2024

[Entwicklung eines DOJO-Kampfschulen Software-Systems]

ÜBUNG 6 – SOFTWAREMANAGMENT I – BSWE

DÖTZL ANDREAS, MIKLOS KOMLOSY

Entwicklung eines Software-Systems für die Verwaltung von DOJO-Kampfschulen unter Einsatz von UML und BPMN

Phase 1 – Analyse

1. Definiert die Anforderungen und Ziele des Use Case:

Anforderungserfassung:

Unser Kunde verwaltet bereits 3 DOJOs (Wiener Neustadt, Bad Blumau, Hartberg) und erweitert im Jahr 2025 diese um mindestens 5 neue Standorte. [i]

2. Use Case

Entwicklung eines Verwaltungssystems als umfassende Softwarelösung, die speziell für die Organisation und Verwaltung von DOJO-Kampfschulen entwickelt wird. Ihr Hauptzweck liegt in der effizienten Verwaltung der Schulen, Schüler, Trainer, Kurse, Prüfungen und Wettkämpfe. Die Anwendung ermöglicht es den Senseis (Trainern), alle Aspekte des Schulbetriebs zu steuern und zu koordinieren.

Die Software soll folgende Funktionalitäten besitzen:

- Verwaltung von Senseis/Schiedsrichter
- Verwaltung von Schülern
- Organisation von Kursen/Unterrichtsstunden
- Organisation von Prüfungen
- Planung/Organisation und Durchführung von Wettkämpfen
- Bewertung der Teilnehmer von Wettkämpfen
- Verwaltung von Standorten
- Verwaltung von Gürtelfarben (Level)

Schüler können an mehreren DOJOs trainieren

Es gibt bereits spezielle DOJO-Verwaltungssoftware auf dem Markt, jedoch gibt es bei diesen Anbietern keine Multi Property Lösung, was bedeutet, dass ein Schüler in jedem DOJO extra verwaltet werden muss. Diese Anforderung ist für den Kunden ein Must-Have Kriterium, da in der Zukunft eine noch größere Expansion angedacht ist.

Ziele:

1. Effiziente Verwaltung von DOJO-Schulen: Die Software soll eine zentrale Plattform bieten, um alle DOJO-Schulen effizient zu verwalten.
2. Transparente Schülerverwaltung: Schüler werden von den Senseis per E-Mail benachrichtigt und verwenden ein Formular zur Anmeldung, Prüfungsanmeldung und Wettkampfanmeldung. Schüler werden durch Sensei mithilfe eines ausgefüllten Formulars registriert, verwaltet und ihren Kursen zugewiesen.
3. Organisation von Wettkämpfen: Die Software soll die Planung und Organisation von Wettkämpfen vereinfachen.

NICHT-Ziele:

1. Die Software steht aktuell nur den Senseis zur Verfügung und es ist aktuell nicht angedacht, dass Schüler auch Zugriff auf die Software bzw. Teile davon erhalten. Das bedeutet, dass keine Termine direkt von Schülern über die Software gebucht werden können.

3. Definiert drei funktionale Anforderungen nach SMART-Prinzip: [1]

Anforderung: Verwaltung von Schüleranmeldungen zu Unterrichtsstunden[ii]

Was das System tun soll: Das System soll Schüleranmeldungen zu verschiedenen Unterrichtsstunden in verschiedenen Dojos verwalten.
Messbarkeit: Das System zeigt für Sensei die Schülerdaten an, um anzuzeigen, wer an welchen Unterrichtsstunden teilnimmt. [iii]
Erreichbarkeit: Das System kann Schüler entsprechend der Anmeldeinformationen den Unterrichtsstunden zuweisen und verwalten, alles innerhalb des festgelegten Zeitrahmens.
Relevanz: Wichtig für die Geschäftsziele des Dojo-Schulsystems, um die Teilnahme der Schüler effizient zu verwalten und einen reibungslosen Schulbetrieb sicherzustellen.
Zeitgebundenheit: Das System muss durch Sensei eingegebenen Schüleranmeldungen rechtzeitig verarbeiten, um den Schulplanungsprozess effektiv zu unterstützen.

Anforderung: Prüfungsverwaltung und Gürtelfarbenaktualisierung

Was das System tun soll: Das System soll Prüfungen für Schüler organisieren und ihre Gürtelfarben entsprechend ihren Leistungen aktualisieren.
Messbarkeit: Das System muss Prüfungen verfolgen und die Ergebnisse dokumentieren, um zu zeigen, welche Schüler welche Prüfungen abgelegt haben und welche Gürtelfarben sie danach haben.
Erreichbarkeit: Das System sollte Prüfungen organisieren, Ergebnisse bewerten und die Gürtelfarben der Schüler aktualisieren können, alles innerhalb des festgelegten Zeitrahmens. [iv]
Relevanz: Wichtig, um den Fortschritt der Schüler zu überwachen und ihre Leistungen anzuerkennen...
Zeitgebundenheit: Das System muss Prüfungen rechtzeitig planen, durchführen und Ergebnisse verarbeiten, damit Sensei die Fortschritte ihren Schülern nachverfolgen und sich auf die nächste Ausbildungsphase vorbereiten können.

Anforderung: Organisation und Verwaltung von Wettkämpfen

Was das System tun soll: Das System soll Wettkämpfe planen und verwalten, einschließlich Veranstaltungsorten, -daten und -teilnehmern. [v]
Messbarkeit: Das System muss verschiedene Wettkämpfe organisieren und anzeigen können, inklusive Veranstaltungsorten, -daten und Teilnehmerregistrierungen.
Erreichbarkeit: Das System sollte die benötigten Funktionen bereitstellen, um Wettkämpfe effizient zu organisieren und zu verwalten, einschließlich der Planung von Veranstaltungsorten und -daten sowie der Teilnehmerregistrierung.
Relevanz: Die Anforderung ist relevant, da Wettkämpfe eine wichtige Rolle im Trainingsprozess spielen und die Fähigkeiten und den Geist der Schüler fördern.
Zeitgebundenheit: Das System muss Wettkämpfe rechtzeitig organisieren und verwalten, um sicherzustellen, dass alle Vorbereitungen getroffen werden können und die Teilnehmer informiert sind.

4. Definiert drei nicht - funktionale Anforderungen nach SMART/INVEST Prinzip [2]

Anforderung: Benutzerfreundlichkeit und Benutzererfahrung

Was das System tun soll: Das System soll eine benutzerfreundliche Oberfläche bieten und eine positive Nutzererfahrung sicherstellen.
Messbarkeit: Das System sammelt Benutzerfeedback und führt Analysen zur Benutzerfreundlichkeit durch. [vi]
Erreichbarkeit: Das System kann Designprinzipien und Best Practices für Benutzerfreundlichkeit implementieren und eng mit Benutzern zusammenarbeiten, um eine positive Benutzererfahrung zu gewährleisten. [vii]
Relevanz: Die Anforderung unterstützt die Geschäftsziele des Dojo-Schulsystems, indem sie die Effizienz verbessert, die Akzeptanz erhöht und die effektive Nutzung des Systems fördert.
Zeitgebundenheit: Das System muss von Anfang an eine positive Benutzererfahrung bieten und kontinuierlich verbessert werden, indem regelmäßig Benutzertests und Feedback-Schleifen durchgeführt werden.

