**Jira Synchronizer für das interne Leistungserfassungssystem**

**Individuelle Praktische Arbeit**

Simon andré Lötscher

Softec AG

Inhaltsverzeichnis

[Teil 1 – Umfeld und Ablauf 6](#_Toc161903122)

[1 Aufgabenstellung 6](#_Toc161903123)

[1.1 Titel der Arbeit 6](#_Toc161903124)

[1.2 Ausgangslage 6](#_Toc161903125)

[1.3 Detaillierte Aufgabenstellung 6](#_Toc161903126)

[1.3.1 Kriterien für den Import von Zeitbuchungen einer Aufgabe 6](#_Toc161903127)

[1.3.2 Felder welche beim Import von Jira Zeitbuchungen abgefüllt sein müssen 7](#_Toc161903128)

[1.3.3 Out of Scope 7](#_Toc161903129)

[1.3.4 Vorgaben 7](#_Toc161903130)

[1.4 Mittel und Methoden 7](#_Toc161903131)

[1.5 Vorkenntnisse 7](#_Toc161903132)

[1.6 Vorarbeiten 8](#_Toc161903133)

[1.7 Neue Lerninhalte 8](#_Toc161903134)

[1.8 Arbeiten in den letzten 6 Monaten 8](#_Toc161903135)

[2 Projektaufbauorganisation 8](#_Toc161903136)

[2.1 Projektumfeld 8](#_Toc161903137)

[2.1.1 Softec AG 8](#_Toc161903138)

[2.2 Projektorganisation 9](#_Toc161903139)

[2.3 Projektmethode 10](#_Toc161903140)

[2.3.1 Vergleich von Projektmethoden 10](#_Toc161903141)

[2.3.1.1 IPERKA 10](#_Toc161903142)

[2.3.1.2 Wasserfall 10](#_Toc161903143)

[2.3.1.3 Scrum 10](#_Toc161903144)

[2.3.2 Auswahl der Projektmethode 10](#_Toc161903145)

[2.3.3 Anwendung der ausgewählten Projektmethode 10](#_Toc161903146)

[2.4 Verwendete Technologien und Tools 11](#_Toc161903147)

[2.4.1 Entwickler-Tools 11](#_Toc161903148)

[2.4.1.1 Editoren 11](#_Toc161903149)

[2.4.1.2 Source Code Verwaltung 11](#_Toc161903150)

[2.4.1.3 Datenbank Management Tool 11](#_Toc161903151)

[2.4.2 Dokumentation 12](#_Toc161903152)

[2.4.3 Organisation und Sicherung der Arbeitsergebnisse 12](#_Toc161903153)

[3 Projektplanung 13](#_Toc161903154)

[3.1 Use Case 13](#_Toc161903155)

[3.2 Anforderungsliste 13](#_Toc161903156)

[3.3 User Stories 15](#_Toc161903157)

[3.4 Zeitplan 21](#_Toc161903158)

[3.4.1 Soll-Ist-Vergleich 22](#_Toc161903159)

[4 Arbeitsjournal 23](#_Toc161903160)

[Teil 2 – Projekt 34](#_Toc161903161)

[5 Kurzfassung 34](#_Toc161903162)

[6 Ausgangslage 35](#_Toc161903163)

[6.1 Einleitung 35](#_Toc161903164)

[7 Funktionalitätsumsetzung 36](#_Toc161903165)

[7.1 Einleitung 36](#_Toc161903166)

[8 Testing 37](#_Toc161903167)

[8.1 Einleitung 37](#_Toc161903168)

[8.2 Umfeld 37](#_Toc161903169)

[8.2.1 Testanlage 37](#_Toc161903170)

[8.2.2 Mittel und Methoden 37](#_Toc161903171)

[8.3 Testdrehbuch 37](#_Toc161903172)

[8.4 Testprotokoll 37](#_Toc161903173)

[8.4.1 Unit-Tests 37](#_Toc161903174)

[8.4.2 Abnahmetests 37](#_Toc161903175)

[8.5 Testreport 37](#_Toc161903176)

[8.5.1 Resultatübersicht 37](#_Toc161903177)

[8.5.2 Auswertung 37](#_Toc161903178)

[9 Schlusswort 38](#_Toc161903179)

[9.1 Fazit 38](#_Toc161903180)

[9.1.1 Projektplanung 38](#_Toc161903181)

[9.1.1.1 Zeitplanung 38](#_Toc161903182)

[9.1.1.2 Projektmethode 38](#_Toc161903183)

[9.1.2 Implementation 38](#_Toc161903184)

[9.1.3 Dokumentation 38](#_Toc161903185)

[9.1.4 Testing 38](#_Toc161903186)

[9.2 Nächste Schritte 38](#_Toc161903187)

[10 Verzeichnisse 39](#_Toc161903188)

[10.1 Glossar 40](#_Toc161903189)

[10.2 Literatur 41](#_Toc161903190)

[10.3 Abbildungsverzeichnis 42](#_Toc161903191)

[11 Anhang 43](#_Toc161903192)

[11.1 Git-Log 43](#_Toc161903193)

Über dieses Dokument

Die Projektdokumentation der IPA ist, wie von PkOrg vorgegeben, in zwei Teile unterteilt. Einzelne Themengebiete werden zusätzlich in Kapitel aufgeteilt. In den Fussnoten werden Informationen zu den in diesem Projekt verwendeten Technologien und Tools angegeben. Die verwendeten Quellen wurden in der Literatur festgehalten.

Akronyme, Glossar, Literatur und das Abbildungsverzeichnis sind im Kapitel Verzeichnisse zu finden.

Typografische Elemente

Im folgenden Abschnitt werden die typografischen Elemente dieses Dokuments genauer gezeigt und erklärt.

**ⓘ Informationsblock**

Ein solcher Informationsblock beinhaltet Informationen über ein zu dieser Arbeit zusammenhängendes Thema. Dieser Block soll dabei helfen, den Gesamtzusammenhang dieser IPA zu verstehen.

/\*\*

\* Source code snippets are also marked in information blocks.

\* The whole source code and all comments are written in english.

\*/

console.log('Hello World!');

Dokumentenhistorie

|  |  |  |
| --- | --- | --- |
| Datum | Git Commits | Beschreibung |
| 20.03.2024 | - | - Dokument erstellt anhand Aufgabenstellung und Kriterienkatalog  - Use Cases definiert und graphisch dokumentiert  - Anforderungen definiert  - User Stories erarbeitet  - Zeitplan erstellt  - Arbeitsjournal Tag 1 |
| 21.03.2024 | 82dc8c7,  6062e7f | - Aufgabenstellung vervollständigt  - Projektplanung bis Soll-Ist-Vergleich vervollständigt  - Projektauftrag dokumentiert  - Ausgangslage dokumentiert  - Architektur begonnen zu dokumentieren  - Arbeitsjournal Tag 2 |
| 22.03.2024 | B8e85a5,  b5bf138,  707768d | - Klassendiagramm erstellt  - Architektur vervollständigt  - Dokument überarbeitet  - Literaturverzeichnis und Glossar geschrieben  - Arbeitsjournal Tag 3 |
| 26.03.2024 | 2a8b1fe,  836e4b9 | - Literaturverzeichnis aktualisiert  - Projekt Kurzfassung ergänzt  - Arbeitsjournal Tag 4 |
| 27.03.2024 | fe5e306,  ad9a73d | - LEIS Datenbank dokumentiert  - Logging dokumentiert  - Arbeitsjournal Tag 5 |
| 28.03.2024 | 77a6675 | - Arbeitsjournal Tag 6 |
| 02.04.2024 | 2c07d44 | - Arbeitsjournal Tag 6  - Projekt Kurzfassung Ergebnisse  - Testprotokoll & Testreport dokumentiert  - Logging dokumentiert  - Development Modus dokumentiert  - Arbeitsjournal Tag 7 |
| 03.04.2024 |  | -  - Arbeitsjournal Tag 8 |
| 04.04.2024 |  | -  - Arbeitsjournal Tag 9 |
| 05.04.2024 |  | -  - Arbeitsjournal Tag 10  - Fertigstellung / Überarbeitung |

Teil 1 – Umfeld und Ablauf

1. Aufgabenstellung
   1. Titel der Arbeit

Jira Synchronizer für das interne Leistungserfassungssystem

* 1. Ausgangslage

An der Softec werden Zeitbuchungen in einem internen Leistungserfassungssystem (fortan auch LEIS genannt) vorgenommen. Ein Grossteil der Mitarbeiter arbeitet zusätzlich mit Jira und erfässt dort dieselben Zeitbuchungen auf den bearbeiteten Tickets. Dies führt zu einem Mehraufwand, welcher sich je nach Mitarbeiter auf wöchentlich mehrere Stunden belaufen kann. Derzeit wird das aktuelle LEIS durch eine Neuentwicklung abgelöst. Im Rahmen dieser Neuentwicklung ist der Wunsch nach einer Synchronisierung der in Jira erfassten Zeiten in das neue LEIS aufgekommen, damit die Zeiten nicht in zwei Systemen manuell eingeben werden müssen.

* 1. Detaillierte Aufgabenstellung

Der Jira Synchronizer soll als Kommandozeilen Programm mit der Programmiersprache C# implementiert werden. Die Parameter für die Ausführung des Kommandozeilen Tools sollen Flags für einen Development Modus (-dev / -d) umfassen. Für die Synchronisierung soll das Kommandozeilen Programm die REST API von Jira verwenden und anhand einer Projekt Whitelist die Aufgaben in Jira ermitteln, welche Zeitbuchungen haben und diese dann für den Import vorsehen. Dabei müssen entsprechende Importregeln beachtet werden. Der Import muss grundsätzlich beliebig oft am Tag gestartet werden können, allerdings darf immer nur eine Instanz des Imports laufen. Der Import und die Businesslogik sollen mittels Unit Tests getestet werden.

* + 1. Kriterien für den Import von Zeitbuchungen einer Aufgabe
* Projekt der Jira Aufgabe ist in der LEIS Datenbank in der Tabelle «T\_WHITELIST» erfasst.
* Die Jira Aufgabe muss über Zeitbuchungen verfügen.
* Das Feld "LEIS Konto" der Jira Aufgabe ist mit einem passenden LEIS Konto abgefüllt.
* Sollte das Feld leer oder mit einem ungültigen LEIS Konto abgefüllt sein, dann muss dies in ein Error Log geschrieben werden.
* Der Mitarbeiter muss auf das LEIS Konto buchen dürfen, sprich er muss berechtigt sein. Ist dies nicht der Fall, so darf die Zeitbuchung nicht in der LEIS Datenbank erfasst werden und es muss ein Eintrag im Error Log erfasst werden.
* Die LEIS Buchung muss mit der Jira ID der Jira Zeitbuchung verknüpft werden.
* Es werden die Jira Zeitbuchungen der letzten 7 Kalendertage importiert, sofern diese nicht aus einem vorherigen Durchlauf importiert wurden.
* Bei aktivierten Development Modus sollen alle Zeitbuchungen auf einem bestimmten Mitarbeiter verbucht werden. Dies dient ausschliesslich der Kontrolle während der Entwicklungszeit. Dies ist nötig, weil die Development (und Test-) Datenbank anonymisiert ist. Der Mitarbeiter für diese Ausführung soll in den App Settings definiert sein.
  + 1. Felder welche beim Import von Jira Zeitbuchungen abgefüllt sein müssen
* Projekt
* Leistungsart (hier wird immer «Realisierung» als Wert verwendet)
* Mitarbeiter
* Datum (von/bis)
* Geleistete Zeit
* Beschreibung (hier wird jeweils die Jira Ticket Nummer verwendet)
* Interner Kommentar (hier wird der Kommentar der Zeitbuchung in Jira verwendet)
* Verrechenbar (wird von der Projekteinstellung in der LEIS Datenbank abgeleitet)
* Zeitklasse (hier wird immer «Normalzeit» als Wert verwendet)
  + 1. Out of Scope
* Die in der Jira Aufgabe eingetragene geleistete Zeit wird eins zu eins übernommen, auch wenn die Zeitbuchung grösser als 24 Stunden ist.
  + Wenn die Geleistete Zeit länger als 24h ist, soll eine Warnung in das Error Log geschrieben werden.
* Zeitklasse "Überzeit" wird nicht automatisch gesetzt.
* Jira Zeitbuchungen, welche schon importiert wurden, werden nicht erneut auf Änderungen geprüft und werden auch nicht automatisch aktualisiert.
* Rundung von Zeitbuchungen auf 15 Minuten werden nicht automatisch vorgenommen (bei den meisten Projekten müssen Mitarbeiter auf 15 Minuten Intervalle runden, jedoch nicht bei allen).
* Projekte werden in LEIS nicht automatisch angelegt.
* Die Whitelist muss manuell in der LEIS Datenbank abgefüllt werden.
* Zeitbuchungen werden immer für die vergangenen sieben Tage importiert.  
  Es wird dabei nicht beachtet, ob bei einem Monatswechsel der Vormonat für die Zeitbuchungen durch Mitarbeiter bereits gesperrt wurde.
  + 1. Vorgaben

Der Jira Synchronizer wird als Kommandozeilen Programm erstellt, welches in einem Scheduler zeitgesteuert und automatisch ausgeführt werden können soll.

* 1. Mittel und Methoden
* Arbeitslaptop (durch Softec zur Verfügung gestellt)
* Visual Studio Professional 2022 und Visual Studio Code
* Microsoft SQL Server Management Studio 19
* Microsoft SQL Datenbankserver 2016
* Als Programmiersprachen werden C# und T-SQL verwendet.
  1. Vorkenntnisse

Als Lehrlingsprojekt durfte ich mit der Neuentwicklung des neuen LEIS beginnen. Voraussetzung für das neue LEIS ist es, auf die alte LEIS Datenbank aufzusetzen. Als solches ist auch die Kenntnis des alten Datenbankschemas und seiner Eigenheiten aufzuführen. Da das LEIS und seine Datenbank beide mittlerweile weit über zehn Jahre alt sind, verfügt die Datenbank über ein paar Eigenheiten. Als solches arbeitet sie exklusiv mit Triggern und verfügt über keine Foreign Keys.

Ich arbeite täglich mit der Programmiersprache C# und mit der Sprache T-SQL bin ich durch meine Arbeit mit der LEIS Datenbank sowie einigen anderen Projekten der Softec vertraut. Ich verwende Visual Studio Professional 2022 und Visual Studio Code praktisch täglich, das Microsoft SQL Server Management Studio 19 habe ich auch schon oft verwendet, und mit dem Microsoft SQL Datenbankserver 2019 bin ich vertraut.

* 1. Vorarbeiten

Zu den Vorarbeiten gehört eine Machbarkeitsstudie, in welcher ich schaute, ob ich die Jira API wie erwartet ansteuern und Daten davon importieren kann. Das aus der Studie hervorgegangene Werkstück wurde wieder restlos gelöscht. Backups gibt es ebenfalls keine.

Weiterhin ist hier die begonnene Neuentwicklung vom LEIS sowie das Studium des Datenbankschemas aufzuführen. Für die IPA wird eine anonymisierte Version der LEIS Datenbank verwendet.

