledné hodnocení, které pomůže identifikovat výhody a omezení AI ve webovém vývoji.

3.1 Multikriteriální analýza metod (Saatyho metoda)

Saatyho metoda je vícekriteriální rozhodovací nástroj, který umožňuje systematicky porovnávat různé varianty na základě více kritérií. Základem této metody je párové porovnávání, při kterém jsou jednotlivým kritériím přiřazovány relativní váhy podle jejich důležitosti.

Hodnocení je následně zpracováno do matice preferencí, jejíž konzistence je ověřována pomocí vlastních čísel a indexu konzistence, čímž se minimalizuje subjektivita rozhodování a zajišťuje se logická správnost hodnocení.

Tato metoda je využívána v situacích, kde je třeba zohlednit více faktorů současně a přiřadit jim různou váhu, aby bylo možné vybrat nejoptimálnější variantu.

3.1.1 Hodnocená kritéria

Výsledné webové aplikace budou hodnoceny podle následujících kritérií podle Jalolova (2024):

- Časová náročnost vývoje
    - Jak dlouho trvá vytvoření funkční aplikace při využití programátora a AI?
    - Zahrnuje nejen napsání kódu, ale i ladění, testování a úpravy.
    - Měřitelné například v hodinách/dnech potřebných k dokončení projektu.
- Kvalita kódu
    - Přehlednost, čitelnost a udržovatelnost kódu.
    - Dodržování programátorských standardů a správných postupů.
    - Možnost snadné úpravy kódu v budoucnu.
- Přístupnost aplikace
    - Dostupnost pro čtečky obrazovky
    - Navigace klávesnicí: Zajištění, že web je plně ovladatelný pomocí klávesnice.
    - Dostatečný kontrast a barevné schéma.
- Výkonnost aplikace
    - Rychlost načítání stránky (měřeno časem načtení v milisekundách).
    - Počet požadavků, které server zvládne obsloužit za sekundu.

- Efektivita generovaného kódu – například optimalizace skriptů, načítání obrázků a používání mezipaměti (cache).

3.1.2 Váhy kritérií

Váhy jednotlivých kritérií se určují na základě párového porovnávání. Každé kritérium porovnáme s ostatními podle Saatyho škály, kde:

Textový ekvivalent	Hodnota
Obě kritéria jsou stejně důležitá.	1
Kritérium v řádku je mírně důležitější než v sloupci.	3
Kritérium v řádku je podstatně důležitější.	5
Kritérium v řádku je výrazně preferováno.	7
Kritérium v řádku je absolutně preferováno.	9

Tabulka 1: Saatyho škála (Autorské zpracování)

Vytvoření párové matice hodnocení

Kritérium	Čas vývoje	Výkonnost	Kvalita kódu	Přístupnost	Váha
Čas vývoje	1	3	5	7	**56.5 %**
Výkonnost	1/3	1	3	5	**26.2 %**
Kvalita kódu	1/5	1/3	1	3	**11.8 %**
Přístupnost	1/7	1/5	1/3	1	**5.5 %**

Tabulka 2: Párové srovnání a výpočet vah kritérií (Autorské zpracování)

V této tabulce se nachází váhy čtyř klíčových kritérií pro vyhodnocení webové aplikace. Nejdůležitějším faktorem je čas vývoje (56.5 %), poté výkonnost (26.2 %), kvalita kódu (11.8 %) a nakonec přístupnost (5.5 %). Poměr konzistence matice 4.3 % naznačující, že matice je v konzistentní a logika hodnocení je smysluplná.

4 Praktická část

4.1 Nástroje využité v praktické části

V praktické části této práce bude provedeno porovnání dvou přístupů k vývoji webových aplikací: tradičního vývoje kódu juniorním programátorem (dále jen „coder“) a vývoje s využitím umělé inteligence, konkrétně generativního jazykového modelu (LLM). Cílem je zjistit, jaké výhody a nevýhody přináší zapojení AI do webového vývoje a jak se tyto přístupy liší z hlediska efektivity, kvality a dalších praktických aspektů.

Za tímto účelem byl navržen jednoduchý responzivní webový projekt, který bude obsahovat dvě funkční části a jednu složitější komponentu. Každá část bude vytvořena dvakrát – jednou ručně a jednou s využitím AI nástroje. Projekt lze zařadit do kategorie středně náročného webu pro malé až střední podniky.

Následně budou jednotlivé výstupy porovnány na základě čtyř klíčových kritérií:

- Časová náročnost vývoje.
- Kvalita kódu.
- Přístupnost.
- Výkonnost aplikace.

K vyhodnocení bude použita Saatyho metoda (AHP), která umožňuje vícekriteriální rozhodování a přináší objektivní porovnání jednotlivých přístupů.

4.1.1 Vodopádový model

Na základě analýzy metodik SDLC se autor rozhodl použít vodopádový model. Projekt je malý, s minimálním množstvím funkcionalit, které lze jednoduše specifikovat již v úvodní fázi. Autor navíc neočekává, že by se projekt potýkal s nevýhodami vodopádového přístupu, jako jsou komplikace při změně požadavků.

Použití agilní metodiky by v tomto případě nemělo žádnou přidanou hodnotu, zejména proto, že se nejedná o týmový vývoj. Agilní principy, jako jsou denní stand-upy, by zde neměly praktické využití, protože autor pracuje samostatně a není třeba koordinovat práci v týmu.

4.1.2 Figma

K návrhu uživatelského rozhraní bude využita Figma, moderní online nástroj pro tvorbu prototypů a návrh webových aplikací. Umožňuje vytvářet responzivní rozhraní, vizuálně zkoušet různé varianty a připravit realistický návrh ještě před samotnou implementací. Díky intuitivnímu rozhraní a možnosti pracovat ve vrstvách je vhodná i pro méně zkušené designéry. Figma také pomáhá při testování použitelnosti a rozvržení prvků na stránce, čímž

zvyšuje kvalitu výsledného produktu. Její výhodou je také možnost rychlé úpravy návrhu na základě zpětné vazby.

4.1.3 Zvolené webové technologie

Pro vývoj webové aplikace byly zvoleny základní technologie HTML, CSS a JavaScript, bez použití moderních frontendových frameworků. Toto rozhodnutí vychází ze zohlednění aktuálních dovedností autora a zároveň snahy vytvořit srovnatelný a transparentní vývojový základ pro obě testované metody – ruční programování i generování kódu pomocí AI. Použití čistých technologií umožňuje lépe sledovat rozdíly v efektivitě, kvalitě i výsledném kódu, aniž by výsledek ovlivňovala logika konkrétní knihovny nebo frameworku. Tento přístup zajišťuje maximální kontrolu nad výstupem a usnadňuje analýzu výsledků experimentu.

4.1.4 ChatGPT

Pro generování části kódu bude využit ChatGPT 4o mini, což je pokročilý jazykový model vyvinutý společností OpenAI. Tento nástroj umožňuje vytvářet funkční úseky kódu, navrhovat strukturu webu, analyzovat chyby a rychle generovat opakující se části projektu.

4.2 Struktura a návrh webové aplikace

Webová aplikace navržená v této práci je určena pro fitness trenéra a její hlavní funkcí je prezentace jeho služeb široké veřejnosti a zároveň nabídka personalizovaného prostředí pro registrované zákazníky. Celý návrh aplikace vznikl ve vizuálním návrhovém nástroji Figma, kde byly vytvořeny wireframy i finální rozvržení jednotlivých obrazovek.

Aplikace je rozdělena na dvě hlavní části:

