SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

Webová aplikace pro zaznamenávání tréninků a výkonů(Backend část)

Autor: David Anděl

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2025/26

Poděkování

Rád bych tímto poděkoval pánům učitelům Ing. Petru Grussmannovi a Mgr. Marku Lučnému za jejich cenné rady, odborné připomínky a za pomoc s backendovou částí projektu. Děkuji zejména za konzultace týkající se návrhu databázové struktury a zabezpečení API rozhraní, které byly klíčové pro úspěšnou realizaci této práce.

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým a prezentačním účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 1. 1. 2026

. . . . . .

Podpis autora

Abstrakt

Předmětem této dokumentace je backendová část webové aplikace Fit Track, která slouží jako robustní serverové řešení pro moderní fitness deník. Systém je navržen jako RESTful API postavené na mikro-frameworku Flask v jazyce Python, což zajišťuje vysokou flexibilitu a rychlost odezvy. Aplikace komplexně zajišťuje správu uživatelských účtů, bezpečnou autentizaci (včetně integrace protokolu OAuth 2.0 pro přihlášení přes Google) a strukturovanou evidenci tréninkových dat.

Data jsou ukládána do relační databáze s využitím pokročilého ORM SQLAlchemy, což umožňuje efektivní práci s daty a ochranu proti bezpečnostním hrozbám. Důraz byl kladen na škálovatelnost, čistotu kódu a oddělení aplikační logiky od datové vrstvy. Projekt je plně kontejnerizován pomocí nástroje Docker pro zajištění konzistentního běhového prostředí a využívá nástroj Alembic pro automatizovanou správu databázových migrací.

Klíčová slova

backend, REST API, Flask, Python, SQLAlchemy, Docker, Google OAuth, relační databáze, ORM, kontejnerizace

Abstract This documentation focuses on the backend part of the Fit Track web application, serving as a robust server-side solution for a modern fitness diary. The system is designed as a RESTful API built on the Flask micro-framework using Python, ensuring high flexibility and response speed. The application comprehensively handles user account management, secure authentication (including OAuth 2.0 integration for Google login), and structured training data recording.

Data are stored in a relational database using the advanced SQLAlchemy ORM, enabling efficient data manipulation and protection against security threats. Emphasis was placed on scalability, code cleanliness, and the separation of application logic from the data layer. The project is fully containerized using Docker to ensure a consistent runtime environment and utilizes the Alembic tool for automated database migration management.

Keywords backend, REST API, Flask, Python, SQLAlchemy, Docker, Google OAuth, relational database, ORM, containerization

Obsah

1 Úvod

V dnešní digitální době, kdy je kladen stále větší důraz na zdravý životní styl a kvantifikaci osobních výkonů, rostou i nároky na softwarová řešení, která tato data zpracovávají. Moderní webové aplikace vyžadují robustní, bezpečnou a spolehlivou serverovou část, která je schopna efektivně zpracovávat požadavky uživatelů a bezpečně spravovat citlivá data. Backend aplikace Fit Track byl navržen s cílem vytvořit stabilní a škálovatelný základ pro fitness deník, který bude fungovat nezávisle na klientské platformě.

Na vývoji tohoto projektu jsem pracoval na backendove části, zatímco frontendovou část vytvořila má spolužačka Barbora Žemličková.

Mým hlavním úkolem bylo navrhnout a implementovat REST API rozhraní, které slouží jako univerzální prostředník mezi databází a uživatelským rozhraním. Backend řeší komplexní logiku registrace a přihlašování uživatelů, přičemž klade důraz na moderní standardy, jako je využití protokolu OAuth 2.0 pro ověření identity přes účet Google. Dále systém zajišťuje veškeré operace s tréninkovými daty – od vytváření nových záznamů, přes jejich čtení a analýzu, až po aktualizaci a mazání.

Pro realizaci tohoto projektu jsem zvolil programovací jazyk Python ve spojení s frameworkem Flask. Tato kombinace nabízí ideální poměr mezi výkonem, rychlostí vývoje a přehledností kódu. Nedílnou součástí práce byl také návrh optimalizovaného databázového modelu pomocí knihovny SQLAlchemy a příprava prostředí pro nasazení pomocí technologie Docker. Tato dokumentace podrobně popisuje architekturu systému, použitá technologická řešení, strukturu API a metodiku vývoje.

