

## Einzelbeispiel (SE&PM/JAVA), Wintersemester 2022

**Bitte lesen Sie dieses Dokument aufmerksam und bis zum Ende durch, bevor Sie mit der Arbeit beginnen. Nur so können Sie sicherstellen, dass Sie die gesamte Angabe verstanden haben!**

Um an der Gruppenphase der Lehrveranstaltung Software Engineering & Projektmanagement teilnehmen zu können, muss das Einzelbeispiel **selbständig** erfolgreich gelöst werden.

### Zu verwendende Technologien

Programmiersprache	Java OpenJDK 17
Java-Framework	Spring Boot 2.7.x
JavaScript Runtime	Node.js 18.x.x
Frontend Framework	Angular 14.x.x
Datenbank	H2 2.1.x
Test-Framework	JUnit 5.x.x
Build & Dependency Management	Maven 3.x.x
	npm 8.x.x
Versionierung	Git 2.x.x

Als Entwicklungsumgebung empfehlen wir IntelliJ IDEA in der aktuellen Version<sup>1</sup>.

### Betreuung durch unsere Tutor/innen während der Eingangsphase

Bei Fragen und Unklarheiten während der Eingangsphase stehen Ihnen unsere Tutor/innen per TUWEL Diskussionsforum zur Verfügung. Wir erwarten, dass Sie die Aussendungen via Tuwel lesen und das Forum durchsuchen, bevor Sie eine Frage doppelt stellen. Zusätzlich gibt Betreuungsstunden, zu denen unsere Tutor/innen online zur Verfügung stehen, um bei der Problemlösung behilflich zu sein.

**Wichtig:** Je genauer Sie Ihre Fragen formulieren, desto besser kann Ihnen geholfen werden.

**Online-Supportstunden:** Termine und Informationen finden Sie in TUWEL.

## 1 Erwartete Vorkenntnisse

Fachliche und methodische Kompetenzen:

- Objektorientierte Analyse, Design und Programmierung
- Grundlagen der Unified Modeling Language (UML)
- Grundkenntnisse aus Algorithmen und Datenstrukturen
- Grundkenntnisse zu Datenbanksystemen

Kognitive und praktische Kompetenzen:

- Eine praxisrelevante Programmiersprache und -werkzeuge (z.B. Java) anwenden
- Eine IDE und Quellcodeverwaltung anwenden

<sup>1</sup><https://www.jetbrains.com/idea/download/>

# Inhaltsverzeichnis

<b>1 Erwartete Vorkenntnisse</b>	<b>1</b>
<b>2 Angabe</b>	<b>3</b>
2.1 Domänenmodell . . . . .	3
2.2 Dokumentation . . . . .	3
2.3 Implementierungsreihenfolge . . . . .	3
2.3.1 Userstories . . . . .	3
2.3.2 Techstories . . . . .	4
2.4 Implementierung . . . . .	4
2.5 Userstories . . . . .	4
2.5.1 Pferdebesitzer/in . . . . .	4
2.5.2 Mockups . . . . .	8
2.6 Techstories . . . . .	11
2.6.1 Qualitätsmanager/in (20 Storypoints) . . . . .	11
2.6.2 Usability Engineer (20 Storypoints) . . . . .	15
2.6.3 Technische/r Architekt/in (20 Storypoints) . . . . .	17
2.6.4 Datenmanager/in (20 Storypoints) . . . . .	22
<b>3 Implementierung</b>	<b>24</b>
3.1 Erste Schritte . . . . .	24
3.1.1 Backend . . . . .	24
3.1.2 Frontend . . . . .	24
3.1.3 Aufbau des Templates . . . . .	25
3.2 Spring . . . . .	25
3.3 Backend Build- und Dependencymanagement . . . . .	25
3.4 Angular CLI . . . . .	26
3.5 Vom Domänenmodell zur Datenbank . . . . .	26
3.6 Reihenfolge der Implementierung . . . . .	27
3.6.1 Persistenz . . . . .	27
3.6.2 Service . . . . .	28
3.6.3 REST . . . . .	28
3.7 Testen . . . . .	28
3.7.1 Normalfall und Fehlerfall . . . . .	29
3.7.2 Manuelle Tests . . . . .	29
3.7.3 Automatisierte Tests . . . . .	29
3.8 Versionskontrolle . . . . .	30
3.8.1 Initialisieren . . . . .	30
3.8.2 Add & Commit . . . . .	30
3.8.3 Remotes, Push & Pull . . . . .	31
3.9 Weiterführende Links & Literatur . . . . .	33
<b>4 Bewertung</b>	<b>34</b>
4.1 Bestehen der Einzelphase . . . . .	34
4.1.1 Einstiegstest (max. 10 Punkte) . . . . .	34
4.1.2 Einzelbeispiel (max. 80 Punkte) . . . . .	34
4.2 Einfluss auf die Endnote . . . . .	34
<b>5 Abgabe</b>	<b>35</b>
5.1 Vorbedingungen . . . . .	35
5.2 Wichtige Hinweise zur Abgabe . . . . .	35
5.3 Nach der Bewertung . . . . .	35

## 2 Angabe

Sie sind als Softwareentwickler/in in einem kleinen österreichischen Unternehmen tätig. Ihr Unternehmen wurde beauftragt eine Plattform für *Wendy's Family Tree* zur Verwaltung von Pferden und deren Vorfahren zu entwickeln. Diese Plattform soll als Web Applikation ausgerollt werden und aus einem Backend und einem Frontend bestehen.

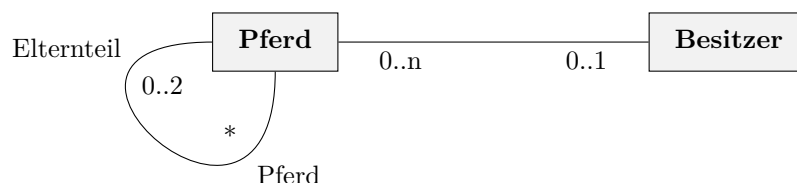
Die Anforderungsanalyse des Gesamtsystems wurde bereits durchgeführt und die Wünsche des Kunden in Stories erfasst. Desweiteren haben einige andere Stakeholder zusätzliche Anforderungen an die Umsetzung formuliert.

In der Aufgabenstellung wird zwischen Userstories und Techstories unterschieden. Userstories beschreiben konkrete Anwendungsfälle der zu erstellenden Software aus Sicht eines bestimmten Stakeholders. Userstories, die von einer fehlerhaften Implementierung betroffen sind, werden nicht abgenommen. Techstories enthalten Anforderungen an die Implementierung sowie weitere Implementierungsdetails. Lesen Sie alle Stories, insbesondere die Techstories aufmerksam durch, bevor Sie mit der Implementierung beginnen. Nicht alle Techstories können nachträglich erfüllt werden. Diese müssen von Anfang an beachtet werden.

Das bereit gestellte Template beinhaltet sowohl einen generellen Durchstich durch alle Schichten, als auch die Implementierung von manchen User Stories entweder im Frontend oder im Backend. Dadurch soll ein besserer Überblick über den gewünschten Stil bereit gestellt werden. Es wird erwartet, dass Sie dennoch für jede Story sicher stellen, dass diese am Ende vollständig implementiert ist. Im Template gibt es keinerlei Garantien auf Vollständigkeit - weder im Hinblick auf User- noch auf Techstories. Weiters wird erwartet, dass Sie bestehenden Code durchlesen, verstehen und weiter verwenden.

### 2.1 Domänenmodell

Um die Domäne etwas zu veranschaulichen, wurde bereits ein Domänenmodell erstellt. Sie müssen dieses Modell vollständig implementieren um positiv bewertet werden zu können.



### 2.2 Dokumentation

Im Rahmen der Stories werden Sie Code-Dokumentation erstellen.

**Wichtig:** Bitte drucken Sie die Code-Dokumentation nicht aus!

Außerdem müssen Sie eine Stundenliste führen, in der Sie Datum und Dauer sowie die Story an der Sie arbeiten festhalten. Diese müssen Sie in einer plaintext Datei im obersten Verzeichnis des Repositories ablegen. Vermerken Sie zudem Ihren Namen und Ihre Matrikelnummer darauf.

### 2.3 Implementierungsreihenfolge

#### 2.3.1 Userstories

Am besten implementieren Sie vertikal nach Userstories, alle Tests und Schichten einer Userstory sollten fertig sein, bevor Sie mit der Nächsten beginnen. Natürlich können Sie später auch bei

bereits implementierten Userstories nacharbeiten. Dieses Vorgehen hilft Ihnen dabei möglichst viele der Userstories abzuschließen.

### 2.3.2 Techstories

Techstories haben Einfluss auf die Implementierung aller Userstories. Sie sind daher laufend während der gesamten Entwicklung zu berücksichtigen.

## 2.4 Implementierung

Ihr/e technische/r Architekt/in hat zusätzlich zu den Anforderungen verschiedene Hilfestellungen im Abschnitt Implementierung für Sie zusammengestellt, um Ihnen den Aufbau des Programms zu erleichtern.

## 2.5 Userstories (80 Storypoints)

Userstories stellen Anforderungen eines Stakeholders an das Gesamtsystem dar. Jede Userstory muss dabei in allen Schichten des Backend (REST, Businesslogik, Persistenz) und im Frontend implementiert werden. Alle in der Userstory angegebenen Daten müssen im Frontend sichtbar sein.

Für die Erfüllung von Userstories erhalten sie Punkte entsprechend den angegebenen Storypoints (oder bei Mängeln entsprechend weniger). Beachten Sie auch den Abschnitt 2.5.2, in welchem Sie beispielhafte Screenshots für den Aufbau und das Aussehen der einzelnen Ansichten finden. Orientieren Sie sich an diesen, wenn Sie ihr Frontend entwickeln!

**Wichtig:** Die Aufschlüsselung der Userstories auf Stakeholder dient ausschließlich dem leichteren Verständnis. Sie sollen kein Mehrbenutzersystem entwickeln. Sie können daher auch die Möglichkeit von Race Conditions durch gleichzeitiges Arbeiten mehrerer User ignorieren. Weiters wird in der Einzelphase auch auf Log-in bzw. Authentifizierung verzichtet.