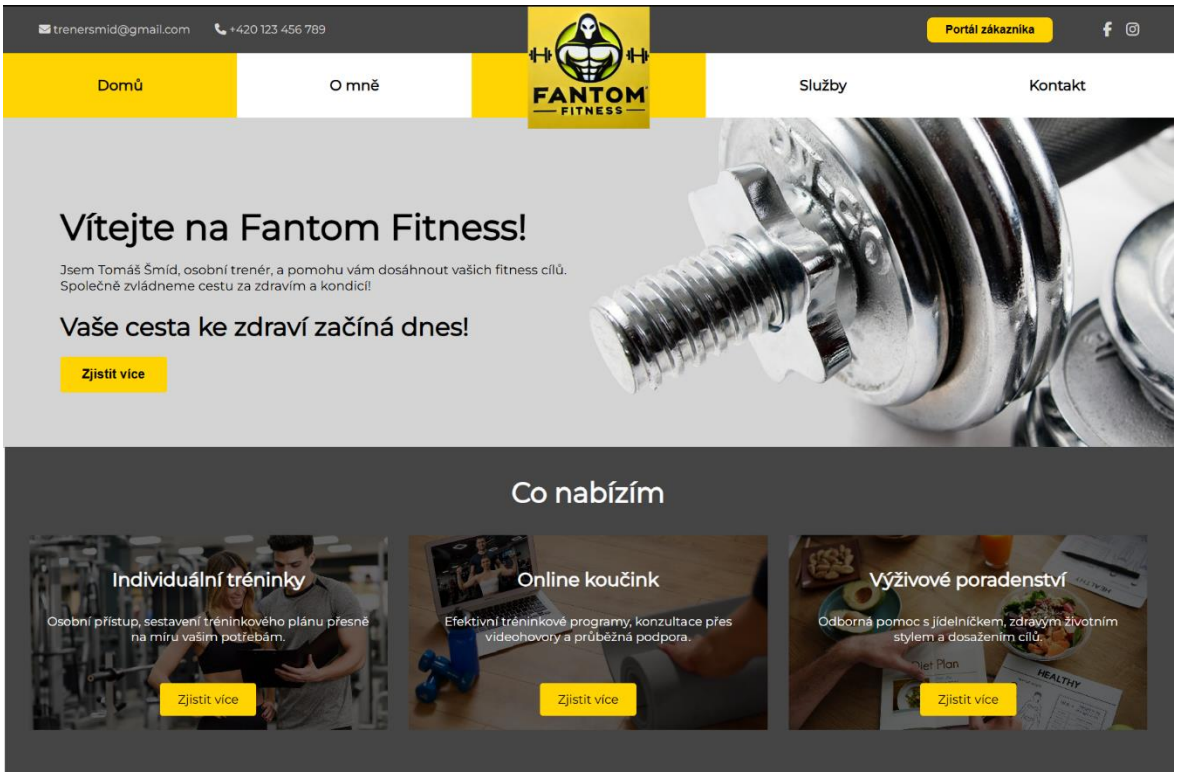
- Veřejná část webu – dostupná všem návštěvníkům bez přihlášení:
    - Domovská stránka – úvodní prezentace trenéra a motivace ke spolupráci.
    - Informace o trenérovi – profil, zkušenosti a certifikace.
    - Nabízené služby – přehled tréninkových plánů, konzultací a dalších služeb.
    - Kontakt – formulář pro zájemce, možnost přímého spojení s trenérem.
- Zákaznický portál – dostupný po přihlášení:
    - Dashboard s přehledem pokroku a denním přehledem.
    - Kalendář tréninků a jejich popis.
    - Kalkulačky (např. BMI, denní příjem tekutin, nutriční deník).
    - Návody a články (výživa, technika cvičení).

Tato struktura umožňuje kombinovat klasický prezentační web s interaktivním prostředím pro klienty.

4.3 Tvorba veřejné části

4.3.1 Popis a účel

Tato část webové aplikace je určena pro všechny návštěvníky bez nutnosti přihlášení. Jejím hlavním cílem je představit trenéra, jeho služby a umožnit jednoduché spojení. Slouží jako výkladní skříň, která má za úkol zaujmout nové klienty a poskytnout jim potřebné informace.



Obrázek 4: Úvodní obrazovka (Autorské zpracování)

4.3.2 Tvoření webu tradičním přístupem

Při vývoji veřejné části webové aplikace klasickým způsobem bylo cílem vytvořit každou stránku ručně, pomocí čistého HTML, CSS a JavaScriptu, bez využití předpřipravených šablon nebo frameworků. Postup probíhal krok za krokem podle návrhu vytvořeného ve Figmě a jednotlivé části byly tvořeny s důrazem na přehlednost a strukturovanost kódu.

Vytvoření základní šablony a struktury

Jako první byl vytvořen základní HTML soubor s rozvržením stránky. Tento soubor obsahoval typické části webové stránky:

- <head>: zde byly uvedeny základní informace o stránce, včetně odkazu na externí CSS soubor a nastavení znakové sady.
- <body>: hlavní tělo stránky bylo rozděleno na tři klíčové sekce:

- <header> – horní část obsahující logo a hlavní navigační menu.
- <main> – proměnná část, která se liší podle obsahu konkrétní stránky.
- <footer> – dolní část stránky s kontaktními údaji a případně dalšími informacemi.

Tvorba navigačního menu

Navigace byla implementována ve formě seznamu (
), přičemž každý bod obsahoval odkaz (<a>) na jednu z podstránek (např. index.html, sluzby.html, kontakt.html). Menu bylo ručně stylizováno pomocí CSS tak, aby vypadalo přehledně a bylo snadno použitelné. Byly definovány barvy pozadí, barvy textu, rozestupy mezi položkami a hover efekty pro vizuální odezvu při najetí myší. Menu bylo následně zkopírováno do všech čtyř HTML souborů, aby zůstalo na každé stránce shodné.

Patička (footer)

Stejně jako hlavička s navigací, i patička (footer) byla vytvořena jako opakující se součást všech stránek. Obsahuje základní kontaktní informace – například e-mailovou adresu trenéra, telefonní číslo nebo odkaz na sociální sítě. Footer obsahuje i logo, krátké motto a právní informace (copyright). Struktura i vzhled byly definovány jednotně pro všechny stránky, a proto byl kód této části zkopírován do všech HTML souborů.

Rozdílný obsah jednotlivých stránek

Zatímco hlavička a patička zůstávají na všech stránkách stejné, část uvnitř značky <main> se liší podle konkrétního účelu dané stránky.

- Na domovské stránce se nachází uvítací text, vizuální banner a základní informace o trenérovi a jeho službách.
- Stránka **O mně** obsahuje životopis, zkušenosti, přehled certifikací a klientské recenze s fotkami jejich proměn.
- Sekce **Služby** nabízí výčet nabízených tréninkových plánů a konzultací, včetně popisu jednotlivých možností.
- Stránka **Kontakt** obsahuje kontaktní formulář pro návštěvníky a potřebné informace pro navázání komunikace.

Tento rozdílný obsah je implementován přímo v každém HTML souboru, vždy pod stejnou strukturou stránky.

Stylování pomocí CSS

Pro vizuální stylování celé aplikace byl vytvořen externí soubor style.css, který byl připojen ke každé stránce přes <link> v hlavičce. Pomocí tohoto souboru byly definovány styly pro základní HTML prvky – nadpisy, odstavce, seznamy, tlačítka a formuláře. Každá stránka dostala vlastní třídy pro své sekce, aby bylo možné jednotlivé části stylizovat odděleně.

Kromě barevné palety odpovídající návrhu ve Figmě byl nastaven i jednotný font, velikosti písma, rozvržení jednotlivých bloků a implementace responzivity pro jednotlivá zařízení.

Interaktivita pomocí JavaScriptu

Ve veřejné části webu byl vytvořen vlastní JavaScriptový soubor, který zajišťuje základní interaktivitu bez použití externích knihoven. Skript obsahuje několik funkcí:

- Posuvník – umožňuje ruční i automatické posouvání prvků pomocí tlačítek. Automatický posun se spouští každých 10 sekund.
- Reakce na scroll – při posunutí stránky přidává logu speciální třídu, např. pro změnu velikosti nebo stylu.
- Modální okna – po kliknutí na prvky s atributem data-modal se otevře příslušné modální okno. Zavírání je řešeno klikem na křížek nebo mimo okno.
- Otevírání modálu z URL – pokud stránka obsahuje parametr: `?service=modal1`, automaticky se otevře odpovídající modální okno.

Tento skript doplňuje statický obsah webu o jednoduché, ale funkční interaktivní prvky. Veškerý JavaScriptový kód byl psán bez externích knihoven a byl umístěn odděleně v externím souboru script.js

4.3.3 Tvoření webu pomocí AI

Pro druhou variantu vývoje veřejné části webové aplikace byl využit jazykový model ChatGPT, který generoval strukturu, obsah a stylování jednotlivých stránek na základě pokynů zadaných autorem. Jako vstupní podklady byly použity návrhy vytvořené ve Figmě, které byly exportovány ve formátu PDF a následně popsány v promptu.

Úvodní prompt obsahoval manuálně zadané specifikace barev v hexadecimálním formátu, typografie (např. názvy fontů a jejich velikosti), základní strukturu designu a požadavky na funkčnost. První stránkou byla domovská stránka, která sloužila jako výchozí bod pro nastavení celkového vizuálního stylu.

Základní strukturu webu včetně hlavičky, navigace, hlavního obsahu a patičky dokázal ChatGPT vygenerovat poměrně dobře. I přesto si výstup vyžádal více iterací, zejména kvůli nedodrženým proporcím a odlišnostem v rozmístění jednotlivých prvků.

Mezi hlavní problémy patřila nefunkčnost a nekonzistence karuselu – posuvného bloku s prvky a tlačítky. Model sice navrhl základní HTML a CSS, ale chyběla logika správného přesouvání prvků i animací. Dále měl model potíže s umístěním loga a jeho zmenšením při scrollování, které nefungovalo dle očekávání.

