



**Politechnika
Śląska**

Dokumentacja projektowa

2023/2024

Praktyczne wykorzystanie REST API

Kierunek: Informatyka

Członkowie zespołu:

Aleksander Kulpa

Mikołaj Macura

Paweł Habrzyk

Gliwice, 2023/2024

Spis treści

1	Wprowadzenie	2
1.1	Cel projektu	2
1.2	Zespół projektowy	2
1.3	Instrukcja obsługi	2
1.4	Instrukcja wdrożenia	9
2	Założenia projektowe	10
2.1	Opis działania	10
2.2	Algorytmy	10
2.3	Bazy danych	11
2.3.1	Opis bazy i tabel	11
2.3.2	Operacje na bazie danych	11
3	Implementacja	11
3.0.1	app.py	11
3.0.2	db_manager.py	11
3.0.3	Elementy pomocnicze	13
3.1	Testy	13
3.2	test_train_model_endpoint	13
3.3	get_login	14
3.4	test_predict_diabetes_endpoint	14
3.5	test_get_data_endpoint	14
3.6	test_signup_endpoint	14
3.7	Wykonanie Testów	14

1 Wprowadzenie

1.1 Cel projektu

Zaimplementuj system wspomagania lekarzy (poprzez użycie sieci neuronowej) poprzez automatyczną analizę danych medycznych. System posiada:

- dwa tryby – uczenie algorytmu/klasyfikacja nowej próbki
- dane medyczne szyfrowane i bezpieczne (o tym w następnej sekcji)
- dane lekarzy odpowiednio zabezpieczone (o tym w następnej sekcji)
- możliwość dodawania nowego pacjenta/lekarza
- wszystkie moduły zostały przetestowane
- sieć neuronowa została przeanalizowana pod względem ilości neuronów/warstw
- W zadaniu wykorzystaliśmy bazę danych: <https://www.kaggle.com/uciml/pima-indians-diabetes-database>

1.2 Zespół projektowy

Podział pracy był następujący:

- Mikołaj Macura (mikomac405) - pisanie backend'u, baza danych i zabezpieczenia
- Paweł Habrzyk (PriestOfAdanos) - sieć neuronowa i jej dostrajanie + pomoc w backendzie
- Aleksander Kulpa (AlexKulpa) - pisanie frontend'u, odpowiedzialność za przesyłanie odpowiednich danych do backend'u

1.3 Instrukcja obsługi

(Aby uruchomić aplikację potrzebny jest Docker: <https://docs.docker.com/get-docker/>)

Ściągamy kod źródłowy i wypakowujemy w dowolny miejscu na dysku. Jeżeli Docker jest zainstalowany uruchamiamy konsolę i z poziomu głównego folderu naszej aplikacji wykonujemy następujące komendy:

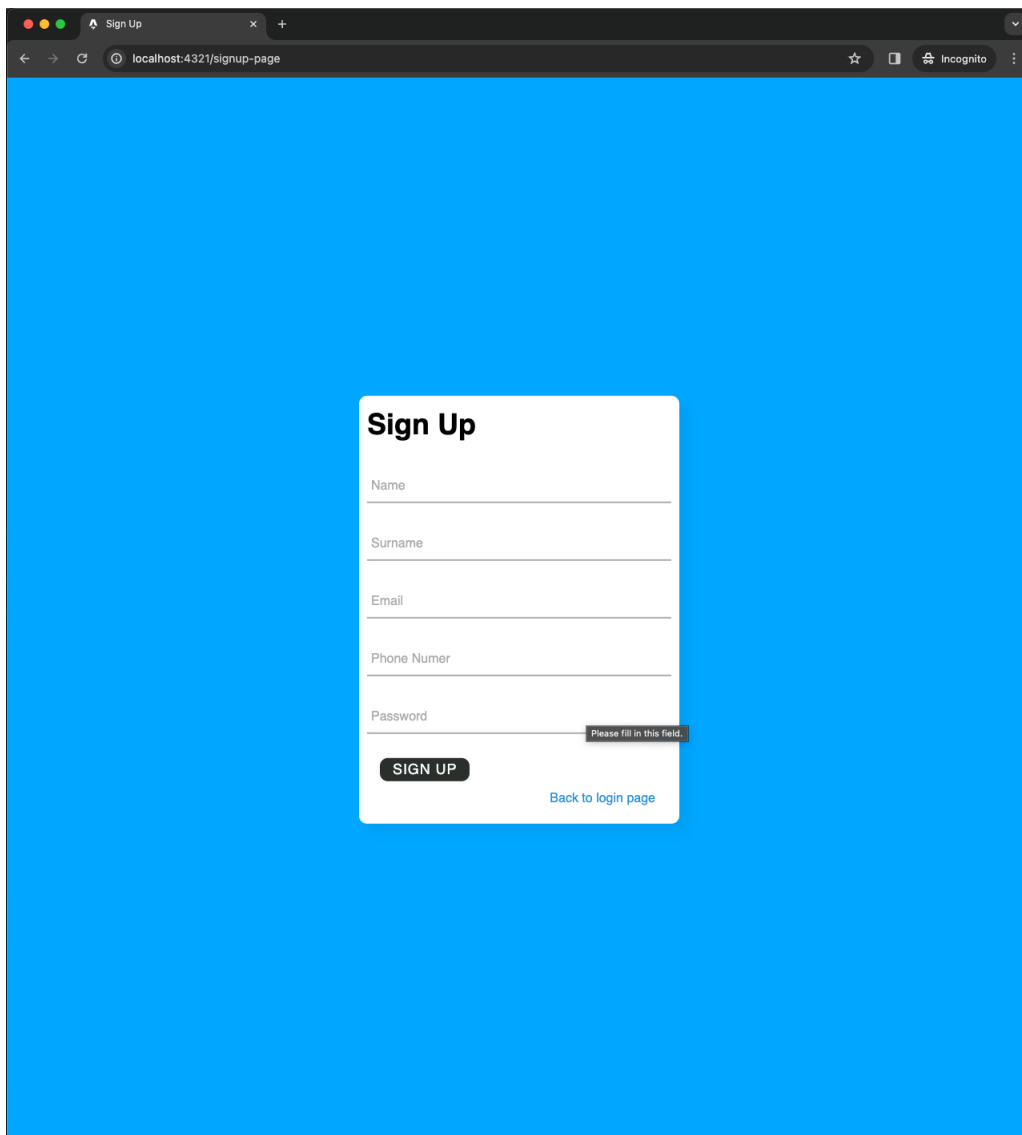
- 'docker compose build' - ściąga wszystkie pakiety dla naszych programów
- 'docker compose up' - uruchamia nasze aplikacje

