Střední průmyslová škola, Ústí nad Labem, Resslova 5



Správce neuronových sítí

Dokumentace k ročníkové práci

**Autor:** Mykhailo Ivashchenko

**Třída:** 4ITC

**Vedoucí práce:** Ing. Petr Haberzettl 2025/2026

Prohlášení

Prohlašuji, že jsem ročníkovou práci na téma „Správce neuronových sítí“ vypracoval samostatně a s použitím uvedené literatury a pramenů.

V Ústí nad Labem dne 14.09.2025

……..…………………….

Poděkování

Chtěl bych poděkovat Ing. Petru Haberzettlovi za vedení mé ročníkové práce, cenné rady a odborný dohled.

Anotace

V úvodu je popsán původní cíl projektu. Dokument dále obsahuje teoretickou část, která se dělí na rešerši a technologie. V rešerši je popsáno, čím byl projekt inspirován, a v části „technologie“ jsou popsány všechny technologie použité při vytvoření projektu. Následuje praktická část, která se skládá ze tří částí: návrh, produktizace a popis z pohledu uživatele. Návrh obsahuje screenshoty ze stránek. V produktizaci jsou ukázky kódu a popsané algoritmy. Popis z pohledu uživatele se zaměřuje na to, jak web vypádá z pohledu uživatele. V závěru je popsáno, co se skutečně podařilo vytvořit z původního cíle uvedeného v úvodu.

Klíčová slova

Neuronová síť, trénovací data, umělá inteligence, strojové učení, hluboké učení, zpracování obrazů

Obsah

[1 Teoretická část 8](#_Toc218847692)

[1.1 Rešerše 8](#_Toc218847693)

[1.1.1 Google Colab 8](#_Toc218847694)

[1.1.2 Kaggle 8](#_Toc218847695)

[1.1.3 Hugging Face 9](#_Toc218847696)

[1.2 Technologie 10](#_Toc218847697)

[1.2.1 Python 10](#_Toc218847698)

[1.2.2 Py Torch 10](#_Toc218847699)

[1.2.3 Docker 10](#_Toc218847700)

[1.2.4 Alembic 10](#_Toc218847701)

[1.2.5 PostgreSQL 10](#_Toc218847702)

[1.2.6 Redis 10](#_Toc218847703)

[1.2.7 React 10](#_Toc218847704)

[1.2.8 ChatGPT 10](#_Toc218847705)

[2 Praktická část 12](#_Toc218847706)

[2.1 Návrhy 12](#_Toc218847707)

[2.2 Produktizace 13](#_Toc218847708)

[2.3 Popis pro uživatele 19](#_Toc218847709)

Úvod

Hlavním cílem ročníkové práce je vytvořit webovou aplikaci sloužící jako správce neuronových sítí, která umožní uživatelům vytvářet, konfigurovat, učit a spravovat vlastní modely neuronových sítí v přehledném a uživatelsky přívětivém prostředí. Projekt je zaměřen na propojení teoretických principů umělé inteligence s praktickým využitím moderních technologií a má za cíl zpřístupnit práci s neuronovými sítěmi i méně zkušeným uživatelům.

Motivací k výběru tohoto tématu je rostoucí význam umělé inteligence a strojového učení v současném světě, zejména v oblasti zpracování obrazu, automatizace a analýzy dat. Neuronové sítě se dnes využívají v mnoha praktických aplikacích, avšak jejich používání bývá často spojeno s technickou složitostí a nutností práce v programovacím prostředí. Tento projekt si klade za cíl tuto bariéru snížit a nabídnout nástroj, který umožní experimentování s neuronovými sítěmi prostřednictvím webové aplikace bez nutnosti hlubokých znalostí programování. Zároveň je téma zvoleno z osobního zájmu o oblast umělé inteligence a její praktické využití.

Dalším cílem práce je návrh systému, který umožní práci s datovými sadami, jejich stahování, zpracování a opakované použití. Aplikace by měla podporovat ukládání projektů, modelů a datasetů tak, aby bylo možné navázat na předchozí práci bez nutnosti opakované konfigurace. Součástí cíle je také vizualizace průběhu učení neuronových sítí pomocí grafů, které uživateli poskytnou přehled o vývoji ztrátové funkce a celkové kvalitě učení modelu.

Projekt si dále klade za cíl umožnit nastavování základních parametrů učení, jako je volba optimalizátoru, ztrátové funkce nebo plánovače učící rychlosti, a nabídnout možnost rozšíření systému o pokročilejší modely, například pro detekci objektů v reálném čase. Aplikace by měla být navržena modulárně tak, aby bylo možné její další rozšiřování a úpravy.

Očekáváním od této práce je vytvoření funkčního a stabilního systému, který bude schopen demonstrovat celý proces práce s neuronovou sítí – od vytvoření projektu, přes přípravu dat, až po učení modelu a vyhodnocení výsledků. Výsledná aplikace by měla sloužit nejen jako praktický nástroj, ale také jako vzdělávací pomůcka, která usnadní pochopení principů neuronových sítí a strojového učení.

