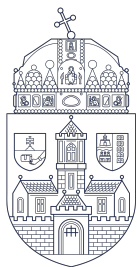


Email kliens fejlesztés - Projektmunka

Fejlesztői dokumentáció

Tóth Balázs - MWZX0D



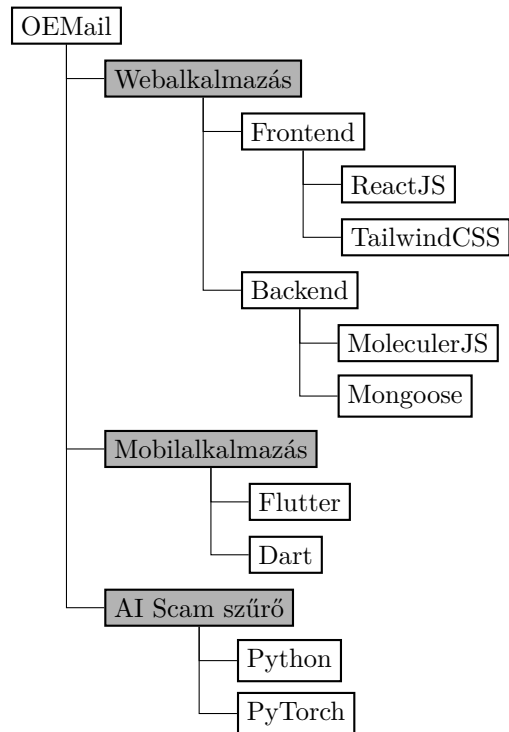
ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Tartalomjegyzék

| | | |
|----------|---|-----------|
| 1 | Projektek | 2 |
| 1.1 | Technológiák | 2 |
| 1.2 | Szoftverek | 2 |
| 1.3 | Előkövetelmények, csomagok, keretrendszerek | 3 |
| 1.3.1 | Webalkalmazás | 3 |
| 1.3.2 | AI Scam szűrő | 3 |
| 2 | Összehasonlítások | 4 |
| 3 | DNS szerver | 5 |
| 3.1 | Indítás | 5 |
| 3.2 | Konfiguráció | 5 |
| 3.2.1 | Zóna hozzáadása | 5 |
| 3.2.2 | Rekordok hozzáadása | 5 |
| 3.2.3 | Tesztelés nslookup parancs segítségével | 5 |
| 4 | SMTP, IMAP szerver | 6 |
| 4.1 | hMailServer | 6 |
| 4.2 | Alapértelmezett mappák | 6 |
| 5 | Backend | 7 |
| 5.1 | Bejelentkezés | 7 |
| 5.2 | Regisztráció | 7 |
| 5.3 | Adott email lekérése mappából | 8 |
| 5.4 | Összes email lekérése mappából | 8 |
| 5.5 | Email törlése mappából | 8 |
| 5.6 | Spam szűrés | 9 |
| 5.7 | Mappa létrehozása | 9 |
| 5.8 | Mappák lekérése | 9 |
| 6 | AI Scam szűrő API | 10 |
| 6.1 | Tanítási adatok felépítése | 10 |
| 6.2 | Modell | 10 |
| 6.3 | Tanítás | 11 |
| 6.4 | Spam szűrése | 14 |

1. Projektek

1.1 Technológiák



1.2 Szoftverek

- **Docker**
 - Compose miatt szükséges a használata
 - <https://docs.docker.com/get-docker/>
- **Postman**
 - API végpontok teszteléséhez
 - <https://www.postman.com/downloads/>
- **MongoDB Compass**
 - Adatbázis menedzsment
 - <https://www.mongodb.com/products/tools/compass>
- **Visual Studio Code** (*ajánlott*)
 - Fejlesztői környezet
 - <https://code.visualstudio.com/download>

1.3 Előkövetelmények, csomagok, keretrendszerek

1.3.1 Webalkalmazás

- NodeJS
 - <https://nodejs.org/en>

1.3.2 AI Scam szűrő

- Python
 - <https://www.python.org/downloads/>
- Pip
 - <https://pip.pypa.io/en/stable/installation/>
- torch
- scikit-learn
- numpy
- pandas
- matplotlib
- flask

2. Összehasonlítások

| Funkció | OEMail | GMail | ProtonMail |
|-------------------------|--------|-------|------------|
| Mobilon működik | ✓ | ✓ | ✓ |
| Dedikált autentikátor | ✓ | ✓ | ✗ |
| Ingyenes spam szűrő API | ✓ | ✗ | ✗ |

