

MiniProject - Restauracja

Imiona i nazwiska autorów : Antoni Dulewicz, Marcin Serafin, Wojciech Wietrzny

Technologie

Technologia bazodanowa - MongoDB

Technologia serwerowa - Express

Technologia backendowa - Node.js

Wiele elementów znajduje się w odpowiednich plikach podanych przy nagłówkach dla czytelności sprawozdania

Kolekcje:

Poniżej znajdują się schematy tabel zaimplementowanych w MongoDB, które wykorzystuje nasza aplikacja

Clients

Kolekcja przechowująca dane klientów logujących się do aplikacji restauracji. Każdy klient ma przypisane ID, dane osobiste oraz kontaktowe, hasło i historię zamówień.

```
[
  {
    "_id": "client_id",
    "name": "Imie",
    "surname": "Nazwisko",
    "email": "Imie.Nazwisko@example.com",
    "password": "hashed_password",
    "phone": "123-456-7890",
    "address": "Ulica 1 00-000 Miasto",
    "history": [
      {
        "order_id": "order_id",
        "date": "YYYY-MM-DD",
        "price": 37.50,
        "address": "Ulica 1 00-000 Miasto",
        "status": "status",
        "dishes": [
          {
            "dish_id": "dish_id",
            "name": "name1",
            "quantity": 2,
            "price": 25.00
          }
        ]
      }
    ]
  }
]
```

```
        {
          "dish_id": "dish_id",
          "name": "name2",
          "quantity": 1,
          "price": 12.50
        }
      ]
    },
    "reservations": [
      {
        "date": "YYYY-MM-DD",
        "time": "HH:MM",
        "people": 4,
        "isCanceled": false
      }
    ]
  }
]
```

Dishes

Kolekcja zawierająca dostępne do zamówienia potrawy. Każda potrawa ma swoje ID, nazwę, opis, cenę oraz produkty z których została przygotowana.

```
[
{
  "_id": "dish_id",
  "name": "dish name",
  "description": "dish description",
  "price": 12.99,
  "products": [
    {"name": "product name 1", "supplier_name": "supplier_name_1", "quantity": 100, "unit": "grams"},
    {"name": "product name 2", "supplier_name": "supplier_name_2", "quantity": 200, "unit": "ml"}
  ]
}
]
```

Products

Kolekcja produktów podstawowych z których składają się dania w kolekcji dishes. Każdy produkt posiada swoje ID, nazwę, nazwę dostawcy, cenę oraz ilość w magazynie.

```
[
{
  "_id": "product_id",
  "name": "product name",
```

```
"supplier_name": "supplier_name_1",
"price": 12.99,
"stock": {
  "quantity": 5000,
  "unit": "grams"
}
}
```

Suppliers

Kolekcja w której znajdują się dane dostawców produktów podstawowych używanych do produkcji dań w kolekcji dishes. Dane te zawierają ID dostawcy, nazwę, dane kontaktowe oraz ID produktów które dostarczają.

```
[
{
  "_id": "supplier_id",
  "name": "supplier name",
  "contact": {
    "phone": "987-654-3210",
    "email": "supplier@exaple.com",
    "address": "Ulica 2 00-000 Miasto"
  },
  "products_supplied": [
    "product_id_1",
    "product_id_2"
  ]
}
```

Carts

Kolekcja w której trzymane są aktualne koszyki osób dokonujących zamówień na stronie restauracji

```
[
{
  "_id": "cart_id",
  "client_id": "client_id",
  "dishes": [
    {"dish_id": "dish_id_1", "quantity": 2},
    {"dish_id": "dish_id_2", "quantity": 1},
  ]
}
```

Orders

Kolekcja w której trzymane są złożone już zamówienia klientów. Każde zamówienie posiada swoje ID, dane klienta który złożył zamówienie oraz zawartość zamówienia wraz z datą, ceną i statusem.

```
[
{
  "_id": "order_id",
  "client": {
    "client_id": "client_id",
    "name": "name",
    "surname": "surname"
  },
  "date": "YYYY-MM-DD",
  "cart": "cart_id",
  "dishes": [
    {"dish_id": "dish_id_1", "dish_name": "name1", "quantity": 2, "price": 25.00},
    {"dish_id": "dish_id_2", "dish_name": "name1", "quantity": 1, "price": 12.50},
  ],
  "price": 37.50,
  "address": "Ulica 1 00-000 Miasto",
  "status": "status"
}
]
```