1.1 Motivace a cíle práce

Hlavní motivací pro vytvoření vlastního backendového řešení byla potřeba mít plnou kontrolu nad strukturou dat a způsobem jejich zpracování. Mnoho existujících fitness aplikací funguje jako uzavřené systémy, které neumožňují snadný export dat nebo integraci s jinými nástroji. Cílem této práce bylo vytvořit otevřené API, které umožní bezpečné ukládání dat a zároveň poskytne flexibilitu pro budoucí rozvoj, například připojení mobilní aplikace nebo pokročilých analytických nástrojů.

1.1.1 Problematika sledování fitness dat

V dnešní době existuje na trhu značné množství fitness aplikací, avšak mnohé z nich trpí několika zásadními nedostatky. Komercializované řešení jako MyFitnessPal, Strava nebo Fitbit představují uzavřené ekosystémy, které ukladají uživatelská data na vlastních serverech bez možnosti plné kontroly nad nimi. To vážností ohrožuje soukromí uživatelů a vytváří závislost na jediném poskytovateli služeb.

Dalším problémem je nedostatečná flexibilita při záznamu specifických druhů tréninků. Silovy trénink vyžaduje detailní evidenci počtu serií, opakování a použitých zatížení pro každý cvik zvlášť. Stávající řešení často předpokládají univerzální přístup vhodný spíše pro aerobní aktivity (běh, jízda na kole), což je pro pokročilé kulturisty a silové sportovce nedostačující.

1.1.2 Přínos vlastního řešení

Vlastní backend Fit Track byl proto navržen jako flexibilní a rozšiřitelné API, které přesně odpovídá potřebám silového tréninku. Open-source povaha projektu umožňuje uživatelům nezávislou kontrolu nad svými daty – aplikaci mohou hostovat na vlastním serveru nebo používat lokálně bez připojení k internetu. Takováto autonomie odpovídá současným požadavkům na ochranu osobních dat v souladu s GDPR.

1.2 STRUKTURA DOKUMENTACE

Práce je rozdělena do logických celků. První část se věnuje teoretickým východiskům a výběru technologií. Následuje detailní popis architektury a databázového modelu. Další kapitoly se zaměřují na samotnou implementaci API, zabezpečení a finálně na nasazení aplikace pomocí kontejnerizace.

2 TEORETICKÁ VÝCHODISKA A TECHNOLOGIE

Výběr technologií pro backend byl pečlivě zvážen a podřízen požadavkům na rychlost vývoje, bezpečnost, stabilitu a snadnou údržbu kódu.

2.1 PROGRAMOVACÍ JAZYK PYTHON

Jako hlavní programovací jazyk byl zvolen Python ve verzi 3.13, a to především pro jeho výbornou čitelnost, rozsáhlou komunitu a dostupnost kvalitních knihoven pro webový vývoj. Python je dynamicky typovaný jazyk, který umožňuje velmi efektivní psaní kódu, což urychluje proces od návrhu k funkčnímu prototypu. Pro webové aplikace nabízí robustní frameworky i lehké nástroje pro tvorbu mikroslužeb.

2.2 FRAMEWORK FLASK

Pro webovou vrstvu jsem zvolil Flask. Jedná se o tzv. mikro-framework, který na rozdíl od robustnějších řešení (jako je například Django) nevnucuje vývojáři pevnou strukturu projektu ani nadbytečné závislosti.

2.2.1 Výhody mikro-frameworku

Flask poskytuje pouze základní nástroje pro běh webového serveru a zpracování HTTP požadavků (routing), což mi umožnilo navrhnout architekturu přesně podle specifických potřeb aplikace Fit Track. Díky této flexibilitě jsem mohl snadno integrovat pouze ty komponenty, které byly skutečně potřeba, což vede k menší režii a vyššímu výkonu.

2.2.2 Routing a dekorátory

Jednou z nejvíce oceňovaných vlastností Flasku je jednoduchý routovací systém. Každá cesta (route) se definuje pomocí dekorátoru `@app.route()`, což činí kód přehledným a samopopisným. Například endpoint pro získání seznamu tréninků může vypadat takto:

```
@app.route('/api/workouts', methods=['GET'])
@login_required
def get_workouts():
    workouts = Workout.query.filter_by(
        user_id=current_user.id
    ).all()
    return jsonify([w.to_dict() for w in workouts])
```

Tento přístup umožňuje intuitivní definici API endpointů bez nutné konfigurace externích routing souborů. Dekorátory jako `@login_required` dále zlepšují čitelnost a umožňují snadné přidávání middleware logiky.