# Teoretická část

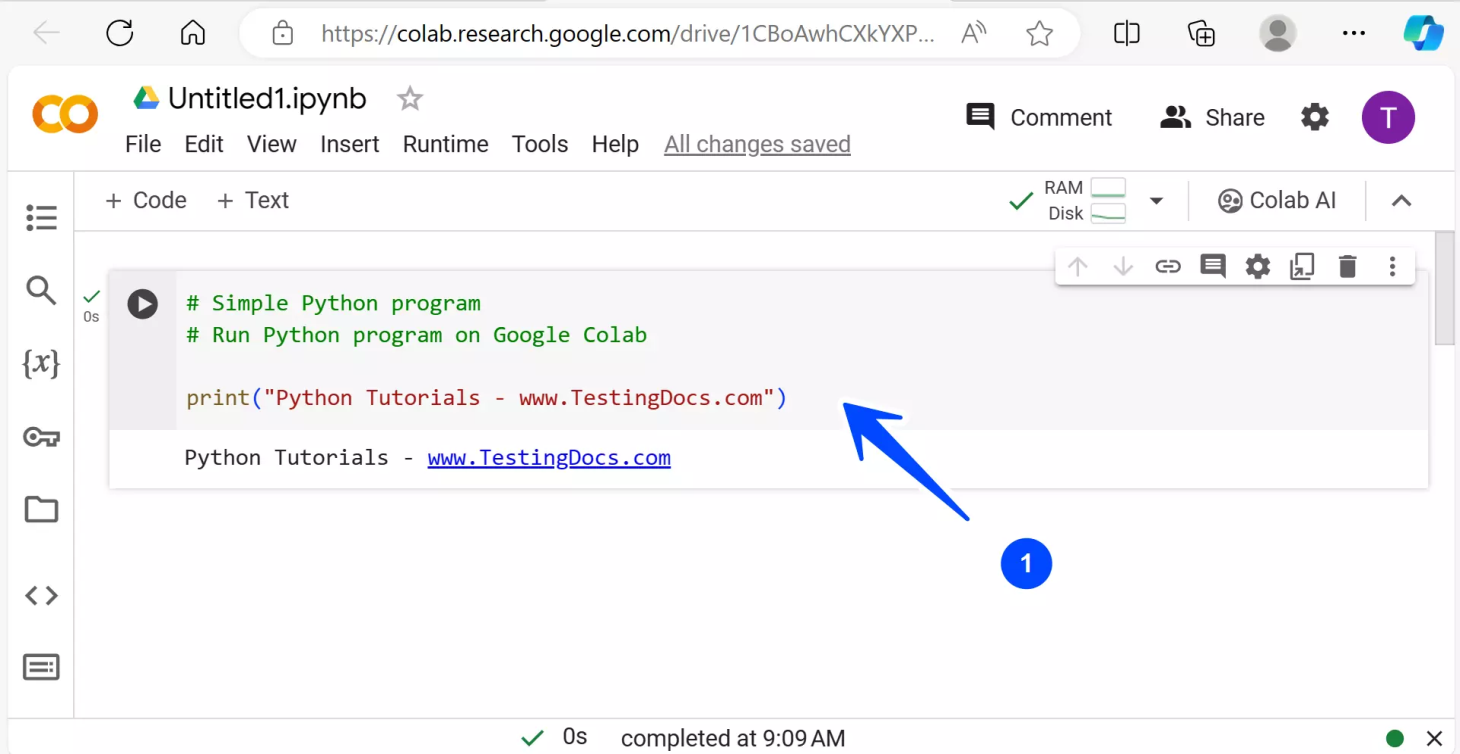
## Rešerše

Sem napište text vaší rešerše.

### Google Colab

Google Colab je webová platforma umožňující spouštění Python kódu přímo v prohlížeči bez nutnosti lokální instalace vývojového prostředí. Je často využívána při výuce strojového učení, protože nabízí snadný přístup k výpočetním prostředkům a podporuje populární knihovny, jako je PyTorch nebo TensorFlow.

Inspirací pro tento projekt byla především jednoduchost práce v prostředí, kde se uživatel může soustředit na samotné experimentování s modely a nemusí řešit technické detaily instalace či konfigurace systému. Google Colab rovněž umožňuje ukládání a opětovné načítání výsledků práce, což je důležitý aspekt i pro navrhovanou aplikaci. (1)

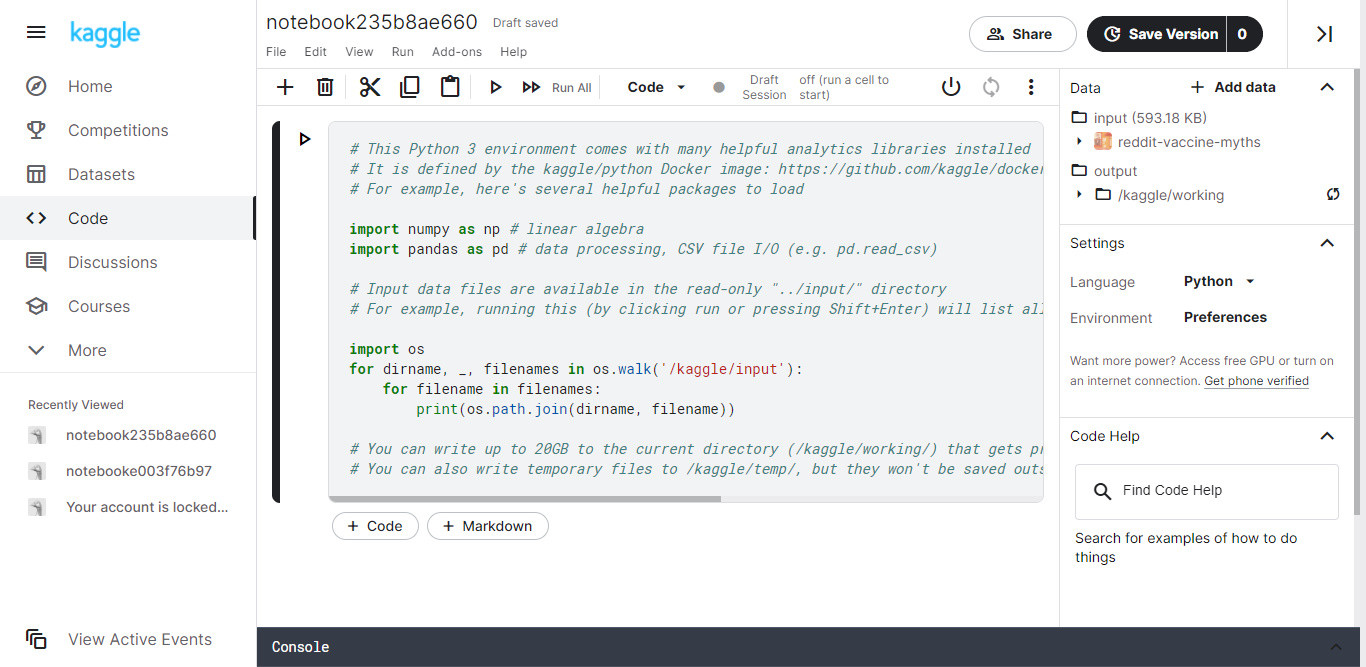


Obrázek - Prostředí Google Colab

### Kaggle

Platforma Kaggle je známá především jako prostředí pro datové soutěže a práci s datovými sadami. Nabízí velké množství veřejně dostupných datasetů a nástroje pro jejich zpracování a analýzu. Pro uživatele je výhodné, že má na jednom místě data, kód i možnost vizualizace výsledků.