Dostęp do interfejsu webowego naszej aplikacji znajduje się pod adresem "localhost:4321" a dostęp do endpointów RestAPI znajduje się pod adresem "localhost:8000/docs"

Aby zalogować się, do obu aplikacji domyślnym kontem administratora jest:
email: senior_registrar@hospital.com

hasło: passwd

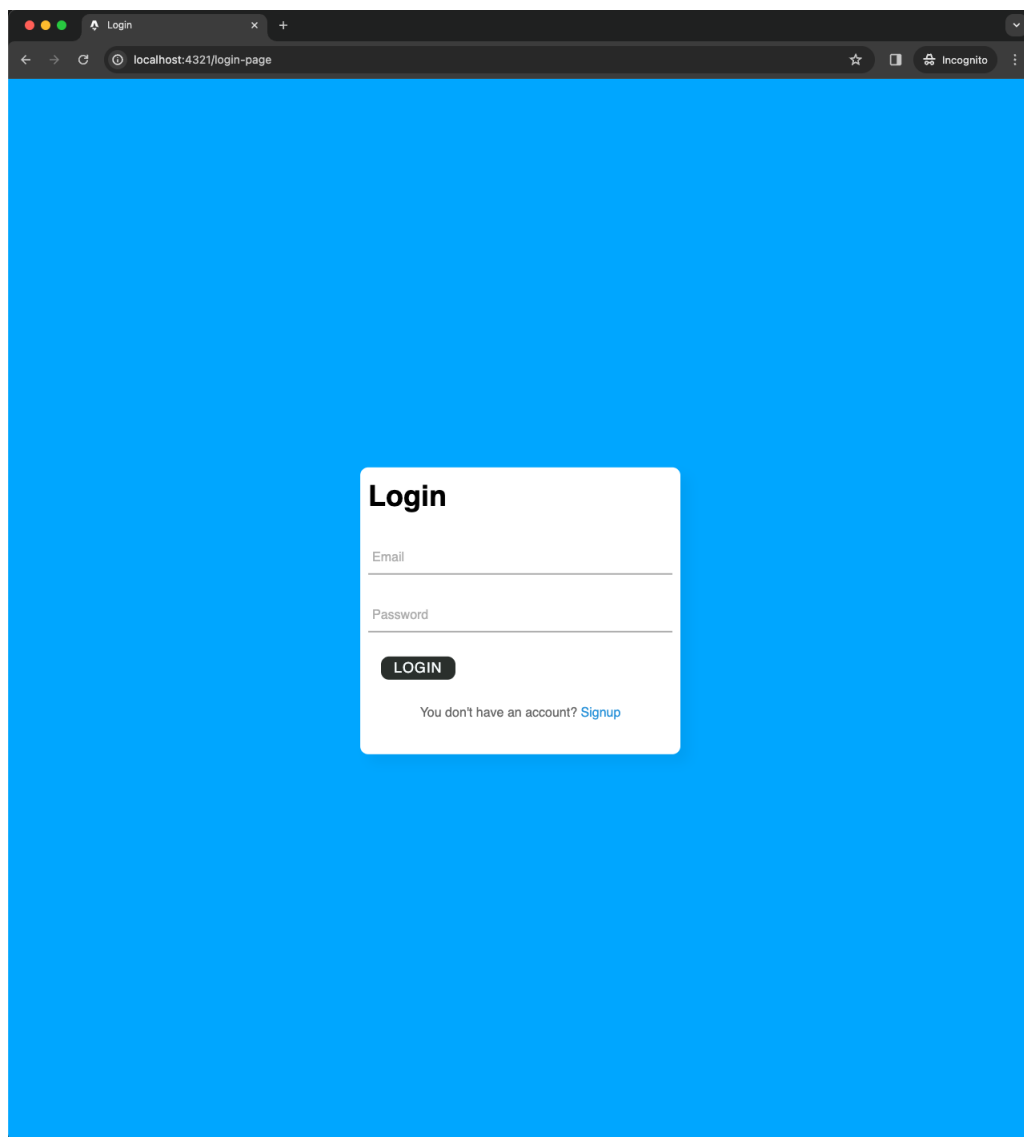
Gdy obie aplikacje działają mamy dostępne możliwości jak poniżej.