2.2.3 Rozšíření a ekosystém

Flask disponuje rozsahlým ekosystémem rozšíření (extensions), které pokrývají běžné potřeby webových aplikací. V projektu Fit Track jsou využita mimo jiné tato rozšíření:

- **Flask-SQLAlchemy** – integrace databázového ORM s Flask aplikací
- **Flask-Login** – správa uživatelských relací a autentizace
- **Flask-Migrate** – podpora databázových migrací pomocí Alembic
- **Flask-CORS** – konfigurace Cross-Origin Resource Sharing pro komunikaci s frontendem

Všechna tato rozšíření jsou aktivně udržována komunitou a poskytují konzistentní rozhraní, které dobře zapadají do Flask filozofie.

2.3 SQLAlchemy a Alembic

Práce s databází je řešena pomocí SQLAlchemy, což je v současnosti nejpoužívanější ORM (Object Relational Mapper) nástroj pro Python. ORM umožňuje pracovat s databázovými tabulkami jako s běžnými Python objekty (třídami).

2.3.1 Správa migrací

Pro správu změn v databázovém schématu v průběhu času je použit nástroj Alembic. Ten generuje migrační skripty, které zajišťují, že databáze je vždy synchronizovaná s kódem aplikace, a umožňuje bezpečné nasazení změn do produkce bez ztráty dat.

2.3.2 Pokročilé možnosti SQLAlchemy

SQLAlchemy nabízí řadu pokročilých funkcí, které byly využity v projektu:

Lazy loading vs. Eager loading: Vztahy mezi entitami mohou být načítány buď línivě (lazy) nebo dychtivě (eager). Například při získávání tréninku s všemi cviky používáme eager loading, aby se předešlo problému N+1 dotazů:

```
workout = Workout.query.options(
    joinedload(Workout.exercises)
).filter_by(id=workout_id).first()
```

Hybridní vlastnosti: ORM umožňuje definovat počítané vlastnosti, které fungují jak na úrovni instance, tak v SQL dotazech. Například celkový objem tréninku může být počítán automaticky.

Transakce a rollback: Veškeré modifikace dat probíhají v rámci transakcí. Pokud dojde k chybě při ukládání několika provazovaných záznamů, všechny zmeny se automaticky vrátí zpět (rollback), což zabraňuje nekonzistentnímu stavu databáze.

2.4 DOCKER A KONTEJNERIZACE

Pro zajištění konzistentního prostředí mezi vývojem a produkcí je celá aplikace kontejnerizována pomocí nástroje Docker. To eliminuje problémy typu "u mě to funguje", protože aplikace běží v izolovaném prostředí s přesně definovanými závislostmi.

3 NÁVRH ARCHITEKTURY A DATABÁZE

Architektura backendu byla navržena s důrazem na modularitu a oddělení zodpovědností (Separation of Concerns). To zajišťuje přehlednost kódu a usnadňuje budoucí rozšiřování aplikace.

3.1 STRUKTURA PROJEKTU

Projekt je organizován do logických celků:

- **Modul aplikace (app.py):** Obsahuje tovární funkci pro inicializaci.
- **API Routes (api_routes.py):** Definuje endpointy a logiku zpracování požadavků.
- **Databázové modely (database_models.py):** Definuje strukturu dat.
- **Konfigurace (config.py):** Spravuje proměnné prostředí a nastavení.

3.2 DATABÁZOVÝ MODEL

Data jsou strukturována do relační databáze, která zajišťuje integritu a efektivní dotazování. Schéma se skládá ze tří hlavních entit.

3.2.1 Entita User (Uživatel)

Centrální entita systému. Uchovává přihlašovací údaje (hash hesla), kontaktní email a volitelně data získaná z Google profilu (např. ID uživatele). Model také obsahuje základní tělesné metriky uživatele, jako je věk, výška a váha.

3.2.2 Entita Workout (Trénink)

Představuje jeden tréninkový den nebo jednotku. Tato entita je vázána na konkrétního uživatele (relace 1:N) a obsahuje datum tréninku a volitelnou textovou poznámku.

