Dokumentacja Techniczna Backend

Tomasz Chady

January 18, 2024

Contents

1 Dokumentacja API **2**
1.1 Endpointy . . . 2
1.2 Autoryzacja . . . 2
1.2.1 Login . . . 2
1.2.2 JWT . . . 2

2 Architektura Backendu **2**

3 Ścieżki **3**

4 Baza danych **4**
4.1 Schemat . . . 4
4.2 Modele . . . 4
4.2.1 Order . . . 4
4.2.2 Doctor . . . 5
4.2.3 Patient . . . 6
4.2.4 Hospital . . . 7

1 Dokumentacja API

1.1 Endpointy

1.2 Autoryzacja

Autoryzacja w systemie odbywa się na dwa sposoby. Są one zdefiniowane w authController.js. Pierwszym sposobem jest autoryzacja poprzez hasło i login. Drugim z nich to autoryzacja przez JWT token. Autoryzacja poprzez hasło i login jest wykorzystywana do logowania się do systemu. Metody autoryzacyjne są zdefiniowane w ramach Passport.js. Dodawane są one do endpointów w zależności od potrzeb.

1.2.1 Login

Autoryzacja poprzez login i hasło jest wewnętrznie nazwana `login`. Login i hasło są przesyłane w body requestu w formacie JSON. Przykładowy request jest przedstawiony poniżej.

```
...
"authInfo":{
    "login": "admin",
    "password": "admin"
}
...
```

1.2.2 JWT

Autoryzacja poprzez JWT token jest wewnętrznie nazwana `jwt`. JWT token jest przesyłany w nagłówku `Authorization` w formacie `Bearer <token>`. W systemie token JWT jest generowany po poprawnym zalogowaniu się do systemu.

2 Architektura Backendu

Backend został napisany w formie monolitycznego REST API. Głównym zadaniem backendu jest udostępnienie danych z bazy danych. Dodatkowo backend jest odpowiedzialny za interakcję z powiązanymi systemami oraz autoryzację użytkowników. Relatywna pozycja backendu w systemie jest przedstawiona na schemacie 1.

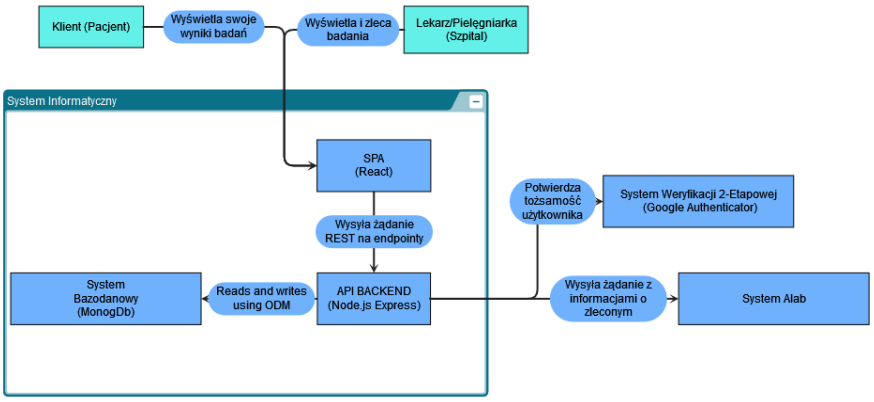

Figure 1: Schemat architektury

Backend jest napisany w języku JavaScript z użyciem środowiska Node.js. Aby zapewnić czytelność i łatwiejszy rozwój kod został podzielony na moduły w poszczególnych plikach i folderach. Poniżej znajduje się lista modułów wraz z krótkim opisem.

- **index.js** - główny plik aplikacji, zawiera konfigurację serwera, endpointów oraz uruchamia go.
- **env.js** - moduł odpowiedzialny za wczytanie zmiennych środowiskowych.
- **routes** - folder zawierający pliki z endpointami.

- **controllers** - folder zawierający pliki z kontrolerami.
- **services** - folder zawierający pliki z serwisami.
- **db** - folder zawierający pliki z modelami bazy danych.