V rámci této práce byl Kaggle inspirací zejména v oblasti správy datových sad. Myšlenka mít dataset uložený na jednom místě a být schopen jej opakovaně využívat bez nutnosti neustálého ručního nastavování byla jedním z důvodů, proč je správa datasetů důležitou součástí navrhované aplikace. (2)



Obrázek - Prostředí Kaggle

### Hugging Face

Knihovna Hugging Face Datasets poskytuje standardizované rozhraní pro práci s velkým množstvím datasetů určených pro strojové učení. Umožňuje jejich snadné stahování, načítání a předzpracování.

V tomto projektu byla knihovna využívána především jako zdroj inspirace pro strukturovaný přístup k práci s daty. Důraz je kladen na to, aby uživatel nemusel opakovaně psát stejný kód pro načítání a zpracování datasetů, ale mohl se soustředit na samotné učení neuronových sítí. (3)

## Technologie

### Python

Python je vysokoúrovňový programovací jazyk, který je широко využíván v oblasti datové analýzy, strojového učení a vývoje backendových aplikací. V rámci projektu je Python použit jako hlavní jazyk serverové části aplikace. Slouží k implementaci logiky trénování neuronových sítí, zpracování datasetů a komunikace mezi jednotlivými službami systému. Volba Pythonu byla ovlivněna především autorovou předchozí zkušeností s tímto jazykem a jeho silnou podporou v oblasti AI.

### Py Torch

PyTorch je knihovna pro strojové učení a hluboké neuronové sítě, vyvinutá s důrazem na flexibilitu a práci s dynamickými výpočetními grafy. V projektu je PyTorch využit pro návrh architektury neuronových sítí, jejich trénování a vyhodnocování výsledků. Volba této knihovny vychází z praktických zkušeností autora a její časté využití v akademickém i průmyslovém prostředí.

### Docker

Docker je nástroj pro kontejnerizaci aplikací, který umožňuje spouštět software v izolovaném a reprodukovatelném prostředí. V projektu je Docker použit k zajištění konzistentního běhu jednotlivých služeb (např. backendu a databáze) bez ohledu na konkrétní operační systém. Díky tomu je možné aplikaci snadno nasadit a spustit i na jiném zařízení.

### Alembic

Alembic je nástroj pro správu verzí databázového schématu.  
V rámci projektu slouží k řízení změn struktury databáze, například při přidávání nových tabulek nebo úpravě existujících polí. Tím je zajištěna konzistence databáze během vývoje aplikace.

### PostgreSQL

PostgreSQL je relační databázový systém s otevřeným zdrojovým kódem.  
V projektu je využit pro ukládání informací o projektech, nastaveních trénování, datech a výsledcích experimentů. PostgreSQL byl zvolen díky své stabilitě, rozšířeným možnostem a častému využití v serverových aplikacích.

### Redis

Redis je databázový systém typu key-value, optimalizovaný pro rychlou práci s daty v operační paměti. V tomto projektu je Redis využíván pro ukládání autentizačních tokenů uživatelů. Díky tomu je možné efektivně spravovat přihlášení, ověřování uživatelů a případné odhlašování bez nutnosti častého přístupu k hlavní relační databázi. Použití Redis přispívá ke zvýšení výkonu a bezpečnosti systému, zejména při práci s časově omezenými přístupovými tokeny.

### React

React je knihovna pro tvorbu uživatelského rozhraní webových aplikací. V tomto projektu je React použit k vytvoření webového rozhraní, které umožňuje uživateli nastavovat parametry projektu, vybírat dataset, volit ztrátovou funkci a sledovat průběh trénování modelu. React byl zvolen kvůli přehledné správě stavu aplikace a modulárnímu přístupu k tvorbě rozhraní.

### ChatGPT

ChatGPT je jazykový model založený na umělé inteligenci. V rámci projektu byl využíván jako podpůrný nástroj při návrhu architektury aplikace, ladění kódu a formulaci textové dokumentace. ChatGPT nebyl použit jako součást výsledné aplikace, ale pouze jako pomocný prostředek během vývoje.

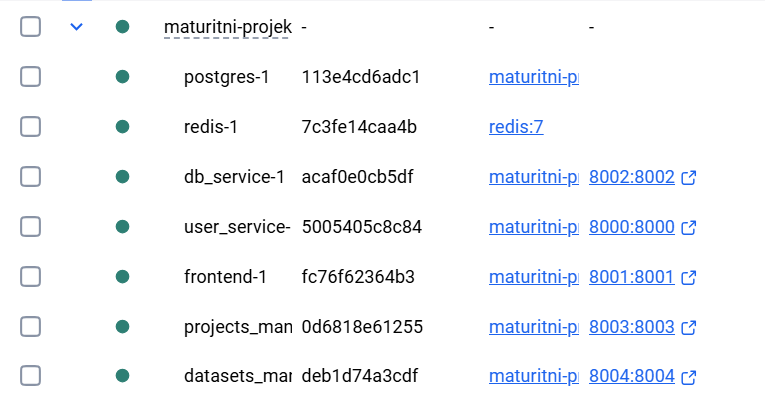
# Praktická část

Praktická část práce se zaměřuje na implementaci distribuované aplikace pro správu a trénování neuronových sítí.

## Návrhy

Na začátku projektu bylo nutné navrhnout celkovou architekturu aplikace. Cílem bylo vytvořit modulární a přehledné řešení, které umožní snadné rozšiřování o další funkce v budoucnu. Aplikace je rozdělena na backendovou a frontendovou část, které spolu komunikují pomocí rozhraní API.

Backendová část byla navržena jako sada samostatných služeb, zodpovědných za správu projektů, uživatelů, datasetů a procesu učení neuronových sítí. Důraz byl kladen na oddělení logiky učení modelů od uživatelského rozhraní. Pro ukládání dat byl navržen relační databázový model, který zahrnuje entity jako projekt, uživatel, dataset a konfigurace učení.



