Hinweise:

* Schriftart Calibri, 12pt, 1.15facher Zeilenabstand, Blocksatz
* Ränder „Layout→Normal“
* Umfang: mind. 10 und max. 12 Seiten einschl. Titelblatt
* Die dunkelgrau gedruckten Hinweise (Stil „subtle emphasis“) sind zu entfernen.

Praktikum zum Modul  
Software Engineering

Sommersemester 2019

Louis Seubert, 3246700

Lukas von Loefen, 3232060

Moritz Wein, 3232286

Jens Wöhler, 3255305

# Kurzbeschreibung des Projekts

Unsere Anwendung wurde entwickelt um den Aufwand von manuell erstellten Backups von privaten Dateien zu reduzieren. Das Programm erstellt automatisch Backups von, vom Benutzer ausgewählte Dateien, in bestimmten Verzeichnissen. Um dem Benutzer die Konfiguration der Anwendung zu vereinfachen gibt es eine graphische Benutzeroberfläche, die unabhängig vom Back-End funktioniert. Die vom Benutzer eingestellte Konfiguration für das Programm wird mithilfe einer Datei gespeichert und zwischen den beiden Anwendungen ausgetauscht. Das Back-End läuft unsichtbar im Hintergrund, protokoliert Dateiänderungen im angegebenen Ordern und kopiert die geänderten Dateien in den zugeordneten Zielordner falls nötig. Für die Umsetzung unseres Projektes haben wir uns für C# und das .NET-Framework entschieden.

# Anforderungsanalyse

## User Stories

* Als **Benutzer** muss es möglich sein nach dem Öffnen der Benutzeroberfläche eine Übersicht von allen bestehenden Aufgaben zusehen, um diese gegebenenfalls bearbeiten zu können.
* Dem **Benutzer** muss es ermöglicht werden Einstellungen an den bestehenden Aufgaben über die Benutzeroberfläche vor zu nehmen, diese müssen dabei auf ihre Richtigkeit überprüft werden.
* Falls der **Benutzer** die Benutzeroberfläche beendet, erscheint eine Anfrage, ob die Änderungen gespeichert werden soll, damit keine verloren gehen.
* Das **System** kann die Einstellungen aus einer Datei laden und ausführen

# Anforderungen

1. Das Programm muss dem Benutzer die Möglichkeit bieten die Konfiguration über eine intuitive Benutzeroberfläche zu verändern.
2. Das Programm darf keine Daten verlieren, wenn es abrupt beendet wird.
3. Das Programm darf die Dateien nicht verändern, auch wenn während des Kopiervorgang ein Fehler auftritt.
4. Das Programm sollte dem Benutzer die Möglichkeit bieten, sowohl in einem bestimmten Zeitabstand, als auch manuell das Backup zu starten.
5. Das Programm sollte dem Benutzer die Möglichkeit bieten, das Backup zu komprimieren.
6. Das Programm sollte dem Benutzer die Möglichkeit bieten, eine andere Sprache einzustellen.