SupplierOrders

Kolekcja zawierająca zamówienia, które restauracja złożyła u pewnego dostawcy, każde takie zamówienie posiada swoje ID, ID dostawcy, date, produkty które obejmuje, cenę oraz status realizacji.

```
[
{
  "_id": "SupplierOrders_id",
  "supplier_id": "supplier_id",
  "date": "YYYY-MM-DD",
  "product": {
    "product_id": "product_id",
    "price": 75.00
  },
  "Price": 75.00,
  "status": "status"
}
]
```

Operacje

CRUD - CRUD.js

Ponizej opisane zostały operacje połączenia się z bazą MongoDB, zamknięcia tego połączenia oraz operacje CRUD - create, read, update, delete na podstawie kolekcji Products. Operacje CRUD dla innych kolekcji zrealizowane są analogicznie i zapisane w pliku CRUD.js

```
const { MongoClient } = require('mongodb');

const uri = 'mongodb://localhost:27017';
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

async function connect() {
  try {
    await client.connect();
    console.log('Connected to MongoDB');
  } catch (error) {
    console.error('Error connecting to MongoDB:', error);
  }
}

async function close() {
  try {
    await client.close();
    console.log('MongoDB connection closed');
  } catch (error) {
    console.error('Error closing MongoDB connection:', error);
  }
}

const Products = {
  async create(name, supplier_name, price, stock) {
    try {
      const product = { name, supplier_name, price, stock };
      const database = client.db('RestaurantDataBaseProject');
      const collection = database.collection('Products');
      const result = await collection.insertOne(product);
      return result.insertedId;
    } catch (error) {
      console.error('Error creating product:', error);
      return null;
    }
  },

  async read(name) {
    try {
      const database = client.db('RestaurantDataBaseProject');
      const collection = database.collection('Products');
      return await collection.findOne({ name: name });
    } catch (error) {
      console.error('Error reading products:', error);
      return [];
    }
  },
}
```

```
    async update(name, updates) {
      try {
        const database = client.db('RestaurantDataBaseProject');
        const collection = database.collection('Products');
        await collection.updateOne({ name: name }, { $set: updates });
        return true;
      } catch (error) {
        console.error('Error updating product:', error);
        return false;
      }
    },

    async delete(name) {
      try {
        const database = client.db('RestaurantDataBaseProject');
        const collection = database.collection('Products');
        await collection.deleteOne({ _id: id });
        return true;
      } catch (error) {
        console.error('Error deleting product:', error);
        return false;
      }
    }
  }
};
```

Agregacje - Aggregate.js

Poniższy paragraf opisuje stworzenie raportu wydatków poszczególnych klientów

```
const { MongoClient } = require('mongodb');

const uri = 'mongodb://localhost:27017';
const client = new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true });

async function baseAggregate(db) {
  const ordersCollection = db.collection('Orders');
  const clientsCollection = db.collection('Clients');

  const pipeline = [
    {
      $lookup: {
        from: 'Clients',
        localField: 'client.client_id',
        foreignField: '_id',
        as: 'clientDetails'
      }
    },
    {
      $unwind: '$clientDetails'
    }
  ],
```

```

    {
      $group: {
        _id: '$client.client_id',
        totalSpent: { $sum: '$price' },
        totalDishesOrdered: { $sum: { $size: '$dishes' } },
        clientInfo: { $first: '$clientDetails' }
      }
    },
    {
      $project: {
        _id: 0,
        client_id: '$_id',
        clientName: { $concat: ['$clientInfo.name', ' ', '$clientInfo.surname'] },
        totalSpent: 1,
        totalDishesOrdered: 1
      }
    },
    {
      $sort: { totalSpent: -1 }
    }
  ];

  const result = await ordersCollection.aggregate(pipeline).toArray();
  console.log('Client Spending Report:', result);
}

async function main() {
  const uri = "mongodb://localhost:27017/"; // replace with your MongoDB
  connection string
  const client = new MongoClient(uri, { useNewUrlParser: true,
    useUnifiedTopology: true });

  try {
    await client.connect();
    const database = client.db('RestaurantDataBaseProject');
    await baseAggregate(database);
  } finally {
    await client.close();
  }
}

main().catch(console.error);