Obrázek - Seznam jednotlivých služeb

Databázový kontejner **PostgreSQL** obsahuje relační databázi s tabulkami určenými pro ukládání uživatelských dat, informací o projektech a jejich konfiguracích. Tato databáze slouží jako hlavní perzistentní úložiště aplikace.

Služba **Redis** je využita jako paměťové úložiště pro autentizační tokeny. Tyto tokeny umožňují ověřování uživatelů bez nutnosti opakovaného přístupu do databáze, čímž se zvyšuje výkon a bezpečnost systému.

**DB-service** představuje nadřazený servis nad databází PostgreSQL. Zajišťuje komunikaci s databází prostřednictvím API endpointů a abstrahuje databázovou logiku od ostatních částí systému. Ostatní služby tak s databází komunikují výhradně přes tento servis.

**User service** je zodpovědná za uživatelskou logiku aplikace. Zajišťuje registraci, přihlášení a odhlášení uživatelů, správu uživatelských profilů a práci s autentizačními tokeny.

Frontendový kontejner obsahuje uživatelské rozhraní aplikace, tvořené HTML stránkami, CSS styly a JavaScriptovými skripty. Tato část aplikace umožňuje uživateli interakci se systémem, nastavování projektů a sledování průběhu učení neuronových sítí.

Služba **Projects manager** spravuje všechny uživatelské projekty a jejich konfigurace. Je zodpovědná za řízení procesu učení neuronových sítí, včetně spouštění trénování, ukládání výsledků a sledování průběhu učení.

**Datasets manager** zajišťuje práci s trénovacími daty. Umožňuje stahování datasetů, jejich zpracování, přípravu pro učení neuronových sítí a jejich ukládání pro opakované použití.

## Produktizace

### Celkový popis

Aplikace je navržena jako systém mikroslužeb, které jsou provozovány v samostatných kontejnerech pomocí technologie Docker. Tento přístup umožňuje oddělení jednotlivých funkčních částí systému, jejich nezávislé nasazování a jednodušší údržbu. Pro orchestraci služeb je využit nástroj Docker Compose, který definuje závislosti mezi službami, jejich konfiguraci a způsob spuštění.

Databázový kontejner PostgreSQL obsahuje relační databázi s tabulkami určenými pro ukládání uživatelských dat, informací o projektech a jejich konfiguracích. Databáze slouží jako hlavní perzistentní úložiště celé aplikace.

Služba Redis je využita jako paměťové úložiště pro autentizační tokeny. Tyto tokeny umožňují ověřování uživatelů bez nutnosti opakovaného přístupu do databáze, čímž se snižuje zátěž databázového systému a zvyšuje se výkon aplikace.

Služba DB-service představuje aplikační vrstvu nad databází PostgreSQL. Zajišťuje komunikaci s databází prostřednictvím API endpointů a abstrahuje databázovou logiku od ostatních částí systému. Ostatní služby s databází komunikují výhradně prostřednictvím tohoto rozhraní.

User service je zodpovědná za uživatelskou logiku aplikace. Zajišťuje registraci, přihlášení a odhlášení uživatelů, správu uživatelských profilů a práci s autentizačními tokeny.

Frontendový kontejner obsahuje uživatelské rozhraní aplikace tvořené HTML stránkami, CSS styly a JavaScriptovými skripty. Umožňuje uživateli interakci se systémem, nastavování projektů a sledování průběhu učení neuronových sítí.

Služba Projects manager spravuje uživatelské projekty a jejich konfigurace. Je zodpovědná za řízení procesu učení neuronových sítí, včetně spouštění trénování, ukládání výsledků a sledování průběhu učení.

Datasets manager zajišťuje práci s trénovacími daty. Umožňuje stahování datasetů, jejich zpracování, přípravu pro učení neuronových sítí a jejich ukládání pro opakované použití.

### Databazový model

Následující část popisuje databázový model aplikace, který je realizován pomocí ORM knihovny SQLAlchemy. Datový model definuje základní entity systému a jejich vzájemné vztahy, zejména uživatele, projekty a datasety používané pro učení neuronových sítí.

from **enum** import **Enum**

from **sqlalchemy** import **Column**, **Integer**, **String**, **Date**, **DateTime**, func, **ForeignKey**, **Enum** as **EnumType**, **Text**

from **sqlalchemy**.**orm** import **relationship**

from **sqlalchemy**.**ext**.**declarative** import **declarative\_base**

Base = **declarative\_base**()

class **DataType**(**Enum**):

    IMAGE = "image"

    VECTOR = "vector"

class **LossType**(**Enum**):

    CROSS\_ENTROPY = "CrossEntropyLoss"

    MSE = "MSELoss"

    L1 = "L1Loss"

    SMOOTH\_L1 = "SmoothL1Loss"

class **User**(**Base**):

    \_\_tablename\_\_ = "users"

**id** = **Column**(**Integer**, primary\_key=True, index=True)

    name = **Column**(**String**(50), nullable=False)

    surname = **Column**(**String**(50), nullable=False)

    username = **Column**(**String**(50), unique=True, nullable=False)

    password\_hash = **Column**(**String**(255), nullable=False)

    born\_date = **Column**(**Date**, nullable=False)

    bio = **Column**(**String**(100), default="")

    registration\_date = **Column**(**DateTime**(timezone=True), server\_default=func.now())

    projects = **relationship**("Project", back\_populates="owner")

class **Project**(**Base**):

    \_\_tablename\_\_ = "projects"

**id** = **Column**(**Integer**, primary\_key=True, index=True)

    name = **Column**(**String**(100), nullable=False)

    description = **Column**(**String**(255), default="")

    created\_at = **Column**(**DateTime**(timezone=True), server\_default=func.now())

    owner\_username = **Column**(**String**(50), **ForeignKey**("users.username"), nullable=False)

    input\_type = **Column**(**EnumType**(**DataType**), nullable=False)

    output\_type = **Column**(**EnumType**(**DataType**), nullable=False)

    architecture\_json = **Column**(**Text**(), nullable=True)

    dataset\_id = **Column**(**Integer**, **ForeignKey**("datasets.id"), nullable=True)

    dataset\_preprocess\_json = **Column**(**Text**(), nullable=True)

    optimizer\_json = **Column**(**Text**(), nullable=True)

    scheduler\_json = **Column**(**Text**(), nullable=True)

    loss\_function = **Column**(**EnumType**(**LossType**), nullable=True, default=**LossType**.MSE)

    dataset = **relationship**("Dataset")

    owner = **relationship**("User", back\_populates="projects")

class **Dataset**(**Base**):

