Cereal API

Opgavebeskrivelse

I denne opgave skal du udvikle et REST API til håndtering af et datasæt over forskellige morgenmadsprodukter (cereals). API'et skal understøtte CRUD-operationer, filtrering og adgangskontrol.

Afleveringsformat

Afleveringen skal uploades i et Github-repository bestående af:

1. Et API, der kan håndtere CRUD-operationer på en database.
2. En driverklasse til at demonstrere systemets funktionalitet.
3. Koden bør være velstruktureret og dokumenteret
4. README med dokumentation af API'et, herunder endpoints og svarformater (eventuelt med eksempler på brug), samt en beskrivelse af design og begrundelse for designbeslutninger. Tilføj evt. Diagrammer til jeres designbeskrivelser.

Teknologier

- .Net C# & Rider
- Nuget packages:
  - MySql.data & MySql.EntityFrameworkCore (til MySql database)
  - Pomelo.EntityFrameworkCore.MySql (til MySql database)
  - Swashbukler. AspNetCore (Til Swagger integration)
- MySQL & MySQL workbench

Casebeskrivelse

Du arbejder for en virksomhed, der ønsker at analysere og vise ernæringsdata for forskellige morgenmadsprodukter. Virksomheden har fået adgang til et datasæt over cereals, men de mangler en effektiv måde at søge og hente data på.

Din opgave er at udvikle et API, der kan håndtere forespørgsler på cereals, tillade filtrering af data baseret på næringsindhold, og tillade tilføjelse og opdatering af nye produkter. Derudover skal API'et sikre, at kun autoriserede brugere kan tilføje eller slette produkter.

Virksomheden ønsker, at API'et understøtter følgende funktioner:

1. Lagre produktdata i en lokal database.
2. Hentning af produktinformation.
3. Mulighed for at filtrere produkter efter næringsindhold.
4. Tilføjelse, opdatering og sletning af produkter (kun for autoriserede brugere).

Datasæt

Datasættet består af forskellige morgenmadsprodukter med følgende attributter:

Felt	**Beskrivelse**
Name	Navn på morgenmadsproduktet
mfr	Producent (A, G, K, N, P, Q, R) A = American Home Food Products; G = General Mills K = Kelloggs N = Nabisco P = Post Q = Quaker Oats R = Ralston Purina
type	Type (hot/cold)
calories	Kalorier pr. portion
protein	Gram protein pr. portion
fat	Gram fedt pr. portion
sodium	Milligram natrium pr. portion
fiber	Gram kostfibre pr. portion
carbo	Gram kulhydrater pr. portion

sugars	Gram sukker pr. portion
potass	Milligram kalium pr. portion
vitamins	Vitaminindhold (0, 25 eller 100 %) indikerer den typiske procentuelle FDA-anbefaling
shelf	Placering på hylde (1, 2, 3, tælles fra gulvet)
weight	Vægt i ounces pr. portion
cups	Antal kopper pr. portion
rating	Bedømmelse af produktet

Datasættet kan findes i den vedhæftede CSV-fil: 'Cereal.csv'.

Billederne kan findes i den vedhæfte zip mappe: 'Cereal pictures.zip'

Opgaver og opmærksomhedspunkter

Overvej at bruge Singleton-mønstret til at sikre, at der kun er én database-forbindelse i systemet.

Overvej at bruge Factory-pattern til at oprette objekter i systemet, både ved hentning fra databasen og ved kald fra API'et.

Brug gerne:

Brug ikke:

Opgave 1: Opsætning af database og datamodellering

- Opret en database til at lagre cereal.csv.
- Lav en parser, der kan indlæse produktdata fra CSV-filen og indsætte dem i databasen.
- Sørg for, at parseren kan håndtere ekstra data, hvis flere produkter tilføjes senere.
- Sørg for, at databasen understøtter CRUD-operationer.
- Sikre dataintegritet med relevante constraints (f.eks. unikke felter, ikke-null værdier).

Opgave 2: Implementering af API

- Implementér et REST API, der tillader GET, POST, PUT og DELETE (CRUD-operationer) på produktdata.
- GET skal kunne returnere alle produkter eller et enkelt produkt baseret på ID.
- Tilføj filtreringsmuligheder, så brugere kan hente produkter baseret på parametre som:
    - Kalorier (f.eks. /products?calories=120 for produkter med præcis 120 kcal pr. portion).
    - Producent (f.eks. /products?manufacturer=Kelloggs).
- Opret et endpoint, der tillader sletning af produkter baseret på ID.
- Opret et endpoint, der tillader oprettelse af nye produkter via POST-request.
    - Hvis et ID angives og produktet findes, så skal det opdateres.
    - Hvis et ID angives og objektet ikke findes, returneres en fejl, da ID'er ikke må vælges manuelt.
    - Hvis ingen ID angives, oprettes et nyt objekt.

Opgave 3: Sikkerhed og autentifikation

- Implementér brugerautentifikation, hvor:
    - GET-requests er offentligt tilgængelige.
    - POST, PUT og DELETE kræver login med brugernavn og password.
- Brug en sikker metode til password-håndtering (f.eks. hashing med bcrypt).

Opgave 4: Billedhåndtering

- Tilføj et endpoint, hvor man kan hente et billede af et produkt via produktets ID.
    - Forenklet løsning: Produkter fra samme producent kan dele det samme billede.
- Billeder skal kunne gemmes lokalt eller ved at lagre stierne i databasen.

Opgave 5 (Valgfri): Ekstra funktionalitet

Hvis I er blevet færdige og vil have inspiration til yderligere ting, der kan implementeres, så er nogle forslag:

- Intervalbaserede søgninger (f.eks. /products?calories>=100&calories<=200).
    - Understøt følgende operatorer:
    - = (eksakt værdi)
    - >= (større end eller lig med)
    - <= (mindre end eller lig med)
    - > (større end)
    - < (mindre end)
    - != (ikke lig med)
- Sortering af resultater (f.eks. /products?sort=calories_desc).
- Logging af API-anmodninger.
- Enhedstest og integrationstest.
- Rate limiting på API-kald for at forhindre misbrug.
- Kommunikation via HTTPS.

Pensum og Ressourcer

- **Teoridokumenter:** Designmønstre, filhåndtering, SQL primer, undtagelseshåndtering, REST API, HTTP, MySQL
- **Flask API:** **FLASK API**
- **SQL connector:** **SQL server**
- **Java:** **Guide to Spring Boot**
- **C#:** **ASP.NET Core Web API**
- **Python:** **Build Modern APIs using Flask**

God arbejdslyst!