Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Life Essentials - Podpora zdravého životního stylu prostřednictvím webové aplikace

Bakalářská práce

Autor: Pavel Trejtnar

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Martina Husáková, Ph.D.

Hradec Králové duben 2026

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a uvedl jsem všechny použité prameny a literaturu. V případě využití nástrojů umělé inteligence jsem uvedl jejich účel a rozsah použití a ověřil správnost takto získaných informací.

V Hradci Králové dne 16. dubna 2026 Pavel Trejtnar

Poděkování:

Děkuji vedoucí bakalářské práce Ing. Martině Husákové, Ph.D. za metodické vedení práce a veškeré rady. Dále bych chtěl poděkovat i všem pedagogům, kteří mě naučili všechny znalosti, které jsem mohl dále rozvinout a využít v tvorbě této práce. V neposlední řadě patří velké poděkování mé rodině, zejména mé matce, za její trpělivost, morální i materiální podporu a neustálé povzbuzování během celého studia.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace zaměřené na podporu zdravého životního stylu. Cílem práce je vytvořit ucelený systém, který umožní uživatelům sledovat vybrané aspekty jejich každodenního života, jako je například příjem stravy a poskytne jim přehlednou zpětnou vazbu.

V teoretické části je provedena analýza doporučení v oblasti výživy, vývoje a bezpečnosti webových aplikací. Na základě této analýzy je navržena architektura systému, včetně databázového modelu a návrhu aplikačního rozhraní. Praktická část se zaměřuje na implementaci backendu pomocí frameworku Flask a vytvoření uživatelského rozhraní s důrazem na jednoduchost a přehlednost.

Výsledkem práce je funkční prototyp webové aplikace, který integruje více oblastí zdravého životního stylu do jednoho systému. Aplikace může sloužit jako základ pro další rozšíření.

Klíčová slova: webová aplikace, zdravý životní styl, Flask, REST API, databázový návrh, bezpečnost webových aplikací

Abstract

Title: Life Essentials - Supporting a healthy lifestyle through a web app

This bachelor's thesis focuses on the design and implementation of a web application aimed at supporting a healthy lifestyle. The main objective is to create a unified system that allows users to track selected aspects of their daily life, such as dietary intake and provides them with clear feedback.

The theoretical part includes an analysis of relevant guidelines in the fields of nutrition, web development and application security. Based on this analysis, the system architecture is designed, including the database model and API design. The practical part focuses on the implementation of the backend using the Flask framework and the development of a user interface with an emphasis on simplicity and clarity.

The result of this thesis is a functional prototype of a web application that integrates multiple aspects of a healthy lifestyle into a single system. The application can serve

as a foundation for further development.

Keywords: web application, healthy lifestyle, Flask, REST API, database design, web application security

Obsah

Seznam obrázků

1 Úvod

V současné postindustriální společnosti čelíme paradoxní situaci. Přestože lidstvo disponuje nejvyspělejšími medicínskými technologiemi v historii, výskyt civilizačních onemocnění, jako je diabetes druhého typu, kardiovaskulární choroby či chronická obezita, neustále narůstá. Hlavní příčinou tohoto stavu není nedostatečná lékařská péče, ale kumulativní dopad každodenních rozhodnutí jednotlivce. Životní styl se stal určujícím faktorem nejen pro délku života, ale především pro jeho kvalitu.

Problematika udržení zdraví však naráží na bariéru informačního přesycení a absence srozumitelné zpětné vazby. Moderní člověk je obklopen množstvím dat o své aktivitě a stravě, často však postrádá nástroj, který by tato surová data interpretoval v širším kontextu dlouhodobého zdraví a biologického stárnutí. Většina dostupných komerčních aplikací se navíc úzce specializuje pouze na jeden aspekt (např. počítání kalorií nebo sledování kroků), čímž uživateli uniká synergický efekt mezi spánkem, výživou a fyzickou aktivitou.

Tato bakalářská práce reaguje na potřebu komplexního, ale přitom uživatelsky přívětivého řešení. Představuje návrh a implementaci modulární webové aplikace. Ta si klade za cíl integrovat klíčové pilíře zdravého životního stylu do jednoho celku a poskytnout uživateli vědecky podloženou interpretaci jeho návyků. V rámci práce budou řešeny následující klíčové otázky:

- **Integrace dat:** Jak efektivně propojit různorodá biometrická a nutriční data, aby poskytovala ucelený obraz o zdravotním stavu uživatele?
- **Interpretace a motivace:** Jakým způsobem transformovat surové nutriční hodnoty a behaviorální návyky do srozumitelných ukazatelů, jako je biologický věk, za účelem zvýšení vnitřní motivace uživatele?
- **Technologická udržitelnost:** Jak navrhnout architekturu webové aplikace tak, aby byla bezpečná dle standardů OWASP, responzivní a snadno rozšiřitelná o budoucí moduly?

1.1 Motivace

Volba tématu této bakalářské práce vychází z kombinace odborného zájmu o moderní webové technologie a osobního přesvědčení o významu preventivní péče o zdraví. Jako aktivní zastánce zdravého životního stylu jsem v průběhu let testoval řadu dostupných digitálních nástrojů. Často jsem však narážel na jejich roztříštěnost. Uživatel je mnohdy

nucen využívat jednu aplikaci pro monitoring spánku, druhou pro záznam stravy a třetí pro analýzu fyzické aktivity, přičemž vzájemná synergie těchto faktorů zůstává skryta.

Osobní motivací autora bylo proto vytvořit integrovanou platformu, která by tyto izolované „střepy“ informací spojila do jednoho logického celku. Snahou bylo vyvinout nástroj, který nejen pasivně sbírá data, ale aktivně pomáhá uživateli pochopit, jak konkrétní rozhodnutí, jako je hodina spánku navíc nebo volba nutričně bohatší potraviny, ovlivňují jeho biologický věk a dlouhodobou vitalitu.

Z technického hlediska byla motivací výzva v podobě full-stack vývoje komplexního systému. Cílem bylo si v praxi vyzkoušet celý životní cyklus softwarového díla, od návrhu relačního modelu, přes implementaci asynchronní logiky v *JavaScriptu* usnadňující interakci s uživatelem, až po zajištění vysoké úrovně zabezpečení. Tato práce tak umožňuje propojit osobní vášeň pro zdravý životní styl s odbornými dovednostmi získanými během studia a vytvořit produkt s reálným přínosem pro koncového uživatele.

2 Cíl práce a metodika zpracování

Hlavním cílem této bakalářské práce je návrh a implementace modulární webové aplikace s názvem *Life Essentials*, koncipované jako rozšiřitelný soubor nástrojů pro podporu zdravého životního stylu. Aplikace využívá interaktivní vizualizační prvky k monitorování klíčových biometrických a behaviorálních dat uživatele.

Stěžejním úkolem řešení je zvýšení povědomí o přímých dopadech každodenních návyků na různé aspekty lidského života jako třeba fyzické zdraví a biologický věk. Prostřednictvím personalizované analýzy a interpretace naměřených hodnot má aplikace motivovat jednotlivce k pozitivní změně životního stylu, což v širším důsledku aspiruje na zlepšení kvality života uživatelů a podporu prevence v rámci veřejného zdraví.

2.1 Dílčí cíle práce

Pro dosažení hlavního cíle a naplnění vize plně funkční aplikace byly stanoveny následující dílčí kroky:

- **Teoretická východiska a analýza:** Zpracování rešerše v oblasti zdravého životního stylu, nutričních standardů (EFSA) a analýza bezpečnostních hrozeb webových aplikací (OWASP).
- **Návrh architektury:** Návrh databázového modelu a modulární architektury backendu s využitím frameworku Flask.
- **Zpracování dat a integrace API:** Implementace propojení s externím REST API databáze potravin pro zajištění přesných hodnot makroživin a mikroživin.
- **Uživatelské rozhraní a vizualizace:** Vytvoření responzivního frontendu, který uživateli poskytuje přehlednou a interaktivní vizualizaci nasbíraných dat
- **Zabezpečení a testování:** Ošetření aplikace proti nejčastějším webovým zranitelnostem.

2.2 Metodika zpracování

Vývojová strategie této práce byla zahájena rešerší teoretických poznatků a doporučení v oblasti zdravého životního stylu a preventivní péče. Na základě studia těchto podkladů a zhodnocení obecných potřeb uživatelů byly definovány klíčové požadavky na funkcionalitu uceleného systému.

Na fázi definování požadavků navázalo vytvoření abstraktního modelu aplikace, který precizně definuje interakce mezi jednotlivými moduly. Tato fáze byla kritická pro pochopení synergie mezi biometrickými vstupy uživatele a výpočetními algoritmy, které transformují surová data do srozumitelných motivačních výstupů.

Implementační fáze se opírá o moderní technologický zásobník, který byl zvolen s ohledem na modularitu a stabilitu. Aplikační logika (*backend*) je realizována v jazyce *Python* za využití frameworku *Flask*. Pro persistenci dat byla vybrána relační databáze *PostgreSQL*, jejíž schéma je navrženo s důrazem na integritu a kaskádové vazby mezi uživatelskými profily a jejich nutriční historií. Komunikace s datovou vrstvou je zprostředkována nástrojem *SQLAlchemy*, což umožňuje efektivní objektově-relační mapování.

Klientské rozhraní (*frontend*) je postaveno na standardech *HTML5*, *CSS3* a *JavaScriptu*. Pro zvýšení interaktivity a asynchronní komunikaci se serverem bez nutnosti obnovy stránky byla využita knihovna *jQuery*. Výpočetní jádro aplikace v reálném čase zpracovává algoritmy biologického věku na základě vědecky podložených koeficientů. Vizuální interpretace těchto dat je zajištěna integrací knihovny *Chart.js*, která generuje responzivní grafické přehledy.

Klíčovým prvkem metodiky byla integrace externích datových zdrojů prostřednictvím API švýcarské databáze potravin (*BLV Food Composition Database*), což umožnilo dynamické mapování nutričních hodnot na vnitřní datové struktury aplikace.

Celý proces tvorby probíhal v iterativních cyklech. Každý funkční celek, od zabezpečeného autentizačního systému přes modul uživatelských metrik až po komplexní nutriční deník byl samostatně vyvíjen, testován a následně integrován do finální podoby aplikace. Tento agilní přístup zajistil vysokou kvalitu kódu a snadnou rozšiřitelnost projektu o budoucí moduly.

Využití nástrojů umělé inteligence

Při zpracování této bakalářské práce a vývoji webové aplikace byly v souladu s platnými vnitřními předpisy univerzity využity nástroje generativní umělé inteligence. Umělá inteligence plnila výhradně roli pasivního konzultanta. Z hlediska účelu a rozsahu posloužila k příležitostnému brainstormingu nad strukturou práce, k diskusím při řešení dílčích programátorských problémů, k efektivnímu vyhledávání specifických pasáží v dodaných odborných textech a k drobným typografickým či jazykovým korekturám.

Správnost a relevance veškerých výstupů generovaných umělou inteligencí byla vždy důsledně ověřována. Získané rady byly podrobeny kritickému zhodnocení a manuálně otestovány v reálném provozu aplikace. V případě vyhledávání v literatuře byly iden-

tifikované pasáže vždy konfrontovány s originálním zdrojem, z něhož autor následně samostatně čerpal.

3 Zdravý životní styl

Zdravý životní styl představuje komplexní přístup k péči o vlastní tělo i mysl, který zásadním způsobem ovlivňuje nejen kvantitu, ale především kvalitu prožitých let. Moderní vědecké poznatky potvrzují, že vědomá volba nízkorizikových faktorů chování dokáže oddálit nástup chronických onemocnění a zajistit delší život v plné aktivitě. Nejde přitom o izolovaná opatření, ale o vyváženou kombinaci faktorů, jako je pravidelná fyzická aktivita, nutričně hodnotná strava a eliminace škodlivých návyků, které ve svém celku tvoří základ dlouhodobého zdraví. [1]

Moderní pojetí zdraví se neomezuje pouze na absenci chorob. Podle Světové zdravotnické organizace zahrnuje zdraví také stav fyzické, mentální a sociální pohody, který jedinci umožňuje lépe využívat příležitosti a radosti v průběhu celého života. Přijetí těchto návyků navíc plní důležitou roli v rámci rodiny, kde jedinec působí jako pozitivní vzor pro ostatní členy, čímž přispívá k dlouhodobému zdraví a spokojenosti budoucích generací. [2]

3.1 Spánek

Tradičně bývá spánek považován za jeden ze tří pilířů zdraví, vedle stravy a pohybu. Moderní poznatky však naznačují, že spánek hraje ještě významnější roli, netvoří pouze jeden z pilířů, ale představuje samotné základy, na nichž ostatní aspekty zdraví spočívají. Při jeho nedostatku se i důsledné dodržování zdravé stravy či pravidelného cvičení stává výrazně méně efektivním. [3]

Význam spánku je dán jeho vlivem na každý významný systém v těle, od buněčné úrovně až po funkci DNA. Spánková deprivace je přímo spojena s kratší délkou života a rozvojem hlavních příčin úmrtí v rozvinutých zemích, mezi které patří srdeční onemocnění, obezita či rakovina. [3]

Kromě samotné délky spánku je pro celkovou regeneraci organismu klíčová také jeho kvalita a vnitřní architektura. Kvalita odpočinku je přímo ovlivněna dodržováním pravidel spánkové hygieny, mezi něž patří zejména pravidelnost doby uléhání, omezení expozice modrému světlu ve večerních hodinách a kontrola prostředí, jako je teplota či hlučnost v místnosti. Poruchy v těchto biorytmech prokazatelně vedou k oslabení imunitního systému, snížení kognitivní výkonnosti a narušení emoční stability. [3]

Je však důležité brát v úvahu individuální potřeby jedince, které se mohou lišit v závislosti na věku, genetických dispozicích či aktuální fyzické a psychické zátěži [3].

3.2 Pohyb a fyzická aktivita

Termín pohybová aktivita lze v obecné rovině chápat jako veškerou tělesnou činnost, jejímž důsledkem je energetický výdej. V prostředí současné postindustriální společnosti však dochází k dramatickému poklesu přirozeného pohybu, což vede k rozvoji neinfekčních chronických onemocnění. [4] Podle rozsáhlé analýzy *Global Burden of Disease Study 2019* patří fyzická inaktivita mezi klíčové modifikovatelné rizikové faktory, které se zásadně podílejí na celosvětové zátěži nemocemi a předčasné úmrtnosti [5]. Závažnost situace v evropském kontextu dokládá fakt, že fyzická inaktivita je v současnosti považována za čtvrtý nejvýznamnější rizikový faktor způsobující úmrtí na celém světě. Naléhavost řešení tohoto problému zvyšuje i zjištění, že značná část dospělé populace tráví více než 5 hodin denně sezením, což představuje nezávislé riziko pro lidské zdraví bez ohledu na úroveň prováděného cvičení [6].

Studie Mandsagera et al. (2018) na vzorku více než 122 000 pacientů prokázala, že kardiorespirační zdatnost představuje jeden z nejsilnějších indikátorů dlouhodobého přežití [7]. Výzkum přinesl zjištění, že rizika spojená s nízkou fyzickou zdatností jsou statisticky srovnatelná, nebo dokonce vyšší, než u tradičních klinických faktorů, jako je aktivní kouření, diabetes mellitus či ischemická choroba srdeční [7]. Autorům se navíc podařilo vyvrátit existenci horní hranice přínosu aerobního cvičení, to znamená, že i extrémně vysoká výkonnost byla spojena s nejnižší mírou úmrtnosti, a to napříč všemi věkovými podskupinami včetně seniorů nad 70 let [7].

V návaznosti na tyto poznatky stanovuje Světová zdravotnická organizace (2025) normativní rámec pro udržení optimálního zdraví. Pro dospělou populaci je doporučeno věnovat týdně minimálně 150–300 minut aktivitě střední intenzity, nebo 75–150 minut aktivitě vysoké intenzity [4].

3.3 Strava

Výživa představuje kritický faktor formující zdraví a pohodu jednotlivců i celých populací, přičemž nevhodné stravovací návyky jsou identifikovány jako hlavní rizikový faktor pro vznik onemocnění a invalidity. Konzumace zdravé stravy v průběhu celého života pomáhá předcházet podvýživě ve všech jejích formách a zároveň chrání před řadou nepřenosných nemocí, mezi které patří diabetes, srdeční choroby, mrtvice a rakovina [8].

Základní principy zdravého stravování zůstávají univerzální, ačkoliv se konkrétní složení jídelníčku může lišit v závislosti na individuálních charakteristikách, jako je věk, pohlaví

či míra fyzické aktivity. Světová zdravotnická organizace (WHO) definuje čtyři základní pilíře zdravé stravy:

- **Adekvátnost:** Naplnění potřeb mikroživin a makroživin bez jejich překračování tak, aby se předešlo nutričním deficitům [8].
- **Rovnováha:** Celkový energetický příjem je v rovnováze s energetickým výdejem, přičemž je zachován adekvátní poměr mezi třemi primárními zdroji energie: bílkovinami, tuky a sacharidy [8].
- **Střídmost:** Omezený příjem živin a potravin, které mohou být ve větším množství zdraví škodlivé [8].
- **Rozmanitost:** Zahrnutí široké škály výživných potravin v rámci jednotlivých skupin i napříč nimi [8].

3.3.1 Makroživiny a energetická bilance

Sacharidy představují primární zdroj energie pro lidské tělo. Světová zdravotnická organizace doporučuje, aby u většiny populace tvořily přibližně 45–75 % celkového denního energetického příjmu, přičemž by měly pocházet především z celozrnných obilovin, zeleniny, ovoce a luštěnin [8]. Evropský úřad pro bezpečnost potravin (EFSA) stanovuje referenční rozmezí příjmu pro sacharidy v užším pásmu 45–60 % celkového energetického příjmu [9]. Nezbytnou součástí příjmu sacharidů je vláknina. Podle WHO i EFSA by měl denní příjem vlákniny u osob starších 10 let dosahovat alespoň 25 g [8, 9]. Naopak příjem volných cukrů by neměl přesáhnout 10 % celkové energie, což odpovídá zhruba 50 g pro osobu s průměrným energetickým výdejem [8].

Tuky jsou nezbytné pro správnou funkci buněk, avšak jejich příjem vyžaduje regulaci. Pro prevenci nezdravého přibírání na váze u dospělé populace doporučuje WHO omezit celkový příjem tuků na 30 % či méně z celkového energetického příjmu [8]. EFSA uvádí referenční rozmezí pro tuky mezi 20–35 % celkového energetického příjmu [9]. Z hlediska kvality jsou preferovány nenasycené mastné kyseliny (ryby, avokádo, ořechy, rostlinné oleje) před nasycenými tuky, jejichž podíl by neměl překročit 10 % energetického příjmu. Průmyslově vyráběné trans-tuky by měly být ze stravy vyloučeny úplně [8].

Bílkoviny plní v organismu zásadní stavební a funkční roli. Pro dospělé jedince je obecně dostačující příjem tvořící 10–15 % celkové energie [8]. Pro přesnější stanovení individuální potřeby definuje EFSA populační referenční příjem bílkovin na hodnotě 0,83 g na kilogram tělesné hmotnosti za den pro zdravé dospělé osoby [9].