### 2.5.1 Pferdebesitzer/in

<b>ID:</b>	1	<b>Titel:</b>	Als Pferdebesitzer/in möchte ich neue Pferde im System anlegen können.
<ul style="list-style-type: none"> <li>• Jedes angelegte Pferd muss in einer Datenbank gespeichert werden.</li> <li>• Zu jedem Pferd müssen dabei folgende Werte gespeichert werden:               <ul style="list-style-type: none"> <li>– Name (verpflichtend)</li> <li>– Beschreibung (optional)</li> <li>– Geburtsdatum (verpflichtend)</li> <li>– Geschlecht (biologisches Geschlecht, männlich/weiblich) (verpflichtend)</li> <li>– Besitzer/in (optional, maximal eine/r)</li> </ul> </li> </ul> <p><i>Diese Story ist im Frontend bereits teilweise umgesetzt. Verwenden Sie den bestehenden Code. Beachten Sie, dass sie diesen möglicherweise anpassen beziehungsweise erweitern müssen.</i></p>			
<b>Storypoints:</b>		8	

<b>ID:</b> 2	<b>Titel:</b> Als Pferdebesitzer/in möchte ich bestehende Pferde bearbeiten können.
<ul style="list-style-type: none"><li>• Jedes Pferd, dass bereits im System erfasst ist, soll bearbeitet werden können.</li><li>• Alle Werte, die beim Erstellen erfasst werden, müssen auch verändert werden können.</li><li>• Der Bearbeitungsdialog soll eine separate Ansicht sein.<ul style="list-style-type: none"><li>– Insbesondere soll das Bearbeiten nicht Teil der Suchansicht sein.</li></ul></li></ul> <p><i>Diese Story ist im Backend bereits teilweise umgesetzt. Verwenden Sie den bestehenden Code. Beachten Sie, dass sie diesen möglicherweise anpassen beziehungsweise erweitern müssen.</i></p>	
<b>Storypoints:</b> 5	

<b>ID:</b> 3	<b>Titel:</b> Als Pferdebesitzer/in möchte ich ein Pferd löschen können.
<ul style="list-style-type: none"><li>• Ein gelöscht Pferd muss restlos aus der Datenbank entfernt werden, und darf nicht wiederherstellbar sein.</li><li>• Ab dem Zeitpunkt der Löschung darf das Pferd nicht mehr im System aufscheinen und keine Aktion mehr damit durchgeführt werden können.</li><li>• Die Beziehungen des Pferdes sollen entfernt, die anderen Pferde jedoch nicht weiter beeinflusst werden.</li><li>• Ein Pferd soll in folgenden Ansichten gelöscht werden können:<ul style="list-style-type: none"><li>– der Bearbeitungsansicht (siehe US 2)</li><li>– der Detailansicht (siehe US 5)</li><li>– den Suchergebnissen (siehe US 6)</li><li>– dem Stammbaum (siehe US 9)</li></ul></li></ul>	
<b>Storypoints:</b> 5	

<b>ID:</b> 4	<b>Titel:</b> Als Pferdebesitzer/in möchte ich für ein Pferd Eltern zuweisen.
<ul style="list-style-type: none"> <li>• Die Eltern sollen bereits beim Erstellen des Pferdes eingetragen werden können. (siehe US 1)</li> <li>• Die Eltern eines existierenden Pferdes sollen verändert werden können. (siehe US 2)</li> <li>• Die Elternteile eines existierenden Pferdes sollen wieder entfernt werden können. (siehe US 2)</li> <li>• Jedes Pferd kann bis zu zwei Elternteile haben.</li> <li>• Sind zwei Elternteile eingetragen, dürfen sie nicht das gleiche Geschlecht haben.</li> <li>• Die Auswahl der Elternteile soll über ein Suchfeld geschehen, bei dem durch Eingabe eines Teils des Namens gefiltert wird und dann aus einer Liste der verbleibenden Kandidaten ausgewählt werden kann.             <ul style="list-style-type: none"> <li>– Es sollen maximal 5 Kandidaten angezeigt werden, auch wenn es noch mehr passende Kandidaten gibt.</li> </ul> </li> </ul>	
<b>Storypoints:</b> 11	

<b>ID:</b> 5	<b>Titel:</b> Als Pferdebesitzer/in möchte ich die Daten eines Pferdes in einer Detailansicht sehen können.
<ul style="list-style-type: none"> <li>• In der Detailansicht sollen alle erfassten Informationen zum Pferd zu sehen sein.</li> <li>• Von dieser Ansicht soll man direkt zum Bearbeitungsdialog (siehe US 2) kommen können.</li> <li>• Von dieser Ansicht soll das Pferd direkt gelöscht werden können (siehe US 3).</li> <li>• Von dieser Ansicht aus soll man direkt zur Detailansicht der Elternpferde kommen können. (siehe US 5)</li> <li>• Die Detailansicht soll eine separate Ansicht sein.             <ul style="list-style-type: none"> <li>– Die Detailansicht unterscheidet sich von der Bearbeitungsansicht darin, dass keine Änderungen vorgenommen werden können. Das soll auch für den/die Benutzer/in als solches erkennbar sein.</li> <li>– Sie soll auch nicht Teil der Suchansicht sein.</li> </ul> </li> </ul> <p><i>Diese Story ist im Backend bereits teilweise umgesetzt. Verwenden Sie den bestehenden Code. Beachten Sie, dass sie diesen möglicherweise anpassen beziehungsweise erweitern müssen.</i></p>	
<b>Storypoints:</b> 7	

<b>ID:</b> 6	<b>Titel:</b> Als Pferdebesitzer/in möchte ich nach Pferden suchen können.
<ul style="list-style-type: none"> <li>Die Pferde sollen nach folgenden Kriterien suchbar sein:               <ul style="list-style-type: none"> <li>Name</li> <li>Beschreibung</li> <li>Geburtsdatum (älter als das eingegebene Datum)</li> <li>Geschlecht</li> <li>Besitzer/in</li> </ul> </li> <li>Die Suchkriterien sollen beliebig kombinierbar sein (logisch UND verknüpft).</li> <li>Textfelder sollen case insensitive sein und bereits bei der Eingabe von einem beliebigen Teilstring matchen.</li> <li>Wenn die Suchkriterien geändert werden, sollen die Suchergebnisse automatisch aktualisiert werden, ohne dass ein weiterer Button gedrückt werden muss. Die Aktualisierung der Suche soll dabei debounced werden.<sup>2</sup></li> <li>Zumindest die Attribute, nach denen man suchen kann, sollen auch bei den Suchergebnissen direkt angezeigt werden.               <ul style="list-style-type: none"> <li>Wird mehr als ein Kriterium eingegeben sollen die Pferde aufgelistet werden, die alle erfüllen.</li> <li>Eine Suche ohne eingegebene Kriterien listet alle Pferde auf.</li> </ul> </li> <li>Mit den Suchergebnissen möchte ich folgende Aktionen durchführen können:               <ul style="list-style-type: none"> <li>Zur Detailansicht gehen (siehe US 5)</li> <li>Zum Bearbeitungsdialog gehen (siehe US 2)</li> <li>Das Pferd löschen (siehe US 3)</li> </ul> </li> </ul>	
<b>Storypoints:</b> 12	

<b>ID:</b> 7	<b>Titel:</b> Als Pferdebesitzer/in möchte ich neue Besitzer/innen eintragen können.
<ul style="list-style-type: none"> <li>Jede/r angelegte Besitzer/in muss in der Datenbank gespeichert werden.</li> <li>Zu jedem/jeder Besitzer/in müssen dabei folgende Werte gespeichert werden:               <ul style="list-style-type: none"> <li>Vorname</li> <li>Nachname</li> <li>E-Mail-Adresse (optional)</li> </ul> </li> </ul> <p><i>Diese Story ist im Backend bereits teilweise umgesetzt. Verwenden Sie den bestehenden Code. Beachten Sie, dass sie diesen möglicherweise anpassen beziehungsweise erweitern müssen.</i></p>	
<b>Storypoints:</b> 6	

<sup>2</sup><https://rxjs.dev/api/operators/debounceTime>

<b>ID:</b> 8	<b>Titel:</b> Als Pferdebesitzer/in möchte ich alle Besitzer/innen sehen können.
<ul style="list-style-type: none"> <li>• Alle Besitzer/innen sollen angezeigt werden.</li> <li>• Zu jedem/jeder Besitzer/in müssen dabei folgende Werte angezeigt werden: <ul style="list-style-type: none"> <li>– Vorname</li> <li>– Nachname</li> <li>– E-Mail-Adresse (optional)</li> </ul> </li> </ul>	
<b>Storypoints:</b> 8	

<b>ID:</b> 9	<b>Titel:</b> Als Pferdebesitzer/in möchte ich den Stammbaum eines Pferdes ansehen können.
<ul style="list-style-type: none"> <li>• Es sollen die Vorfahren des gewählten Pferdes angezeigt werden, nicht die Nachfahren.</li> <li>• Der Stammbaum soll hierarchisch in Form eines Baums dargestellt werden. <ul style="list-style-type: none"> <li>– Zu jedem Pferd soll im Baum Name und Geburtsdatum sichtbar sein.</li> <li>– Der Unterbaum, an dessen Wurzel ein Pferd steht, beinhaltet dessen Vorfahren. Insbesondere sind die direkten Kindknoten eines Pferdes dessen Eltern.</li> <li>– Die einzelnen Knoten sollen auf- und einklappbar sein, das heißt, der Unterbaum der Vorfahren eines bestimmten Pferdes im Baum soll angezeigt beziehungsweise ausgeblendet werden können.</li> <li>– Man soll von dieser Ansicht aus direkt zur Detailansicht jedes aufgelisteten Pferdes kommen können. (siehe US 5)</li> <li>– Man soll von dieser Ansicht aus direkt zur Bearbeitungsansicht jedes aufgelisteten Pferdes kommen können. (siehe US 2)</li> <li>– Es soll möglich sein ein Pferd direkt aus dieser Ansicht zu löschen.</li> </ul> </li> <li>• Es soll eine maximale Anzahl von Generationen, die angezeigt werden sollen, angegeben werden können.</li> <li>• Der so erzeugte Stammbaum soll initial vollständig aufgeklappt sein.</li> </ul>	
<b>Storypoints:</b> 18	

## 2.5.2 Mockups

In diesem Abschnitt werden einige Screenshots für die verschiedenen Ansichten präsentiert. Ihre Implementierung muss nicht hundertprozentig identisch aussehen, halten sie sich aber an den Aufbau und das Layout in diesen Beispielen und stylen Sie ihre Komponenten ähnlich, damit leicht ersichtlich ist, welche Oberflächenelemente welchen Zweck erfüllen.

Für Teile des Frontends, von denen hier keine Screenshots gezeigt sind, orientieren Sie sich bitte an den hier gezeigten beziehungsweise den im Template bereits vorhandenen Teilen.

**US 2 Pferd bearbeiten** Abbildung 1 zeigt die Bearbeitungsansicht mitsamt den Feldern für die Eltern des Pferdes (siehe US 2, US 4). Der Aufbau entspricht dabei der Erstellungsansicht, enthält aber die zusätzlich Buttons zum Löschen und der Navigation zur Detailansicht.



# Edit Horse

[i Details](#)

Name	<input type="text" value="Binky"/>	Date of Birth	<input type="text" value="13.05.1984"/>
Sex	<input style="border: 1px solid #ccc; background-color: #f9f9f9;" type="text" value="Male"/>	Owner	<input type="text"/>
Mother	<input type="text" value="Iltschi"/>	Father	<input type="text" value="Hatatitla"/>
Description	<input type="text" value="Does he hover?"/>		


 Delete
Save

Abbildung 1: Stammbaum

**US 5 Detailansicht** In Abbildung 2 ist die Detailansicht dargestellt. Beachten Sie, dass das Layout dem Layout der Bearbeitungs- und der Erstellungsansicht entspricht, aber keine aktiven Eingabeelemente vorhanden sind.