```

Raport wydatków robimy po kolei najpierw łącząc się z bazą danych, następnie pobierając kolekcje Orders oraz Clients, łącząc te kolekcje po id klienta i sumując cenę zleceń jako totalSpent oraz ilość zamówionych potraw jako totalDishesOrdered

Front-End

Nasza aplikacja zawiera widoki pozwalające na zrealizowanie wszystkich zaimplementowanych operacji dla klientów tj.: dodanie potrawy do koszyka, złożenie zamówienia, zarezerwowanie stolika, podgląd przyszłych

rezerwacji czy podgąd historii zamówień, a także 2 pierwotne czyli rejestracje i logowanie. Widoki zostały zrealizowane przy pomocy javascripta, css oraz pug.js

Rejestracja

Imie

Nazwisko

Email

Haslo

Phone

Address

Zarejestruj

Logowanie

Imie

Nazwisko

Haslo

Zaloguj

Nie masz konta? [Zarejestruj sie](#)

Restauracja

<div><div>Pierogi ruskie ▾</div><div>Tradycyjne pierogi z farszem ziemniaczano-serowym</div><div>Dodaj do koszyka</div><div>Cena: 18.5 zł</div></div>	<div><div>Schabowy z ziemniakami ▾</div><div>Klasyczny kotlet schabowy podawany z ziemniakami i surówką</div><div>Dodaj do koszyka</div><div>Cena: 22 zł</div></div>	<div><div>Zupa pomidorowa ▾</div><div>Zupa pomidorowa z makaronem</div><div>Dodaj do koszyka</div><div>Cena: 12.5 zł</div></div>
<div><div>Salatka grecka ▾</div><div>Świeża sałatka z warzywami, oliwkami i serem feta</div><div>Dodaj do koszyka</div><div>Cena: 15 zł</div></div>	<div><div>Tarta szpinakowa ▾</div><div>Tarta ze szpinakiem i serem feta</div><div>Dodaj do koszyka</div><div>Cena: 17.5 zł</div></div>	<div><div>Ryba z frytkami ▾</div><div>Smażona ryba z frytkami i surówką</div><div>Dodaj do koszyka</div><div>Cena: 20 zł</div></div>
<div><div>Risotto z kurczakiem ▾</div><div>Risotto z kurczakiem i warzywami</div><div>Dodaj do koszyka</div><div>Cena: 19.5 zł</div></div>	<div><div>Makaron bolognese ▾</div><div>Makaron z sosem bolognese</div><div>Dodaj do koszyka</div><div>Cena: 16.5 zł</div></div>	<div><div>Makaron carbonara ▾</div><div>Makaron carbonara</div><div>Dodaj do koszyka</div><div>Cena: 19 zł</div></div>
<div><div>Dodaj do koszyka</div><div>Cena: 18.5 zł</div></div>	<div><div>surówką</div><div>Dodaj do koszyka</div><div>Cena: 22 zł</div></div>	<div><div>Dodaj do koszyka</div><div>Cena: 12.5 zł</div></div>
<div><div>Salatka grecka ▾</div><div>Świeża sałatka z warzywami, oliwkami i serem feta</div><div>Dodaj do koszyka</div><div>Cena: 15 zł</div></div>	<div><div>Tarta szpinakowa ▾</div><div>Tarta ze szpinakiem i serem feta</div><div>Dodaj do koszyka</div><div>Cena: 17.5 zł</div></div>	<div><div>Ryba z frytkami ▾</div><div>Smażona ryba z frytkami i surówką</div><div>Dodaj do koszyka</div><div>Cena: 20 zł</div></div>
<div><div>Risotto z kurczakiem ▾</div><div>Risotto z kurczakiem i warzywami</div><div>Dodaj do koszyka</div><div>Cena: 19.5 zł</div></div>	<div><div>Makaron bolognese ▾</div><div>Makaron z sosem bolognese</div><div>Dodaj do koszyka</div><div>Cena: 16.5 zł</div></div>	<div><div>Makaron carbonara ▾</div><div>Makaron carbonara</div><div>Dodaj do koszyka</div><div>Cena: 19 zł</div></div>

