**GIT-Agreements**

Im folgenden Dokument geben wir die Richtlinien zum Nutzen von GIT an.

**Nutzung von Fork**

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

Automatisch generierte Beschreibung

Abbildung 1 Fork Oberfläche

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungAls Programm zu arbeiten im GIT verwenden wir die Software Fork. Dieses bietet GIT Flow an, womit sich Features erstellen und schließen lassen. Unteranderem bietet GIT Flow eine Bearbeitungsstruktur an, dieses beinhaltet einen develop und einen feature Ordner (Rote Pfeile). Im Programm Fork gibt es immer einen Lokalen (blau) und einen Remote Anteil (grüner Pfeil). Der Lokale Anteil ist, bis man hochspielt nur auf dem Computer verfügbar. Nach einem „Push“ werden Datei Änderungen auf den Remote Spiegel gebracht und andere im Team können diese abholen, um damit arbeiten zu können.

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

Automatisch generierte BeschreibungWenn man ein Feature starten möchte, macht man dies mit Git Flow. Mit einem Rechtsklick auf den gewünschten stand z.B. dem develop, geht man unter den Reiter Git Flow Start Feature und gibt dem Feature einen Namen.

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

Automatisch generierte BeschreibungWenn man ein Feature abgeschlossen hat und dieses Zurückführen möchte, geht man mit Rechtsklick auf Git Flow und auf „Finish Feature“. Dann wird das Feature auf den Ursprünglichen „Branch“ zurückgeführt. Sollten in der Zeit Änderungen auf dem Ursprünglichen „Branch“ vollzogen worden sein und gleiche Dateien bearbeitet sein, entsteht ein „merge“ Konflikt, der gelöst werden muss, bevor man das Feature abschließen kann.

**Review Prozess innerhalb von GIT**

Wir haben uns dazu entschieden Arbeitspakete als Namen von Features zu Nutzen. Dazu wird folgende Formatierung genommen „Arbeitspaket Nummer“ – „Arbeitspaket Name“ also z.B. 4300-Ui-Ux-entwickeln. Dabei ist zu beachten das man statt Leertaste ein Minus nimmt und keine Slashes einbaut da sonst ein Unterordner entsteht. Sollte man ein Feature seines Erachtens fertig gestellt haben meldet man bei einem Team Mitglied ein Review an. Dieses Beinhaltet vor dem zurückführen ein vier Augen Prinzip, d.h. es gibt einen Developer und einen Reviewer der über den Code drüber guckt. Ein Review sollte Code Verbesserung Anregungen beinhalten (best-practice) und auf Richtigkeit prüfen. Zum Abschluss führt der Reviewer nach dem er den Programm Code gesichtet hat, einen Funktionstest aus, indem er das Programm startet und die implementierte Funktion testet. Sollten diese Schritte abgeschlossen sein, wird das Feature auf den develop zurückgeführt.

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

Automatisch generierte BeschreibungCode Guidelines**

Um zu gewährleisten das der Code für alle Einheitlich Formatiert wird, haben wir uns im Back-End (C#) und im Front-End (Angular) für eine vorgefertigte Konfiguration entschieden.

Im Front-End Teil wird das mit einer settings.json Datei angeben diese zieht die tabSize, stickyTabStops und das die default Konsole ein Command Prompt ist und nicht eine Powershell. Die tabSize gibt hierbei an, dass, wenn man eine neue Zeile angibt, diese Korrekt eingerückt wird. Die stickyTabStop Funktion bewirkt, dass VS Code Cursorbewegungen in führenden Leerzeichen ähnlich wie Tabulatoren behandelt.

