Assignment

UML

Objektorientierte Software-Entwicklung, UML

Name, Vorname:	Dein, Name	Studiengang:	Informatik - Master of Science (M. Sc.)
Immatrikualtionsnr.:	234567		
Adresse:	Staße Nr. 01 12345 Ort	Modul	SWE24
		Abgabe am:	28. September 2024
E-Mail	email.adresse@provider.de	Dozent	Max Musterman

AKAD

Abstract

Hier wird das Management Summary leben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung in die Semesterarbeit

1.1 Ausgangslage

1.1.1 Umfeld

Die Isolutions AG ist ein IT-Service Dienstleister mit Fokus auf Microsoft Services und Individualentwicklung. Für die Umsetzung von Individualentwicklungen verwendet Isolutions AG in erster Linie:

- ASP.NET Web API
- C#
- Azure Services
- React, Blazor oder Angular
- SQL Server

Je nach Bedarf kommen auch andere Technologien zum Einsatz. Spezialisiert ist die Isolutions AG auf in Azure gehostete Webapplikationen.

1.1.2 Problemstellung

Zu Begin eines Projekts fällt in der Regel ein Aufwand von 3-5 Personentagen an, um die Entwicklungsumgebung aufzubauen. Die lokale Entwicklungsumgebung muss aufgesetzt werden. Das Beinhaltet unter umständen, dass lokale Datenbanken aufgebaut werden müssen, weitere Dienste installiert oder in Containern bereitgestellt werden. Weiter wird die Sourcecode Verwaltung in Azure Devops benötigt. Dazu gehören Build und Deployment Pipeline mit entsprechenden Tools zur Qualitätsicherung. Mindestens eine Stage muss ebenfalls aufgebaut werden, um das deployment zu testen und die Applikation in einer remote Umgebung zu testen.
Diese initialen Aufwände bedeuten weniger Budget für die Umsetzung von Funktionalität und führen zu einer längeren Time-to-Market. Um diesem Umstand entgegen zu wirken, wurde das so genannte Kickstarter Template entwickelt. Diese Vorlage enthält eine komplette Webapplikation mit wahlweise einem Blazor oder React Frontend, einer

Backend-For-Frontend ASP.Net WebApi und einer SQL Datenbank. Frontend und Backend enthalten UI und Logik um eine Entität zu verwalten. Der komplette CRUD Workflow für diese Entität ist implementiert. Inklusive Entity Framework Migration. Ausserdem sind Security, Logging, Telemetrie, Infrastructure-As-Code mit Terraform und Testing im Template enthalten. Das Template hat den Initialen Aufwand für das Projekt Setup auf einen Tag reduziert.
Mit Aspire .Net stellt ein Microsoft einen neue Technolgie Stack vor, welcher auf die Entwicklung von Cloud-Native Applikationen ausgerichtet ist. Die Frage stellt sich innerhalb von Isolutions, ab Aspire .Net in das Kickstarter Template integriert werden soll. Kann Aspire .Net das Template schlanker machen? Kann es insbesondere den IaC Teil des Templates ersetzen oder vereinfachen?

1.2 Ziele der Arbeit

Know How bezüglich Aspire .Net ist Isolutions weit nicht bekannt. Es gibt auch in der Community noch wenig Erfahrung mit dem Technologie Stack. Deshalb ist das Erreichen der Ziele ungewiss. Eine Anpassung der Zielsetzung kann während der Arbeit notwendig werden. Folgende Ziele sind vorgängig für diese Arbeit definiert worden:

1.2.1 Ziel 1: Analyse von Aspire .Net

Mit der Analyse von Aspire .Net soll herausgefunden werden, ob und welche Teile des Kickstarter Templates durch Aspire .Net verbessert werden kann. Die Analyse erfolgt in folgenden Schritten:

- Knowhow aufbau mit Aspire .Net Dokumentation und Tutorials
- 1:1 Integration von Aspire .Net in das Kickstarter Template

Mit der 1:1 Integration soll das Verständnis rund um die Funktionsweise von Aspire .Net und der Entwicklungsphilosophie vertieft werden.

1.2.2 Ziel 2: Aufbau eines neuen Templates mit Aspire .Net

Basierend auf den Erkenntnissen aus der Analyse soll ein neues Template aufgebaut werden. Dieses Template soll die gleichen Funktionalitäten wie das Kickstarter Template

enthalten, legt jedoch den Fokus auf Azure Cloud Services. Das neue Template soll folgende Funktionalitäten enthalten:

- Infrastructure wird durch Aspire .Net provisioniert
- Deployment der Aspire .Net Applikation via automatisierter Pipeling
- Lokale Entwicklung funktioniert offline mit allen referenzierten Services
- Weitere Azure Services können einfach zum Projekt hinzugefügt werden

Das neue Template soll den Fokus insbesondere auf die Auslagerung von Workloads in dafür vorgesehene Azure Services legen.