    \_\_tablename\_\_ = "datasets"

**id** = **Column**(**Integer**, primary\_key=True, index=True)

    name = **Column**(**String**(100), nullable=False)

    description = **Column**(**String**(255), default="")

    created\_at = **Column**(**DateTime**(timezone=True), server\_default=func.now())

    storage\_id = **Column**(**String**(128), unique=True, nullable=False)

    dataset\_type = **Column**(**String**(50), nullable=False, default="unknown")

    owner\_id = **Column**(**Integer**, **ForeignKey**("users.id"), nullable=False)

Kód - Struktura databáze

Pro práci s databází je využit objektově-relační mapovací přístup (ORM), který umožňuje reprezentovat databázové tabulky jako objektové třídy v programovacím jazyce Python. Tento přístup zjednodušuje práci s databází, zvyšuje čitelnost kódu a snižuje riziko chyb při ručním psaní SQL dotazů.

Třídy DataType a LossType představují výčtové typy (Enum), které slouží k omezení možných hodnot některých atributů projektu.

Typ DataType určuje formát vstupních a výstupních dat neuronové sítě (např. obrazová nebo vektorová data). Typ LossType definuje typ ztrátové funkce použité během procesu učení neuronové sítě. Použití výčtových typů zajišťuje konzistenci dat a zabraňuje ukládání neplatných hodnot do databáze.

Entita User reprezentuje registrovaného uživatele systému. Obsahuje základní identifikační údaje, přihlašovací informace a doplňující metadata, jako je datum registrace nebo krátký popis profilu.

Každý uživatel může vlastnit více projektů, což je realizováno pomocí relační vazby typu one-to-many mezi entitami User a Project.

Entita Project představuje základní pracovní jednotku aplikace. Každý projekt odpovídá jedné konfiguraci neuronové sítě a jejímu procesu učení.

Projekt obsahuje informace o typu vstupních a výstupních dat, architektuře sítě, použitých optimalizačních metodách a ztrátové funkci. Konfigurační části jsou ukládány ve formátu JSON, což umožňuje flexibilní ukládání různých architektur bez nutnosti měnit databázové schéma.

Entita Dataset slouží k evidenci trénovacích dat používaných v projektech. Obsahuje základní popis datasetu a identifikátor fyzického úložiště, kde jsou data uložena.

Dataset je přiřazen konkrétnímu uživateli a může být opakovaně využíván v různých projektech, což podporuje efektivní práci s daty a jejich znovupoužitelnost.

Navržený databázový model reflektuje modulární architekturu celé aplikace a umožňuje její další rozšiřování bez zásadních zásahů do stávající struktury.

### Učení neuronových sítí

Aplikace poskytuje modul pro řízení procesu učení neuronových sítí. Hlavní logiku trénování zajišťují dvě metody: go\_epochs a go\_steps. Obě metody umožňují iterativní optimalizaci váh modelu s využitím gradientního sestupu a umožňují monitorovat průběh trénování v reálném čase.

    def **go\_epochs**(self, data\_loader, project\_id, epochs=1.0, device="cpu"):

        self.model.**to**(device)

        self.model.**train**()

        steps\_per\_epoch = **len**(data\_loader)

        total\_steps = **int**(epochs \* steps\_per\_epoch)

        loader = **infinite\_loader**(data\_loader)

        for step in **range**(total\_steps):

            inputs, targets = **next**(loader)

            inputs, targets = inputs.to(device), targets.to(device)

            self.optimizer.**zero\_grad**()

            outputs = self.model(inputs)

            loss = self.criterion(outputs, targets)

            loss.backward()

            self.optimizer.**step**()

            self.losses.**append**(loss.item())

**print**(f"Step [{step+1}/{total\_steps}] Loss: {loss.item():.4f}")

            if self.scheduler and not **isinstance**(self.scheduler, **ReduceLROnPlateau**):

                self.scheduler.**step**()

            TRAINING\_PROGRESS[project\_id] = {

                "status": "running",

                "current": step / steps\_per\_epoch,

                "total": epochs,

                "loss": loss.item()

            }

        if self.scheduler and **isinstance**(self.scheduler, **ReduceLROnPlateau**):

            self.scheduler.**step**(loss)

        TRAINING\_PROGRESS[project\_id] = {

            "status": "finished",

            "current": total\_steps / steps\_per\_epoch,

            "total": epochs,

            "loss": loss.item()

        }

    def **go\_steps**(self, data\_loader, project\_id, steps=1000, device="cpu"):

        self.model.**to**(device)

        self.model.**train**()

        loader = **infinite\_loader**(data\_loader)

        for step in **range**(steps):

            inputs, targets = **next**(loader)

            inputs, targets = inputs.to(device), targets.to(device)

            self.optimizer.**zero\_grad**()

            outputs = self.model(inputs)

            loss = self.criterion(outputs, targets)

            loss.backward()

            self.optimizer.**step**()

            self.losses.**append**(loss.item())

**print**(f"Step [{step+1}/{steps}] Loss: {loss.item():.4f}")

            if self.scheduler and not **isinstance**(self.scheduler, **ReduceLROnPlateau**):

                self.scheduler.**step**()

            TRAINING\_PROGRESS[project\_id] = {

                "status": "running",

                "current": step,

                "total": steps,

                "loss": loss.item()

            }

        if self.scheduler and **isinstance**(self.scheduler, **ReduceLROnPlateau**):

            self.scheduler.**step**(loss)

        TRAINING\_PROGRESS[project\_id] = {

            "status": "finished",

            "current": steps,

            "total": steps,

            "loss": loss.item()

        }

Kód - Trenování neuronové sítí

Metoda go\_epochs slouží k trénování modelu po zadaný počet epoch. Model je nejprve přesunut na vybraný výpočetní device, například CPU nebo GPU, a přepnut do trénovacího režimu. Data jsou načítána z datového loaderu v nekonečné smyčce, aby bylo možné provést libovolný počet epoch bez nutnosti ručního resetování loaderu. Pro každou dávku dat se nejprve vymažou předchozí gradienty, následně se spočítají výstupy modelu, vyhodnotí ztrátová funkce a provede se zpětná propagace gradientů. Parametry modelu jsou poté aktualizovány pomocí optimalizátoru. V průběhu trénování se průběžně ukládají hodnoty ztrátové funkce a stav projektu do globální struktury TRAINING\_PROGRESS, což umožňuje sledování aktuálního postupu trénování. Scheduler učící rychlosti, pokud je použit, je volán buď po každém kroku, nebo po každé epoše, v závislosti na jeho typu. Po dokončení všech epoch je stav projektu aktualizován na „finished“, čímž se signalizuje dokončení trénování.