3.3.2 Mikroživiny a minerály

Kromě základních makroživin hraje klíčovou roli kontrola příjmu minerálů, zejména sodíku a draslíku. Vysoký příjem sodíku (soli) je asociován se zvýšeným krevním tlakem a rizikem kardiovaskulárních chorob. WHO proto doporučuje omezit příjem soli na méně než 5 g denně (což odpovídá méně než 2 g sodíku) [8]. Naopak dostatečný příjem draslíku, který by měl dosahovat alespoň 3510 mg denně, může negativní účinky sodíku na krevní tlak zmírňovat [8]. Nedostatek klíčových mikronutrientů (např. železa, vitamínu A či jódu) zůstává globálním problémem, kterému lze předcházet konzumací nutričně bohatých potravin [8].

3.4 Mentální zdraví

Mentální zdraví je Světovou zdravotnickou organizací definováno jako stav duševní pohody, který umožňuje lidem zvládat běžné životní stresy, realizovat své schopnosti, efektivně se učit a pracovat a přispívat své komunitě. Tato oblast má pro člověka vnitřní i instrumentální hodnotu a je nedílnou součástí celkového zdraví. V každém okamžiku na psychiku jedince působí kombinace individuálních, rodinných, komunitních a strukturálních faktorů, které ji mohou buď posilovat, nebo podkopávat. [10]

Ačkoliv je většina lidí odolná, jedinci vystavení nepříznivým okolnostem, jako je chudoba, násilí, zdravotní postižení či nerovnost, čelí výrazně vyššímu riziku rozvoje duševních onemocnění. Globální data poukazují na závažnost této problematiky: v roce 2019 žilo na celém světě 970 milionů lidí s duševní poruchou, přičemž mezi nejčastější diagnózy patřily úzkostné poruchy a deprese [10].

Dopady na fyzické zdraví a délku života jsou zásadní. Duševní poruchy představují 1 ze 6 let života prožitých s invaliditou. Alarmující je zjištění, že lidé s těžkými duševními poruchami umírají o 10 až 20 let dříve než běžná populace. Kromě toho přítomnost duševního onemocnění zvyšuje riziko sebevraždy a vystavuje jedince častějšímu porušování lidských práv [10].

3.5 Návyky a závislosti

Životní styl není definován jednorázovými činy, ale souborem opakovaných vzorců chování. Komplexní studie Li et al. (2020), provedená na rozsáhlém vzorku populace, identifikovala pět klíčových nízkorizikových faktorů životního stylu: absence kouření, udržování zdravé tělesné hmotnosti, pravidelná fyzická aktivita, střídmá konzumace alkoholu a vysoká kvalita stravy. Výsledky ukázaly, že dodržování těchto pěti zásad

může prodloužit délku života ve zdraví až o 14 let u žen a o 12 let u mužů v porovnání s jedinci, kteří tyto návyky nemají. [1]

3.5.1 Rizikové faktory

Zatímco zdravé návyky život prodlužují, závislosti jej drasticky zkracují. Podle globální analýzy rizikových faktorů (Global Burden of Disease Study 2019) patří užívání tabáku a nadměrná konzumace alkoholu mezi hlavní příčiny předčasných úmrtí a zdravotního postižení po celém světě [5]. Kouření je kauzálně spojeno s řadou onkologických a kardiovaskulárních onemocnění, přičemž neexistuje žádná bezpečná míra expozice tabákovému kouři [5]. V případě alkoholu WHO a aktuální studie naznačují, že i mírná konzumace s sebou nese zdravotní rizika, která často převyšují potenciální benefity, a proto je doporučována maximální zdrženlivost [1, 4].

3.5.2 Synergie návyků

Jednotlivé návyky nefungují izolovaně, ale vzájemně se posilují v pozitivním i negativním smyslu. Matthew Walker (2017) upozorňuje na kritickou roli spánku v regulaci chování. Chronická spánková deprivace narušuje funkci prefrontálního kortexu, což vede ke snížené sebekontrole a vyšší tendenci k impulzivnímu jednání, včetně konzumace nezdravých potravin či návykových látek [3].

Tento mechanismus vytváří nebezpečnou smyčku, kdy jeden špatný návyk (nedostatek spánku) spouští kaskádu dalších rizikových chování (špatná strava, inaktivita), které ve svém součtu signifikantně zvyšují biologický věk organismu [3, 1].

Efektivní změna životního stylu proto vyžaduje holistický přístup, který se nezaměřuje pouze na eliminaci jednoho zlozvyku, ale na celkovou kultivaci prostředí a rutiny podporující zdraví [4].

4 Technologie a prostředky pro vývoj webové aplikace

Způsob návrhu a distribuce softwaru zaznamenal v posledních desetiletích zásadní proměnu. Od éry centralizovaných mainframů, a následné dominance samostatných desktopových aplikací, se pozornost vývojářů i uživatelů přesunula k webovým aplikacím. Tento model v sobě kombinuje výhody snadné správy a vysoké bezpečnosti na straně serveru s využitím výpočetního výkonu koncových zařízení pro vykreslování uživatelského rozhraní [11].

Moderní webová aplikace, jakou je i *Life Essentials*, je založena na principu dynamického generování obsahu. Na rozdíl od statických stránek je její podoba vytvářena v reálném čase pomocí skriptů, které na základě požadavků uživatele komunikují s databází a výsledná data vkládají do připravených HTML šablon [11].

4.1 Frontend

Prezentační vrstva, často označovaná jako frontend, implementuje vizuální podobu a interaktivní prvky aplikace. Je zodpovědná za zobrazení dat uživateli, přijímání uživatelských událostí a celkové řízení uživatelského rozhraní. Pro tvorbu této vrstvy byly využity standardizované technologie, kde HTML5 definuje sémantickou strukturu obsahu, CSS3 zajišťuje vizuální formátování prvků a JavaScript se stará o dynamickou interaktivitu a validaci dat na straně klienta [12].

4.1.1 HTML5

Jazyk HTML5 (HyperText Markup Language) představuje standardní a základní stavební kámen každé webové stránky, přičemž jednotlivé prvky jsou v něm reprezentovány pomocí značek, které prohlížeči určují způsob interpretace a zobrazení obsahu [13]. Tato technologie slouží jako moderní prostředek pro definování sémantické struktury dokumentu, což je klíčové pro správné vykreslování prvků, jako jsou nadpisy, tabulky, seznamy či formuláře [14]. Každý HTML element je zpravidla tvořen startovacím a ukončovacím tagem, mezi které se vkládá samotný textový nebo multimediální obsah [13]. Správné použití tohoto jazyka umožňuje vytvářet přehledné a strukturované weby, které slouží jako kostra pro další stylování a dynamické funkce [14].

4.1.2 CSS3

Vizuální podoba moderních webů je definována pomocí kaskádových stylů CSS3 (Cascading Style Sheets). Tato technologie umožňuje efektivně oddělit sémantickou struk-

turu dokumentu od jeho designu, čímž zajišťuje konzistenci vzhledu napříč celým systémem a výrazně usnadňuje údržbu kódu [15]. CSS3 se zaměřuje na formátování dokumentů vytvořených v jazyce HTML a nabízí široké možnosti pro definování barevných schémat, typografie, rozvržení prvků i pokročilých vizuálních efektů [16]. Využití moderních selektorů a vlastností umožňuje vývojářům vytvářet esteticky atraktivní rozhraní, která jsou přehledná a funkční na široké škále koncových zařízení [15, 16].

4.1.3 Bootstrap 5

Pro zajištění plné responzivity a zrychlení vývoje uživatelského rozhraní se často využívá framework Bootstrap. Jedná se o výkonnou sadu nástrojů, která využívá strategii mobile-first, což znamená, že design je primárně optimalizován pro mobilní zařízení a následně se škáluje pro větší obrazovky [17]. Bootstrap poskytuje vývojářům rozsáhlou knihovnu hotových komponent a využívá propracovaný systém mřížek (grid system) pro flexibilní uspořádání obsahu na stránce [18]. Integrace tohoto frameworku umožňuje dosáhnout profesionálního vzhledu s vysokou kompatibilitou mezi různými prohlížeči, aniž by bylo nutné psát veškeré CSS styly manuálně [17, 18].

4.1.4 Javascript a moderní alternativy

Interaktivitu na straně klienta zajišťuje programovací jazyk JavaScript, což je univerzální technologie nezbytná pro tvorbu moderních dynamických webů v reálném čase [19]. JavaScript umožňuje běh komplexních aplikací přímo v prohlížeči uživatele, kde se stará o logiku uživatelského rozhraní, jako je validace vstupních dat, obsluha událostí nebo okamžitá reakce na interakci uživatele [20]. Tento jazyk se neustále vyvíjí a v současnosti je klíčovým prvkem pro zajištění plynulého uživatelského zážitku a efektivní komunikaci mezi klientskou částí a serverem [19, 20]. Ačkoliv je JavaScript dominantní, existují technologie, které jeho možnosti rozšiřují nebo nabízejí alternativní přístup k vývoji. Významným zástupcem je TypeScript, nadstavba JavaScriptu vyvinutá firmou Microsoft, která do vývoje vnáší statické typování a prvky objektově orientovaného programování. Zatímco čistý JavaScript využívá dynamický typový systém, TypeScript vyžaduje striktní definici typů proměnných, což představuje tolik potřebný řád a přehlednost při tvorbě rozsáhlých projektů. Hlavní výhodou je schopnost odhalit chyby v kódu již během psaní (v době kompilace) namísto až za běhu aplikace, což výrazně usnadňuje debugging. Nevýhodou však zůstává nutnost využití transpileru, který musí kód převést do standardního JavaScriptu, aby mu engine v prohlížeči rozuměl, což mírně zvyšuje objem zdrojových souborů. [21]

Další moderní alternativu představuje framework Flutter, který umožňuje vývoj pro webové prostředí pomocí jediného sdíleného kódu napříč platformami. Na rozdíl od tradičních řešení nevyužívá Flutter k vykreslování standardní sémantické prvky HTML, ale vykresluje uživatelské rozhraní na grafické plátno (*canvas*) pomocí technologie WebAssembly, což zajišťuje vysoký grafický výkon a plynulé animace blízké nativním aplikacím. Mezi zásadní nedostatky tohoto přístupu však patří problematická optimalizace pro vyhledávače (*SEO*) a pomalejší prvotní načítání aplikace kvůli větší velikosti stahovaných balíčků. I přes technologický pokrok alternativních řešení tak klasické technologie postavené na JavaScriptu zůstávají v současnosti flexibilnější a vhodnější volbou pro standardní webové projekty vyžadující rychlou odezvu a dohledatelnost. [22]

4.1.5 Knihovna jQuery

Doplňkovým nástrojem pro usnadnění práce se skripty je knihovna jQuery, jejímž hlavním účelem je výrazné zjednodušení manipulace s DOM (Document Object Model) a obsluha událostí [23]. Tato knihovna je oblíbená pro svou stručnou a přehlednou syntaxi, která umožňuje vývojářům psát efektivní kód s minimálním množstvím znaků ve srovnání s čistým JavaScriptem [24]. Použití jQuery přispívá k lepší čitelnosti kódu a usnadňuje provádění asynchronních požadavků i tvorbu animací, které jsou standardem moderních webových aplikací [23, 24].

4.2 Backend

Aplikační vrstva, tvořící backend systému, funguje jako samostatný celek, který řídí veškerou funkčnost aplikace skrze detailní výpočty a zpracování logiky. V této části dochází k řešení klíčových operací a uplatňování algoritmů, přičemž vrstva zároveň zajišťuje autentizaci uživatelů a řídí procesy bezpečného načítání dat. Logická vrstva tak tvoří nezbytný mezičlánek, který transformuje požadavky z uživatelského rozhraní na konkrétní akce a výsledky, jež jsou následně předávány zpět k zobrazení [12].

4.2.1 Python a konkurenční řešení

Programovací jazyk Python představuje vysoce úrovňový, interpretovaný a objektově orientovaný nástroj, který se vyznačuje především svou jednoduchou syntaxí podobnou lidské řeči [25, 26]. Tato vlastnost umožňuje vývojářům psát programy s menším počtem řádků kódu ve srovnání s mnoha jinými jazyky, což výrazně zvyšuje produktivitu a čitelnost výsledného softwaru [26, 27]. Python je navržen jako jazyk pro obecné

použití, což znamená, že kromě webového vývoje nachází uplatnění také v datové vědě, umělé inteligenci, automatizaci či vývoji desktopových aplikací [25, 27].

Při volbě technologie pro backendovou část aplikace bývá Python často porovnáván s jazykem PHP. Zatímco PHP bylo vytvořeno primárně jako nástroj pro generování dynamického HTML obsahu a je úzce specializováno na webové prostředí, Python nabízí širší přenositelnost a robustnější podporu knihoven pro různorodé domény. Přestože PHP může v určitých verzích vykazovat vyšší rychlost provádění skriptů, Python nad ním vítězí v přehlednosti syntaxe, snadnější údržbě kódu a efektivnějším procesu ladění. [25]

Zatímco popularita PHP v posledních letech spíše stagnuje, Python zaznamenává masivní nárůst oblíbenosti mezi profesionálními vývojáři i začátečníky, a to především díky svému modernímu ekosystému a silné komunitní podpoře [25, 27]. Pro účely moderního webového vývoje se tak Python v kombinaci s vhodným frameworkem jeví jako ideální volba pro tvorbu udržitelných a snadno rozšiřitelných aplikací [26].

4.2.2 Framework Flask

Flask představuje moderní webový mikrorámec (microframework) určený pro programovací jazyk Python [28]. Označení mikrorámec v tomto kontextu neznamená, že by byl nástroj omezený ve své funkčnosti, ale odkazuje na filozofii udržení co nejjednodušího a nejlehčího jádra [29]. Na rozdíl od monolitických frameworků Flask neobsahuje vestavěné komponenty pro běžné úlohy, jako je validace formulářů nebo abstrakce databázové vrstvy, a místo toho dává vývojářům naprostou svobodu při volbě externích knihoven [28, 29].

Z technického hlediska je Flask postaven na webovém toolkitu Werkzeug zajišťujícím rozhraní WSGI (Web Server Gateway Interface) a šablonovacím systému Jinja2. Architektura frameworku je navržena tak, aby byla snadno rozšiřitelná pomocí široké škály komunitních doplňků, což umožňuje transformovat jednoduchý skript v robustní a komplexní aplikaci. [29] Pro definování trasování požadavků (routing) využívá Flask Pythonovské dekorátory, což přispívá k vysoké čitelnosti a čistotě zdrojového kódu. Tato kombinace minimalismu a rozšiřitelnosti činí z Flasku jeden z nejpopulárnějších nástrojů pro agilní vývoj webových rozhraní a API služeb. [28]

Vzhledem k minimalistické povaze frameworku Flask je pro vývoj komplexnějších systémů nezbytné integrovat specializované knihovny, které zajišťují robustnost a bezpečnost aplikace. Nejdůležitějším prvkem pro správu dat je v projektu využívané rozhraní SQLAlchemy. Jedná se o objektově-relační mapování, které slouží jako abstraktní vrstva nad databází a umožňuje manipulovat s daty pomocí Python tříd namísto přímého

psaní SQL dotazů. Tento přístup výrazně zvyšuje bezpečnost aplikace, neboť automaticky chrání systém před útoky typu *SQL injection* skrze parametrizované dotazy. [30]

Základním kamenem, na kterém Flask staví svou funkčnost, je toolkit Werkzeug. Ten zajišťuje nejen nízkoúrovňovou komunikaci mezi webovým serverem a aplikací, ale obsahuje i kritické bezpečnostní nástroje. V rámci autentizace uživatelů jsou využívány jeho moduly pro bezpečné hashování hesel, konkrétně algoritmy, které zabraňují ukládání citlivých údajů v čitelné podobě a chrání je proti útokům hrubou silou [31].

Pro budoucí standardizaci a pokročilou validaci uživatelských vstupů lze využít knihovnu WTForms, která nabízí deklarativní způsob definice formulářových polí a integrovanou ochranu proti útokům typu *Cross-Site Request Forgery* (CSRF) [32].

Z širokého ekosystému frameworku Flask byly pro potřeby tohoto projektu vybrány pouze ty nástroje a moduly, které jsou stěžejní pro jeho stabilitu a bezpečnost. Detailní analýza dalších dostupných rozšíření není předmětem této práce, neboť by přesahovala její vymezený rozsah.

4.2.3 Šablonovací systém Jinja2

Pro oddělení aplikační logiky od prezentační vrstvy využívá framework Flask šablonovací systém Jinja2, který slouží k dynamickému generování HTML dokumentů na straně serveru [33]. Jedná se o rychlý a expresivní nástroj, který vývojářům umožňuje tvorbu obsahu pomocí syntaxe blízké jazyku Python [34].

Klíčovým mechanismem systému je dědičnost šablon. Tento princip spočívá ve vytvoření hlavní šablony, která definuje společné rozložení prvků, jako jsou navigace či zápatí, zatímco jednotlivé podstránky pouze doplňují specifický obsah do předem vymezených bloků [33]. Tento přístup výrazně usnadňuje údržbu kódu a zajišťuje vizuální konzistenci napříč celou aplikací [34].

Z hlediska bezpečnosti Jinja2 standardně implementuje mechanismus automatického ošetřování znaků (*autoescaping*), který neutralizuje potenciálně nebezpečné vstupy a chrání tak aplikaci před útoky typu *Cross-Site Scripting* (XSS) [34]. Architektura a syntaxe tohoto systému jsou natolik efektivní, že se staly standardem i v jiných odvětvích vývoje, což dokládá například existence systému Twig v jazyce PHP, který z principů Jinja2 přímo vychází [35]

4.3 Databáze

Poslední část systému tvoří datová vrstva, která sestává z databázových serverů a samotného systému správy databáze. Tato vrstva slouží k perzistentnímu ukládání a opětovnému získávání informací, přičemž uchovává data zcela nezávislá na aplikační logice. Oddělení dat do vlastní vyhrazené vrstvy přispívá k lepší škálovatelnosti a celkovému výkonu aplikace, neboť umožňuje optimalizovat přístup k informacím bez přímého ovlivnění uživatelského rozhraní [12].

Databázové systémy představují fundamentální prvek pro perzistentní ukládání dat a zajištění jejich integrity napříč životním cyklem aplikace. V současné architektuře webových systémů lze volit mezi dvěma dominantními přístupy a to relačním (SQL) a nerelačním (NoSQL), přičemž každý z nich nabízí specifické vlastnosti vhodné pro odlišné typy datové zátěže [36, 37]

4.3.1 PostgreSQL a alternativy

Hlavní rozdíl mezi zmíněnými paradigmaty spočívá v metodě organizace dat a způsobu škálování. Relační databáze (SQL) využívají pevnou strukturu tabulek a striktně definované schéma, což zaručuje vysokou integritu dat a plnou podporu transakčních operací dle standardů ACID [36]. Nevýhodou tohoto řešení však může být náročnější horizontální škálování a neohebnost schématu při agilních změnách struktury. Naopak nerelační systémy (NoSQL), jako je například dokumentově orientované MongoDB, nabízejí vysokou flexibilitu díky absenci pevného schématu [37]. Data jsou zde ukládána ve formátu JSON či BSON, což umožňuje snadné vnořování objektů a efektivní horizontální škálování v distribuovaných systémech, ovšem za cenu potenciální nekonzistence dat v určitých scénářích [36].