Tábla 2.1: Szolgáltatások közti különbségek

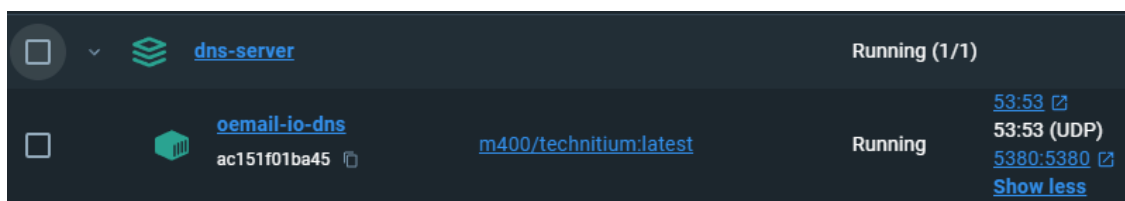
3. DNS szerver

3.1 Indítás

Ahhoz, hogy használni tudjuk az email klienst, kritikus fontosságú a DNS szervernek a futtatása. Természetesen arra vonatkozik ez a megkötés, ha lokálisan futtatjuk az email szervert!

- `docker-compose up -d`

Parancs futtatása után látható, hogy sikeresen elindult a DNS szerver.



3.2 Konfiguráció

Magát a konfigurációt webes környezetben hajthatjuk végre, amely látható is, hogy a **5380** porton fut. Cím, amelyen a konfigurációt elvégezhetjük:

- `http://localhost:5380/`

3.2.1 Zóna hozzáadása

Megadott paraméterek:

- Zóna neve: `oemail.io`
- Típus: Primary Zone

3.2.2 Rekordok hozzáadása

Korábban létrehozott `oemail.io` zónához rekordok hozzáadása szükséges az email szerverhez.

- MX típusú rekord paraméterei:
 - Name: `@` (rámutat az aktuális DNS rekordra)
 - TTL: 3600 (1 óra)
 - Preference: 10
 - Exchange: `mail.oemail.io`
- A típusú rekord paraméterei:
 - Name: `mail`
 - TTL: 3600 (1 óra)
 - IPv4 Address: `127.0.0.1`

3.2.3 Tesztelés nslookup parancs segítségével

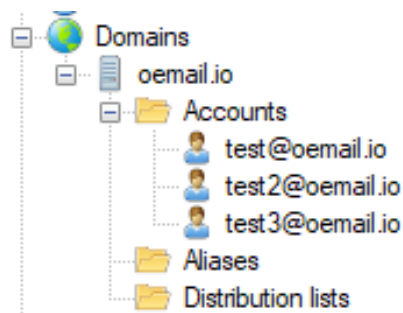
Az alábbi parancs segítségével tesztelhetjük a korábban beállított paramétereket: `nslookup mail.oemail.io`

MX rekord paraméterek ellenőrzése: `nslookup -type=mx oemail.io 127.0.0.1`

4. SMTP, IMAP szerver

4.1 hMailServer

Be kell kapcsolni az SMTP, IMAP protokollokat, ezután meg kell adni a megfelelő domain nevét: `oemail.io`. Itt hozhatunk létre IMAP fiókokat is, amit az **Accounts** menüpontban tehetünk meg.



4.2 Alapértelmezett mappák

Új fiók regisztrációja során az alábbi mappák kerülnek létrehozásra:

- **INBOX:** Bejövő emailek.
- **SENT:** Elküldött emailek.
- **TRASH:** Szemét, törölt emailek.
- **SPAM:** Gyanús emailek.
- **STARS:** Kedvenc emailek.

5. Backend

5.1 Bejelentkezés

- Végpont - `/api/users/login`
 - **POST** (JSON)
- Előfeltétel(ek): **Nincs**
- Bemeneti paraméter(ek):
 - email - Típus: **email**
 - password - Típus: **string**

```
1  {
2      "email": "test2@oemail.io",
3      "password": "test"
4  }
```

Kód 5.1: Példa adatok a bejelentkezéshez

5.2 Regisztráció

- Végpont - `/api/users/`
 - **POST** (JSON)
- Előfeltétel(ek): **Nincs**
- Bemeneti paraméter(ek):
 - user - Típus: **object**

```
1  {
2      "user": {
3          "email": "test2@oemail.io",
4          "password": "test",
5          "firstName": "Test2",
6          "lastName": "Test2"
7      }
8  }
```