Metoda go\_steps poskytuje alternativní způsob trénování, kdy je počet kroků explicitně určen, nezávisle na počtu epoch. Princip jejího fungování je obdobný jako u go\_epochs: model je přesunut na požadovaný device a přepnut do trénovacího režimu, data jsou načítána v nekonečné smyčce, a pro každou dávku se provádí výpočet výstupu, vyhodnocení ztráty, zpětná propagace gradientů a aktualizace parametrů. Během trénování je rovněž sledován průběh prostřednictvím TRAINING\_PROGRESS, který zaznamenává aktuální krok, celkový počet kroků a hodnotu ztráty. Scheduler učící rychlosti je volán podle typu, stejně jako u metody go\_epochs. Po dosažení zadaného počtu kroků je stav projektu nastaven na „finished“, což signalizuje ukončení trénování.

### Neuronové vrstvy

V aplikaci je implementován modul pro definici šablon neuronových vrstev, které lze využít při sestavování architektury modelu. Konstantní objekt LAYER\_TEMPLATES obsahuje předdefinované vrstvy, jako jsou lineární vrstvy, konvoluční vrstvy, aktivace typu ReLU, LeakyReLU, Sigmoid či Tanh, normalizační vrstvy typu BatchNorm a LayerNorm, poolingové vrstvy a speciální bloky, například SEBlock, ResidualBlock nebo ConvolutionalAttention. Každá šablona obsahuje výchozí parametry, například počet vstupních a výstupních kanálů, velikost jádra, krok nebo parametr redukce, což umožňuje rychlé vytváření vrstev bez nutnosti manuálního nastavování všech atributů.

Takový přístup zajišťuje modulární a flexibilní architekturu, kdy je možné snadno experimentovat s různými konfiguracemi modelu, přidávat nové vrstvy či měnit jejich parametry bez zásahu do základního kódu. Šablony rovněž zahrnují operace pro změnu rozměrů dat, jako jsou flatten, upsampling nebo pixel shuffle, což podporuje tvorbu komplexních sítí s různými topologiemi. Tento systém umožňuje dynamicky sestavovat modely podle potřeb jednotlivých projektů a experimentů, čímž se zvyšuje efektivita práce s neuronovými sítěmi a minimalizuje riziko chyb při ruční konfiguraci vrstev.

const LAYER\_TEMPLATES = {

  Linear: { in\_features: 1, out\_features: 1 },

  Conv2D: { in\_features: 1, out\_features: 8, kernel\_size: 3, stride: 1, padding: 1 },

  DSConv2D: { in\_features: 1, out\_features: 8, kernel\_size: 3, stride: 1, padding: 1 },

  ReLU: { in\_features: 1, out\_features: 8 },

  LeakyReLU: { in\_features: 1, out\_features: 8, alpha: 0.01 },

  PReLU: { in\_features: 1, out\_features: 8 },

  Sigmoid: { in\_features: 1, out\_features: 8 },

  Tanh: { in\_features: 1, out\_features: 8 },

  Softmax: { in\_features: 1, out\_features: 8 },

  BatchNorm1d: { in\_features: 1, out\_features: 8 },

  BatchNorm2d: { in\_features: 1, out\_features: 8 },

  LayerNorm: { in\_features: 1, out\_features: 8 },

  Conv2DTranspose: { in\_features: 1, out\_features: 8, kernel\_size: 2, stride: 1, padding: 0, scale\_factor: 2 },

  UpscaleBlock: { in\_features: 1, out\_features: 8, scale\_factor: 2, mode: "nearest", kernel\_size: 3, padding: 1 },

  Upsample: { in\_features: 1, out\_features: 8, scale\_factor: 2, mode: "nearest" },

  MaxPool2D: { in\_features: 1, out\_features: 8, kernel\_size: 2, stride: 2 },

  AvgPool2D: { in\_features: 1, out\_features: 8, kernel\_size: 2, stride: 2 },

  ConvolutionalAttention: { in\_features: 1, out\_features: 8, num\_heads: 4},

  SEBlock: { in\_features: 1, out\_features: 8, reduction: 16 },

  ResidualBlock: { in\_features: 1, out\_features: 8, kernel\_size: 3, stride: 1, padding: 1, depth: 2 },