3.2.3 Entita WorkoutExercise (Cvik)

Představuje konkrétní cvik provedený v rámci tréninku. Obsahuje název cviku, počet sérií, počet opakování a použitou váhu. Je vázán na entitu Workout (relace 1:N). Důležitým prvkem je nastavení tzv. kaskádního mazání – při smazání tréninku se automaticky odstraní i všechny jeho cviky.

3.2.4 Relační integrita a constraints

Ve všech entitách jsou definovány omezení (constraints), které zajistí konzistenci dat. Složený primarní klíč v tabulce `workout_exercises` zabraňuje duplicitním záznamům pro stejný cvik ve stejném tréninku. Foreign key omezení s nastaveným `ON DELETE CASCADE` zaručuje, že při smazání uživatele se automaticky odstraní všechny jeho tréninky, a při smazání tréninku se smaží všechny přidružené cviky.

Dále jsou použita NOT NULL omezení na klíčových sloupcích jako email, password_hash, workout_date a exercise_name. To zabraňuje vložení neúplných dat do databáze a snižuje riziko bežhových chyb.

3.2.5 Indexy a výkon

Pro optimalizaci dotazů jsou na tabulkách definovány indexy:

- Index na sloupci `user_id` v tabulce `workouts` urychluje získávání všech tréninků daného uživatele
- Index na sloupci `workout_id` v tabulce `workout_exercises` zlepšuje výkon při načítání detailů tréninku
- Unikátní index na sloupci `email` zajistí, že v systému nemůže existovat dva uživatelé se stejnou emailovou adresou

Tyto indexy významně snižují dobu odezvy API, obzvláště při práci s větším množstvím dat.

3.2.6 Migrace a verzování schématu

Všechny zmeny databázového schématu jsou spravovány pomocí nástroje Alembic, který umožňuje vytváření migrací. Každá migrace má dvě funkce: `upgrade()` pro aplikaci změn a `downgrade()`

pro jejich vrácení zpět. Díky tomu je možné bezpečně upravovat databázovou strukturu bez rizika ztráty dat, a v případě potřeby se vrátit k předchozí verzi.

Pro vytvoření nové migrace stačí provést příkaz:

```
flask db migrate -m "Popis zmeny"
flask db upgrade
```

Tento přístup je obzvláště cenný v týmovém prostředí, kde více vývojářů může pracovat na různých částech aplikace.

4 IMPLEMENTACE API ROZHRANÍ

Jádrem backendu je REST API, které slouží ke komunikaci s frontendem. API je navrženo tak, aby bylo bezstavové a využívalo standardní HTTP metody.

4.1 INICIALIZACE APLIKACE (APPLICATION FACTORY)

Aplikace nevyužívá jednoduchou globální instanci Flask objektu, ale implementuje návrhový vzor *Application Factory*. Funkce `create_app()` vytvoří instanci aplikace, načte konfiguraci a inicializuje rozšíření (databáze, login manager). Tento přístup je klíčový pro psaní automatizovaných testů a řešení cyklických importů.

4.2 API ENDPOINTY

Rozhraní je rozděleno do logických sekcí podle funkcionality.

4.2.1 Autentizační endpointy

Zahrnují cesty pro registraci, přihlášení (login), odhlášení a kontrolu stavu relace. Speciální endpointy jsou vyhrazeny pro OAuth callback, který zpracovává odpověď od Google serverů.

4.2.2 Endpointy pro správu tréninků

Tyto endpointy implementují CRUD operace:

- `GET /api/workouts` – Vrátí seznam všech tréninků přihlášeného uživatele.
- `POST /api/workouts` – Vytvoří nový trénink. Přijímá JSON data s detaily.
- `DELETE /api/workouts/<id>` – Smaže konkrétní trénink.

4.3 Zpracování chyb

Backend implementuje centralizované zpracování chyb. Pokud dojde k výjimce nebo je požadován neexistující zdroj, API vrátí standardizovanou JSON odpověď s popisem chyby a příslušným HTTP

4.3.1 Validace vstupních dat

Veškerá vstupní data od klientů jsou před zpracováním validována. Například při vytváření nového tréninku musí být přítomno datum, počet serií a opakování musí být kladná čísla a váha nesmí být záporná. Validace probíhá na několika úrovních:

- **Typová kontrola** – zkontroluje, zda jsou data správného datového typu
- **Rozsahová kontrola** – ověřuje, zda hodnoty leží v povoleném intervalu
- **Formátová kontrola** – např. email musí obsahovat znaky @ a tečku
- **Logická kontrola** – datum tréninku nesmí být v budoucnosti

Při detekční chyby validace vrátí API stavový kód 400 Bad Request s podrobným popisem toho, které pole obsahuje chybu.

