Projekt Todo Liste

Olga Peshkova

Daniela Haumer

Beschreiben Sie mindestens 3 Services inkl. Bounded Context, Context Map, Datenmodel und Datenvalidierung  
10 Punkte

1. Task Service:

Bounded Context: TaskManagement

Beschreibung: Der Task-Service ist verantwortlich für das Verwalten von Tasks in der Todo-Liste. Er enthält Logik zur Erstellung, Aktualisierung, Löschung und Abfrage von Tasks. Jeder Task hat folgende Attribute:

* TaskID
* Beschreibung
* Fälligkeitsdatum
* Status

2. User Service:

Bounded Context: UserManagement

Beschreibung: Der User-Service ist für die Verwaltung von Userdaten zuständig. Hier können User erstellt, aktualisiert und gelöscht werden. Jeder User hat folgende Attribute:

* UserID
* Name
* E-Mail-Adresse
* Rolle

3. Assignment Service:

Bounded Context: AssignmentManagement

Beschreibung: Der Assignment-Service kümmert sich um die Zuweisung von Tasks an User. Es handelt sich um eine Verbindung zwischen den Services "Task" und "User". Der Service enthält Logik zur Erstellung, Aktualisierung und Löschung von Zuweisungen. Eine Zuweisung könnte Informationen darüber enthalten, welcher Benutzer für welche Aufgaben verantwortlich ist. Jedes Assignment hat folgende Attribute:

* TaskID
* UserID
* AssignDate

Context Map:

Eine Context Map visualisiert die Beziehungen und Abhängigkeiten zwischen den einzelnen Bounded Contexts.

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

In dieser Context Map sind die einzelnen Bounded Contexts (TaskManagement, UserManagement, AssignmentManagement) dargestellt, sowie ihre Beziehungen zueinander.

Das Datenmodel und die Datenvalidierung finden Sie im Code.

Entwicklen Sie mindestens einen ASP.NET Core Web.API Controller, welcher CRUD-Funktionaltitäten zur Verfügung stellt. Beschreiben Sie die REST-Prinzipien im Zusammenhang mit Ihrem Projekt.  
10 Punkte

Stateless:

Jede Anfrage vom Client an den Task-Service enthält alle Informationen, die zur Verarbeitung der Anfrage erforderlich sind. Der Server speichert keine Client-Zustandsinformationen zwischen den Anfragen.

Uniform Interface

Ressourcenidentifikation:

Eindeutigen URIs zur Identifikation von Tasks werden verwendet.

Repräsentation:

JSON als einheitliches Datenformat wird für die Darstellung von Tasks verwendet.

Nachrichten:

Es gibt eindeutige API-Nachrichten für die CRUD-Aktionen für Tasks.

Client - Server

Der Client und der Server sind voneinander unabhängig und können unabhängig entwickelt, skaliert und gewartet werden.

Als Client verwenden wir die Angular Anwendung aus einer früheren LV. Die Anbindung wurde Clientseitig angepasst (URL angepasst, etc.).

Server: In unserem Todo-Projekt ist der Task-Service die serverseitige Komponente.

Code on Demand

Angenommen, der TaskService möchte eine Funktion zum Sortieren von Tasks auf der Client-Seite hinzufügen. Der Server könnte einen JavaScript-Sortieralgorithmus an den Client senden. Der Client würde diesen Algorithmus verwenden, um Aufgaben zu sortieren, ohne dass der Server den gesamten sortierten Satz senden muss.

Cacheable

Caching könnte verwendet werden, um die Leistung zu verbessern. Der Server kann die entsprechenden Header in den API-Antworten setzen, um den Client zu beeinflussen.

Layered System

Der Task-Service kann als eine Schicht betrachtet werden. Andere Schichten können beispielsweise eine Datenbank oder ein Authentifizierungsdienst sein. Jede Schicht hat klare Verantwortlichkeiten und kann unabhängig von anderen entwickelt und skaliert werden.

**Beschreiben Sie die OpenAPI-Spec Ihres Service. Stellen Sie IDL, WSDL und OpenApi-Spec gegenüber  
10 Punkte**

Die OpenAPI-Spezifikation ist eine maschinenlesbare Beschreibung von RESTful APIs. Sie definiert die Endpunkte, Anfragemethoden, Antwortcodes und Datenmodelle für verschiedene Ressourcen in Ihrem Service.

- Name des Dienstes: Campus02WebService

- Version: 1.0

ToDoItems:

- Endpunkte:

- `/api/ToDoItems` (GET, POST)

- `/api/ToDoItems/{id}` (GET, PUT, DELETE)

- Datenmodell: `ToDoItem`

- Eigenschaften: `id` (integer, long), `description` (string, nullable), `dueDate` (string, nullable), `status` (string, nullable)

Users:

- Endpunkte:

- `/api/Users` (GET, POST)

- `/api/Users/{id}` (GET, PUT, DELETE)

- Datenmodell: `User`

- Eigenschaften: `id` (integer, long), `name` (string, nullable), `email` (string, nullable), `role` (string, nullable)

Vergleich mit IDL und WSDL:

IDL (Interface Description Language):

* IDL wird normalerweise in verteilten Systemen verwendet, um die Schnittstelle zwischen verschiedenen Komponenten oder Diensten zu beschreiben.
* In der OpenAPI-Spezifikation fungiert die Beschreibung der Pfade, Anfragemethoden und Datenmodelle als eine Art IDL für die RESTful API.