* 1. Neue Lerninhalte

Trotz der durchgeführten Machbarkeitsstudie ist die Jira API als neuer Lerninhalt aufzuführen, da die Machbarkeitsstudie lediglich grob testete, ob die Jira API wie erwartet ansteuerbar ist, nicht wie sie detailliert funktioniert, wie man sich darauf zu orientieren hat, oder was potentielle Schwierigkeiten sind.

* 1. Arbeiten in den letzten 6 Monaten
* Neuentwicklung vom LEIS
* Reports auf Basis von DevExtreme Reports erstellen
* Erweiterungen an bestehenden ASP.net Core Web APIs für die Reports und für die Angular Webapplikationen
* QR Code Rechnungsgenerator als Kommandozeilen Programm

Es waren alle aufgeführten Aufgaben von grösserem Umfang. Dabei kamen die Programmiersprachen C# und T-SQL, sowie das Framework Angular zum Einsatz. Diese Arbeiten wurden in Vollzeit ausgeführt.

1. Projektaufbauorganisation
   1. Projektumfeld
      1. Softec AG

Im Bereich Software Engineering entwickelt die Softec AG in Steinhausen Applikationen für diverse Kunden innerhalb der Schweiz. Ihre Produkte dienen unter anderem der interkantonalen Landwirtschaftsdatenerhebung, der Einsatzerfassung polizeilicher Dienste im Kanton Zürich, oder dem Aufbau und Betrieb von Geoinformationssystemen. Die Softec ist jedoch nicht nur im Bereich Software Engineering tätig, sondern auch in den Bereichen Platform Engineering, Network Engineering, Security Engineering und Operation Services.

* + - 1. Code Fathers

«Code Fathers» ist eines der beiden Software Engineering Teams der Softec. Im Team Code Fathers werden Individualsoftware für diverse Kunden und integrierte Businessprozesse entwickelt, sowie bestehende Applikationen gewartet.

* + - 1. LAWIS

«LAWIS» ist ebenfalls ein Software Engineering Team der Softec. Das Team LAWIS beschäftigt sich mit der interkantonalen Landwirtschaftsdatenerfassung der Deutschschweiz, mit der Organisation Agricola als Kunde.

* + 1. bildxzug Lehre im Verbund

Die bildxzug ist ein Lehrbetrieb welcher Lehrlinge an andere Firmen vermittelt und sie in ihrer Lehre begleitet und unterstützt. Sie unterstützt Lehrlinge mit verschiedensten Angeboten, organisiert Anlässe und berät Lehrlinge sollten Schwierigkeiten während der Lehre auftauchen.

* 1. Projektorganisation

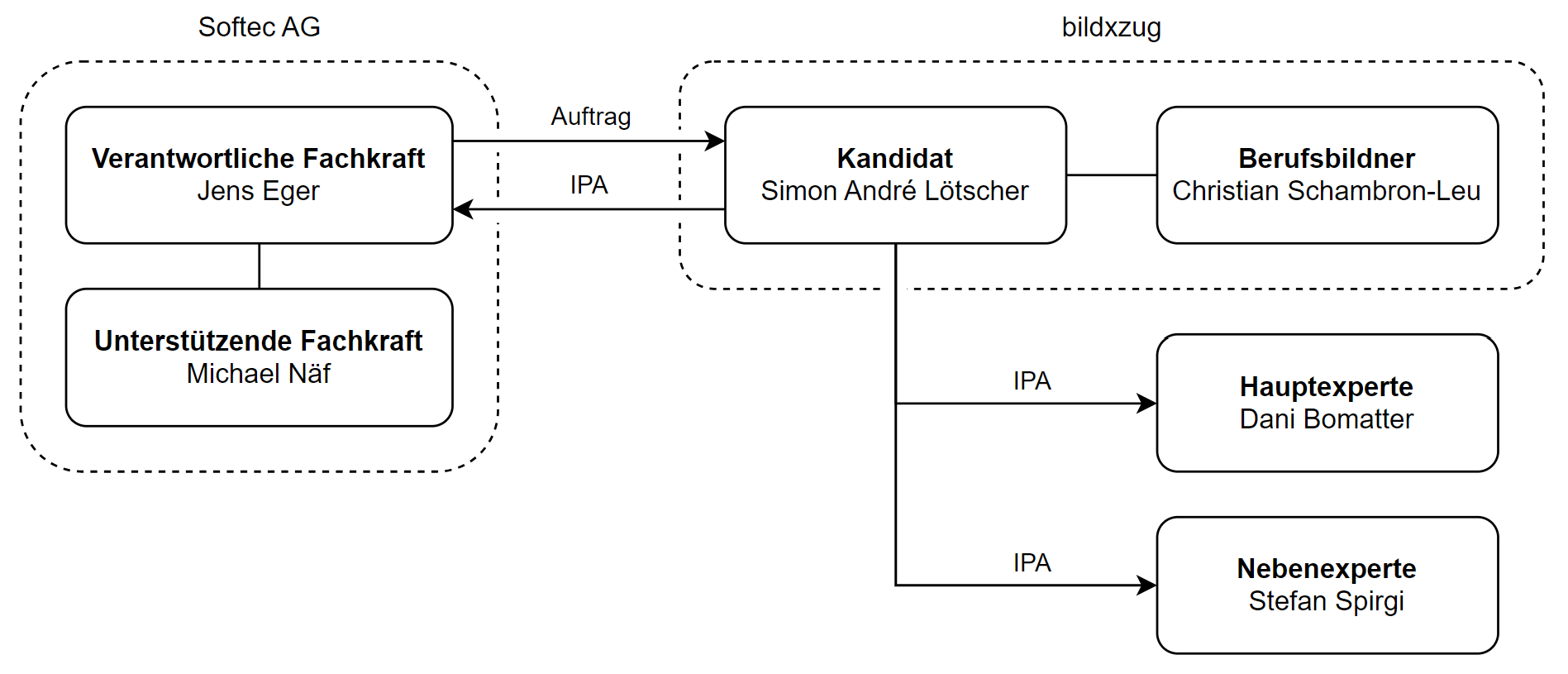


Abbildung 2.2 Projektorganisation

* + 1. Beteiligte Personen

**Dani Bomatter Hauptexperte**

E-Mail dani.bomatter@sisag.ch

Telefon 041 875 07 33

**Stefan Spirgi Nebenexperte**

E-Mail stefan.spirgi@vontobel.com

Telefon 079 217 74 18

**Jens Eger Verantwortliche Fachkraft**

E-Mail jens.eger@softec.ch

Telefon 041 747 07 23

**Michael Näf Unterstützende Fachkraft**

E-Mail michael.naef@softec.ch

Telefon 041 747 07 39

**Christian Schambron-Leu Berufsbildner**

E-Mail christian.schambron-leu@bildxzug.ch

Telefon 041 544 77 04

**Simon André Lötscher Lernender / Entwickler**

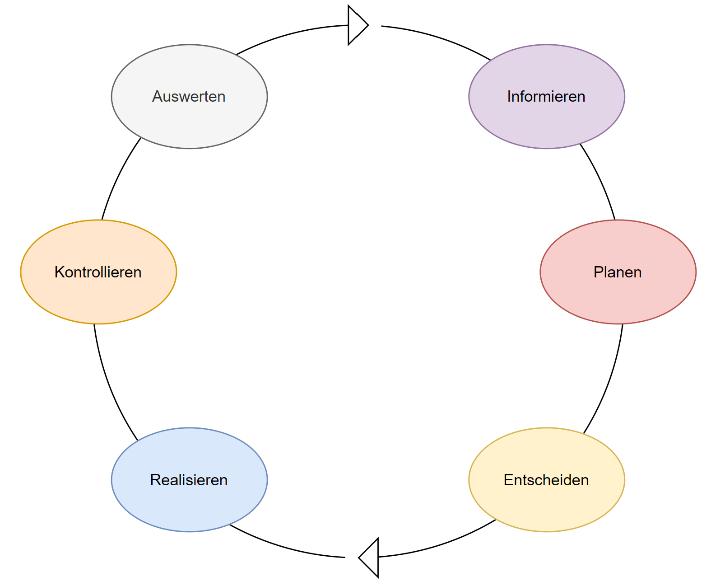
E-Mail simon.loetscher@softec.ch

Telefon 041 755 34 06

* 1. Projektmethode
     1. Vergleich von Projektmethoden

Ich möchte mich gut informiert für eine Projektmethode entscheiden, deswegen vergleiche ich hier die mir vertrautesten und generell am weitesten verwendeten Methoden.

* + - 1. IPERKA



IPERKA ist eine Methode mit sechs Phasen. Die einzelnen Phasen werden eine nach der anderen abgearbeitet und sollen möglichst voneinander getrennt werden.

Bei IPERKA liegt der Schwerpunkt ganz klar auf der Planung. Bevor die Arbeit überhaupt beginnt, sollen so viele Informationen wie möglich gesammelt werden. Es fördert Reflexion der Prozesse, ist jedoch immer noch sehr produktivorientiert. [1]

Abbildung 2.3.1.1 IPERKA

* + - 1. Wasserfall

Die Wasserfallmethode ist linear, alle Phasen haben definierte Start- und Endpunkte mit klar definierten Ergebnissen. Aktivitäten müssen in der vorgegebenen Reihenfolge durchgeführt werden, hierbei sind die Vorgänge sequenziell, das heisst eine Aktivität muss beendet sein, bevor eine weitere beginnt. Am Ende jeder Phase soll dokumentiert werden. [2]

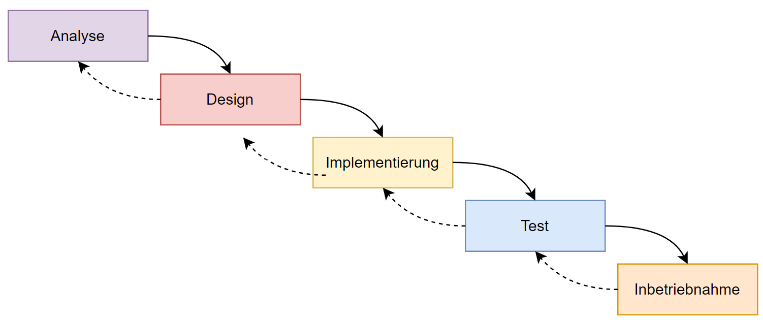


Abbildung 2.3.1.2 Wasserfall

* + - 1. Kanban

Die Kanban Methode ist eine agile Arbeitsmethode. Ursprünglich kommt sie aus der Lean-Produktion und wurde später auf Softwareentwicklung und andere Bereiche übertragen. Das Hauptziel von Kanban ist es, den Workflow zu visualisieren, Durchlaufzeiten zu verkürzen, und die Effizienz zu verbessern. Normalerweise wird versucht den Workflow mit einem sogenannten Kanban Board zu visualisieren. [3]

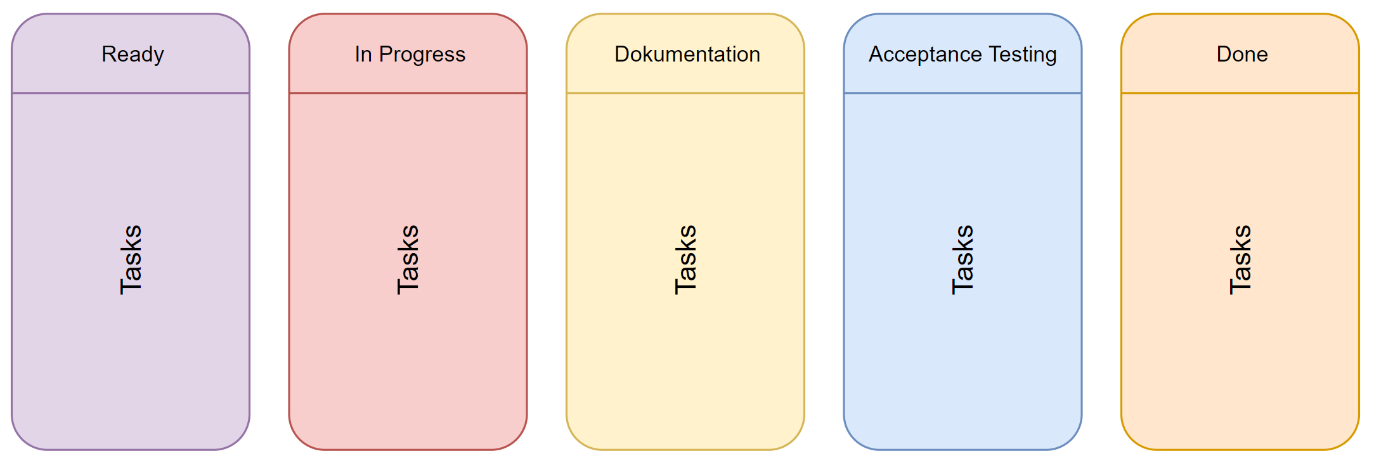


Abbildung 2.3.1.3 Kanbanboard

* + 1. Auswahl der Projektmethode

Weder IPERKA noch die Wasserfall Methode eignen sich besonders gut für mein Projekt. Keine dieser beiden Methoden ist flexibel genug für den Jira Synchronizer, da ich mir gut vorstellen kann im Nachhinein noch kleine Änderungen machen zu müssen. So weiss ich zum Beispiel trotz intensiver Planung noch nicht, ob die bisher geplanten Anpassungen an das bestehende Datenbankschema ausreichen, oder ob einzelne Spalten oder sogar ganze Tabellen noch erweitert werden müssen, welche ich bisher noch gar nicht in Betrachtung gezogen habe, was durchaus der Fall sein könnte, wenn ich zum Beispiel von der Jira API unerwartete Daten erhalte.

Des Weiteren erlaubt es mir Kanban flexibel auf unerwartete Umstände zu reagieren. Ich weiss zum Beispiel noch nicht genau, was für Daten ich von der Jira API zu erwarten habe. Wenn ich in einer linearen, starren Projektmethode feststecken würde, könnte dies zum Problem werden. Auch wenn mir beim Testen auffällt, dass etwas nicht so funktioniert wie ich es mir wünsche, kann ich bei Kanban einfach einen neuen Task einfügen oder einen bestehenden anpassen und diesen weiterentwickeln.

Zu guter Letzt hilft mir Kanban auch, den Arbeitsfortschritt zu visualisieren. Mit den verschiedenen Listen und Tasks sehe ich sehr schnell, wo ich stehe, wie weit ich bin, was noch fehlt, oder woran ich gerade arbeite. Ausserdem kann ich diese visuelle Unterstützung gut auch in der Dokumentation verwenden, um meinen Fortschritt visuell festzuhalten.

Aus diesen Gründen entscheide ich mich für die agile Methode Kanban als meine Projektmethode.

* + 1. Anwendung der ausgewählten Projektmethode

Im Rahmen des Projektes werden User Stories erstellt welche die verschiedenen Aufgaben, welche noch zu erledigen sind, abdecken. Diese User Stories werden in einem Kanban Board auf Trello festgehalten. Ich entschied mich bewusst für Trello, da es mir wichtig war eine klare Trennung zwischen meinem Projekt und den Tasks meines Projektes zu haben, welche mir potenziell gefehlt hätte, wenn ich Jira verwendet hätte.