Schemat zależności między modułami znajduje się na schemacie 2.

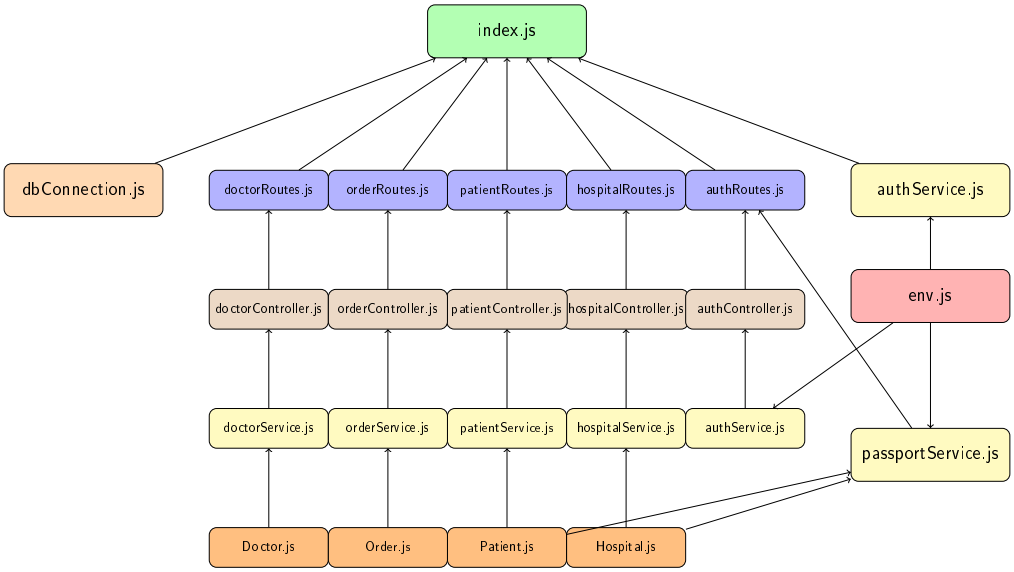

Figure 2: Schemat wymagań

Bardzo dobrze widać na tym wykresie warstwowość architektury backendu. Index.js wykorzystuje ścieżki zdefiniowane w routes do wywoływania odpowiednich funkcji zdefiniowanych w kontrolerach. Z kolei kontrolery wykorzystują serwisy do wykonywania operacji na bazie danych. Serwisy polegają na wywoływaniu odpowiednich funkcji zdefiniowanych w modelach bazy danych.

Dokładniejszy schemat zależności między funkcjami, z perspektywy API znajduje się na schemacie 3. Przedstawiono na nim jakie sposoby autentykacji są wykorzystywane w poszczególnych ścieżkach. Ilość endpointów jest za duża aby je wszystkie przedstawić na schemacie.

