Einzelbeispiel (SE&PM/JAVA), Sommersemester 2020

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 11
Java-Framework	Spring Boot 2.2.x
JavaScript Runtime	Node.js 12.x.x
Frontend Framework	Angular 9.x.x
Datenbank	H2 1.4.x
Test-Framework	JUnit 5.x.x
Build & Dependency Management	Maven 3.6.x
Versionierung	Git 2.x.x

Als Entwicklungsumgebung empfehlen wir IntelliJ IDEA 2019.x.x[1]. Sämtliche Tools stehen im Informatiklabor zur Verfügung und unsere Tutor/innen bieten hierfür Unterstützung an.

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. Zusätzlich gibt es betreute Laborzeiten, zu denen unsere Tutor/innen im Informatiklabor anwesend sind, um vor Ort bei der Problemlösung behilflich zu sein. ***Wichtig:*** *Je genauer Sie Ihre Fragen formulieren, desto besser kann Ihnen geholfen werden.*

Labor Fragestunden: Termine und Anwesenheitszeiten finden Sie im 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

[1] https://www.jetbrains.com/idea/download/

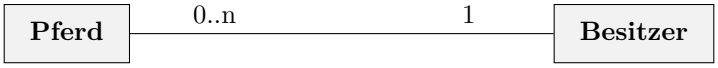
Inhaltsverzeichnis

2 Angabe

Sie sind als Softwareentwickler/in in einem kleinen österreichischen Unternehmen tätig. Ihr Unternehmen wurde beauftragt eine Plattform für *Wendy's Friends* zur Verwaltung von Pferden und Besitzer/innen 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. Ihr/e technische/r Architekt/in und Ihr/e Qualitätsmanager/in haben zusätzliche Anforderungen an die Umsetzung der Software formuliert. Userstories, die von einer fehlerhaften Implementierung betroffen sind, werden nicht abgenommen.
In der Aufgabenstellung wird zwischen Userstories und Techstories unterschieden. Userstories beschreiben konkrete Anwendungsfälle der zu erstellenden Software aus Sicht eines bestimmten Stakeholders. Techstories enthalten Anforderungen an die Implementierung sowie weitere Implementierungsdetails.

2.1 Domänenmodell

Um die Domäne etwas zu veranschaulichen, wurde bereits ein Domänenmodell erstellt.



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 ausgedruckt zum Abgabegespräch mitnehmen. 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.

Wichtig: *Die Aufschlüsselung der Userstories auf Stakeholder dient ausschließlich dem leichteren Verständnis. Sie sollen kein Mehrbenutzersystem entwickeln. Weiters wird in der Einzelphase auch auf Log-in bzw. Authentifizierung verzichtet.*

2.5.1 Pferdebesitzer/in

ID: 01	**Titel:** Als Pferdebesitzer/in möchte ich neue Pferde im System anlegen können.

- Jedes angelegte Pferd muss in einer Datenbank gespeichert werden.
- Zu jedem Pferd müssen dabei folgende Werte gespeichert werden:
  - Name (verpflichtend)
  - Beschreibung (optional)
  - Bewertung (1-5, verpflichtend)
  - Geburtsdatum (verpflichtend)
  - Zeitpunkt der Erstellung (automatisch)

Storypoints: 10

ID: 02	**Titel:** Als Pferdebesitzer/in möchte ich außerdem immer ein Foto und die Rasse meines Pferdes abspeichern können.

- Zu jedem Pferd soll ein Foto im Format JPEG oder PNG abgelegt werden.
- Außerdem soll zu jedem Pferd die Rasse abgespeichert werden, wobei ausschließlich folgende Rassen unterstützt werden:
  - ARABIAN
  - MORGAN
  - PAINT
  - APPALOOSA

Storypoints: 4

ID: 03	**Titel:** Als Pferdebesitzer/in möchte ich Pferde bearbeiten können.

- Beim Bearbeiten von Pferden müssen alle erfassten Werte geändert werden können.
- Jedem Pferd muss ein/e neue/r Besitzer/in zugewiesen werden können.
- Bei jedem Pferd muss vermerkt sein, wann es zuletzt bearbeitet wurde.
- Die Zeitpunkte der Erstellung sowie der letzten Bearbeitung dürfen von Benutzer/innen nicht verändert werden können.

Storypoints: 10

ID: 04	**Titel:** Als Pferdebesitzer/in möchte ich Pferde aus dem System löschen können.

- Das Löschen von Pferden darf nicht rückgängig gemacht werden können. Möchte man ein Pferd erneut im System haben, muss es neu angelegt werden.