# Details of Horse

[Edit](#)

Name	Binky	Date of Birth	13.5.1984
Sex	Male	Owner	
Mother	Iltschi	Father	Hatatitla
Description	Does he hover?		


 Delete

Abbildung 2: Detailansicht

**US 6 Suche** In Abbildung 3 ist die Suchansicht dargestellt. Diese ist zum größten Teil im Template bereits vorgegeben. Beachten Sie die Buttons auf der rechten Seite, von welche die ersten drei zu den Details (siehe US 5), dem Stammbaum (siehe US 9) und der Bearbeitung (siehe US 2) verlinken und der letzte das Pferd löscht (siehe US 3).

**US 8 Besitzerliste** Abbildung 4 zeigt die Liste der Besitzer. Beachten Sie den Button rechts oben, der zur Erstellung von Besitzern verlinkt.

**US 9 Stammbaum** In Abbildung 5 ist der Stammbaum dargestellt. In diesem Beispiel wurde der Stammbaum auf 5 Generationen begrenzt, wobei der Knoten „Iltschi“ eingeklappt ist, also dessen Eltern nicht sichtbar sind. Auf der linken Seite eines jeden Knotens, befindet sich ein Button zum auf- und zuklappen des Unterbaumes (also der Vorfahren des Pferdes). Auf der

### Abbildung 3: Suchansicht

Abbildung 4: Suchansicht

# Family Tree

Abbildung 5: Stammbaum

## 2.6 Techstories (80 Storypoints)

Techstories stellen Anforderungen eines Stakeholders an die Qualität der Umsetzung des Produkts dar. Jede Techstory muss dabei in allen Schichten und allen Userstories beachtet werden. Im Gegensatz zu den Userstories erfolgt die Aufteilung der Storypoints bei den Techstories nach Stakeholder. Verstöße gegen Anforderungen in Techstories führen zu Punkteabzügen aus dem Kontingent des jeweiligen Stakeholders.

Sie müssen zeigen, dass Sie technische Herausforderungen lösen können. Daher müssen Sie, wenn sie einen Aspekt gar nicht implementieren, mit Abzügen bei Techstories rechnen.

### 2.6.1 Qualitätsmanager/in (20 Storypoints)

<b>ID:</b> 10	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass das Backend Logfiles schreibt.
<ul style="list-style-type: none"> <li>Log-Einträgen müssen mit sinnvollen Loglevels versehen werden. Einige Beispiele für Dinge, die sie in dieser Aufgabe loggen <i>müssen</i>:               <ul style="list-style-type: none"> <li><code>error</code> für alle Arten von Fehlern, insbesondere Exceptions</li> <li><code>info</code> für vom User durchgeführte Aktionen, zB. am Anfang einer REST-Endpointmethode</li> <li><code>debug</code> für Nachrichten in komplexeren Codestücken die beim Suchen von Problemen wertvoll sein können</li> <li><code>trace</code> am Anfang aller Methoden die nicht schon andertwertig geloggt wurden</li> </ul> </li> <li>Grundsätzlich sollen Meldungen ab Level <code>info</code> im Log aufscheinen. Für das Paket in dem das Programm selbst liegt (im Gegensatz zu den Bibliotheken) soll zumindest <code>debug</code> ebenfalls ausgegeben werden.</li> <li>Exceptions müssen exakt einmal geloggt werden. Dabei muss der volle Stacktrace in das Log geschrieben werden. Wird eine Exception gewrappt, so muss die wrappende und nicht die ursprüngliche Exception geloggt werden, da die wrappende (=äußere) einen längeren Stacktrace besitzt und damit mehr Informationen bereit stellt.</li> <li>Das Logfile muss mindestens täglich rotiert werden. Der Dateiname von älteren Logfiles muss das Datum enthalten. Für das aktuelle Logfile ist dies nicht notwendig.</li> <li>Das Programm darf ausschließlich über das Logging Framework Ausgaben auf die Standardausgabe (stdout) sowie die Standardfehlerausgabe (stderr) schreiben.</li> <li>Wenn Exceptions gewrappt / weitergereicht werden, so darf die ursprüngliche Exception inklusive Stacktrace nicht verloren gehen.</li> <li>Die Konfiguration des Loggers darf ausschließlich in der application.yml erfolgen. Verwenden Sie keine XML Dateien dafür.</li> <li>Wenn Sie den Aufruf von Methoden loggen, so stellen Sie sicher, dass auch die Parameter sinnvoll im Log aufscheinen.</li> <li>Es soll nachvollziehbar sein, zu welchem HTTP-Request ein Log-Eintrag gehört.</li> </ul> <p><i>Einige geforderte Dinge sind so bereits im Template umgesetzt. Es ist Teil der Aufgabe, die Dokumentation der beteiligten Technologien zu lesen und daraus abzuleiten, was Sie noch anpassen müssen.</i></p>	

<b>ID:</b> 11	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass das Programm in Englisch geschrieben und gut dokumentiert ist.
<ul style="list-style-type: none"><li>• Der Quellcode (Klassennamen, Variablenamen sowie Methodennamen) muss auf Englisch verfasst sein.</li><li>• Für alle Interfaces und Deklarationen in Interfaces muss JavaDoc existieren.</li><li>• Zweck und Bedeutung von Exceptionklassen müssen beschrieben sein.</li><li>• Die JavaDoc enthält Informationen zu:<ul style="list-style-type: none"><li>– Übergabeparametern.</li><li>– Rückgabewerten.</li><li>– möglichen Fehlerfällen und Exceptions.</li></ul></li><li>• Die JavaDoc ist auf Englisch geschrieben.</li></ul>	

<b>ID:</b> 12	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass eine Stundenliste bei der Entwicklung geführt wird.
<ul style="list-style-type: none"><li>• Die Stundenliste muss in Form einer Plaintextdatei im obersten Verzeichnis des Projekts zu finden sein.</li><li>• Folgende Daten müssen auf der Stundenliste zu finden sein:<ul style="list-style-type: none"><li>– Name</li><li>– Matrikelnummer</li><li>– Gesamtsumme der Arbeitszeit am Projekt</li></ul></li><li>• Jeder Eintrag muss folgende Daten enthalten:<ul style="list-style-type: none"><li>– Datum</li><li>– Dauer</li><li>– bearbeitete Story</li></ul></li></ul>	

<b>ID:</b> 13	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass Kernfunktionalitäten des Programms getestet sind.
<ul style="list-style-type: none"><li>• Es müssen mindestens vier Tests für jede Backendschicht (REST, Service, Persistence) erstellt werden.</li><li>• Es müssen sowohl Positivtests (Normalfall) als auch Negativtests (Fehlerfall) in jeder Schicht erstellt werden.</li><li>• Mindestens ein Positivtest pro Schicht muss eine schreibende Operation betreffen.</li><li>• In Summe sollen mindestens 12 neue Tests geschrieben werden.</li><li>• Sie können sowohl Unit als auch Integration Tests erstellen.</li><li>• Tests verwenden umfangreiche Assertions um sicher zu stellen, dass sich das Programm auch richtig verhält. Es reicht zum Beispiel bei einem Integrationstest, welcher ein Pferd nach seiner ID lädt, nicht, wenn Sie nur den HTTP Status überprüfen. Sie sollen hier zum Beispiel auch sicher stellen, dass sich in allen Feldern auch wirklich die erwarteten Daten befinden.</li></ul>	
<b>ID:</b> 14	<b>Titel:</b> Als Qualitätsmanager/in erwarte ich einen sauberen Git Workflow.
<ul style="list-style-type: none"><li>• Es müssen regelmäßig Commits erstellt werden.</li><li>• Ein Commit gehört meistens zu genau einer Userstory und/oder Techstory. Es kann Überschneidungen von Stories geben, jedoch sollte dies nicht die Regel sein. Die ID der Story soll am Anfang der Commitmessage angegeben werden.</li><li>• Jeder Commit enthält eine aussagekräftige Commitmessage, die genau beschreibt, was Sie implementiert haben.</li><li>• Vom Build und Dependency Management erstellte Dateien dürfen nicht committed werden. Dies betrifft unter anderem den <code>target</code> Ordner im Backend und die <code>dist</code> und <code>node_modules</code> Ordner im Frontend.</li><li>• Daten des laufenden Betriebs, die nicht zum Programm gehören, wie die Datenbank-Datei, dürfen nicht committed werden.</li><li>• Force-push ist auf unserem Server deaktiviert. Sie dürfen die History von gepushten Commits nicht bearbeiten.</li><li>• Alle bewertungsrelevanten Commits müssen vor Ende der Deadline auf den master Branch des bereitgestellten GitLab Repository gepusht werden.</li></ul>	

<b>ID:</b> 15	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass alle Eingaben validiert werden.
<ul style="list-style-type: none"> <li>• Eingabeparameter (Zahlenwerte oder Textwerte) müssen den Userstories entsprechend validiert werden.</li> <li>• Bei der Validierung muss darauf geachtet werden, ob Parameter verpflichtend oder optional sind.</li> <li>• Es muss zumindest in der Serviceschicht des Backends validiert werden.</li> <li>• Validierung soll in einem (oder mehreren) Validatoren implementiert sein.</li> </ul>	

<b>ID:</b> 16	<b>Titel:</b> Als Qualitätsmanager/in erwarte ich sinnvolles Code-Design mit Einhaltung üblicher Coding Conventions und Best Practices.
<ul style="list-style-type: none"> <li>• Einhaltung von Namenskonventionen der jeweiligen Sprache/Technologie. <sup>34</sup> <ul style="list-style-type: none"> <li>– Dies betrifft zum Beispiel die Namen (und deren Schreibung) von Variablen, Klassen und Packages.</li> <li>– Klassen, die eine bestimmte Rolle einnehmen, sollen das in ihrem Namen reflektieren (beispielsweise Services, DAOs, etc.)</li> </ul> </li> <li>• Einheitliche Code-Formatierung (Einrückung, Spacing bei Klammern, ...).</li> <li>• Vermeidung von aus früheren Versionen übrig gebliebenem Code.           <ul style="list-style-type: none"> <li>– unbenutzte Imports</li> <li>– auskommentierte Code-Blöcke</li> </ul> </li> <li>• Sinnvolle Wahl von Datentypen (zum Beispiel ein Enum für fixe Aufzählungen).</li> </ul>	