Das Kanban Board verfügt über die Listen «Ready» für Tasks, welche noch nicht in Bearbeitung sind, «In Progress» für Tasks welche aktuell in Bearbeitung sind, «Dokumentation» für Tasks welche grundsätzlich fertig bearbeitet sind, jedoch noch dokumentiert werden müssen, «Acceptance Testing» für Tasks welche noch getestet werden müssen (vor allem bei Programmieraufgaben anwendbar), und «Done» für erledigte Tasks.

Um die Organisation einfacher zu gestalten sind Tasks des Weiteren den User Stories, welchen sie angehören entsprechend farbkodiert und verfügen über eine Checkliste namens «Akzeptanzkriterien» mit welcher überprüft werden kann, ob die Tasks bereit sind von «In Progress» auf «Acceptance Testing» über zu gehen.

Ich möchte vermeiden mehr als drei Tasks auf einmal in «In Progress» zu haben und am Ende jedes Tages will ich diese Anzahl auf maximal zwei reduzieren. Der Task «Arbeitsjournal führen» wird während der ganzen Arbeit in der Liste «In Progress» stehen, als Teil des Arbeitsjournals soll am Ende jedes Tages ein Screenshot des aktuellen Standes des Kanban Boards angefügt werden.

* 1. Verwendete Technologien und Tools
     1. Entwickler-Tools

Der ganze Jira Synchronizer wird auf dem durch die Softec zur Verfügung gestelltem Arbeitslaptop realisiert.

* + - 1. Editoren

Als Editor wird hauptsächlich Visual Studio Professional 2022 verwendet, wobei in vereinzelten Fällen auch Visual Studio Code verwendet wird. Visual Studio Code ist ein gratis Editor, welcher mit verschiedenen Erweiterungen ausgestattet werden kann. Visual Studio Professional 2022 ist eine durch Softec bereitgestellte kostenpflichtige Version des Visual Studio Editors. Beide Editoren sind Firmenstandard an der Softec.

* + - 1. Source Code Verwaltung

Für die Verwaltung des Source Codes und der Dokumentation wurde ein öffentliches Repository auf GitHub erstellt. Ziel ist es, pro Tag mindestens einen Commit auf dieses Repository zu pushen, dank täglicher Commits kann jederzeit auch auf frühere Versionen zugegriffen werden.

* + - 1. Datenbank Management Tool

Als Datenbank Management Tool kommt das SQL Server Management Studio 19 zum Einsatz. Das Tool ist gratis und in der Softec Firmenstandard.

* + 1. draw.io

Für selbsterstellte Grafiken wird draw.io verwendet. draw.io ist ein gratis Online-Tool welches primär zur Erstellung von Diagrammen und Flowcharts entwickelt wurde.

* + 1. Google Sheets

Google Sheets wird verwendet für die Erstellung von Tabellen wie dem Zeitplan oder der dazugehörigen Legende. Google Sheets gehört zu Google Docs und ist ein gratis online Editor mit ähnlichen Funktionen wie Microsoft Office Excel.

* + 1. Trello

Für die Verwaltung des Kanbanboards wird Trello, ein gratis Online-Tool, verwendet.

* + 1. Dokumentation

Für die Dokumentation wurde Microsoft Office Word (Word) verwendet. Ein bereits bestehendes Template wurde für Gestaltung sowie Formatierung der Dokumentation verwendet, wobei vor Beginn der IPA noch kein Inhalt in das Template eingefügt wurde.

* + 1. Organisation und Sicherung der Arbeitsergebnisse

Die Arbeitsergebnisse werden täglich mit einem Push auf das Github Repository gesichert. Ausserdem wird über ein privates Verzeichnis auf dem Softec Server gearbeitet, sollten also der Laptop sowie das Repository verloren gehen, sind die Daten hier immer noch vorhanden.

1. Projektplanung
   1. Use Case

Das Use Case Diagramm stellt die Benutzer- sowie Entwickleranforderungen grafisch dar.

Der Benutzer ist bei uns der Mitarbeiter, welcher auf Jira seine Leistungen in den jeweiligen Projekten festhält.

Der Entwickler ist die Person, welche die Applikation entwickelt und in der Zukunft verwaltet.

Ein Bild, das Text, Entwurf, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 3.1 Use Case Diagramm

* 1. Anforderungsliste

Die Anforderungsliste wird der detaillierten Aufgabenstellung sowie den individuellen Bewertungskriterien entnommen.

**Nummer Beschreibung Priorität**

A1 **Testing**

A1.1 Die Applikation wird durch Unit Tests getestet. Hoch

A1.2 Die Unit Tests decken den Grossteil der Applikation ab. Hoch

A1.3 Manuelle Tests der Applikation sind erstellt. Mittel

A1.4 Die Durchführung aller Tests inklusive deren Resultate ist dokumentiert. Hoch

A1.5 Die Solution verfügt über ein separates Unit Test Project. Mittel

A2 **Logging**

A2.1 Während der Laufzeit der Applikation wird ein Log geführt. Hoch

A2.2 Unerwartete Fehler in der Applikation werden im Log als Fehlermeldung Hoch

hinterlegt.

A2.3 Fehlt das Feld «Leis Konto» im Jira Projekt, muss dies im Log als Fehler Hoch

hinterlegt werden.

A2.4 Hat der Mitarbeiter im Ticket keine Berechtigung auf das Projekt wird Hoch

die Leistung nicht abgefüllt und es wird ein Fehler im Log hinterlegt.

A2.5 Fehlen andere Angaben müssen diese als Warnung im Log erwähnt werden. Mittel

A3 **Schichtentrennung**

A3.1 Die Applikation ist in eine Persistenz-, Service-, und eine Applikationsschicht Hoch

getrennt (Core, Infrastructure, Application).

A3.2 Das Projekt ist gemäss anderen ähnlichen Firmenprojekten aufgebaut . Mittel

A4 **Software-Architektur**

A4.1 Es wurde eine passende Klassenstruktur gewählt. Hoch

A4.2 Die Klassen haben wohldefinierte, dokumentierte Schnittstellen. Mittel

A4.3 Die Klassen sind in passenden Modulen untergebracht. Hoch

A5 **Code**

A5.1 Mögliche Fehler werden erkannt und behandelt. Hoch

A5.2 Der Code ist gemäss Softec Vorgaben aufgebaut (Namensgebung, Lesbarkeit). Hoch

A6 **Robustheit**

A6.1 Abfüllen von fehlenden Daten wird verhindert. Hoch

A6.2 Gültigkeit angegebener Daten wird geprüft. Mittel

A6.3 Gefundene Unstimmigkeiten werden im Log erfasst. Mittel

A7 **Git-Repository**

A7.1 Der Code ist auf einem Github Repository vorhanden. Hoch

A7.2 Die Dokumentation ist auf demselben Repository vorhanden. Hoch

A7.3 Notizen, Entwürfe, Bilder sind im Repository vorhanden. Tief

A8 **Applikation**

A8.1 Die Applikation kann über einen Scheduler aufgerufen werden. Hoch

A8.2 Der Applikation kann ein Development Flag ab Aufruf übergeben werden. Hoch

A8.3 Die fertige Applikation ist ein Kommandozeilenprogramm. Hoch

* 1. User Stories

Die User Stories werden benötigt um den Zeitplan sowie das im Verlauf des Projekts verwendete Kanban Board zu erstellen. Sie werden für eine möglichst genaue Planung des Projektes benötigt.

Projektplanung – S1

Anforderungen

A4.1, A4.2

Beschreibung

Als Kandidat ist es mir wichtig, mein Projekt möglichst gut zu planen damit ich im vorgegebenen Zeitraum alle Anforderungen realisieren und das Projekt umsetzen kann.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T1.1 Use Cases definieren, graphisch darstellen. 1h 1

T1.2 Anforderungen definieren. 2h 2

T1.3 User Stories erarbeiten. 3h 3

T1.4 Zeitplan erstellen. 1h 4

T1.5 Vervollständigung Aufgabenstellung. 2h 5

T1.6 Vervollständigung Projektplanung. 2h 6

T1.7 Projektauftrag dokumentieren. 5h 7

T1.8 Ausgangslage dokumentieren. 1h 8

T1.9 Architektur dokumentieren. 1h 9

T1.10 Klassenstruktur mit Diagramm erstellen. 2h 10

**Aufwand Total** 20h

Akzeptanzkriterien

* Use Cases wurden definiert sowie per Grafik der Dokumentation hinzugefügt.
* Anforderungen wurden auf der detaillierten Aufgabenstellung aufbauend festgelegt.
* User Stories wurden erarbeitet und mit realistischer Aufwandsschätzung ausgestattet.
* Zeitplan wurde erstellt und auf PkOrg abgegeben.
* Die Aufgabenstellung wurde dokumentiert.
* Die Projektplanung wurde dokumentiert.
* Der Projektauftrag wurde dokumentiert.
* Die Ausgangslage wurde auf der detaillierten Aufgabenstellung aufbauend dokumentiert.
* Der Architekturentscheid wurde dokumentiert.
* Eine Klassenstruktur mit allen voraussichtlich benötigten Klassen wurde erstellt. Deren Beziehungen sind klar definiert und gut ersichtlich.

Projektinitialisierung – S2

Anforderungen

A1.5, A3, A4.3, A7, A8.1, A8.3

Beschreibung

Als Kandidat möchte ich meine Solution gemäss meiner gewählten Architektur erstellen und soweit vorbereiten dass ich damit in den nächsten Schritten ohne Unterbrechungen weiter arbeiten kann.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T2.1 Github Repository aufsetzen und mit lokalem Repository verbinden. 1h 1

T2.2 Solution mit allen Projekten (Core, Infrastructure, Application, 1h 2

Unit Tests) erstellen.

T2.3 Datenbank um «T\_WHITELIST» Tabelle erweitern. 1h 3

T2.4 «T\_LEISTUNGSERFASSUNG» Tabelle in der Datenbank wird um 1h 4

Spalte «LEI\_JIRA\_ID» erweitert.

T2.5 Alle in der Klassenstruktur enthaltenen Klassen werden als 2h 5

Entitäten im Core Projekt und als ViewModel im Application Projekt

Erstellt.

**Aufwand Total** 6h

Akzeptanzkriterien

* Initial Commit auf Github Repository wurde erstellt.
* Solution ist wie finale Applikation strukturiert (Application Project soll Kommandozeilen Project sein)
* Datenbank ist auf die weitere Entwicklung des Projektes vorbereitet.
* Alle in der Klassenstruktur enthaltenen Klassen wurden in die Solution eingebaut. Vorhanden sind die Entitäten im Core Projekt welche mit der Datenbank kommunizieren und die ViewModels im Application Projekt welche von der Applikation verwendet werden.

Abfragen Jira Rest API – S3

Anforderungen

A5.1, A6.1, A6.2

Beschreibung

Als Kandidat möchte ich erste Abfragen an die Jira API senden und Antwort erhalten. Hierbei möchte ich nur Anfragen an Projekte senden, welche in der «T\_WHITELIST» Tabelle in der Datenbank enthalten sind.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T3.1 «T\_WHITELIST» Tabelle in der Datenbank befüllen. 1h 1

T3.2 Atlassian API-Token für Jira erstellen. 1h 2

T3.3 Alle Zeitbuchungen welche noch nicht in der Datenbank 2h 3

vorhanden sind, werden für die letzten 7 Tage importiert.

T3.4 Testen ob alle nötigen Angaben mit plausiblen Werten gefüllt sind. 1h 4

T3.5 Testen ob hinterlegter Mitarbeiter innerhalb einer Buchung die 1h 5

benötigten Rechte auf das hinterlegte Leis Konto hat.

**Aufwand Total** 6h

Akzeptanzkriterien

* «T\_WHITELIST» Tabelle enthält einige Jira Projekte sowie deren entsprechende Leis Konten.
* Gültiger Atlassian API-Token wurde für die IPA erstellt.
* Zeitbuchungen werden, sofern sie in der Datenbank nicht schon enthalten sind, von der Jira REST API importiert.
* Es wird überprüft ob alle nötigen Angaben in den importierten Zeitbuchungen plausibel vorhanden sind.
* Es wird überprüft, ob der in der Zeitbuchung hinterlegten Mitarbeiter Rechte auf das hinterlegte Leis Konto hat.

Einfügen von Daten in Datenbank – S4

Anforderungen

-

Beschreibung

Als Benutzer möchte ich, dass, wenn ich meine Daten korrekt in Jira eingegeben habe, diese ohne dass ich weiter etwas machen muss, für mich im Leis verbucht werden.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T4.1 Nach erfolgreichen Plausibilitätstests und potenziellem 3h 1

Error-Handling werden Leistungen in «T\_LEISTUNGSERFASSUNG»

mit dem passenden User sowie der Id des Jira Projektes verbucht.

**Aufwand Total** 3h

Akzeptanzkriterien

* Sofern Plausibilitäts- und Berechtigungstests erfolgreich abgeschlossen wurden, werden Daten in die Tabelle «T\_LEISTUNGSERFASSUNG» geschrieben.
* Die Id des Jira Projekts wird für künftige Datenerfassungen hinterlegt.
* Zeiten werden nicht gerundet oder angepasst.

Logging – S5

Anforderungen

A2, A6.1

Beschreibung

Als Entwickler möchte ich verhindern, dass ungültige Daten in die Datenbank geschrieben werden oder zum Absturz der Applikation führen, Fehler sollen geloggt werden und nicht plausible Daten sollen ebenfalls als Warnungen geloggt werden.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T5.1 Zum Start der Applikation soll das bestehende Logfile geöffnet 1h 1

werden, oder falls keines vorhanden ist, ein neues erstellt werden.

T5.2 Werden unvollständig erfasste Zeiterfassungen von der Jira REST 1h 2

API importiert, soll ein Fehler im Log erfasst werden.

T5.3 Werden Daten, welche nicht plausibel sind von der Jira REST API 2h 3

importiert, soll eine Warnung im Log erfasst werden.

T5.4 Sollten sonstige Fehler innerhalb der Applikation vorkommen, 3h 4

sollen diese ebenfalls geloggt werden.

T5.5 Wird die Applikation beendet, soll dies im Log vermerkt sein. 1h 5

**Aufwand Total** 8h

Akzeptanzkriterien

* Ein Logfile soll vorhanden sein, welches mit Zeitstempeln den Start sowie das Ende eines Applikationsdurchlaufes beschreiben.
* Im Logfile werden Fehlermeldungen mit passenden Fehlercodes ebenfalls mit Zeitstempel erfasst.
* Im Logfile werden Warnungen mit passenden Codes ebenfalls mit Zeitstempel erfasst.

Development Modus – S6

Anforderungen

A8.2

Beschreibung

Als Entwickler möchte ich einen Development Modus, in welchem alle importierten Daten auf einen Development User gespeichert werden. Nötig ist dieser Development Modus da die Datenbank auf der Testumgebung anonymisiert ist, und somit Daten keinem spezifischen User zugewiesen werden können.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T6.1 Wird die Applikation ab Start mit dem Flag «-d» oder «-dev» 1h 1

aufgerufen, wird sie im Development Modus gestartet.

T6.2 Im Development Modus werden alle Daten mit dem Development 1h 2

User gespeichert.

**Aufwand Total** 2h

Akzeptanzkriterien

* Applikation kann in einem Development Modus gestartet werden.
* Läuft die Applikation im Development Modus, werden alle Zeiterfassungen mit dem Development User gespeichert.