Storypoints: 4

ID: 05	**Titel:** Als Pferdebesitzer/in möchte ich Pferde suchen können.

- Pferde sollen nach folgenden Parametern gesucht werden können:
  - Nach einem Teil des Namens
  - Nach einem Teil der Beschreibung
  - Nach der Rasse
  - Nach der Bewertung
  - Nach einem bestimmten Datum, wobei der Geburtstag des Pferdes vor oder an diesem Datum liegen muss.
- Dabei können die Parameter beliebig kombiniert werden.
- Suche nach textuellen Parametern muss Groß- und Kleinschreibung ignorieren.

Storypoints: 10

2.5.2 Plattformbetreiber/in

ID: 06	**Titel:** Als Plattformbetreiber/in möchte ich neue Besitzer/innen im System anlegen können.

- Jeder/Jede angelegte Besitzer/innen muss in einer Datenbank gespeichert werden.
- Zu jedem/jeder Besitzer/inn müssen dabei folgende Werte gespeichert werden:
  - Name (verpflichtend)
  - Zeitpunkt der Erstellung (automatisch)

Storypoints: 8

ID: 07	**Titel:** Als Plattformbetreiber/in möchte ich Besitzer/innen bearbeiten können.

- Beim Bearbeiten von Besitzer/innen müssen alle Werte geändert werden können.
- Bei jedem/jeder Besitzer/innen muss vermerkt sein, wann er zuletzt bearbeitet wurde.
- Die Zeitpunkte der Erstellung sowie der letzten Bearbeitung dürfen von Benutzer/innen nicht verändert werden können.

Storypoints: 10

ID: 08	**Titel:** Als Plattformbetreiber/in möchte ich Besitzer/innen aus dem System löschen können.

- Das Löschen von Besitzer/innen darf nicht rückgängig gemacht werden können. Möchte man eine/n Besitzer/in erneut im System haben, muss er/sie neu angelegt werden.
- Pferdebesitzer/innen dürfen nicht gelöscht werden, solange sie ein Pferd zugewiesen haben.

Storypoints: 6

ID: 09	**Titel:** Als Plattformbetreiber/in möchte ich Besitzer/innen suchen können.

- Besitzer/innen sollen nach folgenden Parametern gesucht werden können:
  - Nach einem Teil des Namens

Storypoints: 10

ID: 10	**Titel:** Als Plattformbetreiber/in möchte ich sehen welche Pferde zu welchen Besitzer/innen gehören.
• Es soll die Möglichkeit geben zu sehen welche Pferde zu welchen Besitzer/innen gehören.	
Storypoints: 8	

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.

2.6.1 Qualitätsmanager/in

ID: 11	**Titel:** Als Qualitätsmanager/in möchte ich, dass das Programm Logfiles schreibt.

- Das Logfile muss alle auftretenden Fehler in Form einer Errormessage enthalten.
- Das Logfile muss alle vom User durchgeführten Aktionen als Infomessage enthalten.
- Das Logfile muss alle fürs Debugging relevanten Informationen als Debugmessage enthalten.
- Das Logfile muss täglich rotiert werden. Der Dateiname jedes einzelnen Logfiles muss das Datum enthalten.
- Das Programm darf ausschließlich über das Logging Framework Ausgaben auf die Standardausgabe (stdout) sowie die Standardfehlerausgabe (stderr) schreiben.

Storypoints: 5

ID: 12	**Titel:** Als Qualitätsmanager/in möchte ich, dass das Programm in Englisch geschrieben und gut dokumentiert ist.

- Der Quellcode (Klassennamen, Variablennamen sowie Methodennamen) muss auf Englisch verfasst sein.
- Für alle Interfaces und Deklarationen in Interfaces muss JavaDoc existieren.
- Die JavaDoc enthält Informationen zu:
  - Übergabeparametern.
  - Rückgabewerten.
  - möglichen Fehlerfällen und Exceptions.
- Die JavaDoc ist auf Englisch geschrieben.