<b>ID:</b> 17	<b>Titel:</b> Als Qualitätsmanager/in möchte ich, dass CI sinnvoll eingesetzt wird.
<ul style="list-style-type: none"> <li>• Das bereitgestellte .gitlab-ci.yml muss sich im Wurzelverzeichnis befinden.</li> <li>• Bei der Abgabe soll die Pipeline grün sein.</li> <li>• Die Pipeline darf um weitere Checks erweitert werden und somit strenger werden, aber keinesfalls entschärft (wie z.B. durch das ausschalten von Tests).</li> </ul>	

<sup>4</sup><https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>

<sup>4</sup><https://v2.angular.io/docs/ts/latest/guide/style-guide.html#!#naming>

## 2.6.2 Usability Engineer (20 Storypoints)

<b>ID:</b> 18	<b>Titel:</b> Als Usability Engineer möchte ich ein übersichtliches Frontend.
<ul style="list-style-type: none"> <li>• Auf einer Seite soll nicht zu viel Funktionalität zusammengefasst werden, um es für den/die Benutzer/in übersichtlich zu halten (zum Beispiel Bearbeitungsformular separat von Suchmaske).</li> <li>• Funktionen sollen intuitiv auffindbar sein (zum Beispiel nicht in den Bearbeitungsdialog eines Elementes gehen müssen um es löschen zu können).</li> <li>• Es sollen sinnvolle UI-Elemente und Styles verwendet werden (zum Beispiel Tabellen für tabellarische Daten, Elemente die auf Klicks reagieren als Button, Link oder ähnliches erkennbar sein).</li> </ul>	
<b>ID:</b> 19	<b>Titel:</b> Als Usability Engineer möchte ich, dass das Frontend für einen Benutzer sinnvoll zu bedienen ist.
<ul style="list-style-type: none"> <li>• Wählen Sie für Eingaben die jeweils passendste Eingabemodalität, zum Beispiel:             <ul style="list-style-type: none"> <li>– Ein numerisches Eingabefeld für Zahlen</li> <li>– Eine Textbox für Freitext</li> <li>– Filterbarer Drop-Down zur Auswahl aus einer großen Menge von Optionen</li> </ul> </li> <li>• Es dürfen keine Datenbank-IDs im UI angezeigt oder eingegeben werden müssen.             <ul style="list-style-type: none"> <li>– Eine Datenbank-ID ist ein technisches Detail, das für den/die Benutzer/in keine Bedeutung hat.</li> <li>– Bei der Kommunikation mit dem Backend über REST dagegen dürfen IDs sowohl in den DTOs als auch den URLs verwendet werden.</li> <li>– In den Routen im Frontend dürfen IDs verwendet werden um zum Beispiel die zu bearbeitende Entität zu kommunizieren, obwohl diese genau genommen für den Benutzer sichtbar sind.</li> </ul> </li> <li>• Ein vollständiges Neuladen der Seite im Browser durch den/die Benutzer/in darf nicht notwendig sein. Auch die Seite zu wechseln und wieder zur Ausgangsseite zu wechseln gilt in diesem Sinne als <i>Neuladen</i>.</li> <li>• Die Navigation muss vollständig aus der Funktionalität der Seite selbst möglich sein. Es darf nicht notwendig sein, die Adresse manuell zu bearbeiten.</li> <li>• <code>window.alert</code> sowie <code>window.confirm</code> dürfen nicht verwendet werden.</li> </ul>	

<b>ID:</b> 20	<b>Titel:</b> Als Usability Engineer möchte ich, dass das Frontend den/die Benutzer/in über Ereignisse aussagekräftig informiert.
<ul style="list-style-type: none"><li>• Der/Die Benutzer/in soll über Ereignisse aller Art immer erkennen können, was gerade passiert ist.</li><li>• Bei Aktionen, bei welchen möglicherweise nicht sofort ein Effekt sichtbar ist, soll eine Erfolgsmeldung angezeigt werden.</li><li>• In Fehlerfällen muss dem Benutzer eine angemessene Fehlermeldung angezeigt werden.<ul style="list-style-type: none"><li>– Insbesondere soll bei Validierungsfehlern darauf hingewiesen werden, welche Eingabe den Fehler verursacht hat.</li></ul></li></ul>	

<b>ID:</b> 21	<b>Titel:</b> Als Usability Engineer möchte ich, dass die Anwendung von möglichst vielen Benutzer/innen mit ihrer gewohnten Umgebung genutzt werden kann.
<ul style="list-style-type: none"><li>• Das Frontend soll zumindest lauffähig sein in:<ul style="list-style-type: none"><li>– Firefox (Version 105)</li><li>– Chrome/Chromium (Version 106)</li></ul></li></ul>	



### 2.6.3 Technische/r Architekt/in (20 Storypoints)

<b>ID:</b> 22	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich die Verwendung eines Build und Dependency Managements.
<ul style="list-style-type: none"><li>• Abhängigkeiten des Backends müssen mittels Maven verwaltet werden.</li><li>• Das Backend muss über Maven gebaut, gestartet und getestet werden können.</li><li>• Das Backend muss mit dem Befehl: <code>mvn clean compile</code> erfolgreich gebaut werden können.</li><li>• Das Backend muss mit dem Befehl: <code>mvn clean test</code> erfolgreich getestet werden können.</li><li>• Das Backend muss mit dem Befehl: <code>mvn clean compile spring-boot:run</code> gestartet werden können.</li><li>• Für das Backend muss mit dem Befehl: <code>mvn clean package</code> ein ausführbares jar File erstellt werden können.</li><li>• Abhängigkeiten des Frontends müssen, mittels npm verwaltet werden. Es dürfen keine externen URLs (z.B. zu einem CDN) verwendet werden um weitere Funktionalitäten hinzuzufügen.</li><li>• Das Frontend muss mit dem Befehl: <code>ng serve</code> gestartet werden können.</li><li>• Die Abhängigkeiten des Frontends müssen mit dem Befehl: <code>npm install</code> heruntergeladen werden können.</li><li>• Das Frontend muss mit dem Befehl: <code>npm run build</code> fehlerfrei gebaut werden können.</li></ul>	

<b>ID:</b> 23	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich eine gute Umsetzung der Programmarchitektur.
	<ul style="list-style-type: none"><li>• Das Programm muss eine saubere Schichtentrennung aufweisen.</li><li>• Die Austauschbarkeit der Schichten muss gegeben sein.</li><li>• Zur Trennung der Schichten muss das Interface Pattern<sup>5</sup> verwendet werden.</li><li>• Außerdem dürfen andere Komponenten nie direkt instanziiert werden, sondern müssen mittels Dependency Injection<sup>6</sup> injiziert werden.</li><li>• Es muss eine saubere Trennung zwischen DTO- und Entity-Klassen vorhanden sein.<ul style="list-style-type: none"><li>– Entity-Objekte stellen bereits in der Datenbank gespeicherte Entitäten dar und werden nicht selbst über die REST-Schnittstelle übertragen. Zum Beispiel:<ul style="list-style-type: none"><li>* Pferd</li><li>* Besitzer</li></ul></li><li>– DTOs werden zur Kapselung von nicht-Entity-Daten innerhalb des Backends und zur Übertragung über REST verwendet. Hier bieten sich Java Records an, sofern Sie unveränderliche Datenstrukturen verwenden wollen. Zum Beispiel:<ul style="list-style-type: none"><li>* Suchkriterien</li><li>* Daten zum Erstellen eines Pferdes</li><li>* Stammbaum</li></ul></li></ul></li><li>• Suchkriterien sollen auch im REST-Endpointcode nicht als einzelne Parameter sondern in einem DTO gekapselt aufgenommen werden.</li><li>• Verwenden von Exceptions zum Transportieren von Fehlern.</li><li>• Ein vollständiges Neuladen des Frontend (z.B. mit <code>window.location.reload()</code>) ist nicht erlaubt.</li><li>• Verwenden Sie für Formulare Angulars's <i>Template Driven Forms</i> oder <i>Reactive Forms</i>. <i>Template Variablen</i> zu verwenden, um die Eingabewerte einzelnen Elemente abfragen zu können, sind keine geeignete Lösung.</li><li>• Es dürfen im Frontend keine externen Ressourcen eingebunden werden.</li></ul>

<b>ID:</b> 24	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich sinnvolle Verwendung des Routers <sup>7</sup>
<ul style="list-style-type: none"> <li>• Jede der Ansichten der Anwendung soll eine eigene Route haben.</li> <li>• Navigation zwischen den Ansichten soll ausschließlich durch den Router passieren.             <ul style="list-style-type: none"> <li>– In Programmcode soll dafür die Angular-Klasse <code>Router</code> verwendet werden.</li> <li>– Links in den HTML-Templates sollen <code>routerLink</code><sup>8</sup> verwenden, anstatt von <code>href</code>.</li> </ul> </li> <li>• Die Routen sollen sinnvoll aufgebaut sein.             <ul style="list-style-type: none"> <li>– Es soll eine sinnvolle Hierarchie erkennbar sein.</li> <li>– Eventuelle Parameter, die die Ansicht benötigt, sollen über die Route übertragen werden.</li> <li>– Routen dürfen – im Gegensatz zu REST-URIs – Verben enthalten.</li> <li>– Routen und zugehörige Links dürfen – im Gegensatz zum restlichen UI – für den/die Benutzer/in sichtbare IDs enthalten.</li> <li>– Beispiele für konkrete Links:                 <ul style="list-style-type: none"> <li>* Erstellen eines Pferdes: <code>/horses/create</code></li> <li>* Bearbeiten des Pferdes mit ID 42: <code>/horses/42/edit</code></li> <li>* Stammbaum des Pferdes mit ID 42 mit 3 Generationen: <code>/horses/42/familytree?generations=3</code></li> </ul> </li> </ul> </li> <li>• Sie sollen auf Änderungen der Router Parameter sinnvoll reagieren. Ändert sich z.B. in der Detailansicht der Parameter für die ID des Pferdes, so sollen Sie die Daten des entsprechenden Pferdes laden und anzeigen.</li> </ul>	

<b>ID:</b> 25	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich vorausschauenden Umgang mit Ressourcen.
<ul style="list-style-type: none"> <li>• Schließen von externen Resources (z.B. Datenbank-Verbindung), sofern vorhanden.</li> <li>• Suchen müssen aus Performancegründen in der Datenbank vorgenommen werden.</li> <li>• Es dürfen nicht unnötig viele HTTP-Requests ausgeführt werden.             <ul style="list-style-type: none"> <li>– Insbesondere in Fällen, wo eine Liste oder Tabelle von Elementen oder im Fall der Baumstruktur aus (siehe US 9) sollen alle für die Anzeige notwendigen Daten in einem Request übertragen werden.</li> </ul> </li> <li>• Es dürfen nicht mehr Datensätze in einem Request übertragen werden, als tatsächlich benötigt.             <ul style="list-style-type: none"> <li>– Übertragen Sie zum Beispiel nicht alle Pferde, wenn Sie nur zehn anzeigen.</li> </ul> </li> <li>• Sie müssen bei diesem Projekt aus Gründen der Einfachheit noch nicht mit Pagination oder ähnlichem auseinandersetzen.</li> </ul>	