  InstanceNorm1D: { in\_features: 1, out\_features: 8},

  InstanceNorm2D: { in\_features: 1, out\_features: 8},

  GroupNorm: { in\_features: 1, out\_features: 8, num\_groups: 1 },

  Dropout: { in\_features: 1, out\_features: 8, p: 0.5 },

  PixelShuffle: { in\_features: 8, out\_features: 8, upscale\_factor: 1 },

*// Reshapes*

  Flatten: { in\_features: 1, out\_features: 1 },

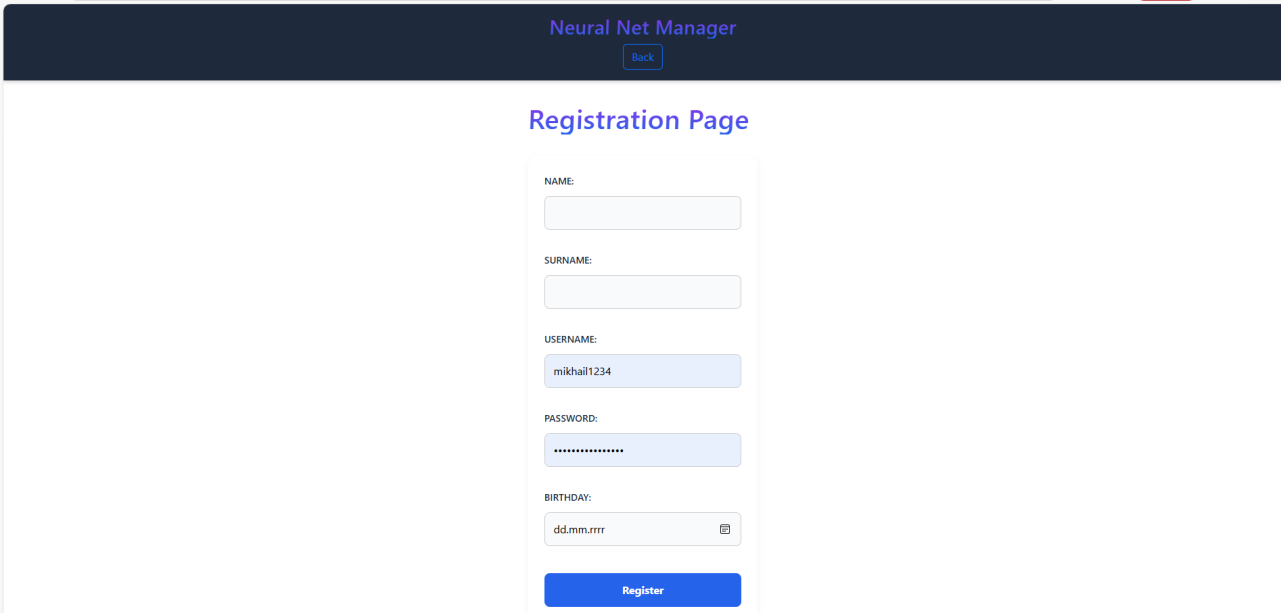
  GlobalAveragePool2D: { in\_features: 1, out\_features: 1 },

};

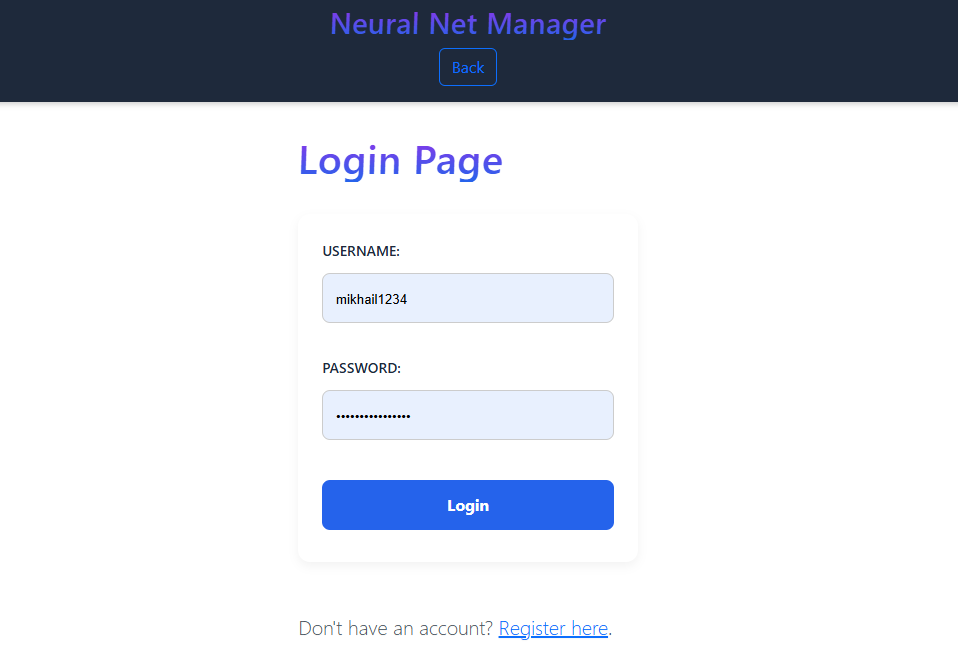
Kód - Definice seznamu vrstev neuronových sítí

## Popis pro uživatele

Pro práci se systémem se uživatel nejprve musí zaregistrovat. Uživatel vyplní své základní údaje, jako je jméno, příjmení, datum narození a unikátní uživatelské jméno. Po zadání hesla a potvrzení registrace systém vytvoří účet a uloží informace do databáze. Po úspěšné registraci uživatel vidí potvrzení o vytvoření účtu a je připraven přihlásit se do aplikace.



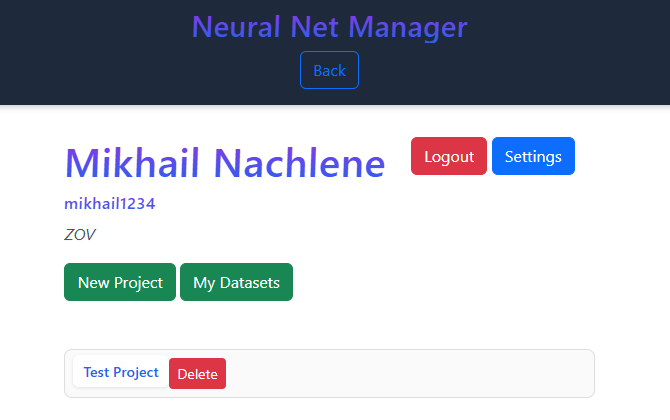
Obrázek - Registrační stránka



Obrázek - Příhlašovácí stránka

Pro přihlášení musí uživatel zadat své uživatelské jméno a heslo. Po ověření údajů systém vygeneruje autentizační token, který umožňuje bezpečný přístup ke všem funkcím aplikace bez opakovaného zadávání přihlašovacích údajů. Uživatel vidí informační zprávu o úspěšném přihlášení a je přesměrován na hlavní stránku, kde může spravovat své projekty, sledovat trénování modelů a pracovat s datovými sadami. V případě nesprávného uživatelského jména nebo hesla systém zobrazí chybovou hlášku a umožní uživateli opakovat přihlášení.