Určité nedostatky se objevily také u patičky (footeru), zejména v zarovnání a rozložení obsahu do sloupců, což bylo třeba ručně upravit. Lehčí komplikace nastaly i u ikon – ChatGPT často nedodržel jejich požadovanou velikost nebo šířku mezery mezi nimi, což působilo nevyváženě a vyžadovalo dodatečné ladění stylů.

Naopak velmi kvalitně model zvládl vizuální stránku – správně integroval obrázek na pozadí úvodní hero sekce a obrázky do pozadí boxů s nabídkou služeb. Tyto části často odpovídaly návrhu již v první verzi, včetně vhodného ořezu, umístění a překrytí textu.

S každou další iterací a dodáním předchozích výstupů jako reference se ChatGPT lépe přizpůsoboval celkovému designu a udržoval jednotný vzhled i styl napříč stránkami, což výrazně usnadnilo generování zbylých částí webu a při implementaci responzivity pro zobrazení na různých zařízeních.

4.3.4 Porovnání přístupů

Časová náročnost

Základní struktura každé stránky byla sestavena ručně, avšak opakující se prvky jako navigační menu nebo patička byly po vytvoření první verze kopírovány mezi jednotlivými soubory, aby se předešlo zbytečné duplikaci práce. I přesto bylo nutné každou stránku upravit zvlášť, přizpůsobit obsah a upravit detaily. Celkový čas potřebný k vytvoření všech čtyř stránek činil přibližně 120 minut.

Při vývoji s využitím ChatGPT byl postup založen na iterativním zadávání přesně definovaných promptů, které popisovaly vzhled, barvy, fonty, rozvržení a funkčnost každé stránky. ChatGPT vygeneroval HTML a CSS strukturu, která byla dále upravována na základě výstupu. Díky tomu, že vizuální styl se ustálil hned na první stránce, šla tvorba dalších částí rychleji – zejména díky tomu, že model dokázal přenášet konzistenci do dalších výstupů. Celkový čas včetně iterací a korekcí se pohyboval okolo 50 minut.

Z hlediska efektivity byl tedy přístup s využitím umělé inteligence časově rychlejší. Je však důležité zdůraznit, že rozdíl vznikl zejména díky rychlému generování kódu a úspoře času při psaní základní struktury a stylů.

Výkonnost výsledného kódu

Dalším sledovaným kritériem v porovnání dvou vývojových přístupů byla výkonnost výsledných webových stránek, která byla měřena pomocí nástroje Google PageSpeed Insights. Tento nástroj analyzuje načítání stránky, optimalizaci kódu, velikost zdrojů a další faktory ovlivňující rychlost načtení a celkovou plynulost uživatelského zážitku.

Pro testování byly nahrány všechny čtyři veřejné stránky v obou variantách – ručně vytvořená verze i verze vygenerovaná pomocí ChatGPT. U každé z nich byl zaznamenán celkový skóre výkonu (performance score), které se pohybuje v rozmezí od 0 do 100 bodů, přičemž vyšší hodnota značí lépe optimalizovanou stránku.

Výsledky ukázaly, že ručně psaný kód vykazoval o něco vyšší výkonnost než kód vygenerovaný nástrojem ChatGPT. Například u domovské stránky dosáhla ruční verze skóre 93/100, zatímco verze generovaná AI měla skóre 87/100. Podobný trend byl zaznamenán i u dalších stránek, kde ruční implementace mírně převyšovala AI výstup v oblastech jako je čas načítání, velikost CSS souborů nebo zbytečné skripty.

Rozdíl lze přičíst tomu, že generovaný kód často nevolil nejefektivnější cestu k dosažení požadovaného výsledku, což se negativně projevilo na výsledném výkonu. Naproti tomu ruční implementace byla více cílená, což vedlo k lepší efektivitě při načítání i vykreslování stránky.

	Domovská stránka	O mně	Služby	Kontakt
Ručně	93	88	95	98
AI	87	72	92	95

Tabulka 3: Naměřené hodnoty výkonnosti (Autorské zpracování)

Přístupnost webu

Pro účely této práce byla přístupnost měřena pomocí nástroje Google PageSpeed Insights, který v rámci auditu zobrazuje metriky přístupnosti na škále 0–100. Obě testované varianty vývoje (ruční a s pomocí ChatGPT) vycházely z totožného návrhu vytvořeného ve Figmě, který přístupnost zohledňoval již od začátku.

Výsledné skóre se proto u obou variant lišilo jen minimálně – zpravidla v jednotkách bodů. V některých případech model ChatGPT opomněl alt atributy nebo méně přesně dodržel hierarchii nadpisů (např. chybějící H2), což bylo nutné ručně upravit. Tyto drobnosti se promítly do lehce nižšího skóre u AI varianty, ale celkově byly obě verze považovány za přístupné.

	Domovská stránka	O mně	Služby	Kontakt
Ručně	82	86	83	88
AI	77	80	79	88

Tabulka 4: Naměřené hodnoty přístupnosti (Autorské zpracování)

Kvalita kódu

Při ručním psaní kódu autor dbal na správné používání sémantických značek, správnou strukturu a přehlednost kódu. I přesto bylo nutné vícekrát použít podobný kód pro stejné funkce (např. zobrazení obrázků nebo sekcí), což vedlo k částečné duplicitě. V některých případech bylo nutné psát specifické třídy pro různé prvky, i když by šlo použít jednu obecnou třídu s jinými hodnotami.

Naopak ChatGPT generoval kód, který byl z hlediska struktury přehlednější a jednotnější. Například pro úpravy v CSS ChatGPT použil více opakovaně použitelné třídy, které zjednodušily práci a eliminovaly zbytečnou redundanci. V některých případech však byla potřeba ruční úprava (např. pro lepší pojmenování tříd nebo přidání alt atributů u obrázků), ale celkově byl výstup konzistentní a dobře strukturovaný.

Další výhodou AI generovaného kódu bylo to, že model automaticky dodržoval základní principy kódování, jako je dodržování CSS specifikací a strukturování HTML elementů. I když ne všechna generovaná řešení byla ideální, AI přistupovalo k úkolům s větší strukturovaností.

V závěru lze tedy říci, že ChatGPT generovaný kód měl v obecnosti lepší přehlednost a strukturu, ale vyžadoval minimální dodatečné úpravy pro doladění detaily a konkrétní požadavky. Ruční kód byl o něco více efektivní a mohl být přizpůsoben přesněji podle potřeb autora, ale byl více nepřehledný.

4.4 Tvorba zákaznického portálu

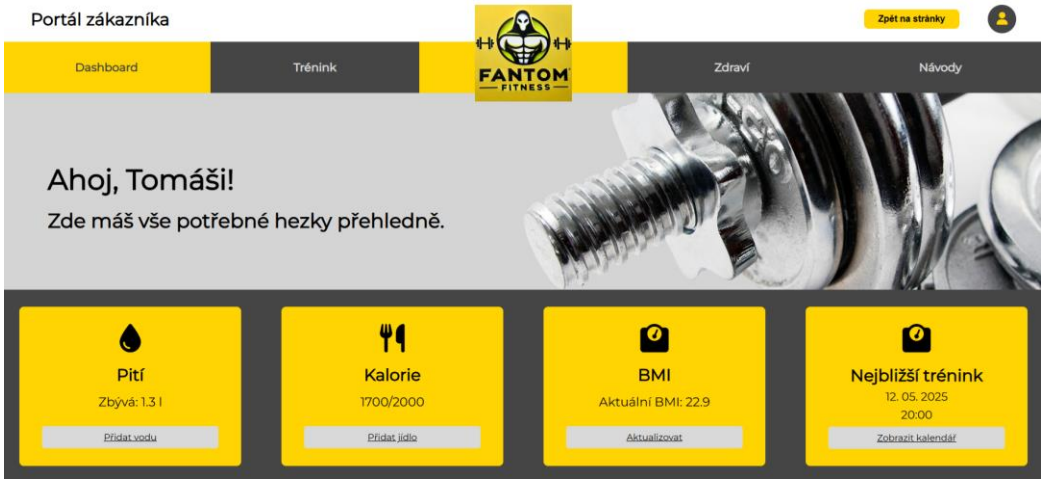
4.4.1 Popis a účel

V rámci této práce byl vytvořen zákaznický portál pro fitness trenéra, který nabízí zákazníkům možnost sledovat jejich pokrok, využívat kalkulačky, číst návody, a sledovat statistiky. Cílem bylo vytvořit intuitivní a přehledné rozhraní, které bude uživatelům poskytovat hodnotné nástroje a přehledné informace o jejich tréninkových plánech, výkonnostních statistikách a dalších relevantních údajích.

