Dokumentationskonzept

Erstellt von:	**Verifiziert von:**
• Matti Bos	• Fabian Hamacher

Version	Erstellung	Beschreibung
0.1	31.03.2025	Erstellung Konzept
1.0	01.04.2025	Ausführliche Beschreibungen
1.1	02.04.2025	1. Korrektur
2.0	03.04.2025	Einarbeitung Feedback

Inhalt

Qualitätskriterien

Dokumente

Für jeden Dokumententyp gibt es eine Vorlage, die zu verwenden ist. Über verschiedene Arbeiten hinweg entsteht so ein zusammenhängendes Gesamtbild. Ebenfalls nimmt die Vorlage repetitive Aufgaben wie das Einstellen von Formatierungsoptionen vorweg.

Jedes Text-Dokument enthält eine deskriptive Kopf- und Fußzeile, in der Informationen zum gesamten Projekt und dem jeweiligen Dokument enthalten sind. In der Kopfzeile finden sich der Name des Projekt-Teams sowie die Zugehörige Vorlesung und ihr Zeitraum. Auf der rechten Seite ist das Logo positioniert. Die Fußzeile besteht aus dem Titel des Dokuments, seinem Erstellungsdatum und der Seitenzahl. So kann innerhalb eines Berichts besser navigiert. Auch die Übersicht zwischen mehreren Dokumenten wird dadurch erleichtert.

Code

Korrekte Funktionsweise ist eine elementare Eigenschaft, die in jedem Softwareprodukt zu finden sein sollte. Auch ein gewisser Komfort bei der Benutzung wird in aller Regel erwartet.

In diesem Projekt sind unsere Kunden allerdings nicht die Endbenutzer der Anwendung. Stattdessen wird das Produkt zum Ende des Projekts vollkommen an den Kunde übergeben. So kann dieser es mit eigenen Mitteln weiter entwickeln und warten.

Bei diesem Übergang geht zwangsläufig Know-How verloren, da die bei der Entwicklung entstandenen Gedanken der ursprünglichen Entwickler nicht verlustfrei überliefert werden können. Um dem entgegenzuwirken ist eine gut verständliche und ausführliche Dokumentation des Quellcodes unerlässlich. Ebenfalls muss der Code selbst mit besonderem Augenmerk auf Wartbarkeit und Erweiterbarkeit gestaltet werden. Aus diesem Grund wird eine Designstudie durchgeführt, um ein solides Fundament schaffen zu können.

Effizienz einzelner Algorithmen und Operationen kann im Nachhinein leichter verbessert werden als grundlegende architektonische Entscheidungen. Des Weiteren können Erweiterbarkeit und Modularisierung nur erschwert mit automatisierten Kontrollen überprüft werden. Die Funktionalität hingegen kann durch Tests nach jeder Änderung stets gewährleistet werden. Wir verwenden hierzu Jest, Swagger und xUnit. Die detaillierte Beschreibung des Test-Konzepts wird in einem dedizierten Dokument separat festgehalten.

Code-Conventions

Auch innerhalb unseres Teams sind die Erfahrungen in der Entwicklung unterschiedlich. Um trotzdem Code zu schreiben der an jeder Stelle von jedem verstanden werden kann, werden bestimmte Konventionen angewandt. Diese werden auch international verwendet, um die Kompatibilität von Modulen und Teammitgliedern zu erhöhen. Wir werden uns ebenfalls an diesen global verwendeten Standards ausrichten.

In jeder Programmiersprache gibt es Unterschiede zwischen den Standards, eine Übersicht zu Verschriftlichungen derselben findet sich im Folgenden:

- HTML https://google.github.io/styleguide/htmlcssguide.html
- Typescript https://google.github.io/styleguide/tsguide.html

- C# https://google.github.io/styleguide/csharp-style.html

Diese Sammlung an Regeln ist deutlich umfangreicher als es sinnvoll ist vollkommen zu überprüfen. Ihre Einhaltung ist darum explizit nicht in jeder Zeile auf jede Unterregel zu kontrollieren. Trotz dessen dienen sie als gute Richtlinien an denen sich orientiert werden soll.

Immer eingehalten werden sollen jedoch die folgenden Vorgaben:

- Aussagekräftige Benennungen, keine Abkürzungen
- Schreibweise der Namen
  - Dateien/Ordner Pascal-Case
  - Klassen Pascal-Case
  - Methoden Pascal-Case
  - Variablen Camel-Case
- Eine Codeebene pro Zeile
  - Einrückung mit Leerzeichen

Besonders die Handhabung von Einrückung ist bei den verwendeten React-Komponenten wichtig, da ihre Strukturen tiefe Verschachtelungen erreichen können.