Testing – S7

Anforderungen

A1

Beschreibung

Als Entwickler ist es mir wichtig, dass meine Applikation fehlerfrei läuft. Ich möchte, dass ich wenn ich die Applikation weiter entwickle oder verändere mögliche Fehler sofort auffallen. Mein Ziel ist es nach Test Driven Development vorzugehen.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T7.1 Mit Unit Tests soll eine möglichst hohe Code Coverage erreicht 8h 1

werden, wobei nur sinnvolle Methoden mit Unit Tests getestet

werden sollen.

T7.2 Wo Unit Tests nicht möglich oder sinnlos sind, sollen manuelle 4h 2

Tests durchgeführt werden. Für diese wird ein Testskript

geschrieben, sodass sie immer gleich durchgeführt werden

können.

T7.3 Am Ende der Entwicklungszeit sollen End-To-End Tests 2h 3

durchgeführt werden, womit die Funktionsweise der Applikation

als Ganzes getestet werden.

**Aufwand Total** 14h 4

Akzeptanzkriterien

* Methoden werden wo sinnvoll getestet, private Methoden werden nicht getestet oder gemockt. Die Datenbank wird ebenfalls gemockt.
* Ein Testskript soll klar beschreiben, wie ein Test durchzuführen ist und soll nur Teile der Applikation testen welche durch die Unit Tests nicht schon abgedeckt sind.
* End-To-End Tests sind Teil der manuellen Tests und verfügen ebenfalls über ein Testskript.
* Alle Tests werden erfolgreich abgeschlossen.

Finalisierung Applikation – S8

Anforderungen

A5.1, A5.2

Beschreibung

Als Entwickler möchte ich mit einer sauber geschriebenen Applikation abschliessen. Dafür möchte ich sowohl während als auch am Ende der Entwicklung des Projektes Code Guidelines einhalten. Am Ende der Entwicklungszeit möchte ich das Projekt nochmals überarbeiten, um zu gewährleisten, dass diese Guidelines eingehalten werden.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T8.1 Klassen-, Methoden-, und Variablennamen sollen sinnvoll und 1h 1

aussagekräftig sein.

T8.2 Wo nötig Code kommentieren (soll grösstenteils self-documenting 1h 2

sein, das heisst Methoden und Variablennamen sollen beschreiben

was sie machen und Kommentare sollten nicht nötig sein).

T8.3 Code aufräumen, unschönen Code refactorn. 2h

**Aufwand Total** 4h

Akzeptanzkriterien

* Klassen-, Methoden-, und Variablennamen sind selbstbeschreibend.
* Kommentare sind nur an wenigen Orten notwendig und verwendet.
* Code ist sauber und gut lesbar.

Finalisierung Dokumentation – S9

Anforderungen

-

Beschreibung

Als Kandidat will ich eine detaillierte und inhaltreiche Dokumentation erstellen. Wichtig ist mir hierbei, dass ich so viele Bewertungskriterien wie möglich abdecken kann und parallel zur Applikationsentwicklung dokumentieren kann.

Tasks

**Nummer Beschreibung Zeit Ablauf**

T9.1 Arbeitsjournal führen. 10h 1

T9.2 Dokumentation vervollständigen. 7h 2

**Aufwand Total** 17h

Akzeptanzkriterien

* Das Arbeitsjournal wurde täglich geführt, überarbeitet und für jeden Tag final abgeschlossen.
* Die Dokumentation ist vollständig, ohne Lücken und wurde gegen Ende nochmals überarbeitet.
* Alle Bewertungskriterien konnten erfolgreich abgearbeitet werden.
  1. Risikoanalyse
     1. Motivationsverlust – R1

Beschreibung

Der Kandidat verliert die Motivation seriös an der Arbeit weiterzuarbeiten oder erleidet ein Burnout.

Massnahmen

Arbeiten, für welche sich der Kandidat nur schwer motivieren kann, werden in den ersten paar Tagen erledigt. Während der ersten paar Tage der Arbeit wird ausserdem länger gearbeitet, sodass Zeitdruck vermindert wird und der Kandidat sich auf entspanntere weitere Arbeitstage freuen kann. Weiterhin wurde der Zeitrahmen der IPA so gewählt, dass die zehn Arbeitstage in drei Wochen aufgeteilt sind, und zwischen den Arbeitsblöcken grössere Pausen vorkommen (Schule, Ostern). Im Zeitplan wurde ausserdem darauf geachtet, dass grosse Arbeitsblöcke (wie den Projektauftrag zu dokumentieren) auf zwei Tage aufgeteilt wurden, statt ein Tag für ein Thema aufzuwenden.

Risikohöhe

Eintrittswahrscheinlichkeit: Tief

Auswirkungen: Hoch

* + 1. Unfall – R2

Beschreibung

Der Kandidat erleidet eine Verletzung und kann aufgrund dieser für einen längeren Zeitraum nicht weiterarbeiten.

Massnahmen

Das Expertenteam und die Fachkräfte werden über den Unfall sofort aufgeklärt. Weitere Schritte werden abgeklärt.

Risikohöhe

Eintrittswahrscheinlichkeit: Tief

Auswirkungen: Hoch

* + 1. Krankheit – R3

Beschreibung

Der Kandidat kann aufgrund einer Krankheit für eine Zeit nicht am Projekt weiterarbeiten.

Massnahmen

Das Expertenteam und die Fachkräfte werden über den Unfall sofort aufgeklärt. Weitere Schritte werden abgeklärt.

Risikohöhe

Eintrittswahrscheinlichkeit: Tief

Auswirkungen: Mittel

* + 1. Technische Störung (Verlust Laptop, Datenverlust) – R4

Beschreibung

Der Laptop des Kandidaten geht kaputt oder die Daten darauf gehen verloren.

Massnahmen

Daten werden auf dem persönlichen Verzeichnis des Kandidaten auf dem Softec Server gespeichert und es werden alle Daten zusätzlich täglich auf einem Github Repository gespeichert. Bei Geräteverlust muss ein Ersatzgerät eingesetzt werden.

Risikohöhe

Eintrittswahrscheinlichkeit: Mittel

Auswirkungen: Tief

* + 1. Zeitplan verschätzt – R5

Beschreibung

Der Kandidat verschätzt sich im Zeitplan und ist im Verzug.

Massnahmen

Der Zeitplan ist flexibel gestaltet und da die Mindestgrösse eines Eintrages eine Stunde beträgt wurden viele Schritte sehr grosszügig geschätzt.

Risikohöhe

Eintrittswahrscheinlichkeit: Hoch

Auswirkungen: Mittel

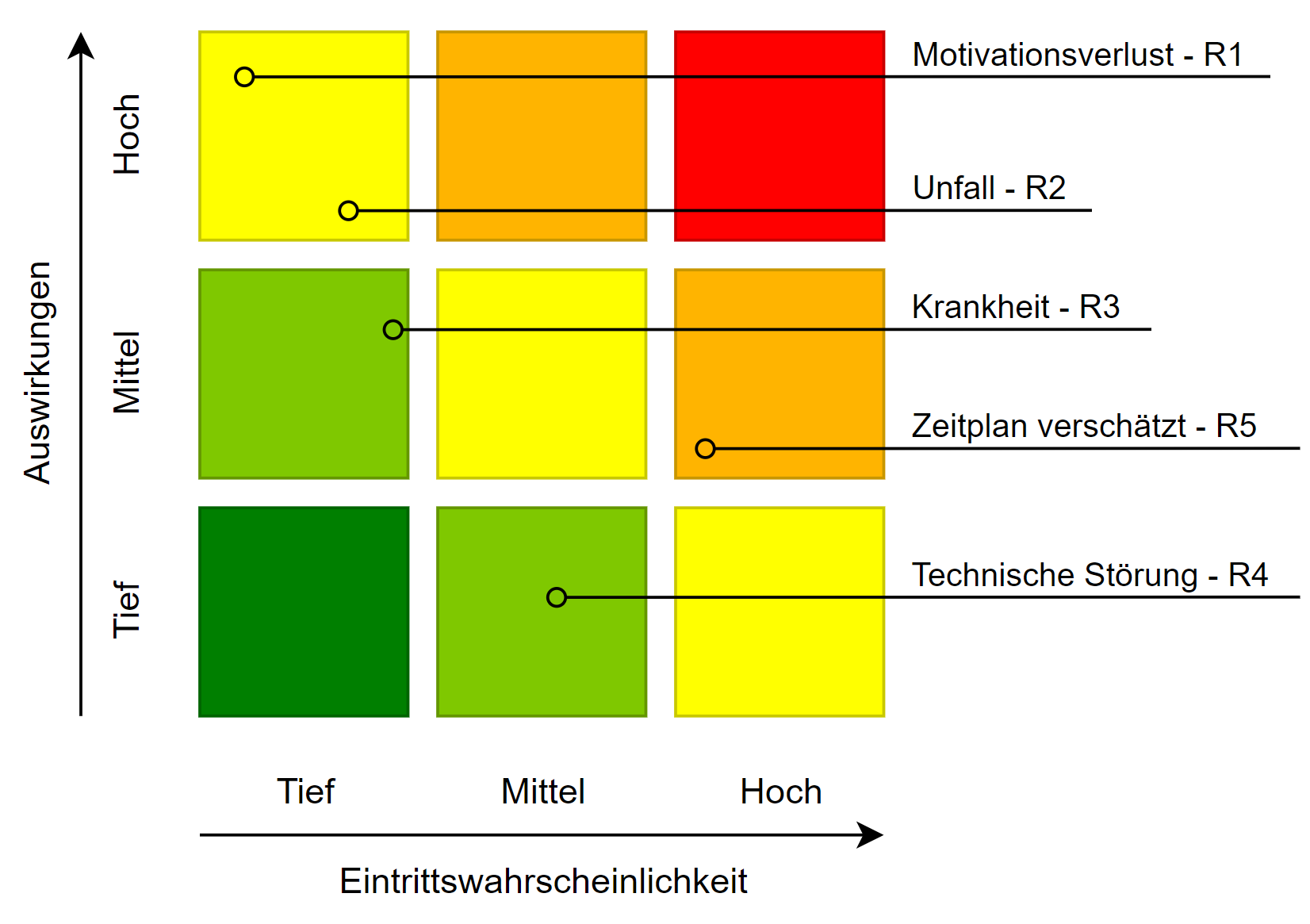


Abbildung 3.4 Risikomatrix

* 1. Machbarkeit

Der Planung nach ist der Jira Synchronizer für das interne Leistungserfassungssystem innerhalb des vorgegebenen Zeitrahmens der IPA umsetzbar.

* 1. Zeitplan

Der Zeitplan besteht aus den in den User Stories definierten Tasks. Die Tasks sind gemäss ihrer User Story gefärbt und sollen möglichst der Reihenfolge nach abgearbeitet werden. Die geschätzte Dauer der jeweiligen Tasks enthält auch Zeit für die Dokumentation. Da der Zeitplan in Stundenblöcke aufgeteilt ist, dienen kleinere Tasks ausserdem auch als Puffer, die Minimaldauer eines Tasks im Zeitplan beträgt eine Stunde.

* + 1. Soll-Ist-Vergleich

In einem ersten Schritt erstellte ich zu Beginn meiner IPA die Use Cases, Anforderungen, User Stories und schliesslich den Zeitplan. Wie geplant konnte ich somit den Zeitplan auch am ersten Tag schon abgeben, jedoch hatte ich während des Erstellens des Zeitplans nicht beachtet, dass ich plante während den ersten beiden Tagen der IPA etwas Überzeit zu generieren, um mir allfälligen Stress in späteren Etappen der Arbeit zu ersparen. Das führte dazu, dass sich ab dem zweiten Tag meine geplanten Tage und meine wirklichen Tage um drei Stunden unterschieden, was wiederum zu einer ungewöhnlichen Gestaltung des Ist Zeitplans führte. Dank meiner Überzeiten während der ersten beiden Tage war ich schon nach dem zweiten Tag vor meinem ursprünglichen Zeitplan.

Die Projektinitialisierung verlief um einiges schneller als im Zeitplan geplant, was unter anderem der Mindestgrösse von einer Stunde pro Task zu verdanken ist, da einige der im Zeitplan erfassten Tasks in Praxis sehr schnell realisiert werden konnten. Dank den grosszügig geschätzten Tasks und des bisher generierten Vorsprungs konnte ich meinen Vorsprung dem Zeitplan gegenüber gegen Ende des dritten Tages auf fast einen kompletten Tag aufbauen.

Auch während des Abfragens der Jira REST API traf ich auf keine Schwierigkeiten, was mich positiv überraschte, und mit meinem bisherigen Vorsprung fühlte sich die Arbeit um einiges entspannter und stressfreier an. Ich konnte mich darauf konzentrieren die Dokumentation zu erweitern und sorgfältig Unit Tests zu schreiben. Immer noch am vierten Tag entschloss ich mich damit zu beginnen, dass die von Jira importierten Daten in der LEIS Datenbank verbucht werden. Dieser Task dauerte eine Stunde weniger lang als geplant, was hiess, dass ich tatsächlich noch am selben Tag damit fertig wurde und nun dem Zeitplan einen vollen Tag voraus war.