Po úspěšném přihlášení se uživatel dostává na hlavní stránku aplikace, nazvanou „Neural Net Manager“. Uživatel vidí své jméno a uživatelské jméno, což potvrzuje, že je přihlášen do správného účtu. Na hlavní stránce jsou zobrazeny všechny existující projekty uživatele. Stránka poskytuje intuitivní rozhraní pro rychlý přístup k projektům, umožňuje přidávání nových projektů, správu datasetů a sledování průběhu učení neuronových sítí.



Obrázek - Hlavní stránka

Uživatel má k dispozici ovládací prvky pro vytvoření nového projektu, spuštění trénování nebo zobrazení detailů stávajícího projektu. Každý projekt je vizualizován jako samostatná položka, která zobrazuje základní informace a stav procesu učení. Uživatel může vybrat konkrétní projekt a přejít na stránku s detailními nastaveními, kde má možnost upravovat konfiguraci modelu, přidávat datové sady a monitorovat metriky trénování.

Závěr

Sem rozepište zhodnocení cílů ročníkové práce.

POKYNY: V závěru rozeberte, co se vám z výše uvedených cílů povedlo/nepovedlo. NEPOUŽÍVEJTE zde odrážky/body! Většinou se závěr píše tak, že si otevřete Úvod a sepisujete, jak jste naplnili/nenaplnili původní očekávání.

Zároveň sem můžete připsat, co vám projekt dal, kdo vám pomáhal, jestli pro něj máte nějaké plány do budoucna, nebo jestli je/bude někde nasazen, atd.

Rozsahově by měl mít závěr cca **jednu stránku** (jako úvod). NEMÁ obsahovat obrázky ani seznam bodů. MÁ to být jen souvislý text uspořádaný do několika odstavců. Na text odstavců použijte předpřipravený styl s názvem ***Normální text***.

Seznam použitých zdrojů

1. GOOGLE. Google Colab. Online. 2017. Dostupné z: https://colab.research.google.com/. [cit. 2026-01-07].

2. Kaggle. Online. 2010. Dostupné z: https://www.kaggle.com/datasets. [cit. 2026-01-07]. [Online]

3. Hugging Face. Online. 2016. Dostupné z: https://huggingface.co/docs/datasets/index. [cit. 2026-01-07]. [Online]

4. Python. Online. 2024. Dostupné z: https://www.python.org/doc/. [cit. 2026-01-07].

5. Py Torch. Online. 2024. Dostupné z: https://docs.pytorch.org/docs/stable/index.html. [cit. 2026-01-07].

6. Docker. Online. 2024. Dostupné z: https://docs.docker.com/. [cit. 2026-01-07].

7. Alembic. Online. 2024. Dostupné z: https://alembic.sqlalchemy.org/en/latest/. [cit. 2026-01-07].

8. PostgeSQL. Online. 2024. Dostupné z: https://www.postgresql.org/docs/. [cit. 2026-01-07].

9. Redis. Online. 2024. Dostupné z: https://redis.io/docs/latest/. [cit. 2026-01-07].

10. React. Online. 2024. Dostupné z: https://react.dev/learn. [cit. 2026-01-07].

11. Chat GPT. Online. 2024. Dostupné z: https://chatgpt.com/. [cit. 2026-01-07].

12. Py Torch. Online. 2016. Dostupné z: https://docs.pytorch.org/docs/stable/index.html. [cit. 2026-01-07].

Seznam obrázků

[Obrázek 1 - Prostředí Google Colab 8](#_Toc218846830)

[Obrázek 2 - Prostředí Kaggle 9](#_Toc218846831)

[Obrázek 3 - Seznam jednotlivých služeb 12](#_Toc218846832)

[Obrázek 4 - Registrační stránka 19](#_Toc218846833)

[Obrázek 5 - Příhlašovácí stránka 19](#_Toc218846834)

[Obrázek 6 - Hlavní stránka 20](#_Toc218846835)

Seznam kódů

Sem vygenerujte seznam kódů (Reference -> Titulky).

[Kód 1 - Struktura databáze 15](#_Toc218847151)

[Kód 2 - Trenování neuronové sítí 17](#_Toc218847152)

[Kód 3 - Definice seznamu vrstev neuronových sítí 19](#_Toc218847153)

Obsah média

Sem vložte popis přiloženého média (jen pokud je součástí práce).

POKYNY: Popište stručně adresářovou strukturu pro všechny důležité soubory (použijte např. víceúrovňový seznam). Pokud médium (CD, DVD, Flashdisk) obsahuje celý projekt s velkým množstvím knihoven, nevypisujte cesty ke všem souborům. Uveďte pouze kde se nachází projekt, kde se nachází build, apod.

Médium fyzicky označte vaším **jménem, třídou, školním rokem**!Médium připojte k dokumentaci tak, aby nevypadávalo, ale aby se zároveň dalo vyndat a použít.

Médium by mělo obsahovat následující části:

* Projekt
* Export databáze (pokud ji obsahuje)
* Spustitelný build (nebo aspoň odkaz, kde se nachází spustitelná verze)
* Dokumentaci v PDF formátu + vhodný další editovatelný formát (docx, odt)
* Prezentaci připravenou k obhajobě

**Závěrečné poznámky (před odevzdáním vymažte):**

* Dokumentace může obsahovat různá poděkování.
* U jednotlivých odstavců textu dokumentace zkontrolujte zarovnání do bloku.
* Před exportem do PDF aktualizujte všechny vygenerované seznamy (obsah, zdroje, seznam obrázků) a zkontrolujte, že je vše v pořádku.
* Před tiskem si dokumentaci vyexportujte do PDF formátu a zkontrolujte odsazení textu a umístění obrázků.
* Dokumentace může být vytisknutá černobíle.
* Dokumentace musí být vytisknutá jednostranně.
* **V pololetí se dokumentace netiskne!**
* Vytištěná dokumentace by měla být svázána kroužkovou vazbou s průhlednou přední stranou a neprůhlednou zadní stranou (barva zadní strany a vazby je na vás).

**V případě dotazů k dokumentaci kontaktujte vedoucího práce nebo vyučujícího předmětu Projekty!**