Erstellung und Review

Kein Inhalt wird gänzlich von einer einzelnen Person erstellt und geprüft. Bei Dokumenten ist auf dem Titelblatt hierfür schriftlich Autor und Prüfer festgehalten. So ist gesichert, dass immer mindestens vier Augen einer Abgabe in Form und Inhalt zustimmen. Ebenfalls enthalten ist eine kleine Liste an Versionen, durch die das Dokument während seiner Erstellung verlaufen ist. Damit ist die Entstehung nachvollziehbarer.

Dokumente

Schriftliche Dokumente werden primär von ein bis zwei Haupt-autoren erstellt. Ziel ist selbst ein abgabefähiges Produkt zu erstellen. Beim Review wird die Einhaltung der Standards und die Qualität geprüft und gegebenenfalls korrigiert. Reviewer ist hier entweder der Projektleiter oder der Verantwortliche für Dokumentation. Der Projektleiter sollte jedoch nur sekundär als Reviewer tätig sein, falls das Dokument sonst von derselben Person geschrieben und abgenommen wird.

Code

Code soll regelmäßig kontrolliert werden. Jede Veränderung der Funktion muss von mindestens einer weiteren Person, als dem Autor selbst begutachtet werden. Um die Einhaltung leichter zu gewährleisten, werden neue Erweiterungen nur auf einem separaten Branch des Repositories erstellt und später per „Pull-Request“ mit dem aktuellen Stand zusammengeführt. Bei der Kontrolle der Anfrage wird auf die folgenden Punkte gesondert geachtet:

- Fehler in der Logik
- Existenz verständlicher Beschreibungen in Form von Dokumentationskommentaren
- Code-Style (Beachtung der Code-Conventions)
- Abdeckung durch entsprechende Tests

Ebenfalls ist es empfohlen im Rahmen des „Pair-Programming“ zu mehreren Personen an einem Stück Code zu arbeiten. Durch die Abstimmung untereinander werden Probleme mit den oben genannten Vorgaben schneller erkannt. So wird die Qualität des gesamten Produkts gesteigert, da der zweiten Person Fehler bereits während ihrer Entstehung auffallen können.

Auch bei „Pair-Programming“ entstandener Code muss durch ein Code-Review geprüft werden. Dabei darf der Reviewer kein Teil des Paares an Autoren sein.

Qualität und Korrektheit des Codes können teilweise automatisch getestet werden. Dazu verwenden wir zum einen die entsprechenden Tools innerhalb der IDEs, als auch verschiedene Software-Tests.

Dokumentation des Quellcodes

Zur Dokumentation des Quellcodes selbst werden Tools eingesetzt, die automatisch Beschreibungen aus dem Code und den enthaltenen Kommentaren generieren können. Dabei werden nicht die Beschreibungen selbst erklärt, sondern die Dokumentationskommentare extrahiert und in Form einer HTML-Seite dargestellt. Die Kommentare sollen parallel mit der Funktionalität selbst entwickelt werden.

Für den Code im Front-end verwenden wir **TypeDoc**. Der C# Code aus dem Back-end wird durch **Docfx** verarbeitet. Beide Tools sind speziell auf die jeweilige Programmiersprache ausgelegt und können durch besondere Formatierung innerhalb der Code-Kommentare unterstützt werden.

Genauere Informationen und Hinweise zur Verwendung finden sich auf den jeweiligen Homepages der Tools:

- TypeDoc Overview | TypeDoc
- Docfx Quick Start | docfx

Zur vollständigen Dokumentation werden zu jeder Klasse und jedem Interface ein Kommentar verfasst, der immer mindestens die folgenden Informationen enthält:

- Kurze Beschreibung
  Hier sollte mehr auf die Benutzung und Existenzbegründung eingegangen werden, als auf die exakte Implementierung um die Funktion zu bieten

Funktionen und Methoden enthalten noch zusätzliche Informationen:

- Eingabewert
  Bei sehr abstrakten Parametern ist ein Hinweis auf übliche Werte sinnvoll
- Ausgabewert

Bei Funktionen, die keine Ein- oder Ausgabewerte besitzen, muss dies explizit angegeben werden. Besonders bei polymorphen Varianten ist dies nützlich, um Fehler vorzubeugen.