<sup>4</sup><https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

<sup>6</sup><https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

<sup>8</sup><https://angular.io/guide/router>

<sup>8</sup><https://angular.io/api/router/RouterLink>

<b>ID:</b> 26	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich eine durchdachte Fehlerbehandlung.
<ul style="list-style-type: none"><li>• Je nach Art des Fehlers wird eine passende Exception gewählt.</li><li>• Exceptions werden ausschließlich zum Transportieren von Fehlern verwendet.</li><li>• Exceptionbehandlung folgt dem im Architekturtutorial beschriebenen Schema:<ul style="list-style-type: none"><li>– Für Fälle, die im Zuge des normalen Betriebs des Systems erwartet werden, werden Checked Exceptions verwendet. Beispiele dafür sind:<ul style="list-style-type: none"><li>* Vom Frontend kommen ungültige Daten</li></ul></li><li>– Für Fälle, wo unerwartete Fehler auftreten, werden Unchecked Exceptions verwendet.<ul style="list-style-type: none"><li>* Programmierfehler</li><li>* Problem mit der Laufzeitumgebung</li><li>* Zugriff auf andere benötigte Services (z.B. Datenbank)</li></ul></li></ul></li><li>• Es darf keine catch-all Klauseln geben.</li><li>• Fehler müssen im Log aufscheinen.</li><li>• Fehler müssen <u>entweder</u> behandelt <u>oder</u> weitergereicht werden.</li><li>• Das Programm darf zu keinem Zeitpunkt einfach abstürzen.</li></ul>	

ID: 27	<b>Titel:</b> Als Technische/r Architekt/in erwarte ich eine REST-konforme <sup>9</sup> Umsetzung des Backend.
	<ul style="list-style-type: none"><li>• Für jede Response muss der korrekte HTTP-Statuscode zurückgegeben werden. Dabei soll vor allem zwischen 2xx (OK), 4xx (Client Error) und 5xx (Server Error) unterschieden werden.</li><li>• Es soll immer der HTTP Status Code gewählt werden, dessen Beschreibung in der HTTP-Spezifikation und übliche Verwendung am besten zur Situation passt. Insbesondere folgende Status Codes werden dabei erwartet.<ul style="list-style-type: none"><li>– 200 <i>OK</i></li><li>– 201 <i>Created</i></li><li>– 204 <i>No Content</i></li><li>– 400 <i>Bad Request</i> im Fall eines syntaktisch falschen Requests</li><li>– 404 <i>Not Found</i> Im Fall, dass eine bestimmte angefragte Resource nicht existiert, wie zum Beispiel, wenn ein Pferd mit bestimmter ID angefragt wird. Nicht aber, bei einer leeren Menge.</li><li>– 422 <i>Unprocessable Entity</i> im Fall eines Validierungsfehlers</li><li>– 409 <i>Conflict</i> im Fall, dass die Entity im Request selbst valide ist, aber aufgrund des Gesamtzustands trotzdem nicht akzeptiert werden kann</li><li>– 500 <i>Internal Server Error</i> Programmierfehler, nicht behandelbarer Fehler</li></ul></li><li>• Bei jedem Request und Response sollen nur Informationen übertragen werden, die auch wirklich benötigt werden.</li><li>• Für jeden HTTP-Request wird immer die richtige HTTP-Anfragemethode (Verb) verwendet.</li><li>• Das Backend an sich speichert niemals den Status eines Requests ab. Der Zustand der Applikation wird ausschließlich in der Datenbank gespeichert.</li><li>• Uniform Resource Identifiers (URI) bestehen hauptsächlich aus Nomen im Plural und enthalten keine Verben.</li><li>• Kommunikation mit dem Backend erfolgt über JSON (Content-type: application/json). Ausgenommen davon sind Bilder. Diese dürfen alleine stehend mit einem zum Bildtyp passenden Content-type übertragen werden.</li></ul>

## 2.6.4 Datenmanager/in (20 Storypoints)

<b>ID:</b> 28	<b>Titel:</b> Als Datenmanager/in erwarte ich, dass die Anwendung Ihre Daten in einem relationalen Datenbanksystem (RDBMS) persistiert.
<ul style="list-style-type: none"> <li>• Es dürfen neben dem RDBMS keine Daten anderwertig persistiert werden.               <ul style="list-style-type: none"> <li>– Ausgenommen hiervon sind Logs (siehe 10), welche keine eigentlichen Anwendungsdaten darstellen, sondern dem Debugging und der Nachvollziehbarkeit der Systemausführung dienen.</li> </ul> </li> <li>• Der Zustand der Daten muss nach einem Neustart der Anwendung erhalten bleiben.</li> <li>• Daten der Anwendung werden ausschließlich vom Backend persistiert, das Frontend speichert keine Daten.</li> <li>• Die Anwendung darf beim Start bestehende Daten nicht löschen.</li> <li>• Die Anwendung muss in der Lage sein, beim Start die benötigten Strukturen in der Datenbank selbst zu erstellen.</li> <li>• Die Anwendung muss in der Lage sein, ein angemessenes Set von Testdaten (mindestens 10 pro Entität) in die Datenbank einzufügen, wenn sie mit dem Profil <b>datagen</b> gestartet wird. Daten, welche nicht durch diesen Mechanismus eingefügt wurden, dürfen dabei nicht verändert werden. Ihr Programm muss mit diesen Testdaten vollständig testbar sein. Dies impliziert einen Bedarf nach einer ausreichenden Anzahl von Beziehungen und auch Pferde mit einem großen Stammbaum.</li> </ul>	

<b>ID:</b> 29	<b>Titel:</b> Als Datenmanager/in erwarte ich ein sinnvolles Datenmodell.
<ul style="list-style-type: none"> <li>• Primärschlüssel müssen laufende Nummern sein, die von der Datenbank erstellt werden.</li> <li>• Um Relationen abbilden zu können müssen sinnvolle Fremdschlüssel gewählt werden.               <ul style="list-style-type: none"> <li>– Die Integrität eines Fremdschlüssels muss mit einem Constraint sichergestellt werden.</li> </ul> </li> <li>• Verwenden Sie nur so viele Tabellen wie (unter Einhaltung der dritten Normalform) notwendig.</li> <li>• Verwenden Sie sinnvolle Datentypen für die Tabellenspalten (zum Beispiel für einen Zeitpunkt nicht <b>VARCHAR</b> sondern <b>DATE</b> oder <b>TIMESTAMP</b>).</li> </ul>	

<sup>9</sup><https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fdbf>

<b>ID:</b> 30	<b>Titel:</b> Als Datenmanager/in möchte ich, dass der Zugriff auf die Datenbank sauber durchgeführt wird.
<ul style="list-style-type: none"><li>• SQL-Statements müssen als Prepared Statement ausgeführt werden.</li><li>• Es darf keine SQL-Injection möglich sein.</li><li>• Es sollen nicht mehr Daten aus der Datenbank abgefragt werden, als benötigt werden.<ul style="list-style-type: none"><li>– Zum Beispiel fragen Sie nicht alle Einträge einer Tabelle ab, wenn Sie wissen, dass Sie nur die brauchen, die ein bestimmtes Kriterium erfüllen.</li></ul></li></ul>	

<b>ID:</b> 31	<b>Titel:</b> Als Datenmanager/in erwarte ich eine Trennung von Produktiv- und Testdatenbank.
<ul style="list-style-type: none"><li>• Bei einem Testlauf darf es keinen Zugriff auf die Produktivdatenbank geben.</li><li>• Die Testdatenbank muss sich am Start jedes Testlaufs im gleichen Zustand befinden. Ein vorhergehender Durchlauf der Tests darf den Aktuellen nicht beeinflussen.</li></ul>	

## 3 Implementierung

Der folgende Abschnitt der Angabe stellt einen Leitfaden zur Implementierung dar, er soll Ihnen helfen die richtige Herangehensweise an die Erstellung Ihres Programms zu wählen.

### 3.1 Erste Schritte

Im ersten Schritt sollten Sie Java *OpenJDK 17* herunterladen und installieren. Öffnen Sie das Terminal und führen Sie den Befehl `java -version` aus, um zu überprüfen, ob ihr Betriebssystem auch wirklich Java OpenJDK 17 verwendet. Desweiteren benötigen Sie *Maven*.

Als Nächstes sollten Sie *Node.js 18.x.x* installieren. Überprüfen Sie auch wieder hier, ob Sie die richtige Version installiert haben. Führen Sie dazu den Befehl `node -v` aus. Zuletzt muss noch die *Angular 14.x.x* CLI mit dem Befehl `npm install -g @angular/cli` installiert werden.

Danach sollten Sie Ihre Entwicklungsumgebung einrichten. Nachdem Sie *IntelliJ IDEA* installiert haben, laden Sie das Template aus TUWEL herunter. Entpacken Sie das Template in einen Ordner ihrer Wahl. Das Template enthält die beiden Ordner `backend` und `frontend` sowie die Datei `.gitlab-ci.yml`. Öffnen Sie die beiden Projekte/Ordner jeweils in einem eigenen Fenster in Ihrer Entwicklungsumgebung. Initialisieren Sie ein Git Repository im Ordner, welcher sowohl die beiden Ordner und die Datei enthält. Damit befindet sich die Datei `.gitlab-ci.yml` sowie die beiden Ordner `backend` und `frontend` im Wurzelverzeichnis Ihres Git-Repositories.

#### 3.1.1 Backend

Im Backend wird als Buildsystem Maven verwendet. Eine systemweit installierte Version von Maven wird über den Befehl `mvn` aufgerufen. Unter Windows sollten Sie Git-BASH und nicht CMD/PowerShell verwenden. In den vergangenen Semestern gab es mit letzteren immer wieder Probleme mit Crashes der JVM, wenn viel Text in kurzer Zeit (z.B. bei der Testausführung) ausgegeben wurde. Generell beziehen sich die in diesem Dokument für Windows angegebenen Befehle auf Git-Bash und nicht auf CMD oder PowerShell.

Das zur Verfügung gestellte Template beinhaltet bereits einen einfachen Durchstich durch alle Schichten, von der Persistenz- über die Service- bis zur REST-Schicht und bis in das Frontend. Um das Backend zu starten, führen Sie den Befehl `mvn spring-boot:run` im Ordner `backend` des Templates aus. Alternativ kann die Klasse `java/.../SepmIndividualAssignmentApplication.java` über Ihre IDE gestartet werden. Das Datenbankschema wird beim Start der Anwendung automatisch erstellt. Um die Anwendung zu kompilieren verwenden Sie den Befehl `mvn clean package`. Um initiale Testdaten in die Datenbank einzufügen, können Sie die Applikation mit dem Profil `datagen` starten (die Anwendung wie beschrieben builden und mit `java -Dspring.profiles.active=datagen -jar target/e01234567-0.0.1-SNAPSHOT.jar` ausführen).