# Modellierung

## Use Cases (Diagramme und tabellarische Form)

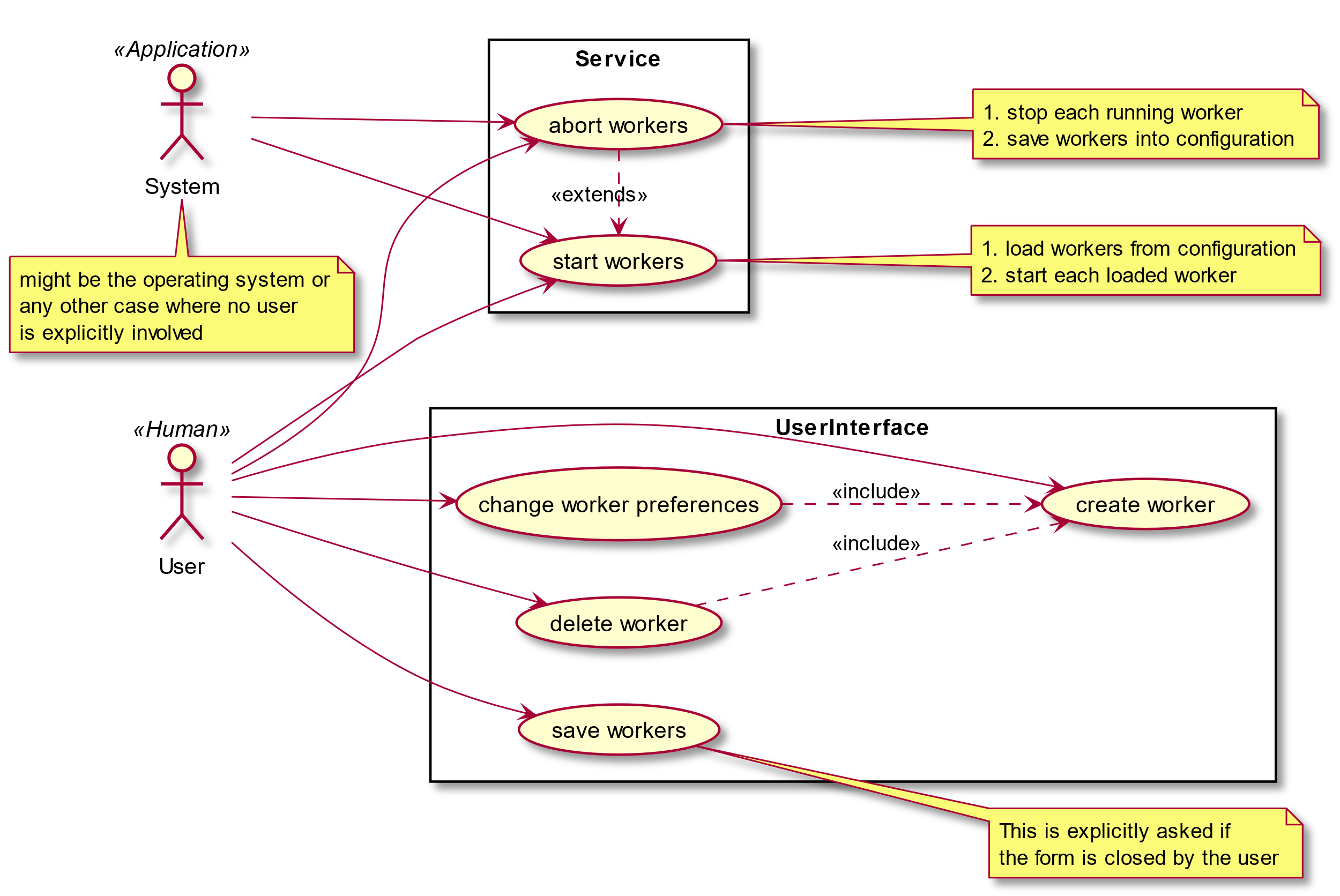


Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:  
<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Anwendungsfall/TinyTasks.svg>