Storypoints: 7

ID: 13	**Titel:** Als Qualitätsmanager/in möchte ich, dass Kernfunktionalitäten des Programms getestet sind.
• Es müssen mind. vier Tests für jede Backendschicht (REST, Service, Persictence) erstellt werden. • Es müssen sowohl Positivtests (Normalfall) als auch Negativtests (Fehlerfall) erstellt werden. • In Summe sollen 12 neue Tests geschrieben werden. • Sie können sowohl Unit als auch Integration Tests erstellen.	
Storypoints: 8	

ID: 14	**Titel:** Als Qualitätsmanager/in möchte ich, dass alle Eingaben validiert werden.
• Eingabeparamter (Zahlenwerte oder Textwerte) müssen den Userstories entsprechend validiert werden. • Bei der Validierung muss darauf geachtet werden, ob Parameter verpflichtend oder optional sind. • Es muss zumindest in der Serviceschicht validiert werden.	
Storypoints: 7	

2.6.2 Technische/r Architekt/in

ID: 15	**Titel:** Als Technische/r Architekt/in erwarte ich einen sauberen Git Workflow.
• Es müssen regelmäßig Commits erstellt werden. • Ein Commit gehört meistens zu genau einer Userstory und/oder Techstory. Es kann Überscheidungen von Stories geben, jedoch sollte dies nicht die Regel sein. • Jeder Commit enthält eine aussagekräftige Commitmessage.	
Storypoints: 7	

ID: 16	**Titel:** Als Technische/r Architekt/in erwarte ich die Verwendung eines Build und Dependency Managements.

- Abhängigkeiten des Backends müssen, mittels Maven verwaltet werden.
- Das Backend muss über Maven gebaut, gestartet und getestet werden können.
- Das Backend muss mit dem Befehl:
  `mvnw clean compile`
  erfolgreich gebaut werden können.
- Das Backend muss mit dem Befehl:
  `mvnw clean test`
  erfolgreich getestet werden können.
- Das Backend muss mit dem Befehl:
  `mvnw clean compile spring-boot:run`
  gestartet werden können.
- Für das Backend muss mit dem Befehl:
  `mvnw clean package`
  ein ausführbares jar File erstellt werden können.
- Abhängigkeiten des Frontends müssen, mittels npm verwaltet werden.
- Das Frontend muss mit dem Befehl:
  `ng serve`
  gestartet werden können.

Storypoints: 7

ID: 17	**Titel:** Als Technische/r Architekt/in erwarte ich eine saubere Umsetzung der Datenbank.

- SQL-Statements müssen als Prepared Statement ausgeführt werden.
- Primärschlüssel müssen laufende Nummern sein, die von der Datenbank erstellt werden.
- Um Relationen abbilden zu können müssen sinnvolle Fremdschlüssel gewählt werden.

Storypoints: 6

ID: 18	**Titel:** Als Technische/r Architekt/in erwarte ich eine gute Umsetzung der Programmarchitektur.

- Das Programm muss eine saubere Schichtentrennung aufweisen.
- Die Austauschbarkeit der Schichten muss gegeben sein.
- Zur Trennung der Schichten muss das Interface[2] Pattern verwendet werden.
- Außerdem dürfen andere Komponenten nie direkt instanziiert werden, sondern müssen mittels Dependency Injection[3] injiziert werden.
- Suchkriterien sollen nicht als einzelne Parameter zwischen den Schichten übertragen werden, sondern gekapselt in einem Objekt.
- Verwenden von Exceptions zum Transportieren von Fehlern.

Storypoints: 10

ID: 19	**Titel:** Als Technische/r Architekt/in erwarte ich vorausschauenden Umgang mit Ressourcen.

- Schließen von externen Resources (z.B. Datenbank-Verbindung), sofern vorhanden.
- Suchen müssen aus Performancegründen in der Datenbank vorgenommen werden.

Storypoints: 3

ID: 20	**Titel:** Als Technische/r Architekt/in erwarte ich eine durchdachte Fehlerbehandlung.

- Je nach Art des Fehlers wird eine passende Exception gewählt.
- Exceptions werden ausschließlich zum Transportieren von Fehlern verwendet.
- Fehler müssen im Log aufscheinen.
- Fehler müssen entweder behandelt oder weitgereicht werden.
- Das Programm darf zu keinem Zeitpunkt einfach abstürzen.

Storypoints: 10

[2] https://docs.oracle.com/javase/tutorial/java/concepts/interface.html
[3] https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

ID: 21	**Titel:** Als Technische/r Architekt/in erwarte ich eine REST-konforme[4] Umsetzung des Backend.

- Für jede Response muss der korrekte HTTP-Statuscode zurückgegeben werden. Dabei soll vor allem zwischen 2xx, 4xx und 5xx unterschieden werden.
- Bei jedem Request und Response sollen nur Informationen übertragen werden, die auch wirklich benötig werden.
- Für jeden HTTP-Request wird immer die richtige HTTP-Anfragemethode (Verb) verwendet.
- Das Backend an sich speichert niemals den Status eines Request ab. Der Zustand der Applikation wird ausschließlich in der Datenbank gespeichert.
- Uniform Resource Identifiers (URI) bestehen hauptsächlich aus Nomen im Plural.