Die SMART Methode verstehen und anwenden mit Beispiel (scribbr.de)

Funktionale Anforderungen: Beispiele und Vorlagen – Visure-Lösungen (visuresolutions.com)

[2] https://chat.openai.com/ Prompt: Schreib meinen Text so kurz wie möglich ohne Inhalt Verlust.

***Anforderung: Wartbarkeit und Erweiterbarkeit*[3]**

Was das System tun soll: Das System soll leicht wartbar und erweiterbar sein, um zukünftige Änderungen effizient zu ermöglichen, sowohl in der Architektur als auch im Code.
Messbarkeit: Die Wartbarkeit und Erweiterbarkeit können anhand von Kennzahlen wie der Zeit für neue Funktionen, Änderungskomplexität und Anzahl der Codeanpassungen bewertet werden.
Erreichbarkeit: Das System soll eine modulare und gut dokumentierte Codebasis verwenden, um Änderungen zu erleichtern, ohne die Integrität des Systems zu beeinträchtigen, durch bewährte Softwareentwicklungsmethoden.
Relevanz: Wichtig, um sicherzustellen, dass das System flexibel ist und sich an die sich ändernden Anforderungen des Schulbetriebs anpassen kann, um neue Funktionalitäten zu integrieren.
Zeitgebundenheit: Das System muss von Anfang an wartbar und erweiterbar sein, mit regelmäßigen Code-Reviews, Tests und Aktualisierungen, um kontinuierliche Verbesserungen zu ermöglichen.

Anforderung: Zuverlässigkeit und Ausfallsicherheit

Was das System tun soll: Das System muss zuverlässig und ausfallsicher sein, um kontinuierlichen Schulbetrieb sicherzustellen und Ausfälle zu minimieren.
Messbarkeit:[4] Zuverlässigkeit und Ausfallsicherheit können anhand von Kennzahlen wie MTBF (Mean Time Between Failures), MTTR (Mean Time to Recovery) und der Anzahl unerwarteter Ausfälle gemessen werden.
Erreichbarkeit:[5] Das System kann Redundanzmechanismen, Failover-Systeme und Backups implementieren, um kontinuierliche Verfügbarkeit zu gewährleisten.
Relevanz: Wichtig, um den Schulbetrieb nicht zu beeinträchtigen und Unannehmlichkeiten zu vermeiden, da Ausfälle zu Unterbrechungen führen können.
Zeitgebundenheit: Das System muss von Anfang an zuverlässig und ausfallsicher sein, mit regelmäßigen Tests und Aktualisierungen, um eine kontinuierliche Verfügbarkeit zu gewährleisten.

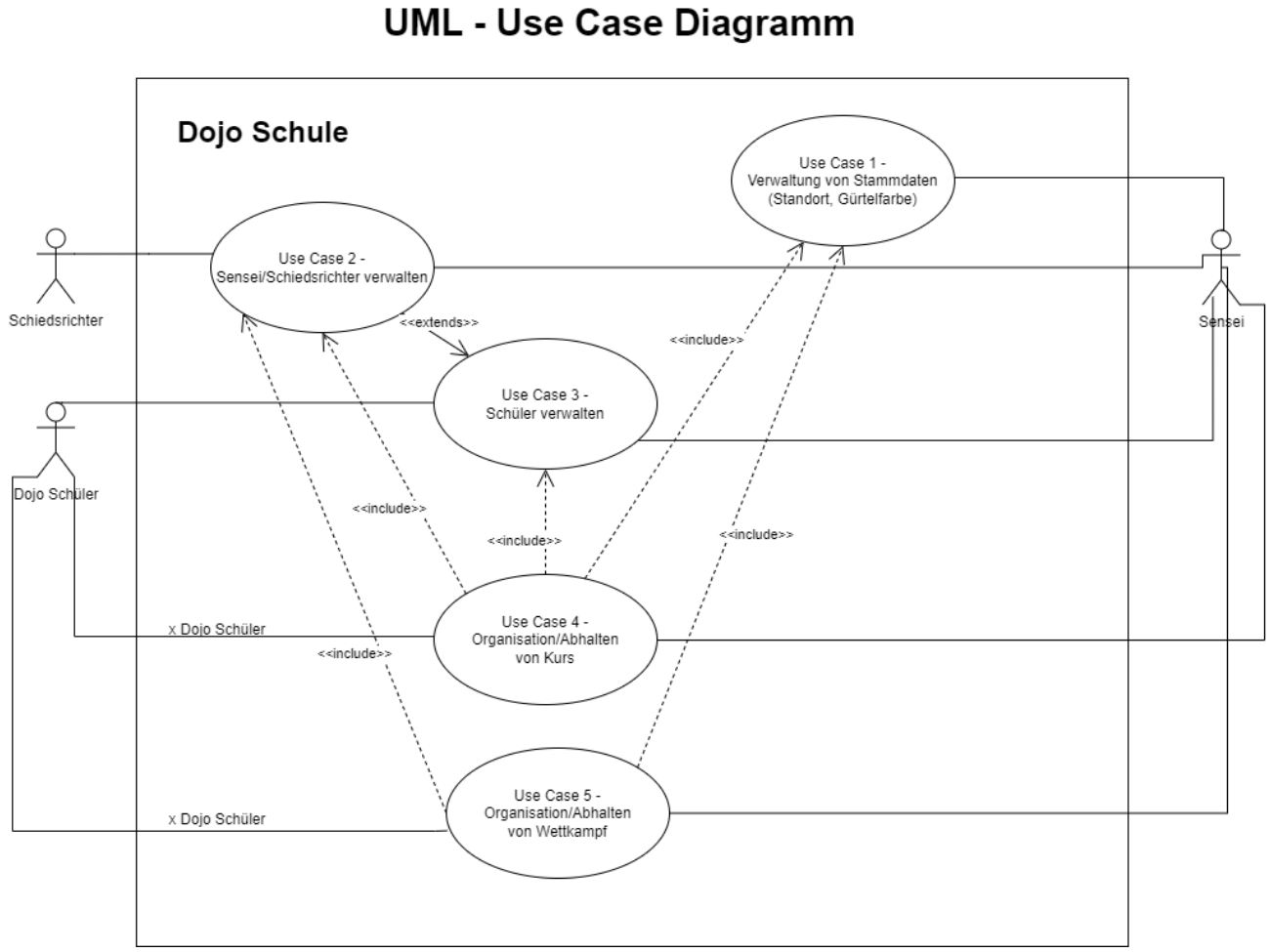
5. Identifiziert die Stakeholder und deren Bedürfnisse.