4.4.2 Tvoření webu tradičním způsobem

Styl a design zákaznického portálu

Zákaznický portál je navržen tak, aby byl vizuálně podobný veřejné části webu, ale s několika změnami, které jej odlišují a přizpůsobují pro přihlášené uživatele. Základní struktura a layout jsou zachovány, avšak barevné schéma je upraveno tak, aby bylo jasně patrné, že se jedná o soukromou sekci dostupnou pouze po přihlášení. Tyto změny zahrnují jemné odstíny modré a zelené barvy, které symbolizují zdraví a aktivitu, což souvisí s cílem portálu podporovat zdravý životní styl a dosažení cílů uživatelů.



Obrázek 5: Úvodní stránka zákaznického portálu (Autorské zpracování)

Hlavní stránka: Dashboard

Dashboard je hlavní stránkou zákaznického portálu, kde se uživatelé okamžitě dozvědí klíčové informace o svém pokroku. Na této stránce jsou zobrazeny:

- Nejbližší trénink – zobrazuje datum a čas nadcházejícího tréninku.
- Hydratace – aktuální stav hydratace na základě doporučeného denního příjmu vody.
- Dnešní příjem kalorií – počet kalorií, které uživatel snědl za aktuální den.
- Aktuální BMI – vypočítané BMI uživatele, zobrazené s barevnými indikátory pro snadné pochopení.
- Progres k dosažení cílové váhy – grafické zobrazení pokroku uživatele směrem k jeho cílové váze.
- Splněné tréninky za tento týden – počet tréninků, které uživatel úspěšně splnil v aktuálním týdnu.

Tato stránka je dynamická, aktualizuje se automaticky a poskytuje přehlednou vizualizaci klíčových metrik, které motivují uživatele k dosažení jejich cílů.

Stránka Trénink

Stránka Trénink zobrazuje seznam všech naplánovaných tréninků na daný týden. Každý trénink obsahuje:

- Seznam tréninků – zobrazení různých typů tréninků (např. kardio, síla, flexibilita) a jejich časového rozvrhu.
- Možnost zaklikávání splnění tréninku – uživatel může označit trénink jako splněný, což se propíše do dashboardu pro sledování pokroku.
- Detailní popis tréninku – každý trénink obsahuje podrobný popis, včetně cvičení, počtu opakování, doporučené délky trvání a specifikací pro úpravy podle úrovně uživatele.

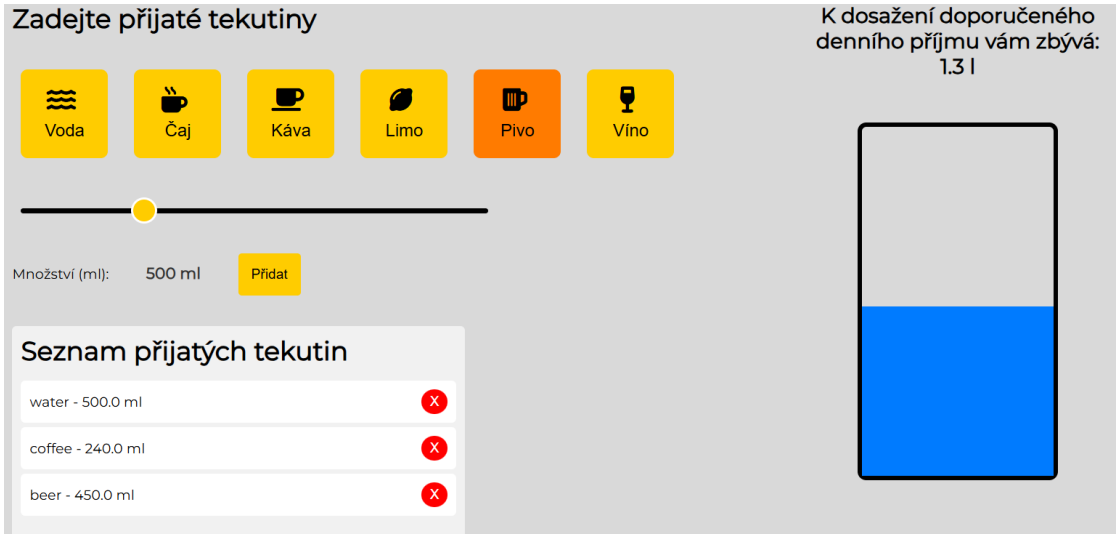
Stránka je navržena tak, aby usnadnila plánování a sledování tréninků a motivovala uživatele k pravidelnému cvičení.

Stránka Zdraví

Na stránce Zdraví najde uživatel několik užitečných nástrojů:

- Kalkulačky:
    - Hydratace – výpočet doporučeného denního příjmu vody na základě tělesné hmotnosti a aktivity.
    - BMI – kalkulačka pro výpočet BMI na základě výšky a hmotnosti.
    - Výživový deník – uživatel může zadávat svou stravu a sledovat příjem kalorií, makroživin a mikronutrientů.
- Grafy – u každé kalkulačky je zobrazen graf, který ukazuje, jak si uživatel vedl v předchozích dnech nebo týdnech, což mu umožňuje snadno sledovat svůj pokrok a identifikovat oblasti pro zlepšení.

Tato stránka poskytuje uživatelům nástroje pro detailní sledování jejich zdravotního stavu a stravovacích návyků, což je klíčové pro dosažení jejich cílů.



Obrázek 6: Kalkulačka přijatých tekutin (Autorské zpracování)

Stránka Návody

Stránka Návody obsahuje různé vzdělávací materiály, které podporují zdravý životní styl:

- Videa – uživatelé zde naleznou instruktážní videa o cvičení, technikách, stravování a dalších tématech týkajících se zdraví.
- Příručky – podrobné textové návody a články o správném stravování, cvičení a celkovém zdraví, které mohou uživatelé využít pro zlepšení svého životního stylu.

Tato stránka slouží jako edukativní zdroj, který pomáhá uživatelům získat potřebné informace a dovednosti pro efektivní trénink a správnou výživu.

JavaScriptové řešení zákaznického portálu

Autentizace uživatelů

Autentizace je klíčová pro zabezpečení přístupu do zákaznického portálu. Uživatelé se musí přihlásit pomocí e-mailu a hesla, které jsou ověřeny proti údajům uloženým v databázi.

Pro implementaci autentizace jsem použil jednoduchý formulář, který uživatelé vyplní na přihlašovací stránce. Po odeslání formuláře se provede kontrola zadaných údajů proti těm, které jsou uloženy v databázi. Pokud jsou údaje správné, uživatel je přesměrován na dashboard. Pro uchování přihlášení jsem použil cookies, které uchovávají informaci o uživatelském stavu (přihlášení nebo odhlášení).

Implementace databáze

Databázi jsem použil k uchovávání informací o uživatelských účtech, tréninkových plánech a dalších datech, jako jsou kalorie, BMI a výživový deník. Vytvořil jsem tabulky, které odpovídají jednotlivým sekcím webu a uchovávají všechny potřebné údaje.

Pro propojení databáze s frontendem jsem implementoval komunikaci mezi JavaScriptem a backendem pomocí AJAX (Asynchronous JavaScript and XML). Pomocí AJAXu jsem pravidelně načítal data z databáze, jako jsou tréninkové plány, kalorické údaje a BMI, a zobrazoval je na stránce bez nutnosti reloadu celé stránky.

Dynamické zobrazení obsahu

Dynamické zobrazení informací na stránkách jsem realizoval pomocí JavaScriptu. Po přihlášení uživatele byly na dashboardu automaticky zobrazeny údaje jako nejbližší trénink, hydratace, snědené kalorie, aktuální BMI a progres k dosažení cílové váhy.

Pro každý prvek na stránce jsem vytvořil samostatné funkce, které automaticky načítaly data z databáze a aktualizovaly příslušné HTML elementy. Tyto funkce jsem volal při načtení stránky a při každé změně hodnoty (například po přidání nového tréninku nebo po záznamu nových kalorií).

Všechny údaje byly zobrazeny v reálném čase, což poskytovalo uživatelům aktuální přehled o jejich pokroku. Pro výpočty jako BMI nebo doporučený příjem vody jsem použil jednoduché funkce, které prováděly potřebné výpočty na základě zadaných hodnot.

Stránka Trénink – seznam a označení splněných tréninků

Na stránce Trénink jsem zobrazoval seznam naplánovaných tréninků na aktuální týden. Každý trénink obsahoval název, čas, popis a možnost označení, zda byl trénink splněn. Pro tuto funkci jsem použil checkboxy. Po označení checkboxu jako splněného se změnila barva tréninku na stránce a údaje se automaticky uložily do databáze, aby se zachoval aktuální stav.

Pro aktualizaci databáze jsem využil AJAX požadavky, které odesílaly informace o splněných trénincích na server, kde byly údaje uloženy. Uživatelé tak mohli sledovat svůj pokrok a mít přehled o všech absolvovaných trénincích.