Twój Koszyk

Schabowy z ziemniakami - Cena: 22 zł	Delete
Pierogi ruskie - Cena: 18.5 zł	Delete

Wyczyść koszyk

Złóż zamówienie

[Wróć na stronę główną](#)

Twóje zamówienia

Schabowy z ziemniakami - Cena: 22 zł	Data: 2024-06-07 Cena zamówienia: 40.5 zł Adres: 123 Status: pending
Pierogi ruskie - Cena: 18.5 zł	

[Wróć na stronę główną](#)

Data

dd.mm.rrrr

Czas

--:--

Liczba osób

Zrób rezerwację

Twoje rezerwacje

Data: 2024-09-13, godzina: 15:42, liczba osób: 5

Cancel

[Wróć na stronę główną](#)

Dodatkowo zaimplementowano również widoki dla administratora, które obejmują sprawdzanie obecnych zamówień, przeglądanie historii dostarczonych zamówień, przeglądanie rezerwacji, a także zamawianie dostaw produktów do restauracji

Oczekujące zamówienia

Nie ma żadnych oczekujących zamówień

Dostarczone zamówienia

[Pokaż dostarczone zamówienia](#)

[Wróć na stronę główną](#)

Rezerwacje

Nie ma żadnych rezerwacji

[Wróć na stronę główną](#)

Zamówienia Produktów

<div>Ziemniaki</div> <div>Dostawca Janek</div> <div>Zamów</div> <div>Cena: 2.5 zł</div>	<div>Ser twarogowy</div> <div>Dostawca Anna</div> <div>Zamów</div> <div>Cena: 3.5 zł</div>	<div>Schab</div> <div>Dostawca Marek</div> <div>Zamów</div> <div>Cena: 5.5 zł</div>
<div>Pomidor</div> <div>Dostawca Anna</div> <div>Zamów</div> <div>Cena: 1.5 zł</div>	<div>Makaron</div> <div>Dostawca Marek</div> <div>Zamów</div> <div>Cena: 2 zł</div>	<div>Warzywa mieszane</div> <div>Dostawca Janek</div> <div>Zamów</div> <div>Cena: 3 zł</div>
<div>Ser feta</div> <div>Dostawca Marek</div> <div>Zamów</div> <div>Cena: 4 zł</div>	<div>Szpinak</div> <div>Dostawca Janek</div> <div>Zamów</div> <div>Cena: 2.5 zł</div>	<div>Ryba</div> <div>Dostawca Anna</div> <div>Zamów</div> <div>Cena: 6 zł</div>
<div>Frytki</div> <div>Dostawca Marek</div> <div></div> <div></div>	<div>Kurczak</div> <div>Dostawca Marek</div> <div></div> <div></div>	<div>Sos holenderski</div> <div>Dostawca Anna</div> <div></div> <div></div>