|  |  |  |
| --- | --- | --- |
| **Name des Anwendungsfalls:** | Change Worker Preference | |
| **Nummer:** | Anwendungsfall 1 | |
| **Ziel:** | Eine Änderung der Parameter für einen bestimmten „Worker“ über das Benutzerinterface | |
| **Geltungsbereich:** | Die Benutzeroberfläche | |
| **Level:** | Hauptfunktionalität | |
| **Kategorie:** | Primär | |
| **Beteiligte Klassen/Objekte:** | Benutzerinterface, Benutzer | |
| **Vorbedingung:** | Der Benutzer hat bereits das Benutzerinterface geöffnet und einen „Worker“ angelegt | |
| **Nachbedingung (Erfolg):** | Die Eingaben werden übernommen und beim Schließen der Oberfläche gespeichert | |
| **Nachbedingung (Fehlschlag):** | Der entstandene Fehler wird über eine Message Box dem Benutzer angezeigt sollte dies Jedoch ein Vorheergesehener Fehler sein wird eine erneute Eingabe verlangt | |
| **Akteure:** | Benutzer | |
| **Auslösendes Ereignis:** | Der Benutzer wählt einen im Dashboard angezeigten „Worker“ mit Doppelklick aus | |
| **Auszuführende Aktionen (Erfolg):** | 1. | Die bereits vor definierten Parameter lesen und gegebenenfalls eine Fehlerkorrektur bei falsch Definition durchführen |
| 2. | Das anzeigen der Parameter über einen neuen Dialog mit der Möglichkeit diese zu bearbeiten |
| 3. | Beim schließen des Parameter Dialogs eine Überprüfung der eingegebenen Daten sowie gegebenenfalls eine Reaktion, wenn benötigte Daten fehlen |
| **Auszuführende Aktionen (Fehlschlag):** | 1. | Das Öffnen einer Message Box mit dem abgefangenen Fehler *(Dies ist immer der Fall für unvorhergesehene Fehler)* |
| 1. | Das erneute öffnen des Dialoges und eine Akustische Mitteilung an den Benutzer das eine der Eingaben nicht korrekt gewesen ist *(Dies ist der Fall falls eine der getroffenen eingaben nicht dem vordefinierten Muster bzw. Typen entsprechen)* |
| **Erweiterungen:** | -- | |
| **Verweise:** | Anforderung 1 | |

|  |  |  |
| --- | --- | --- |
| **Name des Anwendungsfalls:** | Create Worker | |
| **Nummer:** | Anwendungsfall 2 | |
| **Ziel:** | Einen neuen „Worker“ über das Benutzerinterface erstellen | |
| **Geltungsbereich:** | Die Benutzeroberfläche | |
| **Level:** | Hauptfunktionalität | |
| **Kategorie:** | Primär | |
| **Beteiligte Klassen/Objekte:** | Benutzerinterface, Benutzer | |
| **Vorbedingung:** | Der Benutzer hat bereits das Benutzerinterface geöffnet | |
| **Nachbedingung (Erfolg):** | Der neu erstellte „Worker“ wird nach dem erstellen in den aktuellen geladenen Kontext mitaufgenommen und beim Schließen der Oberfläche gespeichert. Sollte der „Worker“ weitere Eingabe werte verlangen wird der Benutzer dazu aufgerufen diese einzugeben, sollte das nicht der Fall sein wird der erstell Prozess abgebrochen | |
| **Nachbedingung (Fehlschlag):** | Der entstandene Fehler wird über eine Message Box dem Benutzer angezeigt sollte dies Jedoch ein Vorheergesehener Fehler sein wird eine erneute Eingabe verlangt | |
| **Akteure:** | Benutzer | |
| **Auslösendes Ereignis:** | Der Benutzer klickt im Dashboard doppelt auf eine Leere Zeile | |
| **Auszuführende Aktionen (Erfolg):** | 1. | Anzeigen eines Auswahl Dialogs zum wählen des „Worker“ Typen |
| 2. | Das instanziieren eines neuen „Worker“ Objektes |
| 3. | Bei weiteren benötigten Parametern soll das Parameter Fenster geöffnet werden, sollten diese eingaben fehlerbehaftet sein wird der Vorgang abgebrochen |
| **Auszuführende Aktionen (Fehlschlag):** | 1. | Das Öffnen einer Message Box mit dem abgefangenen Fehler *(Dies ist immer der Fall für unvorhergesehene Fehler)* |
| 1. | Das abbrechen des Vorgangs bei einer Fehlerhaften Konfiguration *(Dies ist der Fall falls eine der getroffenen eingaben nicht dem vordefinierten Muster bzw. Typen entsprechen)* |
| **Erweiterungen:** | -- | |
| **Verweise:** | Konfigurationsvorgang siehe Anwendungsfall 1 Anforderung 1 | |