Storypoints: 10

[4] https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fcdbf

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 11 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 11 verwendet. Als nächstes sollten Sie Node.js 12.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 9.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. Öffnen Sie die beiden Projekte jeweils in einem eigenen Fenster in Ihrer Entwicklungsumgebung.

Im Backend wird als Buildsystem Maven verwendet. Es kann sowohl systemweit installiert werden, als auch über den Maven-Wrapper verwendet werden, der im Projekt bereits vorhanden ist. Eine systemweit installierte Version wird über den Befehl `mvn` aufgerufen, der Wrapper über die `mvnw`-Skripts. Hier gibt es leichte Unterschiede wenn Sie Windows oder ein unixoides System (z.B. GNU/Linux, FreeBSD, macOS, ...) verwenden. Auf Windows kann der Wrapper einfach mit `mvnw` aufgerufen werden, sofern sie sich im Wurzelverzeichnis des Projekts befinden. Auf einem unixartigen System muss der Pfad zum Skript üblicherweise explizit angegebenen werden (`./mvnw`). *Achtung:* Die Windows-Version des Maven-Wrappers hat Probleme mit Leerzeichen in Pfaden. Windows-User die beabsichtigen der Wrapper zu verwenden, sollten dies bei der Wahl des Arbeitsverzeichnisses beachten.

Generell beziehen sich die in diesem Dokument für Windows angegebenen Befehle auf cmd und nicht auf 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 `mvnw spring-boot:run` im backend Ordner des Templates aus. Alternativ kann die Klasse `java/.../WendysRennpferdeApplication.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 Postman[5] an die URL http://localhost:8080/owners/1 bekommen Sie den Besitzer mit der ID 1 von Ihrem Backend-System zurück.
Wenn das funktioniert hat, wechseln Sie in den Ordner `wendys-friends` der im `frontend` Ordner liegt. Führen Sie hier den Befehl `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 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

[5] https://www.getpostman.com/downloads/

für ihre Matrikelnummer <e01234567> durch Ihre eigene Matrikelnummer ersetzen. Danach können Sie mit der Implementierung beginnen.

3.1.1 Aufbau des Template

Das zur Verfügung gestellte Template folgt einer streng vorgegeben Orderstruktur, diese sollte von Ihnen nicht geändert werden. 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:

`.mvn/` beinhaltet das Build- und Dependencymanagement Tool. Dieser Ordner sollte nicht verändert werden.

`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.

`.gitignore` enthält eine Liste an Dateien die nicht in Git versioniert werden sollen.

`mvnw` ist nur für Unix-Nutzer/innen (z.B. Linux oder macOS) relevant, es führt das Build- und Dependencymanagement Tool aus. Die Datei sollte nicht verändert werden.

`mvnw.cmd` ist nur für Windows-Nutzer/innen relevant, es führt das Build- und Dependencymanagement Tool aus. Die Datei sollte nicht verändert werden.

`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 `wendys-friends/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. Das Beispielprojekt enthält bereits eine mitgelieferte Version von Apache Maven, da Maven in Java

programmiert ist und in der JVM läuft, müssen Sie nichts weiteres installieren.

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.

`mvnw clean` löscht alle beim Kompilieren erstellten Dateien.
`mvnw compile` kompiliert das Programm.
`mvnw 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.
`mvnw clean package` erstellt ein ausführbares Jar File, wenn alle Tests erfolgreich ausführbar sind.
`mvnw clean package -DskipTests` erstellt ein ausführbares Jar File, ohne die Tests auszuführen.

Wichtig: Der Befehl `mvnw` kann nur im Hauptordner des Projekts ausgeführt werden. Unter Linux und macOS muss `./mvnw` mit führenden `./` eingegeben 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.

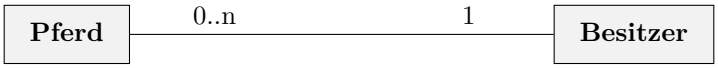
3.4 Angular CLI

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 eine $1 : n$ Relation auch in der Datenbank abbilden zu können, benötigen Sie einen 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 Injections. 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 [6]
- DAO-Pattern [7]
- Data Transfer Objects [8]

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 im Homeverzeichnis des Users 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.*

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.