Stránka Zdraví – kalkulačky a grafy

Na stránce Zdraví jsem implementoval několik kalkulaček, jako například pro výpočet BMI, doporučený příjem vody nebo denní kalorický příjem.

Kromě kalkulaček jsem na této stránce zobrazoval grafy, které vizualizovaly pokrok uživatele v oblasti BMI a hydratace. Pro vykreslování grafů jsem použil jednoduché HTML canvas elementy, do kterých jsem vykresloval čárové grafy na základě historických údajů uživatele z databáze.

Stránka Návody – videa a příručky

Na stránce Návody jsem uživatelům poskytl přístup k různým videím a příručkám týkajícím se stravování, cvičení a zdraví. Videa byla integrována pomocí HTML5 <video>. Příručky byly textové dokumenty, které se zobrazovaly na stránce a byly interaktivní, umožňující například snadné procházení jednotlivých kapitol.

4.4.3 Tvoření webu pomocí AI

Vývoj začal odesláním Figma wireframe pro design zákaznického portálu umělé inteligenci, která na základě designu a kódu veřejné části webu začala generovat návrh kódu pro portál. AI nejprve vygenerovala základní strukturu a následně byly pomocí promptů dolaďovány detaily vzhledu a funkcionality.

Autentizace uživatelů s Firebase Authenticator

Při implementaci autentizace doporučila umělá inteligence použití Firebase Authenticator jako efektivní řešení pro správu uživatelských účtů. Po registraci a nastavení Firebase Authenticatoru byl vygenerován kód pro přihlášení a registraci uživatelů do portálu.

Firebase Authenticator zajišťuje bezpečné uchovávání hesel a správu uživatelských dat, čímž odstraňuje potřebu implementace složitého backendového řešení pro autentizaci. Dále byly implementovány cookies pro uchování informací o přihlášení, což usnadnilo správu uživatelských relací.

```
// Spustíme vše až po načtení DOMu
window.addEventListener("DOMContentLoaded", () => {
  // Přihlášení
  const loginForm = document.getElementById("login-form");
  if (loginForm) {
    loginForm.addEventListener("submit", function(e) {
      e.preventDefault();
      const email = document.getElementById("email").value;
      const heslo = document.getElementById("password").value;

      signInWithEmailAndPassword(auth, email, heslo)
        .then((userCredential) => {
          window.location.href = "portal.html";
        })
        .catch((error) => {
          alert("Chyba přihlášení: " + error.message);
        });
    });
  }
```

Obrázek 7: Inicializace autorizace uživatele (Autorské zpracování)

Databáze s Firestore

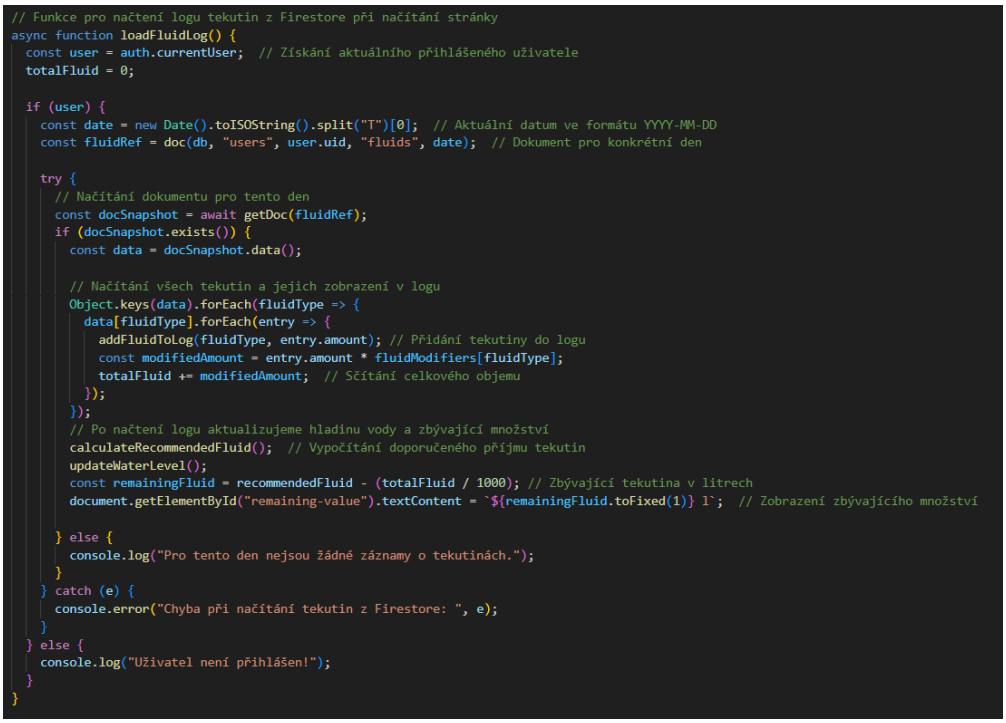
Po nastavení autentizace byla jako řešení pro správu uživatelských dat zvolena Firestore, databáze od Firebase. Umělá inteligence poskytla podrobný návod na inicializaci Firestore a napsala kód pro vytváření a správu databázových kolekcí.

Firestore byl vybrán pro svou jednoduchou integraci s Firebase a pro možnost efektivního uložení dat jako jsou tréninkové plány, kalorické údaje a BMI. Celý proces zahrnoval komunikaci mezi frontendem a databází prostřednictvím AJAX požadavků, které pravidelně načítaly a aktualizovaly data.

Funkcionality jednotlivých stránek

Po nastavení databáze byla začnuta implementace jednotlivých funkcionalit pro stránky. Každá stránka portálu, včetně Dashboardu, Tréninku, Zdraví a Návodů, měla specifikované požadavky, které byly formulovány pomocí popisů funkcí. Na základě těchto popisů umělá inteligence generovala konkrétní kód pro jednotlivé implementace.

- Dashboard – stránka, která zobrazuje uživatelský pokrok, včetně BMI, kalorií a tréninkových plánů, byla implementována tak, aby se údaje načítaly z Firestore a dynamicky se zobrazovaly na stránce.
- Trénink – na stránce Trénink byly implementovány funkce pro zaznamenávání splněných tréninků a jejich ukládání do databáze, aby uživatelé mohli sledovat svůj pokrok.
- Zdraví – na stránce Zdraví byly vytvořeny kalkulačky pro BMI a hydrataci, které na základě zadaných údajů prováděly výpočty a generovaly grafy pro vizualizaci pokroku.
- Návody – stránka obsahující videa a příručky byla vytvořena s pomocí AI, která generovala interaktivní texty a videopřehrávače.



```
// Funkce pro načtení logu tekutin z Firestore při načítání stránky
async function loadFluidLog() {
  const user = auth.currentUser;  // Získání aktuálního přihlášeného uživatele
  totalFluid = 0;

  if (user) {
    const date = new Date().toISOString().split("T")[0];  // Aktuální datum ve formátu YYYY-MM-DD
    const fluidRef = doc(db, "users", user.uid, "fluids", date);  // Dokument pro konkrétní den

    try {
      // Načítání dokumentu pro tento den
      const docSnapshot = await getDoc(fluidRef);
      if (docSnapshot.exists()) {
        const data = docSnapshot.data();

        // Načítání všech tekutin a jejich zobrazení v logu
        Object.keys(data).forEach(fluidType => {
          data[fluidType].forEach(entry => {
            addFluidToLog(fluidType, entry.amount); // Přidání tekutiny do logu
            const modifiedAmount = entry.amount * fluidModifiers[fluidType];
            totalFluid += modifiedAmount;  // Sčítání celkového objemu
          });
        });
        // Po načtení logu aktualizujeme hladinu vody a zbývající množství
        calculateRecommendedFluid();  // Vypočítání doporučeného příjmu tekutin
        updateWaterLevel();
        const remainingFluid = recommendedFluid - (totalFluid / 1000); // Zbývající tekutina v litrech
        document.getElementById("remaining-value").textContent = `${remainingFluid.toFixed(1)} l`;  // Zobrazení zbývajícího množství

      } else {
        console.log("Pro tento den nejsou žádné záznamy o tekutinách.");
      }
    } catch (e) {
      console.error("Chyba při načítání tekutin z Firestore: ", e);
    }
  } else {
    console.log("Uživatel není přihlášen!");
  }
}
```

Obrázek 8: Funkce načtení tekutin z databáze (Autorské zpracování)

Využití API – OpenFoodFacts

Pro funkci výživového deníku byla integrována externí API služba OpenFoodFacts, která poskytuje údaje o potravinách, včetně jejich nutričních hodnot. Umělá inteligence pomohla s implementací volání API, filtrováním a zobrazením relevantních informací o potravinách, které uživatel zadal do svého výživového deníku. API poskytlo snadný přístup k nutričním datům o tisících potravinách, což zjednodušilo implementaci této funkce.