V rámci segmentu otevřených relačních systémů jsou nejčastěji skloňovány nástroje MySQL a PostgreSQL. Systém MySQL je dlouhodobě etablovaný jako rychlé a snadno konfigurovatelné řešení, které vyniká především v aplikacích s vysokou frekvencí operací čtení nad méně komplexními datovými strukturami. Mezi jeho limity však patří méně striktní implementace SQL standardů a omezenější podpora pokročilých datových typů. [38]

Oproti tomu PostgreSQL představuje pokročilý objektově-relační systém, který klade maximální důraz na bezpečnost, rozšiřitelnost a integritu uložených informací. Jeho klíčovými výhodami jsou plná podpora komplexních SQL dotazů, možnost definice vlastních datových typů a funkcí a vysoká spolehlivost při zpracování náročných transakčních úloh.[39] Ačkoliv může být PostgreSQL pro začínající vývojáře náročnější na

úvodní konfiguraci, v rámci profesionálních projektů je oceňován pro svou robustnost, která minimalizuje riziko ztráty či poškození dat [38].

4.4 Bezpečnost

Bezpečnost v digitálním prostředí představuje komplexní disciplínu, která se vyvíjí v přímé závislosti na technologickém pokroku a měnící se povaze interakcí v kybernetickém prostoru. Pro správné pochopení problematiky je nezbytné nejprve ukotvit teoretické rozdíly mezi tradiční ochranou dat a moderním pojetím kybernetické bezpečnosti. [40]

4.4.1 Kybernetická a informační bezpěcnost

V odborné diskuzi dochází k častému zaměňování pojmů informační bezpečnost a kybernetická bezpečnost, ačkoliv se jedná o koncepty s odlišným rozsahem a cíli. Tradiční informační bezpečnost je historicky ukotvena jako disciplína zaměřená na ochranu informačních aktiv. Jejím primárním účelem je zajištění kontinuity procesů a minimalizace rizik plynoucích z narušení důvěrnosti, integrity nebo dostupnosti dat, která organizace či systém spravuje. Jak uvádějí Von Solms a Van Niekerk [40], tento přístup vnímá informace jako statická nebo procesní aktiva, u nichž je nutné hlídat jejich perimetr a přístupová oprávnění.

Výrazným specifikem informační bezpečnosti je její pohled na lidský faktor. Člověk je v tomto paradigmatu vnímán především funkčně, jako uživatel s přidělenou rolí, správce systému, nebo v negativním smyslu jako slabý článek řetězce, tedy vulnerabilita, skrze kterou může dojít k bezpečnostnímu incidentu [40]. Bezpečnostní kontroly jsou pak navrhovány tak, aby omezovaly chybovost nebo zneužití pravomocí ze strany těchto subjektů.

S rozvojem globálně propojeného digitálního prostředí se však hranice bezpečnosti rozšiřují a vzniká koncept kybernetické bezpečnosti. Ten, na rozdíl od informační bezpečnosti, nechrání pouze „informace“ jako takové, ale zaměřuje se na ochranu veškerých aktiv, která jsou v kyberprostoru dosažitelná. Kybernetický prostor je zde chápán jako komplexní, neustále se měnící prostředí tvořené interakcemi mezi lidmi, softwarem a službami napříč technologickou infrastrukturou [40].

Zásadní teoretický posun, který kybernetická bezpečnost přináší, spočívá v redefinici postavení člověka. Von Solms a Van Niekerk [40] argumentují, že v kybernetické bezpečnosti již člověk není pouhou rolí nebo hrozbou, ale stává se samostatným aktivem, které je cílem ochrany. Tento posun má hluboké etické a společenské implikace. Kyber-

netické útoky totiž nemusí cílit na krádež firemních dat, ale mohou být vedeny přímo proti jednotlivci s cílem narušit jeho soukromí, digitální identitu, nebo mu způsobit psychickou či společenskou újmu. Kybernetická bezpečnost se tak stává multidisciplinárním oborem, který v sobě integruje nejen technické aspekty, ale i psychologii, právo a etiku.

Zatímco tedy informační bezpečnost řeší otázku jak chránit data pro potřeby systému, kybernetická bezpečnost se ptá jak chránit všechny subjekty a jejich zájmy v propojeném digitálním světě [40]. Tento rozdíl je fundamentální pro pochopení moderních hrozeb. V kyberprostoru totiž neexistuje jasně definovaný perimetr a útočníci mohou zneužívat nejen technické slabiny kódu, ale i sociální vazby a důvěru samotných uživatelů. Studium kybernetické bezpečnosti proto vyžaduje holistický pohled, který uznává, že selhání jedné technické komponenty může mít kaskádový efekt na bezpečnost konkrétních osob a celých společenských struktur. Tato teoretická báze je nezbytným předpokladem pro následnou analýzu specifických hrozeb a návrh obranných mechanismů, které se již neomezují pouze na technickou stabilitu, ale reflektují odpovědnost tvůrce systému vůči bezpečí uživatele v celém kybernetickém prostoru. [40]

4.4.2 Webová bezpečnost

Webová bezpečnost se primárně zaměřuje na ochranu citlivých informací, jako jsou uživatelská jména, hesla, bankovní údaje či soukromé algoritmy, před neoprávněným přístupem. Kompromitace těchto dat může mít za následek vážné důsledky, od finančních ztrát a krádeží identity uživatelů až po úplné ovládnutí služeb útočníkem, známé jako hijacking. V této souvislosti je nezbytné rozlišovat mezi pojmy bezpečnost a soukromí. Zatímco bezpečnost představuje akt aktivní ochrany soukromých dat a systémů před neautorizovaným přístupem, soukromí se týká kontroly uživatele nad tím, jak jsou jeho data sbírána, uchovávána a dále využívána. Platí přitom fundamentální zásada, že robustní bezpečnost je nezbytným předpokladem pro zachování soukromí; bez zabezpečeného webu jsou veškeré zásady ochrany údajů neúčinné. [41]

Podle dokumentace MDN základní bariéru proti útokům tvoří bezpečnostní model implementovaný přímo ve webových prohlížečích. Tento model se opírá o několik klíčových mechanismů, které definují pravidla pro interakci s obsahem a přenos dat [41]:

- **Same-origin policy (SOP):** Tento základní bezpečnostní mechanismus omezuje způsob, jakým může skript z jednoho zdroje interagovat s prostředky z jiného zdroje. Tato izolace pomáhá eliminovat vektory útoků a brání neoprávněnému přístupu k datům napříč různými weby.

- **Šifrovaná komunikace (HTTPS/TLS):** Protokol HTTP využívá vrstvu TLS (Transport Layer Security) k šifrování dat během jejich transportu po síti. To zabraňuje třetím stranám v odposlechu nebo škodlivé manipulaci s přenášenými informacemi.
- **Bezpečné kontexty a oprávnění:** Přístup k výkonným funkcím prohlížeče, jako je webkamera, systémová oznámení nebo platby, je omezen pouze na zabezpečená spojení a vyžaduje explicitní souhlas uživatele.

Tyto integrované funkce tvoří nezbytný základ, ovšem pro komplexní ochranu aplikace je nutné je kombinovat s odpovědným přístupem na straně vývojáře, který musí eliminovat zranitelnosti vznikající přímo v aplikační logice. [41]

Přestože moderní prohlížeče nabízejí širokou škálu ochranných prvků, samotné zabezpečení systému leží z velké části na zodpovědnosti vývojáře. I drobné chyby v logice kódu mohou vytvořit zranitelnosti, které útočníkům umožní získat neautorizovanou kontrolu nad aplikací nebo daty. Z tohoto důvodu je nutné doplňovat nativní ochranu prohlížeče o specifické techniky, jako je důsledná sanitace vstupů z formulářů, používání bezpečnostních hlaviček a pravidelná analýza rizik podle mezinárodně uznávaných standardů. [41]

4.4.3 Hrozby webových aplikací a možnosti jejich obrany

Identifikace a pochopení hrozeb představuje kritický krok pro zajištění bezpečnosti v kyberprostoru, neboť moderní útoky často necílí pouze na technické zranitelnosti, ale zneužívají i logické chyby v návrhu systému a soukromí uživatelů. Aby vývojáři dokázali efektivně chránit integritu dat a identitu subjektů, musí operovat s aktuálními daty o nejčastějších vektorech útoků. [41]

Celosvětově nejuznávanějším standardem v této oblasti je projekt OWASP Top 10, spravovaný neziskovou nadací OWASP Foundation. Tato organizace pravidelně zveřejňuje a aktualizuje seznam deseti nejvýznamnějších a nejčastěji se vyskytujících druhů hrozeb, které představují největší bezpečnostní riziko pro webové aplikace. Využití této metodiky umožňuje prioritizovat obranné mechanismy tam, kde je dopad potenciálního útoku nejzávažnější. [42] Následující přehled podrobněji rozebírá klíčové kategorie hrozeb, vysvětluje princip fungování nejznámějších útoků a definuje konkrétní možnosti obrany, které by měly být implementovány již v raných fázích vývoje systému.

Selhání řízení přístupu

Tato kategorie dlouhodobě dominuje žebříčku nejzávažnějších zranitelností, přičemž

některá z forem této chyby byla identifikována u všech aplikací zahrnutých do testování. Podstatou problému je selhání mechanismů, které mají garantovat, že uživatelé nebudou moci vykonávat aktivity nad rámec svých legitimních práv. Nedostatečná kontrola vede k neautorizovanému odhalení dat, jejich nechtěné úpravě či smazání, nebo k provádění operací, které by měly být běžnému uživateli zapovězeny. [43]

Mezi typické příklady těchto nedostatků patří nedodržení principu minimálních oprávnění (*principle of least privilege*), obcházení bezpečnostních bariér skrze manipulaci s URL parametry nebo neoprávněný přístup k cizím účtům prostřednictvím nezabezpečených přímých odkazů na objekty (IDOR). Značné riziko představuje také neoprávněné získání administrátorských práv či podvržení identifikačních metadat, jako jsou například modifikované soubory cookies nebo přístupové tokeny JWT. [43]

Klíčem k úspěšné obraně je striktní vynucování přístupových pravidel výhradně na straně serveru. Standardem by mělo být nastavení „implicitního zákazu“ (*deny by default*), kdy je přístup k jakémukoliv zdroji s výjimkou veřejných stránek nejprve blokován. Moderní aplikace by měly využívat prověřená rozhraní pro správu oprávnění, vynucovat striktní vlastnictví datových záznamů a omezovat automatizované útoky pomocí limitování četnosti požadavků na API. [43]

Nesprávná konfigurace zabezpečení

Tato kategorie se v aktuálním žebříčku posunula na druhou pozici, přičemž určitá forma chybné konfigurace byla zaznamenána u všech testovaných aplikací. Riziko narůstá zejména s přechodem na vysoce konfigurovatelný software a cloudové služby. Podstatou zranitelnosti je nesprávné nebo neúplné nastavení bezpečnostních prvků napříč celým technologickým zásobníkem, což útočníkům otevírá prostor pro kompromitaci systému. [44]

Aplikace bývají zranitelné především tehdy, pokud jsou ponechány aktivní výchozí účty s původními hesly nebo jsou povoleny nadbytečné funkce, porty a služby, které v produkčním prostředí nemají opodstatnění. Kritickým nedostatkem je také absence centrálního ošetření chyb, kdy systém uživatelům (a potenciálním útočníkům) odhaluje podrobné informace, jako jsou výpisy zásobníku (*stack traces*) nebo verze komponent. K oslabení bezpečnosti přispívá i nedostatečné zasílání bezpečnostních hlaviček v HTTP odpovědích nebo přehnaná orientace na zpětnou kompatibilitu na úkor aktuálních ochranných prvků. [44]

Prevence vyžaduje zavedení opakovatelného a automatizovaného procesu posilování bezpečnosti (*hardening*), který zajistí, že vývojová, testovací i produkční prostředí budou nakonfigurována identicky a bezpečně. Doporučuje se provozovat minimální možnou platformu bez zbytečných ukázkových aplikací a dokumentace. Nezbytnou sou-

částí ochrany je segmentace architektury (např. pomocí kontejnerizace nebo cloudových ACL), používání krátkodobých přihlašovacích údajů namísto statických klíčů v konfiguracích a pravidelné ověřování účinnosti nastavení prostřednictvím automatizovaných nástrojů. [44]

Selhání v dodavatelském řetězci softwaru

Tato kategorie představuje jednu z nejkritičtějších hrozeb současnosti, což potvrzuje i fakt, že ji polovina respondentů v komunitním průzkumu zařadila na první místo. Riziko se již neomezuje pouze na používání komponent se známými zranitelnostmi, ale zahrnuje jakékoliv narušení procesu sestavování, distribuce nebo aktualizace softwaru. Přestože je identifikace těchto selhání náročná, v testovaných datech vykazuje tato kategorie nejvyšší průměrnou míru výskytu, a to přes 5 %. [45]

Organizace jsou zranitelné především v případech, kdy postrádají přehled o verzích všech používaných prvků, včetně vnořených závislostí, nebo pokud využívají neudržované a zastaralé knihovny. Značné nebezpečí představuje absence procesů pro řízení změn v rámci celého řetězce, který zahrnuje nejen kód, ale i vývojářské nástroje a repozitáře. Pokud mají systémy v dodavatelském řetězci slabší zabezpečení než samotné výsledné aplikace, nebo pokud chybí důsledné oddělení povinností, stávají se snadným cílem pro útočníky. [45]

Prevence vyžaduje zavedení robustního procesu správy záplat a centrální evidenci softwarových komponent prostřednictvím dokumentu SBOM (*Software Bill of Materials*). Je nezbytné kontinuálně monitorovat bezpečnostní bulletiny (např. CVE či NVD) a automatizovat analýzu složení softwaru. Mezi klíčová opatření patří využívání pouze oficiálních a podepsaných balíčků, posílení bezpečnosti vývojářských stanic pomocí MFA a striktní izolace prostředí. Doporučuje se také nasazovat aktualizace po etapách (*canary deployments*), aby se minimalizoval dopad v případě kompromitace důvěryhodného dodavatele. [45]

Kryptografická selhání

Tato slabina se v aktuálním žebříčku posunula na čtvrtou pozici a primárně se zaměřuje na rizika spojená s absencí šifrování, použitím slabých algoritmů nebo únikem kryptografických klíčů. Moderní webové aplikace musí chránit citlivá data nejen během přenosu po síti, ale také při jejich ukládání. Výrazným problémem v této oblasti zůstává využívání predikovatelných generátorů náhodných čísel (PRNG) s nízkou entropií, což útočníkům usnadňuje odhadnutí šifrovacích klíčů nebo jiných bezpečnostních tokenů. [46]

Aplikace jsou považovány za zranitelné, pokud využívají zastaralé protokoly, jako je prosté HTTP namísto vynuceného šifrovaného spojení, nebo pokud pracují s expirovanými a slabými certifikáty bez řádné validace celého řetězce důvěry. Kritickým pochybením je rovněž ukládání hesel v čitelném formátu nebo s využitím slabých a neosolených hashovacích funkcí, jako jsou MD5 či SHA1. Zvláštní důraz je kladen na ochranu specifických dat, jako jsou zdravotní záznamy, čísla platebních karet či osobní údaje spadající pod regulace typu GDPR, které vyžadují dodatečné šifrování i na aplikační vrstvě. [46]

Základem prevence je důsledná klasifikace zpracovávaných dat a aplikace ochranných prvků odpovídajících jejich citlivosti. Veškerá komunikace musí být šifrována pomocí protokolu TLS verze 1.2 nebo vyšší s preferencí moderních šifer zajišťujících dopředné tajemství (*forward secrecy*). Pro ukládání hesel je nezbytné používat silné adaptivní hashovací funkce, například Argon2 nebo scrypt, které efektivně ztěžují útoky hrubou silou. Mezi další nezbytná opatření patří ukládání šifrovacích klíčů v dedikovaných hardwarových či cloudových modulech (HSM), vynucování HTTPS pomocí hlavičky HSTS a včasná příprava systémů na přechod k post-kvantové kryptografii (PQC). [46]

Útoky typu injection

Skupina zranitelností jako je tato se v aktuálním žebříčku nachází na pátém místě, přestože patří k nejtestovanějším oblastem, určitou formou vkládání kódu trpí v podstatě každá prověřovaná aplikace. Podstatou problému je selhání aplikace při zpracování nedůvěryhodných vstupů, které jsou následně odeslány interpretu (např. databázi, operačnímu systému nebo prohlížeči) jako součást příkazu. Útočník tak může pomocí speciálně upravených dat oklamat systém a vynutit si provedení neautorizovaných instrukcí. Tato kategorie je velmi obsáhlá a zahrnuje jak SQL či NoSQL injection s vysokým dopadem, tak i velmi častý Cross-site Scripting (XSS). [47]

Aplikace se stává zranitelnou zejména v momentě, kdy jsou uživatelská data přímo spojována do dynamických dotazů bez předchozí validace nebo sanitace. Riziko představuje také slepá důvěra ve frameworky (např. ORM nástroje), kde nesprávně parametrizované volání může stále vést k úniku citlivých záznamů. Kromě databázových dotazů mohou útoky cílit i na příkazovou řádku operačního systému, LDAP dotazy nebo výrazové jazyky (Expression Language), což v případě úspěchu umožňuje útočníkovi vykonávat libovolný kód přímo na serveru [47]

Nejúčinnější prevence spočívá v důsledném oddělení dat od samotných příkazů. Prioritní volbou by mělo být používání bezpečných API s parametrizovaným rozhraním, která nativně znemožňují změnu logiky dotazu prostřednictvím vstupu. Pokud nelze data a příkazy zcela oddělit, je nezbytné implementovat striktní validaci vstupů na

straně serveru (allow-listing) a používat kontextově orientované kódování speciálních znaků pro konkrétní interpret. Pro včasnou detekci těchto vad je doporučeno integrovat nástroje pro statickou a dynamickou analýzu (SAST/DAST) přímo do CI/CD řetězce. [47]

Nebezpečný návrh

Kategorie *Nebezpečný návrh* se zaměřuje na rizika spojená s architektonickými nedostatky a hlubokými logickými chybami v samotné koncepci aplikace. Zásadním specifikem těchto vad je skutečnost, že je nelze napravit pouhou změnou implementace nebo dodatečnou konfigurací, pokud v systému od počátku absentují nezbytné bezpečnostní kontroly určené k obraně proti konkrétním útokům. Aktuální šestá pozice v žebříčku reflektuje rostoucí význam metodik jako, které kladou důraz na bezpečnost již v raných fázích vývoje. [48]

Kritickým faktorem přispívajícím k nebezpečnému návrhu je absence profilování obchodních rizik, což vede k neschopnosti určit potřebnou úroveň zabezpečení pro vyvíjený systém. Mezi typické slabiny patří selhání při správě privilegií, nechráněné ukládání citlivých údajů nebo chybějící definice nežádoucích změn stavu v rámci obchodní logiky. Příkladem může být proces obnovy hesla využívající nespolehlivé kontrolní otázky nebo absence ochrany proti automatizovaným botům při nákupních transakcích, což útočníkům umožňuje zneužít samotnou podstatu fungování aplikace. [48]

Účinná prevence vyžaduje zavedení bezpečného životního cyklu vývoje softwaru (S-SDLC), který integruje bezpečnostní specialisty již do fáze sběru požadavků a plánování. Nezbytností je pravidelné využívání modelování hrozeb pro kritické části aplikace, jako je autentizace, řízení přístupu a klíčové datové toky. Vývojáři by měli využívat knihovnu osvědčených návrhových vzorů, integrovat bezpečnostní kontroly přímo do uživatelských příběhů a provádět důslednou segmentaci vrstev systému na aplikační i síťové úrovni. Robustní ochrana rovněž zahrnuje psaní jednotkových a integračních testů pro ověření odolnosti všech kritických procesů proti identifikovaným hrozbám. [48]

