Doc CapWatch

Release init-294-gc6ace3a

Rafael Fuhrer **Jonas Hauser** **Christoph Scheiwiller**
Pascal Schlumpf **Pascal Schneider**

25.05.2021

Inhaltsverzeichnis

Kürzel Capwatch

Beschreibung Dokumentation für das CapWatch Projekt

Autoren Rafael Fuhrer, Jonas Hauser, Christoph Scheiwiller, Pascal Schlumpf, Pascal Schneider (E-Mail an alle)

Version init-294-gc6ace3a

KAPITEL 1

Dokumentation

1.1 Projektantrag

Projektkürzel	Datum
CAPWATCH	16.12.2020

1.1.1 Team

- Pascal Schlumpf pascal.schlumpf@ost.ch
- Christoph Scheiwiller christoph.scheiwiller@ost.ch
- Jonas Hauser jonas.hauser@ost.ch
- Rafael Fuhrer rafael.fuhrer@ost.ch
- Pascal Schneider pascal.schneider@ost.ch

1.1.2 Beratungs- und Review-Zeitslots

*Beratungs- und Review-Zeitslots angeben, an denen **alle** Teammitglieder anwesend sein können:*

```
 X  = Slot ist dem Team möglich
(X) = Slot ist für das Team nicht optimal, wäre aber möglich
 O  = Treffen online möglich
 P  = Treffen physisch (Campus OST-RJ) möglich
 OP = Treffen online sowie physisch (Campus OST-RJ) möglich
    = Slot ist nicht möglich
```

Zeitslot	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
08:00-09:00					
09:00-10:00					
10:00-11:00					
11:00-12:00					
12:00-13:00	(XO)				
13:00-14:00	XO				
14:00-15:00	XO				
15:00-16:00	XO				
16:00-17:00	XO				
17:00-18:00					
18:00-19:00					

1.1.3 Motivation

Real-time Daten zur Anzahl Personen in einem Raum oder Gebäude, welche eine Personenbeschränkung aufgrund einer Pandemie oder einem sonstigen Grund haben, wie z. B. Geschäfte, Saunen, Bäder, Sportanlagen etc. um das persönliche Einkaufs- und Konsumverhalten besser planen bzw. anpassen zu können. Es entstehen dabei Vorteile für die Anbieter und die Konsumenten. Die Anbieter können Dienstleistungsengpässe vermeiden, welche zu einem Besucherverlust führen könnten. Die Konsumenten profitieren von kurzen Wartezeiten und somit auch besseren Verkaufsdienstleistung wie z. B. Beratungen.

1.1.4 Projektidee

Eine Webapplikation, in der man seine Lieblingsgeschäfte, Saunas, etc. abonnieren kann. Die Daten werden bei einem Abonnement dann in Echtzeit aktualisiert. Die Daten werden von den Dienstleistern über eine API an das Backend geliefert und in einer Datenbank abgelegt. Das Backend besteht aus insgesamt zwei zentralen APIs. Die zweite Schnittstelle ist für die Datenauslieferung an das Frontend bzw. die Webseite, welche schlussendlich der Benutzer für die Abonnements verwendet. Das Frontend ist ein reiner, einseitiger Empfänger ohne Authentifizierung (Speicherung in einem lokalen Speicher beim Benutzer) in einer ersten Version.

Mögliche weitere Features:

- Automatische selektion von Abonnements (z. B. basierend auf dem Standort)
- Authentifizierung des Benutzers
- Benutzerprofile serverseitig speichern
- Vertikale Skalierung (mehr Workload verarbeiten können)
- Personalisierte Notifikationen an Benutzer (z. B. per E-Mail)
- Prognosen und Vorhersagen
- Möglichkeit zum Reservieren

Projektidee besprochen mit: Thomas Kälin

1.1.5 Realisierung

Frontend: React oder Angular mit TypeScript (evtl. PWA)

Backend: Java oder Node.JS Framework für API-Entwicklung

Datenbank: noch nicht festgelegt

1.2 Projektplan

1.2.1 Einführung

Zweck

Dieses Dokument gibt eine Übersicht über die Art und Weise wie wir unser Projekt CapWatch durchführen werden.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Vorgehen zur Überarbeitung der Planung

Wenn wir die Planung nachträglich überarbeiten, wird diese Version von allen Zusammen in einem Meeting geprüft und abgenommen.

Referenzen

GitLab Dokumentation GitLab Time Tracker (gtt) YouTrack

1.2.2 Projektübersicht

Eine Webapplikation, in der man seine Lieblingsgeschäfte, Saunas, etc. abonnieren kann. Die Daten werden bei einem Abonnement dann in Echtzeit aktualisiert. Das Backend besteht aus mehreren API Endpoints. Die Daten werden von den Dienstleistern über eine API an das Backend geliefert und in einer Datenbank abgelegt. Über eine weitere Schnittstelle können die Daten vom Frontend bzw. der Webseite abgefragt werden. Im Frontend kann sich der Benutzer Favoriten speichern. Das Frontend ist in der ersten Version ein reiner einseitiger Empfänger ohne Authentifizierung.

Zweck und Ziel

Real-time Daten zur Anzahl Personen in einem Raum oder Gebäude, welche eine Personenbeschränkung aufgrund einer Pandemie oder einem sonstigen Grund haben, wie z.B. Geschäfte, Saunen, Bäder, Sportanlagen etc. um das persönliche Einkaufs- und Konsumverhalten besser planen bzw. anpassen zu können. Es entstehen dabei Vorteile für die Anbieter und die Konsumenten. Die Anbieter können Dienstleistungsengpässe vermeiden, welche zu einem Besucherverlust führen könnten. Die Konsumenten profitieren von kurzen Wartezeiten und somit auch besseren Verkaufsdienstleistung wie z.B. Beratungen.

Die Ziele des Teams sind eine allgemein gut funktionierende, erweiterbare, performante und benutzerfreundliche Plattform für die Endkunden, wie auch für die Datenlieferanten. Die Weblösung soll einfach zu bedienen sein und das

Leben vereinfachen, vor allem in Zeiten einer Pandemie wo Einschränkungen vom Konsumverhalten der Bevölkerung gefordert wird.

Lieferumfang

- Offizielle Dokumentation des Projektes in Form des Templates der OST
- Frontend für den Endbenutzer
- Backend-API bestehend aus zwei logisch getrennten APIs

Annahmen und Einschränkungen

Das Projekt unterliegt keinen speziellen Annahmen oder Einschränkungen. Die Dokumentation wird im Markdown-Format erstellt und hat somit in der Erstellung einige Einschränkungen.

1.2.3 Projektorganisation

Das Projekt wird durch fünf Software-Engineering Studenten der Ostschweizer Fachhochschule am Campus Rapperswil-Jona durchgeführt. Alle Projektbeteiligten studieren im berufsbegleitenden Studienmodell und arbeiten nebenbei bis zu 70 Prozent für ihre Arbeitgeber. Einige Teammitglieder haben Spezialgebiete, nach welchen auch die Verantwortlichkeiten verteilt werden. Betreut wird das Projekt durch den Advisor Herrn Thomas Kälin.

Im Team verwenden wir Microsoft Teams für die Meetings und die Dateiablage. Microsoft OneNote wird für diverse Notizen, Meeting-Protokolle und weiteres genutzt, was nicht direkt in die offizielle Dokumentation muss. Zusätzlich verwenden wir alle, für unser Projekt sinnvolle, Funktionen von GitLab. Die Nutzung von folgenden Funktionen ist geplant:

- Repositories in Subgruppen
- Merge Requests
- GitLab CI/CD
- Packages & Registries
- Analytics
- Integration von diversen Tools (SonarQube und Renovate)

Die Nutzung der folgenden GitLab Funktionen wurden wegen diverser Probleme (siehe Abschnitt Arbeitspakete) verworfen:

- Issue Tracking mit Kanban Board
- Script für den Time Tracking Report (gtt)

Als Ersatz zu den oben erwähnten, gestrichenen GitLab Funktionen verwenden wir YouTrack von JetBrains mit den folgenden Funktionen:

- Issue Tracking mit Kanban Board
- Integriertes Time Tracking

Organisationsstruktur

Name	Verantwortlichkeiten
Jonas Hauser	Projektleitung, Kommunikation, Arbeitsorganisation, Qualitätssicherung und Meeting-Moderation
Rafael Fuhrer	Alles rund um Systemumgebungen wie z. B. CI/CD, GitLab und Deployment-Server
Pascal Schneider	Lead für Software-Architektur und -Design und API-Design, Issue- und Time-Tracking
Christoph Scheiwiller	Allgemeine Codequalität (Guidelines und Testing)
Pascal Schlumpf	Daten- bzw. Datenbank-Spezialist

Alle Beteiligten sind gleichberechtigte Teammitglieder und wir verfolgen eine komplett flache Hierarchie untereinander. Der Projektleiter ist in keinem Fall ein Mitarbeiter mit höheren Befugnissen oder höherer Entscheidungsmacht.

Da wir Scrum als Projektmanagementmethode verwenden, haben wir ausserdem die folgenden Scrum Rollen besetzt.

Name	Scrum Rolle
Jonas Hauser	Scrum Master
Pascal Schlumpf	Product Owner

Externe Schnittstellen

Wir werden unterstützt und betreut durch den Advisor Herrn Thomas Kälin. In einzelnen Fällen ziehen wir die Expertise von anderen Studenten, anderen Betreuern und Modulverantwortlichen hinzu oder fragen bestimmte Zielgruppen für manuelle Benutzertests an.

1.2.4 Management Abläufe

Kostenvoranschlag

Unser Projekt wird auf 15 Semesterwochen aufgeteilt, was dem Standard der Vorgaben für das Engineering Projekt entspricht. Dies bedeutet ein durchschnittliches Arbeitspensum von ungefähr 8 Stunden pro Person pro Woche. Je nach Arbeitsaufwand pro Woche kann dieser Wert stark variieren. Der Endwert von etwa 120 Stunden pro Person nach 15 Wochen wird aber zwingend eingehalten.

Zeitliche Planung

Die grobe Zeitplanung wurde nach den Vorgaben von RUP durchgeführt.

	Inception		Elaboration				Construction							Transition	
Studiumswoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Project Management	45		20		25		25		25		30			25	
Documentation	20		28		11		9		8		11			0	
Design	0		20		10		10		2		0			0	
Implementation	0		0		15		20		20		30			4	
Testing	0		0		5		5		5		20			10	
Deployment	10		2		7		10		10		10			5	
Configuration & Change Management	10		5		4		6		7		10			10	
Reserve	0		6		6		6		6		6			6	
	85		81		83		91		83		117			60	
Sprints			Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5			Sprint 6	
Milestones		M1		M2		M3		M4		M5			M6	M7	
Reviews			R1		R2		R3		R4		R5				R6
Studiumswoche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

M1: Project Plan	R1: Project Plan
M2: Requirements	R2: Requirements
M3: End of Elaboration / Prototype	R3: End of Elaboration / Prototype
M4: Architecture	R4: Architecture
M5: Quality	R5: Quality
M6: Beta Version	R6: Project Presentation
M7: Final Submission	

Phasen / Iterationen

In der Entwicklung wird mit Sprints gearbeitet. Es gibt in jedem Sprint einen Meilenstein, welcher abgeschlossen werden muss. Für die grobe Planung beziehungsweise für die Phasen gehen wir nach dem RUP-Modell vor.

RUP-Phasen

Inception

In der ersten einwöchigen Phase werden die Anwendungsfälle grob beschrieben, damit haben wir ein klares Ziel für die Applikation CapWatch definiert.

Elaboration

In der Elaboration wird während vier Wochen ein Architekturprototyp erstellt. Die Anwendungsfälle werden detailliert beschrieben. Projektplanung sowie Risikoanalyse werden durchgeführt.

Construction

In dieser Phase wird das Produkt CapWatch entwickelt und getestet. Während acht Wochen entsteht die Applikation auf Grundlage der vorher definierten Anwendungsfälle.