|  |  |  |
| --- | --- | --- |
| **Name des Anwendungsfalls:** | Start Workers | |
| **Nummer:** | Anwendungsfall 3 | |
| **Ziel:** | Das Laden einer Konfigurationsdatei, welche die von Benutzer erstellten „Worker“ beinhaltet. | |
| **Geltungsbereich:** | Der Service | |
| **Level:** | Hauptfunktionalität | |
| **Kategorie:** | Primär | |
| **Beteiligte Klassen/Objekte:** | Betriebssystem, Benutzer | |
| **Vorbedingung:** | Der Benutzer hat bereits über Benutzerinterface „Worker“ erstellt und diese gespeichert | |
| **Nachbedingung (Erfolg):** | Die geladenen „Worker“ werden verwaltet vom Service und gegebenenfalls gestoppt | |
| **Nachbedingung (Fehlschlag):** | Der entstandene Fehler wird in der Konsole ausgegeben, wenn diese sichtbar ist, ansonsten wird der Fehler ignoriert und gegebenenfalls das Programm beendet | |
| **Akteure:** | Betriebssystem | |
| **Auslösendes Ereignis:** | Die Applikation wird gestartet mit entsprechenden Parametern | |
| **Auszuführende Aktionen (Erfolg):** | 1. | Überprüfen ob Datei existiert |
| 2. | Lesen des Dateiinhaltes |
| 3. | Deserialisierung des Inhaltes und Wiederherstellung der Objekte mit entsprechenden Daten |
| 4. | Sicheres starten der geladenen Aufgaben |
| **Auszuführende Aktionen (Fehlschlag):** | 1. | Das ausgeben des Fehlers auf der Konsole *(Dies ist immer der Fall für unvorhergesehene Fehler)* |
| 1. | Die angemessene Reaktion auf den Fehler, sollte diese die gesamte Funktionalität beinträchtigen, wird das Programm frühzeitig beendet *(Dies ist der Fall falls eine der benötigten Eingabe Parameter fehlt)* |
| **Erweiterungen:** | -- | |
| **Verweise:** | ? | |