Das Backend besteht aus einem Webserver, welcher über die URL `http://localhost:8080` erreichbar ist. Nach dem Senden einer GET-Anfrage mittels Browser oder Curl<sup>10</sup> an die URL `http://localhost:8080/horses` bekommen Sie die Liste der aktuell gespeicherten Pferde von Ihrem Backend-System zurück.

#### 3.1.2 Frontend

Wenn Sie das Backend erfolgreich gestartet haben, wechseln Sie in den Ordner `frontend`. Führen Sie hier den Befehl `npm install` aus, um die Abhängigkeiten herunterzuladen. Anschließend führen Sie `ng serve` aus und wechseln Sie in Ihrem Browser zur URL `http://localhost:4200/`. Wenn Sie ein grünes Feld mit der Meldung „Well done!“ sehen hat die Integration von Frontend und Backend funktioniert, sehen Sie ein rotes Feld mit der Meldung „Error!“, dann ist etwas

---

<sup>10</sup><https://curl.se/>



schief gelaufen. Zusätzlich zu diesem einfachen Durchstich ist in dem Backend-Template bereits das Build und Dependency Management Tooling mittels Maven aufgesetzt.

Nachdem Sie das Projekt nun erfolgreich geöffnet haben sollten Sie als Erstes die Platzhalter für ihre Matrikelnummer `<e01234567>` durch Ihre eigene Matrikelnummer ersetzen. Solche Platzhalter gibt es unter anderem im Backend in `pom.xml` oder im Frontend in `package.json`. Danach können Sie mit der Implementierung beginnen.

### 3.1.3 Aufbau des Templates

Das zur Verfügung gestellte Template folgt einer streng vorgegeben Orderstruktur, die Sie auch nicht ändern sollten. Bestimmte Dateien und Ordner sollten ebenfalls weder umbenannt noch verändert werden. Genauere Informationen dazu finden Sie in der nachfolgenden Liste.

Folgende Ordner und Dateien sind im Backend-Template enthalten:

- `src/main/java` beinhaltet den Quellcode Ihrer Anwendung.
- `src/main/resources` beinhaltet alle Ressourcen auf die Ihr Programm zugreifen muss, wie z.B. die Spring-Konfigurationsdatei `application.yml`.
- `src/test/java` beinhaltet den Quellcode Ihrer Tests.
- `target/` wird automatisch vom Build und Dependencymanagement Tool erstellt und sollte nicht verändert werden.
- `.editorconfig` enthält eine einfache Konfiguration für die Formatierung von Quellcode, die von vielen Texteditoren und IDEs beachtet wird.
- `.gitignore` enthält eine Liste an Dateien die nicht in Git versioniert werden sollen.
- `pom.xml` enthält die Konfiguration für das Build- und Dependencymanagement Tool.
- `README.md` enthält Informationen über das Projekt.

Alle für Sie wichtigen Order und Dateien des Frontend Templates liegen unter `src/app`:

- `component` beinhaltet die Angular Komponenten welche zusammen die Web UI bilden.
- `dto` beinhaltet die Data Transfer Objects.
- `global` beinhaltet globale Konstanten.
- `service` beinhaltet die Services, die auf das REST-Backend zugreifen.
- `app.component.*` diese vier Dateien bilden die Grundstruktur der Angular App. Sie sollten diese Dateien nicht ändern, außer Sie haben schon Erfahrung mit Angular und wollen die App grundlegen anders aufbauen.
- `app.module.ts` ist die Konfigurationsdatei ihres Angular Modules.
- `app.routing.module.ts` in dieser Datei werden die Routen der App verwaltet.

## 3.2 Spring

Spring Boot ist ein Java Framework, dass verschiedene Best Practices und andere Frameworks, wie z.B. das Logging Framework, JUnit oder Jackson vereint. Weiters beinhaltet das Framework den Webserver Tomcat, der das Backend hostet. Die Datei `resources/application.yml` enthält die Konfiguration des Frameworks.

## 3.3 Backend Build- und Dependencymanagement

Als Build- und Dependencymanagement Tool werden Sie Apache Maven einsetzen. Installieren Sie eine aktuelle Version von Maven.

Maven erfüllt zwei verschiedene Aufgaben. Zum einen wird es als *Buildtool* verwendet. Die

Aufgabe eines Buildtools ist die verschiedenen Schritte des Buildprozesses zu automatisieren und damit reproduzierbar zu machen.

Dazu bietet Maven folgende, für Sie wichtigen, Befehle an.

```
mvn clean
```

 löscht alle beim Kompilieren erstellten Dateien.  

```
mvn compile
```

 kompiliert das Programm.  

```
mvn test
```

 lässt alle erstellten Testfälle laufen.  

```
java -jar target/e01234567-0.0.1-SNAPSHOT.jar
```

 startet das Programm.  

```
java -Dspring.profiles.active=datagen -jar target/e01234567-0.0.1-SNAPSHOT.jar
```

 startet das Programm und fügt initiale Testdaten in die Datenbank ein.  

```
mvn clean package
```

 erstellt ein ausführbares Jar File, wenn alle Tests erfolgreich ausführbar sind.  

```
mvn clean package -DskipTests
```

 erstellt ein ausführbares Jar File, ohne die Tests auszuführen.

**Wichtig:** Der Befehl `mvn` kann nur im Hauptordner des (Backend-)Projekts ausgeführt werden.

Die zweite wichtige Aufgabe von Maven ist das *Dependencymanagement*. Dependencymanagement oder auch Abhängigkeitsmanagement wird vor allem dann verwendet wenn das erstellte Programm Abhängigkeiten auf Libraries oder Frameworks hat. Im Fall des Einzelbeispiels sind das beispielweise Abhängigkeiten auf Spring Boot oder H2.

Maven wird über die `pom.xml`-Datei konfiguriert. Sie enthält die Abhängigkeiten sowie weitere Informationen über das zu erstellende Projekt. Bei den vorhandenen Projektinformationen ist für Sie vor allem die `artifactId` relevant. Dieses müssen Sie entsprechend Ihrer Matrikelnummer verändern.

Für die Versionen werden dabei Platzhalter verwendet die unter `properties` definiert sind.

Maven unterscheidet bei den Abhängigkeiten zwischen vier verschiedenen `scopes`.

```
compile
```

 sagt aus, dass die Bibliothek bereits beim Kompilieren benötigt wird.  

```
test
```

 sagt aus, dass die Bibliothek nur zum Testen benötigt wird.  

```
runtime
```

 sagt aus, dass die Bibliothek nur zur Laufzeit benötigt wird.  

```
provided
```

 sagt aus das die Bibliothek extern zur Verfügung gestellt wird.

### 3.4 Angular CLI

Zum Verwalten der Angular Applikation sollten Sie ausschließlich die Angular CLI verwenden. Wenn Sie beispielsweise eine neue Komponente anlegen wollen, besteht diese immer aus mehreren Teilen und muss auch im `app.module.ts` eingetragen werden. Verwenden Sie jedoch die Angular CLI dafür passiert das alles automatisch und hilft Ihnen Fehler zu vermeiden. Um ihre Angular Applikation zu erweitern, wechseln Sie im Terminal in den Ordner in dem Sie die Dateien generieren wollen und führen eine der folgenden Befehle aus:

```
ng generate component
```

 legt eine neu Komponente an.  

```
ng generate service
```

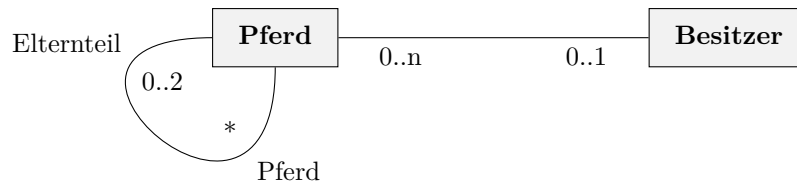
 legt einen neuen Service an.  

```
ng generate class
```

 legt eine neue Klasse an.

### 3.5 Vom Domänenmodell zur Datenbank

Ein Domänenmodell stellt Ihre Sicht/Ihr Verständnis des Datenmodells in der realen Welt dar. Es ist daher keine eins-zu-eins Abbildung des Datenbankdesigns. Um die Domäne etwas zu veranschaulichen, wurde bereits ein Domänenmodell erstellt.



Wie Sie sehen, besteht die Domäne aus zwei Entitäten, die über eine 1 :  $n$ -Beziehung miteinander verbunden sind. Um Relationen auch in der Datenbank abbilden zu können, benötigen Sie Fremdschlüssel. Diese ist ein technisches Implementierungsdetail und daher nicht Teil des Domänenmodells.

Überführen Sie die Entitäten in `CREATE TABLE...` Statements Ihrer Datenbank. Sie müssen und sollen nicht das gesamte Domänenmodell sofort in SQL umsetzen. Am besten ist es, wenn Sie Ihre SQL-Dateien laufend erweitern und verbessern. Wichtig ist, dass Sie regelmäßig die Einhaltung aller Userstories und Techstories überprüfen.

### 3.6 Reihenfolge der Implementierung

Bei der Reihenfolge der Implementierung sollten Sie nach Kundenpriorität vorgehen. Es bringt Ihnen beispielsweise nichts, wenn Sie Daten schön filtern können, wenn es nicht einmal die Möglichkeit gibt neue Daten in Ihr Programm einzuspielen. Als erstes wählen Sie eine Userstory mit möglichst hoher Kundenpriorität zum Beispiel Userstory 1.

Diese Userstory implementieren Sie dann durch alle Backend-Schichten: Persistenz, Service und REST und im Frontend. Vergessen Sie dabei nicht auf die Einhaltung der Techstories.

Um die verschiedenen Schichten voneinander zu entkoppeln verwenden Sie Data Transfer Objects, Exceptions, Interfaces und Dependency Injection. Auch hier empfiehlt es sich wieder zwischen Interface und Implementierung zu trennen. Außerdem ist es wichtig, dass die REST-Schicht nie direkt auf die Persistenzschicht zugreift und umgekehrt.

#### 3.6.1 Persistenz

Die Persistenzschicht regelt den Zugriff auf Ihre Daten, dabei ist es egal ob die Daten im Filesystem oder in einer Datenbank gespeichert sind.

Erstellen Sie als Erstes die benötigten DAOs (Data Access Objects) für Ihre Entitäten beziehungsweise erweitern Sie die bereits erstellten DAOs um die für die zu implementierende Userstory benötigte Funktionalität.

Die geläufigsten DAO-Operationen sind:

- create
- read/find/search
- update
- delete

Erstellen Sie immer nur die Methoden, die sie sicher brauchen werden.

Informieren Sie sich über den Sinn und Zweck von Data Access Objects und die Umsetzung des DAO-Patterns. Stellen Sie sicher, dass Sie verstanden haben, warum das DAO-Pattern verwendet wird und warum es eine Trennung zwischen Interface und Implementierung gibt.