Transition

Die Applikation CapWatch ist in der ersten Version zur Auslieferung bereit. Während zwei Wochen kann die Software eingeführt und getestet werden.

Scrum

Nach dem Erreichen des ersten Meilensteins beginnen wir iterativ mit Scrum zu arbeiten. Es wurden sechs Sprints definiert. Jeder Sprint dauert zwei Wochen, mit Ausnahme des fünften Sprints, welcher drei Wochen dauert. Am Ende jedes Sprints ist der vorgegebene Meilenstein zu erreichen.

Für die Priorisierung der Tasks im Backlog verwenden wir die MoSCoW Methode. Die Aufwandsschätzungen der Tasks werden in absoluten Stunden vorgenommen.

Meilensteine

M1 Project Plan (06.03.2021)

- Projektorganisation
- Arbeitspakete definiert
- Phasen und Meilensteine
- Qualitätsmassnahmen
- Richtlinien für Code und Dokumentation
- Build Server & Continuous Integration geplant
- Infrastruktur geplant
- Zeitplanung erstellt

M2 Requirements (20.03.2021)

- Use Cases erstellt
- Funktionale Anforderungen formuliert
- Nichtfunktionale Anforderungen formuliert
- Schnittstellen beschreiben
- Wireframes erstellt
- Domainmodell Diagramme gezeichnet

M3 End of Elaboration / Prototyp (03.04.2021)

- Architekturprototyp erstellt
- Architekturprototyp getestet
- Architekturprototyp Dokumentation vorbereitet
- Anforderungen vollständig definiert
- Test- und Reviewprozess definiert
- Risikoanalyse nachgeführt
- Entwicklungsumgebung komplett eingerichtet

M4 Architecture (17.04.2021)

- Physische und logische Architektur
- Persistenz
- User Interface
- Ausbau-Szenarien berücksichtigt
- Performance-Szenarien
- Verwendete Technologien
- Architektur Entscheidungen dokumentiert
- Besondere Merkmale der Architektur

M5 Quality (01.05.2021)

- Dokumentationsreview
- Test cases geplant, dokumentiert und umgesetzt
- Verwendung von Tools zur Sicherstellung der Codequalität (Bsp. Coderichtlinien)

M6 Beta Version (22.05.2021)

- Gesamter Funktionsumfang implementiert
- Funktionale Anforderungen getestet
- Manuelle Tests durchgeführt
- Vorhandene Bugs identifiziert

M7.1 Final Submission (Project) (26.05.2021, 12:00)

- Code-Repositories in abgabefähigem Zustand
- Dokumentations-Repository in abgabefähigem Zustand
- Fehler aus Betaversion behoben

M7.2 Final Submission (Presentation) (28.05.2021, 12:00)

- Projekt- und Produktpräsentation vorbereitet
- Präsentation als PDF abgegeben

Besprechungen

Hier sind alle fixierten Termine aufgeführt. Falls nötig können auch spontane Besprechungen dazukommen, wobei nicht alle Gruppenmitglieder anwesend sein müssen. Diese spontanen Besprechungen können von jedem Gruppenmitglied einberufen werden und ersetzen bei uns das Daily Standup Meeting, da dieses bei uns keinen Sinn machen würde.

Ort: Bis auf Weiteres finden alle Besprechungen digital in Microsoft Teams statt.

Wann	Wer	Grund
Jeden Montag 16:00 Uhr	Team / Advisor bei Bedarf	Fragen zu Projektablauf oder Vorgehen, Reviews
Jeden Donnerstag Nachmittag	Alle im Team	Aktueller Stand, Offene Fragen besprechen, Weiteres Vorgehen
Jeweils zu Beginn des Sprints 10:00 Uhr	Alle im Team	Backlog Refinement & Sprint Planning
Jeweils am Ende des Sprints 16:00 Uhr	Alle im Team	Sprint Review
Jeweils am Ende des Sprints 17:00 Uhr	Alle im Team	Retrospektive
08.03.2021 16:00 Uhr	Team / Advisor	Review 1
22.03.2021 16:00 Uhr	Team / Advisor	Review 2
06.04.2021 19:00 Uhr	Team / Advisor	Review 3
19.04.2021 16:00 Uhr	Team / Advisor	Review 4
03.05.2021 16:00 Uhr	Team / Advisor	Review 5
31.05.2021 16:00 Uhr	Team / Advisor	Schlusspräsentation

Um zu verhindern, dass unsere Scrum-Events zeitlich aus dem Ruder laufen, haben wir die folgenden Timeboxings für die Events definiert.

Scrum Event	Timeboxing	Bemerkung
Sprint Planning	1 Stunde	•
Sprint Review	1 Stunde	•
Sprint Retrospektive	30 Minuten	•
Backlog Refinement	15 Minuten	Wird vom PO vorbereitet und falls notwendig im Team ergänzt

1.2.5 Risikomanagement

Risiken

Die Risikoanalyse wird im Dokument risikoanalyse.md detailliert beschreiben.

Umgang mit Risiken

Aufgrund der Risikoanalyse wurde ein durchschnittliches Schadenspotenzial von vier Stunden berechnet, deshalb werden in jedem Sprint vier Stunden als Reserve eingerechnet um sicherstellen zu können, dass der Projektplan eingehalten werden kann.

Wir haben neu das Risiko Ri10 Pipeline aufgenommen. Diese Komponente hat sich als sehr zentral und anfällig herausgestellt. Daher wollen wir einen besonderen Fokus darauf legen.

1.2.6 Arbeitspakete

Zur Planung der Arbeit, sowie dem Tracking der aufgewendeten Arbeitszeit hatten wir ursprünglich GitLab angedacht. Für die Auswertung der Zeiterfassung hätte gtt zum Einsatz kommen sollen.

Leider hat sich in der ersten Projektwoche gezeigt, dass die uns von GitLab zur Verfügung gestellten Tools nur unzureichend funktionieren. Nach einem nicht lösbaren Problem mit dem Time Tracking, haben wir in Absprache mit unserem Advisor den Umstieg auf YouTrack beschlossen.

1.2.7 Infrastruktur

Für die Verwaltung des Quellcodes und der Projektdokumentation verwenden wir das von der OST bereitgestellte GitLab, inklusive der zur Verfügung gestellten Continuous Integration Lösung. Für das Deployment verwenden wir eine einzelne virtuelle Maschine, basierend auf Ubuntu, welche uns ebenfalls von der OST zur Verfügung gestellt wird. Über die CI Lösung werden automatisch Docker Images generiert, welche dann auf der Deployment-VM ausgeführt werden können. Die Entwicklung erfolgt auf unseren persönlichen Notebooks oder Desktop-PCs. Weitere spezielle Geräte werden nicht benötigt.Zur Unterstützung verwenden wir im CI-Prozess zusätzlich das Codequalitätstool SonarQube und das Tool Renovate für das automatisierte Updaten von Dependencies.

1.2.8 Qualitätsmassnahmen

Erstens werden für den Umgang mit Git diverse Richtlinien definiert um eine professionelle und einfache Zusammenarbeit im Softwareentwicklungsprozess mit dem ganzen Team zu erreichen. Zweitens wird nach einem Vorgehen gearbeitet, bei dem grundsätzlich keine Änderungen in die Git Repositories einfliessen, welche nicht mindestens von einem weiteren Teammitglied in einem Review validiert und bestätigt wurden. Ausnahmen hiervon sind triviale Konfigurationsarbeiten und Fehlerbehebungen, die sofort eingebracht werden müssen. Drittens werden mit jedem Build und auch mit jedem Deployment der Applikationen Sicherheit- und Integrationchecks durchgeführt durch SonarQube.

Um diese Massnahmen für die Verbesserung der Qualität umsetzen zu können, verwenden wir die folgenden GitLab Features:

- Merge- bzw. Pull-Requests
- GitLab CI/CD

Jedes Product Backlog Item, bei dem es Sinn macht, wird ausserdem mit einer Definition of Done versehen, welche dem Bearbeitenden und dem Reviewer dabei hilft, die Vollständigkeit dieses Product Backlog Items zu prüfen.

Zusätzlich setzen wir die folgenden Techniken ein, um eine hohe Qualität innerhalb des Projekts zu gewährleisten:

- Spezifikation von funktionalen und nicht funktionalen Requirements
- Einsatz von Unit, Usability, System und Performance Tests
- Nutzung der Report Funktion (Dashboards) von YouTrack zur Informationsgewinnung

Um den erfolgreichen Verlauf des Projekts zu garantieren, ist der Projektleiter dafür verantwortlich, wöchentlich die Einhaltung der Qualitätsmassnahmen zu überprüfen und Abweichungen den Teammitgliedern mitzuteilen. Der Projektleiter arbeitet nach einer wöchentlichen Checkliste, damit nichts vergessen wird.Das Team arbeitet nach dem Erreichen des ersten Meilensteins nach Scrum und garantiert somit pro Sprint fertige Teile des Produkts vorweisen zu können.

Dokumentation

Die Dokumentation wird mit Markdown erstellt, basierend auf dem offiziellen Template der Ost. Der Produktionsbranch (master) wird jeweils als Basis für den aktuellen Stand der Dokumentation verwendet. Neue Bereiche oder Dokumentationsvorschläge werden über Supportbranches (feature und bugfix) erstellt, validiert und in den Produktionsbranch zusammengeführt. Jeder neue Teil der Dokumentation wird jeweils immer von mindestens einem Teammitglied überprüft. Vor jedem Reviewtermin mit dem Advisor werden nochmals erneut alle geänderten Bereiche seit dem letzten Review auf dem Produktionsbranch überprüft.

Projektmanagement

Wir wollten ursprünglich die Issue Integration von GitLab selbst verwenden. Wegen den im Punkt Arbeitspakete beschriebenen Probleme kommt seit der zweiten Projektwoche aber YouTrack zum Einsatz. Die Issues durchlaufen einen klaren Workflow der als Kanban Board angezeigt wird. Jedes Issue durchläuft folgende Schritte: open, planned, work in progress, review und closed. Die Issues werden am Anfang in einem groben Format erstellt und warten dann im Status *open* auf deren Einplanung in einen Sprint, wodurch sie in den Status *planned* wechseln. Das Arbeitspaket wird beim Start in den Status *work in progress* und vor Beendigung in den Status *review* versetzt, um ein Review und je nachdem auch ein Testing durchzuführen. Im Status *closed* ist das Issue dann komplett fertig und somit auch auf einem archivierten Stand.

Entwicklung

Der Sourcecode vom Backend und Frontend befindet sich in einer eigenen Untergruppe in unserem GitLab Projekt als jeweils eigene Repositories.

Spezielle Qualitätsmassnahmen spezifisch für den Sourcecode wurden bereits durch die allgemeinen Qualitätsmassnahmen an das ganze Projekt definiert.

Vorgehen

Wir entwickeln vollständig nach dem Prinzip der agilen Softwareentwicklung mit Scrum und entwickeln demnach in Iterationen nach und nach Teile der Applikation. Der Sourcecode wird klassisch nach Features entwickelt, ohne dabei zuerst Tests zu schreiben (kein Test Driven Development). Die Logik der Features wird zuerst implementiert und danach werden die Unit Tests dazu geschrieben. Im Frontend verzichten wir auf automatisierte Tests, weil der Projektumfang zu klein ist und der Aufwand dafür zu gross wäre. Das ganze Frontend wird manuell getestet.

Unit Testing

Automatisierte UnitTests werden ausschliesslich im Backend geschrieben. Der Grund hierfür ist, dass jegliche Business-Logik, welche auf die Daten angewandt werden muss, bei der Aufbereitung der Daten vor dem Bereitstellen durch die jeweiligen APIs ausgeführt wird. Das Frontend wird lediglich die Rückgaben aus den APIs visuell zur Verfügung stellen.

Die Tests im Backend werden technologisch mit xUnit, FluentAssertions und FakeItEasy umgesetzt.

Um sicherzustellen, dass die Testabdeckung ausreichend ist, werden Merge Requests mit weniger als 80% Code Coverage von SonarQube automatisch abgelehnt.