Problémy při integraci

Při pokusu o integraci všech jednotlivých funkcionalit do jednoho celkového portálu se objevily problémy se synchronizací různých částí aplikace. Problém nastal při inicializaci, protože některé skripty se pokusily provádět operace před tím, než byla všechna potřebná data načtena. Tato asynchronní povaha načítání dat způsobovala, že některé části stránky se nezobrazovaly správně nebo vůbec.

Pro řešení těchto problémů byl využit debugging prostřednictvím konzolových chybových hlášek. Umělá inteligence pomohla implementovat nástroje pro inspekci chyb, které umožnily identifikovat, kde docházelo k problémům. Chybové hlášky v konzoli ukázaly, že některé operace s daty probíhaly před tím, než byla všechna data načtena z databáze.

Pro řešení těchto problémů byla použita technika asynchronního načítání dat, což zajistilo, že všechny skripty a funkce byly provedeny ve správném pořadí. Díky těmto úpravám a několika iteracím se podařilo stabilizovat aplikaci a dosáhnout správné synchronizace mezi frontendem a backendem.

4.4.4 Porovnání přístupů

Časová náročnost

Při tradičním přístupu k vývoji zákaznického portálu byla každá stránka implementována ručně, včetně funkcí pro autentizaci, připojení k databázi, dynamické zobrazení údajů a interaktivní prvky. Po vytvoření základních funkcí bylo nutné každý prvek přizpůsobit a ladit individuálně, což znamenalo časovou náročnost při testování a optimalizaci. Celkový čas na vývoj všech funkcionalit a stránek činil přibližně 300 minut, zahrnoval implementaci logiky pro přihlášení uživatelů, správu tréninků, výpočty pro zdraví a propojení s databází.

Při využití umělé inteligence byl proces vývoje mnohem rychlejší. ChatGPT generoval kód pro autentizaci, správu databáze a funkcionality jednotlivých stránek na základě definovaných promptů. Například pro implementaci autentizace doporučila AI využití Firebase Authenticatoru a pro databázi Firestore, což výrazně zjednodušilo připojení k databázi a správu uživatelských údajů. Po generování základního kódu byly jednotlivé funkce, jako sledování tréninků, výpočet BMI a hydratace nebo integrace s externími API rychle implementovány a následně pouze doladěny. Celkový čas na implementaci včetně úprav a testování byl přibližně 120 minut.

Díky AI-driven přístupu došlo k významné úspoře času při implementaci funkcí, které by tradičně vyžadovaly ruční psaní složitého kódu a ladění. Vývoj tak byl výrazně efektivnější, protože AI nejen že generovala základní kód, ale také usnadnila ladění a optimalizaci chyb.

Výkonnost výsledného kódu

Pro porovnání výkonnosti výsledného kódu obou přístupů byl využit nástroj Google PageSpeed Insights, který měří faktory ovlivňující rychlost načítání stránky a celkový výkon. Byly otestovány všechny stránky portálu, a to jak v ručně napsané verzi, tak v generované pomocí AI.

Výsledky ukázaly, že kód vytvořený ručně vykazoval lepší výkonnost než ten vygenerovaný umělou inteligencí. Například u domovské stránky dosáhla ručně napsaná verze skóre 94/100, zatímco AI verze měla skóre 83/100. Tento trend se opakoval i na dalších stránkách, kde ruční kód vykazoval nižší čas načítání a lepší optimalizaci velikosti souborů.

Rozdíl ve výkonu je způsoben tím, že generovaný kód často volil obtížnější a náročnější cestu k dosažení požadované funkcionality, zatímco ruční kód byl vždy efektivně zamířen na jednotlivé funkcionality.

	Domovská stránka	**Trénink**	**Zdraví**	**Návody**
Ručně	94	93	92	95
AI	83	81	80	90

Tabulka 5: Naměřené hodnoty výkonnosti (Autorské zpracování)

Přístupnost webu

Pro hodnocení přístupnosti webu byl použit nástroj Google PageSpeed Insights, který poskytuje skóre přístupnosti na škále od 0 do 100. Obě testované verze vývoje (ručně napsaná a generovaná pomocí ChatGPT) vycházely ze stejného návrhu, který zohledňoval přístupnost již od počátku vývoje.

Skóre přístupnosti se mezi oběma variantami lišilo pouze o několik bodů. I přesto, že obě verze splňovaly základní přístupnostní standardy, u verze generované AI se objevily menší nedostatky, jako například chybějící alt atributy pro některé obrázky nebo nesprávná hierarchie nadpisů (např. chybějící nadpis H2), které byly nutné manuálně opravit. Tyto detaily způsobily nižší skóre AI verze, ale přesto byly obě varianty považovány za přístupné pro většinu uživatelů.

	Domovská stránka	**Trénink**	**Zdraví**	**Návody**
Ručně	84	83	85	91
AI	80	80	81	89

Tabulka 6: Naměřené hodnoty přístupnosti (Autorské zpracování)

Kvalita kódu

Při ručním psaní JavaScriptového kódu byl kladen důraz na správné používání sémantických značek, strukturování funkcí a přehlednost kódu. Ačkoliv se autor snažil udržet kód co nejčistší, občas bylo nutné psát podobné funkce nebo skripty opakovaně pro různé části stránky, což vedlo k určité duplicitě. Například pro zobrazení obrázků nebo manipulaci s DOM prvky byly některé funkce a metody použity vícekrát, i když by se dalo využít obecných funkcí nebo metod, které by zjednodušily kód. V některých případech bylo třeba definovat specifické třídy a ID pro jednotlivé prvky, místo použití jedné univerzální třídy, což mělo vliv na přehlednost a údržbu kódu.

Na druhé straně, kód generovaný ChatGPT byl více strukturovaný a jednotný. Například v JavaScriptu model využíval opakovaně použitelné funkce a proměnné, čímž se zjednodušil kód a eliminovala redundantnost. Kód generovaný AI automaticky dodržoval osvědčené principy, jako je správná struktura funkcí a efektivní manipulace s DOM. I když některé části vyžadovaly drobné úpravy, například při pojmenovávání proměnných nebo přidání potřebných atributů k HTML elementům, celkově byl výstup dobře strukturovaný a konzistentní.

Výhodou generovaného kódu bylo i to, že AI automaticky dodržovala zásady dobrého kódování, například oddělení logiky od vizuální prezentace a používání čistých, snadno čitelných funkcí pro manipulaci s daty. To znamenalo, že kód byl přehledný a udržitelný. Na druhou stranu, ručně psaný kód nabízel více flexibility pro konkrétní přizpůsobení a detailní úpravy, což znamenalo, že byl snadněji přizpůsobitelný konkrétním požadavkům.

V závěru lze říci, že AI generovaný kód měl výhodu v lepší organizaci a kompaktnosti. Přesto bylo nutné provést drobné úpravy pro zajištění přesnosti a souladu s konkrétními požadavky. Ruční vývoj poskytl větší flexibilitu a efektivitu, ale vedl k větší míře nepřehlednosti.

4.5 Porovnání metod pomocí Saatyho metody

4.5.1 Srovnání přístupu při tvorbě veřejné části

Časová náročnost

Tradiční přístup vyžadoval více času na vývoj, protože každý prvek musel být ručně napsán a přizpůsoben specifickým požadavkům. To zahrnovalo opakování některých funkcí, jako je zobrazení obrázků nebo sekcí, což vedlo k duplicitě a tím i delší době vývoje. Naopak, AI-driven přístup zrychlil vývoj díky automatickému generování kódu pro opakující se prvky, což vedlo k výrazné úspoře času.

- Tradiční přístup: 6
- AI přístup: 9

Výkonnost kódu

Ručně napsaný kód byl efektivnější díky optimalizaci a absenci zbytečných prvků. Kód generovaný pomocí AI byl čistší a lépe strukturovaný, ale někdy obsahoval redundantní části, které vedly k horšímu výkonu.

- Tradiční přístup: 8
- AI přístup: 6

Přístupnost

Přístupnost byla u obou přístupů téměř shodná, ale AI-driven kód měl občas drobné nedostatky, jako chybějící alt atributy nebo nesprávně nastavenou hierarchii nadpisů. Tyto drobnosti způsobily, že AI verze měla mírně nižší skóre. Přístupnost byla přesto splněna u obou verzí.

- Tradiční přístup: 8
- AI přístup: 7

Kvalita kódu

Ručně napsaný kód byl flexibilnější a poskytoval větší kontrolu, ale obsahoval více redundantních částí, což zvyšovalo časovou náročnost. Kód generovaný AI byl přehlednější a bez zbytečných deklarací, ale vyžadoval drobné úpravy pro dosažení dokonalé kvality.

