

Sprawozdanie Backend

Technologie użyte w projekcie

Backend aplikacji RecipeHub został stworzony przy użyciu następujących technologii:

- **Node.js**: Środowisko uruchomieniowe JavaScript, które pozwala na uruchamianie kodu po stronie serwera.
- **TypeScript**: Język programowania będący nadzbiorem JavaScript, zapewniający statyczne typowanie i lepszą kontrolę nad strukturą kodu.
- **TypeORM**: ORM (Object-Relational Mapping) do zarządzania bazą danych, umożliwiający mapowanie obiektów na tabele w bazie danych.
- **GraphQL**: Język zapytań do API, umożliwiający pobieranie danych w elastyczny sposób, z precyzyjnym określeniem struktury odpowiedzi.
- **Apollo Server**: Implementacja serwera GraphQL, która zapewnia obsługę zapytań, mutacji i subskrypcji.
- **SQLite**: Lekka baza danych używana w środowisku deweloperskim.

Struktura projektu

Projekt backendu jest zorganizowany w modularny sposób, co ułatwia rozwój i utrzymanie. Główne elementy struktury to:

- **package.json**: Plik konfiguracyjny projektu, zawierający zależności i skrypty, np.:

```
"scripts": {  
  "start": "ts-node src/index.ts",  
  "seed": "ts-node src/seed.ts"  
}
```

- **tsconfig.json**: Konfiguracja TypeScript, definiująca m.in. ścieżki do plików źródłowych i docelowych.
- **src/**: Główny katalog źródłowy, zawierający:
 - **data-source.ts**: Konfiguracja połączenia z bazą danych, np.:

```
import { DataSource } from "typeorm";  
import { User } from "./entity/User";  
  
export const AppDataSource = new DataSource({  
  type: "sqlite",  
  database: "database.sqlite",  
  entities: [User],  
  synchronize: true,  
});
```

- **index.ts**: Punkt wejściowy aplikacji, uruchamiający serwer GraphQL.
- **seed-data.ts** i **seed.ts**: Skrypty do inicializacji danych w bazie.

- **typeDefs.ts**: Definicje schematów GraphQL.
- **entity/**: Definicje encji bazy danych.
- **resolvers/**: Implementacje resolverów GraphQL.

Struktura plików projektu

Poniżej przedstawiono strukturę plików projektu backendu w formie drzewa:

```

backend/
├── package.json
├── tsconfig.json
└── src/
    ├── data-source.ts
    ├── index.ts
    ├── seed-data.ts
    ├── seed.ts
    ├── typeDefs.ts
    └── entity/
        ├── Category.ts
        ├── Comment.ts
        ├── Ingredient.ts
        ├── Rating.ts
        ├── RecipeCategory.ts
        ├── RecipeImage.ts
        ├── RecipeIngredient.ts
        ├── RecipeStep.ts
        ├── RecipeTag.ts
        ├── Recipies.ts
        ├── Role.ts
        ├── Subscribers.ts
        ├── Tag.ts
        └── User.ts
    └── resolvers/
        ├── categoryResolvers.ts
        ├── commentResolvers.ts
        ├── imageResolvers.ts
        ├── ingredientResolvers.ts
        ├── ratingResolvers.ts
        ├── recipeResolvers.ts
        ├── roleResolvers.ts
        ├── subscribersResolvers.ts
        ├── tagResolvers.ts
        └── userResolvers.ts
    └── images/

```

Struktura ta odzwierciedla modularne podejście do organizacji kodu, gdzie każda funkcjonalność jest odseparowana w odpowiednich katalogach.

Struktura encji i bazy danych

Encje w projekcie są zdefiniowane w katalogu [src/entity/](#). Każda encja odpowiada tabeli w bazie danych. Przykładowa encja [User](#):

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id: number;

    @Column()
    name: string;

    @Column()
    email: string;

    @Column()
    password: string;
}
```

Encje są definiowane za pomocą dekoratorów TypeORM, takich jak [@Entity\(\)](#), [@Column\(\)](#), [@PrimaryGeneratedColumn\(\)](#), co pozwala na łatwe mapowanie obiektowo-relacyjne.

Sposób implementacji resolverów

Resolvery GraphQL znajdują się w katalogu [src/resolvers/](#). Każdy resolver odpowiada za obsługę zapytań, mutacji i subskrypcji dla konkretnego typu danych. Przykładowy resolver dla użytkowników:

```
import { User } from "../entity/User";

export const userResolvers = {
    Query: {
        users: async () => {
            return await User.find();
        },
    },
    Mutation: {
        createUser: async (_: any, { name, email, password }: any) => {
            const user = User.create({ name, email, password });
            await user.save();
            return user;
        },
    },
};
```

Resolvery są zaimplementowane jako obiekty, które definiują funkcje obsługujące zapytania ([Query](#)) i mutacje ([Mutation](#)).

Spójność projektu

Projekt jest spójny dzięki zastosowaniu:

- **TypeScript**: Zapewnia statyczne typowanie, co minimalizuje błędy i ułatwia refaktoryzację.
- **TypeORM**: Ujednolica sposób pracy z bazą danych, umożliwiając łatwe zarządzanie migracjami i relacjami.
- **GraphQL**: Umożliwia elastyczne i jednoznaczne definiowanie API.
- **Modularna struktura katalogów**: Każdy moduł (np. encje, resolvers) jest odseparowany, co ułatwia rozwój i utrzymanie projektu.

Definicja schematu GraphQL

Schemat GraphQL jest zdefiniowany w pliku `src/typeDefs.ts`. Zawiera definicje typów, zapytań, mutacji i subskrypcji. Przykładowe definicje:

```
type User {  
    id: ID!  
    name: String!  
    email: String!  
}  
  
type Query {  
    users: [User!]!  
}  
  
type Mutation {  
    createUser(name: String!, email: String!, password: String!): User!  
}
```

Zapytania i mutacje są zdefiniowane w sposób umożliwiający łatwe rozszerzanie funkcjonalności API.

Dodatkowe punkty

- **Seedowanie danych**: Skrypty `seed-data.ts` i `seed.ts` umożliwiają inicjalizację bazy danych przykładowymi danymi.
- **Obsługa błędów**: Projekt zawiera mechanizmy obsługi błędów, co zapewnia stabilność aplikacji.
- **Wydajność**: Dzięki zastosowaniu GraphQL i TypeORM, aplikacja jest wydajna i łatwa w skalowaniu.
- **Bezpieczeństwo**: Hasła użytkowników są przechowywane w postaci zaszyfrowanej, co zwiększa bezpieczeństwo danych.

Widoki, Triggery i Funkcje w Bazie Danych

Widoki w Bazie Danych

Poniżej przedstawiono widoki zaimplementowane w bazie danych. Widoki te umożliwiają łatwiejsze pobieranie danych z różnych tabel w uporządkowany sposób:

SQLiteStudio (3.4.4) - [view_recipe_stats (recipeHub)]

Databases

Filter by name

recipeHub (SQLite 3)

- Tables (14)
 - categories
 - comments
 - ingredients
 - ratings
 - recipe_categories
 - recipe_images
 - recipe_ingredients
 - recipe_steps
 - recipe_tags
 - recipes
 - roles
 - subscribers
 - tags
 - users
- Views (3)
 - All Categories
 - view_recipe_stats
 - view_recipes_ingredients