Kód 5.2: Példa adatok a regisztrációhoz

5.3 Adott email lekérése mappából

- Végpont - `/api/mail/getEmailByMailBox`
 - **POST** (JSON)
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - `mailBoxName` - Típus: **string**

```
1  {  
2    "mailBoxName": "INBOX"  
3  }
```

Kód 5.3: Példa adatok a legújabb email lekérésére adott mappából

5.4 Összes email lekérése mappából

- Végpont - `/api/mail/getAllEmailByMailBox`
 - **POST** (JSON)
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - `mailBoxName` - Típus: **string**

```
1  {  
2    "mailBoxName": "INBOX"  
3  }
```

Kód 5.4: Példa adatok az összes email lekérésére adott mappából

5.5 Email törlése mappából

- Végpont - `/api/mail/deleteMessage`
 - **DELETE** (JSON)
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - `mailBoxName` - Típus: **string**

```
1  {  
2    "mailBoxName": "INBOX"  
3  }
```

Kód 5.5: Példa adatok adott email törlésére mappából

5.6 Spam szűrés

- Végpont - `/api/mail/filterInboxFromSpam`
 - **POST** (JSON)
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - `mailBoxName` - Típus: **string**

```
1  {  
2      "mailBoxName": "INBOX"  
3  }
```

Kód 5.6: Példa adatok spam emailek szűrésére

5.7 Mappa létrehozása

- Végpont - `/api/mail/createMailBox`
 - **POST** (JSON)
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - `mailBoxName` - Típus: **string**

```
1  {  
2      "mailBoxName": "INBOX"  
3  }
```

Kód 5.7: Példa adatok mappa létrehozására

5.8 Mappák lekérése

- Végpont - `/api/mail/listMailboxes`
 - **GET**
- Előfeltétel(ek): **Felhasználói bejelentkezés (token)**
- Bemeneti paraméter(ek):
 - Nincs

6. AI Scam szűrő API

6.1 Tanítási adatok felépítése

Maga a bemeneti fájl egy `csv` kiterjesztésű fájl, mely tartalmaz `Text` és `Class` besorolásokat. A `Text` besorolás jelzi az email üzenetet. A `Class` besorolás pedig két értékkel bírhat: 0 és 1. A 0 érték jelzi, hogy az adott email üzenet nem veszélyes, csak egy általános email üzenet. 1-es érték jelzi a scam email üzenetet.

Fontossága a tanításnak, hogy maga az API képes legyen meghatározni, hogy majd a bejövő email üzenet káros-e vagy sem, ezáltal védi meg a felhasználót, hogy ne kattintson potenciális veszélyes email-re.

6.2 Modell

Konstruktoron belül, három teljesen összekapcsolt (lineáris) réteget határozzunk meg. Ezek a rétegek felelősek a bemeneti adatok átalakításáért a megtanult súlyok segítségével:

- `self.fc1` → Az `input_dim` dimenziójú bemenetet fogadja és 64 dimenziós kimenetet állít elő.
- `self.fc2` → Az `fc1` kimenetét veszi (64 dimenzió) és egy 32 dimenziós kimenetet állít elő.
- `self.fc3` → Az `fc2` kimenetét veszi (32 dimenzió) és létrehozza a 2 dimenziós végső kimenetet, amely két osztályt (0 és 1) képvisel.

```
1  import torch
2  import torch.nn as nn
3
4  class TextClassifier(nn.Module):
5      def __init__(self, input_dim):
6          super(TextClassifier, self).__init__()
7          self.fc1 = nn.Linear(input_dim, 64)
8          self.fc2 = nn.Linear(64, 32)
9          self.fc3 = nn.Linear(32, 2)
10
11     def forward(self, x):
12         x = torch.relu(self.fc1(x))
13         x = torch.relu(self.fc2(x))
14         x = self.fc3(x)
15         return x
```

Kód 6.1: Szövegosztályozó modell

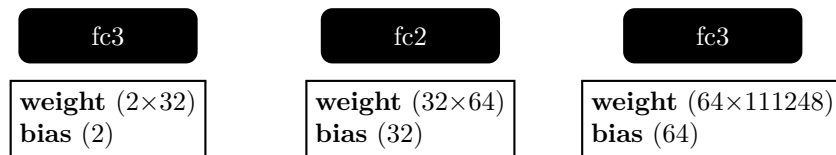
A `forward` metódus meghatározza a neurális hálózat haladási irányát. Vár egy `x` tenzor paramétert, és minden egyes definiált rétegnek átadja azt a **rectified linear unit** (ReLU) aktiválás függvény (`torch.relu`) segítségével. Ezután a végső eredményt adja vissza, ami a hálózat kimenete.

```

1  def forward(self, x):
2      x = torch.relu(self.fc1(x))
3      x = torch.relu(self.fc2(x))
4      x = self.fc3(x)
5      return x