Code Reviews

Wie bereits im Kapitel zu den allgemeinen Qualitätsmassnahmen beschrieben, erstellen wir grundsätzlich immer Merge Requests die von mindestens einem weiteren Teammitglied überprüft werden. Es wird immer mit Supportbranches gearbeitet und nach dem Code Review erst die Zusammenführung auf einen Mainbranch gemacht.

Code Style Guidelines

Die folgenden Guidelines gelten als Ausgangspunkt, wie der Quellcode im Front- und Backend formatiert werden soll. Etwaige Abweichungen, welche sich als sinnvoll herausstellen, müssen im Team besprochen und von diesem abgenommen werden.

Programmiersprache	Grundlage	Link
.NET	C# Coding Conventions von Microsoft	https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions
HTML / CSS	Google HTML/CSS Style Guide	https://google.github.io/styleguide/htmlcssguide.html
TypeScript	Google TypeScript Style Guide	https://google.github.io/styleguide/tsguide.html
React	React + Typescript Cheatsheets	https://github.com/typescript-cheatsheets/react
JSX	React JSX	https://reactjs.org/docs/introducing-jsx.html

Abweichungen von den Grundlagen

.Net

- Wir verwenden nicht die Standard IDE Einstellungen von Microsoft Visual Studio, sondern diejenigen aus dem Einstellungs-Backup

```
Exported-2021-02-04.vssettings
```

- Wir verwenden LINQ ausschliesslich in der Methoden-Syntax

HTML / CSS

- Wir verwenden die Standard IDE Einstellungen von JetBrains Webstorm (auch die Standardformatierung)
- Bei HTML und CSS haben wir keine spezifischen Abweichungen von den Code Guidelines definiert

TypeScript

- Wir verwenden die Standard IDE Einstellungen von JetBrains Webstorm (auch die Standardformatierung)
- Bei der Aufsetzung vom Frontend-Repository wird TSLint installiert und konfiguriert

React + Typescript und JSX

- Wir verwenden die Standard IDE Einstellungen von JetBrains Webstorm (auch die Standardformatierung)
- Bei React, zusammen mit TypeScript, haben wir keine spezifischen Abweichungen von den Cheatsheets definiert
- Bei JSX haben wir keine spezifischen Abweichungen von der offiziellen React JSX Dokumentation definiert

Software-Engineering Prinzipien

Um Maintainability und Qualität der Code Basis zu gewährleisten werden gängige Software-Engineering Prinzipien stets berücksichtigt. Diese umfassen KISS, YAGNI, DRY, BDUF und S.O.L.I.D.

Testen

Unit Tests

Siehe Kapitel *Unit Testing* unter *Entwicklung*.

Systemtests

Systemtests werden manuell nach Fertigstellung jedes Sprintumfangs durchgeführt. Hierfür wird beim Erstellen der Detailspezifikationen eine Liste an benötigten Tests erstellt, welche in eine Gesamtliste an durchzuführenden Systemtests eingepflegt wird.

Performancetests

Da beim Backend über die Update API der Geschäfte theoretisch relativ viel Traffic eingehen kann, wird das Backend vor jedem Release auf seine Performance getestet. Hierfür verwenden wir Apache JMeter.

Usability Tests

Um die Benutzerfreundlichkeit unseres Projekts sicherzustellen, werden wir auf manuell durchgeführte Usability Tests setzen, da diese mit vergleichsweise wenig Aufwand sehr viel wertvolles Feedback zur Qualität und zu möglichen Verbesserungen liefern.

1.3 Risikoanalyse

1.3.1 Einführung

Zweck

In diesem Dokument werden Risiken identifiziert und beurteilt. Aufgrund dieser Beurteilung wird definiert wie mit den Risiken zu verfahren ist.

Gültigkeitsbereich

Dieses Dokument ist für das EPJ CapWatch gültig. Eine Überprüfung der Risiken ist während des Projektes mehrfach durchzuführen.

Referenzen

Vorlesung SE1 FS2021: Woche 1 - W01 - Introduction and Project Planning

1.3.2 Risikomanagement

- Projekt: CapWatch
- Erstellt am: 02.03.2021
- Autor: Christoph Scheiwiller
- Gewichteter Schaden Projekt gesamt: *20 Stunden*
- Durchschnitt Schadenspotenzial pro Sprint (gerundet): *6 Stunden**

Nr	Titel	Beschreibung	Schadenspotenzial [1-3] Stunden	Eintrittswahrscheinlichkeit [1-3]	Gewichteter Schaden** [1-9] Stunden	Vorbeugung	Verhalten beim Eintreten
Ri1	Internetausfall	Durch die Corona-pandemie müssen alle Besprechungen online durchgeführt werden. Dabei sind wir auf eine funktionierende Internetverbindung angewiesen.	1	2	2	Dokumentation relevanter Inhalte während Besprechungen	Meetingprotokoll lesen
Ri2	Ausfall Teammitglied	Ein Teammitglied fällt für eine bestimme/ unbestimmte Zeit aus.	3	1	3	keine Massnahmen	Aufgaben auf andere Teammitglieder verteilen oder Sprint Umfang kürzen.
Ri3	Ausfall der von der Ost bereitgestellten Infrastruktur	Die durch die Ost bereitgestellte Infrastruktur (z.B. Gitlab) fällt aus und es kommt zu Datenverlust.	3	1	3	Spiegeln des Repository auf Github	Wechsel auf das Github Repository
Ri4	Ungenügende Spezifikation	Anforderungen wurden nicht klar definiert und müssen während der Implementierung überarbeitet werden oder in den nächsten Sprint verschoben	0	0	0	Anforderung reviewen	Anforderungsanalyse überarbeiten

* *Durchschnitt Schadenspotenzial (aufgerundet) = Gewichteter Schaden Projekt / Anzahl Sprints*

** _Gewichteter Schaden = Schadenspotenzial _ Eintrittswahrscheinlichkeit*

1.3.3 Risikoüberwachung

Die Risiken sollen nach jedem Sprint überprüft werden.

Risikoüberprüfung durchgeführt:

- 13.03.2021 (Christoph Scheiwiller): Risiken ergänzt / Neuberechnung Schadenspotenzial
- 02.04.2021 (Pascal Schlumpf): Risiken neu beurteilt
  - Ri4 bleibt gleich da im Prototyp die MUSS-Anforderungen nicht vollständig umgesetzt worden sind.
  - Ri5 bleibt trotz ersten Erfahrungen gleich. Wir müssen unsere Erkenntnisse in den nächsten Sprint einfliessen lassen und schauen ob wir uns verbessert haben. Erst dann können wir die Eintrittswahrscheinlichkeit reduzieren.
  - Bei Ri6 reduziert sich die Eintrittswahrscheinlichkeit, da wir den Grundstein der Architektur im Prototyp schon umgesetzt haben und es sich bewährt hat.
  - Bei Ri8 reduziert sich die Eintrittswahrscheinlichkeit, da im Prototyp schon viel neues Wissen erlangt worden ist.
- 15.04.2021 (Pascal Schlumpf): Risiken neu beurteilt.
  - Bei Ri4 reduziert sich die Eintrittswahrscheinlichkeit, da wir nun fast alle MUSS Anforderungen umgesetzt haben. Eine Anforderung mussten wir auf den nächsten Sprint verschieben, da der Umfang zu gross wurde. Bis jetzt wurde noch keine Kritik an den Anforderungen ausgesprochen.
  - Bei Ri5 reduziert sich die Eintrittswahrscheinlichkeit. Wir haben auch in diesem Sprint die Tasks für Entwicklung zu tief geschätzt. In diesem Punkt müssen wir uns noch verbessern. Die Abweichungen bei der Gesamtzeit sind aber soweit im Rahmen, dass wir das Risiko als kleiner einstufen.
- 29.04.2021 (Pascal Schlumpf): Risiken neu beurteilt.
  - Bei Ri7 reduziert sich das Schadenspotential um 2 auf 1, da alle wichtigen Komponenten in einer guten Qualität umgesetzt worden sind und das Team eingespielt ist. Alle KANN Anforderungen können nun mit tieferem Druck umgesetzt werden.
  - Es wurde das Risiko Ri10 erfasst. Aufgrund der Erfahrungen der letzten Wochen hat sich dieses Risiko aufgedrängt. Es hat sich herausgestellt, dass die Pipeline sehr diffizil und gleichwohl für die Erfüllung unseres Auftrags zentral ist.
- 18.05.2021 (Pascal Schlumpf): Risiken neu beurteilt.
  - Ri4, Ri6, Ri7, Ri8 wurden auf 0h gesetzt, da die Entwicklungsarbeiten abgeschlossen sind und nur noch Dokumentation ansteht.

1.4 Anforderungsspezifikationen

1.4.1 Einführung

Zweck

Dieses Dokument gibt einen groben Überblick über das Produkt CapWatch, sowie die geplanten Anforderungen, die es erfüllen muss und solche die optional noch umgesetzt werden können.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

ISO/IEC 9126OWASP Threat Modeling

Übersicht

Im Abschnitt „Allgemeine Beschreibung“ werden allgemeine Informationen zum Produkt gegeben. Anschliessend werden im Abschnitt „Funktionale Anforderungen“ die geplanten Funktionalitäten beschrieben. Im letzten Abschnitt werden die nicht Funktionalen Anforderungen genauer umschrieben und nicht kategorisierbare Themen zusammengefasst.

1.4.2 Allgemeine Beschreibung

Produkt Perspektive

Im Zuge der Corona Pandemie ist es plötzlich an sehr vielen Orten zu Einschränkungen der erlaubten Anzahl Personen innerhalb eines definierten Bereichs gekommen. Das führte dazu, dass man einkaufen gehen wollte und die Eintrittsbegrenzungssystem auf Rot (Eintritt verweigert) war. Dadurch war man gezwungen zu warten und hat Zeit verloren. Dieses Problem könnte gemindert werden, wenn man sich Zuhause im Voraus darüber informieren kann, ob es gerade ein guter Zeitpunkt ist, einkaufen zu gehen. Da die Systeme zur Personenzählung nun fest installiert sind, ist es gut möglich, dass sie auch in Zukunft vorhanden sein werden. Hier setzt unser Produkt an, indem es die ganzen gesammelten Daten konsolidiert und auf einer einfachen Benutzeroberfläche abrufbar macht. Dies hilft sowohl den Geschäften, wie auch den Konsumenten.

Produkt Funktion

CapWatch ist dazu da, um als Kunde einen einfachen Überblick zu bekommen wie viele Personen sich aktuell in einem Laden, Restaurant oder einer Sauna aufhalten. Da die Firmen aktuell gezwungen sind, das Besucheraufkommen zu erfassen und zu steuern, wollen wir diese Daten bündeln und somit den Kunden die Möglichkeit geben, das eigene Verhalten anzupassen und so unnötige Wartezeiten zu vermeiden.

Benutzer Charakteristik

Jede Person die eine Einrichtung mit Personenbeschränkung besuchen möchte und sich im Voraus über das Besucheraufkommen informieren will.

Einschränkungen

In der Grundausführung ist unser Produkt nur eine Zusammenfassung der vorhandenen Daten, welche in einer Übersicht angezeigt werden können. Die von den Firmen angelieferten Daten werden nicht ausgewertet. Es sind jeweils nur die aktuellsten Daten einsehbar. Bei den Kunden werden die Favoriten lokal im Browser gespeichert.

Annahmen

Wir gehen davon aus, dass die Firmen uns die gesammelten Daten zur Verfügung stellen werden und dass sie die Daten in einer Art erfassen, die es uns erlaubt sie weiterzuverarbeiten. Im Rahmen des Projektes werden wir nur mit Testdaten arbeiten, um externe Abhängigkeiten zu vermeiden.

Abhängigkeiten

Wir sind von Firmen abhängig, die uns aktuelle Daten liefern wollen und können.