Query Data Triggers DDL

Grid view Form view

Filter data Total rows loaded: 30

recipe_id	recipe_name	avg_rating	ratings_count	comments_count
1	Classic Margherita Pizza	5	1	1
2	Creamy Chicken Alfredo	4	1	1
3	Spicy Thai Basil Noodles	4	1	1
4	Slow-Roasted Beef Brisket	3	1	1
5	Lemon Garlic Butter Salmon	3	1	1
6	Vegetable Ratatouille	3	1	1
7	Hearty Beef Stew	3	1	1
8	Grilled Shrimp Tacos	3	1	1
9	Mushroom Risotto	3	1	1
10	Butternut Squash Soup	4	1	1
11	Honey Soy Glazed Chicken	4	1	1
12	Pesto Pasta with Cherry Tomatoes	5	1	1
13	Greek Salad with Feta	3	1	1
14	Pulled Pork Sandwiches	5	1	1
15	Teriyaki Salmon Bowl	4	1	1
16	Spinach and Feta Frittata	3	1	1
17	Citrus Quinoa Salad	4	1	1
18	Garlic Butter Steak Bites	3	1	1
19	Vegetarian Chili	5	1	1
20	Chicken Tikka Masala	4	1	1
21	Soba Noodle Salad	3	1	1
22	Creamy Tomato Basil Soup	4	1	1
23	Roasted Cauliflower Steaks	3	1	1
24	Classic Beef Burger	4	1	1
25	Shrimp Scampi Linguine	5	1	1
26	Eggplant Parmesan	5	1	1
27	Thai Green Curry	3	1	1
28	Pan-Seared Scallops	5	1	1
29	Manila Glazed Pork Chops	3	1	1

Status

- [16:49:46] Reached the end of document. Hit the find again to restart the search.
- [16:50:36] Error while executing SQL query on database 'recipeHub': no such column: ri.path
- [16:50:49] Query finished in 0.001 second(s).
- [16:51:07] Error while executing SQL query on database 'recipeHub': near "AS" SELECT : syntax error
- [16:51:21] Error while executing SQL query on database 'recipeHub': near "SELECT": syntax error
- [16:51:28] Query finished in 0.009 second(s).
- [16:52:02] Committed changes for view 'All Categories' (named before 'test') successfully.

SQLiteStudio (3.4.4) - [view_recipes_ingredients (recipeHub)]

Databases

Filter by name

recipeHub (SQLite 3)

- Tables (14)
 - categories
 - comments
 - ingredients
 - ratings
 - recipe_categories
 - recipe_images
 - recipe_ingredients
 - recipe_steps
 - recipe_tags
 - recipes
 - roles
 - subscribers
 - tags
 - users
- Views (3)
 - All Categories
 - view_recipe_stats
 - view_recipes_ingredients

Query Data Triggers DDL

Grid view Form view

Filter data Total rows loaded: 30

recipe_id	recipe_name	ingredients_list
1	Classic Margherita Pizza	All-purpose flour
2	Creamy Chicken Alfredo	Granulated sugar
3	Spicy Thai Basil Noodles	Salt
4	Slow-Roasted Beef Brisket	Black pepper
5	Lemon Garlic Butter Salmon	Olive oil
6	Vegetable Ratatouille	Unsalted butter
7	Hearty Beef Stew	Eggs
8	Grilled Shrimp Tacos	Milk
9	Mushroom Risotto	Heavy cream
10	Butternut Squash Soup	Parmesan cheese
11	Honey Soy Glazed Chicken	Mozzarella cheese