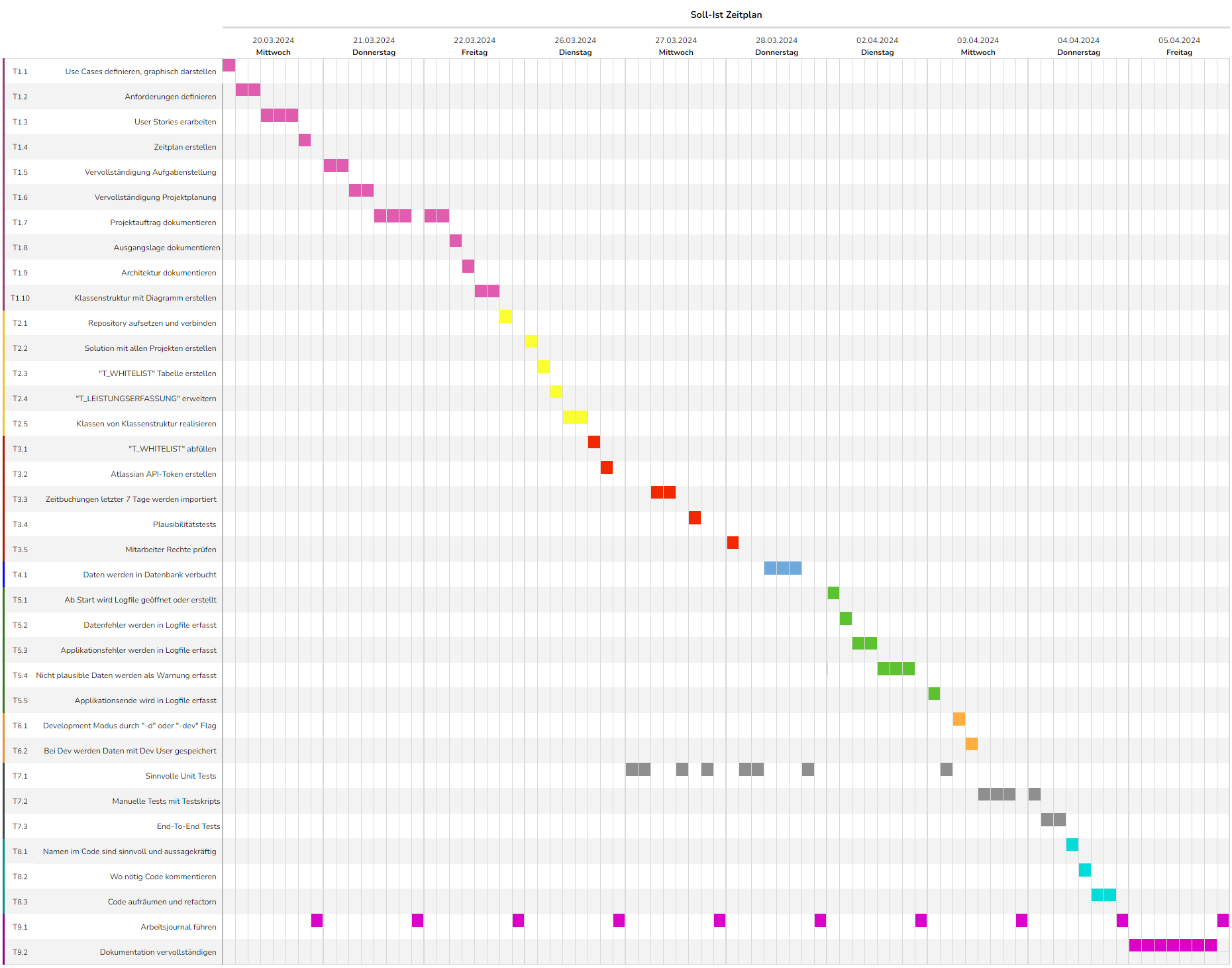
Die Applikation konnte ich nach einer Episode Refactoring und einem neuen Aufbauen der Architektur am sechsten Tag grösstenteils abschliessen, wobei mir während der nächsten paar Tage immer wieder kleine Verbesserungen in den Sinn kamen, welche ich während der Dokumentation als Abwechslungen realisieren konnte. Insbesondere Tag sieben fühlte sich etwas träge an und war sehr Dokumentationslastig, weshalb ich mich über die Ablenkung im Code freute.

Während der letzten Tage entschied ich mich vom Zeitplan abzuweichen und die Vervollständigung der Dokumentation als Hauptpriorität zu setzen, und die Finalisierung der Applikation als Abwechslungsblöcke einzusetzen, um mir eine gewisse Vielfalt im Endspurt der IPA zu garantieren. TODO: Letzte Tage fancy und so :3

* + 1. Fazit

TODO

Zeitplan mit finalem Zeitplan ersetzen! TODO



Ein Bild, das Text, Screenshot, Farbigkeit enthält.

Automatisch generierte Beschreibung

Abbildung 3.6 – 2 Zeitplan Legende

1. Arbeitsjournal

Tag 1 – Projektplanung (Teil 1)

|  |  |
| --- | --- |
| **Datum** | 20.03.2024 |
| **Projektstatus** | ⦁ Im Zeitplan |
| **Überzeiten** | 1.5h |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Ich habe heute Morgen mit meiner IPA begonnen. Zunächst spielten meine Nerven etwas verrückt, doch nach einer Stunde Orientierung und Einarbeitung fand ich meinen Rhythmus ziemlich schnell und konnte innerhalb des Zeitplans weiterarbeiten.  In meiner nervöseren Anfangsphase setzte ich das Github Repository auf, wobei ich bei der Zeitplanung vergass dieses am Anfang aufzuschreiben und ich nun gemäss Zeitplan einen Task, welcher erst auf Freitag geplant war heute schon erledigt habe.  Als nächstes definierte ich den Use Case meiner Applikation und setzte diesen mittels einer einfachen Grafik um. Danach definierte ich meine Anforderungen welche ich der detaillierten Aufgabenstellung sowie den individuellen Bewertungskriterien entnahm. Aus der detaillierten Aufgabenstellung konnte ich schliesslich die User Stories erarbeiten, und aus diesen wiederum meinen Zeitplan gestalten. Mit der Gestaltung des Zeitplans war ich längere Zeit unzufrieden, dies und die Gestaltung des Kanban Boards als Teil der Zeitplanung führten dazu, dass die Erstellung des Zeitplans länger dauerte als erwartet. |
| **Reflexion** | Insbesondere zu Beginn des Tages war ich sehr zerstreut und die vielen komplexen Arbeiten, die mich noch erwarteten, überforderten mich etwas. Ich entschied mich die Arbeit langsam und bestimmt anzugehen, und nach etwas Einarbeitungszeit konnte ich mich entspannen und die Arbeit auch tatsächlich geniessen.  Ich habe es mir zu Ziel gesetzt zu Beginn der Arbeit etwas Überzeit zu generieren damit ich es gegen Ende der Arbeit etwas entspannter angehen kann und die trockeneren ersten Schritte (Dokumentation) schneller erledigt habe. Dieser Plan hat gut funktioniert, wobei ich mir unsicher war wie ich ihn im Zeitplan widerspiegeln sollte, als solches sind alle Tage immer noch in acht Stunden unterteilt. |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Abbildung 4 – 1 Kanbanboard Tag 1

Tag 2 – Projektplanung (Teil 2)

|  |  |  |
| --- | --- | --- |
| **Datum** | 21.03.2024 | |
| **Projektstatus** | | ⦁ Vor dem Zeitplan |
| **Überzeiten** | | 1.5h |
| **Ungeplante Arbeiten** | | Keine |
| **Tagesablauf** | | Auch heute bin ich etwas nervös in den Tag gestartet, jedoch um einiges entspannter als gestern. Als erstes habe ich die Aufgabenstellung dokumentiert, wobei ein Grossteil davon aus der detaillierten Aufgabenstellung von PkOrg entnommen wurde. Die Aufgabenstellung konnte schneller erledigt werden als erwartet, ich bin aktuell dem Zeitplan etwas voraus. Der bisherige Vorsprung im Zeitplan konnte mit der Dokumentation des Projektauftrages etwas ausgebaut werden, was mich sehr gefreut hat, da mir aufgefallen ist, dass ich gewisse Teile der Dokumentation im Zeitplan nicht aufgeführt habe (Schlusswort zum Beispiel), während der Dokumentation der Ausgangslage ging jedoch wieder etwas Zeit verloren, jedoch war ich auch nach der Dokumentation der Architektur noch vor dem Zeitplan. |
| **Reflexion** | | Gegen Ende des Tages verliess mich die Motivation weiter an der Dokumentation zu arbeiten etwas, was sich denke ich auch in der Dokumentation widerspiegelt. Gegen Ende der IPA werde ich die Dokumentation der Ausgangslage sowie die der Architektur nochmals durchlesen und möglicherweise nachbessern müssen. |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Abbildung 4 – 2 Kanbanboard Tag 2

Tag 3 – Projektplanung (Teil 3) und Projektinitialisierung

|  |  |
| --- | --- |
| **Datum** | 22.03.2024 |
| **Projektstatus** | ⦁ Vor dem Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Während der Nacht von gestern auf heute ist mir aufgefallen, dass ich nicht alles, was zur Dokumentation gehört im Zeitplan erfasst habe, das heisst ich muss heute definitiv weiter gut Fortschritt machen. Ich konnte die Architektur heute mit einem Klassendiagramm abschliessen. Heute durfte ich endlich die ersten Schritte in Richtung Programmieren machen; Ich erstellte meine Solution mit allen nötigen Projekten und bereitete die Datenbank auf die weitere Entwicklung vor, dieser Vorgang dauerte ungefähr eine Stunde, was mir etwas mehr Vorsprung vor dem Zeitplan gab, als ich jedoch die Klassen aus dem Klassendiagramm in der Applikation umsetzen wollte, musste ich im Datenbankkontext der Applikation ein Mapping einbauen, in diesem Mapping wurden die bestehenden Entitäten auf ihre entsprechenden Tabellen in der Datenbank gemappt. Ich entschied mich heute, meinen Vorsprung dem Zeitplan gegenüber zu nutzen, um den Kriterienkatalog mit den Standardkriterien nochmals durchzugehen, das Abbildungsverzeichnis und Glossar auf den neusten Stand zu bringen, sowie die Quellen zu aktualisieren. Ab nächster Woche kann ich endlich anfangen richtig zu programmieren, die Vorfreude ist gerade riesig. Ich bin sehr zufrieden habe ich meinen bisherigen Vorsprung für die Dokumentation aufgebracht, ich glaube ich konnte mir dadurch vielen künftigen Stress ersparen. |
| **Reflexion** | Ich fand es schwierig, mich zu entscheiden welche Ausdrücke im Glossar erwähnt werden sollten und welche nicht, das führte dazu, dass das Glossar eher auf der aufgeblähten Seite ist und wahrscheinlich einige Ausdrücke darin vorkommen die nicht hingehören. |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Tag 4 – Erste API-Abfragen

|  |  |
| --- | --- |
| **Datum** | 26.03.2024 |
| **Projektstatus** | ⦁ Vor dem Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Als erstes füllte ich heute ein paar Projekte in meine «T\_WHITELIST» Tabelle in der LEIS Datenbank. Anhand dieser Tabelle werden alle erlaubten Projekte von der Jira REST API importiert. Danach konnte ich einen API-Token erstellen, welcher es mir erlaubt, die Jira API überhaupt anzusteuern. In einem weiteren Schritt wurden testweise alle Zeiterfassungen der erfassten Projekte über die letzten sieben Tage importiert und in der Konsole ausgegeben. Für diesen Zweck mussten erst die Daten der «T\_WHITELIST» Tabelle ausgelesen werden, also eine Verbindung mit der LEIS Datenbank erstellt werden, und auf Grund dieser Projekte die verschiedenen URL angefragt werden.  Ursprünglich hatte ich geplant die Daten per Controller und Repository anzufragen, mir ist jedoch aufgefallen, dass das nicht funktioniert, da mein Projekt kein Web-Projekt ist. Glücklicherweise konnte ich auch schon Erfahrungen mit reinen Konsolenapplikationen sammeln, und konnte so das Hindernis ohne grosse Zeitverluste umgehen. Ich weiss nicht, ob meine Lösung wie sie aktuell steht besonders schön ist, ich bin aktuell einfach froh habe ich eine gefunden. Als Resultat dieser Realisation kann ich alle meine bisher erstellten Repositories löschen, was etwas schade ist, mich aber nicht wirklich zurücksetzt.  Die Jira Zeiterfassungen verfügen lediglich über ein Startdatum, vorerst wird demnach in den Leistungserfassungen das Enddatum gleich dem Startdatum gesetzt. Mir ist aufgefallen, dass ich von den Jira Zeiterfassungen lediglich die E-Mail-Adresse des verfassenden Benutzers, das Startdatum der Zeiterfassung, die geleistete Zeit, und allfällige Kommentare abfrage, das heisst, dass Plausibilitätstests an dieser Stelle wenig Sinn machen. Was eingebaut werden muss ist eine Warnung im Log sollte eine Zeiterfassung über 24 Stunden betragen und ein Test, ob der User Rechte hat auf das Konto zu buchen, das heisst der Plausibilitätstest Schritt des Zeitplans wird hier übersprungen.  Die anonymisierte Datenbank bereitete mir beim Test, ob Mitarbeiter berechtigt sind auf ein Projekt zu buchen etwas Schwierigkeiten, da ich meine Funktionen nicht wirklich testen konnte. Der Ablauf dieses Testes wäre wie folgt:  Die Applikation schaut, ob die E-Mail-Adresse erfassten Mitarbeiters in der «T\_USER» Tabelle vorhanden ist, falls dem so ist, wird die Id des Mitarbeiters aus der Tabelle genommen, und es wird geschaut, ob sie in der Zwischentabelle «TZ\_PROJEKT\_MITARBEITER» auf die Id des Projektes gemappt ist. Stimmt das, hat der Mitarbeiter die nötigen Rechte, um auf das Projekt zu buchen. Da jedoch die Datenbank anonymisiert ist, werden die Rechte schon bei Schritt eins abgelehnt, ich musste also temporäre Fake-User anlegen, welche die E-Mail-Adressen, die in den Zeiterfassungen hinterlegt waren, enthalten.  Ausserdem kam noch hinzu, dass ich versuchte Test Driven zu programmieren (also zuerst Unit Tests schreiben, dann den Code), was sich als ziemlich schwierig herausstellte, da ich mehrere Unit Tests hatte, welche erst mit den Logs Sinn machen würden und ich noch nicht genau wusste wie ich diese umsetzen würde. Aus diesem Grund entschied ich mich, diese Tests so weit auszuschreiben wie zu diesem Zeitpunkt möglich, sie jedoch nicht komplett fertig zu schreiben (also ohne Log Testing) und später zu ihnen zurückzukehren. |
| **Reflexion** | Ich bin heute umso zufriedener entschied ich mich die ersten zwei Tage etwas Überzeit zu generieren, ich bin noch immer mit gemütlichem Vorsprung vor dem Zeitplan, das erlaubt es mir, mich mehr auf die Dokumentation zu konzentrieren, ohne bei meiner Applikation in den Stress zu geraten, oder umgekehrt.  Ich habe heute sehr viel programmiert und viel Fortschritt gemacht, jedoch glaube ich, dass es sich lohnen wird den Code gegen Ende nochmals schön zu überarbeiten. Es funktioniert alles wie es soll, ich bin jedoch überzeugt, dass schönere Lösungen existieren (insbesondere im Bezug darauf, wie ich mit den Json Daten umgehe, welche ich von der Jira API erhalte). |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Tag 5 – Datenbank Einträge und Logging

|  |  |
| --- | --- |
| **Datum** | 27.03.2024 |
| **Projektstatus** | ⦁ Vor dem Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Nachdem ich fertig war, zu prüfen, ob Mitarbeiter auf ihre jeweiligen Projekte die benötigten Rechte verfügen, begann ich die Daten, welche ich von der Jira REST API importierte auf meine Datenbank zu schreiben. Mir fiel hierbei auf, wie lange es dauerte, bis die Applikation ihre Daten fertig importiert hat. Ich versuchte meinen Code zu optimieren, erzielte jedoch keine begeisternden Ziele und die Laufzeit der Applikation betrug noch immer mehrere Sekunden. Während ich versuchte die Wartezeiten zu optimieren, entschied ich mich auch gleich mein Projekt zu überarbeiten und aufzuräumen. Es stellte sich heraus, dass das ein ziemlich grosser Aufwand wurde, aber jetzt habe ich meine saubere Architektur wieder hergestellt.  Nach meinen Aufräumaktionen beschloss ich, das Logging für die Applikation zu realisieren. Während dieses Prozesses fand ich heraus, weshalb meine Applikation so langsam läuft; Die Jira API antwortet um einiges langsamer als ich erwartet hätte. |
| **Reflexion** | Es hat mir sehr viel Spass gemacht den Grossteil der Applikation zu realisieren. Aktuell fehlt nur noch der Development Modus, wobei ich definitiv auch noch über meine Unit Tests gehen sollte. Ich entwickle zwar Test Driven, wenn ich die Tests aber geschrieben habe, gehen sie oft vergessen. |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Tag 6 – Testing und Development Modus

|  |  |
| --- | --- |
| **Datum** | 28.03.2024 |
| **Projektstatus** | ⦁ Vor dem Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Als erstes entschied ich mich heute die Unit Tests für die Applikation zu überarbeiten, was dazu führte, dass ich einen Grossteil davon neu schrieb, weil ich die Tests während der Entwicklung nicht richtig mitführte und insbesondere als ich die Architektur neu baute einige davon obsolet wurden oder so wie sie geschrieben waren nicht mehr optimal auf die Applikation anwendbar waren. Ich entschied mich bei den Tests möglichst jede öffentliche Methode zu testen, gemockt wurden hier die Jira API sowie die Repositories der Applikation, sodass bei den Methoden, welche die Jira API oder ein Repository aufrufen einfach getestet wurde, ob sie die Daten überhaupt abfragen und wenn ja, ob sie die Daten im erwarteten Format ausgeben.  Nach der Entwicklung der Unit Tests ergänzte ich die Applikation um einen Development Modus, welcher über ein Flag beim Aufrufen der Applikation eingestellt werden kann. Im Development Modus werden alle Daten auf einen in den Appsettings definierten User gespeichert, statt auf die User, welche die Daten in der Jira API tatsächlich abgespeichert haben. |
| **Reflexion** | Ich hätte die Unit Tests eigentlich von Anfang an immer wieder aktualisieren sollen, statt sie ganz zu Beginn einmal zu schreiben und erst jetzt wieder auf den aktuellen Stand zu bringen.  Trotz der suboptimalen Art wie ich die Unit Tests durchgeführt habe bin ich heute sehr gut vorangekommen, ein Vorteil davon die Tests separat von der eigentlichen Entwicklungsarbeit zu gestalten war definitiv, dass ich mir Zeit ersparen konnte und mich auf eine Sache voll konzentrieren konnte. Ich plane mir künftig trotzdem anzugewöhnen, Tests parallel zur Entwicklung durchzuführen, denn ich glaube, dass ich mit genügend Übung die Tests ohne Zeitverluste parallel zum restlichen Code entwickeln kann. |