1.4.3 Funktionale Anforderungen

Wir haben uns aufgrund des einfachen Geschäftsfalles dagegen entschieden Use Cases einzusetzen, da diese einen hohen Mehraufwand mit sich bringen würden. Wir setzen dafür auf einen schlanken Mix aus User Stories und MUSS/KANN Anforderungen.

Anlieferer / Anbieter

Als Anlieferer möchte ich mich registrieren können, damit ich den Konsumenten die aktuellen Besucherzahlen liefern kann.

Anforderungs-ID	Kategorie	Beschreibung
AL-1	MUSS	Der Anlieferer liefert regelmässig die aktuelle Auslastung mit Secret, Timestamp und Maximal erlaubter Auslastung an.
AL-2	KANN	Der Anlieferer kann sich per Request registrieren, indem er Firmenname, Ortschaft, einen Geschäftstyp und ein optionales Logo als Attachment mitschickt. Er erhält als Antwort ein Secret oder eine Fehlermeldung.

Konsument

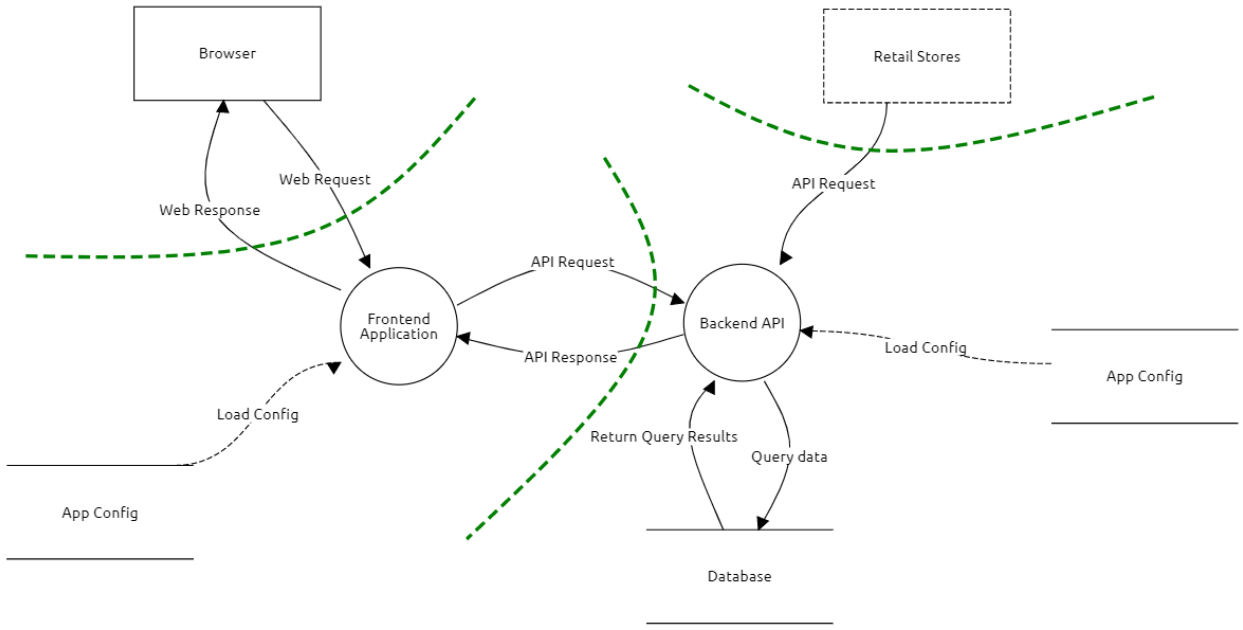
Als Konsument möchte ich mich über das aktuelle Besucheraufkommen informieren können.

Anforderungs-ID	Kategorie	Beschreibung
AW-1	MUSS	Als Konsument kann ich eine Liste aller verfügbaren Firmen anzeigen lassen.
AW-2	MUSS	Als Konsument kann ich nach einer Firma suchen.
AW-3	MUSS	Als Konsument kann ich eine Firma als Favorit markieren. Favoriten können in der Favoritenansicht angezeigt werden.
AW-4	KANN	Als Konsument kann ich mir eine Liste von Geschäften in meiner Umgebung nach Distanz sortiert anzeigen lassen.
AW-5	KANN	Ich kann mich als Konsument registrieren und anschliessend nach einem Login über mehrere Geräte hinweg auf meine Favoriten zugreifen.
AW-6	KANN	Als Konsument kann ich meine E-Mailadresse hinterlegen und mich so über wichtige Ereignisse informieren lassen.
AW-7	KANN	Als Konsument kann ich über eine Detailansicht einer Firma die Prognosen und Verläufe des Besucheraufkommens anzeigen lassen.
AW-8	KANN	Als Konsument kann ich mir bei einer Firma einen begrenzten Zeitraum reservieren.

Für das Minimal Viable Product (MVP) sind die MUSS-Anforderungen relevant. Falls Kapazität vorhanden ist, können aber auch noch KANN-Anforderungen umgesetzt werden. Es ist möglich, dass nicht alle KANN-Anforderungen umgesetzt werden.

1.4.4 Weitere Anforderungen

Threat Model nach OWASP



Da wir neben dem Engineering-Projekt gerade das Modul Secure Software besuchen, bot es sich zu Übungszwecken an ein Threat Model für CapWatch zu erstellen. Dieses Threat Model hilft, um das Produkt bestmöglich gegen Gefahren von ausserhalb abzusichern. Es wird laufend aktualisiert und es können neue nicht Funktionale Anforderungen daraus entstehen. Hierzu machen wir uns Gedanken zu folgenden Punkten:

- **Was sind unsere Assets:** In unserem MVP sind in unserem System nur Daten vorhanden die uneingeschränkt über unsere Webseite einsehbar sind. In der Datenbank sind nur regelmässige Einträge mit Timestamp, Anzahl Personen und maximal erlaubter Anzahl Personen vorhanden. Diese Metriken sind völlig anonym und werden als Zahlen angeliefert, Rückschlüsse auf einzelne Personen sind unmöglich. Zusätzlich speichern wir noch die Secrets der Firmen, die uns Daten anliefern. Diese Daten sind schützenswert, da es als Angreifer mit dem Secret möglich wäre unser System zu überlasten. In der späteren Ausbaustufe, in der wir Kundendaten erfassen und speichern, ist die Datenbank mit Kundendaten ein wichtiges Asset.
- **Threat Agents und mögliche Angriffe:** Interne Angriffe könnten die Kundendaten abgreifen um die gespeicherten E-Mailadressen weiterzuverwenden. Dies wäre möglich über einen Zugriff auf die Kundendatenbank, wenn sie nicht genügend gut per Zugriffsrechte abgesichert wird. Ein externer Angreifer, welcher eine Organisierte Verbrecherbande oder ein einzelner Hacker sein kann, müsste sich die Zugangsdaten der Datenbank beschaffen um direkt darauf zuzugreifen oder er schafft es über die API an mehr Daten als eigentlich vorgesehen zu kommen.
- **Mögliche Schwachstellen:** Bei der Entwicklung unseres Produktes können mehrere Schwachstellen entstehen. Dazu gehören unsauber aufgesetzte Berechtigungen, fehlende Inputsäuberung und Inputvalidierung, überdimensionierte Schnittstellen mit zu vielen Feldern, sowie schwache Passwörter und fehlende Passwortverwaltung.
- **Gegenmassnahmen:** Den internen Angriff kann man mit einem Berechtigungsmodell, welches nach dem Prinzip *so wenig wie nötig* aufgesetzt ist, mitigieren. Der Zugriff auf die Kundendatenbank wird über einen Supportuser gelöst, welcher nur mit Begründung und Dokumentation der Tätigkeiten benutzt werden kann. Um die externen Angriffe zu erschweren kommen sichere Passwörter und klar definierte Schnittstellen zum Einsatz. Die Schnittstellen dürfen nur die klar definierten Felder verwenden und der Inhalt der Anfragen wird vor der Verarbeitung gesäubert um unerwünschte Effekte zu vermeiden.
- **Aktuelle Schwachstellen:** Bei der Generierung der Datenbank wird aktuell das Passwort des Benutzers im Klartext in das Init-Script reingeschrieben. Da ansonsten die Entwicklung mühsam ist und das automatische Aufsetzen der Entwicklungsumgebung nicht möglich ist. Für den späteren Projektverlauf gibt es die Möglichkeit, die Passwörter erst in der Pipeline einzufügen. Zusätzlich gibt es die Möglichkeit, Data-at-Rest in der Datenbank zu verschlüsseln, was aktuell auch noch nicht umgesetzt worden ist. Dies wäre insbesondere für die Secrets der Stores Pflicht bevor das Produkt live eingesetzt werden könnte.

Qualitätsmerkmale

Die Nicht Funktionalen Anforderungen, aufgeteilt in MUSS und KANN Anforderungen, werden im folgenden Abschnitt aufgelistet. Die Auflistung ist nicht abschliessend und wird ständig den aktuellen Gegebenheiten angepasst.

Anforderungs-ID	Kategorie	Beschreibung
NF-1	KANN	In Benutzerumfragen wollen wir eine durchschnittliche Bewertung von mindestens 4 von 5 Sternen erreichen in den Punkten: Attraktivität, Bedienbarkeit, Erlernbarkeit und Verständlichkeit
NF-2	MUSS	Der Kunde soll innerhalb von einer Sekunde das Resultat einer Abfrage zur Verfügung haben.
NF-3	MUSS	Unser Produkt ist unter einer Last von 100 Anfragen pro Sekunde noch schnell genug, dass man aus Kundensicht keine Einschränkungen bemerkt.
NF-4	MUSS	Die Webseite soll mit 50 Stores noch flüssig laufen.
NF-5	MUSS	Wir stellen die Anforderungen des Datenschutzgesetzes (DSG) sicher.
NF-6	MUSS	Die häufigsten Angriffspunkte und Schwachstellen nach OWASP werden berücksichtigt.
NF-7	KANN	Ein mit dem Projekt nicht vertrauter, erfahrener Entwickler sollte bei einem einfach Problem innerhalb von 15min die betroffene Codestelle gefunden haben.
NF-8	KANN	Die Ergebnisse sollten nicht älter als 15min sein.

Schnittstellen

Im Dokument bzw. Bereich „API Dokumentation“ gibt es noch eine genaue Beschreibung der Schnittstellen.

- **Webapplikation**
  - Schnittstelle zum Backend um Besucherdaten abzufragen (1. Schritt)
  - Schnittstelle zum Backend für Authentifizierung und Autorisierung (2. Schritt)
- **Backend**
  - Schnittstelle zum Frontend um Besucherdaten zu liefern (1. Schritt)
  - Schnittstelle zur Datenbank um Daten abzulegen und zu lesen (1. Schritt)
  - Schnittstelle um Daten von den Firmen entgegenzunehmen (1. Schritt)
  - Schnittstelle um die Firmenregistrierung zu ermöglichen (2. Schritt)
  - Schnittstelle zum Frontend für die Authentifizierung und Autorisierung (2. Schritt)
- **Datenbank**
  - Schnittstelle zu Backend um Daten entgegenzunehmen oder zu liefern

Screendesigns

Wir haben direkt Screendesigns entwickelt, weil ein Grobdesign (z.B. Wireframes) für solch einen kleinen Funktionsumfang nicht optimal ist und nur einen unnötigen Mehraufwand bedeuten würde. Unsere Plattform wird Mobile First entwickelt, weil wir viel mehr Benutzer unterwegs bzw. auf ihrem Mobiltelefon erwarten.

Nur die Startpage auf Mobile ist relevant für das Minimum Viable Product.

Alle Screendesigns werden unter diesem Link zu Figma produziert und abgelegt.

Randbedingungen

Wir sind abhängig von der Serverinfrastruktur der Ost. Wir haben keinen Einfluss auf diese und vertrauen darauf, dass sie uns ohne längere Unterbrüche und vor allem bei der Präsentation zur Verfügung steht.

Ansonsten gibt es keine Randbedingungen.