The screenshot shows a web browser window with the address bar displaying 'localhost:4321/signup-page'. The page has a solid blue background. In the center, there is a white rectangular form titled 'Sign Up'. The form contains five input fields: 'Name', 'Surname', 'Email', 'Phone Number', and 'Password'. Below these fields is a black button with white text that says 'SIGN UP'. Underneath the button is a blue link that says 'Back to login page'. A small red error message 'Please fill in this field.' is visible next to the 'Password' field.

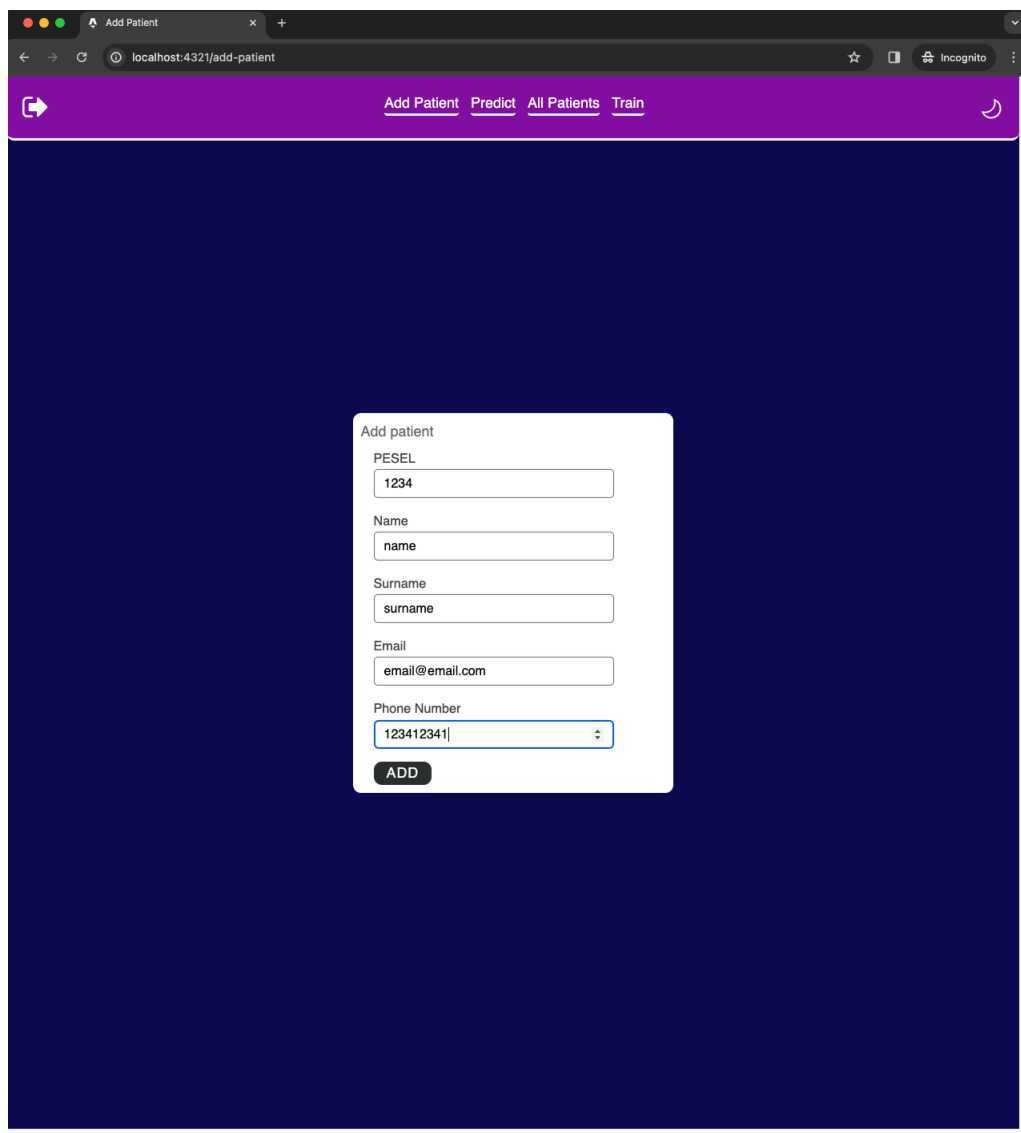
Rysunek 1: Możemy dodać lekarza

Pola Name, Surname są zabezpieczone przed wpisaniem znaków poza literami, Phone Number wymaga jedynie cyfr, których w sumie jest dziewięć. Pole email wymaga wpisania poprawnego emaila. Nie da się utworzyć konta bez któregoś z pól uzupełnionego.



Rysunek 2: Strona logowania się lekarzy posiadających konta stworzone w rysunku powyżej

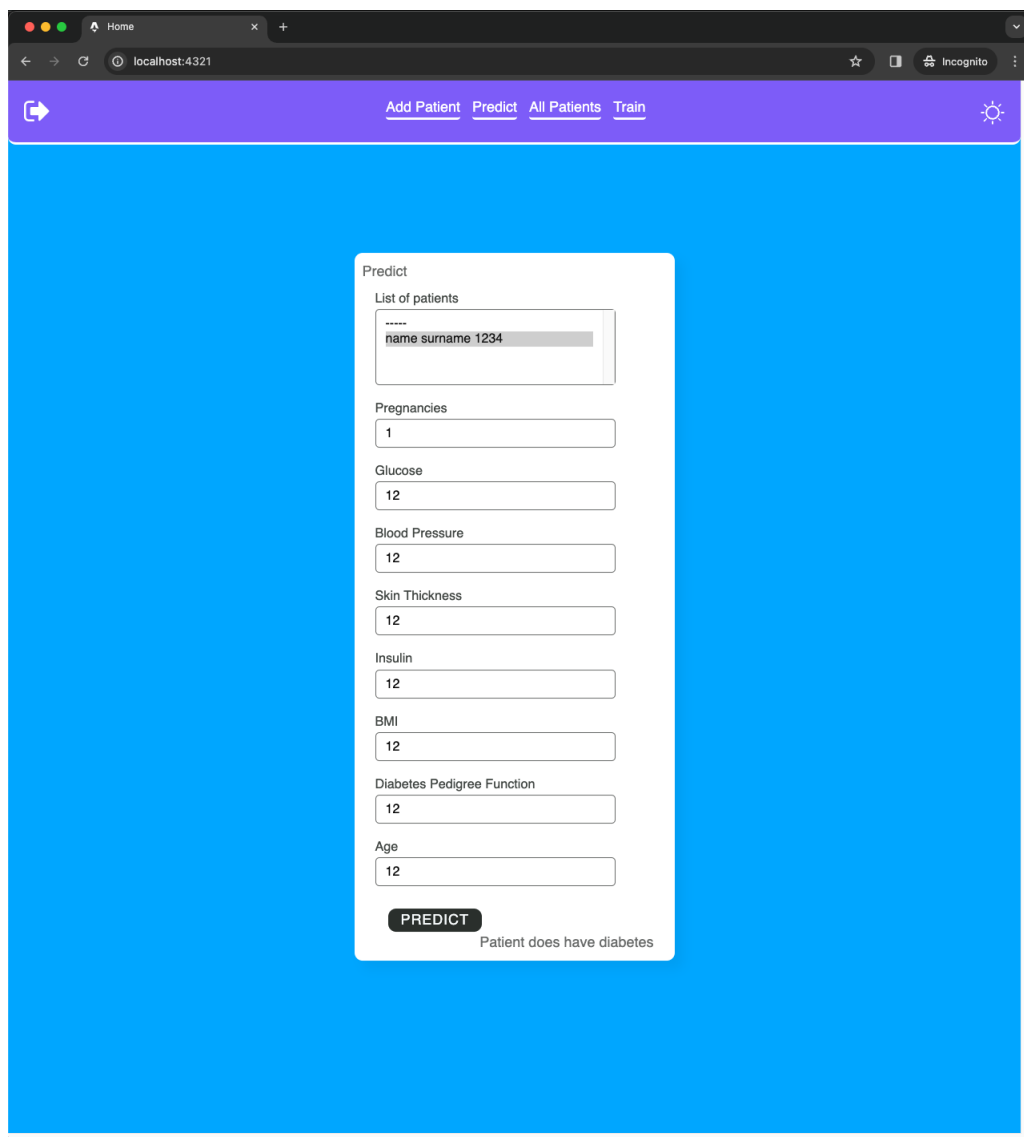
Pole email wymaga wpisania poprawnego emaila. Nie da się utworzyć konta bez któregoś z pól uzupełnionego.