1.3 Vorgehen und Methodik

Da es unklar ist ob die Ziele erreicht werden können, primär wegen fehlendem Know How, wird die Arbeit iterativ durchgeführt. Geplant sind 3 Iterationen:

- Iteration 1: Know-How Aufbau
- Iteration 2: Umbau des Kickstarter Templates
- Iteration 3: Aufbau eines Aspire .Net Kickstarter Template

2 Analyse von Aspire .Net

2.1 Was ist Aspire .NET

Aspire .Net ist ein Technologie Stack und deckt deshalb verschiedene Bereiche ab. Hier soll eine Übersicht über die verschiedenen Bereiche gegeben werden, ohne den Anspruch, dass die Übersicht vollständig ist.

2.1.1 Orchestration

Dieser Bereich fokusiert primär die lokale Entwickler Experience. Aspire .Net Vereinfacht die Verwaltung von Services und deren Verbindung untereinander. Erreicht wird dies durch eine Abstraktion der Service Konfiguration, Umgebungsvariablen und Container Konfiguration. Der Entwickler muss sich nicht um die Konfiguration eines Connection Strings zu einer Datenbank kümmern. Aspire .Net übernimmt diese Aufgabe. Die Orchestrierung beinhaltet die Komposition von individuellen Backend Serivces, Frontends, Datenbanken, Caching und Azure native Serivce wie Azure Service Bus, usw.

2.1.2 Integration

Die Integration von Services löst Aspire .Net in dem es die Services in Nuget Packete kapselt. Jedes dieser Packete ist dafür ausgelegt mit der Aspire .Net Orchestrierung zusammen zu arbeiten. Jede Service Integration besteht aus zwei Nuget Packeten. Ein Packet hostet den Service wärend das andere Packet die konsumierende Seite darstellt. Dieser Client stellt die Verbindung zum Host her und registriert Clientservices im Dependency Injection Container vom konsumierenden Service.

2.1.3 Templates und Tooling

Aspire .Net ist um ein standartisiertes Design aufgebaut. Dieses besteht aus:

- AppHost: Dies ist ein Projekt, welches die Orchestrierung der Services übernimmt.
- ServiceDefaults: Dieses Projekt enthält die Standartkonfigurationen für Aspire .Net Projekte und kann um individuelle Konfigurationen erweitert werden. Die

Konfiguration dreht sich unter anderem um Health Checks, Telemetrie und weiters.

- .Net Projekte: Weitere Projekte enthalten indivudelle .Net Projekte wie Backend Services oder Frontendcode.

2.2 Wie wird Aspire .NET verwendet

Beschreiben von Use-Cases von Aspire .Net

- Code Beispiele
- Verschiedene Services
- Telemetrie

3 Implementierung Aspire .NET Kickstartertemplate

In dieser Sektion werden die Arbeiten und Ergebnisse in den Iterationen beschreiben.

3.1 Iteration 1: Know-How Aufbau

Das gewonnene Know-How über Aspire .Net ist im Analyse Abschnitt dieses Berichtes beschrieben. In diesem Abschnitt wird für die arbeit bessonders notwendiges Know-How dokumentiert.
Ein Aspire .Net Projekt besteht, wie bereits beschreiben, aus mindestens drei Projekten. Dem AppHost, ServiceDefaults und einem Service mit individueller Applikationslogik. AppHost und ServiceDefaults sollen hier genauer beschreiben werden.

3.1.1 AppHost

Das AppHost Projekt ist das Hauptprojekt einer Aspire .Net Applikation. Es ist das Startprojekt einer Solution und beinhaltet die Servicekomposition.

```
var postgres = builder.AddPostgres("postgres").PublishAsAzurePo
var postgresdb = postgres.AddDatabase("postgresdb");

var api = builder.AddProject<MinimalApi>("messagesapi")
    .WithReference(postgresdb)
    .WithExternalHttpEndpoints();
```

3.1.2 AppHost

3.2 Iteration 1: Erkenntisse

3.3 Iteration 2: Umbau des Kickstarter Templates

3.4 Iteration 2: Erkenntnisse

3.5 Iteration 3: Aufbau eines neuen Protoypes

3.6 Iteration 3: Erkenntnisse

4 Ergebnis

4.1 Architektur des Prototypes

4.2 Funktion

4.3 Entwicklung und Deployment

5 Fazit

5.1 Learnings

5.2 Persönliche Betrachtung

5.3 Ausblicke

Anhang

A Shared-Memory-API

B Fehleruntersuchung