Projekt Todo Liste

Olga Peshkova

Daniela Haumer

**GITHUB-Link: https://github.com/DanielaHaumer/WSLMicroService.git**

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:

Eindeutige 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.

HATEOAS:

Hier nicht vorhanden.

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 im 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)

Als Beispiel Auszug aus OpenAPI-Spec (GET mit id):

"/api/Campus02/{id}": {

"get": {

"tags": [

"ToDoItems"

],

"parameters": [

{

"name": "id",

"in": "path",

"required": true,

"schema": {

"type": "integer",

"format": "int64"

}

}

],

"responses": {

"200": {

"description": "Success",

"content": {

"text/plain": {

"schema": {

"$ref": "#/components/schemas/ToDoItem"

}

},

"application/json": {

"schema": {

"$ref": "#/components/schemas/ToDoItem"

}

},

"text/json": {

"schema": {

"$ref": "#/components/schemas/ToDoItem"

}

}

}

}

}

},

//…..

}

Auszug mit Datenmodell (“components“/ “schemas“) aus OpenAPI-Spec:

"ToDoItem": {

"type": "object",

"properties": {

"id": {

"type": "integer",

"format": "int64"

},

"description": {

"type": "string",

"nullable": true

},

"dueDate": {

"type": "string",

"nullable": true

},

"status": {

"type": "string",

"nullable": true

}

},

"additionalProperties": false

},

IDL (Interface Description Language):

* IDLs dienen dazu, die Schnittstelle von verteilten Systemen zu beschreiben. Sie werden oft in der Kommunikation zwischen verschiedenen Programmiersprachen oder Plattformen verwendet.
* Beispiel: CORBA (Common Object Request Broker Architecture) verwendet eine IDL, um die Schnittstelle zwischen Objekten in unterschiedlichen Sprachen zu definieren.

WSDL (Web Services Description Language):

* WSDL ist eine XML-basierte Beschreibungssprache, die verwendet wird, um Webdienste zu definieren und ihre Schnittstellen zu beschreiben. Es definiert die Methoden, Nachrichten, Datenstrukturen und Endpunkte, die von einem Webdienst unterstützt werden.
* Beispiel: Bei SOAP-basierten Webdiensten verwendet man WSDL, um die Struktur und den Endpunkt des Dienstes zu beschreiben.

OpenAPI-Spezifikation:

* OpenAPI ist ein modernes Werkzeug zur Beschreibung von RESTful Webdiensten. Es bietet eine maschinenlesbare Schnittstellenbeschreibung, die sowohl für Menschen als auch für Computer verständlich ist.
* OpenAPI verwendet JSON oder YAML, um die API-Spezifikation zu definieren.
* Beispiel: Wenn man eine RESTful API erstellt, kann man OpenAPI verwenden, um die Ressourcen, Endpunkte, Parameter, Antworten und andere relevante Informationen zu dokumentieren.

Insgesamt sind IDL, WSDL und die OpenAPI-Spezifikation alle Werkzeuge zur Beschreibung von Schnittstellen, aber sie haben unterschiedliche Anwendungen und Schwerpunkte. IDLs sind eher auf die Beschreibung von objektorientierten Schnittstellen fokussiert, während WSDL speziell für Webdienste entwickelt wurde und die OpenAPI-Spezifikation spezifisch für RESTful-APIs ist.

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

Siehe Code (User Controller -> IMyService)

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

Angular Client aus frühere LV angepasst

**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 ist {id} ein Parameter.

**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 verwendet ein benutzerdefiniertes Routing-Schema, 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.

Beispiel aus unsere Anwendung:  
Standard-Routing:

User Controller -> [Route("api/[controller]")]

Benutzerdefinierte Route:

ToDoItems Controller -> [Route("api/Campus02")]

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

Siehe Code:

"ProductSuffix": "Campus02"

* Wird in ToDoItemsController by POST-Methode angewendet

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

Aufgabenbereich: Code -> Olga Peshkova

Dokumentation -> Daniela Haumer

+ gemeinsame Durchsicht und Aufbereitung

Eingesetzter Aufwand: zusätzlich zu den LV Stunden, haben wir jeweils 3 Stunden zusätzlich zu Hause angewendet.