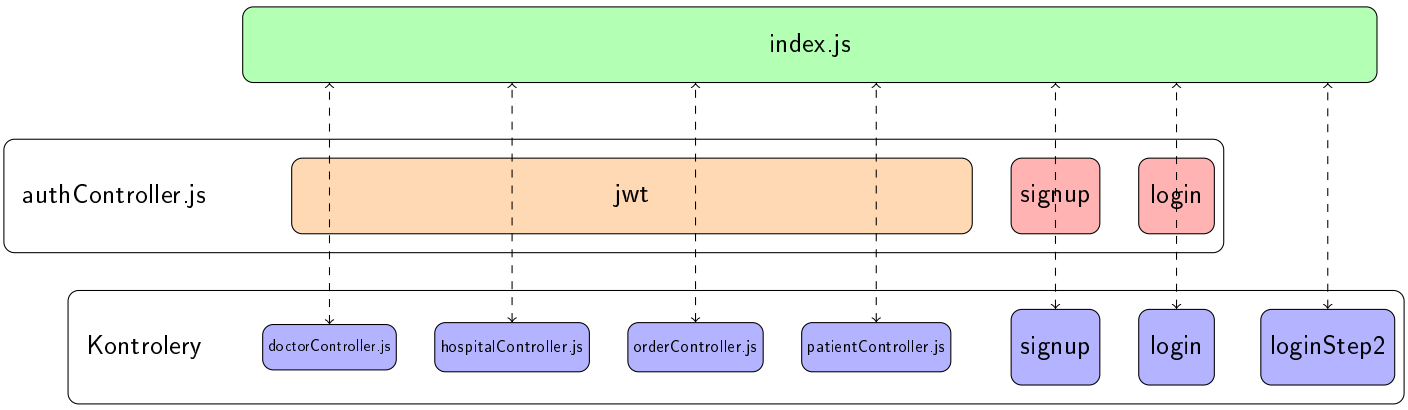

Figure 3: Schemat architektury wewnętrznej API

3 Ścieżki

W systemie istnieje kilka sekwencji operacji odpowiedzialnych za różne operacje. Kilka z nich jest przedstawionych poniżej.

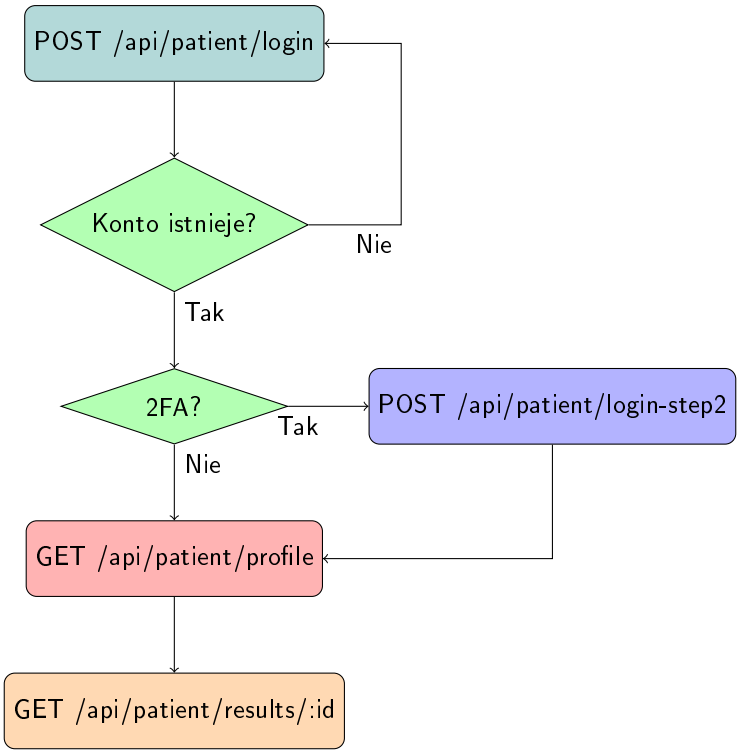

Figure 4: Schemat wyświetlania wyników

4 Baza danych

Do projektu został wybrany silnik bazodanowy Mongoose.db. Jest to silnik bazodanowy napisany w języku JavaScript, który działa na silniku MongoDB. MongoDB jest bazą danych typu NoSQL, która przechowuje dane w formacie JSON. Dzięki temu można w łatwy sposób przechowywać dane w formacie JSON, a także w łatwy sposób je przetwarzać. Moduł odpowiedzialny za połączenie z bazą danych to dbConnection.js.

4.1 Schemat

Schemat bazy danych jest przedstawiony na schemacie 6.

4.2 Modele

W systemie funkcjonują 4 modele danych. Są to `Doctor`, `Patient`, `Hospital` oraz `Order`. Zdefiniowane są one w formacie JSON i są przedstawione poniżej.