Stakeholder	**Bedürfnis**
Sensei	Hat vollen Zugriff auf das System. … möchte Senseis in der Software anlegen/bearbeiten/löschen. … möchte Schiedsrichter in der Software anlegen/bearbeiten/löschen. … möchte Schüler in der Software anlegen/bearbeiten/löschen. … möchte Kurse anlegen/bearbeiten/löschen/abhalten. … möchte Prüfungen anlegen/bearbeiten/löschen/abhalten. … möchte Wettkämpfe anlegen/bearbeiten/löschen/abhalten. … möchte DOJOs anlegen/bearbeiten/löschen. … möchte Gürtelfarben anlegen/bearbeiten/löschen.
Schüler	Hat keinen Zugriff auf das System. … möchte sich zu einem Kurs in einem DOJO anmelden. … möchte sich zu einem Wettkampf in einem DOJO anmelden. … möchte sich zu einer Prüfung in einem DOJO anmelden.
Schiedsrichter	Hat keinen Zugriff auf das System. … möchte teilhaben an einem Wettkampf

[3] https://chat.openai.com/ Prompt: Welche Nicht-funktionale Anforderungen sind, im Wartbarkeit nach SMART Prinzip
[4] Software-Metriken: die Qualität Ihrer Software messen (scnsoft.de)
[5] https://chat.openai.com/ Prompt: Was ist **Erreichbarkeit** in zuverlässigheit und ausfallsicherheit nach SMART Prinzip

Phase 2 – Design

1. Architektur der Anwendung – Darstellung mittels UML – Use Case Diagramm



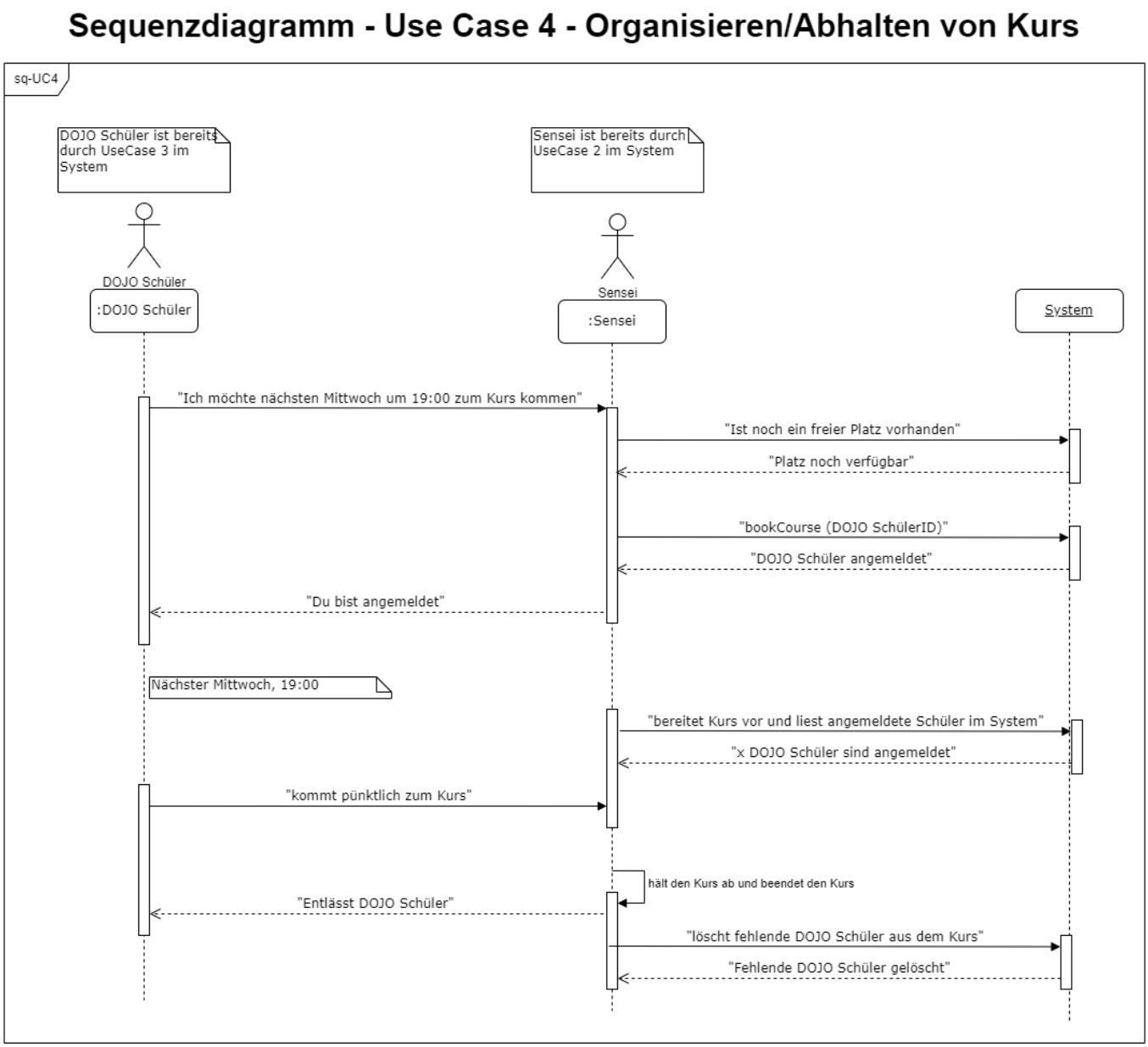
Use Case 1: Verwaltung von Stammdaten (Standort, Gürtelfarbe, usw…)
Ziel: Der Benutzer möchte Stammdaten in der Software bearbeiten, löschen und hinzufügen.
Akteure: Primärer Akteur -> Sensei; Sekundärer Akteur -> Stammdatenservice
Auslöser: Der Benutzer öffnet die Software und navigiert zum Tab Stammdatenverwaltung
Vorbedingungen: Der Benutzer muss das System erreichen und öffnen können
Nachbedingungen: Der Benutzer hat Stammdaten bearbeitet, gelöscht oder hinzugefügt
Hauptszenario: Der Benutzer öffnet die Software, navigiert zur Stammdatenverwaltung. Danach wählt er die gewünschten Stammdaten aus (Bsp.: Standort) und klickt auf den „+“ Button. Das System öffnet einen Dialog, wo die Stammdaten eingetragen werden können. Danach schließt der Benutzer diesen Dialog mit „OK“ und das System speichert die Stammdaten.
Alternativszenario: Anstelle des „+“ Buttons wählt der Benutzer einen Eintrag in der Tabelle aus und klickt auf den „-“ Button. Danach bestätigt der Benutzer den Hinweisdialog mit der Meldung
„Möchten Sie wirklich x löschen?“ mit OK. Anstelle von x sieht der User den Typ und danach den Namen des Eintrags. (Bsp.: Standort - Hartberg)
Ausnahmen: Wenn das System die Stammdaten nicht speichern kann oder diese ungültige Werte enthalten, erhält der Benutzer eine Fehlermeldung.