**Funktionierende Gesamtlösung  
10 Punkte**

Funktioniert mit vorhandenen Client

**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**

Wir haben einen Service mit einem API-key abgesichert. (Beispiel: Methode POST bei ToDoItems)

HTTP-Authentifizierung, API-Schlüssel (API Keys) und OAuth sind verschiedene Mechanismen für die Sicherung von APIs und Webdiensten. Hier sind die grundlegenden Unterschiede zwischen ihnen:

HTTP-Authentifizierung:

HTTP Basic Auth ist eine einfache Methode zur Erstellung einer Authentifizierung im Benutzername- und Passwortstil für HTTP-Anfragen. Dabei wird ein Header namens Authorization verwendet, der eine base64-codierte Darstellung von Benutzername und Passwort enthält. Je nach Anwendungsfall kann HTTP Basic Auth den Benutzer der Anwendung oder die Anwendung selbst authentifizieren.

**Vorteile:**

Einfach anzuwenden

**Nachteile:**

Anwendung speichert das Passwort des Benutzers.

Die Anwendung erhält vollen Zugriff auf das Konto, und es gibt keine andere Möglichkeit für den Benutzer, den Zugriff zu widerrufen, als das Passwort zu ändern.

Multi-Faktor-Authentifizierung ist nicht möglich.

API-Schlüssel (API Keys):

Die Verwendung von API-Schlüsseln ist eine Möglichkeit, eine Anwendung bei der Nutzung der API zu authentifizieren, ohne einen tatsächlichen Benutzer zu referenzieren. Die Anwendung fügt jedem API-Anforderung den Schlüssel hinzu, und die API kann den Schlüssel verwenden, um die Anwendung zu identifizieren und die Anforderung zu autorisieren. Der Schlüssel kann dann verwendet werden, um Dinge wie Begrenzung der Anfragenrate, Statistiken und ähnliche Aktionen auszuführen.

Wie der Schlüssel übermittelt wird, unterscheidet sich je nach API. Einige APIs verwenden Query-Parameter, einige verwenden den Authorize-Header, einige verwenden die Body-Parameter usw.

**Vorteile:**

Einfach anzuwenden

**Nachteile:**

Identifiziert nur die Anwendung, nicht den Benutzer der Anwendung.

Schwierig, den Schlüssel geheim zu halten.

Anforderungs-URLs können in Logs landen.

OAuth (Open Authorization):

Eine Token-basierte Architektur basiert darauf, dass alle Dienste einen Token erhalten, um zu beweisen, dass die Anwendung berechtigt ist, den Dienst aufzurufen. Der Token wird von einer vertrauenswürdigen Drittpartei ausgestellt, die von sowohl der Anwendung als auch dem Dienst vertraut wird. Das OAuth 2.0-Protokoll wird häufig verwendet, um diese Tokens zu erhalten.

OAuth löst die Nachteile der andere beiden Mechanismen, indem es einen Autorisierungsserver (AS) bereitstellt, der Tokens ausstellt und den Zugriff auf Dienste im Namen des Benutzers verwaltet. Der AS authentifiziert den Benutzer und stellt sicher, dass nur autorisierte Anwendungen auf seine Daten zugreifen können. Dadurch wird ein sichererer und kontrollierterer Zugriff auf Dienste ermöglicht.

**Vorteile:**

Anmeldeinformationen werden nur an vertrauenswürdige Seiten weitergegeben.

Multi-Faktor-Authentifizierung kann verwendet werden.

Der Benutzer kann den Zugriff für die App widerrufen ohne das Passwort zu ändern.

Der Benutzer kann der Drittanbieter-App nur den Zugriff auf bestimmte Informationen ermöglichen.

Informationen über den Benutzer können direkt über die Anfrage an den Dienst übermittelt werden.

Der Ablauf ist standardisiert.

**Nachteile:**

Lt. Artikel keine