Rysunek 3: Lekarz może dodać pacjenta, który poda mu swoje dane

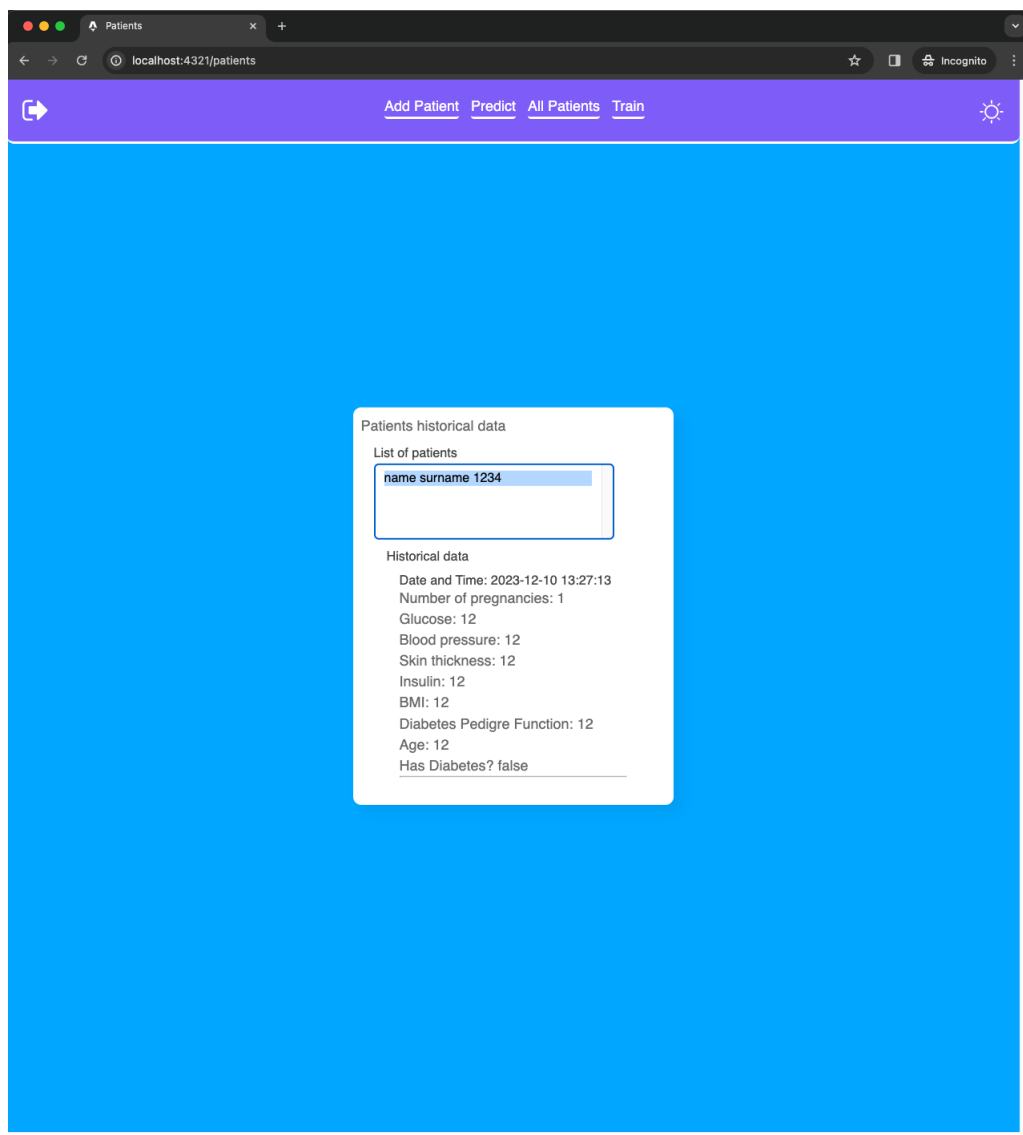
Pole PESEL wymaga jedynie cyfr, minimum jednej. Pola Name, Surname są zabezpieczone przed wpisaniem znaków poza literami, Phone Number wymaga jedynie cyfr, których w sumie jest dziewięć. Pole email wymaga wpisania poprawnego emaila. Nie da się utworzyć pacjenta bez któregoś z pól uzupełnionego.

Również można zauważyć, iż zmieniło się tło na tryb nocny, w prawym górnym rogu steruje nim przycisk księżyca/słoneczka.