Use Case 2: Sensei/Schiedsrichter verwalten
Use Case 3: Schüler verwalten
Use Case 2 und Use Case 3 sind ident mit Use Case 1. Einziger Unterschied ist, dass der Benutzer nicht in die Stammdatenverwaltung wechselt, sondern in den Tab Sensei/Schiedsrichter bzw. in den Tab Schüler

Use Case 4 – Organisieren und Abhalten von Kurs:
Use Case 5 – Organisieren und Abhalten von Wettkampf:
Ziel: Der Benutzer möchte Kurs/Wettkampf in der Software bearbeiten, löschen und hinzufügen.
Akteure: Primärer Akteur -> Sensei

Auslöser: Der Benutzer öffnet die Software und navigiert zum Tab Kurs oder zum Tab Wettkampf
Vorbedingungen: Der Benutzer muss das System erreichen und öffnen können. Der Benutzer hat notwendige Stammdaten angelegt. Z.B.: Standort, Gürtelfarbe, …
Nachbedingungen: Der Benutzer hat Kurs/Wettkampf bearbeitet, gelöscht oder hinzugefügt
Hauptszenario: Der Benutzer öffnet die Software, navigiert zum Tab Kurs oder zum Tab Wettkampf.
Der Benutzer klickt auf den „+“ Button. Das System öffnet einen Dialog, wo der Kurs/der Wettkampf eingetragen werden kann. Der Benutzer wählt das Datum, die Uhrzeit, den Standort den Trainer bzw. Schiedsrichter aus und wählt die Anwesenden Schüler aus. Danach schließt der Benutzer diesen Dialog mit „OK“ und das System speichert den Kurs/den Wettkampf.
Alternativszenario: Anstelle des „+“ Buttons wählt der Benutzer einen Eintrag in der Tabelle aus und klickt auf den „-“ Button. Danach bestätigt der Benutzer den Hinweisdialog mit der Meldung
„Möchten Sie wirklich x löschen?“ mit OK. Anstelle von x sieht der User den Typ und danach den Namen des Eintrags. (Bsp.: Kurs – Hartberg – 23.06.2024 14:00)
Ausnahmen: Wenn das System den Kurs/Wettkampf nicht speichern kann oder diese ungültige Werte enthalten, erhält der Benutzer eine Fehlermeldung.

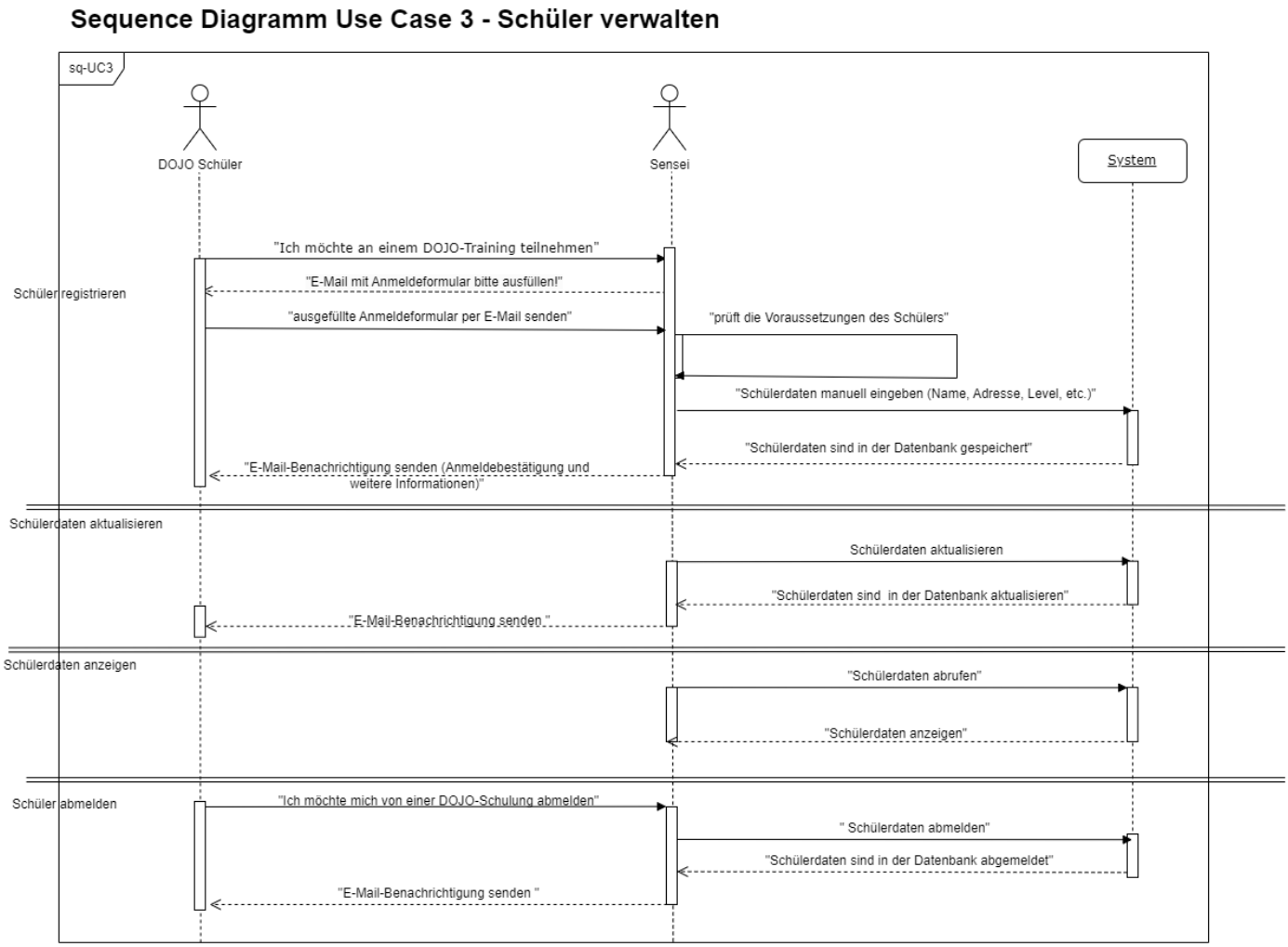
2. Architektur der Anwendung – Darstellung mittels UML – Sequenzdiagramm



Mit dieser UML – Sequenzdiagrammdarstellung des Use Case 4, wird der genaue zeitliche Ablauf zum Organisieren eines Kurses visualisiert.