[6] https://docs.oracle.com/javase/tutorial/java/concepts/interface.html
[7] https://www.oracle.com/technetwork/java/dataaccessobject-138824.html
[8] https://www.oracle.com/technetwork/java/transferobject-139757.html

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 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. Besitzer mit Namen Joe anlegen
2. Pferd für Joe hinzufügen
3. Alle Pferde von Joe 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 `mvnw 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:

```
1 git init
2 git status
3 git clone
4 git add
5 git commit
6 git push
7 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.*

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. Nach dem „added“ befinden sich die Änderungen im „staging“ Bereich. Um diese Änderungen permanent zu Ihrem Repository hinzuzufügen müssen Sie die Änderungen mit dem Befehl `git commit -m <message>` commiten.
Beim Commiten ist es wichtig eine sinnvolle Commitmessage zu wählen die relevante Informationen für die durchgeführte Änderung enthält.

Abbildung 1: Git Commit: https://xkcd.com/1296/

Versuchen Sie Ihre Commitmessages auf Englisch zu halten und achten Sie dabei darauf, die 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 Owners“
- „Fix Bug“

Beispiele für gute Commitmessages:

- „Fixed bug where name is not stored correctly [story: 1]“
- „Added validation to horse [1, 14]“

3.8.3 Push & Pull

Nachdem Sie Änderungen zu Ihrem lokalen Repository hinzugefügt haben müssen Sie diese auf den Server pushen, das machen Sie mit dem Befehl `git push`. Damit Ihre Abgabe gewertet wird muss diese rechtzeitig auf den Server gepushed werden. Am besten Sie pushen 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 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.

3.9 Weiterführende Links & Literatur

- Angular 9.x.x
  - https://angular.io/
- Angular Tutorial - Tour of Heroes
  - https://angular.io/tutorial
- H2 1.4.x
  - https://www.h2database.com/html/main.html
- JUnit 5.x.x
  - http://junit.org/junit5/
- Maven 3.6.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
  - 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 2019.x.x
  - 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
  - Google Testing Blog: http://googletesting.blogspot.co.at/
- Clean Code
  - 2009; Clean Code: A Handbook of Agile Software Craftsmanship
- 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 einzelner 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 maximalen Punkteabzüge einer Techstory entsprechen den definierten Storypoints für die jeweilige Techstory.
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), Live Beispiel (bis zu 30 Punkte) und Einzelbeispiel (bis zu 80 Punkte) mindestens 60 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 Live Beispiel (max. 30 Punkte)

Für das Lösen des Live Beispiels haben Sie 50 Minuten Zeit, während der Sie ein bestehendes Programm um Funktionalität erweitern müssen. Dabei sollen Sie zeigen, dass Sie die Technologien der Einzelphase beherrschen.
Das Live Beispiel wird automatisiert bewertet und Sie müssen keine Mindestpunkte erreichen.

4.1.3 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 ¾ Ihrer Endnote, ¼ Ihrer Endnote macht die Einzelphase aus. Der Notenschlüssel für die Einzelphase entspricht der Standardnotenverteilung.

Prozent	Punkte	Note
100,00 % - 88,00 %	120 - 105	S1
87,99 % - 75,00 %	104 - 90	U2
74,99 % - 63,00 %	89 - 75	B3
62,99 % - 50,00 %	74 - 60	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 ≤ 0.5 wird abgerundet).

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

5 Abgabegespräch

5.1 Vorbedingungen

- Sie sind in RESET zu einem **Abgabegespräch angemeldet**.
- 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.**
- Die Abgaben finden ausschließlich auf den Laborrechnern statt!

5.2 Ablauf des Abgabegesprächs

1. Live-Beispiel (Programmieraufgabe)
   - **Selbständiges Lösen einer Programmieraufgabe** (max. 50min)
   - **Automatisierte Bewertung**
2. Abgabe des Einzelbeispiels
   - **Produktcheck**: Der/die Tutor/Tutorin lässt sich alle **Userstories** von Ihnen erklären und vorführen, außerdem prüft er/sie die Einhaltung der **Techstories** und führt selbständig Tests durch.
   - **Verständnisfragen**: Der/die Tutor/Tutorin überprüft Ihr Verständnis zum Produkt, zu den Technologien sowie zur Entwicklungsumgebung.

5.3 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.
- Während der Abgabe müssen Sie den **Sourcecode** Ihres Programms an beliebigen Stellen jederzeit und ohne Vorbereitungszeit flüssig **erklären** können.
- 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).

5.4 Nach dem Abgabegespräch

Nach dem Abgabegespräch 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.

Wichtig: *Melden Sie sich auch* ***rechtzeitig für ein Abgabegespräch*** *an. Nutzen Sie die Hilfestellungen durch das* ***TUWEL-Forum*** *und unsere* ***Tutor/innen****.*

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

Deadline: Sonntag, 29. März 2020, 23:55h