```

Kód 6.2: Forward metódus



Ábra 6.1: Modell súlyok

6.3 Tanítás

Először is beolvasásra kerülnek a tanítási adatok, melyek 'Text' és 'Class' értékekből állnak, így ezeket el kell szeparálni egymástól.

```

1  data = pd.read_csv('fraud_email_.csv')
2
3  data['Text'].fillna('', inplace=True)
4
5  texts = data['Text'].tolist()
6  labels = data['Class'].tolist()

```

Kód 6.3: Tanítási adatok betöltése csv fájlból

Létre kell hozni egy `CountVectorizer()` példányt, mely szöveges dokumentumok gyűjteményét alakítja át mátrixba, ahol minden sor egy dokumentumot és minden oszlop egy egyedi szót a korpuszban reprezentál. Majd a `fit_transform` metódus segítségével illesztjük a `vectorizer`-t a megadott szöveges adathoz (`texts`) és átalakítjuk a szöveges adatot egy mátrixba (`X`).

Hogy később fel lehessen használni ezen adatokat a későbbiekben, szükséges menteni őket: `count_vectorizer_vocab.pkl`; `count_vectorizer.pkl` néven. A `count_vectorizer_vocab.pkl` egy szótár, amely a szavakat indexekkel társítja a mátrixban. A `count_vectorizer.pkl` tartalmazza a szótárt és a hozzátartozó szükséges egyéb paramétereket/konfigurációkat.

```
1     vectorizer = CountVectorizer()
2     X = vectorizer.fit_transform(texts)
3
4     with open('count_vectorizer_vocab.pkl', 'wb') as vocab_file:
5         pickle.dump(vectorizer.vocabulary_, vocab_file)
6
7     with open('count_vectorizer.pkl', 'wb') as vectorizer_file:
8         pickle.dump(vectorizer, vectorizer_file)
```

Kód 6.4: Beolvasott adatok előfeldolgozása

Adatok konverválása PyTorch tenzorokká, ahol először az `X` változó kerül átalakításra a `torch.tensor` segítségével. Így eredményül kapunk egy sűrűbb tenzort. Ezt követően a `dtype=torch.float32` beállításával biztosítjuk, hogy a tenzor elemei lebegőpontos számok legyenek. Az `y` változót is PyTorch tenzorá alakítjuk, de `int64` típust használunk, mert ezek a címkék osztályokat jelentenek.

Adatok felosztása tanító és teszt halmazokra a `train_test_split` függvény segítségével. Az `X_train` és a `y_train` változók tartalmazzák a tanító adathalmazt, míg az `X_test` és `y_test` változók a teszt adathalmazt. A `test_size=0.2` azt jelenti, hogy a teszt halmaz a teljes adathalmaz 20%-át fogja tartalmazni és a `random_state=42` a véletlenszerűség vezérlésére szolgál, így az eredmények ismételhettek lesznek.

Bemeneti dimenzió és modell létrehozása, ahol az `input_dim` változóban eltároljuk a bemeneti dimenziót, ami a tanító adathalmazban lévő jellemzők számát jelenti. Ezután létrehozunk a már korábban létrehozott modellt (`TextClassifier`) és átadjuk paraméterként a bemeneti dimenziót.

```
1     X = torch.tensor(X.toarray(), dtype=torch.float32)
2     y = torch.tensor(labels, dtype=torch.int64)
3
4     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
5
6     input_dim = X_train.shape[1]
7     model = TextClassifier(input_dim)
```

Kód 6.5: PyTorch tenzorok

Először létrehozunk egy `nn.CrossEntropyLoss` példányt, ami felel a veszteségekért. Azért került használatra, mert jól alkalmazható osztályozási problémákra, ahol az egyes példányok egyetlen helyes kategóriákhoz tartoznak.