1.5 Domainanalyse

1.5.1 Einführung

Zweck

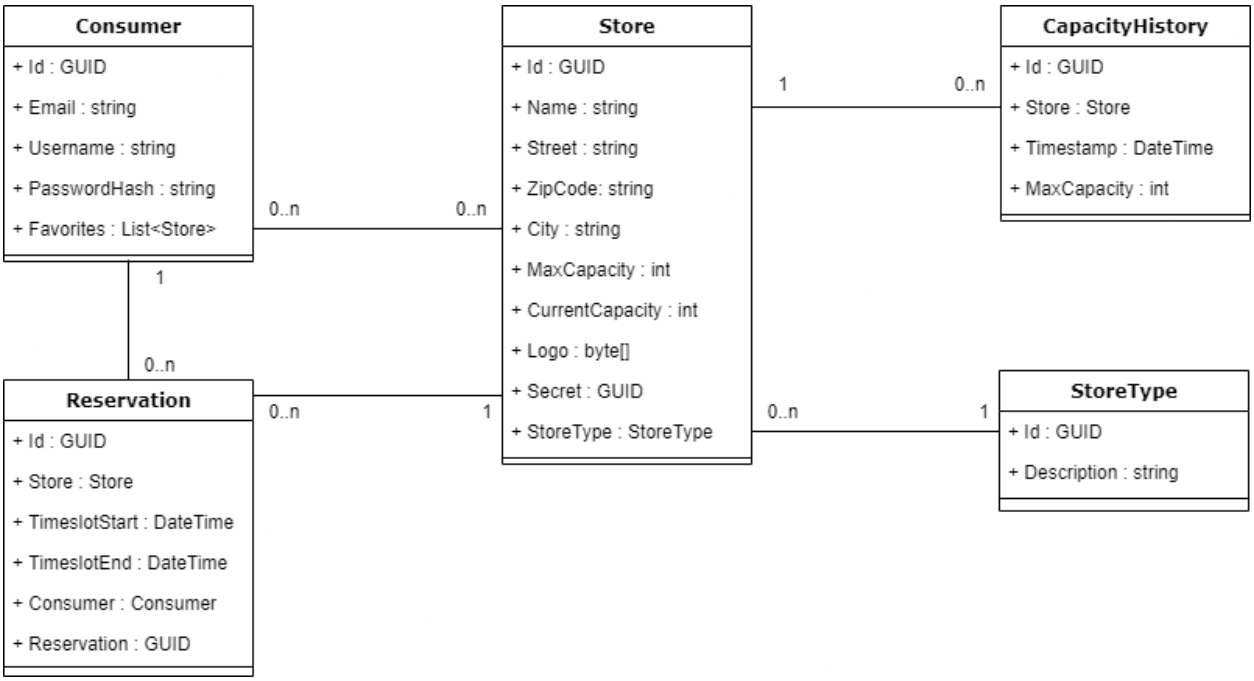
Dieses Projekt gibt eine Übersicht über die Domain des Projektes CapWatch.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

1.5.2 Domain Modell

Strukturdiagramm



Da C# bereits als Backendtechnologie festgelegt wurde, werden im Domain Model technologiespezifische Datentypen verwendet.

Wichtige Konzepte

Store

Store ist die einzige Klasse, welche für den MVP relevant ist. In dieser Klasse sind Name, Adresse und Logo enthalten, sowie die aktuelle/maximale Auslastung. Die Applikation unterstützt nur Stores in der Schweiz.

Consumer

Die Klasse Consumer wird für das Login und das Speichern von Favoriten des Konsumenten verwendet.

CapacityHistory

Um die erwartete und durchschnittliche Auslastung zu berechnen, wird die Auslastung mit einem Zeitstempel als Verlauf abgelegt.

Reservation

Der Konsument kann sich bei einem Store ein Zeitfenster reservieren und erhält dafür eine Reservations-ID.

1.5.3 Systemsequenzdiagramme

Die aktuell geplanten Funktionalitäten sind einfach im Aufbau, Systemsequenzdiagramme bringen daher keinen Mehrwert. Aus diesem Grund werden sie in dieser Dokumentation bewusst weggelassen.

1.6 CapWatchBackend.WebApi v1

> Scroll down for code samples, example requests and responses. Select a language for code samples from the tabs above or the mobile navigation menu.

1.6.1 Stores

get__Stores

Code samples

```
fetch('/Stores',
{
  method: 'GET'

})
.then(function(res) {
    return res.json();
}).then(function(body) {
    console.log(body);
});
```

```
GET /Stores
```

Responses

This operation does not require authentication

patch__Stores

Code samples

```
const inputBody = '{
    "id": "string",
    "name": "string",
    "street": "string",
    "zipCode": "string",
    "city": "string",
    "maxCapacity": 1,
    "currentCapacity": 0,
    "logo": "string",
    "secret": "string",
    "storeType": {
        "id": "string",
        "description": "string"
  }
}';
const headers = {
  'Content-Type':'application/json'
};

fetch('/Stores',
{
  method: 'PATCH',
  body: inputBody,
  headers: headers
})
.then(function(res) {
    return res.json();
}).then(function(body) {
    console.log(body);
});
```

`PATCH /Stores`

Body parameter

```
{
  "id": "string",
  "name": "string",
  "street": "string",
  "zipCode": "string",
  "city": "string",
  "maxCapacity": 1,
  "currentCapacity": 0,
  "logo": "string",
  "secret": "string",
  "storeType": {
    "id": "string",
    "description": "string"
  }
}
```

Parameters

Responses

This operation does not require authentication

post__Stores

Code samples

```
const inputBody = '{
  "name": "string",
  "street": "string",
  "zipCode": "string",
  "city": "string",
  "maxCapacity": 1,
  "logo": "string",
    "storeType": {
        "id": "string",
        "description": "string"
    }
}';
const headers = {
  'Content-Type':'application/json'
};

fetch('/Stores',
{
  method: 'POST',
  body: inputBody,
  headers: headers
})
.then(function(res) {
    return res.json();
}).then(function(body) {
    console.log(body);
});
```

`POST /Stores`

Body parameter

```
{
  "name": "string",
  "street": "string",
  "zipCode": "string",
  "city": "string",
  "maxCapacity": 1,
  "logo": "string",
  "storeType": {
    "id": "string",
    "description": "string"
  }
}
```

Parameters

Responses

This operation does not require authentication

get__Stores_{id}

Code samples

```
fetch('/Stores/{id}',
{
  method: 'GET'

})
.then(function(res) {
    return res.json();
}).then(function(body) {
    console.log(body);
});
```

```
GET /Stores/{id}
```

Parameters

Responses

This operation does not require authentication

1.6.2 Schemas

NewStoreModel

```
{
  "id": "string",
  "name": "string",
  "street": "string",
  "zipCode": "string",
  "city": "string",
  "maxCapacity": 1,
  "currentCapacity": 0,
  "logo": "string",
  "storeType": {
    "id": "string",
    "description": "string"
  }
}
```

Properties

StoreModel


```
{
  "id": "string",
  "name": "string",
  "street": "string",
  "zipCode": "string",
  "city": "string",
  "maxCapacity": 1,
  "currentCapacity": 0,
  "logo": "string",
  "secret": "string",
  "storeType": {
    "id": "string",
    "description": "string"
  }
}
```


Properties

1.7 Softwarearchitektur

1.7.1 Einführung

Zweck

Dieses Dokument dient als Übersicht über die Architektur des Softwareprojekts CapWatch.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

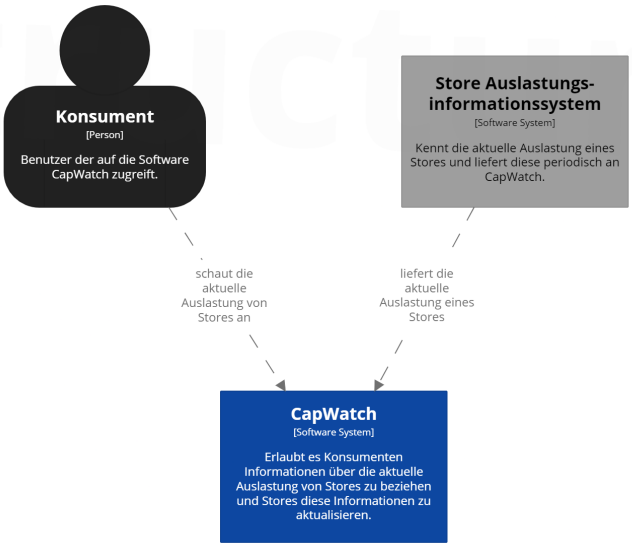
1.7.2 Referenzen

Die C4 Diagramme wurden mit Structurizr erstellt.

1.7.3 C4 Modell

Wir haben uns entschieden zur Visualisierung unserer Architektur das C4 Modell zu verwenden, da dieses mit übersichtlichen und einfach zu verstehenden Diagrammen sehr viel Information über ein System vermittelt. Dies hilft insbesondere Personen, die sich neu in das Projekt einlesen, sich schnell zurecht zu finden.

System Kontext Diagramm der Software CapWatch



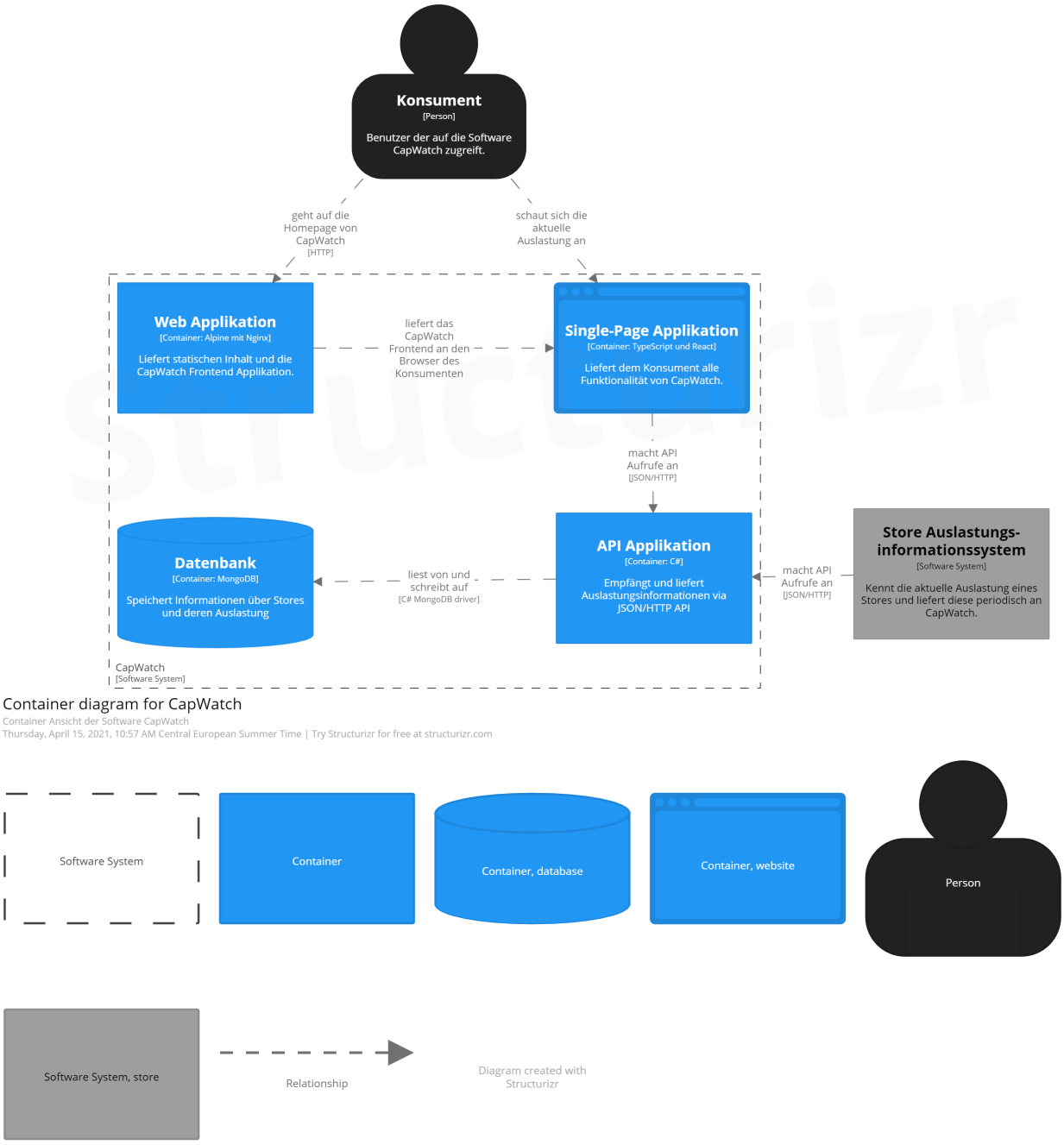
System Context diagram for CapWatch

System Context Diagramm der Software CapWatch

Diagram created with Structurizr | Wednesday, April 14, 2021, 10:16 PM Central European Summer Time