Darüber hinaus, um gleichen Code zu produzieren haben wir uns auf folgende Strukturen geeinigt:

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

Automatisch generierte BeschreibungVier Dateien

1. \*.html für html Teil
2. \*.scss für den Style Teil
3. \*.ts für den TypeScript Teil
4. \*.spec.ts für den Test Teil

Diese Dateien werden automatisch mit dem Befehl „*ng generate component [name-der-Komponente]“* erstellt

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

Automatisch generierte Beschreibung\*.ts Dateien:

1

1. Zuerst alle Imports

3

2

1. Decorator der Klasse

a

1. Klassen Definition mit:
   1. Parent Komponente (Input): Zur Objekte Initialisierung von außen

b

* 1. Private Variablen: Namen beginnen mit „\_“

c

* 1. Konstruktor
  2. Public Methoden

d

* 1. Private Methoden

e

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

Automatisch generierte Beschreibung\*.html Datei:

1

1. Section
2. Content

2

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

Automatisch generierte Beschreibung

\*.scss Datei:

beinhaltet die einzelnen Styles zwischen den einzelnen Styles wird eine Zeile frei gelassen

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

\*spec.ts Datei:

1

Beinhaltet die Tests der einzelnen component

2

1. Zuerst alle imports
2. Erstellung der Testumgebung bzw. der component

3

1. Das Setup was vor jedem Test ausgeführt wird
2. Inhalt des Tests
3. Die Prüfung

4

5

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

Automatisch generierte BeschreibungIm Back-End nutzen wir eine Visual Studio Extension, die für uns die Code Guidelines übernimmt. Diese Extension nennt sich CodeMaid. Dieses Überprüft bei jedem Speichern die Guidelines und setzt diese durch. Unteranderem überprüft diese auch die Namensgebung, ob diese Pascal Case sind und gibt Warnungen aus wenn dies nicht der Fall ist.

Ein paar Funktionen die CodeMaid übernimmt:

* Einrückung von Code
* Entfernen von nicht benutzten usings
* Sortieren von Klassen nach privaten Variablen, Konstruktor, Public Methoden und Private Methoden

**Teststrategien**

In diesem Abschnitt gehe ich kurz auf die Teststrategien ein, die wir in C# und Angular angewendet haben.

Im Backend (C#) haben wir uns dazu entschieden die Logik Schicht zu testen. Diese Schicht wird bei uns im Programm als **BLL** (**B**usiness **L**ogik **L**ayer) bezeichnet. Dort ist der Kern unserer API, diese Kommuniziert zwischen den Controllern und dem **DAL** (**D**ata **A**cces **L**ayer). Aus diesem Grund haben wir uns entschieden nur die BL Schicht zu testen. Diese Tests haben eine eigene Testdatenbank, die uns hilft, ohne die Tatsächliche Datenbank, Einträge zu verändern und die Datenbank Zugriffe zu Simulieren. Bei den Tests haben wir uns dafür entschieden einen Workflow Test durzuführen, das heißt wir bilden einen kompletten Workflow ab.

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

Automatisch generierte BeschreibungEin Bild, das Text, Schrift, Screenshot, Zahl enthält.

Automatisch generierte BeschreibungAls Framework für die Tests haben wir uns für FluentAssertions und NUnit entschieden. NUnit ist hierbei das Framework, womit man Testet, damit z.B. man Klassen bzw. Methoden als Test Markieren kann und diese von Visual Studios Test Explorer auch als Test erkannt wird. FluentAssertions wird verwendet, um die Ergebnisse am Tatsächlichen Objekt abzufragen und es mit einem erwarteten Wert zu vergleichen.

NUnit

Als Beispiel wird hier eine Transportbox hinzugefügt und geprüft, ob es hinzugefügt wurde. Diese Transportbox wird dann verändert und geupdatet. Nach einer Prüfung wird die Box wieder aus der Datenbank gelöscht. Dieser Test deckt nun den DAL und die BLL ab und somit unsere Logik und den Datenbank Zugriff.

FulentAssertions

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

Automatisch generierte Beschreibung

Im Frontend (Angular) haben wir uns stattdessen nur für Tests entschieden die Logik von components testen. Dazu haben wir das Jasmine Framework verwendet. Dieses Framework bietet so wie FluentAssertions eine Vergleichs Option an und erlaubt es \*.ts Dateien zu testen. Beispielsweise werden Sortierungen, Erstellung der Components, Validierungen und Formatierungen getestet. Der Befehl, um das Jasmine Framework zu starten ist „*ng t“.* Daraufhin bekommt man die im rechten Bild zusehende Ausgabe im Browser dargestellt und man sieht ob irgendwelche Probleme bei den Tests aufgetaucht sind.