Selhání autentizace

Tato oblast rizik se zaměřuje na situace, kdy útočník dokáže oklamat systém, aby uznal neoprávněnou identitu jako legitimní. I přes rozmach standardizovaných frameworků si tato kategorie drží svou pozici v žebříčku, což reflektuje přetrvávající potíže při správě uživatelských identit a relací. Problém nastává především tehdy, pokud aplikace dovoluje automatizované útoky typu *credential stuffing* nebo *password spray*, při kterém se útočníci pokoušejí o přístup pomocí běžných variací hesel. [49]

Slabiny v ověřování se projevují zejména povolením triviálních či výchozích přístupových údajů a používáním neúčinných procesů pro obnovu zapomenutého hesla, jako jsou nespolehlivé znalostní otázky. Významné nebezpečí představuje také nesprávná správa relací, kdy systém po odhlášení korektně nezneplatní tokeny nebo vystavuje identifikátor relace v URL adrese, což usnadňuje jejich zneužití. Moderní hrozbou jsou rovněž hybridní útoky, které zneužívají lidské chování k predikci změn v heslech pro nadcházející období. [49]

Klíčovým pilířem obrany je plošné prosazování vícefaktorového ověřování (MFA), které účinně blokuje většinu automatizovaných útoků založených na zcizených údajích. Doporučuje se eliminovat používání výchozích hesel, implementovat kontroly proti seznamům nejčastěji zneužívaných přihlašovacích údajů a sjednotit chybové zprávy tak, aby nedocházelo k prozrazení existence konkrétních uživatelských účtů (*account enumeration*). Pro správu relací je nezbytné využívat vestavěné serverové manažery s vysokou entropií identifikátorů a zajistit jejich okamžitou invalidaci po odhlášení či delší nečinnosti. Ideální strategií je delegování správy identit na prověřené a specializované systémy, čímž se minimalizuje riziko chyby v lokální implementaci. [49]

Selhání integrity softwaru nebo dat

Udržení důvěryhodných hranic mezi aplikací a externími zdroji je hlavním tématem této kategorie, která se zaměřuje na chyby při ověřování integrity kódu a datových artefaktů. Riziko spočívá především v nekritickém přijímání softwarových aktualizací nebo knihoven bez potvrzení, že pocházejí z legitimního zdroje a nebyly cestou modifikovány. Tato hrozba úzce souvisí s bezpečností dodavatelského řetězce, avšak zaměřuje se na nižší technickou úroveň validace jednotlivých objektů a infrastrukturních prvků. [50]

Kritické zranitelnosti vznikají zejména v situacích, kdy se aplikace spoléhá na pluginy či moduly z nedůvěryhodných repozitářů a sítí CDN, nebo pokud automatické aktualizace probíhají bez digitálního podpisu. Velmi nebezpečným projevem je nebezpečná deserializace, kdy útočník upraví serializovaná data (např. v souborech cookies nebo API požadavcích) tak, aby po jejich opětovném načtení vynutil spuštění škodlivého kódu na serveru. Nedostatečná integrita se může projevit i v CI/CD pipeline, pokud systém stahuje artefakty z neprověřených míst bez kontroly jejich kontrolních součtů. [50]

Obrana proti těmto selháním vyžaduje důsledné využívání digitálních podpisů k ověření původu veškerého softwaru a dat. Vývojové týmy by měly konzumovat knihovny pouze z důvěryhodných a ideálně interně prověřených repozitářů. Nezbytností je zavedení striktního procesu revize změn v kódu i konfiguraci a zajištění, aby CI/CD proces

disponoval adekvátním řízením přístupu pro ochranu integrity celého řetězce. V případě přenosu serializovaných dat od klientů je nutné buď digitálně podepisovat samotná data pro detekci neoprávněné manipulace, nebo se serializaci nedůvěryhodných objektů zcela vyhnout. [50]

Selhání protokolování a monitorování

Absence včasné detekce a reakce na incidenty představuje zásadní slabinu, která v aktuálním žebříčku obhájila svou devátou pozici. Tato kategorie prošla drobnou změnou názvu, aby se zdůraznila nezbytnost aktivního varování, které je klíčové pro rychlou odezvu během bezpečnostního incidentu. Bez efektivního monitorování mohou úniky dat probíhat bez povšimnutí i několik let, což v konečném důsledku znemožňuje forenzní analýzu a prohlubuje dopady útoku. [51]

Zranitelnost systému se projevuje především tehdy, pokud nejsou konzistentně protokolovány kritické události, jako jsou neúspěšná přihlášení či transakce s vysokou hodnotou. Riziko představuje také nedostatečná ochrana integrity samotných záznamů před neoprávněnou manipulací nebo jejich ukládání pouze lokálně bez zálohování. Častým pochybením je rovněž generování nejasných chybových zpráv nebo situace, kdy aplikace nedokáže detekovat aktivní skenování nástroji typu DAST v reálném čase. Navíc hrozí únik citlivých údajů, pokud jsou tyto informace neodborně zahrnuty přímo do obsahu protokolů. [51]

Obrana vyžaduje, aby vývojáři zajistili záznam všech selhání bezpečnostních kontrol s dostatečným kontextem pro identifikaci podezřelých účtů. Protokoly musí být generovány v dobře zpracovatelném formátu pro řešení centralizované správy logů a musí být chráněny proti pozměnění, například pomocí append-only databázových tabulek. Nezbytností je zavedení prahových hodnot pro varování a vytvoření jasných scénářů reakce pro týmy SecOps. Moderním přístupem je rovněž nasazení tzv. *honeytokens* – návnad, jejichž jakýkoliv přístup okamžitě generuje výstrahu s minimem falešných poplachů. [51]

Nesprávné ošetření výjimečných stavů

Tato kategorie představuje novinku pro rok 2025 a zaměřuje se na specifické hrozby plynoucí z nekorektního zpracování chyb, logických vad a situací, kdy systém v případě anomálie přejde do nezabezpečeného stavu. Nahrazuje dřívější, příliš obecné klasifikace kvality kódu a poskytuje vývojářům přesnější vodítka pro řešení abnormálních podmínek, se kterými se software může setkat. Průměrná míra výskytu těchto selhání v testovaných aplikacích dosahuje téměř 3 %. [52]

K narušení bezpečnosti dochází v momentě, kdy program nedokáže adekvátně zabránit, detekovat nebo reagovat na nepředvídatelné situace, což vede k pádům aplikace či neočekávanému chování. Mezi kritické zranitelnosti patří generování chybových hlášení obsahujících citlivé systémové informace, selhání při zpracování chybějících parametrů nebo tzv. „failing open“, kdy systém při chybě neúmyslně uvolní přístupová omezení. Útočníci mohou tyto nedostatky zneužít k vyvolání stavu *Denial of Service* skrze vyčerpání systémových zdrojů, k průzkumu infrastruktury nebo k manipulaci s finančními transakcemi, které nejsou v případě chyby správně vráceny do původního stavu. [52]

Správné ošetření vyžaduje, aby vývojáři plánovali nejhorší možné scénáře a zachytávali veškeré systémové chyby přímo v místě jejich vzniku. Klíčovou zásadou je „fail closed“ v případě jakéhokoliv přerušení transakce musí dojít k jejímu úplnému vrácení zpět (*roll-back*), aby se předešlo nekonzistenci dat. Obrana zahrnuje nasazení globálních obslužných rutin pro neočekávané výjimky, striktní validaci vstupů a zavedení limitů pro čerpání prostředků, které brání zneužití chyb k zahlcení serveru. Celá organizace by měla sjednotit způsob správy výjimečných stavů a využívat monitorovací nástroje ke sledování opakujících se chyb, které mohou signalizovat probíhající útok. [52]

5 Analýza současného stavu

V současné době existuje na trhu široké spektrum digitálních nástrojů a aplikací zaměřených na podporu zdravého životního stylu. Při detailnějším průzkumu dostupných řešení je však patrné, že většina těchto platforem je úzce profilovaná na jednu specifickou oblast. Uživatelé mají k dispozici vysoce pokročilé aplikace pro monitorování fyzické aktivity (např. *Strava*, *Garmin Connect*) nebo specializované nástroje pro evidenci kalorického příjmu a makroživin (např. *Kalorické tabulky*, *MyFitnessPal*).

Přestože jsou tyto izolované aplikace efektivní ve svých dílčích oblastech, z pohledu komplexní preventivní péče uživateli chybí jednotný ekosystém. Pokud chce jednotlivec sledovat svůj životní styl celostně, tedy kombinovat vliv stravy, pohybu a dalších biometrických metrik, je nucen využívat několik navzájem nepropojených platforem. Tento stav vede k roztříštěnosti dat, ztrátě širšího kontextu a často i k poklesu motivace z důvodu nutnosti duplicitního zadávání informací. Z uživatelského hlediska je velmi obtížné hledat korelace mezi tím, jak se člověk stravuje, a jaké podává fyzické výkony, pokud jsou tato data uzamčena ve dvou různých systémech.

Právě tato identifikovaná mezera na trhu je hlavním motivem pro vývoj této aplikace. Cílem navrhovaného řešení není konkurovat úzce specializovaným platformám v množství detailních funkcí, ale poskytnout ucelený nástroj, který integruje klíčové aspekty zdravého životního stylu do jednoho srozumitelného uživatelského rozhraní. Aplikace má za cíl agregovat data z různých oblastí, vyhodnocovat je v širším kontextu a poskytovat uživateli centralizovanou a snadno interpretovatelnou zpětnou vazbu.

6 Návrh systému

Tato kapitola se věnuje technické specifikaci a koncepčnímu návrhu webové aplikace *Life Essentials*. Návrh vychází z teoretických poznatků definovaných v předchozích kapitolách a transformuje je do konkrétních požadavků na softwarové dílo. Cílem je navrhnout systém, který uživateli poskytne nejen nástroje pro sběr dat, ale především pro jejich interpretaci v kontextu dlouhodobého zdraví.

6.1 Cílová skupina

Správná definice cílové skupiny je nezbytná pro volbu vhodného uživatelského rozhraní a tónu komunikace aplikace. Aplikace cílí na dva primární segmenty:

- **Osoby se zájmem o kvantifikaci zdraví:** Uživatelé, kteří jsou zvyklí využívat technologie k monitorování svých životních funkcí a stravy. Očekávají detailní data a grafické výstupy.
- **Jednotlivci motivovaní prevencí:** Široká dospělá populace, která si uvědomuje rizika civilizačních chorob a chce aktivně ovlivnit svůj „biologický věk“ a délku dožití změnou životního stylu.

6.2 Funkční a nefunkční požadavky

Definice požadavků na systém představuje klíčovou fázi vývoje, která transformuje představy o cílech aplikace do konkrétní technické specifikace. Na základě hloubkové analýzy potřeb cílové skupiny a syntézy teoretických poznatků z oblasti zdravého životního stylu byly stanoveny následující požadavky. Ty jsou v souladu se standardy softwarového inženýrství rozděleny na funkční a nefunkční.

6.2.1 Funkční požadavky

Funkční požadavky definují, jaké konkrétní služby musí systém poskytovat:

- **F1 – Správa uživatelské identity:** Systém umožní registraci nového uživatele, bezpečné přihlášení a správu profilu. Součástí profilu musí být biometrické údaje (věk, váha, výška, pohlaví, úroveň aktivity) nezbytné pro výpočty.
- **F2 – Nutriční monitoring:** Aplikace poskytne rozhraní pro vyhledávání potravin v databázi, jejich přidávání do denního přehledu a automatický výpočet přijatých makroživin (bílkoviny, tuky, sacharidy) a energie (kcal).

- **F3 – Kalkulace metabolických metrik:** Systém bude na základě profilu uživatele automaticky počítat index tělesné hmotnosti (BMI) a bazální metabolismus (BMR).
- **F4 – Analýza životního stylu:** Funkce pro zadání návyků (kouření, konzumace alkoholu, spánek) a výpočet jejich dopadu na zkrácení či prodloužení předpokládané délky života.

6.2.2 Nefunkční požadavky

Nefunkční požadavky definují technické parametry a kvalitativní vlastnosti systému:

- **N1 – Bezpečnost dat:** Hesla uživatelů nesmí být ukládána v otevřené formě, ale musí být zabezpečena silným hašovacím algoritmem.
- **N2 – Architektura:** Aplikace bude postavena na modulární architektuře (využití *Blueprints* ve frameworku Flask) pro snadnou rozšiřitelnost.
- **N3 – Responzivita:** Uživatelské rozhraní musí být plně funkční na mobilních zařízeních i desktopových monitorech (*Mobile First*).
- **N4 – Perzistence:** Data musí být uchovávána v relační databázi s důrazem na integritu vazeb mezi uživatelem a jeho záznamy.

6.3 Modelování systému

Pro formální vizualizaci funkčních požadavků a interakcí uživatelů se systémem byl zvolen standardizovaný jazyk UML (Unified Modeling Language). Modelování v této fázi slouží jako most mezi definovanými požadavky a následnou implementací, přičemž důraz je kladen na identifikaci klíčových procesů a rolí v systému.

6.3.1 Diagram případů užití

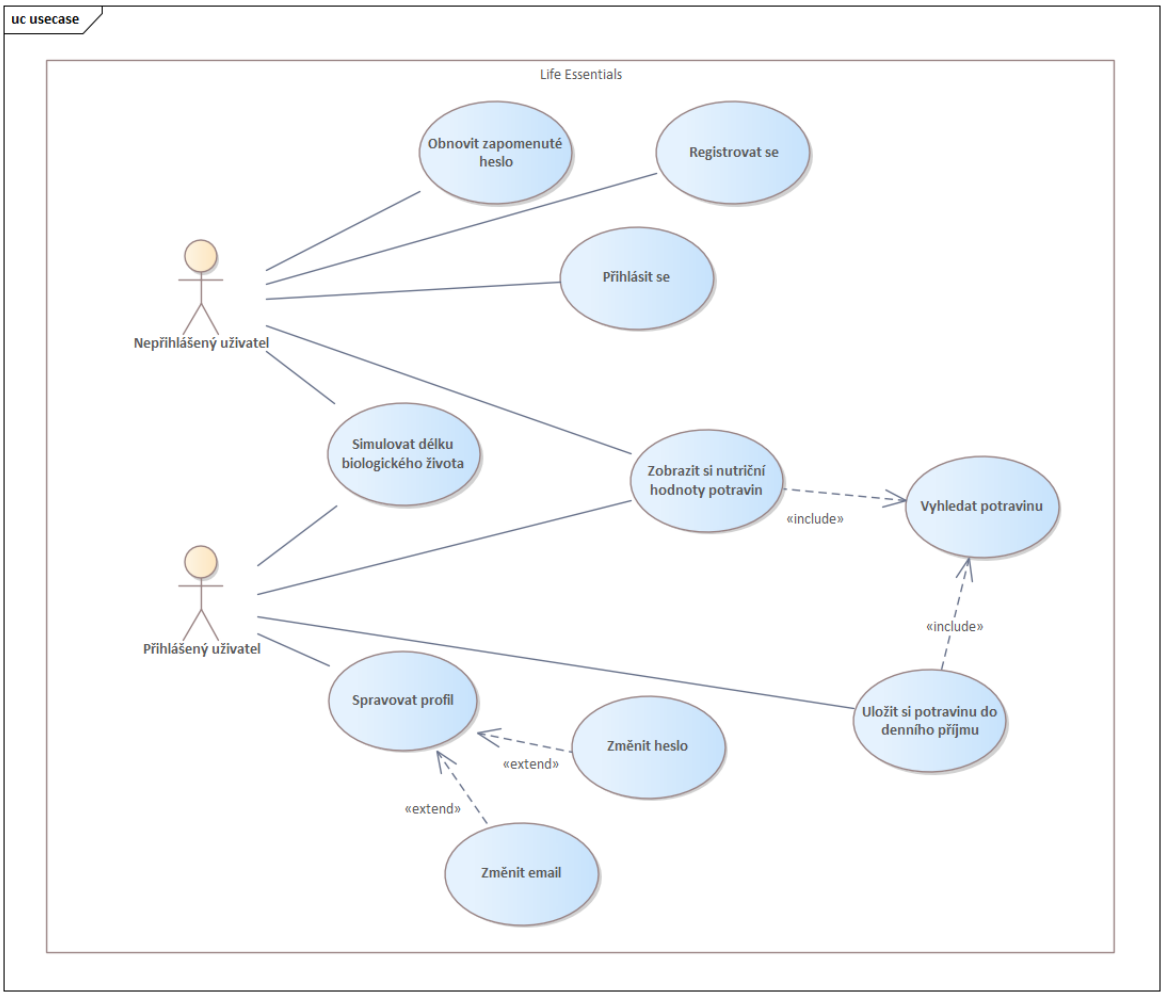
Navržený diagram (viz Obrázek 1) definuje rozsah funkčnosti aplikace a specifikuje oprávnění pro jednotlivé role.

Definice aktérů a hierarchie: Systém rozlišuje dvě úrovně oprávnění, které jsou v modelu vyjádřeny vztahem dědičnosti:

- **Nepřihlášený uživatel:** Reprezentuje anonymního, neregistrovaného uživatele. Tento aktér má přístup k veřejným částem aplikace, jako je registrace, přihlášení či obnova zapomenutého hesla. Klíčovou vlastností je možnost využívat simulační

nástroje v omezeném režimu, kdy uživatel neukládá data trvale, ale zadává je ad-hoc pro jednorázový výpočet.

- **Přihlášený uživatel:** Představuje autentizovaného uživatele, který úspěšně prošel procesem přihlášení. Tento aktér sdílí s nepřihlášeným uživatelem volně dostupné funkce systému, avšak získává navíc přístup k personalizovaným nástrojům. Těmi jsou především správa vlastního uživatelského profilu a možnost perzistentního ukládání záznamů do osobního stravovacího deníku.



Obrázek 1: Diagram případů užití

Zdroj: vlastní zpracování

Specifikace klíčových vazeb a modulů: Diagram využívá pokročilých vazeb `«include»` a `«extend»` pro přesný popis chování aplikace v konkrétních scénářích:

- **Nutriční modul:** Případ užití *Zobrazit si nutriční hodnoty potravin* logicky vyžaduje interakci s databází potravin. Tato povinná závislost je modelována relací `«include»` směrem k případu *Vyhledat potravinu*. Tuto funkční komponentu sdílí i nepřihlášený návštěvník, avšak bez možnosti finálního uložení do příjmu.

- **Správa identity:** Základní případ užití *Spravovat profil* představuje komplexní proces, který může být volitelně rozšířen o specifické bezpečnostní úkony. Tyto volitelné kroky, jako je *Změna hesla* nebo *Změna e-mailu*, jsou k základnímu procesu připojeny vazbou `«extend»`, jelikož nejsou podmínkou pro každou editaci profilu.
- **Longevity modul:** Případ užití *Simulovat délku biologického života* je dostupný oběma aktérům, avšak liší se vstupními daty. Zatímco návštěvník musí parametry zadávat manuálně, pro přihlášeného uživatele systém automaticky inicializuje výpočet na základě dlouhodobě sledovaných biometrických dat z jeho profilu, což zvyšuje uživatelský komfort.

6.4 Architektura a datový model

Aplikace je navržena jako server-side webová aplikace využívající architektonický vzor **MVT (Model-View-Template)**, který je nativní pro framework Flask.

