Architekt

Grundlegendes

Prinzipien:

* Schichtenarchitektur
* Single Responsibilty Principle
* Modularisierung
* Dependency Injection
* Vererbung (Angularseitig z.B. BaseStore)
* API
* Authentication

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.

* Schichtenarchitektur durch verschiedene C# Projekte und ein Client (Angular-Projekt) Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

  Automatisch generierte Beschreibung
* 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 Angularseitig in 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 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, Schrift, Software 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 die von Microsoft bereitgestellten „Microsoft.AspNetCore.Authorization;“ Paket umgesetzt. Jeder abgesicherte Controller wird mit dem [Auhorize]-Decorator dekoriert, damit man nur mit einem gültigen Token rein kommt.  
  In der Startup wird diese 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.

Modularisierung

Frameworks, Patterns  
 Wie oben schon beschrieben haben wir die Patterns MVC, Dependency Injection 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:

C#:

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

  Automatisch generierte Beschreibung

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

UML, Andere Formate