Uruchomienie

Stworzenie bazy danych - Site/dbCreator.js

Poniżej znajduje się wycinek ze skryptu dbCreator.js służący do tworzenia bazy danych MongoDB oraz dodawania do niej podstawowych tabel wraz z bazowymi informacjami na temat m.in. potraw czy produktów. Całość skryptu - wraz z brakującym tutaj schematem bazy danych dostępna jest w pliku dbCreatord.js

```
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
const { MongoClient } = require('mongodb');
const fs = require('fs');

const jsonFiles = [
  {collectionName: 'Products', filePath: 'JSONY/Products.json' },
  {collectionName: 'Dishes', filePath: 'JSONY/Dishes.json' },
```

```
    {collectionName: 'Suppliers', filePath: 'JSONY/Suppliers.json' },
    {collectionName: 'Admins', filePath: 'JSONY/Admins.json' },
    {collectionName: 'Clients', filePath: 'JSONY/Clients.json' },
  ];

  const collections = [
    {collectionName: 'Products'},
    {collectionName: 'Clients'},
    {collectionName: 'Dishes'},
    {collectionName: 'Suppliers'},
    {collectionName: 'Carts'},
    {collectionName: 'Orders'},
    {collectionName: 'SupplierOrders'},
    {collectionName: 'Admins'},
    {collectionName: 'Sessions'},
    {collectionName: 'Reservations'}
  ];

  async function removeCollection(database, collections) {
    for (const {collectionName} of collections) {
      try {
        const collection = database.collection(collectionName);
        await collection.drop();
        console.log(`Removed ${collectionName} collection.`);
      } catch (err) {
        console.error('Error removing collection:', err);
      }
    }
  }

  async function createCollectionsWithSchemas(db, schemas) {
    for (const [collectionName, schema] of Object.entries(schemas)) {
      try {
        console.log(`Creating collection ${collectionName} with schema validation...`);
        if (collectionName === 'Clients') {
          await db.createCollection(collectionName, {
            validator: { $jsonSchema: schema.$jsonSchema },
            validationLevel: "strict",
            validationAction: "error",
          });
          console.log(`Collection ${collectionName} created with schema validation.`);
          continue;
        }
        else{
          await db.createCollection(collectionName, {
            validator: { $jsonSchema: schema.$jsonSchema },
            validationLevel: "strict",
            validationAction: "warn",
          });
          console.log(`Collection ${collectionName} created with schema validation.`);
        }
      }
    }
  }
```

```
    } catch (err) {
      console.error(`Error creating collection ${collectionName}:`, err);
    }
  }
}

async function clearCollections(database, collections) {
  for (const { dbName, collectionName } of collections) {
    try {
      const collection = database.collection(collectionName);
      const result = await collection.deleteMany({});
      console.log(`Cleared ${result.deletedCount} documents from the
${collectionName} collection.`);
    } catch (err) {
      console.error('Error clearing collection:', err);
    }
  }
}

async function uploadData(database, jsonFiles) {
  for (const { collectionName, filePath } of jsonFiles) {
    try {
      const collection = database.collection(collectionName);
      console.log(`Importing data to ${collectionName} collection...`);
      const data = JSON.parse(fs.readFileSync(filePath, 'utf8'));

      if (Array.isArray(data)) {
        await collection.insertMany(data);
      } else {
        await collection.insertOne(data);
      }

      console.log(`Data successfully imported to ${collectionName}
collection.`);
    } catch (err) {
      console.error('Error importing data:', err);
    }
  }
}

async function main() {
  const uri = "mongodb://localhost:27017/"; // replace with your MongoDB
  connection string (probably same but with 27017)
  const client = new MongoClient(uri, { useNewUrlParser: true,
  useUnifiedTopology: true });

  try {
    await client.connect();
    const database = client.db('RestaurantDataBaseProject');
    await removeCollection(database, collections);
    await createCollectionsWithSchemas(database, schemas);
    await clearCollections(database, collections);
    await uploadData(database, jsonFiles);
  }
}
```

```
    } catch (err) {  
      console.error(err);  
    } finally {  
      await client.close();  
    }  
  }  
  
  main().catch(console.error);
```

Uruchomienie serwera - Site/app.js

Aby uruchomić serwer należy najpierw wykonać `npm install` - służy pobraniu wszelkich pakietów niezbędnych do wystartowania serwera na naszym komputerze, następnie wykonać `node dbCreator.js`, aby zainicjalizować bazę danych trzymającą informacje z których korzystamy na stronie restauracji, a następnie włączyć stronę restauracji przy użyciu `node app.js`

```
npm install  
node dbCreator.js  
node app.js
```