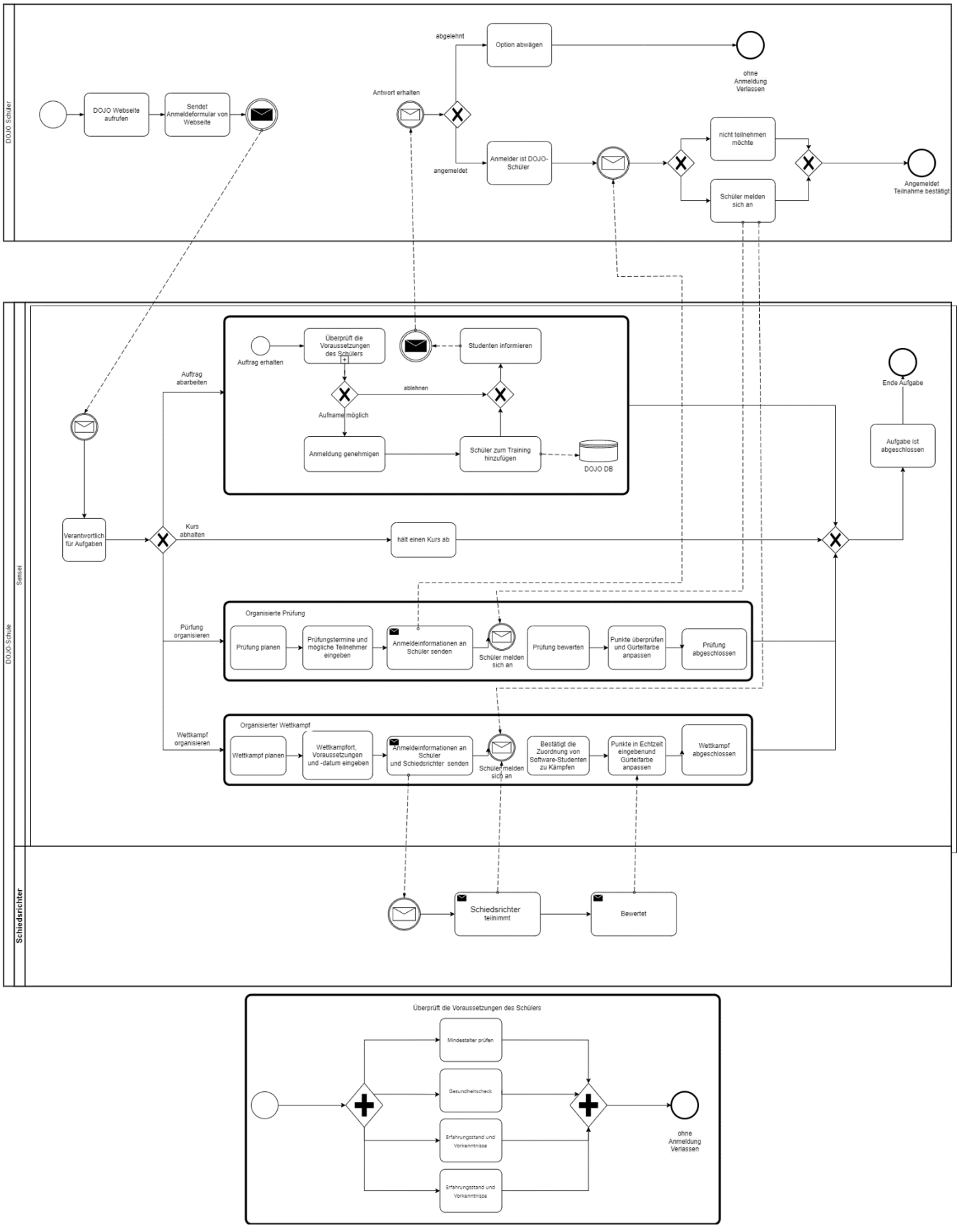
Gefundene Entitäten:
EmailServer (MailServerAddresse: String, MailServerPort: integer, MailServerUser: String, MailServerPasswort: String)
Land (LandID: integer, Bezeichnung: String, gelöscht: boolen)
Stadt (stadtID: integer, Bezeichnung: String, Postleitzahl: String, gelöscht: boolen)
Adresse (AdressID: integer, LandID: integer, StadtID: integer, Straße: String, Hausnummer: String, gelöscht: boolean)
DOJO Standort (AdressID: integer, Bezeichnung: String, gelöscht: boolean)
Level (LevelID: integer, Bezeichnung: String, Farbe: integer (RGB code), gelöscht: boolean)
Person (PersonID: integer, Vorname: String, Nachname: String, AdressID: integer, gelöscht: boolean, eMail: String)
DOJO-Person(LevelID: integer)
Sensei extends DOJO-Person (loginErlaubt: boolean)
Schüler extends DOJO-Person(StartLevelID: integer, Eintrittsdatum: date)
Schiedsrichter extends Person (extern: boolean)
Kurs (KursID: integer, Kursdatum: date, SenseiPersonenID: integer, DOJO-StandortID: integer, abgehalten: boolean)
Kursteilnehmer (KursteilnehmerID: integer, SchülerPersonenID: integer)
Prüfung (SenseiPersonenID: integer, SchülerPersonenID: integer, bestanden: boolean, LevelID: integer, NewLevelID: integer, Prüfungsdatum: date, DOJOStandortID: integer)
Wettkampf (SchiedsrichterPersonenID: integer, HerausforderPersonenID: integer, GegnerPersonenID: WettkampfDatum: date, DOJOStandortID: integer, HerausfordererPunkte: integer, GegnerPunkte: integer