6.4.1 Architektura aplikace

Systém je rozdělen do logických vrstev:

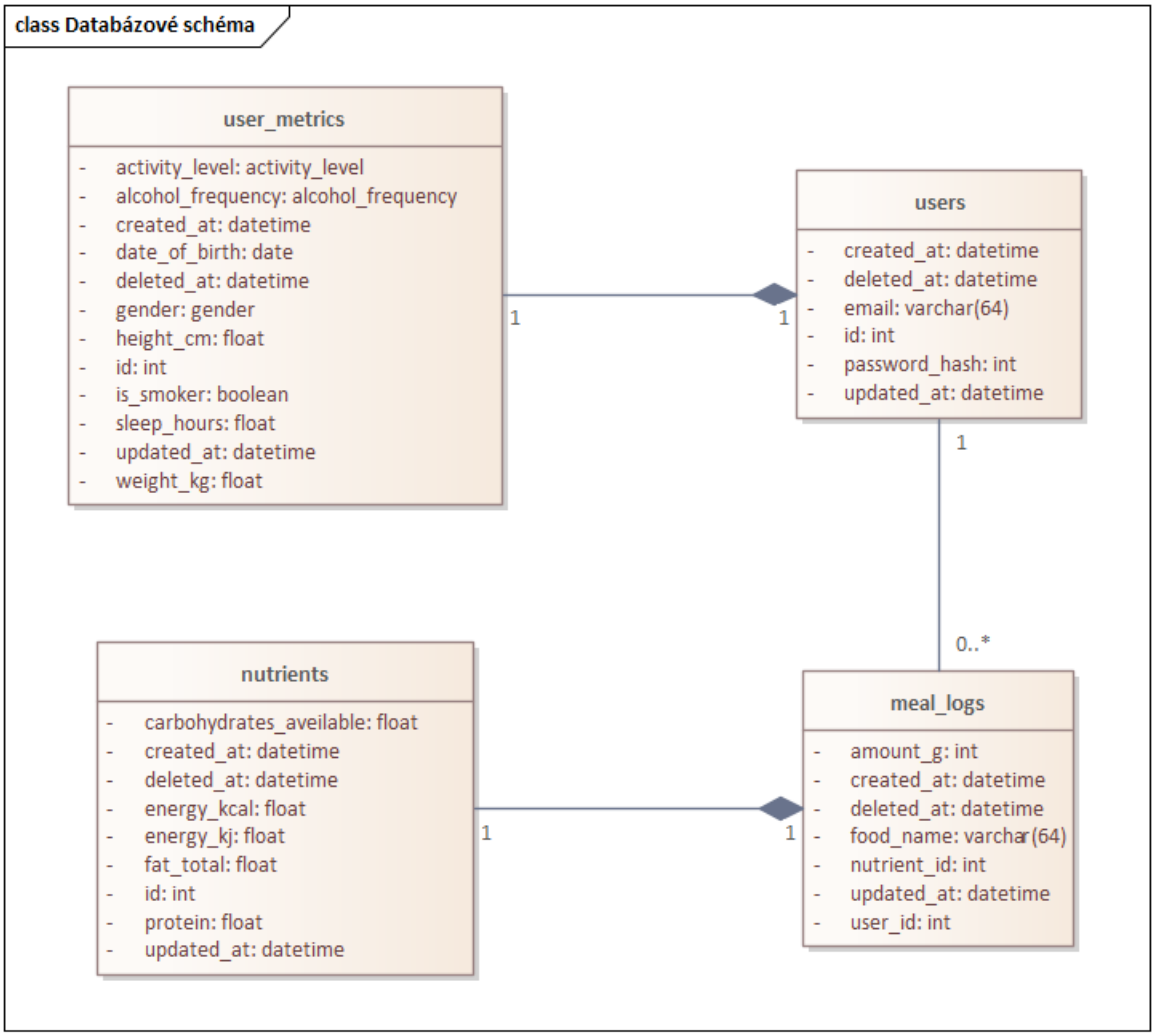
1. **Prezentační vrstva):** Tvořena HTML šablonami renderovanými pomocí enginu *Jinja2*. O vzhled a responzivitu se stará CSS framework Bootstrap, dynamiku na straně klienta zajišťuje JavaScript (jQuery, Chart.js).
2. **Aplikační vrstva:** Python moduly zpracovávající HTTP požadavky. Zajišťují validaci vstupů, volání výpočetní logiky a komunikaci s databází.
3. **Datová vrstva:** Využívá ORM knihovnu SQLAlchemy, která abstrahuje práci s SQL databází do podoby Python objektů.

6.4.2 Datový model (ERD)

Datová vrstva aplikace je postavena na relačním modelu, který důsledně odděluje autentizační údaje od provozních dat. Model využívá principu kompozice pro zajištění integrity dat při odstranění uživatele nebo záznamu (viz Obrázek 2).

Popis entit:

- `users`: Základní entita zajišťující identitu. Obsahuje pouze nezbytné údaje pro autentizaci (`email`, `password_hash`).



Obrázek 2: Class Diagram databázového modelu

Zdroj: vlastní zpracování

- `user_metrics`: Uchovává citlivá osobní data (věk, váha, výška) odděleně od přihlašovacích údajů.
- `meal_logs`: Hlavní transakční tabulka reprezentující konzumaci pokrmu. Obsahuje název jídla, čas konzumace a množství (`amount_g`).
- `nutrients`: Tabulka uchovávající konkrétní nutriční hodnoty (makroživiny, vitamíny) vypočítané pro danou porci.

Relace v modelu:

- **users ↔ user_metrics:** Kompozitní vazba. Každý uživatel vlastní právě jednu sadu aktuálních biometrických údajů.
- **users ↔ meal_logs:** Kompozitní vazba. Jeden uživatel vytváří v čase historii mnoha záznamů. Při smazání uživatele zaniká i jeho deník.

- **meal_logs ↔ nutrients:** Kompozitní vazba realizující uložení vypočítaných dat. Každý záznam v deníku vlastní právě jednu sadu unikátních nutričních hodnot platných pro danou porci.

Auditní metadata a životní cyklus dat

Všechny entity v datovém modelu obsahují sadu systémových atributů, které zajišťují sledovatelnost změn a bezpečnost dat:

- **Auditní stopy (`created_at`, `updated_at`):** Tyto atributy automaticky zaznamenávají časové razítko vytvoření záznamu a jeho poslední modifikace. To umožňuje zpětnou analýzu aktivity uživatele a debugování.
- **Měkké mazání (`deleted_at`):** Z důvodu ochrany před nechtěnou ztrátou dat aplikace nevyužívá fyzické odstranění záznamů (*Hard Delete*). Místo toho je implementován mechanismus tzv. *Soft Delete*. Při požadavku na smazání je pouze nastaveno časové razítko do sloupce `deleted_at`. Záznam se následně v aplikaci nezobrazuje, ale fyzicky v databázi zůstává, což umožňuje jeho případnou obnovu.

6.5 Návrh uživatelského rozhraní

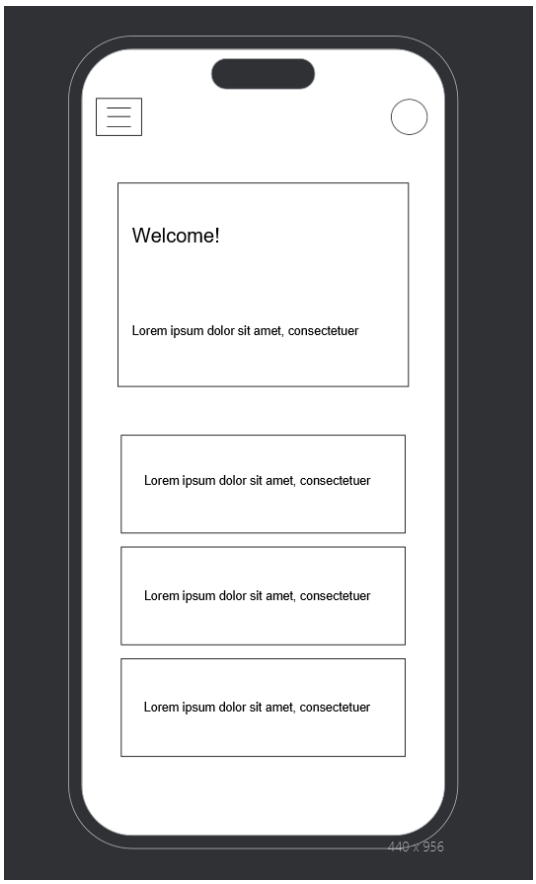
Návrh uživatelského rozhraní byl koncipován s důrazem na minimalismus a přehlednost, aby aplikace neodváděla pozornost uživatele od prezentovaných dat. Designový jazyk využívá principů *Mobile First*, jelikož se předpokládá časté zadávání dat z mobilního telefonu, avšak je plně optimalizován i pro desktopová rozlišení.

Pro ověření ergonomie ovládání byly nejprve vytvořeny nízkoúrovňové drátěné modely, které definují rozložení prvků a navigaci.

6.5.1 Mobilní zobrazení (Mobile First)

Vzhledem k omezené šířce displeje mobilních telefonů řadí návrh všechny klíčové prvky vertikálně pod sebe (viz Obrázek 3). Prioritu má okamžitý přehled stavu uživatele.

- **Navigace:** Je řešena formou skrytého menu (Hamburger menu) nebo spodní lišty, aby nezabírala cenné místo na obrazovce.
- **Obsah:** Dominantním prvkem je uvítací karta s rychlým přehledem, následovaná kartami jednotlivých modulů.



Obrázek 3: Wireframe mobilního dashboardu

Zdroj: vlastní zpracování

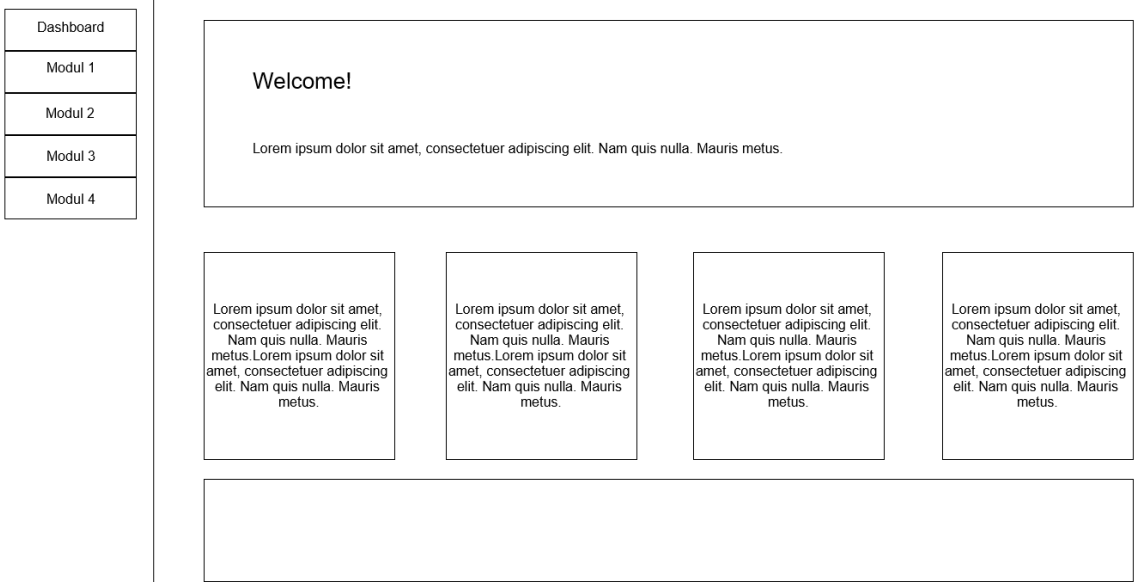
6.5.2 Adaptace pro desktop

Pro větší obrazovky (tablet, desktop) dochází k transformaci layoutu (viz Obrázek 4). Návrh využívá responzivního mřížkového systému(Grid):

- **Sidebar:** Navigace se přesouvá do trvale viditelného levého postranního panelu, což urychluje přechod mezi sekcemi.
- **Grid Layout:** Informační karty jednotlivých modulů nejsou řazeny pod sebe, ale vedle sebe. To umožňuje uživateli vidět souvislosti mezi metrikami (např. vliv spánku na aktivitu) na jeden pohled bez nutnosti posouvání stránky.

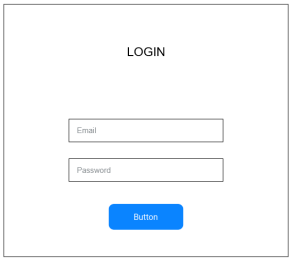
6.5.3 Autentizační rozhraní

Vstupní bod do aplikace tvoří přihlašovací obrazovka (viz Obrázek 5). Návrh je záměrně oproštěn od jakýchkoliv rušivých elementů (navigace, postranní panely), aby se uživatel soustředil pouze na proces autentizace.



Obrázek 4: Wireframe desktopového rozhraní

Zdroj: vlastní zpracování



Obrázek 5: Návrh přihlašovací obrazovky

Zdroj: vlastní zpracování

Konzistence rozhraní

Uvedené drátěné modely Dashboardu slouží jako referenční šablona (tzv. Master Layout) pro celou aplikaci. Ostatní funkční obrazovky, jako jsou jednotlivé moduly či správa profilu, dědí tuto globální navigační strukturu a postranní panel, přičemž do vyhrazené hlavní části stránky pouze vkládají svůj specifický obsah. Díky tomuto principu je zachována vizuální konzistence napříč celým webem, a proto nejsou další dílčí pohledy v textu duplicitně rozkreslovány.

7 Implementace

Tato kapitola se věnuje technické realizaci aplikace *Life Essentials* a popisuje transformaci navrženého modelu do podoby funkčního softwarového díla. Cílem implementační fáze bylo vytvořit stabilní, bezpečný a uživatelsky přívětivý systém, který naplňuje všechny definované požadavky.

7.1 Použité technologie a prostředky

Výběr technologického zásobníku byl podřízen dvěma klíčovým kritériím. Primárním faktorem byla vhodnost daných nástrojů pro vývoj škálovatelné webové aplikace s důrazem na rychlé prototypování a snadnou údržbu. Druhým, neméně významným aspektem, byly dosavadní zkušenosti autora s vybranými jazyky a frameworky. Realizace této práce tak sloužila nejen k vytvoření samotné aplikace, ale rovněž jako prostředek k širšímu rozvinutí těchto dovedností a osvojení pokročilých konceptů, jako jsou návrhové vzory, bezpečnostní mechanismy či práce s ORM vrstvou, které navazují na základy získané během studia.

Aplikace je postavena na ověřené architektuře klient-server. Prezentační vrstva, zajišťující interakci s uživatelem, vychází ze standardů **HTML5** a **CSS3**. Pro zajištění responzivity a vizuální konzistence napříč zařízeními byl implementován framework **Bootstrap 5**, který doplňuje skriptovací jazyk **JavaScript** spolu s knihovnou **jQuery** pro obsluhu dynamických prvků na straně klienta.

Aplikační logiku na straně serveru obsluhuje jazyk **Python** s využitím mikroframeworku **Flask**. Ten zajišťuje směrování požadavků a komunikaci s databází, přičemž generování výsledného HTML kódu probíhá prostřednictvím šablonovacího systému **Jinja2**. O trvalé uložení a integritu dat se stará relační databázový systém **PostgreSQL**. Systém je navíc otevřený komunikaci s externími službami, kdy aplikace využívá **REST API** pro získávání nutričních dat z externích zdrojů.

7.1.1 HTML5 a šablonovací systém Jinja2

Jazyk HTML5 tvoří sémantickou kostru celé aplikace. V prostředí frameworku Flask však není HTML kód psán staticky pro každou stránku zvlášť. Místo toho je využit šablonovací systém *Jinja2*, který umožňuje aplikovat princip dědičnosti šablon (Template Inheritance) a dynamické propojení s backendem.

Hierarchie šablon a rodičovská struktura

Pro zajištění konzistentního vzhledu byly navrženy dvě rodičovské šablony. Hlavní šablonou je `base.html`, která definuje společnou kostru pro většinu stránek. Jak je vidět na Obrázku 6, tato šablona obsahuje definici hlavičky (`<head>`) a navigačních prvků (postranní panel, horní lišta), které se automaticky promítají do všech podstránek.

Pro autentizační procesy existuje odlehčená varianta `base_without_navs.html`, která je záměrně zbavena navigace, aby uživatele nerozptylovala.

```
    <div class="container">
      {% with messages = get_flashed_messages(with_categories=true) %} {% if
      messages %} {% for category, message in messages %}
      <div
        class="alert alert-{{ 'danger' if category == 'error' else category }} alert-dismissible fade show"
        role="alert"
      >
        {{ message }}
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="alert"
          aria-label="Close"
        ></button>
      </div>
      {% endfor %} {% endif %} {% endwith %}
    </div>
    {% block body %}{% endblock %}
  </main>

  <script src="../static/js/jquery-3.7.1.min.js"></script>
  <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
    defer
  ></script>
  <script src="../static/js/chart.umd.min.js" defer></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="../static/js/datatables.min.js" defer></script>

  {% for js_file in js %}
  <script src="../static/js/{{ js_file }}" defer></script>
  {% endfor %}
 </body>
</html>
```

Obrázek 6: Ukázka struktury hlavní šablony base.html (navigační prvky)

Zdroj: vlastní zpracování

Dynamická data a optimalizace zdrojů

Klíčovou vlastností Jinja2 je schopnost vkládat do HTML kódu data přímo z aplikační logiky. Pomocí syntaxe složených závorek `{{ proměnná }}` se v reálném čase vypisují informace o přihlášeném uživateli (např. jméno v hlavičce) nebo chybové hlášky.