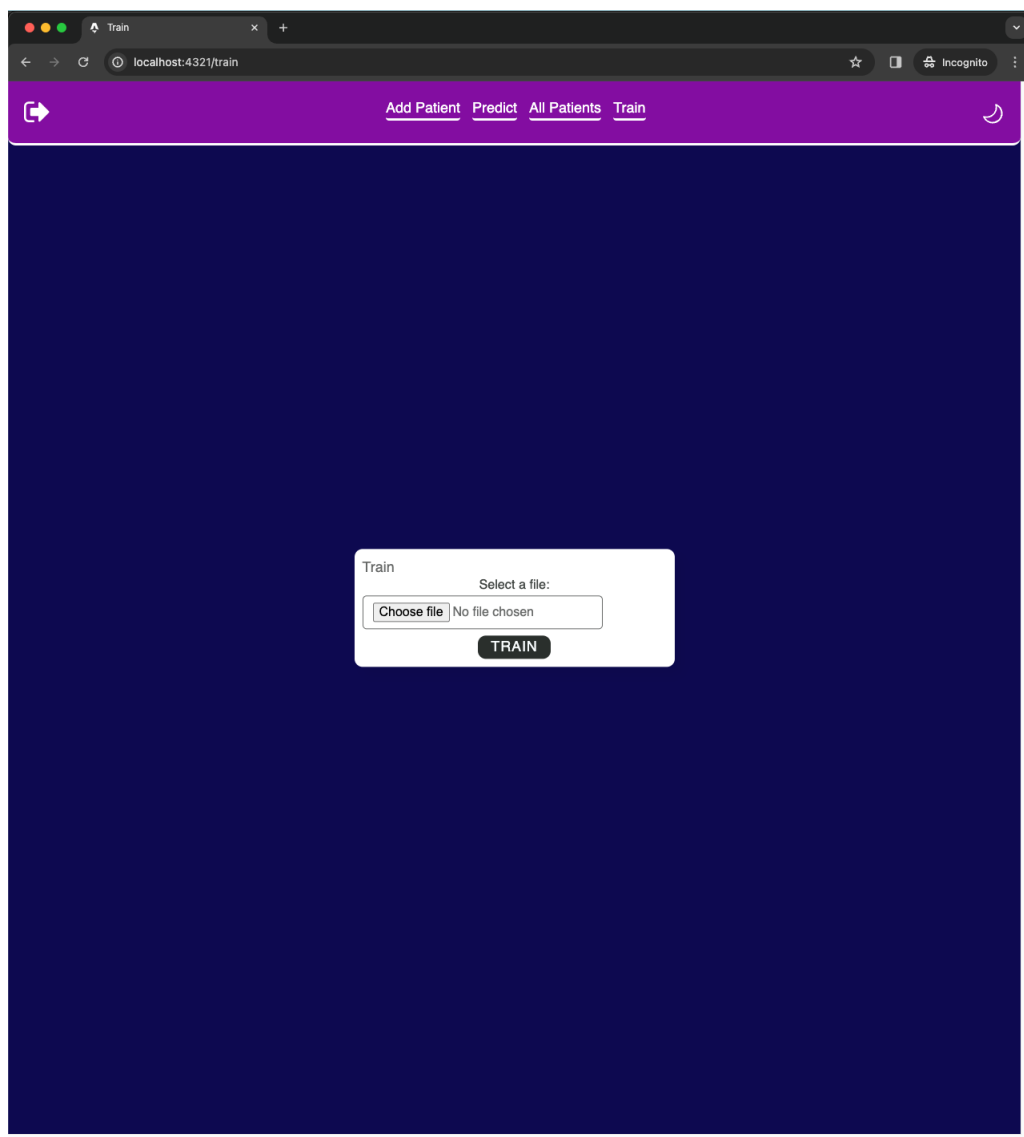
Rysunek 4: Wybieramy pacjenta bądź pole puste, aby przetestować bez zapisywania danych do pacjenta

Nie ma możliwości nie wybrania pola z listy, wszystkie pola wymagają cyfr, nie ma możliwości wpisania innych znaków.



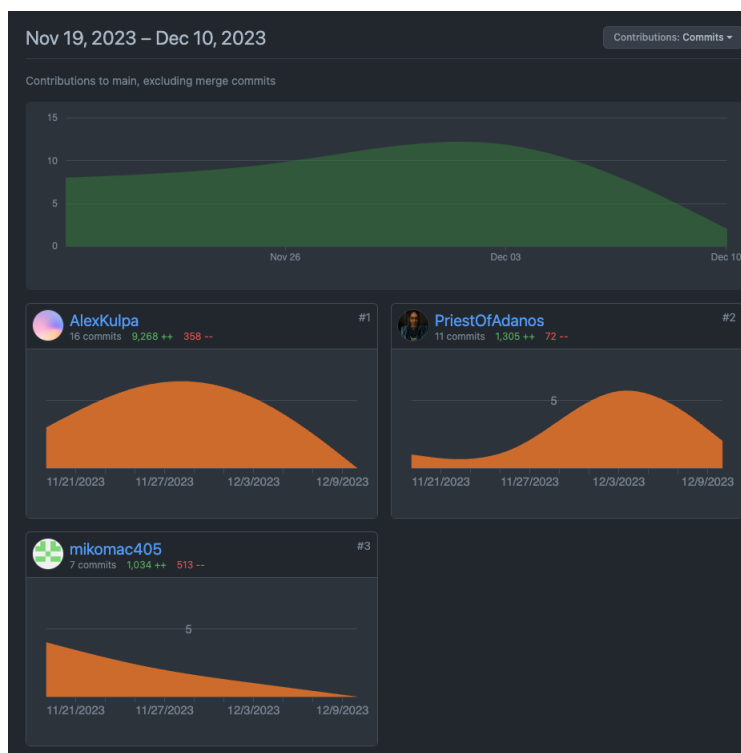
Rysunek 5: Wybierając z listy pacjenta możemy sprawdzić jego historyczne dane predykcji

Dane z predykcji są widoczne w postaci listy posortowanej chronologicznie.



