MongoDB projekt

Mikołaj Gosztyła, Michał Dydek

27.05.2024

Spis treści

1 Kolekcje

1.1 Kolekcja employees

```
const employeeSchema = new mongoose.Schema({
    name: { type: String, required: true },
    surname: { type: String, required: true },
    employee_number: { type: Number, required: true, unique: true },
    password: { type: String, required: true }
});
```

Przypis 1: Employees

- **name** - imie pracownika
- **surname** - nazwisko pracownika
- **employee** number - numer pracownika
- **password** - hasło konta pracownika

1.2 Kolekcja res

```
const resSchema = new mongoose.Schema({
    employee_id: { type: String, required: true },
    date: { type: String, required: true },
    time: { type: String, required: true },
    duration: { type: Number, required: true },
    table: { type: String, required: true },
});
```

Przypis 2: Reservations

- **employee id** - numer pracownika, który dokonał rezerwacji
- **date** - dzień, miesiąc i rok rezerwacji
- **time** number - godzina rozpoczęcia rezerwacji
- **duration** - długość rezerwacji
- **table** - numer stolika, którego dotyczy rezerwacja

1.3 Kolekcja expenses

```
const expenseSchema = new mongoose.Schema({
  employee_number: { type: Number, required: true },
  item: { type: String, required: true },
  quantity: { type: Number, required: true },
  unit_price: { type: Number, required: true },
  date: { type: String, required: true },
});
```

Przypis 3: Expenses

- **employee number** - numer pracownika, który zerejestrował wydatek
- **item** - rzecz, której dotyczy dodany wydatek
- **quantity** - ilość
- **unit_price** - cena jednostkowa
- **date** - data wydatku

1.4 Kolekcja incomes

```
const incomeSchema = new mongoose.Schema({
    employee_number: { type: Number, required: true },
    order_id: { type: String, required: true },
    price: { type: Number, required: true },
    date: { type: String, required: true },
});
```

Przypis 4: Incomes

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD.

- **employee number** - numer pracownika, który zerejestrował wpływ
- **order id** - rzecz, której dotyczy dodany wpływ
- **price** - cena
- **date** - data wpływu

1.5 Kolekcja properties

```
const propertiesSchema = new mongoose.Schema({
    numberOfTables: { type: Number, required: true },
    openingTime: { type: String, required: true },
    closingTime: { type: String, required: true },
    closedDays: [{ type: String, required: true }]
});
```

Przypis 5: Properties

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD.

- **numberOfTables** - ilość stolików w restauracji
- **openingTime** - godzina otwarcia restauracji
- **closingTime** - godzina zamknięcia restauracji
- **closedDays** - dni, w które restauracja będzie zamknięta

2 Endpointy

2.1 GET

- \config - pobiera konfigurację restauracji, która znajduje się w folderze `Config` i pliku `restaurantProperties.json`
- \employees - pobiera listę wszystkich pracowników, bez hasła należącego do nich konta
- \reservations - pobiera listę wszystkich rezerwacji
- \expensesList - pobiera listę wszystkich wydatków
- \incomesList - pobiera listę wszystkich dochodów

2.2 POST

- \employeeAdd - dodaje nowego pracownika
- \login - loguje pracownika na jego konto
- \deleteAccount - usuwa konto pracownika
- \res - dodaje nową rezerwację
- \expense - dodaje nowy wydatek
- \income - dodaje nowy dochód
- \saveTurnoverData - zapisuje raport do pliku data.csv

2.3 PUT

- \expenses \ - aktualizuje wydatek o podanym ID
- \incomes \ - aktualizuje dochód o podanym ID

2.4 DELETE

- \reservations \ - usuwa rezerwację o podanym ID
- \expenses \ - usuwa wydatek o podanym ID
- \incomes \ - usuwa dochód o podanym ID

3 Raport

Raport, który umożliwia nasza aplikacja łączy 3 kolekcje: incomes, expenses, employees. Polega on na wygenerowaniu całkowitego obrotu i ilości dodanych rekordów do dwóch pierwszych kolekcji. Zdecydowaliśmy się na taki wybór, ponieważ uważamy, że jest to przydatna i potrzebna informacja dla pracodawcy, który pewnie chciałby mieć wgląd w niektóre dane konkretnych pracowników.

```
db.incomes.aggregate([
{
  $addFields: {
    year: { $year: { $toDate: "$date" } },
    month: { $month: { $toDate: "$date" } }
  }
},
{
  $project: {
    employee_number: 1,
    price: 1
  }
},
{
  $unionWith: {
    coll: "expenses",
    pipeline: [
    {
      $addFields: {
        year: { $year: { $toDate: "$date" } },
        month: { $month: { $toDate: "$date" } }
      }
    },
    {
      $project: {
        employee_number: 1,
        price: { $multiply: ["$unit_price", "$quantity"] }
      }
    }
    ]
  }
},
{
  $group: {
    _id: "$employee_number",
    count: { $sum: 1 },
    monetary_turnover: { $sum: "$price" }
  }
},
{
  $sort: {
    count: -1
  }
},
{
  $lookup: {
    from: "employees",
    localField: "_id",
    foreignField: "employee_number",
    as: "employee_info"
```

```
  }
},
{
  $unwind: "$employee_info"
},
{
  $project: {
    name: "$employee_info.name",
    surname: "$employee_info.surname",
    _id: 0,
    count: 1,
    monetary_turnover: 1
  }
}
])
```

Przypis 6: Zapytanie do bazy danych realizujące raport

4 Dyskusja zrealizowanych technik

4.1 Oprogramowanie dla pracowników

Zdecydowalismy się na stworzenie oprogramowania dla pracowników, a nie dla klientów ponieważ jako pracownik mamy większą kontrolę nad wprowadzanymi danymi. Dodatkowo podczas rozmowy na temat projektu pojawił się temat "złego człowieka", klienta który chciałby jak najbardziej uprzykrzyć życie. Zmieniając początkowe założenie eliminujemy jego negatywny wpływ na całą aplikacje poprzez zabranie możliwości wykonania ręcznie rezerwacji przez klienta. Klient dzwoni telefonicznie, natomiast pracownik wprowadza te dane do systemu.

4.2 Tworzenie i logowanie się do konta

Przy zakładaniu konta pracowniczego nowy pracownik jest proszony o wprowadzenie unikalnego numeru id. Ułatwia to następnie logowanie, a także łatwo można weryfikować, który pracownik jakie dane wprowadził do bazy danych.

4.3 Trigger przy dodawaniu nowej rezerwacji

Podczas dodawania nowej rezerwacji, by nie przechowywać zbędnych starych rezerwacji, usuwane są wszystkie rezerwacje, których data jest wcześniejsza niż aktualny dzień. CleanReservations to funkcja, która odpowiada za wyszukanie i usunięcie starych rezerwacji

```
1  resSchema.post('save', function(doc) {
2    cleanReservations();
3  })
```

Przypis 7: Trigger

4.4 Możliwość zapisu raportu

Nasza aplikacja wspiera zapisywanie wygenerowanego raportu do pliku .csv, który można potem łatwo analizować i obrabiać. Zdecydowaliśmy się na to, ponieważ uważamy, że samo wyświetlanie się danych nie byłoby wystarczające dla klienta i na pewno takie usprawnienie zwiększa wygodę używania aplikacji.