## Klassendiagramme

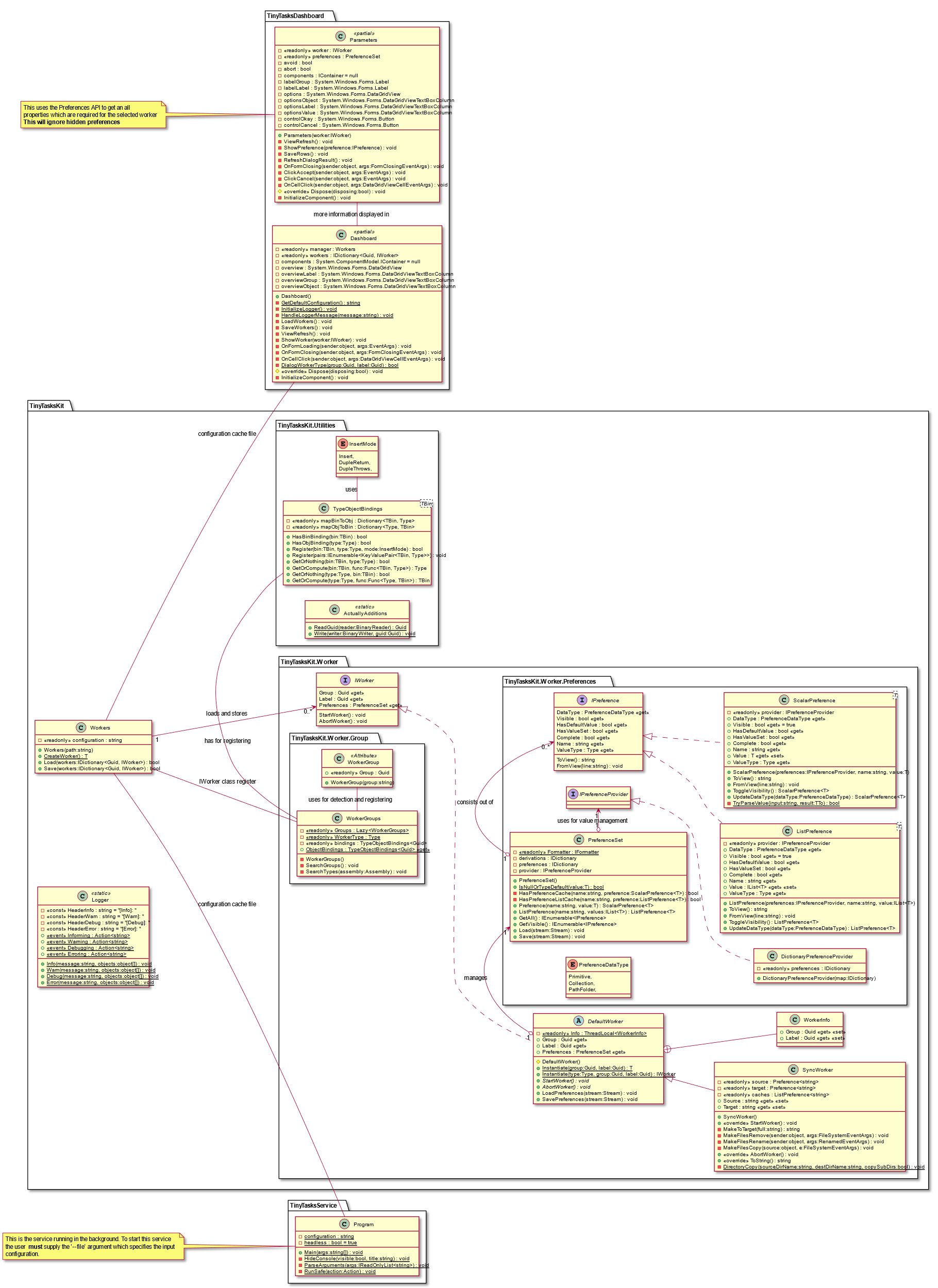
**

Diagramm ist auch auf GitHub zu finden, für eine genauere Betrachtung:   
<https://github.com/Louis9902/Sosse19-SE/blob/master/Diagramme/Klassen/TinyTasks.svg>

Objektdiagramme

Mind. 1 Objektdiagramm für einen geeigneten Use Case; verzeichnen Sie auch externe Schnittstellen bzw. Module.

## Sequenzdiagramme

Mind. 1 Sequenzdiagramm für einen geeigneten Use Case.

## Zustandsdiagramme

Mind. 1 Zustandsdiagramm für eine Komponente, einen Use Case oder die gesamte Applikation.

# Implementierung

Beschreiben Sie die Rahmenbedingungen und begründen Sie kurz die Wahl der verwendeten Toolkits. Listen Sie die erfüllten sowie nicht erfüllten Anforderungen, und erläutern Sie, welchen individuellen Beitrag jedes Teammitglied geleistet hat.

## Erfüllte Anforderungen

## Nicht erfüllte Anforderungen

## User Interface

Zeigen Sie anhand von Fotos oder Screenshots, wie Ihr Prototyp die Eingangs beschriebenen User Stories nachbildet. Referenzieren Sie entsprechend die (nicht) erfüllten Anforderungen (max. 2 Seiten).

# Reflexion

Vergleichen Sie die frühe Planung in den ersten Wochen mit den dem Ergebnis am Ende.

* Entspricht das Ergebnis Ihrer ursprünglichen Vision?
* Was lief wie geplant, welche (überraschenden?) Änderungen haben sich ergeben?
* An welcher Stelle sind Sie nicht weitergekommen, wo haben Sie am meisten dazugelernt?
* Hat sich die geplante Arbeitsaufteilung bewährt?
* Was würden Sie für das nächste Projekt anders machen?