**Aufgabenliste**

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

Automatisch generierte Beschreibung

Tag 7 – Dokumentation

|  |  |
| --- | --- |
| **Datum** | 02.04.2024 |
| **Projektstatus** | ⦁ Vor dem Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | Als erstes vervollständigte ich heute Morgen das Arbeitsjournal des letzten Tages, da ich dieses erst gegen Tagesende begonnen habe und grösstenteils lediglich Stichworte erfasst hatte.  In einem nächsten Schritt ging ich die bisherige Dokumentation nochmals durch, dabei erweiterte ich die Kurzfassung des Projektes um die Ergebnisse der Entwicklung und führte manuelle Tests durch, welche ich ebenfalls noch dokumentierte.  Im späteren Vormittag dokumentierte ich endlich das Logging der Applikation, was ich schon etwas aufgeschoben hatte. |
| **Reflexion** | [Kritische Würdigung Arbeit & Vorgehensweise] |

**Aufgabenliste**

Tag 8 –

|  |  |
| --- | --- |
| **Datum** | TT.MM.JJJJ |
| **Projektstatus** | ⦁ Im Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | [Erfolge / Misserfolge] |
| **Reflexion** | [Kritische Würdigung Arbeit & Vorgehensweise] |

**Aufgabenliste**

Tag 9 –

|  |  |
| --- | --- |
| **Datum** | TT.MM.JJJJ |
| **Projektstatus** | ⦁ Im Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | [Erfolge / Misserfolge] |
| **Reflexion** | [Kritische Würdigung Arbeit & Vorgehensweise] |

**Aufgabenliste**

Tag 10 – Abgabe

|  |  |
| --- | --- |
| **Datum** | TT.MM.JJJJ |
| **Projektstatus** | ⦁ Im Zeitplan |
| **Überzeiten** | Keine |
| **Ungeplante Arbeiten** | Keine |
| **Tagesablauf** | [Erfolge / Misserfolge] |
| **Reflexion** | [Kritische Würdigung Arbeit & Vorgehensweise] |

**Aufgabenliste**

Teil 2 – Projekt

1. Kurzfassung

Ausgangslage

Die Softec verwendet Jira als Projektmanagement Tool, Zeitbuchungen können direkt auf dem Kanban Board von Jira vorgenommen werden. Aktuell müssen Mitarbeiter ihre Zeitbuchungen jedoch auf Jira sowie als auch im internen Leistungserfassungssystem (LEIS) vornehmen, was bei potenziell mehreren Dutzend Leistungen pro Mitarbeiter pro Woche einen beträchtlichen Mehraufwand bedeutet.

Das alte LEIS wird durch eine Neuentwicklung abgelöst, im Rahmen dieser Neuentwicklung sind einige Wünsche und Vorschläge aufgekommen, einer davon ein Tool zur automatischen Synchronisierung von auf Jira verbuchten Leistungen. Der Jira Synchronizer für das interne Leistungserfassungssystem würde den Mitarbeitern dabei einiges an Zeitaufwand ersparen.

Umsetzung

Der Jira Synchronizer wird als Kommandozeilenapplikation realisiert, diese Kommandozeilenapplikation kann automatisch von einem Scheduler aufgerufen werden, und soll auf künftig auf einem passenden Server laufen.

Der Jira Synchronizer importiert Buchungen von der Jira REST API, wobei berücksichtigt wird, dass eine Buchung noch nicht in der LEIS Datenbank erfasst wurde, hinterlegte Mitarbeiter die Rechte auf die hinterlegten Projekte besitzen und keine nicht plausiblen Daten abgespeichert werden.

Es soll ein Log geführt werden welchen Start und Schluss der Applikationsinstanz festhält. Weiterhin sollen Warnungen für unerwartete Daten vermerkt werden sowie Fehler für ungültige Daten.

Ergebnis

Die Applikation «Jira Synchronizer für das interne Leistungserfassungssystem» konnte innerhalb des Zeitplans entwickelt werden. Das resultierende executable kann planungsgemäss mit einem Scheduler mit oder ohne Flags aufgerufen werden, wobei wichtig ist dass die appsettings im Ordner in welchem die Applikation aufgerufen wird vorhanden sind.

// TODO: TOTAL NICHT MEHR ALS 1 SEITE

1. Ausgangslage
   1. Einleitung

Der Jira Synchronizer soll eine eigenständige Konsolenapplikation sein, welche ab Aufruf Daten von der Jira REST API importiert und in der LEIS Datenbank einträgt, damit diese nicht in zwei Systemen manuell eingegeben werden müssen. Das Ziel ist es, die Applikation auf dem Datenbankserver per Scheduler in regulären Abständen laufen zu lassen.

* 1. Jira

Die Softec verwaltet ihre Kanbanboards auf Jira, Leistungen werden direkt auf den bearbeiteten Tasks im Worklog erfasst mitsamt Zeit und Beschreibung. In einem Task ist erfasst, zu welchem Projekt der Task gehört, und in welchem Konto im LEIS die Leistung erfasst werden soll.

Die Jira REST API kann mit gültigem API-Token angesteuert werden und über die Ids der Tasks kann auf deren Daten zugegriffen werden.

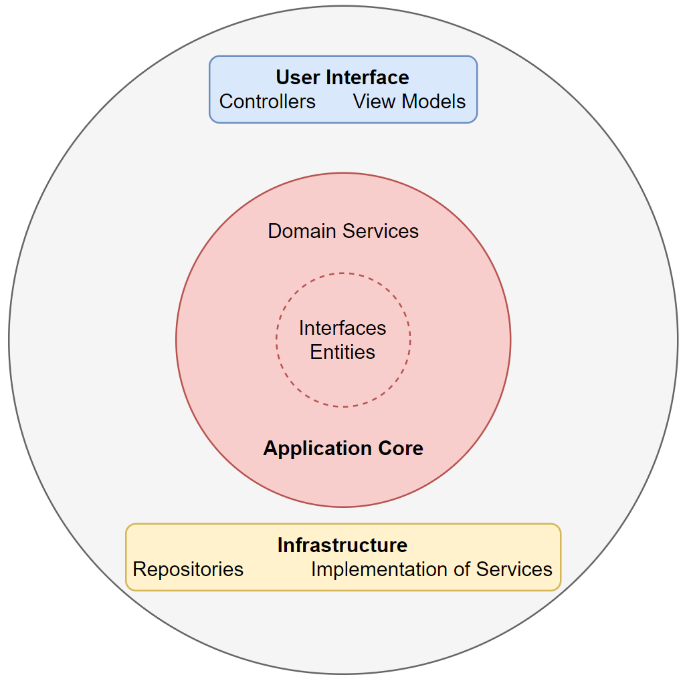
* 1. LEIS Datenbank

Eine anonymisierte Version der LEIS Datenbank wird für die Umsetzung der IPA zur Verfügung gestellt, hier ist wichtig zu erwähnen, dass die Datenbank über keine Foreign Keys verfügt, alle Beziehungen sind per Datenbanktrigger geregelt. Nebenstehendes Datenbankschema ist auf die für die IPA relevanten Tabellen begrenzt, Tabellen und Spalten, welche noch in der Datenbank ergänzt werden müssen, sind blau markiert, die für die IPA irrelevanten Spalten sind ausgegraut. Die Namensgebung neuer Spalten und Tabellen ist der Namensgebung der alten Datenbanktabellen und Spalten nach empfunden da unterschiedliche Naming Conventions bei künftiger Wartung verwirrend sein könnten.

Abbildung 6.3 LEIS Datenbankschema

1. Architektur
   1. Einleitung

In diesem Kapitel wird die Architektur und der Aufbau der Applikation erklärt. Allfällige Anpassungen während der Implementation werden in der relevanten Implementationsdokumentation festgehalten.

* 1. Clean Architecture

Bei dieser Applikation wird nach dem Clean Architecture Prinzip vorgegangen.

Clean Architecture stellt die Geschäftslogik und das Anwendungsmodell in den Mittelpunkt der Applikation. Anstatt dass die Geschäftslogik vom Datenzugriff oder anderen Infrastrukturproblemen abhängt, hängen Infrastruktur- und Implementationsdetails bei der Clean Architecture vom Kern der Applikation ab. Dies wird durch die Definition von Abstracts oder Interfaces im Kern der Anwendung erreicht, die dann durch Typen in der Infrastrukturschicht implementiert werden. Eine gängige Methode zur Visualisierung dieser Architektur ist der sogenannte Onion View.[4]

* 1. Schichtentrennung

Abbildung 7.2 Clean Architecture Onion View

Da ich nach Clean Architecture vorgehe, kann ich meine Applikation sehr einfach in verschiedene Schichten einteilen.

* + 1. Core

Das Core Projekt übernimmt bei mir die Rolle des Application Cores, das heisst hier werden Entitäten, Interfaces, Data Transfer Objects (fortan auch DTO genannt) und Services gespeichert. Die Entitäten werden hier aus den Tabellen der Datenbank generiert, sie sollten sich kaum oder überhaupt nicht verändern, solange sich die Datenbank nicht verändert. DTOs sind Objekte, mit welchen im Application Core gearbeitet wird, sie müssen nicht alle Daten der Entities enthalten und können auch über Methoden verfügen.

* + 1. Infrastructure

Im Infrastructure Projekt werden die Repositories hinterlegt mit welchen der Datenzugriff auf die Datenbank ermöglicht wird, und es werden Services implementiert, welche die im Application Core definierten Interfaces verwenden.

* + 1. Application

Das Application Projekt ist in dieser Applikation das Gegenstück zum User Interface in der Onion View. Im Application Projekt sind Controller, View Models und das Program.cs vorhanden. Die Controller kommunizieren mit der Infrastructure, um Daten zu manipulieren, die View Models sind Klassen, welche nur diese Informationen enthalten, welche während der Laufzeit der Applikation auch wirklich benötigt werden.

Die Konfiguration der ganzen Applikation findet im Program.cs statt.

* + 1. Unit Tests

Da die Applikation exklusiv in C# geschrieben ist können Unit Tests im ganzen Programm verwendet werden. Insbesondere im Core Projekt können einfach automatische Unit Tests geschrieben werden, da das Core Projekt keine äusseren Abhängigkeiten aufweist. Auch das Infrastructure und Application Projekt können gut mit Unit Tests abgedeckt werden, wobei äussere Abhängigkeiten (wie zum Beispiel die Datenbank oder Jira API) gemockt werden müssen.

* + - 1. NSubstitute

Bei NSubstitute handelt es sich um ein NuGet Package, welches das Mocken innerhalb eines Tests erleichtert. Objekte können mit NSubstitute gemockt werden wonach die Rückgabewerte einzelner Methoden der Instanzen definiert werden können.

Hier ein Beispiel:

calculator = Substitute.For<ICalculator>();

calculator.Add(1, 2).Returns(3);

Mit diesen beiden Zeilen wurde definiert, dass die Add Methode der calculator Instanz des Interface ICalculator mit den Eingabewerten 1 und 2 immer 3 zurückgibt.

* + - 1. Fluent Assertions

Fluent Assertions wurde für schönere und lesbarere Tests entwickelt. Fluent Assertions ermöglicht es, den Assert Schritt eines Testes sprachlich flüssig zu formulieren und verhindert mit seiner organischen Ausdrücken Inkonsistenzen zwischen Tests. Als Beispiel wird direkt das gemockte Objekt von NSubstitute verwendet.

result = calculator.Add(1, 2);

result.Should().Be(3);

* 1. Klassendiagramm

Das Klassendiagramm ist in die Application, Infrastructure und Core Projekte aufgeteilt. Das Application Projekt verwendet ViewModels, welche in der Applikation so verwendet werden können wie sie sind, sie werden nicht zur Datenmanipulation verwendet. Des Weiteren verfügt das Application Projekt über Services, welche lediglich die von den ViewModels zur Verfügung gestellten Daten benötigen sowie Controller welche existieren um die Daten zwischen Core Entities und ViewModels zu mappen und auf die Repositories zu schreiben.

Das Core Projekt verfügt über alle Entities, diese Klassen werden zur Manipulation von Daten verwendet. Sie enthalten alle Eigenschaften, welche sie von ihrem Ursprung erhalten (Jira REST API oder LEIS Datenbank). Entitäten werden nicht verändert, sie dienen lediglich dem Transport von Daten. Auch im Core Projekt enthalten ist der Logging Service da dieser auf der ganzen Applikation verfügbar sein muss, und die Interfaces der Repositories über welche Datenmanipulation geschieht.

Das Infrastructure Projekt enthält alle Repositories über welche die Controller des Application Projekts Daten abrufen oder manipulieren können. Ihre Funktionen sind von den Interfaces im Core Projekt abhängig.

Der Leserlichkeit halber wurden im Diagramm die Verbindungen zwischen Klassen und Service Klassen weggelassen.