4.2.1 Order

Model order reprezentuje wynik badania. Jest on przypisany do konkretnego szpitala, pacjenta oraz doktora. Zawiera on również informacje o dacie wykonania badania oraz o wyniku.

```
{
    id: mongoose.Schema.Types.ObjectId,
    hospital: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Hospital',
        required: true,
    },
    doctor: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Doctor',
        required: true,
    },
```

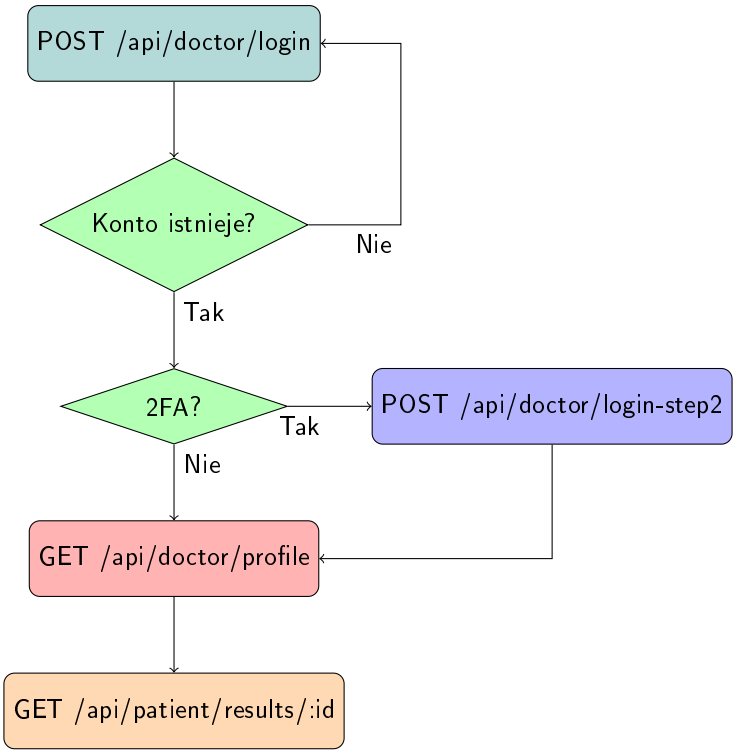

Figure 5: Schemat wyświetlania wyników przez lekarza

```
    patient: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Patient',
        required: true,
    },
    createdAt: {
        type: Date,
        default: Date.now,
    },
    editedAt: Date,
    results: {
        wbc: String,
        rbc: String,
        hct: String,
        mcv: String,
        mch: String,
        plt: String,
        mpv: String,
        rdw: String,
        pdw: String,
        hemoglobin: String,
    }
}
```

4.2.2 Doctor

Model doctor reprezentuje lekarza. Lekarz ma przypisane szpitale oraz nazwisko.

```
{
    id: mongoose.Schema.Types.ObjectId,
    surname: String,
    hospitals: [{
        type: mongoose.Schema.Types.ObjectId,
```

