**OSP-Abgabe Technischer Leiter**

**Grundlegendes**

Prinzipien:

* Schichtenarchitektur
* Single Responsibility Principle
* Modularisierung
* Dependency Injection
* Vererbung (In Angular z.B. Base Store)
* API
* Authentication
* Observer Pattern (In Angular)

**Methoden, Werkzeuge, Techniken:**

* Dependency Injection über C# StartupEin Bild, das Text, Screenshot, Schrift enthält.

  Automatisch generierte Beschreibung

Die Klassen werden einmal erzeugt und in den Service gespeichert, dann bei der Instanziierung der Bll-Klassen werden die Instanzen der Factories injectet. Dadurch wird nur eine Instanz erzeugt und es hat keine Direkte Abhängigkeit.

* Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

  Automatisch generierte BeschreibungSchichtenarchitektur durch verschiedene C# Projekte und ein Client (Angular-Projekt)
* Modularisierung und Single Responsibility Principle werden durch getrennte Dateien umgesetzt.  
    
  Ein Bild, das Text, Screenshot, Schrift, Software enthält.

  Automatisch generierte Beschreibung   
  Z.b. wie hier: Die DbCommands sind für jedes Objekt getrennt, damit man schnell potenzielle Fehlerquellen findet und keine Überladenen Klassen hat.
* Vererbung wird in Angular als Form einer Basisklasse genutzt.  
  Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

  Automatisch generierte Beschreibung  
  Die abstrakte Klasse BaseStoreService wird von anderen Klassen erweitert und ihr wird eine Klasse (hier T) übergeben, damit die Base-Funktionen den richtigen Typ zurückgeben. Außerdem gibt der Service vor, welche Funktionen implementiert werden müssen.
* Die API wird mithilfe von dem MVC-Controller Pattern von Microsoft umgesetzt.  
  Ein Bild, das Text, Screenshot, Software, Schrift enthält.

  Automatisch generierte Beschreibung  
  Der Controller wird von dem bereitgestellten ControllerBase abgeleitet und implementiert mithilfe der Decoratoren die Funktionen bzw. die Endpunkte. Die Route des Controllers ist der Name ohne den Controllerpart (also hier Article).

* Die Authentication wird durch das von Microsoft bereitgestellten  
  „Microsoft.AspNetCore.Authorization;“-Paket umgesetzt. Jeder abgesicherte Controller wird mit dem [Auhorize]-Decorator dekoriert, damit man nur mit einem gültigen Token den Endpunkt erreicht. (Siehe 1. Screenshot davor)

In der Startup-Datei wird die Authentication implementiert:

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

Automatisch generierte Beschreibung  
Aus der app.settings.json werden die Issuer geholt und in den Parametern gespeichert. Bei erfolgreichem Login wird ein Token mit diesem Schema erstellt und an das Frontend zurückgegeben.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Es werden verschiedene „Claims“ in das Token geschrieben (bei uns UserId, UserMail, UserRole), damit wir diese im Frontend auslesen und verwerten können. Unsere Tokens sind immer 120 Minuten gültig, danach muss man sich erneut einloggen.

* Das Observer Pattern wurde mithilfe von Behaviour-Subjects umgesetzt. Bei Veränderung der Werte werden alle Subscriber über die Veränderung benachrichtigt.   
  Ein Bild, das Text, Screenshot, Schrift, Software enthält.

  Automatisch generierte Beschreibung
* Async/Await  
  Ein Bild, das Text, Screenshot, Software enthält.

  Automatisch generierte Beschreibung  
  Außerdem nutzen wir async/await um unsere verschiedenen Aufrufe asynchron zu machen. Dadurch warten wir auf das Ergebnis, um Fehler durch gleichzeitige Operationen zu vermeiden.

**Modularisierung**

**Frameworks, Patterns**  
Wie oben schon beschrieben haben wir die Patterns MVC, Dependency Injection und das Observer Pattern genutzt.

* Als Framework haben wir clientseitig Angular mit dem UI-Framework (Angular-) Material genutzt
* Als CSS-Framework haben wir Tailwind genutzt
* Und serverseitig haben wir ASP.Net Core genutzt

**Struktur der Projekte:**

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungC#:

Typescript (Angular)

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung  
**Features**: Die kleinsten Einheiten an Code (z.B. das Projektroulette zum Anzeigen der Projekte)

**Framework**: Alle erstellten Klassen, die so allgemein sind, dass man sie auch in deren Projekten nutzen könnte (z.B. eine Toolbar)

**Models**: Alle Interfaces und eine enum-Datei

**Pages**: Alle ansteuerbaren Seiten, dort werden die Features zu einer Seite zusammengeschnürt

**Services**: Angular-Service Klassen, die einfach in andere Klassen importiert werden können (Sind einfacher zu injecten, da sie nur eine Instanz haben und so geteilt genutzt werden können)

**AppComponent**: Unser entryPoint, dort sind die Framework Komponenten eingebunden und das RouterOutlet (dort wo die angerouteten Seiten angezeigt werden)

**Visualisierung** (mit hier im Ordner)

**UML**

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

Automatisch generierte Beschreibung

Abbildung 1- Klassendiagramm Bll

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

Automatisch generierte Beschreibung

Abbildung 2 - Klassendiagramm Dal

**Andere Formate**

Ein Bild, das Text, Diagramm, parallel, Quittung enthält.

Automatisch generierte Beschreibung

Abbildung 3 – Krähenfuß

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

Automatisch generierte Beschreibung

Abbildung 4 - Schichtenmodell

**Sonstige Planung vom Anfang**

**Technische Umsetzung**

1. **Datenbank**

SQ-lite Datenbank

1. **ORM**

Als ORM wird Entity Framework genutzt

1. **Erreichbarkeit** 
   1. Die Datenbank wird über ein C#-Projekt erreichbar sein. Dabei wird eine API mithilfe des „Microsoft.AspNetCore.Mvc“-Pakets erstellt und über Visual Studio gehostet.
   2. Die Controller sollen mithilfe des „Microsoft.AspNetCore.Authorization“-Paketes abgesichert werden, damit nicht jeder Zugriff auf die Datenbank hat.
2. **Frontend**

Das Frontend wird ein Angular Projekt, das bedeutet es wird nodeJs, npm-package benötigt. Als UI-Styling wird Angular Material verwendet. Das Projekt soll nach moderner Angular Struktur mit Standalone-Komponenten aufgebaut werden. Zusätzlich wird TailwindCss als Css Framework genutzt.