- Tradiční přístup: 7
- AI přístup: 8

Celkové hodnocení

	Tradiční přístup	AI	Váha	Výsledné skóre (tradiční)	Výsledné skóre (AI)
Časová náročnost	6	9	0.565	3.39	5.09
Výkonnost	8	6	0.262	2.096	1.57
Přístupnost	8	7	0.118	0.944	0,83
Kvalita	7	8	0.055	0.385	0.44

Tabulka 7: Celkové hodnocení kritérií (Autorské zpracování)

V tabulce jsou uvedena hodnocení jednotlivých kritérií pro obě metody. Každé hodnocení bylo následně vynásobeno příslušnou váhou kritéria, která byla získána prostřednictvím párového porovnání. Poté byly součty těchto výsledků pro jednotlivá kritéria sečteny. Celkové skóre pro tradiční metodu dosáhlo hodnoty 5.89, zatímco pro AI metodu to bylo 6.59. Na základě těchto výsledků lze usuzovat, že AI metoda je výhodnější ve srovnání s tradiční metodou podle zvolených kritérií.

4.5.2 Srovnání přístupu při tvorbě zákaznického portálu

Časová náročnost

Tradiční vývoj zákaznického portálu byl časově náročný, protože každá funkce, jako autentizace uživatelů nebo propojení s databází, musela být napsána ručně a přizpůsobena. Naproti tomu AI-driven přístup výrazně urychlil proces díky automatickému generování kódu pro běžné funkcionality.

- Tradiční přístup: 4
- AI přístup: 9

Výkonnost kódu

U zákaznického portálu ruční implementace opět vykazovala lepší výkonnost, což bylo způsobeno optimalizovaným kódem a eliminací nadbytečných prvků. Kód generovaný pomocí AI obsahoval některé nadbytečné skripty, což vedlo k horší výkonnosti.

- Tradiční přístup: 8

- AI přístup: 4

Přístupnost

Přístupnost byla u obou přístupů velmi podobná. U AI verze byly zjištěny drobné problémy, jako chybějící alt atributy nebo nesprávně nastavené nadpisy, které vedly k mírně nižšímu skóre.

- Tradiční přístup: 7
- AI přístup: 6

Kvalita kódu

Ručně napsaný kód poskytoval větší flexibilitu a kontrolu nad implementací, ale opět vedl k větší redundanci. Kód generovaný AI byl efektivní, čistý a strukturovaný, i když bylo potřeba provést drobné úpravy pro dokonalé přizpůsobení.

- Tradiční přístup: 6
- AI přístup: 8

Celkové hodnocení

	Tradiční přístup	AI	Váha	Výsledné skóre (tradiční)	Výsledné skóre (AI)
Časová náročnost	4	9	0.565	2.26	5.09
Výkonnost	8	4	0.262	2.10	1.05
Přístupnost	7	6	0.118	0.83	0.71
Kvalita	6	8	0.055	0.33	0.44

Tabulka 8: Celkové hodnocení kritérií (Autorské zpracování)

Na základě váženého porovnání kritérií pokračoval trend z předchozího měření. I zde získala tradiční metoda nižší celkové skóre 5.51, zatímco AI metoda dosáhla skóre 7.28.

4.5.3 Výsledky

Na základě provedených výpočtů a hodnocení obou přístupů (tradičního a AI-driven) pro veřejnou část webu a zákaznický portál vyplývá následující:

V hodnocení pro veřejnou část webu se ukázalo, že AI přístup dosáhl lepšího celkového skóre (6.59) ve srovnání s tradičním přístupem (5.89). Přestože tradiční vývoj vykazoval

lepší výsledky v oblasti výkonnosti kódu a kvality kódu, AI přístup exceloval v oblasti časové efektivity a rychlosti vývoje, což vedlo k jeho vyššímu celkovému hodnocení.

Podobné výsledky byly zaznamenány i u zákaznického portálu, kde AI přístup opět dosáhl vyššího celkového skóre (7.28) ve srovnání s tradičním přístupem (5.51). AI přístup byl rychlý a přehledný, přičemž tradiční přístup zůstal silnější ve výkonnosti kódu a přístupnosti.

Z výsledků hodnocení vyplývá, že AI-driven přístup vykazuje výhodu v časové efektivitě. Tento přístup výrazně zkrátil dobu vývoje, což bylo výhodné při tvorbě méně komplexních a středně velkých webových stránek, jako byla ta v této práci. AI generovalo kód rychleji a s méně manuálními úpravami, což vedlo k efektivnějšímu vývoji.

Nicméně, i přes rychlost vývoje, AI kód nebyl tak precizní a optimalizovaný jako ručně napsaný kód. V některých případech byly vygenerované kódy méně efektivní, což ovlivnilo výkonnost kódu. I když AI kód měl lepší strukturu a přehlednost, stále vyžadoval menší úpravy pro dosažení dokonalé kvality.

V kontextu této práce, kdy webová stránka měla střední velikost a nebyla příliš složitá na implementaci, byl AI-driven přístup výhodný, protože umožnil rychlé a efektivní vytvoření funkčního webu. Ovšem, pro větší a složitější projekty, kde je vyžadována větší preciznost, optimalizace kódu a flexibilita, by se tradiční vývoj ukázal jako efektivnější, protože poskytuje větší kontrolu nad detaily a kvalitou výsledného kódu.

Závěr

Tato bakalářská práce se zaměřila na zkoumání využití umělé inteligence při vývoji webových aplikací a porovnání její efektivity s tradičními metodami vývoje. Cílem bylo nejen otestovat, jak může AI urychlit proces vývoje, ale také zjistit, jaký vliv má na kvalitu, výkonnost a přístupnost výsledného kódu.

V praktické části byla provedena implementace veřejné části webu a zákaznického portálu, přičemž oba přístupy byly vyhodnoceny podle klíčových kritérií jako časová náročnost, výkonnost kódu, kvalita kódu a přístupnost. Výsledky ukázaly, že AI-driven přístup přinesl významnou úsporu času, což bylo jeho největší předností. Tento přístup výrazně zkrátil dobu vývoje, ale kód generovaný AI vyžadoval úpravy a nebyl vždy tak efektivní nebo optimalizovaný, jak by byl kód napsaný ručně.

Ve výsledku byl tradiční přístup efektivnější z hlediska výkonnosti kódu a kvality, protože umožnil přesnější kontrolu nad každým detailem. Naopak AI přístup poskytl lepší strukturovanost a přehlednost kódu, což bylo výhodné pro rychlý vývoj, ale měl nižší flexibilitu pro přizpůsobení se specifickým potřebám projektu.

V kontextu této práce, která zahrnovala středně náročný webový projekt, se ukázalo, že využití AI je efektivní a přínosné, především pro rychlý vývoj a základní úkoly. Pro složitější projekty nebo projekty s vysokými nároky na výkon a preciznost kódu by bylo lepší použít tradiční metody, které umožňují hlubší optimalizaci a detailní přizpůsobení.

V závěru lze tedy konstatovat, že umělá inteligence může výrazně zrychlit vývoj webových aplikací a ušetřit čas, avšak je nutné být opatrný při jejím nasazení na projekty, kde je vyžadována vysoká kvalita kódu a výkonnost. AI je tedy vhodným nástrojem pro středně náročné projekty, ale pro komplexní webové aplikace, které mají vyšší nároky na optimalizaci a bezpečnost, je stále nezbytný lidský zásah a tradiční vývojové metody.

Doporučení pro využívání AI

Na základě výsledků této práce lze doporučit využití umělé inteligence při vývoji webových aplikací především jako pomocníka pro efektivní a rychlý vývoj, nikoli jako náhradu za vývojáře. AI je ideální pro generování základního kódu, zejména u opakujících se komponent a strukturování aplikací. Tento přístup umožňuje rychlý vývoj a úsporu času, přičemž vývojář se může zaměřit na složitější úkoly, jako jsou optimalizace kódu, bezpečnostní aspekty a specifické úpravy pro projekt.

- Generování základního kódu – AI může být využita pro generování opakujících se částí kódu, jako jsou navigace, patičky, základní HTML struktury, což výrazně urychluje vývoj.
- Prototypování a testování – AI-driven přístup může pomoci při vytváření prototypů a testování aplikace, což umožňuje rychlé získání zpětné vazby od uživatelů a zajištění, že aplikace splňuje základní požadavky.

- Automatizace běžných úkolů – AI může automatizovat úkoly jako validace formulářů, optimalizace obrázků nebo generování API dokumentace, což uvolňuje vývojářům čas pro kreativnější práci.

AI by však neměla být používána k vývoji složitějších částí aplikací bez lidského zásahu, protože AI generuje kód, který není vždy optimální z hlediska výkonu a kvality.