Zároveň je zde řešena optimalizace načítání zdrojů (viz Obrázek 7). Šablona obsahuje cykly, které dynamicky generují odkazy na CSS a JavaScript soubory pouze tehdy, pokud je konkrétní stránka vyžaduje. Tím se zamezuje zbytečnému stahování velkých skriptů na stránkách, kde nejsou potřeba.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>{% block title %}Life essentials{% endblock %}</title>
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
    />

    <link rel="stylesheet" href="../static/css/app.css" />
    {% for css_file in css %}
    <link rel="stylesheet" href="../static/css/{{ css_file }}" />
    {% endfor %}

    <link
      rel="stylesheet"
      href="https://cdn.datatables.net/2.3.2/css/dataTables.bootstrap5.css"
    />
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css"
    />

  </head>
  <body>
    <header>
      <nav
        id="sidebarMenu"
        class="collapse d-lg-block sidebar bg-light shadow-sm"
      >
```

Obrázek 7: Dynamické vkládání stylů a skriptů v hlavičce šablony

Zdroj: vlastní zpracování

Princip dědičnosti

Konkrétní stránky aplikace (tzv. potomci) nevytvářejí celou HTML strukturu znovu, ale dědí ji z rodiče příkazem `{% extends %}`. Jak ukazuje Obrázek 8, dceřiná šablona se stará pouze o svůj unikátní obsah, který vloží do připravených bloků. Flask při vykreslení automaticky spojí rodičovskou kostru s obsahem potomka do výsledné HTML stránky.

nplates > index.html > div.container.py-5 > div.row.mb-4 > div.col-12.text-center.mb-4 > p.text-muted

```
{% extends "base.html" %}

{% block title %}My Dashboard | Life Essentials{% endblock %}

{% block body %}
<div class="container py-5">

    <div class="p-5 mb-5 bg-white rounded-3 shadow-sm border position-relative overflow-hidden">
        <div class="position-relative z-1">
            <h1 class="display-5 fw-bold text-dark">
                Welcome back, {{ current_user.username or "User" }}!
            </h1>

            <figure class="mt-4">
                <blockquote class="blockquote">
                    <p class="fs-4 fst-italic text-muted">"{{ quote.text }}"</p>
                </blockquote>
                <figcaption class="blockquote-footer mt-1">
                    {{ quote.author }}
                </figcaption>
            </figure>

        </div>
        <i class="fas fa-heartbeat position-absolute top-50 end-0 translate-middle-y text-light" style="font-size: 15rem; opacity: 0.1; margin-right: -2rem;"></i>
    </div>

    <div class="row g-4 mb-5">

        <div class="col-md-3 col-sm-6">
```

Obrázek 8: Ukázka dceřiné šablony dědící z rodiče

Zdroj: vlastní zpracování

7.1.2 Framework Bootstrap 5 a responzivní layout

Pro zajištění responzivity a vizuální konzistence byl zvolen framework *Bootstrap 5*. Jeho hlavním přínosem pro projekt je robustní mřížkový systém, který umožňuje definovat rozložení prvků pro různé velikosti obrazovek bez nutnosti psát složitá CSS pravidla od nuly.

Využití Grid systému

Aplikace striktně dodržuje princip *Mobile First*. V kódu je to realizováno pomocí tříd `col-*` a `col-md-*`. Jak ukazuje Obrázek 9, prvky jsou primárně definovány tak, aby na mobilních zařízeních zabíraly 100 % šířky (výchozí chování blokových elementů), a teprve od šířky "medium"(tablet/desktop) se řadí vedle sebe do sloupců.

Tím je zajištěno, že Dashboard je čitelný na telefonu i na širokoúhlém monitoru, aniž by docházelo k deformaci obsahu.

```html
<div class="row g-4 mb-5">

    <div class="col-md-3 col-sm-6">
        <div class="card h-100 border-0 shadow-sm bg-primary bg-opacity-10">
            <div class="card-body text-center">
                <i class="fas fa-bed fa-2x text-primary mb-3"></i>
                <h6 class="text-uppercase text-muted small fw-bold">Sleep & Recovery</h6>
                <h3 class="fw-bold text-dark">7 - 9 h</h3>
                <small class="text-muted">Recommended daily</small>
            </div>
        </div>
    </div>

    <div class="col-md-3 col-sm-6">
        <div class="card h-100 border-0 shadow-sm bg-success bg-opacity-10">
            <div class="card-body text-center">
                <i class="fas fa-walking fa-2x text-success mb-3"></i>
                <h6 class="text-uppercase text-muted small fw-bold">Physical Activity</h6>
                <h3 class="fw-bold text-dark">150 min</h3>
                <small class="text-muted">Recommended weekly</small>
            </div>
        </div>
    </div>

    <div class="col-md-3 col-sm-6">
        <div class="card h-100 border-0 shadow-sm bg-warning bg-opacity-10 position-relative hover-shadow" style="transition: transform 0.2s;">
            <div class="card-body text-center">
                <i class="fas fa-apple-alt fa-2x text-warning mb-3"></i>
                <h6 class="text-uppercase text-muted small fw-bold">Nutrition & Diet</h6>
                <h3 class="fw-bold text-dark">Food Table</h3>
                <small class="text-dark fw-bold">Launch Tool <i class="fas fa-arrow-right ms-1"></i></small>
                <a href="{{ url_for('main.food_table') }}" class="stretched-link"></a>
            </div>
        </div>
    </div>
```

Obrázek 9: Implementace responzivního mřížkového systému v šabloně

Zdroj: vlastní zpracování

Komponenty UI

Kromě layoutu byly využity i předpřipravené komponenty, které urychlily vývoj. Jedná se zejména o modální okna pro přidávání záznamů stravy a chybové hlášky, které poskytují uživateli zpětnou vazbu po odeslání formuláře. Stylizace těchto prvků je sjednocená napříč celou aplikací.

7.1.3 Vlastní CSS styly a specifické úpravy

Ačkoliv Bootstrap pokrývá většinu běžných scénářů, pro specifické vizuální požadavky aplikace byly vytvořeny vlastní kaskádové styly. Ty slouží k přepsání výchozích hodnot frameworku a k definici unikátního chování, jako je například logika postranního panelu.

Modularita stylů

V zájmu přehlednosti a snadné údržby není CSS kód držen v jednom obřím souboru. Místo toho je rozdělen do logických celků:

- **`app.css`:** Globální styly pro layout, typografii a sidebar.
- **`login.css` / `register.css`:** Specifické styly pro přihlašovací stránku, která má odlišný design.

- **`food_table.css`:** Úpravy pro tabulku potravin a skrytí specifických sloupců na mobilech.

Responzivita postranního panelu

Nejsložitějším prvkem z hlediska CSS byl postranní panel. Bootstrap nemá nativní komponentu pro postranní panel, která by se chovala podle požadavků návrhu. Proto byla v souboru `app.css` implementována vlastní logika pomocí *Media Queries*.

Jak je vidět na Obrázku 10, pravidlo `@media (max-width: 991.98px)` definuje, že na menších zařízeních se postranní panel absolutně pozicuje a překrývá obsah, zatímco na desktopu je fixní součástí layoutu. Toto řešení zajišťuje maximální využití prostoru na malých displejích.

```
itic > css > # app.css > { } @media (max-width: 991.98px) > .navbar-nav .dropdown-menu
1   body {
2     background-color: #fbfbfb;
3   }
4   @media (min-width: 991.98px) {
5     main {
6       padding-left: 240px;
7     }
8   }
9
10  .sidebar {
11    position: fixed;
12    top: 0;
13    bottom: 0;
14    left: 0;
15    padding: 58px 0 0;
16    box-shadow: 0 2px 5px 0 rgb(0 0 0 / 5%), 0 2px 10px 0 rgb(0 0 0 / 5%);
17    width: 240px;
18    z-index: 600;
19  }
20
21  @media (max-width: 991.98px) {
22    .sidebar {
23      width: 100%;
24    }
25  }
```

Obrázek 10: Využití Media Queries pro adaptaci sidebaru

Zdroj: vlastní zpracování

7.1.4 JavaScript, asynchronní komunikace a integrace API

Skriptovací jazyk JavaScript v tomto projektu neslouží pouze k drobným vizuálním efektům, ale tvoří klíčovou vrstvu pro interaktivitu a plynulý chod aplikace na straně klienta. Pro efektivnější manipulaci s prvkem DOM (Document Object Model) a zjednodušení syntaxe byla v projektu využita knihovna *jQuery*.

Zpracování událostí a dynamické UI

Díky využití knihovny jQuery je obsluha uživatelských událostí (kliknutí na tlačítka,

psaní do polí) značně zjednodušena. Jak ukazuje Obrázek 11, kód neustále naslouchá akcím uživatele a dynamicky na ně reaguje, například odkrývá specifická pole formuláře nebo aktualizuje hodnoty v grafech bez nutnosti znovunačtení celé webové stránky. O vykreslování samotných grafů se pak stará specializovaná knihovna *Chart.js*.

```
$(document).ready(function() {
    initFoodTable();
    initFoodDetailTable();
    loadMyMeals();
    $('[data-bs-toggle="tooltip"]').tooltip();
    $('#food_table').on('click', 'tr', function() {
        $('#food_selection_col').removeClass('col-12').addClass('col-md-6');
        $('#food_detail_col').removeClass('d-none');
        if ($.fn.DataTable.isDataTable('#food_table')) {
            $('#food_table').DataTable().columns.adjust();
        }
    });
});

$('#user_gender_select').on('change', function() {
    const gender = $(this).val();

    if (gender === 'female') {
        $('#female_status_col').removeClass('d-none');
    } else {
        $('#female_status_col').addClass('d-none');
        $('#user_status_select').val('none');
    }
    updateCalculatedTargets();
});

$('#user_weight_input, #user_status_select, #user_pal_select').on('change input', function() {
    updateCalculatedTargets();
});
```

Obrázek 11: Ukázka obsluhy událostí pomocí knihovny jQuery

Zdroj: vlastní zpracování

Asynchronní komunikace (AJAX)

Zásadní technologií pro uživatelský komfort je asynchronní komunikace se serverem. Když uživatel například přidá nový záznam o stravě, data se neodesílají klasickým HTTP POST požadavkem, který by zablokoval a přesměroval prohlížeč. Místo toho je odeslán asynchronní AJAX dotaz na pozadí. Backend požadavek zpracuje, uloží do databáze a vrátí odpověď ve formátu JSON. JavaScript tuto odpověď zachytí a okamžitě zaktualizuje příslušnou tabulku, což vytváří plynulý zážitek podobný desktopovým aplikacím.

Integrace externího REST API

Aplikace není izolovaným systémem, ale dokáže obohacovat svá data komunikací s externími službami. Klíčovou funkcionalitou je integrace švýcarské veřejné databáze nutričních hodnot BLV.

Princip fungování (viz Obrázek 12) je postaven na asynchronním dvoufázovém načítání dat pomocí rozhraní JavaScript *Fetch API*:

1. **Primární dotaz:** Při inicializaci modulu skript odešle požadavek na koncový bod API pro získání kompletního seznamu potravin. Tato data naplní hlavní vyhledávací tabulku.
2. **Sekundární dotaz:** Jakmile uživatel vybere konkrétní řádek v tabulce, JavaScript zachytí ID potraviny a odešle cílený požadavek na detailní koncový bod (*food/{id}*).
3. **Mapování a zpracování:** Server vrátí komplexní objekt ve formátu JSON, ze kterého skript extrahuje potřebné nutriční hodnoty.

Tento proces rapidně urychluje uživatelskou práci a zajišťuje přesnost dat podle oficiálních nutričních tabulek.

Klientské výpočty, evaluace a simulace

Získaná data jsou následně matematicky zpracovávána přímo v prohlížeči uživatele, což odlehčuje zátěž serveru a poskytuje okamžitou zpětnou vazbu. Tento princip lokálních výpočtů a logiky je klíčový pro dva hlavní moduly aplikace:

- **Modul Kalkulačka živin:** Data poskytovaná z REST API jsou standardizována na referenční porci 100 gramů. JavaScript zde plní roli komplexního kalkulátoru, pomocí jednoduchého vzorce přepočítává hodnoty (kalorie, makroživiny i mikroživiny) na základě uživatelem zadané gramáže.

  Významným prvkem modulu je integrace referenčních hodnot příjmu podle Evropského úřadu pro bezpečnost potravin (EFSA) [9]. Aplikace není striktně vázána pouze na uložený profil uživatele, ale funguje jako otevřený simulátor. Uživatel může měnit parametr (věk, váhu, pohlaví či úroveň aktivity) a nástroj okamžitě přepočítá cílové hodnoty. To umožňuje nejen experimentovat s hypotetickými scénáři (např. jak se změní potřeby při nárůstu hmotnosti), ale také aplikaci využít pro výpočet stravy dětí či jiných rodinných příslušníků.

  Barevná evaluace a práce s daty: U základních makroživin a energie provádí skript automatickou evaluaci plnění denních cílů. Pokud se příjem pohybuje v toleranci (např. $\pm$ 10 %), je řádek tabulky podbarven zeleně, při větší odchylce žlutě a při kritickém nedodržení červeně. Díky komplexnosti švýcarské databáze systém zobrazuje i detailní hodnoty vitamínů a minerálů, které ocení pokročilí uživatelé a nadšenci do výživy. Automatická evaluace se na tyto mikroživiny aktuálně neaplikuje, neboť pro drtivou většinu běžné populace je prioritní zvlád-

nutí základních makroživin. Evaluace mikroživin nicméně představuje otevřený prostor pro budoucí rozšíření aplikace.

- **Modul Vliv životního stylu:** Zde JavaScript zajišťuje vysoce interaktivní zážitek při práci s biometrickými údaji. Skript neustále zachytává události z uživatelských vstupů (např. manipulace s posuvníkem délky spánku nebo změna úrovně fyzické aktivity). Na základě těchto změn okamžitě přepočítává koeficienty a aktualizuje klíčové metriky, které následně vizualizuje pomocí knihovny *Chart.js*. Uživatel díky tomu bezprostředně vidí zdravotní dopad simulované změny svých denních návyků.

```
function initFoodTable() {
    $('#food_table').DataTable({
        ajax: {
            url: 'https://api.webapp.prod.blv.foodcase-services.com/BLV_WebApp_WS/webresources/BLV-api/foods?lang=en&limit=2000',
            dataSrc: json => json
        },
        columns: [
            { data: 'foodName', title: 'Food name' },
            { data: 'categoryNames', title: 'Category' }
        ],
        order: [[1, 'asc']]
    }).on('click', 'tr', function () {
        const table = $('#food_table').DataTable();
        const data = table.row(this).data();
        $(this).addClass('table-primary').siblings().removeClass('table-primary');
        showFoodDetail(data);
        loadFoodDetail(data.id);
    });
```

Obrázek 12: Implementace asynchronního volání na externí API

Zdroj: vlastní zpracování

7.1.5 Jazyk Python a aplikační logika

Základním pilířem aplikační (backendové) vrstvy je programovací jazyk *Python*. Byl zvolen pro svou vysokou čitelnost, striktní syntaxi a vynikající ekosystém knihoven. V rámci projektu neslouží k těžkým matematickým výpočtům, ale funguje mimo jiného jako bezpečný prostředník mezi uživatelským rozhraním a databází.

Objektově orientovaný přístup

Celý backend je striktně postaven na principech objektově orientovaného programování. Jak ukazuje Obrázek 13 ze souboru `models.py`, databázové entity (např. Uživatel, Záznam jídla) nejsou reprezentovány pouze jako pasivní datové struktury, ale jako plnohodnotné třídy.

Tento přístup umožňuje snadnou manipulaci se záznamy v paměti a bezpečné zapouzdření logiky přímo k danému objektu. Ukázková třída `User` tak kromě samotných vlastností (email, uživatelské jméno) obsahuje i vlastní metody, například dynamický výpočet BMI a věku pomocí dekorátoru `@property`, nebo generování bezpečnostních tokenů pro obnovu hesla.

```python
class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True, nullable=False)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(256), nullable=False)
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime, default=db.func.current_timestamp(), onupdate=db.func.current_timestamp())
    deleted_at = db.Column(db.DateTime, nullable=True)


    metrics = db.relationship('User_metrics', back_populates='user', uselist=False, cascade="all, delete-orphan")
    meal_logs = db.relationship('Meal_log', back_populates='user', cascade="all, delete-orphan")


    @property
    def bmi(self):
        if self.metrics and self.metrics.height_cm and self.metrics.weight_kg:
            height_m = self.metrics.height_cm / 100
            bmi_value = self.metrics.weight_kg / (height_m ** 2)
            return round(bmi_value, 1)
        return None

    @property
    def age(self):
        if self.metrics and self.metrics.date_of_birth:
            dob = self.metrics.date_of_birth
            today = datetime.today().date()
            return today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
        return None

    def get_reset_token(self):
        s = Serializer(current_app.config['SECRET_KEY'])
        return s.dumps({'user_id': self.id}, salt='password-reset-salt')
```

Obrázek 13: Ukázka objektově orientovaného návrhu databázového modelu (třída User)

Zdroj: vlastní zpracování

Validace a zpracování dat

Python má na starosti kritickou roli z hlediska bezpečnosti a integrity. Veškerá data odeslaná z frontendových formulářů (např. při registraci nebo odesílání asynchronních AJAX požadavků) jsou na straně serveru před samotným uložením validována. Skript ověřuje datové typy, zachytává chybné vstupy a v případě nesrovnalostí vrací klientovi odpovídající chybový stav.

7.1.6 Mikroframework Flask a architektura serveru

Ačkoliv je Python silný jazyk, sám o sobě neobsahuje nástroje pro obsluhu webového provozu. Pro tyto účely byl zvolen webový *mikroframework Flask*. Oproti robustním řešením (jako je např. Django) nabízí Flask minimalistický základ, který vývojáři ponechává plnou kontrolu nad architekturou aplikace a výběrem dodatečných knihoven.

Směrování a obsluha HTTP požadavků

Klíčovou funkcí frameworku je tzv. Routing, což je překlad URL adresy na spuštění

konkrétní Python funkce (viz Obrázek 14). K tomu Flask využívá systém dekorátorů (např. `@app.route`).

Každý koncový bod v souborech `routes.py` či `auth.py` je definován tak, aby reagoval na specifické HTTP metody. Například při zpracování přihlašovacího formuláře tatáž funkce nejprve metodou `GET` vykreslí prázdnou HTML šablonu, a následně metodou `POST` bezpečně zpracuje odeslaná uživatelská data.

```python
@bp.route('/login', methods=['GET', 'POST'])
def login():
    js = ['login.js']
    css = ['login.css']
    error = None

    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        user = User.query.filter_by(email=email, deleted_at = None).first()

        if user is None:
            error = 'This combination of email and password does not exist for existing account.'
        elif not check_password_hash(user.password_hash, password):
            error = 'This combination of email and password does not exist for existing account.'

        if error is None:
            login_user(user)
            flash('Login successful!', 'success')
            return redirect(url_for('main.home'))

        flash(error, 'info')

    return render_template('login.html', js=js,  css=css)
```

Obrázek 14: Ukázka směrování (Routing) a obsluhy GET/POST metod
Zdroj: vlastní zpracování

Modularita kódu, Blueprints a Application Factory

Aby byl zachován čistý a udržovatelný kód, není veškerá logika aplikace soustředěna v jediném spouštěcím souboru. Místo toho projekt využívá pokročilý návrhový vzor *Application Factory* v kombinaci s technologií *Flask Blueprints*.

Srdcem tohoto modulárního přístupu je inicializační soubor `__init__.py`. Ten obsahuje tovární funkci (zpravidla `create_app()`), která má na starosti vytvoření instance samotné aplikace, načtení konfiguračních proměnných a inicializaci klíčových rozšíření (např. databázové ORM vrstvy či správce uživatelských relací).

Samotná logika je pak pomocí *Blueprints* rozdělena do menších, logicky ucelených komponent v adresáři `flaskr`. Tyto komponenty se chovají jako miniaturní aplikace, které se v souboru `__init__.py` do hlavní aplikace pouze zaregistrují. Architektura se skládá z následujících hlavních modulů:

- **`auth.py`:** Blueprint obsluhující výhradně autentizační procesy (přihlášení, registrace, obnova hesla).
- **`routes.py`:** Blueprint, který zajišťuje směrování a vykreslování hlavních aplikačních pohledů (Dashboard, moduly stravy).
- **`user.py`:** Blueprint oddělující logiku správy uživatelského profilu.
- **`models.py`:** Neslouží jako Blueprint, ale obsahuje centralizované definice všech databázových modelů a schémat.

Tento architektonický přístup výrazně usnadňuje orientaci v projektu, předchází tzv. cyklickým importům a zaručuje, že jednotlivé části systému lze spravovat a rozšiřovat nezávisle na sobě.

