MongoDB projekt

Mikołaj Gosztyła, Michał Dydek

27.05.2024

Spis treści

1 Opis zadania i technik do jego realizacji

Aplikacja obsługuje restaurację i niektóre działania, które mogłyby być przydatne dla jej pracowników. Operacje, które wspiera nasze oprogramowanie to:

- rezerwacje - klient dzwoni do pracownika, który nastepnie wprowadza dane do systemu
- dodawanie wydatków - bieżące wydatki przez firmę mogą być wprowadzane przez pracownika
- dodawanie dochodów - po każdej transakcji pracownik również może wprowadzić dochody wraz z niektórymi informacjami odnośnie nich

Zdecydowalismy się na stworzenie oprogramowania od strony pracowniczej, a nie dla klientów ponieważ jako pracownik mamy większą kontrolę nad wprowadzanymi danymi.

1.1 Techniki wykorzystane do realizacji

todo: screeny frontendu, backendu

- Front-End - do realizacji front-endu wykorzystaliśmy język JavaScript, a także wykorzystaliśmy framework React
- Backend - backend realizuje skrypt napisany również w JavaScripcie przy użyciu frameworka Express, a do komunikacji z bazą danych został użyty framework Mongoose
- Baza danych - rodzaj bazy danych na jaką się zdecydowaliśmy to nierelacyjna baza MongoDB

2 Kolekcje

2.1 Kolekcja employees

W tej kolekcji przechowujemy dane na temat każdego pracownika. Aplikacja umożliwia logowanie każdego z użytkowników w systemie. Dodatkowo można tworzyć nowe konta. todo: zdjęcie danych

```
1 const employeeSchema = new mongoose.Schema({
2     name: { type: String, required: true },
3     surname: { type: String, required: true },
4     employee_number: { type: Number, required: true, unique: true },
5     password: { type: String, required: true }
6 });
```

Listing 1: Employees

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

2.2 Kolekcja res

W tej kolekcji znajdują się aktualne rezerwacje na stoliki w restauracji. Na tej kolekcji jest również nałożony dodatkowo trigger, który usuwa stare rezerwacje. todo: zdjęcie danych

```
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 },
});
```

Listing 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

2.3 Kolekcja expenses

Kolekcja, w której przechowywane są wydatki firmowe i każdy z pracowników ma możliwośc wprowadzenia danych, które są wykorzystywane w raporcie. todo: zdjęcie danych

```
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 },
});
```

Listing 3: Expenses

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD. todo: opis cruda

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

2.4 Kolekcja incomes

Kolekcja, w której przechowywane są przychody dla firmy i każdy z pracowników ma również możliwośc wprowadzenia danych, które są później wykorzystywane w raporcie. todo: zdjęcie danych

```
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 },
});
```

Listing 4: Incomes

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD. todo: opis cruda

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

2.5 Kolekcja properties

todo: zdjęcie danych

```
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 }]
});
```

Listing 5: Properties

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

3 Endpointy

3.1 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

3.2 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

3.3 PUT

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

3.4 DELETE

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

4 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. Dodatkowo można uwzględnić konkretny rok i miesiąc, by wygenerować raport dla konkretnej daty.

```
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
  }
}
])
```

Listing 6: Zapytanie do bazy danych realizujące raport

5 Dyskusja zrealizowanych technik

5.1 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  })
```

Listing 7: Trigger

5.2 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.

todo: jescze jakies jedno by sie przydalo