- Interfaces <sup>11</sup>
- DAO-Pattern <sup>12</sup>
- Data Transfer Objects <sup>13</sup>

Die Datenbankverbindung wird von Spring verwaltet und ist im `application.yml` definiert. Dabei werden der default Benutzername „sa“ und ein leerer Passwortstring verwendet.

Um das Datenbankschema Schrittweise zu erweitern, fügen Sie Ihre `CREATE TABLE...` Statements zur Datei `resources/sql/createSchema.sql` hinzu. Diese wird beim Aufbau der Datenbankverbindung automatisch ausgeführt. Um die Datenbank mit Testdaten zu befüllen, fügen Sie Ihre `INSERT INTO...` Statements zur Datei `resources/sql/insertData.sql` hinzu. Sie wird von der Komponente `at/.../persistence/DataGeneratorBean.java` am Start der Anwendung ausgeführt, sofern das Backend mit dem Profil `datagen` gestartet wird.

Sollte die Datenbank noch nicht existieren wird sie automatisch erstellt. In diesem Beispiel wird das Datenbankfile direkt erstellt. Dabei werden folgende Dateien im Hauptverzeichnis des Backendprojekts angelegt:

- `wendydb.mv.db`
- `wendydb.trace.db`

**Wichtig:** Beachten Sie, dass Sie das Verwalten der Datenbankverbindungen entsprechend der *Techstories* umsetzen müssen.

Die bereit gestellte Datenbank URL startet die Datenbank im „Automatic mixed mode“.<sup>14</sup> Dies erlaubt Ihnen sich zu dieser auch z.B. mittels der IDE zu verbinden, unabhängig davon, ob Ihre Anwendung bereits läuft.

### 3.6.2 Service

Die Serviceschicht stellt das Herzstück einer Anwendung dar. Sie beinhaltet normalerweise die komplette Businesslogik. Da diese Applikation sehr einfach ist, wird es so sein, dass die Serviceschicht nicht viel mehr macht als die Daten aus der REST-Schicht an die Persistenzschicht durchzureichen. Allerdings ermöglicht sie es die Anwendung einfacher anzupassen und modularer zu gestalten.

Achten Sie bei der Erstellung Ihrer Serviceklassen darauf, dass Ihre Services nur kleine zusammenhängende Aufgaben erfüllen und vermeiden sehr große monolithische Serviceklassen.

### 3.6.3 REST

Nun fehlt noch die Implementierung der REST-Schicht. Diese Schicht ist die Schnittstelle nach außen und ermöglicht es Ihrem Web Frontend mit Ihrem Backend zu kommunizieren.

## 3.7 Testen

Die erstellte Software zu testen gehört zu den wichtigsten Aktivitäten des Entwicklungsprozesses. Ziel ist es, Fehler so früh wie möglich zu finden, jeder Entwickler ist dabei auch Tester.

Um sicherzustellen, dass Fehler möglichst früh gefunden werden, wird der Testprozess direkt mit dem Entwicklungsprozess verwoben (zum Beispiel mittels Test Driven Development, hierbei ist die Erstellung der Tests sogar eine Vorbedingung zum Erstellen der Implementierung).

Im Rahmen des Einzelbeispiels werden Sie ein Framework zur Testautomatisierung verwenden um Unittests zu stellen. Außerdem empfiehlt es sich im Laufe des Entwicklungsprozesses insbesondere die Kernfunktionalitäten Ihrer Anwendung immer wieder manuell zu testen.

<sup>11</sup><https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

<sup>12</sup><https://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

<sup>13</sup><https://www.oracle.com/technetwork/java/transferobject-139757.html>

<sup>14</sup>[http://www.h2database.com/html/features.html#auto\\_mixed\\_mode](http://www.h2database.com/html/features.html#auto_mixed_mode)

### 3.7.1 Normalfall und Fehlerfall

Beim Testen wird zwischen Tests für den Normalfall und Tests für den Fehlerfall unterschieden. Ein Normalfall (NF) stellt einen Test einer gültigen Eingabe in das System dar. Hierbei wird geprüft ob sich das Programm bei einer richtigen Eingabe korrekt verhält. Ein Fehlerfall (FF) stellt einen Test einer ungültigen Eingabe in das System dar. Hierbei wird geprüft ob sich das Programm bei einer falschen Eingabe korrekt verhält. Ein typisches Beispiel eines Fehlerfalls wäre wenn der/die Nutzer/in ein Pferd mittels ID laden möchte und diese ID im System nicht existiert. In dem Fall muss eine entsprechende Exception geworfen werden mit deren Hilfe der Fehler behandelt beziehungsweise an den/die Nutzer/in kommuniziert werden kann.

**Wichtig:** Vergessen Sie nicht, dass Sie sowohl für den Normalfall als auch den Fehlerfall in jeder Schicht Tests erstellen müssen.

### 3.7.2 Manuelle Tests

Im Rahmen des Einzelbeispiels werden Sie auch manuelle Tests durchführen. Hierbei ist es wichtig, dass sie strukturiert vorgehen.

Gehen Sie dazu die bereits implementierten Userstories Punkt für Punkt durch und prüfen Sie die Einhaltung aller Punkte der Userstory sowie der Techstories. Am einfachsten fällt das Testen wenn Sie sich dabei einen roten Faden zurecht legen. Ein Beispiel, in stark vereinfachter Form, wäre:

1. einfügen.
2. Pferd hinzufügen und zuweisen.
3. Alle Pferde mit anzeigen.
4. Pferd löschen.

### 3.7.3 Automatisierte Tests

Im Rahmen der Umsetzung bestimmter Stories werden Sie automatisierte Tests erstellen. Dafür verwenden Sie in der Einzelphase das Framework JUnit. Die mittels JUnit erstellten Tests sollten Sie regelmäßig über den Befehl `mvn clean test` ausführen.

Wenn Sie im Rahmen der Softwareentwicklung Tests erstellen, machen Sie das häufig nach dem Prinzip des *Test-Driven-Development* (TDD). Dabei erstellen Sie zuerst die Interfaces für die zu testenden Klassen. Danach erstellen Sie Tests sowie leere Implementierungen für die Interfaces. Diese Tests sollen natürlich fehlschlagen, da es noch keine entsprechend vollständige Implementierung gibt. Im nächsten Schritt implementieren Sie das Interface, danach sollten die Tests fehlerfrei ausgeführt werden können.

Diese Vorgehensweise garantiert, dass jeder Programmcode einen Test besitzt und Sie nur den minimal notwendigen Programmcode erstellen, um der Spezifikation zu genügen. Hier zeigt sich wie wichtig es ist, vollständige und korrekte Dokumentation im Code zu haben. Nur aus einer vollständigen Interface-Dokumentation lassen sich ausreichend viele und gute Tests erstellen.

**Wichtig:** Beachten Sie, dass Sie auf Grund der limitierten Zeit für die Einzelphase nur eine geringere Anzahl an Tests erstellen müssen und nicht jede Klasse getestet sein muss.

JUnit führt beim Ausführen der Tests alle mit `@Test` annotierten Methoden aus. Die Reihenfolge der Testausführung ist nicht definiert, daher müssen Sie darauf achten, dass die verschiedenen Testfälle nicht voneinander abhängig sind.

Außerdem sollten Tests so simpel und selbsterklärend wie möglich geschrieben sein. Vermeiden Sie in Tests die Verwendung von Schleifen sowie kompliziertes Exceptionhandling. Achten Sie auch darauf, dass Ihre Testklassen und Methoden aussagekräftige Namen haben.

### 3.8 Versionskontrolle

Im Rahmen der Einzelphase verwenden Sie das Versionskontrollsystem Git um den Quelltext, Ihre Testdaten, Ihre Dokumente sowie Ihre Programmdateien in einem Repository zu verwalten. Git ist ein verteiltes Versionskontrollsystem, das heißt, dass Sie eine lokale Kopie des Repositories auf Ihrem Entwicklungssystem haben. In diesem machen Sie Ihre Commits, die Sie danach auf das zur Verfügung gestellte Repository pushen.

Das Versionskontrollsystem ist ein sehr mächtiges Werkzeug, insbesondere wenn Sie im Team arbeiten. In der Einzelphase benötigen Sie nur ein minimales Set an Befehlen:

- `git init`
- `git status`
- `git clone`
- `git add`
- `git commit`
- `git push`
- `git pull`

**Wichtig:** Beachten Sie, dass aus organisatorischen Gründen die Zugänge zum Versionskontrollsystem ein paar Tage verzögert zur Verfügung gestellt werden. Sie können und sollen schon davor mit Ihrer Implementierung beginnen und auch (lokale) Git commits erstellen.

#### 3.8.1 Initialisieren

Um Git in einem Projekt nutzen zu können, müssen Sie das Projekt zuerst für Git initialisieren. Dazu wechseln Sie in den Ordner den Sie unter Versionskontrolle stellen wollen und geben folgenden Befehl ein:

```
1 $ cd sepm-individual-assignment/  
2 $ git init  
3 Initialized empty Git repository in /projects/sepm-individual-assignment/.git/  
4 $ git status  
5 On branch master  
6  
7 No commits yet  
8  
9 nothing to commit (create/copy files and use "git add" to track)
```

Mit dem Befehl `git status` können Sie sich den Status ihres Repositories anzeigen lassen.

Alternativ zur Initialisierung eines lokalen Ordners können Sie auch ein bereits initialisierten Ordner klonen. Dazu verwenden Sie den Befehl `git clone <url>`. Weitere Informationen finden Sie nach der Freischaltung des Repositories direkt in RESET.

#### 3.8.2 Add & Commit

Um Änderungen permanent zu Ihrem lokalen Repository hinzuzufügen, benötigen Sie zwei Befehle. Als Erstes verwenden Sie den Befehl `git add .` um alle Änderungen an Ihrem Arbeitsverzeichnis zu markieren. Sie können mittels `git add <filename>` auch Änderungen an einzelnen Dateien markieren und mit `git add --interactive` sogar einzelne Zeilen. Nach dem Sie Änderungen so hinzugefügt haben, befinden sich diese im „staging“ Bereich. Um diese Änderungen permanent zu Ihrem Repository hinzuzufügen müssen Sie die Änderungen mit dem Befehl `git commit -m <message>` commiten.

Beim Comitten ist es wichtig eine sinnvolle Commitmessage zu wählen die relevante Informationen für die durchgeführte Änderung enthält.

Versuchen Sie Ihre Commitmessages auf Englisch zu halten und achten Sie dabei darauf, die



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Abbildung 6: Git Commit: <https://xkcd.com/1296/>

Nachricht kurz und prägnant zu formulieren.

Jedes Set an Änderungen soll zudem genau einer Userstory und/oder Techstory zugeordnet sein. So erreichen Sie, dass die Commits übersichtlich bleiben und nur zusammenhängende Inhalte haben.