Limity práce

- Složitost projektu – tento výzkum se zaměřil na středně složité projekty, kde je využití AI efektivní pro rychlý vývoj. Avšak pro komplexní a rozsáhlé webové aplikace by AI přístup nemusel poskytnout dostatečnou kontrolu nad optimalizací kódu, údržbou a bezpečnostními požadavky.
- Výběr metod a technologií – v této práci byla použita pouze jedna metoda generování kódu, konkrétně AI prostřednictvím ChatGPT. V budoucnu by bylo vhodné zkoumat další metody generování kódu a technologie, například specifické frameworky nebo nástroje pro optimalizaci kódu, a porovnat je s přístupy založenými na AI.
- Objektivita porovnání kódu – porovnání ručně napsaného kódu a AI generovaného kódu v této práci není zcela objektivní, protože závisí na metodologii testování a konkrétním přístupu k hodnocení kvality kódu. V různých projektech mohou být různé priority a kritéria pro hodnocení, což může ovlivnit objektivitu výsledků.

Možnosti budoucího rozšíření

- Větší a složitější projekty – v budoucnosti by bylo zajímavé rozšířit tuto studii na větší a složitější projekty, které by zahrnovaly například integrace s externími API, real-time funkce nebo mobilní aplikace. To by ukázalo, jak AI může ovlivnit vývoj aplikací, které vyžadují více přizpůsobení a složitější logiku.
- Dlouhodobá údržba kódu – bylo by vhodné provést dlouhodobé testování údržby kódu, který byl generován AI, a zjistit, jak se tento kód chová v průběhu času. Důležité by bylo zkoumat aktualizace a úpravy kódu v případě změn ve funkcionalitě aplikace, aby bylo možné porovnat, jak snadno je kód udržovatelný ve srovnání s tradičně napsaným kódem.
- Srovnání více nástrojů AI – dalším směrem pro budoucí rozšíření by mohlo být porovnání různých nástrojů AI pro generování kódu a frameworků pro vývoj, aby bylo možné lépe pochopit, jaké nástroje mohou být nejvhodnější pro různé typy projektů, a optimalizovat tak vývojový proces.
- Porovnání s LCNC platformami – v této práci bylo analyzováno využití umělé inteligence při vývoji webových aplikací, avšak Low-Code/No-Code (LCNC) platformy představují alternativní přístup, který umožňuje rychlý vývoj aplikací s minimálním programováním. Bylo by zajímavé v budoucnu porovnat efektivitu, kvalitu a výkonnost kódu generovaného pomocí AI a LCNC platforem, což by poskytlo širší pohled na možnosti, jak zrychlit vývoj webových aplikací a jaké kompromisy obě technologie přinášejí.

Použitá literatura

Abu Jaber, M., Beganović, A., & Almisreb, A. (2023). *Methods and Applications of ChatGPT in Software Development: A Literature Review*. 8–12. https://doi.org/10.21533/scjournal.v12i1.251

Al-Amin, M., Ali, M. S., Salam, A., Khan, A., Ali, A., Ullah, A., Alam, M. N., & Chowdhury, S. K. (2024). History of generative Artificial Intelligence (AI) chatbots: Past, present, and future development. *arXiv preprint arXiv:2402.05122*.

Anyoha, R. (2017). The history of artificial intelligence. *Science in the News*, *28*.

Astari, S. (2022). What Is HTML? Hypertext Markup Language Basics Explained. *Hostinger. com*, 2004–2024.

Banh, L., & Strobel, G. (2023). Generative artificial intelligence. *Electronic Markets*, *33*(1), 63.

Bhattacharyya, S. S., & Kumar, S. (2023). Study of deployment of "low code no code" applications toward improving digitization of supply chain management. *Journal of Science and Technology Policy Management*, *14*(2), 271–287.

Brownlee, J. (2016). Supervised and unsupervised machine learning algorithms. *Machine Learning Mastery*, *16*(03).

Calonaci, D. (2021). *Designing User Interfaces: Exploring User Interfaces, UI Elements, Design Prototypes and the Figma UI Design Tool (English Edition)*. Bpb Publications.

Colom, R., Karama, S., Jung, R. E., & Haier, R. J. (2010). Human intelligence and brain networks. *Dialogues in clinical neuroscience*, *12*(4), 489–501.

Cyroň, M. (2006). *CSS-kaskádové styly: Praktický manuál*. Grada Publishing as.

Domantas, G. (2023). What is css and how does it work? 2023. *Dostupné na: https://www. hostinger. com/tutorials/what-is-css [22. 7. 2023]*.

Dubey, A., Chaturkar, K., Pawar, R., Sarode, A., & Shirbhate, D. (2024). A Comprehensive Review of JavaScript Frameworks. *Grenze International Journal of Engineering & Technology (GIJET)*, *10*.

Ernst, D., & Louette, A. (2024). Introduction to reinforcement learning. *Feuerriegel, S., Hartmann, J., Janiesch, C., and Zschech, P*, 111–126.

Ertel, W. (2024). *Introduction to artificial intelligence*. Springer Nature.

Flanagan, D. (2011). *JavaScript: The definitive guide*. O'Reilly Media, Inc.

Graham, O., Dowe, D., & Zalta, E. N. (2021). The Turing Test. *The Stanford encyclopedia of philosophy. Metaphysics Research Lab, Stanford University. https://plato. stanford. edu/entries/turing-test*.

Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, *61*(4), 5–14.

Hron, M., & Obwegeser, N. (2022). Why and how is Scrum being adapted in practice: A systematic review. *Journal of Systems and Software*, *183*, 111110.

Jalolov, T. (2024). FRONTEND AND BACKEND DEVELOPER DIFFERENCE AND ADVANTAGES. *Multidisciplinary Journal of Science and Technology*, *4*(2), 178–179.

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). Unsupervised learning. In *An introduction to statistical learning: With applications in Python* (s. 503–556). Springer.

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695.

Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised machine learning: A brief primer. *Behavior therapy*, *51*(5), 675–687.

Jordana, A. (2024). *What Is JavaScript: A Beginner's Guide to the Basics of JS*.

Kanade, V. (2023). *What is ChatGPT? Characteristics, uses, and alternatives*.

Laoyan, S. (2024). *Guide to Waterfall Methodology: Free Template & Examples [2024] • Asana*. Asana. https://asana.com/resources/waterfall-project-management-methodology

Manning, C. (2020). Artificial intelligence definitions. *HAI Stanford Univ*, *September*, 1.

Mitchell, T. M., & Mitchell, T. M. (1997). *Machine learning* (Roč. 1, Číslo 9). McGraw-hill New York.

Model, W. (2015). Waterfall model. *Luettavissa: http://www. waterfall-model. com/. Luettu*, *3*.

Muthukrishnan, N., Maleki, F., Ovens, K., Reinhold, C., Forghani, B., & Forghani, R. (2020). Brief history of artificial intelligence. *Neuroimaging Clinics of North America*, *30*(4), 393–399.

Pargaonkar, S. (2023). A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering. *International Journal of Scientific and Research Publications (IJSRP)*, *13*(08), 345–358.

Picek, R. (2023). *Low-code/no-code platforms and modern ERP systems*. 44–49.

Razavi, S. (2021). Deep learning, explained: Fundamentals, explainability, and bridgeability to process-based modelling. *Environmental Modelling & Software*, *144*, 105159.

Rich, E. (1985). Artificial intelligence and the humanities. *Computers and the Humanities*, *19*(2), 117–122.

Rokis, K., & Kirikova, M. (2022). *Challenges of low-code/no-code software development: A literature review*. 3–17.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8–13.

Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: A modern approach*. Pearson.

Senarath, U. S. (2021). Waterfall methodology, prototyping and agile development. *Tech. Rep.*, 1–16.

Shiklo, B. (2019). Software Development Models: Sliced, Diced and Organized in Charts. *ScienceSoft. Aug*.

Shukla, A. (2023). Modern JavaScript frameworks and JavaScript's future as a full-stack programming language. *Journal of Artificial Intelligence & Cloud Computing*, *2*(4), 2–5.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Staiano, F. (2022). *Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop*. Packt Publishing Ltd.

Stocco, A. (2019). *How artificial intelligence can improve web development and testing*. 1–4.

Team, T. I. (2024). Introduction to microsoft copilot: The ai assistant for microsoft 365. *The Inform Team*. https://www.theinformteam.com/blog/introduction-to-microsoft-copilot/

Telg, R. W., Gorham, L. M., & Irani, T. (2015). The Basics of HTML: AEC568/WC230, 8/2015. *EDIS*, *2015*(7), 5–5.

Upadhyaya, N. (2024). Artificial intelligence in web development: Enhancing automation, personalization, and decision-making. *Artificial Intelligence*, *4*(1).

Verma, P., & Lerman, R. (2022). *What is ChatGPT? Everything you need to know about chatbot from OpenAI. WP Company*.

Wiering, M. A., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, *12*(3), 729.

Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using github copilot: An empirical study. *arXiv preprint arXiv:2303.08733*.