Web Application 2 2020-2021

Exercise 5: Catalogue Service

The Catalogue service is a module of the e-commerce micro-services application that let users browse existing products, inspect their features and discuss them by adding comments. Finding specific products is made easier thanks to the support provided by filtering and sorting operations.

Implement the Catalogue Service using Express.js web application framework for Node.js, relying on MongoDB as a persistence layer and GraphQL as a query language.

To use these technologies, you'll need to install these libraries:

- express
- mongoose
- graphql
- express-graphql
- graphql-tools

The implementation should comply to the following GraphQL schema:

```
scalar DateTime,
enum ProductCategory {
    STYLE
    FOOD
    TECH
    SPORT
},
enum SortingValue {
    createdAt
    price
},
enum SortingOrder {
    asc
    desc
},
input ProductCreateInput {
    name : String!,
    description : String,
    price : Float!,
    category: ProductCategory!
},
input CommentCreateInput {
    title: String!,
    body: String,
    stars: Int!
}
type Comment {
    _id: ID!,
    title: String!,
    body: String,
    stars: Int!,
    date: DateTime!
},
```

```
type Product {
    _id: ID!,
    name: String!,
    createdAt: DateTime!,
    description: String,
    price: Float!,
    comments (numberOfLastRecentComments: Int) : [Comment],
    category: ProductCategory!,
    stars: Float
},
input FilterProductInput {
    categories: [ProductCategory],
    minStars: Int,
    minPrice: Float,
    maxPrice: Float
},
input SortProductInput {
    value: SortingValue!,
    order: SortingOrder!
},
type Query {
    products (filter: FilterProductInput, sort: SortProductInput) : [Product],
    product (id: ID!) : Product,
},
type Mutation {
    createProduct (createProductInput: CreateProductInput!) : Product,
    createComment (
       createCommentInput: CreateCommentInput!,
       productId: ID!
    ) : Comment
}
```

Build the schema using *makeExecutableSchema*({typeDefs, resolvers}) of *graphql-tools* module, instead of *buildSchema(schema).* In this way you should be able to add your custom resolvers for certain types, necessary to solve complex queries at nested levels (for example to retrieve Product comments with the limit condition).

Use the GraphiQL tool to test your implementation.