Rysunek 6: Dodajemy plik csv z danymi do treningu

1.4 Instrukcja wdrożenia



Rysunek 7: Wybierając z listy pacjenta możemy sprawdzić jego historyczne dane predykcji

2 Założenia projektowe

2.1 Opis działania

W tym skrypcie Pythona tworzona jest sieć neuronowa przy użyciu biblioteki **keras**. Sieć ta jest prosta, sekwencyjna, skonstruowana z kilku warstw.

Główna architektura sieci wygląda następująco:

1. Warstwa wejściowa: Warstwa **Dense** (gęstej) z 64 komórkami (neuronami) i funkcją aktywacji typu 'linear'. Ta warstwa przyjmuje dane wejściowe o określonym rozmiarze (rozmiarze cech wejściowych).
2. Druga warstwa: Warstwa robocza - **Dense** z 32 komórkami (neuronami) i funkcją aktywacji 'linear'.
3. Trzecia warstwa: Kolejna warstwa robocza - **Dense** z 16 komórkami (neuronami) i również z funkcją aktywacji 'linear'.
4. Warstwa wyjściowa: Końcowa warstwa - **Dense** z jednym neuronem i liniową funkcją aktywacji. Ta warstwa zwraca końcowy wynik prognozy.

Model sieci neuronowej jest trenowany przy użyciu optymalizatora 'adam' i funkcji straty jest 'mean_squared_error' (średni kwadrat błędu).

Natomiast w procesie tuningu hiperparametrów, stosowany jest algorytm **Random Search**. Wyszukuje on losowo kombinacje hiperparametrów, ocenia model dla każdej kombinacji i wybiera tę, która daje najmniejszą loss (stratę).

Podczas predykcji, model jest wczytywany z pliku, a następnie używany do prognozowania wartości wyjściowej na podstawie danych wejściowych. Prognoza jest następnie konwertowana do **int** i zwracana.

Wzory matematyczne:

- Adam: <https://arxiv.org/abs/1412.6980v8>
- Mean Squared Error:

$$MSE = \frac{1}{n} \sum (actual - prediction)^2$$

2.2 Algorytmy

Reprezentacja matematyczna modelu

$$y = W_4 \cdot (W_3 \cdot (W_2 \cdot (W_1 \cdot x + b_1) + b_2) + b_3) + b_4 \quad (1)$$

Biblioteki

- sklearn dla modelu regresji logistycznej
- keras dla modelu sekwencyjnego
- kerastuner do strojenia hiperparametrów.

Modelowanie

- Użycie Sequential z keras do tworzenia modelu sieci neuronowej.
- Konfiguracja warstw i neuronów w modelu (Dense layers).
- Kompilacja modelu z określonymi parametrami (np. optimizer='adam', loss='mean_squared_error').

Przetwarzanie Danych

- Użycie pandas do manipulacji i analizy danych.
- Podział danych na zestawy treningowe i testowe (train_test_split).

2.3 Bazy danych

2.3.1 Opis bazy i tabel

W projekcie użyta została baza danych SQLite Baza danych zawiera 3 tabele:

- diabetes - zawiera dane do szkolenia modelu
- patients - zawiera informacje o pacjencie oraz jego historię predykcji
- doctors - zawiera listę kont wraz z zaszyfrowanymi hasłami algorytmem bcrypt

Wszystkie table (łącznie z nazwą tabeli i kolumn) są zaszyfrowane algorytmem md5 używając funkcji hashującej SHA512 dzięki bibliotece SQLCipher

2.3.2 Operacje na bazie danych

- Wstawianie i pobieranie danych o cukrzycy.
- Rejestracja użytkowników i autentyfikacja.

3 Implementacja

3.0.1 app.py

Tutaj uruchamiana jest aplikacja Flask. Prócz rozruchu w pliku zawarta jest definicja naszej sieci neuronowej oraz definicje endpointów RestAPI.

3.0.2 db_manager.py

W tym miejscu znajduje się klasa odpowiedzialna za inicjalizację oraz obsługę bazy danych, podczas działania programu.