Beispiele für schlechte Commitmessages:

- „Neue Implementierung hinzugefügt“
- „Added changes to Horse and Sports“
- „Fix Bug“
- „panic!!!“

Beispiele für gute Commitmessages:

- „[US1] Fixed bug where name is not stored correctly.“
- „[1, 15] Added validation to horse.“

### 3.8.3 Remotes, Push & Pull

Nachdem Sie Änderungen zu Ihrem lokalen Repository hinzugefügt haben, müssen Sie diese auf unseren Server befördern. Git verwendet zu solchen Zwecken sogenannte *Remotes*. Ein Remote ist ein anderes Git-Repository, dass im lokalen Repository konfiguriert ist um Daten damit austauschen zu können. Wenn Sie, wie diesem Dokument beschrieben, Ihr lokales Repository erstellt haben, indem sie per `git init` ein neues initialisiert haben, müssen Sie dieses Remote erst einrichten. Mit dem Befehl `git remote add Server <URI>` erstellen Sie ein Remote mit dem Namen **Server**, wobei sie `<URI>` durch die URI des Repositories am Server ersetzen. Wir werden Ihnen die URI zu ihrem Git-Repository im Verlauf des Projekts über RESET zur Verfügung stellen.

Von GitLab werden Sie eine Liste von Optionen ähnlich zu Abbildung 7 erhalten, wenn Sie das Ihnen bereit gestellte Repository in der GUI von GitLab öffnen. Davon ist jedoch nur die *Push an existing Git repository* Variante relevant. Auf keinen Fall sollen Sie *create a new repository* oder *push an existing folder* ausführen. Sie können nämlich nicht mehrere verschiedene lokale Repositories auf den geforderten master Branch vom bereitgestellten remote Repository pushen. Daher wollen Sie eben nicht ein weiteres lokales repository erstellen, sondern folgend *push an existing Git repository* das bereits existierende Repository pushen.

Um Ihre lokalen Commits nun auf den konfigurierten Remote zu befördern, verwenden Sie den Befehl `git push`. Wenn Sie einen Branch zum ersten Mal pushen wollen, ist es notwendig Git darüber zu informieren, zu welchem Remote Sie pushen wollen und unter welchem Namen der

## Command line instructions

You can also upload existing files from your computer using the instructions below.

### Git global setup

```
git config --global user.name "Heimo Stranner"
git config --global user.email "heimo.stranner@inso.tuwien.ac.at"
```

### Create a new repository

```
git clone ssh://git@reset.inso.tuwien.ac.at:2222/heimo.stranner/demo.git
cd demo
git switch -c master
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

### Push an existing folder

```
cd existing_folder
git init --initial-branch=master
git remote add origin ssh://git@reset.inso.tuwien.ac.at:2222/heimo.stranner/demo.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

### Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin ssh://git@reset.inso.tuwien.ac.at:2222/heimo.stranner/demo.git
git push -u origin --all
git push -u origin --tags
```

Abbildung 7: Angebotene Befehle von GitLab

Branch dort aufscheinen soll. Beachten Sie dazu die Anweisungen die Git Ihnen anzeigt, wenn das der Fall ist. Wenn Sie die Anleitung befolgt haben, ist der Remote Branch nun als *Upstream* des lokalen Branch konfiguriert und weitere Push-Befehle werden automatisch dorthin pushen.

Damit Ihre Abgabe gewertet wird, muss diese rechtzeitig auf den Server gepushed werden. Am besten pushen Sie Ihre Änderungen so häufig wie möglich. So stellen Sie sicher das zur Deadline alle relevanten Daten auf unseren Servern vorhanden sind und beugen Datenverlust vor.

Git macht es Ihnen auch möglich von mehreren Rechnern aus am gleichen Code zu arbeiten. Dazu bietet Git, mittels dem Befehl `git pull`, die Möglichkeit alle Änderungen vom Server zu laden und so Ihr lokales Repository auf den aktuellen Stand zu bringen. Wie auch bei Push müssen Sie hier, falls ein Branch noch keinen Upstream hat, einmal explizit sagen, von wo Sie pullen wollen.



### 3.9 Weiterführende Links & Literatur

- Angular 14.x.x
  - <https://angular.io/>
- Angular Tutorial - Tour of Heroes
  - <https://angular.io/tutorial>
- H2 2.1.x
  - <https://www.h2database.com/html/main.html>
- JUnit 5.x.x
  - <http://junit.org/junit5/>
- Maven 3.x.x
  - <https://maven.apache.org/>
- Git 2.x.x
  - <https://git-scm.com/>
  - Deutscher Git Guide: <https://rogerdudler.github.io/git-guide/>
  - Git Cheatsheet: <https://ndpsoftware.com/git-cheatsheet.html>
- Design Patterns
  - <https://sourcemaking.com/>
  - <http://java-design-patterns.com/>
  - Singleton: [https://sourcemaking.com/design\\_patterns/singleton](https://sourcemaking.com/design_patterns/singleton)
  - Interface: <https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>
  - DAO: <https://www.oracle.com/technetwork/java/dataaccessobject-138824.html>
  - DTO: <https://www.oracle.com/technetwork/java/transferobject-139757.html>
- IntelliJ IDEA
  - <https://www.jetbrains.com/idea/download/>
- Spring Boot
  - Spring Boot Documentation Project: <https://spring.io/projects/spring-boot>
- Maven
  - Maven Repository Search: <https://mvnrepository.com/>
  - Maven Central Search: <https://search.maven.org/>
- Vertiefendes Testen
  - Mocking: [http://en.wikipedia.org/wiki/Mock\\_object](http://en.wikipedia.org/wiki/Mock_object)
  - Google Testing Blog: <http://googletesting.blogspot.co.at/>
- Clean Code
  - 2009; Clean Code: A Handbook of Agile Software Craftsmanship
- RESTful API Design
  - <https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fdbf>
- Vorträge und Vorlesungen an der TU
  - 183.239/188.410; Software Engineering und Projektmanagement; VO
  - 180.764; Software-Qualitätssicherung; VU

## 4 Bewertung

Sie erhalten Punkte für die Implementierung jeder einzelnen Userstory. Die Punkte, die Sie für jede Userstory erhalten können, entsprechen den definierten Storypoints für jede der Userstories. Die Gesamtpunkte für die Userstories summieren sich auf 80 Punkte. Die Userstories werden einzeln Punkt für Punkt abgenommen. Mangelhafte Userstories werden mit 0 Punkten beurteilt. Die Einhaltung der Techstories wird Punkt für Punkt abgenommen. Für die Nichterfüllung von Techstories erhalten Sie Punkteabzüge. Die Techstories sind auf 4 Stakeholder aufgeteilt. Jeder Stakeholder hat einen Punktemaximum von 20 Punkte, es wird bei keinem Stakeholder mehr Punkte als dieses Maximum abgezogen. Sollten zum Beispiel bei Techstories eines Stakeholders insgesamt 25 Punkteabzüge zusammenkommen, werden Ihnen tatsächlich nur 20 Punkte abgezogen.

Die einzelnen Teilbereiche einer Userstory sind bezüglich der zu erreichenden beziehungsweise abzuziehenden Punkte einer Story nicht gleichverteilt!

### 4.1 Bestehen der Einzelphase

Um an der Gruppenphase teilnehmen zu können, benötigen Sie in Summe aus dem Einstiegstest (bis zu 10 Punkte) und Einzelbeispiel (bis zu 80 Punkte) mindestens 45 Punkte. Außerdem benötigen Sie mindestens 40 Punkte auf das Einzelbeispiel.

#### 4.1.1 Einstiegstest (max. 10 Punkte)

Für das Lösen des Einstiegstests haben Sie einen Versuch und 60 Minuten Zeit.

Der Einstiegstest wird automatisiert bewertet und sie müssen keine Mindestpunkte erreichen.

#### 4.1.2 Einzelbeispiel (max. 80 Punkte)

Sie müssen mindestens 40 Punkte erreichen.

### 4.2 Einfluss auf die Endnote

Für die Gruppenphase erhalten Sie vom Assistenten eine Note. Diese Note bestimmt  $\frac{3}{4}$  Ihrer Endnote,  $\frac{1}{4}$  Ihrer Endnote macht die Einzelphase aus. Der Notenschlüssel für die Einzelphase entspricht der Standardnotenverteilung.

Prozent	Punkte	Note
100,00 % - 88,00 %	90 - 79	S1
87,99 % - 75,00 %	78 - 67	U2
74,99 % - 63,00 %	66 - 56	B3
62,99 % - 50,00 %	55 - 45	G4

Zur Veranschaulichung noch zwei Beispiele: Wenn Sie in der Einzelphase ein B3 erreichen und in der Gruppenphase ein S1 ergibt das die Notensumme von 1,5 und Sie bekommen als Gesamtnote ein S1. Wenn Sie in der Einzelphase eine U2 und in der Gruppenphase ein B3 erreichen ergibt das die Notensumme von 2,75 und Sie bekommen als Gesamtnote ein B3. Gerundet wird dabei immer auf den nächsten Integer ( $> 0.5$  wird aufgerundet und  $\leq 0.5$  wird abgerundet).

**Wichtig:** Für eine positive Gesamtnote müssen Sie in der Einzelphase und in der Gruppenphase positiv sein.

## 5 Abgabe

### 5.1 Vorbedingungen

- Ihr Projekt ist vollständig (Dokumente, Programmdateien, Testdaten und Quelltext; kein Sourcecode oder Binärdaten von Libraries die Sie über Dependencymanagement beziehen) im SCM vorhanden. **Nur Code und Dokumentation, die im SCM liegen, werden für die Bewertung herangezogen.**

### 5.2 Wichtige Hinweise zur Abgabe

- **Plagiate werden ausnahmslos negativ beurteilt!**
- Änderungen **nach der Deadline** werden **nicht akzeptiert!**
- Das Programm muss lauffähig sein!
- Achten Sie auf die **Vollständigkeit** der Funktionalität Ihres Programms.
- Die **Datenbank** muss mit einer entsprechenden Anzahl an realistischen **Testdatensätzen** befüllt sein (**zumindest 10 Stück** pro Domänenobjekt, achten Sie ebenfalls darauf, dass Relationen in den Testdaten vorhanden sind).

### 5.3 Nach der Bewertung

Nach der Bewertung werden Sie einer Forschungsgruppe zugeordnet, dabei versuchen wir nach Möglichkeit, Ihren Forschungsgruppenwunsch zu berücksichtigen. Je nach Forschungsgruppe haben Sie nach der Zuordnung, **möglichst vor dem ersten Tutorenmeeting**, Folgendes zu tun:

- **QSE:** Erarbeiten Sie einen eigenen Projektvorschlag, den Sie präsentieren müssen.
- **INSO:** Machen Sie sich mit der Umsetzung des Beispielprojekts vertraut.

**Viel Spaß und Erfolg beim Einzelbeispiel aus der SE&PM Laborübung!**

---

**Deadline: Mittwoch, 26. Oktober 2022, 18:00**