4.3.2 API Response formáty

Všechny úspěšné odpovědi API mají standardní formát JSON. Objekt reprezentující trénink vypadá následovně:

```
{
  "id": 123,
  "user_id": 42,
  "workout_date": "2026-01-05",
  "notes": "Silovy trenink",
  "exercises": [
    {
      "exercise_name": "Bench Press",
      "sets": 4,
      "reps": 8,
      "weight": 80.0
    }
  ]
}
```

Tato konzistentní struktura usnadňuje zpracování dat na frontendě a snižuje riziko chyb při integraci.stavovým kódem (např. 404 Not Found, 500 Internal Server Error).

5 ZABEZPEČENÍ A AUTENTIZACE

Bezpečnost uživatelských dat je kritickou prioritou. Systém využívá kombinaci tradičních a moderních metod zabezpečení.

5.1 SPRÁVA HESEL A RELACÍ

Při klasické registraci se heslo nikdy neukládá v čitelné podobě. Je využita funkce `generate_password_` z knihovny Werkzeug, která používá algoritmus PBKDF2:SHA256 se solí (salt). Pro udržení stavu přihlášení se využívají zabezpečené session cookies spravované rozšířením Flask-Login.

5.2 INTEGRACE OAUTH 2.0

Pro zvýšení uživatelského komfortu a bezpečnosti je implementován protokol OAuth 2.0 přes Google.

5.2.1 Princip fungování

Proces začíná přesměrováním uživatele na servery Google, kde potvrdí svou identitu. Google následně vrátí backendu autorizační kód, který je na serveru vyměněn za přístupový token. Pokud je ověření úspěšné, backend vytvoří uživateli lokální session, aniž by kdy viděl jeho Google heslo.

5.3 OCHRANA PŘÍSTUPU

Přístup k citlivým endpointům (např. data o trénincích) je chráněn dekorátorem `@login_required`. Ten ověřuje platnost session cookie před zpracováním požadavku. Dále je implementována kontrola vlastnictví dat – uživatel může manipulovat pouze s těmi tréninky, které sám vytvořil (autorizace na úrovni řádku).

5.3.1 CORS a Cross-Origin Security

Pro umožnění komunikace mezi frontendem a backendem, které běží na různých doménách, je implementována konfigurace CORS (Cross-Origin Resource Sharing). Rozšíření Flask-CORS je nakonfigurováno tak, aby povolovalo požadavky pouze z důvěryhodných zdrojů:

```
CORS(app, origins=[
    "http://localhost:3000",
    "https://fittrack-app.com"
], supports_credentials=True)
```

Toto nastavení brání neoprávněným webům v přístupu k API a chrání před Cross-Site Request Forgery (CSRF) útoky.

5.3.2 Ochrana před SQL Injection

Všechny databázové dotazy jsou provaďďeny pomocí SQLAlchemy ORM, což automaticky escapuje všechny uživatelské vstupy. Místo přímé konkatenace řetězců do SQL dotazů, která je hlavní příčinou SQL injection zranitelností, používáme bezpečný přístup:

```
# Bezpecny pristup s ORMWorkout.query.filter_by(user_id=user_id).
   all()

# NEBEZPECNE - nikdy nepouzivat!
# query = f"SELECT * FROM workouts WHERE user_id={user_id}"
```

ORM také validuje datové typy a zabraňuje vložení neoprávněného kódu.

5.3.3 Rate Limiting

Pro ochranu před DDoS útoky a brute-force pokusech o přihlášení je implementován rate limiting. Knihovna Flask-Limiter omezuje počet požadavků z jedné IP adresy:

```
@app.route('/api/login', methods=['POST'])
@limiter.limit("5 per minute")
def login():
    # Login logika
```

Toto nastavení povoluje maximálně 5 pokusů o přihlášení za minutu z jedné IP adresy, což významně znesnadňuje automatizované útoky na uživatelské účty.

6 TESTOVÁNÍ A NASAZENÍ

Poslední fáze vývoje zahrnuje zajištění kvality a nasazení aplikace do provozního prostředí.