Container Diagramm der Software CapWatch (*)

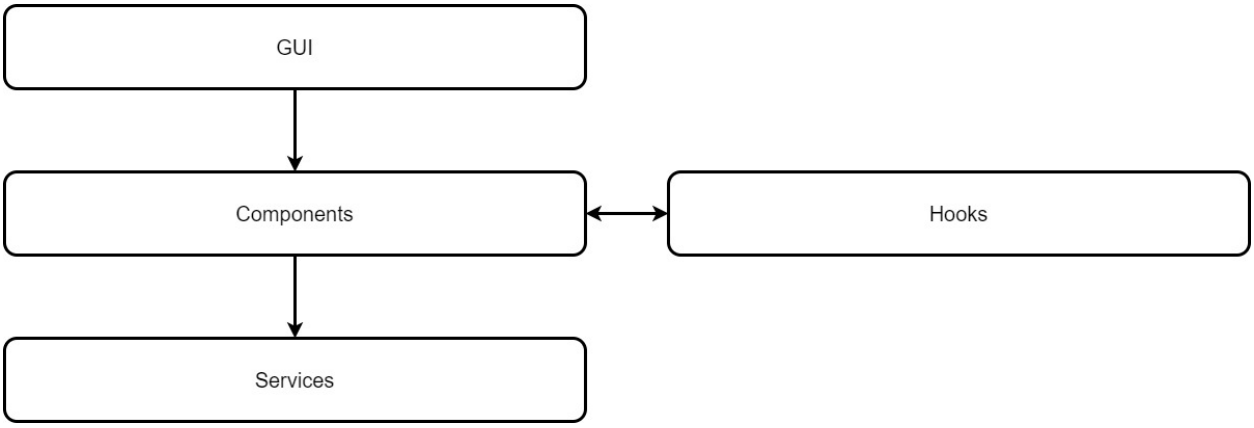
(*) Das Wort Container wird hier in der Definition nach C4 Modell verwendet und nicht im Kontext von Docker.



Das Gesamtsystem besteht aus einer MongoDB Datenbankinstanz und einem C# Backend, welches über eine WebAPI (REST) mit dem React Frontend und externen Systemen der Stores kommuniziert.

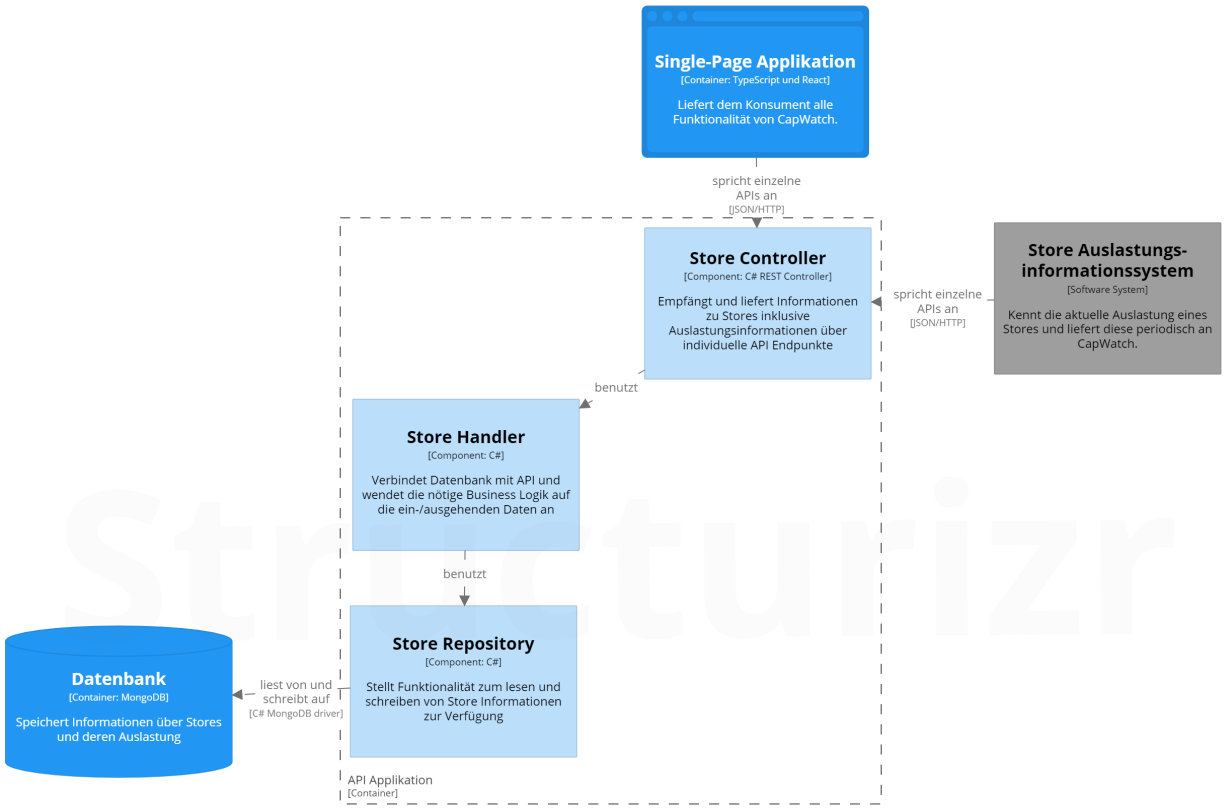
Architektur Frontend

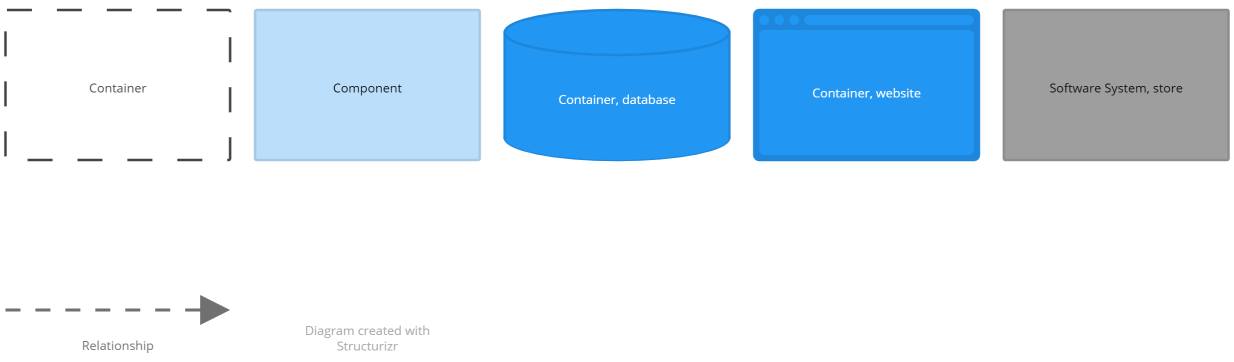
Das Frontend besteht aus einer Single-Page Applikation, welche in React und TypeScript geschrieben wurde. Die Applikation wird auf einem Nginx Webserver betrieben. Das Frontend ist in drei Schichten gegliedert. Zuoberst befindet sich die GUI Schicht, welche die Informationen darstellt. Auf der nächsten Schicht befinden sich die Komponenten und Hooks. Die Komponenten stellen die Daten für die GUI Schicht bereit. Die Hooks verantworten die States und stellen weitere React Features zur Verfügung. Die Service Schicht ist für die Kommunikation und Datenverarbeitung mit dem Backend zuständig.



Als Frontend Frameworks standen Angular, React und Vue zur Auswahl. Die Entscheidung fiel auf React, da Angular für dieses Projekt zu gross ist. React wurde Vue bevorzugt, da wir hier auf existierendes Vorwissen zurückgreifen können und im Team Interesse bestand das React-Framework zu erlernen. Ein Projektmitarbeiter besucht parallel zum EPJ das Modul *Web Engineering und Design 3* und kann somit das gelernte Wissen direkt umsetzen.

Komponenten Diagramm der API Backend Applikation





Store Controller

Der Store Controller bietet eine klassische WebAPI an, über welche Informationen zu Stores angefragt und aktualisiert werden kann. Welche API Schnittstellen im Detail zur Verfügung stehen kann in der API Dokumentation eingesehen werden.

Store Handler

Der Store Handler liegt in der Applikationsschicht und verbindet das Store Repository mit dem Store Controller. Der Handler implementiert die Businesslogik, welche auf ein- und ausgehende Stores angewendet wird.

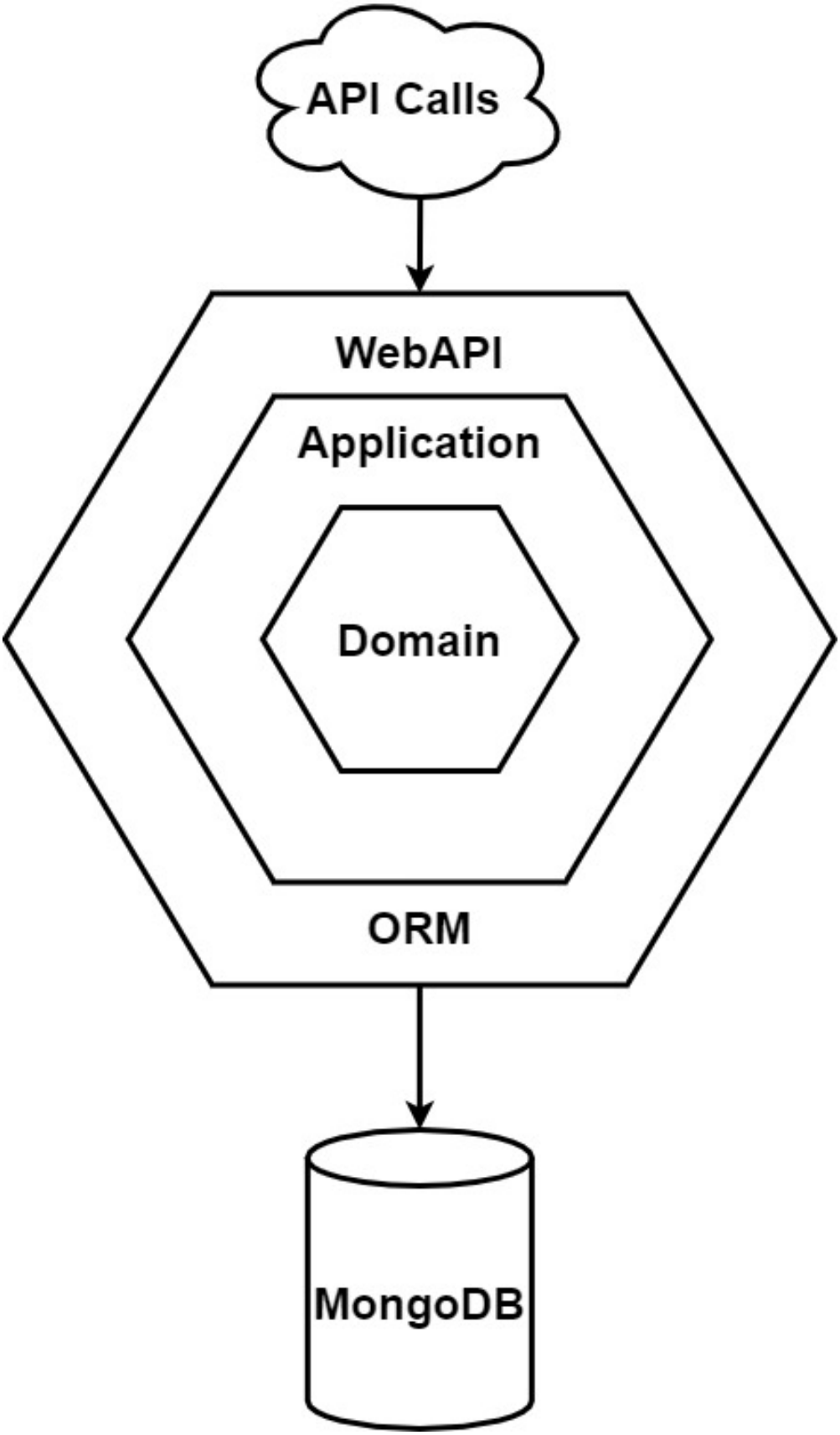
Store Repository

Das Store Repository wird von der Applikationsschicht definiert und vom MongoDB Adapter implementiert. Es stellt einen Satz von Standardoperationen zur Verfügung mit welchen für Stores alle benötigten CRUD Operationen durchgeführt werden können.