Eine C# Klasse könnte also wie folgt dokumentiert sein:

```
/// <summary>
/// Die Calculator-Klasse bietet grundlegende mathematische Funktionen.
/// Diese Klasse soll zur Berechnung einfacher Arithmetik verwendet werden.
/// </summary>
public class Calculator
{
    /// <summary>
    /// Speichert das letzte Berechnungsergebnis.
    /// </summary>
    private double lastResult;

    /// <summary>
    /// Addiert zwei Zahlen und gibt das Ergebnis zurück.
    /// </summary>
    /// <param name="a">Die erste Zahl, die addiert werden soll.</param>
    /// <param name="b">Die zweite Zahl, die addiert werden soll.</param>
    /// <returns>Die Summe der beiden Zahlen.</returns>
    public double Add(double a, double b)
    {
        lastResult = a + b;
        return lastResult;
    }
}
```

Ein Beispiel für Typescript zeigt große Ähnlichkeit:

```
/**
 * Die Calculator-Klasse bietet grundlegende mathematische Funktionen.
 * Diese Klasse soll zur Berechnung einfacher Arithmetik verwendet werden.
 */
export class Calculator {
  /**
   * Speichert das letzte Berechnungsergebnis.
   */
  private lastResult: number = 0;

  /**
   * Addiert zwei Zahlen und gibt das Ergebnis zurück.
   *
   * @param a - Die erste Zahl, die addiert werden soll.
   * @param b - Die zweite Zahl, die addiert werden soll.
   * @returns Die Summe der beiden Zahlen.
   */
  public add(a: number, b: number): number {
    this.lastResult = a + b;
    return this.lastResult;
  }
}
```

GIT, ein Tool zur Verwaltung und Versionierung des Codes, verlangt zu jeder Änderung eine „Commit-Message“. Dies ist meist ein Kommentar, der die Änderung kurz beschreibt. Diese Commit-Messages sollen stets aussagekräftig sein, wobei hier ausdrücklich nicht gefordert ist, dass sie allein zum vollständigen Verständnis ausreichen.

Im Rahmen der API-Dokumentation mit Swagger werden auch die Statuscodes umfassend dargestellt und erläutert. Diese enthalten Informationen über den Erfolg oder Misserfolg einer Anfrage, sodass verschiedene Antworten klar erkennbar sind. Swagger ermöglicht es, für jeden API-Endpunkt die möglichen Statuscodes zu definieren und zu dokumentieren. Diese präzise Dokumentation hilft dabei, Fehler schnell zu identifizieren und die API effizient zu nutzen, wodurch die Integration in andere Systeme vereinfacht wird.

Verantwortlichkeiten

Aufgabe	Zuständig	Kommentar
Erstellung schriftliche Dokumente	Themenleiter	Falls die Zuständigkeit nicht eindeutig ist, delegiert der Projektleiter.
Erstellung Code	Verantwortlicher für Implementierung verteilt Aufträge	
Code-Dokumentation	Autor des zugehörigen Codes	Dokumentation des Codes kann, zusammen mit Tests, vor seiner eigentlichen Entwicklung geschrieben werden

Tool-Übersicht

Erstellung Dokumente

Wir setzen die Apps des **Microsoft Office** Pakets ein. **Word** und **PowerPoint** lassen sich speziell nennen, sie können zur Gestaltung von Textdokumente wie diesem und Präsentationen genutzt werden.

Allgemeine Dateiablage

Verwendet wird **Microsoft Teams**, die Plattform bietet eine zentrale Möglichkeit um auch große Dateien strukturiert zu speichern. Hier werden alle Dokumente oder Dateien abgelegt, die nicht Teil des Quellcode der Applikation sind.

Projektmanagement

YouTrack ist eine Applikation in der verschiedene Aufgaben geplant, delegiert und kontinuierlich verwaltet werden können. Es können Abhängigkeiten zwischen verschiedenen Aufgaben veranschaulicht und deren Fortschritt gemessen werden.

Entwicklungsumgebung

Visual Studio Code und **IntelliJ** werden eingesetzt, um den Code selbst zu entwickeln. In beiden Umgebungen gibt es entsprechende Erweiterungen, die mit Syntax-Highlighting und Problemerkennung helfen.

Quellcode

Der Code selbst wird mit **GIT** verwaltet und bei **GitHub** gehostet. Besonders die Verwaltung unterschiedlicher Versionen ist bei der Entwicklung einer Software-Applikation von hoher Wichtigkeit. Diese Verwaltung ist eine Kernfunktion der eben genannten Tools.

Implementierung

Organisation kleiner Aufgaben wie das Verteilen einzelner Features oder Issues werden direkt bei GitHub über **GitHub-Projects** verwaltet.

Doku-Tools

TypeDoc und **Docfx** für Typescript und C# Code. Für die API selbst wird **Swagger** eingesetzt. Alle drei sind Industriestandards, ihre Verwendung ist nach der initialen Konfiguration einfach und kann teilweise automatisiert werden.

Tests

Für Front-end, API und Back-end werden jeweils unter anderem, aber nicht ausschließlich **Jest**, **Postman** und **xUnit** verwendet. Diese Tools sind weit verbreitet und von aktiven Communities umgeben.