**DOKUMENT VOR ABGABE NOCHMAL PRÜFEN, DA VOR ENTWICKLUNG DER ANWENDUNG GESCHRIEBEN!!**

**Bei der anschließenden Implementierung der Anwendung sollten unbedingt die bisher geplanten Anforderungen sowie Spezifikation und Architektur berücksichtigt werden. Weiterhin ist auf eine gute Code-Qualität zu achten und der Quellcode sollte angemessen und verständlich kommentiert werden.**

**Phase 2: Erarbeitungs- und Reflexionsphase - Architekturdokument**

1. **Technologieübersicht**

**1.1 Programmiersprache**

Die für dieses Projekt ausgewählte Programmiersprache ist C++, folgend werden die wichtigsten Gründe für die Auswahl erläutert.

* Motivation: Vertiefung der eigenen Kenntnisse, wichtig für Ingenieurs- und Software-Engineering Berufe
* Fördert allgemeines Verständnis für Speicherverwaltung, effiziente Algorithmen und systemnahe Entwicklung
* Große Ökosystem, viele Libraries und plattformübergreifend einsetzbar
* Auch wenn Entwicklung schwerer als bspw. mit Python ist, ist der Lerneffekt für Studium/Beruf höher
* GUI-Anbindung mit Qt

**1.2 Frameworks**

In diesem Projekt werden primär die Frameworks Qt (Qt6) und googletest verwendet. In diesem Abschnitt werden die Gründe dafür erklärt.

* Qt (Qt6)
  + Wird für grafische Oberfläche (GUI) verwendet
  + Ermöglicht plattformübergreifende Desktop-Entwicklung
  + Gute Unterstützung für UI-Design, Events und eigene Widgets
  + Integration in C++ Projekt und Build-System (CMake) ist Standard
* googletest
  + Framework für automatisierte Unit-Tests
  + Weit verbreitet im C++ Umfeld, auch im industriellen Einsatz
  + Moderne C++ Features für Testfälle und Test Suites
  + Bringt professionelle Entwicklungsmethodik (Test Driven Development, Absicherung von Code…)

**1.3 Bibliotheken**

Bisher werden im Projekt folgende Bibliotheken verwendet, diesen können sich jedoch je nach Projektablauf noch verändern. **HIER NOCH BIBLIOTHEKEN ERWEITERN**

* Standardbibliothek (STL): Für Vektoren, Strings, Maps usw.
* **ERGÄNZEN**

**Entwicklungswerkzeuge**

* VS Code mit C++ und Qt Extension
* Qt Designer zum erstellen der grafischen Oberfläche
* CMake als Buildsystem
* GitHub zur Versionskontrolle
* windeployqt zum Erstellen von Release-Paketen (Deployment für Windows)

1. **Architekturübersicht**

In diesem Projekt habe ich mich auf eine Schichtenarchitektur entschieden (3-Layer). Diese ist übersichtlich, leicht erweiterbar und testbar. Für das Projekt ist diese Architektur meiner Meinung nach ausreichend.

* **UI-Schicht:** Zeigt Aufgaben & Dialoge, umgesetzt mit Qt
* **Logikschicht:** Verarbeiten der Eingaben, Geschäftsregeln
* **Datenhaltung:** Speichert die Aufgaben lokal

1. **Struktur**

**3.1 Hauptkomponenten**

* MainWindow
  + Verantwortlich für das Hauptfenster und die Steuerung der Oberfläche
  + Stellt Liste der Aufgaben dar, Buttons für Aktionen
* Task
  + Datenklasse für einzelnen Aufgaben (Attribute: Titel, Fälligkeitsdatum, Status…)
  + Methoden: z.B. als erledigt markieren
* TaskManager
  + Verwalten der Sammlung aller Aufgaben (Hinzufügen, Löschen, Suchen)
  + Schnittstelle zwischen UI und Speicherung
  + Verantwortlich für Speichern/Laden der Aufgaben

**3.2 Abhängigkeiten**

* MainWindow nutzt TaskManager für Logik
* TaskManager verwaltet Task-Objekte

**HIER KLASSENDIAGRAMM EINFÜGEN WENN FERTIG!!**

1. **Verhalten**

In diesem Abschnitt wird der Ablauf „Aufgabe anlegen und speichern“ beschrieben. Die beteiligten Objekte sind Benutzer, GUI, TaskController, Task (Aufgabe) und Datenspeicher. Die zentralen Schritte sind: Der Benutzer gibt die Daten ein 🡪 GUI übergibt an Controller 🡪 Controller erstellt Aufgabe 🡪 Aufgabe wird im Datenspeicher gespeichert 🡪 GUI zeigt Rückmeldung. Das folgende Diagramm zeigt die Interaktion mit dieser Komponenten beim Anlegen einer Aufgabe.

**HIER SEQUENZDIAGRAMM EINFÜGEN WENN FERTIG!!**