Inicjalizacja bazy danych:

Otwórz i wczytaj bazę danych pod podaną ścieżką
if *Podany plik nie istnieje lub nie jest bazą danych* **then**
 | Stwórz nową bazę danych pod podaną ścieżką i ją wczytaj
end
if *Baza danych nie zawiera, którejś z tabel "diabetes", "doctors" oraz "patients"* **then**
 | Stwórz tabele "diabetes", "doctors" oraz "patients" zgodnie z ich definicjami oraz
 | dodaj do tabeli "doctors" domyślnego administratora
end

```

1      def _setup_database_object(self) -> None:
2          try:
3              self.db = SqliteCipher(
4                  dataBasePath=os.getenv("DB_PATH"),
5                  checkSameThread=False,
6                  password=os.getenv("DB_PASSWORD"),
7              )
8          except Exception as ex:
9              logging.error(ex)
10
11         for table_name in ["diabetes", "doctors", "patients"]:
12             if not self.db.checkTableExist(table_name):
13                 self._init_database()
14                 break
15
16     def _init_database(self) -> None:
17         self.db.createTable(
18             "diabetes",
19             [
20                 ["pregnancies", "INT"],
21                 ["glucose", "REAL"],
22                 ["blood_pressure", "REAL"],
23                 ["skin_thickness", "REAL"],
24                 ["insulin", "REAL"],
25                 ["bmi", "REAL"],
26                 ["diabetes_pedigree_function", "REAL"],
27                 ["age", "INT"],
28                 ["outcome", "INT"],
29             ],
30             True,
31             True,
32         )
33
34         self.db.createTable(
35             "doctors",
36             [
37                 ["first_name", "TEXT"],
38                 ["last_name", "TEXT"],
39                 ["email", "TEXT"],
40                 ["phone_number", "TEXT"],
41                 ["hashed_password", "TEXT"],
42             ],
43             True,

```

```

44         True,
45     )
46
47     self.db.insertIntoTable(
48         "doctors",
49         [
50             "Senior",
51             "Registrar",
52             "senior_registrar@hospital.com",
53             "+48-111-222-333",
54             password_utils.get_hashed_password(os.getenv("
                    DEV_PASSWORD"))),
55         ],
56         True,
57     )
58
59     self.db.createTable(
60         "patients",
61         [
62             ["PESEL", "TEXT"],
63             ["first_name", "TEXT"],
64             ["last_name", "TEXT"],
65             ["email", "TEXT"],
66             ["phone_number", "TEXT"],
67             ["historical_data", "BLOB"],
68         ],
69         True,
70         True,
71     )

```

3.0.3 Elementy pomocnicze

Pliki `db_models.py`, `jwt_utils.py` oraz `password_utils.py` zawierają modele oraz funkcje pomocnicze dla obsługi bazy danych oraz endpointów.

3.1 Testy

3.2 test_train_model_endpoint

- **Cel:** Testowanie endpointu odpowiedzialnego za trenowanie modelu uczenia maszynowego.
- **Proces:** Wysyłanie żądania POST na endpoint `/train` z plikiem `diabetes.csv`, zawierającym dane do treningu modelu. Używa tokena typu Bearer do autoryzacji.
- **Sprawdzenie:** Status odpowiedzi równy 200, co oznacza pomyślne przetworzenie żądania.
- **Wynik:** Wyświetla "train 200" po pomyślnym ukończeniu.

3.3 `get_login`

- **Cel:** Uzyskanie tokenu dostępu do autoryzacji w innych testach.
- **Proces:** Wysyłanie żądania POST na endpoint `/login` z danymi użytkownika. Funkcja zwraca cały obiekt odpowiedzi.

3.4 `test_predict_diabetes_endpoint`

- **Cel:** Testowanie endpointu do przewidywania cukrzycy.
- **Proces:** Wysyłanie żądania POST na endpoint `/predict` z danymi JSON zawierającymi informacje o pacjencie i jego parametrach zdrowotnych.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"predict: 200"` po pomyślnym ukończeniu.

3.5 `test_get_data_endpoint`

- **Cel:** Testowanie endpointu odpowiedzialnego za pobieranie danych.
- **Proces:** Wysyłanie żądania GET na endpoint `/data`.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"data: 200"` po pomyślnym ukończeniu.

3.6 `test_signup_endpoint`

- **Cel:** Testowanie endpointu do rejestracji użytkowników.
- **Proces:** Wysyłanie żądania POST na endpoint `/signup` z danymi nowego użytkownika.
- **Sprawdzenie:** Status odpowiedzi równy 200.
- **Wynik:** Wyświetla `"signup"`.

3.7 Wykonanie Testów

- Skrypt uzyskuje token dostępu poprzez wywołanie `get_login()`.
- Następnie sekwencyjnie wykonuje `test_get_data_endpoint`, `test_predict_diabetes_endpoint` i `test_train_model_endpoint`, przekazując uzyskany token do autoryzacji.

Testy pozwalały na weryfikację czy zmiany wprowadzanie podczas rozwoju aplikacji były bezpieczne z punktu widzenia api w sposób ciągły