3. Architektur der Anwendung – Darstellung mittels UML – Sequenzdiagramm



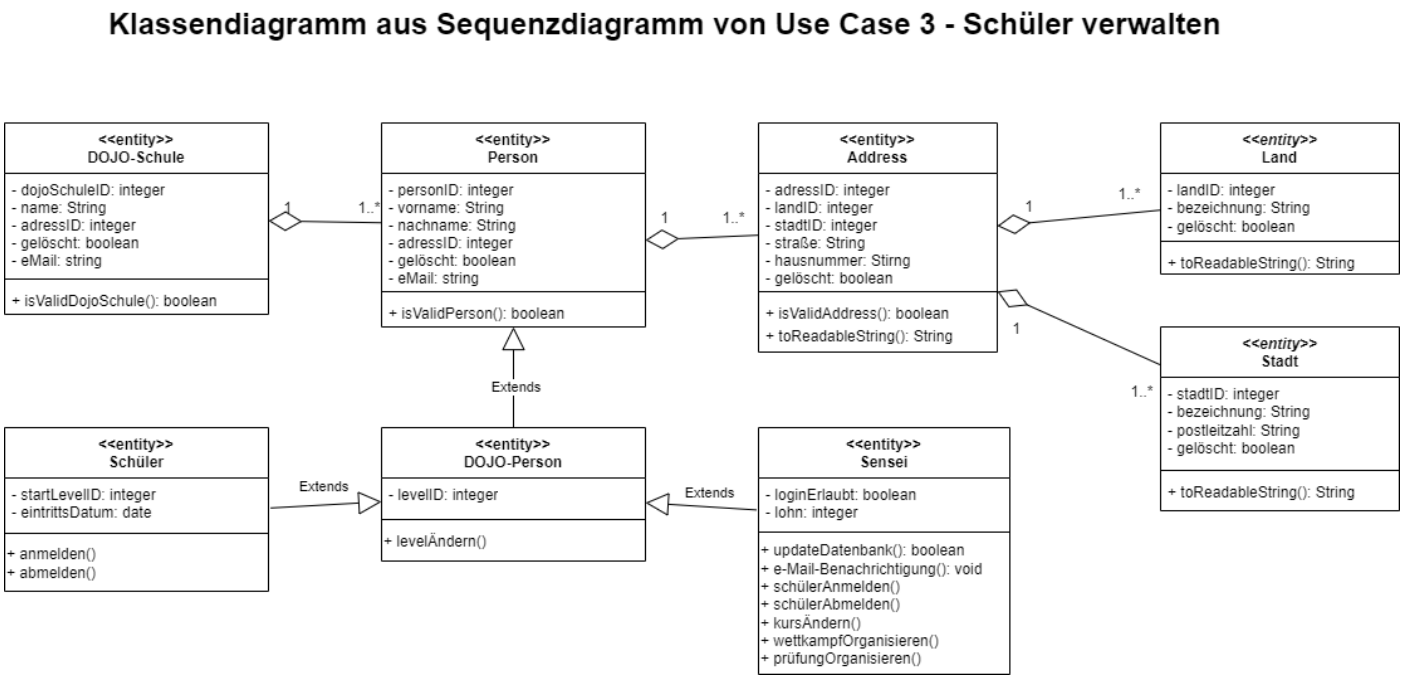
Mit dieser UML – Sequenzdiagrammdarstellung des Use Case 3, wird der genaue zeitliche Ablauf zum Verwalten von Schülern visualisiert.

4. Visualisierung der Geschäftsprozesse mittels BPMN



In diesem BPMN (Business Process Model and Notation) Abbild, sind alle relevanten Geschäftsprozesse visualisiert.

5. Architektur der Anwendung – Darstellung mittels UML – Klassendiagramm



Dieses Bild zeigt ein UML – Klassendiagramm, welches aus dem Sequenzdiagramm von Use Case 3 (Schüler verwalten) abgeleitet wurde.

Phase 3 – Entwicklung

Code Standard:
Als Einheitlicher Coding Standard für Java werden die „Java Code Conventions“ [6] von Oracle verwendet. Ebenfalls werden einheitliche Codeformatierungsregeln verwendet, welche automatisch per IntelliJ Setting Repository auf alle Entwickler PCs ausgerollt werden. Der Code muss außerdem von JUnit-Tests abgedeckt sein. Hierfür gilt als Ziel, eine CodeCoverage von 80% zu erreichen und behalten.

Modularer Code:
Es wird eine komponentenorientierte Architektur forciert. Unabhängig funktionierende Module werden geschaffen, damit die Komplexität der Software reduziert wird und die Wiederverwendbarkeit erhöht wird.
z.B.: Datenbank Basismodul, dass den generischen Zugriff bzw. Speicherung der Daten ermöglicht.

Code-Versionierung:
Als Versionskontrollsystem wird Git verwendet. Für jeden Entwicklungstask muss es einen eigenständigen Commit geben, der von einem anderen Entwickler durch einen CodeReview Prozess abgesegnet wird.
Erst dann, darf dieser Commit in den Hauptbranch gepusht werden.
Es gibt für die Konfiguration, den eigentlichen Quellcode und auch für die Build Pipeline ein eigenes Git Repo. Rollbacks können dadurch schnell und effizient durchgeführt werden. Außerdem ist dadurch eine lückenlose Änderungshistory möglich.

Feature Flags
Feature Flags werden verwendet, um neu Funktionen bereits in der Entwicklungsphase freischalten zu können. Dadurch bleiben Entwicklungen so lange inaktiv bis diese durch die QA bzw. die Tester freigegeben wurde.

[6] https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Logging
Als Logging Tool wird, log4j2 verwendet, welches auch zur Laufzeit umkonfiguriert werden kann.
Folgende vier Logging Levels sollten verwendet werden:

Logging Level	Beschreibung
ERROR	Für Exceptions, Fehler, Ausnahmezustände
DEBUG	Zur reinen Diagnose, um den Quellcode bzw. den Code besser zu verstehen.
INFO	Für generelle Information. Konfigs geladen, Service gestartet, ..
WARN	Zur Verhinderung von Systemausfällen bzw. ungewolltes Verhalten Arbeitsspeicher gering, Benutzeranmeldung fehlgeschlagen

Welche Programmiersprachen und Tools werden wir verwenden?

Backend Technologie: Java
IntelliJ IDEA mit Java wird verwendet, um die Backend-Logik und Datenbankanbindung zu implementieren.
Frontend Technologie: Java
IntelliJ IDEA mit Java wird verwendet, um die Frontend-Logik zu implementieren. Das System wird allerdings so implementiert, dass es leicht auch als Webanwendung verwendet werden kann.
Datenbanksystem: PostgreSQL
Für die Datenbankmanipulation und -abfrage, insbesondere wenn relationale Datenbanken verwendet werden.
Build-Werkzeuge: Maven
Maven wird als Build-Werkzeug verwendet. Die einfache Integration in TeamCity erleichtert den kompletten Build Prozess.

Phase 4 – Testen

Testplan-Entwicklung: Mit TestRail können wir detaillierte Testpläne erstellen, die alle relevanten Testfälle für die Funktionalität und die Performance der Software umfassen. TestRail integriert sich nahtlos mit unserem bestehenden Tool-Set, einschließlich QF-Test, JUnit und TeamCity, und bietet eine zentrale Plattform für das Testmanagement.

Regressionstests:[7]

Verwendung von QF-Test für die automatisierte Durchführung von Regressionstests.
QF-Test: werden wir für unsere Java-basierte Software verwenden, da wir uns zu 100% auf die effiziente Automatisierung von Testläufen verlassen können. QF-Test ermöglicht die Integration von Skripten und ist sehr stabil sowie skalierbar. Es kann Tests sowohl für kleine als auch für komplexe Anwendungen durchführen, was für das Testen einer umfassenden DOJO-Verwaltungssoftware wichtig ist. QF-Test kann für eine Vielzahl von Testszenarien eingesetzt werden, einschließlich End-to-End-Tests von komplexen, applikationsübergreifenden Prozessen.