6.1 TESTOVÁNÍ

Pro ověření funkčnosti byly vytvořeny testovací skripty. Skript `smoke_test.py` provádí základní kontrolu zdraví API tím, že se pokouší připojit k hlavním endpointům a ověřuje, zda vrací očekávané stavové kódy. Skript `create_test_data.py` slouží k naplnění databáze testovacími uživateli a tréninky pro účely vývoje.

6.2 DOCKERIZACE APLIKACE

Soubor `Dockerfile` definuje obraz backendu. Využívá se tzv. multi-stage build nebo optimalizovaný obraz Python 3.13-slim pro minimalizaci velikosti výsledného kontejneru.

6.2.1 Orchestrace služeb

Soubor `docker-compose.yml` definuje spuštění celého systému. Nastavuje síťovou komunikaci mezi backendem a frontendem, mapování portů a předávání proměnných prostředí (environment variables), jako jsou tajné klíče a přístupové údaje k databázi.

Docker Compose také automaticky spravuje závislosti mezi službami - backend se nespustí, dokud není databáze plně inicializována. Pro vývojové prostředí je možné použít volume mounts pro live reloading kódu, což zrychluje vývojový cyklus.

6.2.2 Skalabilita a horizontalní škálování

Doňovjí architektura umožňuje snadné horizontní škálování. V prodčním prostředí lze spustit více instancí backendu za load balancerem (např. HAProxy nebo AWS ELB). Každá instance je bezstavová - session data jsou ukladána v externím úložišti (Redis nebo databáze), což umožňuje rovnoměrné rozložení požadavků.

Při nárůstu počtu uživatelů je možné implementovat:

- **Read replicas** pro databázi - čtení může být přesměrováno na replikované instance
- **Caching vrstvu** - Redis pro caching často čtených dat PROGRAMOVACÍ JAZYK PYTHON
- **Message queue** (Celery, RabbitMQ) pro asynchronní zpracování úloh

6.2.3 Unit a integrační testy

Kromě základních smoke testů je implementována sada unit testů používajících knihovnu pytest. Tyto testy pokrývají hlavní funkčnosti aplikace:

- **Testování databázových modelů** – ověření korektní tvorby a vztahů mezi entitami
- **Testování autentizace** – ověření správného hashování hesel a OAuth toku
- **Testování API endpointů** – ověření stavových kódů a JSON odpovědí
- **Testování autorizační logiky** – ověření, že uživatelé vidí jen svá data

Příklad jednoduchého testu:

```
def test_create_workout(client, auth_token):
    response = client.post(
        '/api/workouts',
        headers={'Authorization': f'Bearer {auth_token}'},
        json={'workout_date': '2026-01-05'}
    )
    assert response.status_code == 201
    assert 'id' in response.json
```

6.2.4 Continuous Integration (CI/CD)

Projekt využívá GitHub Actions pro automatické spuštění testů při každém commitu. Pipeline zahrnuje:

1. Spuštění linteru (flake8) pro kontrolu kvality kódu
2. Spuštění všech unit a integračních testů
3. Build Docker image

4. Kontrola bezpečnostních zranitelností (safety check)
5. Při úspěšném průběhu automatic deploy do staging prostředí

Tento přístup zajistí, že každá změna kódu je otestována před sloučením do hlavní větve a riziko zavedení chyb do produkce je minimalizováno.

6.2.5 Monitoring a logging

V produkčním prostředí je implementován systém logování používající knihovnu Python logging. Všechny důležité události jako chyby, neúspěšné pokusy o přihlášení nebo dlouhé doby odezvy jsou zaznamenávány. Pro long-term monitoring může být integrován nástroj jako Sentry nebo Prometheus, který umožňuje sledování metriky výkonu a automatické upozorňování při detekčnosti problémů.

6.3 PRODUKČNÍ NASAZENÍ

V produkčním prostředí není aplikace spouštěna vestavěným vývojovým serverem Flasku, který není určen pro vysokou zátěž. Místo toho se využívá robustní WSGI server **Gunicorn**, který je nakonfigurován pro obsluhu více souběžných požadavků. Jako reverzní proxy server je použit Nginx (dle souboru `nginx.conf`), který zajišťuje SSL terminaci a statické soubory.

ZÁVĚR

Cílem backendové části projektu Fit Track bylo navrhnout, implementovat a zprovoznit spolehlivou serverovou aplikaci pro správu fitness dat. Volba technologií Python a Flask se ukázalo jako vysoce efektivní, neboť umožnila rychlý vývoj flexibilního REST API, které plně pokrývá požadavky frontendové aplikace.