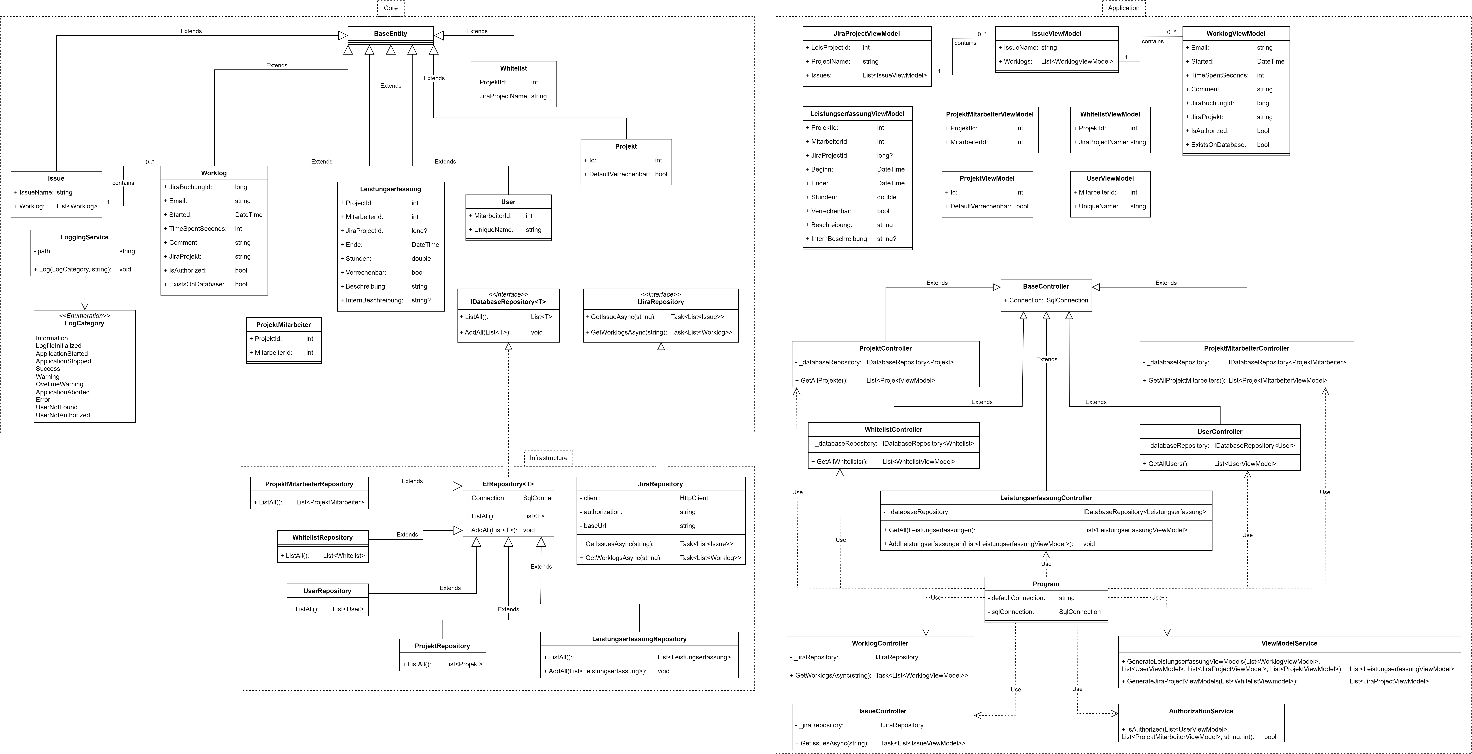


Abbildung 7.4 Klassendiagramm

1. Entwicklung
   1. Erweiterung der Datenbank

Die Datenbank wurde um die Tabelle «T\_WHITELIST» erweitert. Die Applikation entscheidet auf Grund der in dieser Tabelle enthaltenen Einträge, ob die Zeiterfassungen auf ein in Jira vorhandenes Projekt in der Datenbank automatisch geführt werden sollen, oder nicht. Einträge werden während der IPA manuell geführt, mit der Neuentwicklung von LEIS ist jedoch vorgesehen, dass diese Einträge über eine Administratorenseite auf der Webseite getätigt werden.

Des Weiteren wurde die «T\_LEISTUNGSERFASSUNG» Tabelle um die Spalte «LEI\_JIRA\_BUCHUNG\_ID» erweitert. In dieser Spalte erfasst der Jira Synchronizer die Id der Zeiterfassung wie sie von Jira importiert wurde und vergleicht Imports mit bereits vorhandenen Einträgen, um sicher zu stellen, dass kein Eintrag zweimal importiert und in der Datenbank eingefügt wird.

* 1. Klassen Realisierung

Die im Klassendiagramm enthaltenen Klassen wurden in die Applikation eingebaut. Als erstes wurden die Entitäten im Core Projekt erstellt, wobei diese im Datenbankkontext so gemappt werden mussten, dass die Namen mit den Datenbanktabellen übereinstimmten. Die Bezeichnungen der Entitäten und ihrer Eigenschaften wurden anders gewählt als in den Tabellen, da die Leserlichkeit des Codes mit der älteren Naming Convention nicht gegeben war. Im Application Projekt wurde ausserdem das ViewModel erstellt.

* 1. Jira REST API

Die Jira REST API verfügt über eine äusserst detaillierte Dokumentation, welcher leicht entnommen werden konnte, wie einzelne Projekte angesteuert werden konnten.[5] Einige Projekte wurden schliesslich in der «T\_WHITELIST» Tabelle der Datenbank auf beliebige anonymisierte LEIS Konten gemappt. Dabei wurde einfach der Name des Jira Projektes in der Spalte «WHL\_JIRA\_PROJECT\_NAME» hinterlegt und die Id eines «T\_PROJEKT» Eintrages in der Spalte «WHL\_PRJ\_ID». Im weiteren Programmieren wird das dazu führen, dass die Tabelle «T\_WHITELIST» abgefragt werden kann, Projekte, welche auf dieser Tabelle vorhanden sind von der Jira REST API importiert werden können, und Leistungen, welche darauf verbucht sind in der LEIS Datenbank auf dem entsprechenden Projekt verbucht werden können.

In einem nächsten Schritt wurde ein Atlassian API Token für die IPA namens «Jira\_REST\_API\_Token\_IPA» generiert [6]. Da der API-Token nur ein einziges Mal angezeigt wird, wurde dieser im Projektordner zwischengespeichert, jedoch wurde das File mit dem Inhalt des Tokens in ein .gitignore File aufgenommen, damit dieser Token nicht öffentlich ersichtlich ist.

* 1. LEIS Datenbank

Zur Realisierung dieser Applikation wurde eine anonymisierte Version der Datenbank des internen Leistungserfassungssystems zur Verfügung gestellt. Anonymisiert heisst hier einfach, dass alle Personendaten oder Daten, welche zur Identifikation einer Person führen könnten, unwiderruflich kodiert wurden.

Normalerweise ist es mit .NET 8 sehr einfach möglich, Klassen anhand einer Datenbank und ihrer Tabellen erstellen zu lassen. Da die LEIS Datenbank jedoch über einige Sonderheiten verfügt (so gibt es kaum Fremdschlüssel und in manchen Tabellen sogar Datenleichen) wurde entschieden, stattdessen mit dem Microsoft Data SqlClient[7] zu arbeiten. Der SqlClient erlaubt es dem Entwickler direkt in der Entwicklungsumgebung eine Verbindung mit einer Datenbank auf einem SQL-Server aufzubauen, so können auch SQL Queries verwendet werden und Probleme mit fehlenden Fremdschlüsseln oder Datenleichen können ohne Probleme umgangen werden.

* 1. Logging
     1. Log Codes

Die Log Codes wurden durch http Response Codes inspiriert erstellt und als Enum in der Applikation gespeichert. Folgende Codes existieren:

|  |  |  |
| --- | --- | --- |
| **Code** | **Bezeichnung** | **Verwendung** |
| 100 | Information | Eine generelle Information, welche im Logfile vermerkt werden soll. |
| 101 | LogfileInitialized | Das Logfile wurde erstellt. |
| 102 | ApplicationStarted | Die Applikation wurde gestartet. |
| 199 | ApplicationStopped | Die Applikation wurde angehalten. |
| 200 | Success | Daten konnten erfolgreich in der Datenbank gespeichert werden. |
| 300 | Warning | Eine generelle Warnung, welche im Logfile vermerkt werden soll. |
| 301 | OvertimeWarning | Eine in Jira erfasste Leistung geht länger als 24 Stunden. |
| 302 | ApplicationAborted | Die Applikation wurde abgebrochen. |
| 400 | Error | Ein genereller Fehler, welcher im Logfile vermerkt werden soll. |
| 401 | UserNotFound | Der definierte Benutzer konnte in der «T\_USER» Tabelle nicht gefunden werden. |
| 402 | UserNotAuthorized | Dem definierten Benutzer ist es nicht erlaubt auf das definierte Projekt zu buchen. |

* + 1. Vorgehen

Um das Logging für die Applikation umzusetzen, wurde ein Service erstellt, welcher über eine Methode verfügt, welche ein LogCategory enum und eine Nachricht übernimmt. In der LogCategory wurden verschiedene Logging Codes definiert. Generell gibt es vier Hauptcodes, welche verwendet werden, wenn von den spezifischeren Codes keine passen. Diese Hauptcodes sind 100 für Information, 200 für Success, 300 für Warning und 400 für Error. Der Zahlenraum zwischen diesen Codes ist für spezifischere Codes aus derselben Kategorie des letzten Hauptcodes reserviert (alle Codes zwischen 100 und 199 sind zum Beispiel für Informationscodes reserviert). Dieses enum kann nach Bedarf in der künftigen Entwicklung ausgebaut werden.

* 1. Development Modus
     1. Flags

Beim Start überprüft die Applikation die vom User eingegebenen Argumente, falls die Applikation mit den Flags «-dev» oder «-d» aufgerufen wurde, startet sie im Development Modus. Werden der Applikation weitere oder andere Argumente mitgegeben, wird in der Log Datei als Warnung festgehalten, dass die Applikation mit ungültigen Flags aufgerufen wurde.

* + 1. Programmablauf

Der Programmablauf verändert sich im Development Modus nur minim. Wird die Applikation im Development Modus aufgerufen, werden die SQL Queries, welche die von der Jira API erhaltenen Daten in der Datenbank erfassen angepasst, sodass sie ihre Einträge auf den in den appsettings.json definierten User buchen.

1. Testing
   1. Einleitung

Damit die Funktionalität einer Applikation gewährleistet werden kann soll getestet werden. Testing kann zu der Performance einer Applikation beitragen, Entwicklungszeit verkürzen und hilft beim frühen Erkennen von Bugs oder Inkonsistenzen im Code.

* 1. Umfeld

Getestet wird die Applikation zum Grossteil in einem separaten Projekt innerhalb der Solution. Es wurde entschieden eine Sammlung spezifischer Libraries und Erweiterungen zu verwenden um das Testen so natürlich und leserlich wie möglich zu gestalten.

* + 1. Mittel und Methoden

Die Applikation verfügt über ein xUnit Test Projekt. xUnit[8] ist das führende Testing Framework in C# und .NET. Tests in xUnit sind entweder als Fact oder als Theory aufgebaut. Facts sind Tests, in welchen eine Methode mit nur einem Datensatz getestet wird, als Theory wird eine Testmethode bezeichnet, wenn ihr mehrere Datensätze gegeben werden, um beispielsweise mehrere Edge Cases abzudecken.

Die xUnit Testklassen werden des Weiteren um NSubstitute und fluent Assertions ergänzt. NSubstitute[9] ist ein simpler, entwicklerfreundlicher Ersatz für andere Mocking Libraries. Mit NSubstitute können Methoden, Klassen, oder Third Party Software innerhalb eines Tests gemockt werden. fluent Assertions[10] ist eine Sammlung an .NET-Erweiterungen, welche es dem Entwickler erlauben natürliche Sprache in seinen Tests zu verwenden.

* 1. Konzept

Es werden hauptsächlich automatisierte Unit Tests verwendet, um die Applikation so vollständig wie möglich zu testen. Am Ende der Entwicklungszeit werden jedoch zusätzlich End-to-end Tests durchgeführt, diese sollen gewährleisten, dass die Applikation auch als Ganzes funktioniert.

* + 1. Automatisierte Unit Tests

Da die Applikation eine reine Konsolenapplikation ist und über kein Frontend verfügt, sollte es möglich sein fast den ganzen Sourcecode mit sinnvollen automatischen Unit Tests abzudecken. Ausnahmen beinhalten gewisse private Methoden welche in den End-to-end Tests abgedeckt werden, sowie gewisse Interaktionen mit externen Schnittstellen (LEIS Datenbank und Jira API).

Die Unit Tests sind immer gleich aufgebaut; Der Name der Testmethode ist gegliedert in den Methodennamen, welchen sie testet, was die Methode macht, und was das Resultat sein soll (Methode\_\_Macht\_\_SollResultat). Innerhalb des Tests ist die Methode in Arrange, Act, Assert gegliedert. Im Arrange Teil werden hier alle Variablen, welche im Test verwendet werden, instanziiert, im Act Teil werden Methoden ausgeführt, und im Assert Teil wird getestet, ob das Resultat den erwarteten Wert beinhaltet.

* + 1. End-to-end Tests

End-to-end Tests testen die Applikation als Ganzes, um End-to-end Tests korrekt durchführen zu können, müssen die Datenbank sowie alle erforderlichen Dateien (App.config, appsettings.json) vorliegen und mit korrekten Daten befüllt sein. Mit End-to-end Tests wird getestet, ob die ganze Applikation mit Daten wie sie in der Produktion vorhanden sind so funktioniert wie erwartet. Ein End-to-end Test verfügt über Vorbedingungen, welche erfüllt sein müssen, dass es überhaupt Sinn macht den Test auszuführen, sowie ein Verfahren, welchem die Testperson folgen kann, um zu schauen, ob die Applikation so läuft wie sie soll. Das Verfahren soll als grobes Testhandbuch dienen, die Testperson soll nicht zuerst herausfinden müssen wie die Tests genau zu vollziehen sind, bevor sie überhaupt den Test ausführen kann.

* 1. Testprotokoll
     1. Unit-Tests

Um die Lesbarkeit des Testcodes zu erleichtern, werden Listen, welche in Tests vorkommen, hier ohne ihre Daten aufgeführt. Listen, die im Code über Daten verfügen, welche hier weggelassen wurden, sind am Ende ihrer Initialisierung mit drei Punkten markiert und grau hinterlegt.

* + - 1. Durchführung der Tests

Es wurden für jede Methode in jedem Service und jedem Controller je mindestens ein Test ausgeführt, wobei versucht wurde sowohl Edge Cases wie auch normale Use Cases abzudecken. Total wurden 14 Tests erfolgreich ausgeführt, wobei Repositories und die Jira API jeweils gemockt wurden. Wichtig zu beachten ist, dass Tests, welche den Logging Service verwenden, einer nach dem anderen ausgeführt werden müssen, da sie, weil sie beim Testing parallel laufen, sich gegenseitig blockieren können.