WSDL (Web Services Description Language):

* WSDL wird häufig in SOAP-basierten Webdiensten verwendet und beschreibt die Operationen, Nachrichten und Endpunkte.
* Die OpenAPI-Spezifikation enthält ähnliche Informationen, definiert jedoch die RESTful API mit den Ressourcen, Pfaden und HTTP-Methoden anstelle von Operationen und Nachrichten.

Insgesamt stellt die OpenAPI-Spezifikation umfassende Beschreibung bereit, die Entwicklern ermöglicht, effektiv mit dem Service zu arbeiten.

**Erstellen Sie eine Service-Klasse, welche per Dependency Injection aus Ihrem Service aufgerufen wird.  
10 Punkte**

Erstellen Sie eine Client-Applikation, welche Ihr erstelltes Service verwendet. Die Technologie bleibt dabei Ihnen überlassen (C#-Console-App, Java, JavaScript, Python,..)  
10 Punkte

**Beschreiben Sie das Thema Routing im Allgemeinen. Definieren Sie mindestesn eine Route, welche nicht dem ASP.NET Core-Standard entspricht  
10 Punkte**

**Allgemeines**

Routing ist ein entscheidender Aspekt in der Webentwicklung, der die Zuordnung von URLs zu spezifischen Ressourcen oder Funktionen in einer Anwendung ermöglicht. Es bestimmt, wie eine Anwendung auf eingehende HTTP-Anforderungen reagiert und welche Teile der Anwendung für bestimmte URLs zuständig sind.

**Komponenten des Routings**

URL-Pfad:

Der Teil der URL, der den Pfad zur Ressource innerhalb der Anwendung angibt. Zum Beispiel: /produkte/123

HTTP-Methoden:

Die HTTP-Methode (GET, POST, PUT, DELETE usw.), die in der Anforderung verwendet wird. Die gleiche URL kann unterschiedliche Aktionen basierend auf der verwendeten HTTP-Methode auslösen.

Parameter:

Teile der URL, die variable Werte repräsentieren. Zum Beispiel: /benutzer/{id} Hier könnte {id} ein Parameter sein.

**ASP.NET Core Routing**

In ASP.NET Core wird das Routing durch die Routing-Middleware behandelt. Standardmäßig wird das konventionelle Routing verwendet, bei dem der Controller, die Aktion und die Parameter auf Basis des URL-Pfads bestimmt werden.

// Beispiel für eine Route im Controller

[Route("produkte/{id}")]

public IActionResult GetProduct(int id)

{

// Logik zur Rückgabe des Produkts mit der angegebenen ID

}

Hier wird die Aktion GetProduct aufgerufen, wenn die URL /produkte/123 aufgerufen wird, wobei 123 als Wert für den Parameter id übergeben wird.

**Nicht dem Standard von ASP.NET Core entsprechende Route:**

Eine nicht dem Standard von ASP.NET Core entsprechende Route könnte ein benutzerdefiniertes Routing-Schema verwenden, das von den konventionellen Ansätzen abweicht.

// Beispiel für eine benutzerdefinierte Route

[Route("api/v1/products/{productCode}")]

public IActionResult GetProductByCode(string productCode)

{

// Logik zur Rückgabe des Produkts mit dem angegebenen Produktcode

}

Hier wird die Aktion GetProductByCode aufgerufen, wenn die URL /api/v1/products/ABC123 aufgerufen wird, wobei ABC123 als Wert für den Parameter productCode übergeben wird. Diese Route entspricht nicht dem Standard von ASP.NET Core, da sie ein benutzerdefiniertes Routing-Schema verwendet, das explizit die API-Version und den Produktcode im Pfad angibt.

Es ist wichtig zu beachten, dass benutzerdefinierte Routen sorgfältig dokumentiert und gepflegt werden sollten, um sicherzustellen, dass Entwickler und APIs sie verstehen und konsistent verwenden können.

**Verwenden Sie Einträge aus der „appsettings.json“   
10 Punkte**

**Aufbereitung und Präsentation inkl. Aufgabenbereich der einzelnen Gruppenmitgliedern und grober „eingesetzter Aufwand“ in Stunden pro Mitglied und Aufgabenbereich  
10 Punkte**

**Funktionierende Gesamtlösung  
10 Punkte**

**Beschreiben Sie anhand des Artikels „**[The Difference Between HTTP Auth, API Keys, and OAuth | Nordic APIs |](https://nordicapis.com/the-difference-between-http-auth-api-keys-and-oauth/)**“ das Einsatzgebiet und Vor- und Nachteile folgender Authentifizierungs und Autorisierungs-Möglichkeiten (ApiKey, http Basic Auth, OAuth 2.0**

**Sichern Sie mindestens ein Service mit einem „APIKey“ ab.  
10 Punkte**