```python
db = SQLAlchemy()
login_manager = LoginManager()
mail = Mail()
csrf = CSRFProtect()

def create_app():
    app = Flask(__name__, template_folder='../templates', static_folder='../static')
    app.config.from_pyfile('instance/config.py', silent=True)

    db.init_app(app)
    login_manager.init_app(app)
    mail.init_app(app)
    csrf.init_app(app)

    import flaskr.models as models
    @login_manager.user_loader
    def load_user(user_id):
        return models.User.query.filter_by(id=int(user_id), deleted_at=None).first()
    from . import auth
    from . import routes
    from . import user
    from . import email
    app.register_blueprint(auth.bp)
    app.register_blueprint(routes.bp)
    app.register_blueprint(user.bp)
    app.register_blueprint(email.bp)
    return app
```

Obrázek 15: Ukázka návrhového vzoru Application Factory a registrace Blueprints

Zdroj: vlastní zpracování

Řízení relací a ochrana přístupu

Flask nativně řeší ukládání relací uživatele pomocí bezpečně podepsaných cookies. Díky rozšíření *Flask-Login* je v projektu implementována robustní správa stavu přihlášení. Jak ukazuje Obrázek 16, kritické části aplikace jsou chráněny dekorátorem `@login_required`. Tento jednoduchý, ale silný mechanismus zajistí, že pokud se neregistrovaný uživatel pokusí přistoupit např. k profilu, server požadavek automaticky odmítne a přesměruje jej na přihlašovací stránku.

```python

@bp.route('/logout', methods=['GET'])
@login_required
def logout():
    logout_user()
    return redirect(url_for('auth.login'))

@bp.route('/forgotten_password', methods=['GET','POST'])
def forgotten_password():
    js = ['login.js']
    css = ['login.css']

    if request.method == 'POST':
        email = request.form.get('email')
        user = User.query.filter_by(email=email, deleted_at = None).first()

        if user:
            password_reset_email(email, user.get_reset_token())
            flash('If user with this email exists than password reset email was sent! Please check your inbox.', 'info')
        else:
            flash('If user with this email exists than password reset email was sent! Please check your inbox.', 'info')

    return render_template('forgotten_password.html', js=js, css=css)

```

Obrázek 16: Zabezpečení koncového bodu proti neoprávněnému přístupu
Zdroj: vlastní zpracování

7.1.7 Relační databáze PostgreSQL a správa dat

O bezpečné a trvalé uložení veškerých dat aplikace, včetně uživatelských profilů, biometrických údajů a záznamů o stravě, se stará pokročilý relační databázový systém *PostgreSQL*. Zvolen byl především pro svou stabilitu, striktní dodržování standardů a výbornou podporu cizích klíčů, které jsou nezbytné pro udržení referenční integrity celého systému.

Abstrakce dat přes ORM vrstvu

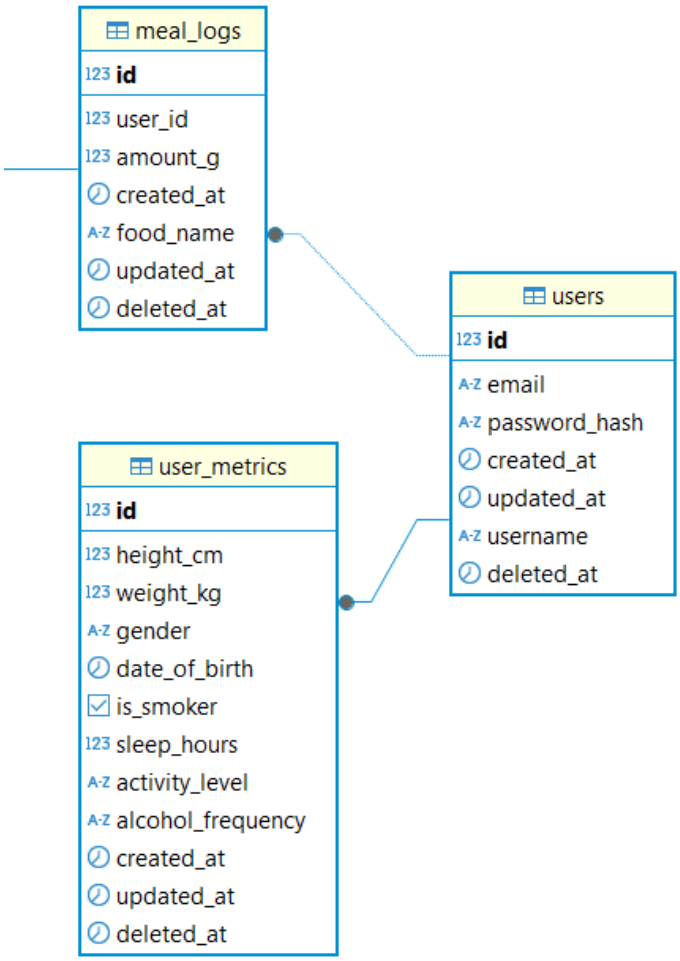
Ačkoliv je na pozadí využíván systém PostgreSQL, samotná aplikace s ním nekomunikuje primárně psaním surových SQL dotazů. Díky integraci knihovny *SQLAlchemy* dochází k automatickému překladu definovaných Python modelů na relační tabulky. ORM vrstva se stará o bezpečné ukládání záznamů, ochranu proti útokům typu SQL Injection a efektivní řešení složitých tabulkových vazeb (např. kaskádové mazání záznamů při smazání uživatele).

Administrace a vizualizace pomocí nástroje DBeaver

Pro efektivní vývoj, přímou správu databáze a ladění chyb byl v průběhu implementace využíván grafický klient *DBeaver*. Jedná se o profesionální nástroj pro administraci databází, který poskytuje vizuální rozhraní pro prohlížení struktury tabulek a kontrolu reálně uložených dat.

Nástroj DBeaver navíc výrazně zjednodušuje vizualizaci celého datového modelu. Jak ilustruje Obrázek 23, klient umožňuje automaticky generovat Entitně-relační diagramy

přímo z aktuálního schématu databáze. To vývojáři poskytuje okamžitý přehled o nastavených relacích (např. provázání centrální tabulky `users` s tabulkou deníku stravy `meal_logs`).



Obrázek 17: Vizualizace relačních vazeb (ER diagram) v nástroji DBeaver
Zdroj: vlastní zpracování

7.2 Bezpečnost

Vzhledem k tomu, že aplikace zpracovává a uchovává osobní i biometrické údaje uživatelů (jako jsou tělesné proporce, věk či deník stravy), byla bezpečnost jedním z hlavních pilířů celého vývojového cyklu. Architektura systému byla navržena tak, aby minimalizovala riziko úniku dat či neoprávněné manipulace.

Návrh bezpečnostních opatření striktně vychází z doporučení mezinárodního standardu OWASP.

7.2.1 Implementované bezpečnostní mechanismy

Následující část detailně popisuje, jakým způsobem jsou v praxi řešeny nejkritičtější zranitelnosti z aktuálního žebříčku OWASP Top 10. Samotná architektura frameworku Flask a jeho ekosystému poskytuje řadu vestavěných nástrojů, které tyto hrozby proaktivně minimalizují.

Ochrana proti SQL Injection

Tradiční a vysoce nebezpečná zranitelnost, při které útočník podsouvá škodlivý SQL kód do vstupních polí, je v aplikaci eliminována použitím ORM vrstvy *SQLAlchemy*. Zásadním bezpečnostním prvkem ORM je, že aplikační logika nekonstruuje surové SQL dotazy pomocí spojování řetězců. Veškerá data předávaná z formulářů jsou frameworkem automaticky sanitizována a dotazy jsou do databáze odesílány parametrizovaně.

Ochrana proti Cross-Site Scripting (XSS)

Obrana proti útokům typu XSS, kde se útočník snaží do stránek vložit škodlivý JavaScript, je řešena na úrovni prezentační vrstvy. Šablonovací systém *Jinja2* má ve výchozím stavu aktivní funkci *Autoescaping*. To znamená, že pokud by uživatel do formuláře zadal například `<script>alert('XSS')</script>`, Jinja2 tento text při vykreslování bezpečně převede na neškodné HTML entity (`&lt;script&gt;...`), čímž zabrání provedení kódu v prohlížeči.

Ochrana proti Cross-Site Request Forgery

Pro obranu proti podvržení žádosti, kdy útočník zneužívá aktivní relaci přihlášeného uživatele, bylo využito rozšíření *Flask-WTF*. U každého formuláře je generován unikátní, kryptograficky bezpečný `csrf_token`, který je spárován s aktuální relací uživatele. Pokud tento skrytý token ve formuláři chybí nebo neodpovídá relaci, server požadavek okamžitě zamítne.

Řízení přístupu

Zajištění, aby uživatelé měli přístup pouze k datům, ke kterým mají oprávnění, obsluhuje modul *Flask-Login*. Klíčovým prvkem obrany je striktní používání dekorátoru `@login_required` nad všemi citlivými koncovými body.

Kryptografická ochrana a autentizační procesy

V souladu s bezpečnostními standardy nejsou hesla uživatelů v databázi nikdy ukládána v otevřené podobě. Pro jejich ochranu se využívá knihovna *Werkzeug.security*, která

hesla transformuje pomocí moderních hashovacích algoritmů doplněných o unikátní kryptografickou sůl.

Kromě samotného hashování je kritickým prvkem obrany vynucení dostatečné složitosti hesla. Aplikace pro tento účel implementuje vlastní backendovou validaci. Namísto závislosti na robustních externích formulářových knihovnách je pro ověření síly hesla využita centralizovaná funkce postavená na regulárních výrazech. Tato logika je volána při každém vytváření či změně hesla (při registraci, obnově i v nastavení profilu) a striktně vyžaduje minimální délku osmi znaků a současnou přítomnost velkých i malých písmen, číslic a speciálních znaků. Pokud heslo těmto pravidlům nevyhovuje, proces je bezpečně přerušen a uživatel je upozorněn odpovídající chybovou hláškou. Tento přístup udržuje kód lehký a plně pod kontrolou vývojáře.

```
148
149  def is_password_strong(password):
150
151      if len(password) < 8:
152          return False, 'The password must have atleast 8 symbols.'
153      if not re.search(r"[a-z]", password):
154          return False, 'The password must contain a lowercase letter.'
155      if not re.search(r"[A-Z]", password):
156          return False, 'The password must contain an uppercase letter.'
157      if not re.search(r"\d", password):
158          return False, 'The password must contain at least one digit.'
159      if not re.search(r"[ !@#$%^&*(),.?\":{}|<>]", password):
160          return False, 'The password must contain a special character.'
161      return True, None
```

Obrázek 18: Vlastní implementace validace složitosti hesla pomocí regulárních výrazů

Zdroj: vlastní zpracování

Významným bezpečnostním detailem v tomto procesu je obrana proti tzv. *User Enumeration* útokům. Pokud uživatel požádá o obnovu hesla, server vždy vrací neutrální hlášku ("Pokud účet existuje, e-mail byl odeslán") nezávisle na tom, zda byl e-mail v databázi reálně nalezen. Útočníkovi je tak znemožněno testovat a zjišťovat, které e-mailové adresy jsou v systému registrovány.

Bezpečná konfigurace

Zásadním krokem před nasazením aplikace do reálného provozu je úprava konfiguračních parametrů. Ponechání vývojových nástrojů v produkci je kritickou chybou. V souboru `run.py` (viz Obrázek 19) je proto absolutní nezbytností deaktivovat vývojový režim, tedy explicitně nastavit parametr `debug=False`. Zapnutý debug mód by totiž

při chybě aplikace v prohlížeči zobrazil interaktivní konzoli s kompletním zdrojovým kódem a výpisem paměti, což představuje obrovské riziko kompromitace celého serveru.

```
from flaskr import create_app

app = create_app()