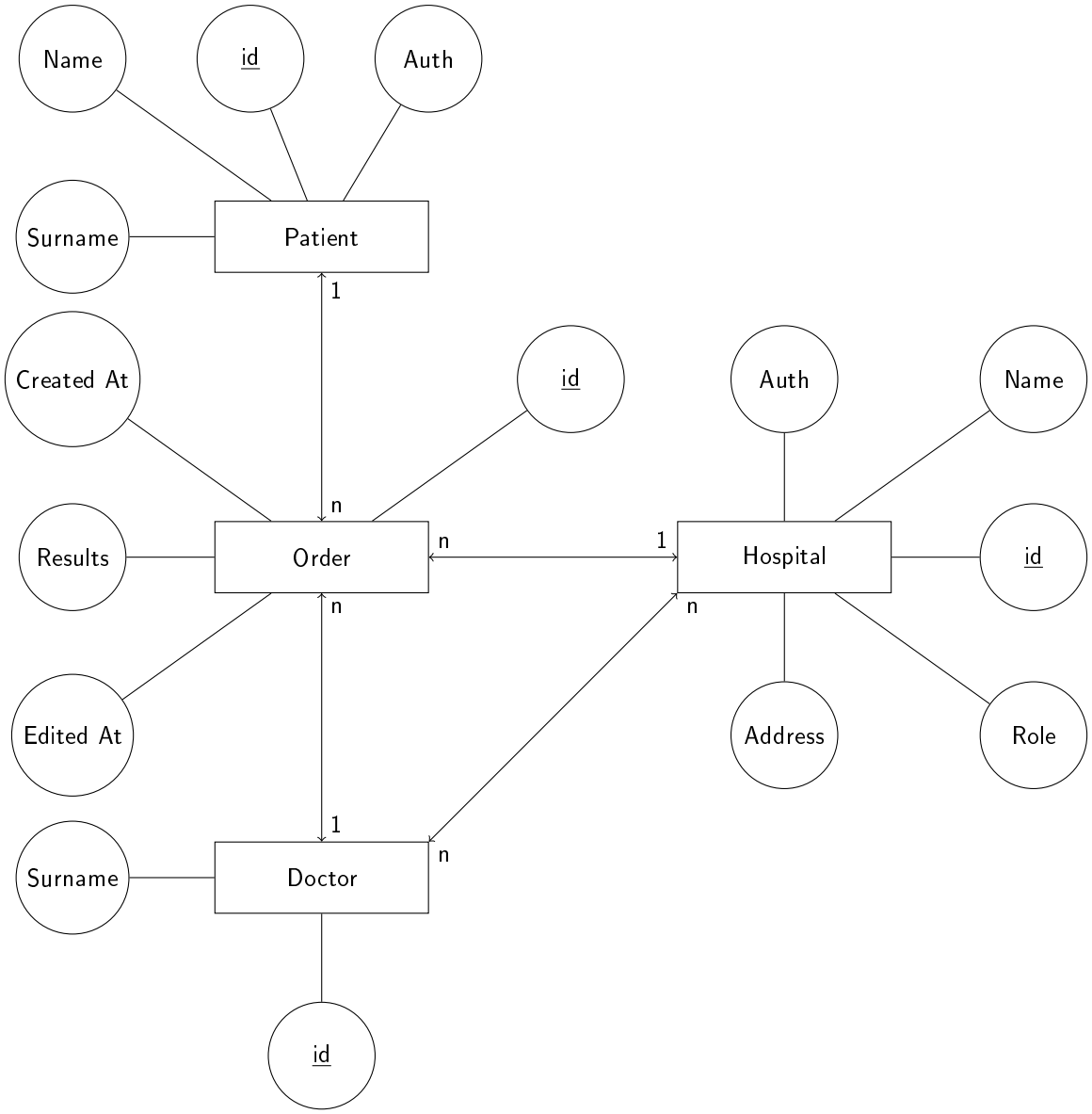

Figure 6: Schemat bazy danych

```
        ref: 'Hospital',
    }]
}
```

4.2.3 Patient

Model patient reprezentuje pacjenta. Pacjent ma przypisane wyniki badań, i co za tym idzie pośrednio ma przypisane szpitale.

```
{
    name: String,
    surname: String,
    authInfo: {
        login: {
            type: String,
            required: true,
            unique: true,
        },
        password: {
            type: String,
```

```
            required: true,
        },
    },
    twofaEnabled: {
        type: Boolean,
        default: false,
    },
    twofaSecret: {
        type: String,
        default: "",
    },
    orders: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Order',
    }]
}
```

4.2.4 Hospital

Model hospital reprezentuje szpital. Szpital jest powiązany z lekarzami oraz badaniami. Dodatkowo szpital ma przypisane dane autoryzacyjne.

```
{
    id: mongoose.Schema.Types.ObjectId,
    name: String,
    role: {
        type: String,
        validate: {
            validator: function (value) {
                return value === 'hospital';
            },
            message: 'Role must be "hospital"',
        },
    },
    authInfo: {
        login: String,
        password: String,
    },
    orders: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Order',
    }],
    doctors: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Doctor',
    }],
    address: {
        street: String,
        zipCode: String,
        city: String,
    },
    twofaEnabled: {
        type: Boolean,
        default: false,
    },
    twofaSecret: {
        type: String,
        default: "",
    }
}
```