* + 1. End-to-end Tests

Generelle Vorbedingungen

* Um End-to-end Tests durchzuführen, muss eine dem Datenbankdiagramm entsprechende Datenbank vorhanden sein
* Eine Datei namens «App.config» soll im Root Verzeichnis des «Application» Projekts der Applikation vorhanden sein. In dieser Datei ist der Connection String der Datenbank sowie der API-Token der Jira API vorhanden. Die Datei ist folgendermassen aufgebaut:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<connectionStrings>

<add name="default" connectionString="Server=SERVER;Database=DATABASE;User Id=USER;Password=PASSWORD;TrustServerCertificate=True" providerName="System.Data.SqlClient"/>

</connectionStrings>

<appSettings>

<add key="api\_token" value="USER:TOKEN" />

</appSettings>

</configuration>

* + - 1. Development Modus

Vorbedingungen

* Eine Datei namens «appsettings.json» soll im Root Verzeichnis des «Application» Projekts der Applikation vorhanden sein. In dieser Datei ist die Id des Benutzers, auf welchen die Dateien der Jira API, während die Applikation im Development Modus läuft, gebucht werden sollen, definiert. Die Datei ist folgendermassen aufgebaut:

{

"DevUser": ID\_AS\_NUMBER

}

Verfahren

Um den End-to-end Test durchzuführen kann einfach die Applikation per Kommandozeile mit dem Flag «-dev» oder «-d» aufgerufen werden. Hier ist wichtig zu notieren, dass nur Projekte welche in der «T\_WHITELIST» Tabelle vorhanden sind synchronisiert werden, und kein Mitarbeiter mit der im appsettings.json definierten Id existiert ein Fehler aufkommen kann.

Die Resultate können anschliessend in der Datenbank in der Tabelle «T\_LEISTUNGSERFASSUNG» überprüft werden.

* + - 1. Default Modus

Vorbedingungen

* Alle Benutzer welche auf Jira über Buchungen innerhalb von Projekten welche in der «T\_WHITELIST» Tabelle in der LEIS Datenbank erlaubt wurden verfügen sollen in der «T\_USER» Tabelle in der LEIS Datenbank erfasst sein.

Verfahren

Im Default Modus kann die Applikation ohne weitere Argumente aufgerufen werden. Im Default Modus werden ebenfalls lediglich Projekte welche in der «T\_WHITELIST» Tabelle vorhanden sind synchronisiert. Sind die Daten eines Users in der «T\_USER» Tabelle nicht vorhanden, wird eine 401 Fehlermeldung im Logfile erfasst.

Die Resultate können anschliessend in der Datenbank in der Tabelle «T\_LEISTUNGSERFASSUNG» überprüft werden.

* 1. Testreport
     1. Resultatübersicht
        1. End-to-end Test – Development Modus

Testspezifikation

Testende Person

Simon André Lötscher

Beschreibung

Der Test wurde wie definiert durchgeführt, die Id des Testusers in appsettings.json wurde auf dieselbe Id gesetzt wie die Id des Logins auf der LEIS Testumgebung sodass die Resultate nach Ausführung des Testes auf der Testumgebung direkt nachgeprüft werden können, statt dass diese in der Datenbank überprüft werden müssten.

Erwartetes Ergebnis

Es werden alle noch nicht erfassten Leistungen der letzten sieben Tage der in der «T\_WHITELIST» Tabelle vorhandenen Projekte synchronisiert, als Mitarbeiter wird der Benutzer mit der in der appsettings.json definierten Id hinterlegt.

Ergebnis

Alle Leistungen wurden wie erwartet erfasst und es wurde der richtige Mitarbeiter hinterlegt.

Massnahmen

Keine.

Resultat

Erfolgreich.

* + - 1. End-to-end Test – Default Modus

Testspezifikation

Testende Person

Simon André Lötscher

Beschreibung

Der Test wurde wie definiert durchgeführt, es wurde ein temporärer Benutzer in der «T\_USER» Tabelle eingefügt mit der E-Mail-Adresse eines auf Jira vorhandenen Mitarbeiters, um das Testen des Default Modus trotz anonymisierter Datenbank zu ermöglichen.

Erwartetes Ergebnis

Es werden alle noch nicht erfassten Leistungen der letzten sieben Tage der in der «T\_WHITELIST» Tabelle vorhandenen Projekte synchronisiert, falls der in Jira hinterlegte Mitarbeiter in der «T\_USER» Tabelle vorhanden ist und alle benötigten Rechte auf die jeweiligen Projekte verfügt. Fehlen Berechtigungen oder ist der Mitarbeiter in der Tabelle nicht vorhanden wird eine Fehlermeldung ausgegeben.

Ergebnis

Alle noch nicht erfassten Leistungen des temporären Benutzers wurden erfasst, alle Leistungen anderer Mitarbeiter führten zu einer Warnung in der Log Datei da die Benutzer nicht in der «T\_USER» Tabelle hinterlegt wurden.

Massnahmen

Keine.

Resultat

Erfolgreich.

* + - 1. Scheduler Test

Testspezifikation

Testende Person

Simon André Lötscher

Beschreibung

Es wurde ein Auftrag im Windows Task Scheduler erstellt mit welchem die Applikation aufgerufen wird. Dieser Auftrag wurde anschliessend ausgeführt.

Erwartetes Ergebnis

Daten sollten, wie bei den manuellen Tests mit Überprüfung erfasst werden.

Ergebnis

Das Programm stürzte mit einer Fehlermeldung ab.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Massnahmen

Nach Überprüfen der Einstellungen im Scheduler wurde der Ausführungsort definiert und in der Dokumentation wurde festgehalten, dass entweder die appsettings.json Datei im Standardausführungsordner des Schedulers vorhanden sein muss, oder der Ausführungspfad des Schedulers so definiert werden muss, dass die Applikation am korrekten Ort ausgeführt wird. Weiterhin wurde innerhalb der Applikation ein Test geschrieben sodass die Applikation falls keine appsettings.json Datei vorhanden ist abgebrochen wird und eine Fehlermeldung in die Log Datei geschrieben wird.

Resultat

Nach Massnahmen erfolgreich.

1. Schlusswort
   1. Fazit

[Kritisch Würdigen, Schlüsse ziehen, Erfolge / Misserfolge, Gelerntes, Persönliche Bilanz]

* + 1. Projektplanung
       1. Zeitplanung
       2. Projektmethode
    2. Implementation
    3. Dokumentation
    4. Testing
  1. Nächste Schritte

1. Verzeichnisse
   1. Akronyme

|  |  |
| --- | --- |
| **API** | Application Programming Interface. |
| **IPA** | Individuelle Praktische Arbeit. |
| **REST** | Representational State Transfer. |
| **SQL** | Structured Query Language. |
| **T-SQL** | Transact-SQL. |

* 1. Glossar

|  |  |
| --- | --- |
| **Agricola** | Agricola bezeichnet das Agrarinformationssystem für Direktzahlungen folgender zwölf Kantone: Aargau, Appenzell Ausserhoden, Appenzell Innerhoden, Glarus, Graubünden, Nidwalden, Obwalden, Schwyz, St. Gallen, Tessin, Uri, Zürich |
| **API** | Eine API ist ein Mittel mit welchem man mit externen Systemen interagieren kann. |
| **API-Token** | Ein API-Token ist eine alphanumerische Zeichenfolge mit welchen der Zugriff auf eine API gesteuert werden kann. |
| **Arrange, Act, Assert** | Arrange, Act, Assert sind die drei Phasen eines Tests. In der ersten Phase (Arrange) werden alle Variablen und Daten auf den Test vorbereitet und initialisiert. In der zweiten Phase (Act) werden Methoden ausgeführt, und in der letzten Phase (Assert) wird bestimmt, ob das erhaltene Ergebnis mit dem erwarteten übereinstimmt. |
| **Branch** | In Git werden sogenannte branches verwendet. Branches erlauben es Entwicklern gleichzeitig an einer Applikation zu arbeiten, ohne sich gegenseitig Daten zu überschreiben. Am Ende der Entwicklungszeit müssen alle Branches vereint werden damit alle Anpassungen aller Entwickler im Endprodukt vorhanden sind. |
| **C#** | C# ist eine weit verbreitete, Objekt orientierte Programmiersprache. |
| **DevExtreme** | DevExtreme ist ein Framework, mit welchem das Programmieren eines Webbasierten Frondendes einer Applikation vereinfacht und um verschiedene Funktionen erweitert wird. |
| **Entität / Entity** | Eine Entität ist ein Objekt mit eindeutig identifizierbaren und eigenständigen Daten. |
| **executable** | Executables sind Dateien mit der Dateiendung .exe. |
| **Flag** | Flags sind zusätzliche Parameter, welche einem Kommandozeilenprogramm beim Start mitgegeben werden können. |
| **Fluent Assertions** | Fluent Assertions ist eine Erweiterung, welche es dem Entwickler erlaubt, beim Assert Schritt in Testfällen natürlichere, lesbarere Sprache zu verwenden. |
| **Foreign Key** | Mit Foreign Key werden Spalten innerhalb von Datenbanktabellen bezeichnet, welche auf andere Tabellen verweisen. |
| **Git** | Git ist ein weit verbreitetes Software Verwaltungstool. |
| **Github / GitHub** | Github ist eine Entwicklerplattform, welche es Entwicklern erlaubt, ihren Code oder ihre anderen Daten zu speichern, managen und teilen. |
| **IPERKA** | IPERKA ist eine lineare Projektmanagement Methode, welche in die sechs Schritte informieren, planen, entscheiden, realisieren, kontrollieren und auswerten unterteilt wird. |
| **Jira** | Jira ist ein von Atlassian entwickeltes Produkt, welches Bug-, Issue-Tracking und agiles Projektmanagement ermöglicht. |
| **Kanban** | Kanban ist eine agile Projektmanagement Methode, welche versucht noch zu erledigende Arbeiten und Fortschritte visuell darzustellen. |
| **Kanbanboard** | Ein Kanbanboard ist ein Hilfstool welches in der Projektmanagement Methode Kanban verwendet wird um Tasks zu tracken und visuell darzustellen. |
| **LAWIS** | LAWIS ist der Name des durch die Softec entwickelte Landwirtschafts-Informationssystems, sowie der Name eines der Entwicklerteams der Softec. |
| **LEIS / Leis / leis** | Das LEIS ist das interne Leistungserfassungssystem der Softec. |
| **Log** | Ein Log ist ein Protokoll eines Computerprogrammes in welchen Informationen, Warnungen und Fehler festgehalten werden. |
| **Mock / Mocking** | In der Softwareentwicklung sind Mock Objekte simulierte Objekte, welche die Funktionsweise produktiver Objekte nachahmen. Mocking ist der Prozess, in welchem diese Objekte erstellt werden. |
| **NSubstitute** | NSubstitute ist eine Erweiterung, mit welcher das Mocking von Daten in Testfällen vereinfacht wird. |
| **NuGet Package** | NuGet Packages sind Erweiterungen, welche in der Form von Paketen einem Projekt hinzugefügt werden können. |
| **Repository** | Ein Repository ist ein Verzeichnis oder Archiv welches zur Verwaltung verschiedenster Daten verwendet wird. |
| **REST** | REST ist ein Software-Architekturstil, der als Leitfaden für das Design und die Entwicklung der Architektur des World Wide Web entwickelt wurde. REST definiert eine Reihe von Einschränkungen, wie sich die Architektur eines Systems im Internet verhalten sollte. |
| **Self-documenting code** | Self-documenting code folgt einer Namenskonvention sowie einer Struktur, welche es Entwicklern einfacher macht, sich in ein System einzuarbeiten und ohne weitere Kommentare auszukommen. |
| **Solution** | Eine Solution ist eine Sammlung verschiedener Projekte, welche zusammen eine Applikation ausmachen. |
| **SQL** | SQL ist eine Sprache, welche verwendet wird, um Daten zu managen. |
| **T-SQL** | T-SQL ist eine Erweiterung des SQL-Standards von Microsoft. |
| **Task** | Ein Task ist eine Aufgabe. |
| **Trigger / Datenbanktrigger** | Trigger sind Funktionen diverser Datenbankmanagement Systeme, welche unter anderem zur Gewährleistung der Datenpersistenz innerhalb einer Datenbank ausgeführt werden können. |
| **Wasserfall** | Im linearen Wasserfallmodell geht eine Projektphase immer in die nächste, nie zurück, ohne dass die Reihenfolge der einzelnen Phasen verändert werden. |
| **Whitelist** | In einer Whitelist werden alle Einträge einer Gruppe (zum Beispiel verschiedene Projekte) festgehalten, welche in einem Programm erlaubt sind. |
| **Worklog** | In Jira wird auf jedem Task ein Worklog geführt, dieses enthält eine Historie aller Zeiterfassungen auf die jeweiligen Tasks. |

* 1. Literatur

1. IPERKA – Detaillierte Dokumentation

* [Microsoft Word - Iperka\_OdA\_200617.docx (ict-berufsbildung-bern.ch)](https://www.ict-berufsbildung-bern.ch/resources/Iperka_OdA_200617.pdf) – Letzter Aufruf: 22.03.2024

1. Wasserfall – Vorteile, Nachteile und Risiken

* [▷ Wasserfallmodell | Brauchen wir agil? | Scheitern mit Ansage (synapsenstau.de)](https://synapsenstau.de/wasserfallmodell/) – Letzter Aufruf: 22.03.2024

1. Kanban – Herkunft, Erklärung, Anwendung und Hintergrundinformationen

* [Kanban (development) - Wikipedia](https://en.wikipedia.org/wiki/Kanban_(development)) – Letzter Aufruf: 22.03.2024
* [Kanban: Eine kurze Einführung | Atlassian](https://www.atlassian.com/de/agile/kanban) – Letzter Aufruf: 22.03.2024

1. Clean Architecture – Prinzipien, Dokumentation (sowie andere Architekturen)

* [Common web application architectures - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures)

1. Jira REST API Dokumentation

* <https://developer.atlassian.com/cloud/jira/platform/rest/v3/api-group-projects/#api-group-projects>

1. Atlassian API Tokens

* <https://support.atlassian.com/atlassian-account/docs/manage-api-tokens-for-your-atlassian-account/>

1. Microsoft Data SqlClient Dokumentation

* [Introduction to Microsoft.Data.SqlClient namespace - ADO.NET Provider for SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/connect/ado-net/introduction-microsoft-data-sqlclient-namespace?view=sql-server-ver16)

1. xUnit Dokumentation

* [Getting started: .NET Core with command line > xUnit.net](https://xunit.net/docs/getting-started/netcore/cmdline)

1. NSubstitute Dokumentation

* [NSubstitute: Getting started](https://nsubstitute.github.io/help/getting-started/)

1. Fluent Assertions Dokumentation

* [Introduction - Fluent Assertions](https://fluentassertions.com/introduction)
  1. Abbildungsverzeichnis

2.2 Projektorganisation 9

2.3.1.1 IPERKA 11

2.3.1.2 Wasserfall 11

2.3.1.3 Kanbanboard 12

3.1 Use Case Diagramm 15

3.4 Risikomatrix 25

3.6 – 2 Zeitplan Legende

4.1 – 1 Kanbanboard Tag 1

4.1 – 2 Kanbanboard Tag 2

4.1 – 3 Kanbanboard Tag 3

4.1 – 4 Kanbanboard Tag 4

4.1 – 5 Kanbanboard Tag 5

4.1 – 6 Kanbanboard Tag 6

4.1 – 7 Kanbanboard Tag 7

4.1 – 8 Kanbanboard Tag 8

4.1 – 9 Kanbanboard Tag 9

4.1 – 10 Kanbanboard Tag 10

6.3 LEIS Datenbankschema

7.2 Clean Architecture Onion View

7.4 Klassendiagramm

1. Anhang
   1. Git-Log