TeamCity[8] spielt eine wichtige Rolle in unserem Entwicklungsprozess, indem es automatisch alle UnitTests nach jedem Push ausführt und ein neues Build erstellt. Dabei berücksichtigt es unsere Anforderungen bezüglich Code-Qualität und Code-Abdeckung, wie sie in unserem Entwicklungsstandard definiert sind.
JUnit: Zur Durchführung von Unit-Tests gemäß den Code-Qualitätsstandards und zur Erreichung der Code-Coverage-Ziele.

[7] Funktionales Testen mit QF-Test - QF-Test (qfs.de)
[8] What is TeamCity and How it works? An Overview and Its Use Cases - DevOpsSchool.com

User Acceptance Tests:

Cucumber: Tests in natürlicher Sprache zu schreiben. Dadurch können nicht-technische Teammitglieder, wie z.B. Produktmanager oder Kunden, Testfälle verstehen und validieren.

Phase 5 – Deployment

Übersicht

Bei der gewählten Lösung handelt es sich um eine On-Premises Lösung. Der Kunde stellt uns einen Server zur Verfügung auf welchem das Postgre Datenbank System installiert wird. Für das Backup der Datenbank ist der Kunde selbst verantwortlich. Ein Backup Task, der ein Backup der Datenbank erstellt und auf einem anderen Speicherort (nicht auf dem erstellten Server) ablegt, wird mit dem Kunden bei der Installation eingerichtet.
Im Falle eines Server Defektes, kann mit diesem Backup, das System auf einen anderen Server wiederhergestellt werden. Weiters wird das Backend auf diesem Server als Dienst installiert, damit dies auch bei einem Serverneustart automatisch zur Verfügung steht.
Die Frontend Lösung ist ein Thin-Client auf Java Basis.

Deployment

Für das Deployment wird ein bereits existierender TeamCity Build Server[9] verwendet. Nach jedem Push eines Commits werden danach automatisch alle UnitTests ausgeführt und ein neues Build erstellt. Das Update dieser Builds wird mit dem Kunden abgestimmt und nicht vollautomatisiert. Es wird zuerst das Testsystem mit der neuen Version aktualisiert, wodurch der Kunde die neue Funktionalität testen kann.
Zum Live-System wird es auch ein Testsystem geben, welches als eigene Applikation gestartet wird. Live und Testsystem sind voneinander unabhängig und die Datenbank des Testsystems ist eine Kopie der Livedatenbank. Das Testsystem dient als explorative Umgebung in der neue Funktionen getestet, Benutzerschulungen durchgeführt werden oder spezielle Abläufe getestet werden können.
Alle Deployment Konfigurationen müssen in einem eigenen Git Repository gespeichert werden. Dadurch ist eine lückenlose Nachverfolgung der CI/CD Prozesse möglich und jederzeit wiederherstellbar.
Bevor mit der eigentlichen Entwicklung des Quellcodes begonnen wird, muss ein Walking Skelton des Build and Deploy Prozess vollständig integriert sein. Dadurch wird sichergestellt, dass der CI/CD-Prozess bereits von Beginn an funktioniert.

Monitoring

Kommt es beim Build zu Fehlern, zum Beispiel aufgrund von fehlgeschlagenen Unit Tests, dann muss eine schnelle Benachrichtigung des Teams erfolgen. Dies könnte zum Beispiel per Mail oder Teams Chat Nachricht passieren.

Docker-Integration

Um die Deployment-Prozesse zu optimieren und die Konsistenz der Entwicklungs- und Produktionsumgebungen sicherzustellen, wird Docker für die Containerisierung der Frontend Anwendung verwendet. Ein Dockerfile wird für das Frontend erstellt, die alle notwendigen Abhängigkeiten und Konfigurationen enthalten.
Die Docker-Builds werden in den CI/CD-Prozess mit TeamCity integriert, um sicherzustellen, dass nach jedem Code-Push automatisch neue Docker-Images gebaut und getestet werden.
Auf den Thin-Clients muss Docker installiert werden und auch der Docker Container bereitgestellt werden.
Durch das Starten eines Scripts, wird danach der Docker Container inkl. der Java Anwendung gestartet.

[9] https://www.jetbrains.com/de-de/teamcity/

Blue/Green Deployment

Anstelle der Canary Deployment-Strategie wird die Blue/Green Deployment-Strategie gewählt, um eine sichere und kontrollierte Einführung neuer Softwareversionen zu gewährleisten. Diese Methode passt besser zu unserem Szenario mit einem zentralen Datenbanksystem. Während eine Umgebung (z.B. Blue) live ist, dient die andere (Green) als Testumgebung. Updates werden zuerst in der Green-Umgebung durchgeführt. Nach erfolgreicher Verifizierung wird der Traffic auf die Green-Umgebung umgeschaltet, und die vorherige Blue-Umgebung wird zur neuen Testumgebung. Dies minimiert das Risiko und ermöglicht schnelle Rollbacks.

Strategie für Datenabfragen und Caching:10

Um die Anzahl der Datenbankabfragen zu minimieren und die Leistung zu verbessern, werden relevante Daten beim Start der Anwendung oder bei der Anmeldung geladen und im Speicher gehalten. Dies kann durch einen DataLoader-Service erfolgen. Der DataLoader greift auf das Dao (Data Access Object) zu, um alle benötigten Daten aus der Datenbank zu laden und im Speicher zu halten. Um sicherzustellen, dass der Cache immer aktuelle Daten enthält, wird dieser regelmäßig aktualisiert (z.B. alle 10 Minuten). Der Cache wird ebenfalls aktualisiert, wenn spezifische Ereignisse eintreten (z.B. wenn eine Veranstaltung organisiert oder ein neuer Benutzer angelegt wird).

Phase 6 – Wartung[11]

Wartungsplan für die Software[viii]:

1. **Regelmäßige Updates und Patches**: Einsatz von Git für die Verfolgung von Code-Änderungen und
2. **Leistungsüberwachung und Fehlerbehebung**: Verwendung von Pinpoint APM zur kontinuierlichen Überwachung und QF-Test für das Testen von Softwarefunktionen.
3. **Datenbankwartung und -optimierung**: PostgreSQL für Wartungsarbeiten wie Indexoptimierung und Backups.
4. **Code-Review und Refactoring**: Git für das Code-Management und IntelliJ IDEA für statische Code-Analyse und Refactoring.
5. **Benutzerfeedback und Anforderungsanalyse**: TestRail für die Organisation von Benutzerfeedback und Anforderungsmanagement.