Az optimalizáló algoritmust az `optim.Adam` osztály segítségével hozzuk létre. Az Adam egy hatékony optimalizáló algoritmus, amely alkalmazodik a tanítás során változó `gradiens`hez ¹. A `model.parameters()` az összes tanítható paramétert jelöli meg a modellben és ezeket fogja optimalizálni az algoritmus. Az `lr=0.001` paraméter beállítja a tanulási ráta értékét, ami azt szabályozza, hogy mennyire legyenek nagyok a lépésközök a paraméterek frissítésekor.

```
1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Kód 6.6: Veszteség és optimalizáló algoritmus definiálása

Maga a tanulási fázis egy cikluson keresztül megy végig (`num_epochs`) és minden epoch kezdetén a gradiensok nullázódnak, hogy ne akkumulálódjának az epochok között (`optimizer.zero_grad()`). A modell előrejelzéseket hoz létre a tanító adathalmazon (`outputs = model(X_train)`). A `loss = criterion(outputs, y_train)` kiszámolja a veszteséget a modell előrejelzések és a valós címkék között a meghatározott veszteségfüggvény segítségével.

A `loss.backward()` visszaterjeszti (backpropagation) a gradiensket a hálózaton, azaz kiszámolja a veszteségfüggvény parciális deriváltját a paraméterek szerint. Az `optimizer.step()` optimalizációs algoritmus végrehajtja a paraméterek frissítését a tanító adathalmazon kiszámolt gradiensok alapján. Végül a veszteség értékét hozzáadjuk a `train_losses` listához és elmentjük a tanítási eredményeket a `spam_classifier_model.pth` néven.

```
1 num_epochs = 20
2 train_losses = []
3
4 for epoch in range(num_epochs):
5     optimizer.zero_grad()
6     outputs = model(X_train)
7     loss = criterion(outputs, y_train)
8     loss.backward()
9     optimizer.step()
10    train_losses.append(loss.item())
11    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
12
13 torch.save(model.state_dict(), "spam_classifier_model.pth");
```

Kód 6.7: Tanítási ciklus

¹A gradiens egy vektor, amely tartalmazza egy függvény minden paraméterének parciális deriváltját egy adott pontban.

A `torch.max` függvény segítségével megkeressük azokat a kimeneti oszlopokat, amelyekben a legnagyobb értékek vannak. A második visszatérési érték a maximális érték indexeit tartalmazza. Ezek az indexek az osztályokhoz tartozó előrejelzések.

A pontosság kiszámításánál az `accuracy_score` felel. Tehát összehasonlítjuk a modell által előrejelzett osztályokat (`predicted`) a valós osztályokkal (`y_test`), majd kiszámoljuk a pontosságot.

```
1     model.eval()
2     with torch.no_grad():
3         y_pred = model(X_test)
4         _, predicted = torch.max(y_pred, 1)
5
6     accuracy = accuracy_score(y_test.numpy(), predicted.numpy())
7     print(f'Accuracy: {accuracy * 100:.2f}%')
```

Kód 6.8: Modell kiértékelése

6.4 Spam szűrése

- Végpont - `/spam-classification`
 - **POST** (JSON)
- Előfeltétel(ek): **Nincs**
- Bemeneti paraméter(ek):
 - text - Típus: **string**

```
1     {
2         "text": "SAMPLE TEXT"
3     }
```

Kód 6.9: Példa adatok a spam üzenet szűrésére

Kódjegyzék

| | | |
|-----|---|----|
| 5.1 | Példa adatok a bejelentkezéshez | 7 |
| 5.2 | Példa adatok a regisztrációhoz | 7 |
| 5.3 | Példa adatok a legújabb email lekérésére adott mappából | 8 |
| 5.4 | Példa adatok az összes email lekérésére adott mappából | 8 |
| 5.5 | Példa adatok adott email törlésére mappából | 8 |
| 5.6 | Példa adatok spam emailek szűrésére | 9 |
| 5.7 | Példa adatok mappa létrehozására | 9 |
| 6.1 | Szövegosztályozó modell | 10 |
| 6.2 | Forward metódus | 11 |
| 6.3 | Tanítási adatok betöltése csv fájlból | 11 |
| 6.4 | Beolvasott adatok előfeldolgozása | 12 |
| 6.5 | PyTorch tenzorok | 12 |
| 6.6 | Vesztéség és optimalizáló algoritmus definiálása | 13 |
| 6.7 | Tanítási ciklus | 13 |
| 6.8 | Modell kiértékelése | 14 |
| 6.9 | Példa adatok a spam üzenet szűrésére | 14 |

Ábrajegyzék

| | | |
|-----|-------------------------|----|
| 6.1 | Modell súlyok | 11 |
|-----|-------------------------|----|

Táblajegyzék

| | |
|--|---|
| 2.1 Szolgáltatások közti különbségek | 4 |
|--|---|