Architektur Backend

Das Backend ist nach hexagonaler Architektur in C# konzipiert. Hierbei steht die Domain im Zentrum und beinhaltet die Entitäten, um welche sich das System dreht. Nach der Domainschicht liegt die Application, welche die Businesslogik beinhaltet und Interfaces definiert, die von den umliegenden Schichten implementiert und genutzt werden. Ganz aussen ist auf der einen Seite ein Adapter für die Datenbankanbindung unserer MongoDB Instanz. Diese implementiert die CRUD Operationen mithilfe eines ORM. Auf der anderen Seite liegt die WebAPI, welche die nötigen Web-Schnittstellen zur Verfügung stellt.

Unsere Entscheidung fiel auf C#, da der Mehrheit des Teams die Technologie besser liegt als Java. Dazu kommt, dass unser Architekt aus seiner Berufstätigkeit viel Wissen in dieser Technologie, insbesondere im Bezug auf Web API Applikationen, mitbringt.


API Calls
WebAPI
Application
Domain
ORM
MongoDB

Wir haben uns für eine hexagonale Architektur entschieden, da es diese sehr einfach macht Adapter auszutauschen. Die Architektur macht auch das Testen einzelner Teile der Applikation unkompliziert. Dies erleichtert es uns gute Unit Tests zu schreiben um eine hohe Softwarequalität zu erreichen. Ein weiterer Vorteil ist, dass korrekt umgesetzte hexagonale Architektur zu loser Kopplung zwischen den einzelnen Teilen der Software führt. Auch ein wichtiger Punkt war die Möglichkeit unser Wissen bezüglich Software Architektur auszuweiten.

Aufgeteilt ist der Backend Quellcode auf vier Projekte, welche den Schichten und Adaptern der Hexagonalen Architektur entsprechen:

- **CapWatchBackend.Domain**
- **CapWatchBackend.Application**
- **CapWatchBackend.DataAccess.MongoDB**
- **CapWatchBackend.WebApi**

Um sicherzustellen, dass diese Projekte nur lose gekoppelt sind, sind Klassen welche Logik implementieren von ausserhalb der Projektgrenze nur über Interfaces verfügbar. Diese werden bei Bedarf per Constructor Dependency Injection instanziert.

1.7.4 Deployment

Die Software wird auf einen einzelnen Linux Server deployt. Dieser wird von der OST zur Verfügung gestellt und befindet sich im dafür vorgesehenen DMZ Netzwerk der OST. Das Deployment selber wird mittels Docker Images umgesetzt, die über die Registry auf dem GitLab direkt bezogen werden. Zur einfachen Verwaltung des Docker basierten Setups wird auf dem Host Docker-Compose eingesetzt. Vorerst ist ein manuelles Deployment und Updaten der Software vorgesehen. Falls es die Umstände während des Projekts später zulassen, wird das Setup um eine dann noch zu definierende Komponente für das automatische Deployment erweitert. Die Container werden jeweils ab dem Master- und dem Develop-Branch mittels CI Pipeline automatisch gebaut und direkt der Registry hinzugefügt.

Aufgrund der Trivialität des Setup verzichten wir in an dieser Stelle auf ein Deployment Diagramm.

1.7.5 Datenspeicherung

Zur Speicherung der Daten benutzen wir MongoDB. Wir haben uns für einen NoSQL Dokumentspeicher entschieden, da CapWatch auch mit optionalen Erweiterungen kaum relationale Abhängigkeiten besitzt. Des Weiteren war bereits ein Grundwissen zur Technologie vorhanden und durch die existierende sehr umfangreiche Dokumentation ist sichergestellt, dass wir bei möglichen Problemen eine gute Informationsquelle haben.

Zum aktuellen Zeitpunkt umfasst die Datenbank nur eine Collection in welcher Store Dokumente verwaltet werden. Die Felder der Dokumente entsprechen den Properties der Store Entitäten.

1.7.6 Grössen und Leistung

Aus den Performance Tests mit JMeter können wir bestätigen, dass die Applikation serverseitig sehr performant ist. CapWatch kann daher auf beliebigen Systemen, auf denen mindestens Docker betrieben werden können, deployt werden.

Bei übermässig vielen Einträgen in der Store Tabelle benötigt das Frontend auf dem Zielsystem des Anwenders sehr viel Leistung. Die User Experience wird dadurch stark eingeschränkt. Dieses Problem könnte durch Paging der Store Einträge jedoch relativ einfach behoben werden und zählt zu den KANN- bzw. den Ausbauanforderungen.

1.7.7 Ausbau-Szenario

Bezüglich Ausbau-Szenarien haben wir uns bereits bei der Erstellung der Anforderungsspezifikation Gedanken gemacht. Mögliche Erweiterungen sind in den Tabellen bei den Funktionalen Anforderungen und Qualitätsmerkmalen unter der Kategorie *KANN* aufgeführt.

1.7.8 Performance-Szenario

Das Frontend braucht unsererseits nur Performance bis die React Applikation einmal beim Konsumenten in den Browser geladen ist. Danach läuft diese dort als Single-Page Applikation und es ist nur noch die Performance der API relevant. Nichtsdestotrotz wird das Frontend natürlich so performant wie möglich geschrieben, damit auch Anwender mit weniger starken Systemen ein gutes Erlebnis haben. Dazu wird der Virtual DOM so sinnvoll wie möglich eingesetzt und es werden nur Komponenten gezeichnet die sich verändert haben.

Das CapWatch Backend ist zustandslos und kann somit mit geringem Aufwand mehrfach hochgefahren und die verschiedenen Instanzen mit einem Load Balancer gleichmässig bedient werden.

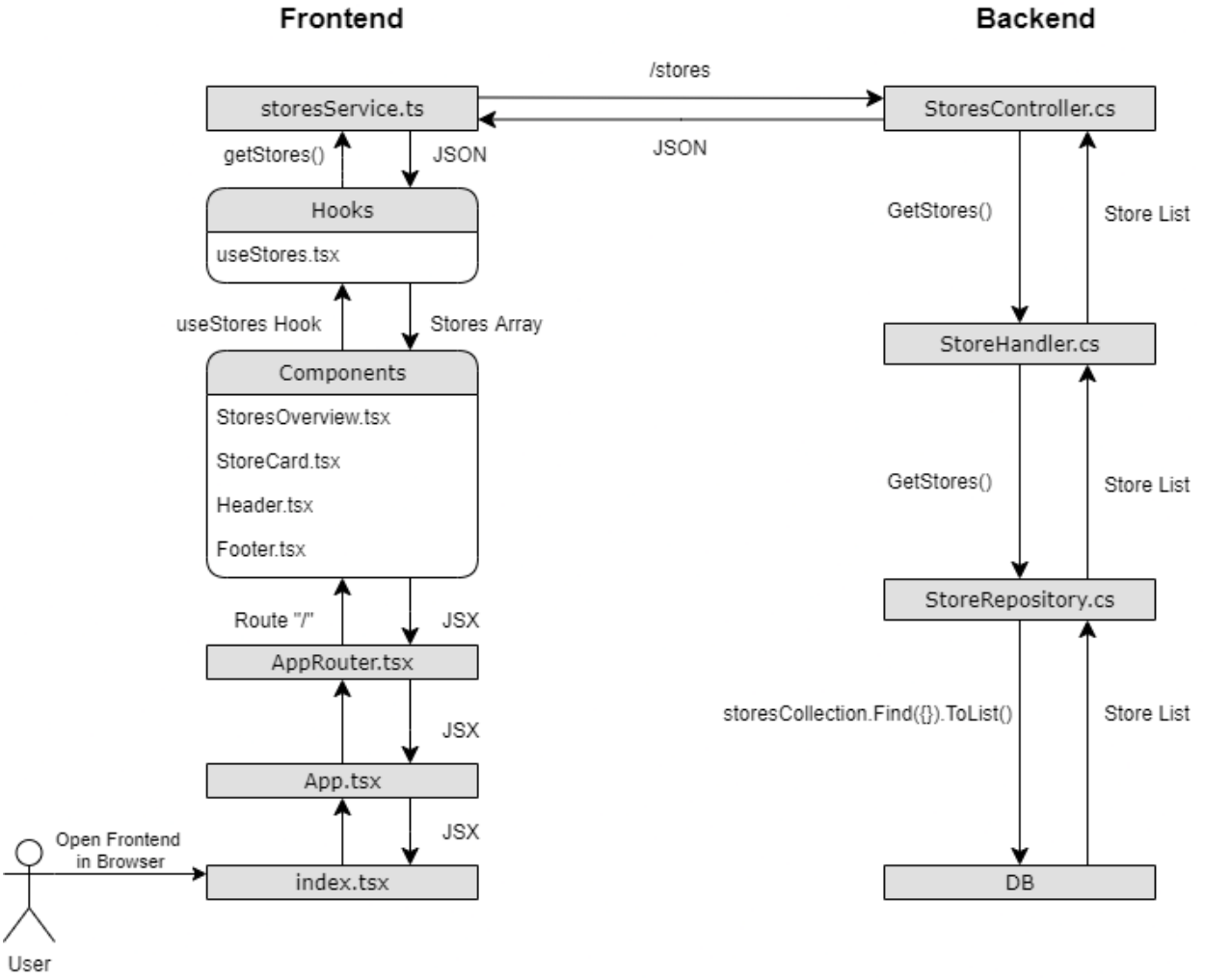
Falls bei der Datenbank Performance Probleme relevant werden sollten, wären die beiden möglichen Optionen Replication und Sharding. Bei der Replication werden die gesamten Daten der Datenbank auf mehreren Systemen redundant gehalten und die Anfragen werden zwischen diesen Systemen verteilt. Beim Sharding werden die Daten ebenfalls auf mehrere Systeme verteilt, allerdings ist hier auf jedem System nur ein Teil der Daten und Anfragen werden auf das jeweils richtige System geleitet. Mit diesen Lösungen haben wir noch keine Erfahrungen und wir müssten noch einiges an Wissen gewinnen, sollte dies nötig werden. Aus diesem Grund können wir auch keine Aussage dazu machen wie aufwändig das Umsetzen einer dieser beiden Lösungen wäre.

1.7.9 Technische Schulden

Um das Frontend aktuell zu halten, muss regelmässig die Liste der Stores und deren Auslastung geladen werden. Hier würde sich ein Websocket anbieten, welcher dem Frontend mitteilt, falls es Änderungen gibt. Bas Backend bietet bereits einen Websocket an, welcher diese Information zur Verfügung stellt. Dieser wird in der Beta Version allerdings vom Frontend noch nicht berücksichtigt, weil unterwartete grössere Restriktionen vorgekommen sind.