Aktualisierung und Pflege der Software:

1. **Versionskontrolle und Code-Management**: Git für effektive Versionskontrolle und Management des Quellcodes.
2. **Automatisierte Builds und Tests**: TeamCity für automatisierte Build- und Testprozesse, um Änderungen sicher zu integrieren.
3. **CI/CD-Pipelines**: Einrichtung von CI/CD-Pipelines mit TeamCity für automatisierte Bereitstellung von Updates.
4. **Überwachung und Fehlerbehebung**:
   PinPoint APM (Application Performance Managment) zur Überwachung und Analyse während der Laufzeit des Systems. Mit diesem Tool muss keine einzige CodeÄnderung durchgeführt werden.
   Außerdem hat es nur eine durchschnittliche Senkung von ca. 3% der Performance des Systems.
   Durch dieses Tool erhält man auch einen vollständigen Überblick über die verteilten Systeme.
   Jede Transaktion kann vom Start bis zum Ende genau verfolgt werden.
5. **Datenbanksicherung und Wartung**: PostgreSQL[12] für automatisierte Backups und Wartungsprozesse der Datenbank.

---

[10] Kurs von **Neelam Dwivedi** Assistant Teaching Professor at Heinz College - Software Design: From Requirements to Release - POC implementation

[11] https://chat.openai.com/ Prompt: Schreib meinen Text so kurz wie möglich ohne Inhalt Verlust

[12] PostgreSQL-Sicherung. Wie sichert man eine PostgreSQL-Datenbank? (baculasystems.com)
Was ist PostgreSQL? | IBM

Literaturverzeichnis

Die SMART Methode verstehen und anwenden mit Beispiel (scribbr.de)

Funktionale Anforderungen: Beispiele und Vorlagen – Visure-Lösungen (visuresolutions.com)

QF-Test Kosten, Erfahrungen & Bewertungen - Capterra Deutschland 2024

https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

https://www.jetbrains.com/de-de/teamcity/

How to start DevOps | Atlassian

Docker for Beginners: Everything You Need to Know (howtogeek.com)

Abbildungsverzeichnis

[i] Durchführung von Stakeholder-Interviews mit Sensei, Administratoren, um deren Bedürfnisse zu verstehen.:

„DOJO-SCHULE - Es gibt mehrere DOJO-Schulen. Nur ein DOJO pro Stadt. Internationale Adresse für jeden DOJO-Kurs. Die Adresse soll überall abgebildet werden. Schüler können an mehreren Schulen/Unterrichtsstunden teilnehmen. Die Namen der Schüler. Es gibt einen Trainer pro Standort, jedoch kann ein Trainer mehrere DOJOs leiten. Jeder Trainer und jeder Schüler haben einen Gürtel. Die Level sind nach Gürtelfarbe unterteilt. Nach einer Prüfung kann sich die Gürtelfarbe (das Level) ändern. Es gibt mehrere Schüler pro Unterrichtsstunde, jedoch nur einen Trainer. Die Schüler trainieren, um an einem Wettkampf teilzunehmen. Für jeden Wettkampf gibt es einen Ort und ein Datum. Bei jedem Kampf ist ein Schiedsrichter anwesend. Es treten immer zwei Teilnehmer gegeneinander an: ein Herausforderer und ein Gegner. Die Bewertung erfolgt anhand von H- und A-Punkten.“

[ii] Unser Ziel ist es, eine Software zu entwickeln, die auf den Laptops der Senseis läuft und eine gemeinsame Datenbank mit allen von ihnen eingegebenen Informationen verwaltet. Die Schüler haben keinen direkten Online-Zugriff auf die Software, sondern erhalten Benachrichtigungen per E-Mail.

[iii] Die Anmeldung erfolgt über ein Formular, das von den Schülern als Antwort auf die E-Mail des Senseis gesendet wird. Dieses Formular umfasst Anmeldungen für das Training, Prüfungen oder Wettkämpfe. Der Sensei trägt die Informationen des Formulars manuell in die Software ein und meldet die Schüler entsprechend ihres Leistungsstands für das entsprechende Training an. Zusätzlich berücksichtigt das System die Level der Schüler und bietet nur Trainings an, die zu ihrem Leistungsstand passen.

[iv] Die Senseis geben Standorte, Termine und weitere relevante Informationen ein und organisieren die Aktivitäten durch einen simplen Klick auf die Schaltfläche "Organisieren". Das System sendet automatisch E-Mails an die Schüler mit Informationen und einem Anmeldeformular für Prüfungen oder Wettkämpfe. Sobald sich ein Schüler per Antwort-E-Mail anmeldet, wird er vom Sensei in der Software registriert und in die entsprechende Liste eingetragen.

Nach jeder Prüfung bewertet der Sensei die Schüler und trägt die Punkte nur in die Software ein. Die Software überprüft, ob genügend Punkte für die Änderung der Gürtelfarbe vorhanden sind, und bietet dem Schüler gegebenenfalls andere optionale Trainings an.

[v] Wir planen, eine ähnliche Funktion wie bei Prüfungen einzurichten, um Wettkämpfe zu organisieren. Der Sensei entscheidet über Ort und Zeitpunkt eines Wettkampfs, sendet für jeden Schüler Informationen über den entsprechenden Termin und bietet die Möglichkeit zur Anmeldung an. Die Software zeigt an, wer sich bereits angemeldet hat. Wenn der Sensei die Anmeldung abschließt, sortiert die Software die Schüler zufällig in Kämpfe ein. Die Schiedsrichter geben die Punkte in das Programm in Echtzeit ein, das dann anzeigt, wer gewonnen hat und welche nächsten beiden Kämpfer antreten werden.

[vi] Es gibt eine regelmäßige Bewertung des Trainings in der Schule, die fünf Fragen mit jeweils fünf Punkten Bewertungsmöglichkeit umfasst. Diese Bewertung kann per E-Mail geschickt werden oder direkt in die Software eingetragen werden. Sie dient dem Sensei und der Verwaltung als Feedback, ob die Schüler mit dem Training zufrieden sind oder nicht.

[vii] Die Software wird so gestaltet, dass die Senseis nach ihren Präferenzen bezüglich des Menüaufbaus befragt werden. Wir legen Wert darauf, dass die Software logisch und benutzerzentriert ist, um sie einfach bedienbar zu machen. Zudem wird auf Barrierefreiheit geachtet und die Farben so gestaltet, dass die Benutzer leicht navigieren können. Ein direkter Kontakt wird angeboten, falls die Senseis Änderungswünsche haben.

[viii] https://chat.openai.com/ Prompt:
Ich werde mein software Wartungsplan schreiben und software soll nach der Bereitstellung aktualisiert und gepflegt sein. Welche punkte sind zu erwähnen? Mit meinen tools :New Relic, Git, TeamCity, QF-Test, JUnit PostgreSQL