if __name__ == '__main__':
    app.run(debug=False)
```

Obrázek 19: Deaktivace vývojového režimu pro produkční nasazení

Zdroj: vlastní zpracování

8 Návrh pro budoucí vývoj

Ačkoliv aktuální verze aplikace *Life Essentials* plně naplňuje stanovené cíle a poskytuje uživatelům robustní nástroj pro sledování stravy a vlivu životního stylu, existuje široký prostor pro její další rozšiřování, vylepšení a optimalizaci. Následující návrhy představují logické kroky pro budoucí iterace, které by mohly zvýšit uživatelský komfort, míru udržení uživatelů a celkovou přidanou hodnotu systému.

8.1 Integrace s nositelnou elektronikou

V současné podobě je modul zaměřený na vliv životního stylu závislý na manuálním zadávání dat, jako je délka spánku či úroveň fyzické aktivity. Zásadním vylepšením by bylo napojení aplikace na API populárních platforem pro sledování zdraví (např. Google Fit, Apple Health, Garmin). Tato integrace by umožnila automatickou synchronizaci biometrických dat a reálného energetického výdeje, čímž by se eliminovala nutnost ručního zadávání a výrazně zpřesnily výpočty.

8.2 Pokročilá analytika a personalizovaná doporučení

Aplikace aktuálně sbírá podrobná data o stravovacích návycích a dokáže barevně vizualizovat plnění denních cílů. Budoucí vývoj by mohl zahrnovat implementaci pokročilejší datové analytiky. Systém by mohl automaticky detekovat dlouhodobé nutriční trendy (např. opakovaný deficit specifických mikronutrientů) a proaktivně doporučovat vhodné potraviny z integrované švýcarské databáze, které by tyto nedostatky přirozeně vykryly.

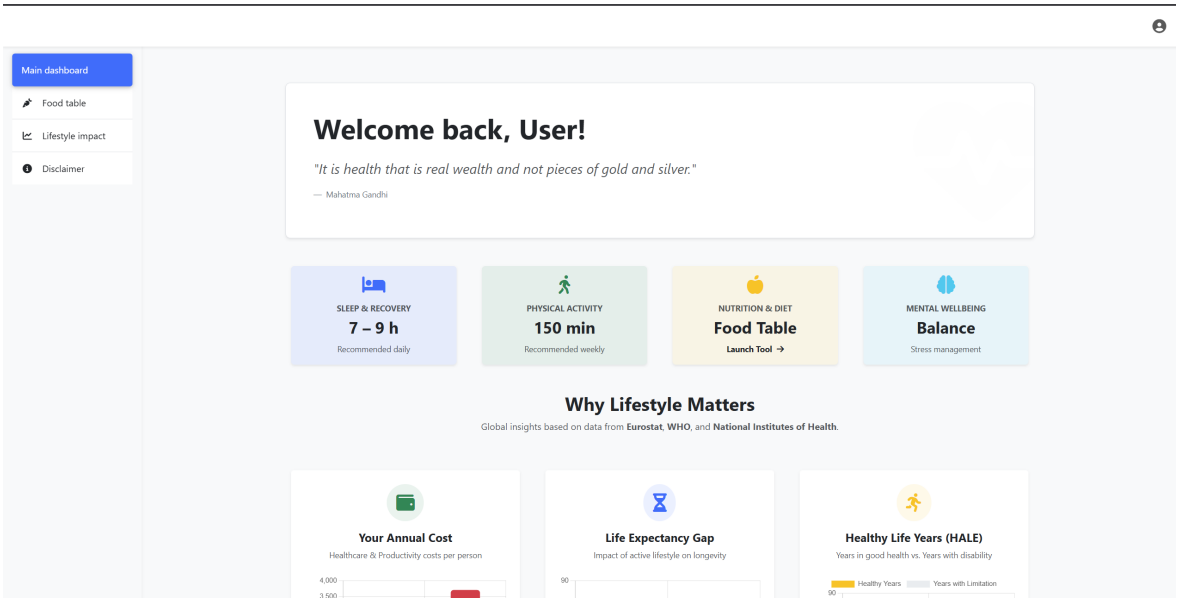
8.3 Rozšíření o expertní a sociální moduly

Pro zvýšení motivace uživatelů by bylo vhodné systém obohatit o možnost sdílení vlastních poskládaných jídel či celých denních plánů. Velkým přínosem by bylo zavedení systému uživatelských rolí (např. Běžný uživatel, Nutriční terapeut). Terapeut by tak mohl, po výslovném udělení souhlasu, nahlížet do stravovacího deníku svého klienta, analyzovat jeho záznamy a poskytovat mu odbornou zpětnou vazbu či přímo upravovat doporučené cílové hodnoty v rozhraní aplikace.

9 Shrnutí a diskuse výsledků

Záměrem této bakalářské práce bylo navrhnout, implementovat a otestovat komplexní webovou aplikaci *Life Essentials*, která uživatelům poskytne centralizovaný nástroj pro sledování stravovacích návyků a analýzu vlivu životního stylu na jejich celkové zdraví. Lze konstatovat, že stanovené cíle byly úspěšně naplněny.

Výsledkem vývoje je plně funkční, responzivní a bezpečný systém postavený na moderním technologickém zásobníku v čele s frameworkem Flask a relační databází PostgreSQL.



Obrázek 20: Finální vzhled dashboardu

Zdroj: vlastní zpracování

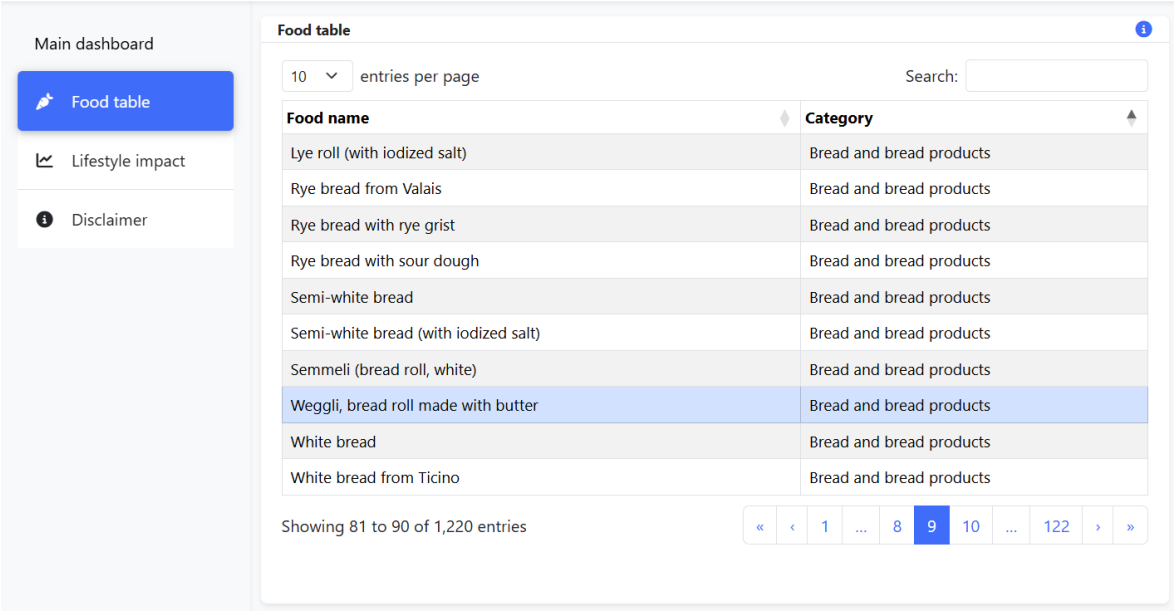
Zhodnocení zvoleného přístupu

Během implementace se jako klíčové ukázalo rozdělení zodpovědností mezi serverovou a klientskou část aplikace. Zatímco Python zajišťuje pevnou strukturu, bezpečnost dat a komunikaci s databází, JavaScript přebírá veškerou asynchronní logiku a okamžité výpočty. Toto oddělení vedlo k vytvoření uživatelského rozhraní, které reaguje plynule bez nutnosti neustálého obnovování stránky, což výrazně zvyšuje celkový uživatelský komfort.

Výzvy a omezení

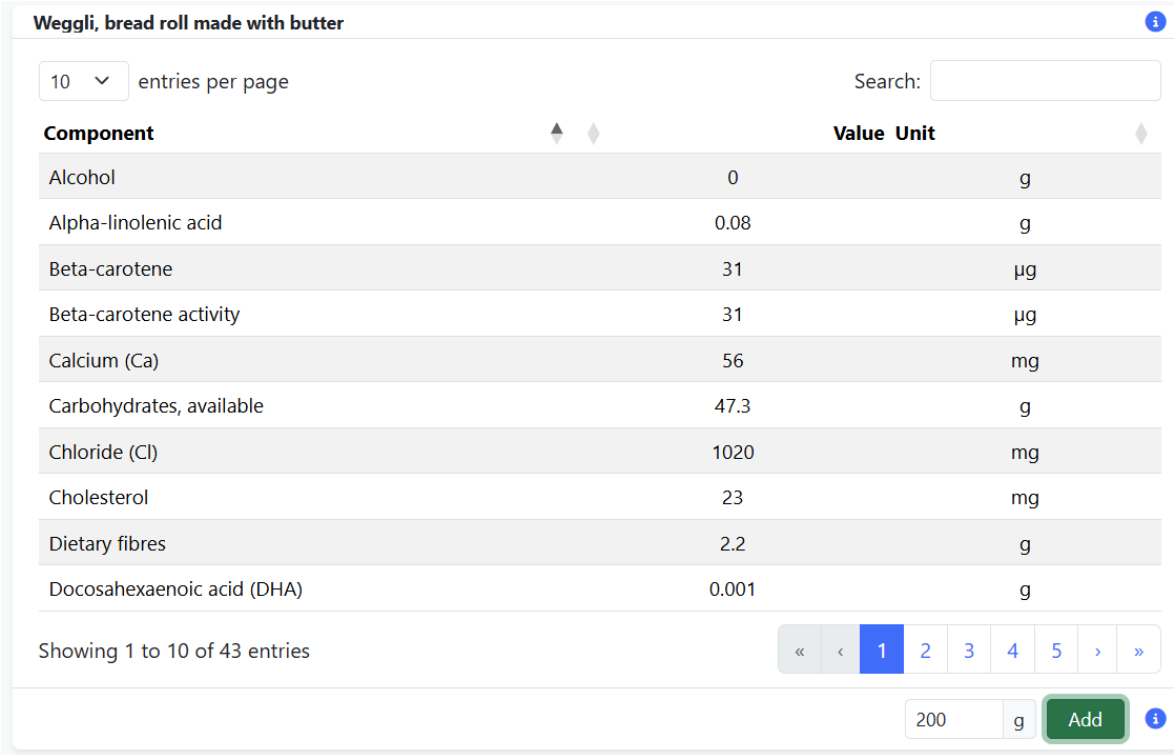
Největší výzvou během vývoje byla správná optimalizace výkonu při zpracování obsáhlých datových sad formátu JSON z externího API a jejich následné namapování na lokální databázové schéma. Stejně tak bylo nutné věnovat zvýšenou pozornost re-

sponzivitě složitějších UI komponent, což bylo úspěšně vyřešeno kombinací frameworku Bootstrap a vlastních CSS pravidel.



Obrázek 21: Tabulka potravin z REST API

Zdroj: vlastní zpracování

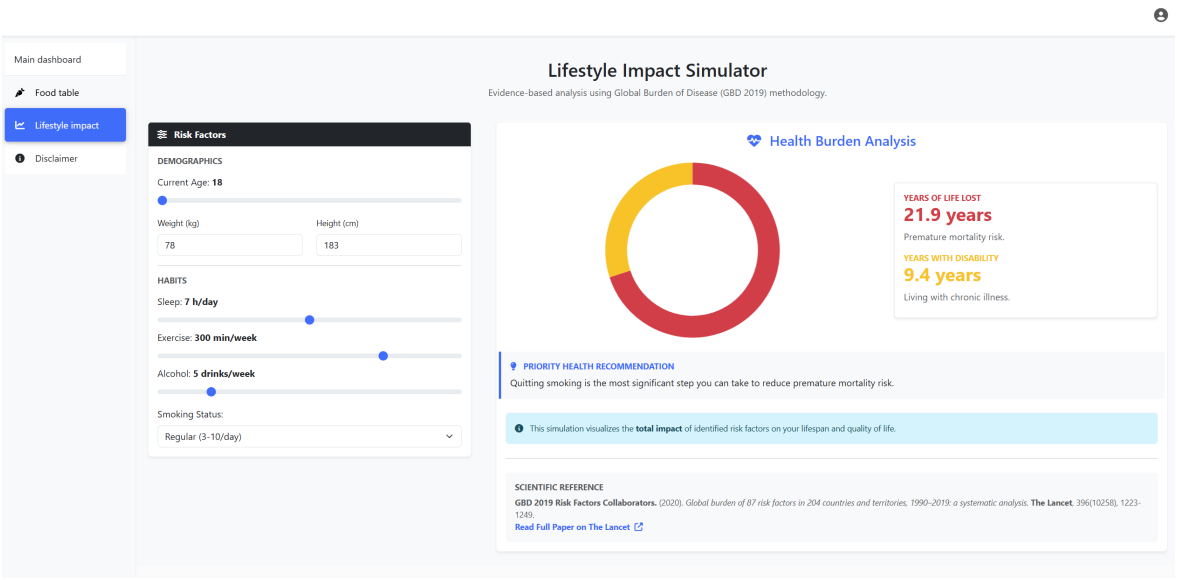


Obrázek 22: Tabulka nutričních hodnot vybrané potraviny

Zdroj: vlastní zpracování

Závěrečné zhodnocení

Vytvořená aplikace *Life Essentials* v současné podobě představuje stabilní produkt, který je plně připraven k reálnému nasazení. Zvolená modulární architektura navíc zaručuje, že systém je snadno rozšiřitelný a udržovatelný. Práce tak nejenže splnila své formální zadání, ale poskytla autorovi cenné praktické zkušenosti s vývojem reálného softwaru od prvotního návrhu přes integraci API až po zabezpečení.



Obrázek 23: Modul simulace dopadu životního stylu

Zdroj: vlastní zpracování

10 Závěry a doporučení

Zatímco předchozí kapitola shrnula konkrétní technické výsledky a implementační výzvy, tento závěrečný oddíl zasazuje celou práci do širšího kontextu. Výsledný systém představuje plně funkční a bezpečný nástroj postavený na moderním technologickém zásobníku, samotný vývojový proces odhalil i řadu úskalí a limitů zvoleného řešení.

Kritická diskuse výsledků a limitů řešení

Přestože je vytvořená aplikace plně provozuschopná a splňuje formální zadání, kritické zhodnocení implementace odhaluje prostor pro optimalizaci. Nejvýraznějším zjištěním je způsob práce s nutričními daty. Aktuální řešení spoléhá na dynamické dotazování externího REST API v reálném čase. I když tento přístup zaručuje aktuálnost, přináší kompromisy v oblasti výkonu a závislosti na službě třetí strany. Vhodnějším řešením pro produkční nasazení by bylo periodické stahování dat do lokální relační databáze.

Další reflexe se týká zvoleného technologického zásobníku na straně klienta. Kombinace serverového vykreslování pomocí *Jinja2* a následného obnovování komponent pomocí *JavaScriptu* a *jQuery* svůj účel splnila. Pro aplikaci s tak vysokou mírou interaktivity by však z dnešního pohledu bylo architektonicky čistší oddělit frontend do samostatné vrstvy pomocí moderního frameworku (např. React nebo Vue.js), což by zvýšilo modularitu a testovatelnost kódu.

Soulad s předpoklady a literaturou

Z hlediska teoretických východisek se aplikace striktně drží odborných doporučení analyzovaných v rešeršní části. Logika výpočtů a cílové hodnoty vycházejí z referenčních hodnot příjmu definovaných Evropským úřadem pro bezpečnost potravin (EFSA). Oproti mnoha běžným komerčním alternativám, které se omezují pouze na základní makroživiny, systém přesně mapuje i mikronutrienty, čímž poskytuje mnohem komplexnější analytický nástroj. Zároveň je architektura plně v souladu s bezpečnostními doporučeními organizace OWASP.

Závěrem lze shrnout, že největší přínos této bakalářské práce nespatřuji pouze v samotném funkčním softwaru, ale v celém procesu jeho vzniku. Práce na projektu mě donutila převzít odpovědnost za zvolenou architekturu i konkrétní implementační detaily, což mi umožnilo hlouběji pochopit souvislosti mezi návrhem databáze, bezpečností a uživatelským rozhraním.

11 Seznam použité literatury

[1] Li, Y.; Schoufour, J. D.; Wang, D. D.; et al. Healthy lifestyle and life expectancy free of cancer, cardiovascular disease, and type 2 diabetes: prospective cohort study. *BMJ*, volume 368, no. 8228, 2020: pp. l6669–l6669, ISSN 1756-1833, doi:10.1136/bmj.l6669. Available from: https://www.bmj.com/content/368/bmj.l6669.abstract

[2] World Health Organization. Regional Office for Europe. Healthy living: what is a healthy lifestyle? Technical report, World Health Organization. Regional Office for Europe, Copenhagen, 1999. Available from: https://iris.who.int/handle/10665/108180

[3] Walker, M. P. *Why We Sleep*. Scribner, an imprint of Simon & Schuster, Inc, second edition, 2017, ISBN 1501144316.

[4] World Health Organization and others. Everyday actions for better health–WHO recommendations. 2025, [Online; cit. 2026-01-27]. Available from: https://www.who.int/europe/news-room/fact-sheets/item/everyday-actions-for-better-health-who-recommendations

[5] Murray, C. J. L.; Aravkin, A. Y.; Zheng, P.; et al. Global burden of 87 risk factors in 204 countries and territories, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019. *The Lancet*, volume 396, no. 10258, 2020: pp. 1223–1249, ISSN 0140-6736, doi:10.1016/s0140-6736(20)30752-2.

[6] World Health Organization and others. 2021 physical activity factsheets for the European Union Member States in the WHO European Region. Technical report, World Health Organization. Regional Office for Europe, 2021.

[7] Mandsager, K.; Harb, S. C.; Cremer, P.; et al. Association of Cardiorespiratory Fitness With Long-term Mortality Among Adults Undergoing Exercise Treadmill Testing. *JAMA Network Open*, volume 1, no. 6, 2018: pp. e183605–e183605, ISSN 2574-3805, doi:10.1001/jamanetworkopen.2018.3605.

[8] Healthy diet. 2026. Available from: https://www.who.int/news-room/fact-sheets/detail/healthy-diet

[9] (EFSA), E. F. S. A. Dietary Reference Values for nutrients Summary report. *EFSA Supporting Publications*, volume 14, no. 12, 2017: p. 92, ISSN 2397-8325, doi:10.2903/sp.efsa.2017.e15121. Available from: https://www.efsa.europa.eu/sites/default/files/2017_09_DRVs_summary_report.pdf

[10] Mental health. ©2026. Available from: https://www.who.int/health-topics/mental-health

[11] Čápka, D. Úvod do PHP a webových aplikací. [Online; cit. 2026-01-29]. Available from: https://www.itnetwork.cz/php/zaklady/php-tutorial-uvod-do-webovych-aplikaci

[12] Baptista, R.; Shah, D. 3-Tier Website As a SAAS Model. In *InIC-CSOD-2018 Conference Proceedings*, 2018, pp. 35–36.

[13] What is HTML? ©2026. Available from: https://www.w3schools.com/whatis/whatis_html.asp

[14] Úvod do HTML. ©2026. Available from: https://www.itnetwork.cz/html-css/webove-stranky/jak-psat-moderni-web-html-tutorial-uvod-do-html

[15] What is CSS? ©2026. Available from: https://www.w3schools.com/whatis/whatis_css.asp

[16] Úvod do CSS3. ©2026. Available from: https://www.itnetwork.cz/html-css/css3/zaklady/uvod-do-css3

[17] Introduction: Getting started with Bootstrap. ©2026. Available from: https://getbootstrap.com/docs/5.3/getting-started/introduction/

[18] What is Bootstrap? ©2026. Available from: https://www.w3schools.com/whatis/whatis_bootstrap.asp

[19] What is JavaScript? ©2026. Available from: https://www.w3schools.com/whatis/whatis_js.asp

[20] Úvod do JavaScriptu. ©2026. Available from: https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

[21] Is Flutter Just For Mobile Apps in 2025? ©2026. Available from: https://www.netglade.cz/en/blog/je-flutter-jen-pro-mobilni-aplikace

[22] Úvod do TypeScriptu. ©2026. Available from: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu

[23] What is jQuery? ©2026. Available from: https://www.w3schools.com/whatis/whatis_jquery.asp

[24] Úvod do jQuery. ©2026. Available from: https://www.itnetwork.cz/javascript/jquery-zaklady/javascript-tutorial-funkcionalni-programovani-a-jquery-webova-kalkulacka

[25] Python vs PHP. 2025. Available from: https://www.geeksforgeeks.org/php/python-vs-php/

[26] Python Introduction. ©2026. Available from: https://www.w3schools.com/python/python_intro.asp

[27] Úvod do Pythonu. ©2026. Available from: https://www.itnetwork.cz/python/zaklady/python-tutorial-uvod-do-pythonu-a-zakladni-matematicke-operace

[28] Úvod do frameworku Flask a webových aplikací v Pythonu. ©2026. Available from: https://www.itnetwork.cz/python/flask/uvod-do-frameworku-flask-a-webovych-aplikaci-v-pythonu

[29] Welcome to Flask (Flask Documentation 3.1.x). ©2026. Available from: https://flask.palletsprojects.com/en/stable/

[30] SQLAlchemy Documentation. ©2010. Available from: https://flask.palletsprojects.com/en/stable/patterns/sqlalchemy/

[31] Utilities: Werkzeug Documentation (v3.1.x). ©2007. Available from: https://werkzeug.palletsprojects.com/en/stable/utils/

[32] WTForms Documentation. ©2008. Available from: https://wtforms.readthedocs.io/en/3.2.x/

[33] WTForms a Jinja2 šablony pro Flask framework. ©2026. Available from: https://www.itnetwork.cz/python/flask/wtforms-a-jinja2-sablony-pro-flask-framework

[34] Jinja2 3.1.6: PyPI. ©2026. Available from: https://pypi.org/project/Jinja2/

[35] Introduction - Twig Documentation. ©2026. Available from: https://twig.symfony.com/doc/3.x/intro.html

[36] SQL vs. NoSQL: The Differences Explained + When to Use Each. ©2026. Available from: https://www.coursera.org/articles/nosql-vs-sql

[37] Úvod do MongoDB. ©2026. Available from: https://www.itnetwork.cz/javascript/nodejs/uvod-do-mongodb

[38] Difference between MySQL and PostgreSQL. 2025. Available from: https://www.geeksforgeeks.org/mysql/difference-between-mysql-and-postgresql/

[39] About PostgreSQL. ©2026. Available from: https://www.postgresql.org/about/

[40] Von Solms, R.; Van Niekerk, J. From information security to cyber security. *computers & security*, volume 38, 2013: pp. 97–102.

[41] Security on the web. 2025. Available from: https://developer.mozilla.org/en-US/docs/Web/Security

[42] OWASP Top 10:2021 Introduction. © 2025. Available from: https://owasp.org/Top10/2025/0x00_2025-Introduction/

[43] A01 Broken Access Control. © 2025. Available from: https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/

[44] A02 Cryptographic Failures. © 2025. Available from: https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/

[45] A03 Software Supply Chain Failures. © 2025. Available from: https://owasp.org/Top10/2025/A03_2025-Software_Supply_Chain_Failures/

[46] A04 Cryptographic Failures. © 2025. Available from: https://owasp.org/Top10/2025/A04_2025-Cryptographic_Failures/

[47] A05 Injection. © 2025. Available from: https://owasp.org/Top10/2025/A05_2025-Injection/

[48] A06 Insecure Design. © 2025. Available from: https://owasp.org/Top10/2025/A06_2025-Insecure_Design/

[49] A07 Authentication Failures. © 2025. Available from: https://owasp.org/Top10/2025/A07_2025-Authentication_Failures/

[50] A08 Software or Data Integrity Failures. © 2025. Available from: https://owasp.org/Top10/2025/A08_2025-Software_or_Data_Integrity_Failures/

[51] A09 Security Logging and Alerting Failures. © 2025. Available from: https://owasp.org/Top10/2025/A09_2025-Security_Logging_and_Alerting_Failures/

[52] A10 Mishandling of Exceptional Conditions. © 2025. Available from: https://owasp.org/Top10/2025/A10_2025-Mishandling_of_Exceptional_Conditions/

12 Přílohy

1) Odkaz na repozitář GitHub

Odkaz na repozitář GitHub

Kompletní zdrojový kód webové aplikace je dostupný ve veřejném repozitáři na platformě GitHub.

Adresa repozitáře:
`https://github.com/Pr3jnar/Life-essentials`

UNIVERZITA HRADEC KRÁLOVÉ

Fakulta informatiky a managementu

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel Trejtnar**
Osobní číslo:	**I2300389**
Studijní program:	**B0613A140033 Aplikovaná informatika**
Specializace:	**Softwarové inženýrství**
Téma práce:	**Life Essentials – Podpora zdravého životního stylu prostřednictvím webové aplikace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cíl práce je vytvořit a implementovat webovou aplikaci pro podporu zdravého životního stylu, která slouží k individualizovanému monitorování a analýze klíčových faktorů životního stylu uživatele. Aplikace zajistí spolehlivé zpracování a vizualizaci dat, čímž uživateli poskytne zpětnou vazbu a zvýší povědomí o vlivu návyků na zdraví.

1. Úvod
2. Analýza současného stavu
3. Návrh řešení
   - Definice požadavků na aplikaci
   - Uživatelské scénáře a návrh uživatelského rozhraní
   - Architektura systému a použité technologie
4. Implementace webové aplikace
   - Popis vývojových prostředí a použitých knihoven/frameworků
   - Realizace klíčových funkcionalit
   - Ukázky implementace
5. Testování a vyhodnocení výsledků
6. Diskuse a doporučení
7. Závěr
8. Seznam použité literatury
9. Přílohy
10. Zadání práce

Rozsah pracovní zprávy:
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. World Health Organization (WHO). Global Strategy on Diet, Physical Activity and Health, Geneve: WHO, 2004
2. World Health Organization (WHO). Healthy Diet: Fact Sheet. Geneva: WHO, aktualizováno 2024
3. Walker, M. (2017). Proč spíme: Odhalte sílu spánku a snů

4. Grinber, M.(2018). FLask Web Development: Developing Web Applications with Python. 2nd ed.
5. Pavlíček, J. (2019). Databázové systémy.
6. Duckett, J. (2014). HTML & CSS: Design and Build Websites.
7. McKinney, W.(2022). Python for Data Analysis. 3rd ed.
8. Hendl, J. (2021). Big Data: Věda o datech – základy a aplikace.

Vedoucí bakalářské práce: **Ing. Martina Husáková, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **25. listopadu 2024**

L.S.

prof. Ing. Mgr. Petra Marešová, Ph.D., MBA v.r.
děkanka

doc. Ing. Pavel Čech, Ph.D. v.r.
vedoucí katedry

V Hradci Králové dne 25. listopadu 2024