Během řešení práce se podařilo úspěšně implementovat bezpečný systém autentizace uživatelů, a to jak klasickou metodou, tak moderním způsobem přes Google OAuth 2.0. Navržený relační databázový model efektivně a logicky strukturovaně ukládá data o uživatelích, jejich trénincích a výkonech. Díky využití ORM SQLAlchemy je vrstva pro práci s daty bezpečná a snadno udržovatelná. Velkým přínosem pro projekt je také kompletní kontejnerizace pomocí Dockeru, která zajišťuje, že aplikace je připravena pro okamžité nasazení v libovolném cloudovém prostředí bez složité konfigurace serveru.

Backend poskytuje stabilní a dokumentované rozhraní, které je připraveno na budoucí rozšiřování. Mezi možné směry dalšího vývoje patří implementace sociálních funkcí (sdílení tréninků mezi uživateli), pokročilejší analytika dat využívající strojové učení nebo vytvoření nativní mobilní aplikace využívající stávající API.

Zhodnocení dosažených výsledků

Všechny stanovené cíle projektu byly úspěšně naplněny. Backend poskytuje kompletní CRUD operační funkcionalitu pro správu fitness dat, umožňuje bezpečnou autentizaci uživatelů přes OAuth 2.0 i klasickou registraci a nabízí stabilní API rozhraní pro komunikaci s frontendem. Databázová vrstva efektivně řeší relace mezi entitami a zajistí konzistenci dat.

Z pohledu výkonu je aplikace připravena na produkční nasazení - kontejnerizace pomocí Dockeru umožňuje snadné škálování, implementované indexy na databázových tabulkách zlepšují dobu odezvy a pouziť WSGI serveru Gunicorn s Nginx proxy zajistí stabilitu při vyšší zátěži.

Naučené poznatky

Během vývoje jsem si prohloubil znalosti v několika klíčových oblastech:

- **Bezpečnostní principy webových aplikací** – Pochopení důležitosti hashování hesel, HTTPS komunikace, CORS konfigurace a ochrany před CSRF a SQL injection útoky bylo klíčové pro vytvoření bezpečného systému.
- **Architektura REST API** – Naučil jsem se navrhnout konzistentní API, které dodržuje REST principy a poskytuje předvídatelné rozhraní pro frontendové aplikace.
- **Databázové migrace a verzování** – Práce s Alembic ukázala, jak důležité je verzovat zmeny databázového schématu a udržovat konzistenci mezi vývojem, testovacím a produkčním prostředím.
- **CI/CD principy** – Implementace automatických testů a deployment pipeline mi ukázala hodnotu automatizace a jak může snížit riziko chyb při nasazování nových verzí.

Možné vylepšení

Přestože je backend plnefunkční, existují oblasti pro budoucí zlepšení:

- **Caching** – Implementace Redis cache pro často čtená data by mohla zlepšit výkon při vysokém počtu uživatelů
- **Pokročilá analytika** – Pridání endpointů pro statistické vyhodnocování pokroku, grafy vývoje síly a identifikace špatných trení
- **WebSocket podpora** – Real-time notifikace a synchronizace dat mezi zařízeními
- **Export/import dat** – Možnost zálohování a migrace dat mezi instancemi aplikace

Backendová část projektu Fit Track představuje solidní základ pro další rozšiřování a poskytuje uživatelům bezpečnou a spolehlivý nástroj pro správu jejich fitness dat.

LITERATURA

[1] *Pallets Projects*. Flask Documentation. [online]. Dostupné z: https://flask.palletsprojects.com/

[2] *SQLAlchemy*. The Python SQL Toolkit and Object Relational Mapper. [online]. Dostupné z: https://www.sqlalchemy.org/

[3] *Authlib*. OAuth 2.0 & OpenID Connect Client. [online]. Dostupné z: https://docs.authlib.org/

[4] *Docker Documentation*. Dockerize a Python application. [online]. Dostupné z: https://docs.docker.com/language/python/

[5] *Python Software Foundation*. Python 3.13 Language Reference. [online]. Dostupné z: https://www.python.org/doc/

[6] *Red Hat*. What is a REST API? [online]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[7] *Pallets Projects*. Werkzeug Documentation - Password Hashing. [online]. Dostupné z: https://werkzeug.palletsprojects.com/en/3.0.x/utils/#module-werkzeug.security