12	Pesto Pasta with Cherry Tomatoes	Chicken breast
13	Greek Salad with Feta	Ground beef
14	Pulled Pork Sandwiches	Beef broth
15	Teriyaki Salmon Bowl	Tomato
16	Spinach and Feta Frittata	Onion
17	Citrus Quinoa Salad	Garlic
18	Garlic Butter Steak Bites	Basil
19	Vegetarian Chili	Oregano
20	Chicken Tikka Masala	Thyme
21	Soba Noodle Salad	Lemon
22	Creamy Tomato Basil Soup	Soy sauce
23	Roasted Cauliflower Steaks	Honey
24	Classic Beef Burger	Brown sugar
25	Shrimp Scampi Linguine	Rice
26	Eggplant Parmesan	Pasta
27	Thai Green Curry	Quinoa
28	Pan-Seared Scallops	Potato
29	Manila Glazed Pork Chops	Carrot

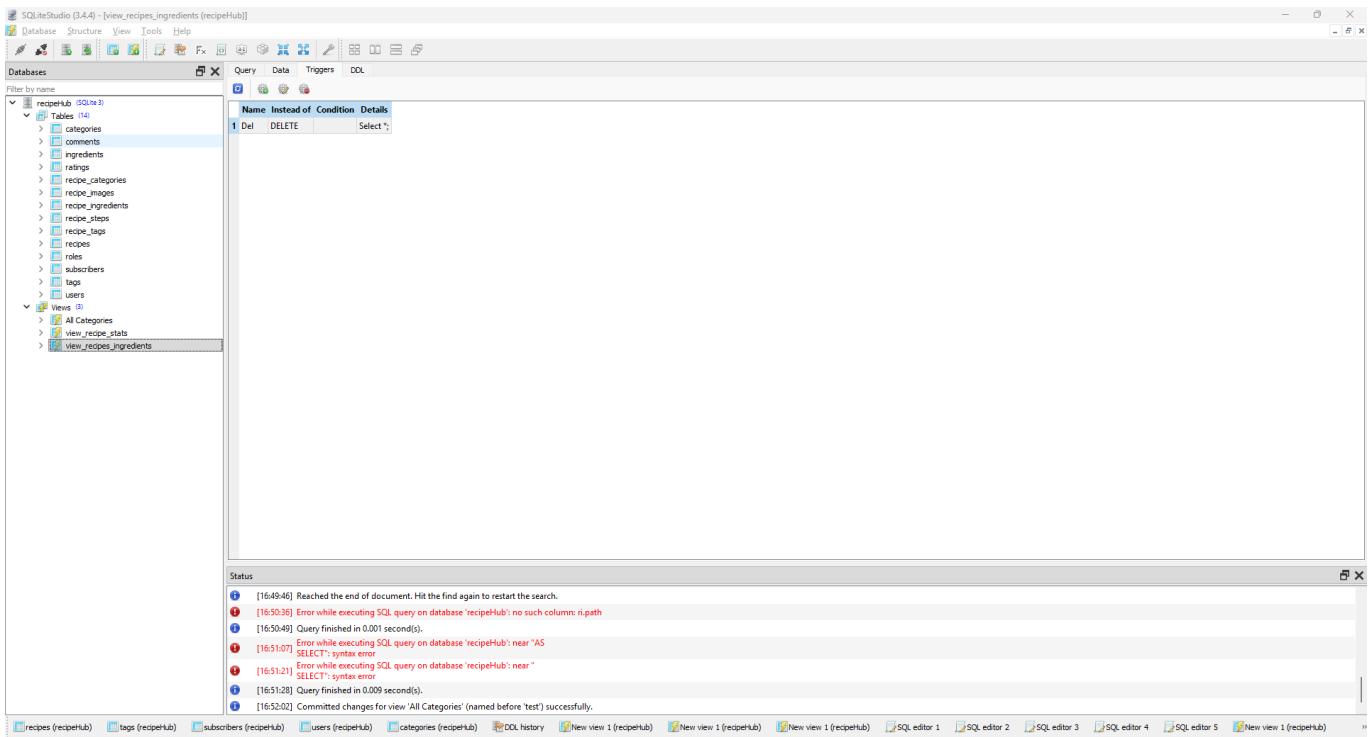
Status

- [16:49:46] Reached the end of document. Hit the find again to restart the search.
- [16:50:36] Error while executing SQL query on database 'recipeHub': no such column: ri.path
- [16:50:49] Query finished in 0.001 second(s).
- [16:51:07] Error while executing SQL query on database 'recipeHub': near "AS" SELECT : syntax error
- [16:51:21] Error while executing SQL query on database 'recipeHub': near "SELECT": syntax error
- [16:51:28] Query finished in 0.009 second(s).
- [16:52:02] Committed changes for view 'All Categories' (named before 'test') successfully.

Recipes (recipeHub) Tags (recipeHub) Subscribers (recipeHub) Users (recipeHub) Categories (recipeHub) DDL History New view 1 (recipeHub) New view 1 (recipeHub) New view 1 (recipeHub) SQL editor 1 SQL editor 2 SQL editor 3 SQL editor 4 SQL editor 5 New view 1 (recipeHub)

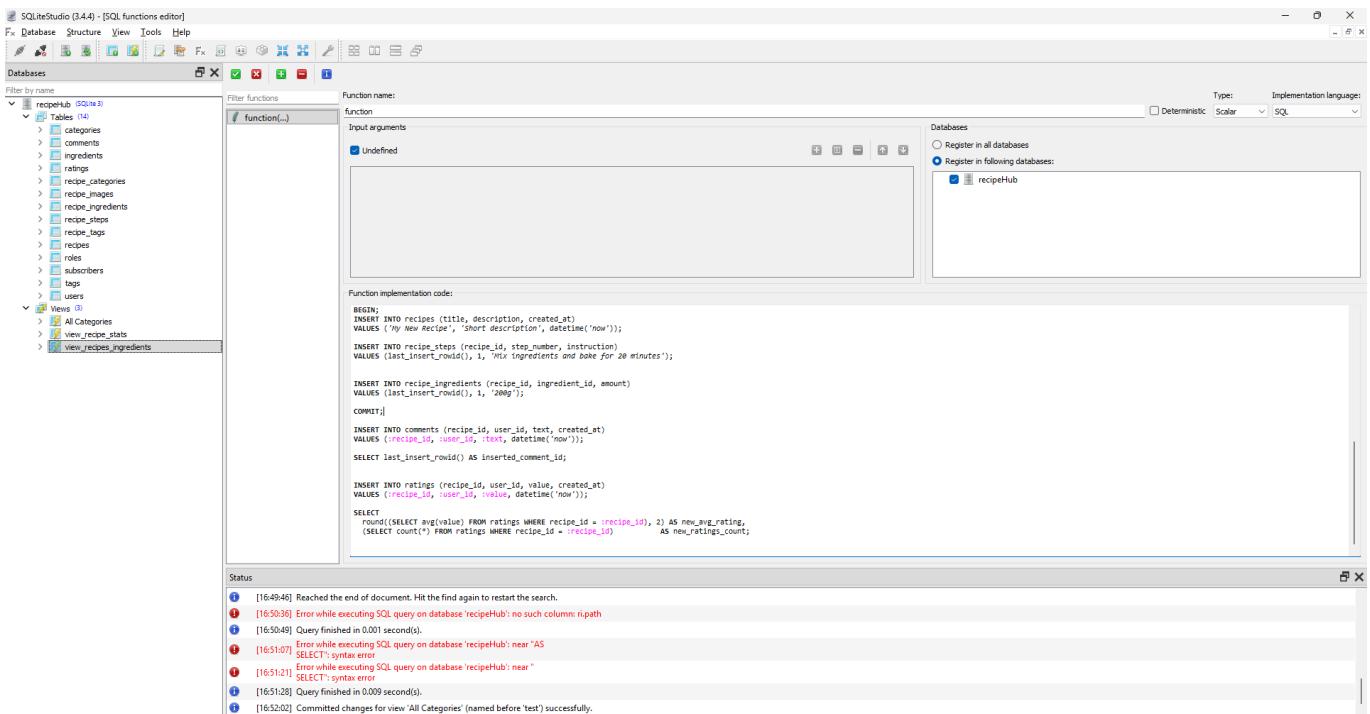
Trigger w Bazie Danych

Triggery zostały zaimplementowane w celu automatyzacji pewnych operacji w bazie danych, takich jak aktualizacje lub walidacje danych. Przykład triggera:



Funkcje w Bazie Danych

Funkcje w bazie danych zostały zaimplementowane w celu obsługi logiki, która normalnie byłaby realizowana w procedurach. Ze względu na ograniczenia SQLite, procedury nie są obsługiwane, dlatego logika została przeniesiona do funkcji.



Procedury

Procedury nie są stosowane w SQLite ze względu na jego ograniczenia. Logika, która normalnie byłaby zaimplementowana w procedurach, została przeniesiona do funkcji bazy danych.

SQLiteStudio (3.4.4) - [SQL functions editor]

Database Structure View Tools Help

Databases

Filter by name: procedure_example_as_function(...)

Function name: procedure_example_as_function

Type: Deterministic Scalar Implementation language: SQL

Input arguments: Undefined

Databases:

- Register in all databases
- Register in following databases:
 - recipeHub

Function implementation code:

```

SELECT
    r.id AS recipe_id,
    r.title AS recipe_title,
    r.description AS recipe_description,
    (SELECT group_concat(c.name, ', ')
     FROM recipe_categories rc
     JOIN categories c ON rc.category_id = c.id
     WHERE rc.recipe_id = r.id) AS categories,
    (SELECT group_concat(r.instruction, '; ')
     FROM recipe_steps rs
     JOIN recipe_steps r ON rs.step_order = r.step_order
     WHERE rs.recipe_id = r.id
     ORDER BY rs.step_number) AS steps_ordered,
    (SELECT group_concat(tc.name, ', ')
     FROM tags tc
     JOIN recipe_tags rt ON rt.tag_id = tc.id
     WHERE rt.recipe_id = r.id) AS tags,
    (SELECT round(avg(value),2) FROM ratings WHERE recipe_id = r.id) AS avg_rating,
    (SELECT count(*) FROM ratings WHERE recipe_id = r.id) AS ratings_count,
    (SELECT count(*) FROM comments WHERE recipe_id = r.id) AS comments_count
  
```

Status

- [16:49:46] Reached the end of document. Hit the find again to restart the search.
- [16:50:36] Error while executing SQL query on database 'recipeHub': no such column: ri.path
- [16:50:49] Query finished in 0.001 second(s).
- [16:51:07] Error while executing SQL query on database 'recipeHub': near "AS" SELECT
- [16:51:21] Error while executing SQL query on database 'recipeHub': near "SELECT"; syntax error
- [16:51:28] Query finished in 0.009 second(s).
- [16:52:02] Committed changes for view 'All Categories' (named before 'test') successfully.