1.7.10 Durchlaufene Klassen

Hier folgt noch eine grobe Übersicht über die wichtigsten Klassen / Methoden, welche bei einem Aufruf der Webseite durchlaufen werden.



1.7.11 Weggelassene Dokumentation

- Wir haben uns dazu entschieden keine Designklassendiagramme zu entwerfen, weil diese uns gegenüber dem Strukturdiagramm im Domainmodell keinen Mehrwert bieten.
- Wir sind bei der Dokumentation der Architektur bewusst nicht bis auf die Ebene Code (nach C4) / Klassendiagramme hinuntergegangen, da diese in den wenigsten Fällen wirklich hilfreich sind. Sollte trotzdem irgendwann ein Klassendiagramm nötig sein, kann dieses aus dem Code generiert werden, was dann auch garantiert, dass es die aktuellsten Informationen enthält.

1.8 Qualitätssicherung

1.8.1 Einführung

Zweck

Dieses Dokument ist eine Übersicht aller Massnahmen zur Qualitätssicherung im Projekt CapWatch.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Git Conventional CommitsGit Flow - GitHubGit Flow - Flow GraphsSemantic Versioning

1.8.2 Qualitätsmassnahmen

Die qualitätssichernden Massnahmen wurden bereits im Kapitel Qualitätsmassnahmen im Projektplan beschrieben. Nachfolgend werden einzelne wichtige Punkte noch genauer ergänzt.

Git

Branching

Die Branches werden mit den Konzepten und dem Tooling von Git Flow (AVH Edition) erstellt und verwendet. Wir verwenden `feature`, `bugfix`, `release` und `hotfix` Branches mit den beiden Hauptbranches `develop` für die Entwicklungsumgebung und `master` für die Produktionsumgebung. Wir brauchen in diesem Projekt keine `support` Branches. Alle Branches werden in `kebab-case` benannt, ausser die `release` und `hotfix` Branches, welche in Semantischen Versionsnummern verfasst werden ohne Prefix.

Commits

Alle Commits müssen den Vorgaben zu Conventional Commits folgen. Dadurch wird sichergestellt, dass die Commit Messages einheitlich sind, die Erweiterungen und Anpassungen genau beschreiben und die Änderungsgeschichte schnell nachvollziehbar ist.

Frontend Tests mit Lighthouse

Mit Lighthouse werden im Frontend die Kategorien Performance, Best Practices und Accessibility getestet. Die Tests haben mobile Endgeräte als primäre Zielplattform.

Konfiguration Renovate

Renovate erkennt, wenn eingesetzte Bibliotheken nicht mehr aktuell sind und erstellt für die Aktualisierung automatisch einen Merge Request, welcher dann nur noch von einem Teammitglied begutachtet und zusammengeführt werden muss.

Renovate im Frontend

Renovate überprüft das Frontend einmal die Woche am Dienstag um 06:00.

Renovate im Backend

Renovate kann aktuell im Backend nicht eingesetzt werden, weil die verwendete NuGet-Version noch nicht unterstützt wird.

SonarQube

Mit SonarQube wird für das Frontend und Backend eine Codeanalyse durchgeführt. Dabei werden folgende Kriterien ausgewertet:

- Bugs
- Vulnerabilities
- Security Hotspots
- Code Smells
- Code Coverage
- Duplication

SonarQube im Frontend

Da wir im Frontend keine automatisierten Tests schreiben, können wir die vorgegebene Code-Coverage von 80 Prozent nicht erreichen. Die Analyse schlägt deshalb in der Code Coverage fehl, da wir die verwendete SonarQube Instanz nicht auf unsere Bedürfnisse anpassen können. Aus diesem Grund läuft die Build-Pipeline auch erfolgreich durch, wenn SonarQube fehlschlägt. Jeder Entwickler ist selbst verantwortlich, die SonarQube checks in jedem Merge Request manuell zu analysieren.

SonarQube im Backend

In der Pipeline des Backends ist SonarQube aktiv. Wenn eine SonarQube Überprüfung fehlschlägt, wird ein Merge Request automatisch zur Zusammenführung gesperrt.

DoD - Definition of Done

Es gibt verschiedene DoD's. Zur Qualitätssicherung können die Tickets mit einer entsprechenden DoD versehen werden. Die DoD dient den Entwicklern als Leitfaden für die Umsetzung. Es gibt standardisierte DoD's für das Backend, Frontend und die Dokumentation. Diese können für jeden Auftrag individuell ergänzt werden.

Standard DoD

-[ ] Auftrag gelesen und verstanden -[ ] Alle Teile des Auftrags umgesetzt -[ ] Anforderungen mit Lösung verglichen -[ ] Ergebnisse im Team besprochen oder alle informiert -[ ] Review erstellt, umgesetzt und Merge durchgeführt (Merge Request)

Backend DoD

Die standard DoD wird für das Backend um folgende Punkte ergänzt.

-[ ] Coding Guidelines und Formatierung überprüft -[ ] Unit Tests erstellt -[ ] Code Review erstellt, umgesetzt und Merge durchgeführt (Merge Request)

Frontend DoD

Die standard DoD wird für das Frontend um folgende Punkte ergänzt.

-[ ] Coding Guidelines und Formatierung überprüft -[ ] Code Review erstellt, umgesetzt und Merge durchgeführt (Merge Request)

Technische Schulden in der Qualitätssicherung

- Renovate im Backend einsetzten, sobald dies unterstützt wird
- SonarQube für die Anforderungen durch CapWatch konfigurieren

1.8.3 Sicherung der Geschichte

Code und Dokumentation

Entwicklung und Dokumentation geschieht in je einem GitLab Projekt mit Repositories, welche auf dem OST GitLab gehostet werden. Somit ist eine Versionierung und Sicherung dieser Daten vorhanden. Durch die dezentrale Natur von Git ist sämtliche Arbeit auch bei allen Entwicklern lokal meistens fast vollständig vorhanden.

Zeitauswertung / Tickets

Um die Arbeitszeit jederzeit nachweisen zu können, werden am Ende jedes Meilensteins die Auswertungen `Time Report per Milestone` und `Time Report per work item` erstellt und in der Dokumentation abgelegt. Die Zeitauflösung in den Auswertungen ist in Minuten angegeben.

1.9 Systemtest-Spezifikation

1.9.1 Einführung

Zweck

Dieses Dokument beschreibt in welcher Form Systemtests an der CapWatch Software durchgeführt werden sollen.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Die verwendeten Anforderungsbezeichnungen beziehen sich auf das Dokument Anforderungsspezifikation.

1.9.2 Voraussetzungen

Vollumfängliches Deployment der Software CapWatch muss vorhanden sein.

1.9.3 Systemtest

Die folgende Liste von Tests deckt alle Funktionalitäten der Beta Version von CapWatch ab.

Anforderung	Beschreibung
AL-1-1	PATCH Request vorbereiten mit korrektem Body im JSON Format. Request an URL localhost:8080/stores senden. Antwort muss 200 sein ohne Rückgabe.
AL-2-1	POST Request vorbereiten mit korrektem Body im JSON Format. Request an URL localhost:8080/stores senden. Antwort muss 200 sein und die id und das secret müssen in der Antwort zurückgesendet werden.
AL-2-2	POST Request vorbereiten mit fehlendem Namen im JSON Format. Request an URL localhost:8080/stores senden. Antwort muss 400 sein und es muss der Fehler „Name muss Vorhanden sein“ als Antwort zurückkommen.
AW-1-1	Datenbank nicht anbinden. URL des CapWatch Frontend aufrufen. Es wird eine Meldung angezeigt, dass ein Fehler aufgetreten ist.
AW-1-2	Datenbank anbinden aber keine Einträge erfassen. URL des CapWatch Frontend aufrufen. Es wird eine Meldung angezeigt, dass keine Geschäfte gefunden worden sind.
AW-1-3	Auf Datenbank manuell Geschäfte hinzufügen. URL des CapWatch Frontend aufrufen. Es wird eine Liste von Geschäften mit Informationen zur Auslastung angezeigt.
AW-2-1	Aufbauend auf AW-1-3. Wenn die Liste angezeigt wird, auf die Lupe klicken und im Suchfeld nach einem Geschäft suchen. Es wird eine nach dem Suchbegriff gefilterte Liste angezeigt.
AW-3-1	Aufbauend auf AW-1-3. Wenn die Liste angezeigt wird, noch kein Geschäft als Favorit markieren und in die Favoritenansicht wechseln. Es dürfen keine Geschäfte angezeigt werden sondern die Meldung aus AW-1-2.
AW-3-2	Aufbauend auf AW-1-3. Wenn die Liste angezeigt wird, auf einem Geschäft das Herzsymbol anklicken und es so als Favorit markieren. In die Favoritenansicht wechseln. Das ausgewählte Geschäft wird nun angezeigt.
AW-3-3	Aufbauend auf AW-3-2. In der Favoritenansicht das Geschäft wieder als Favorit entfernen durch nochmaliges klicken auf das Herzsymbol. Das Geschäft verschwindet aus der Ansicht und es wird die Meldung angezeigt, dass kein Geschäft gefunden worden ist.

1.10 Systemtest-Protokoll 2. April 2021

1.10.1 Einführung

Zweck

Dieses Dokument dient dazu, eine Übersicht zu geben welche Systemtests wann durchgeführt wurden und was das Resultat war.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Systemtestspezifikation

1.10.2 Angaben zur Durchführung

Testdatum: 2021-04-02

Getestet wurde auf folgendem Stand der jeweiligen Git Repositories.

- **Frontend:** f04ca76a
- **Backend:** af817389

Um den Test durchzuführen wurde in beiden Repositories über Docker-Compose die Software gestartet.

1.10.3 Protokoll

Anforderung	Implementiert	Kommentare	Status
AW-1-2	ja		erfüllt

1.10.4 Verbesserungsmöglichkeiten

Da es sich aktuell noch um einen Architekturprototypen handelt wird nicht der bestehende Stand verbessert, sondern mit der Implementierung der effektiven Software fortgefahren.

Bekannte Einschränkungen

Die meisten Funktionalitäten sind noch nicht implementiert, da es sich noch um einen Architekturprototypen handelt.

1.11 Systemtest-Protokoll 16. April 2021

1.11.1 Einführung

Zweck

Dieses Dokument dient dazu, eine Übersicht zu geben welche Systemtests wann durchgeführt wurden und was das Resultat war.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Systemtestspezifikation

1.11.2 Angaben zur Durchführung

Testdatum: 2021-04-16

Getestet wurde auf folgendem Stand der jeweiligen Git Repositories.

- **Frontend:** d6063e72
- **Backend:** e0fcdadf

Um den Test durchzuführen wurde in beiden Repositories über Docker-Compose die Software gestartet.

1.11.3 Protokoll

Anforderung	Implementiert	Kommentare	Status
AL-1-1	ja		erfüllt
AL-2-1	ja		erfüllt
AL-2-2	ja		erfüllt
AW-1-1	ja		erfüllt
AW-1-2	ja		erfüllt
AW-1-3	ja		erfüllt
AW-2-1	nein	Wurde nicht umgesetzt da Umfang zu Gross	nicht getestet
AW-3-1	ja		erfüllt
AW-3-2	ja		erfüllt
AW-3-3	ja		erfüllt

1.11.4 Verbesserungsmöglichkeiten

Der Umfang wurde unterschätzt und darum wurde auf eine Funktionalität verzichtet. Mit mehr Erfahrung werden hoffentlich die Schätzungen besser.

Bekannte Einschränkungen

Eine MUSS Anforderung, die Suche, wurde aus Zeitgründen nicht umgesetzt und wird im nächsten Sprint implementiert.

1.12 Systemtest-Protokoll 29. April 2021

1.12.1 Einführung

Zweck

Dieses Dokument dient dazu, eine Übersicht zu geben welche Systemtests wann durchgeführt wurden und was das Resultat war.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Systemtestspezifikation

1.12.2 Angaben zur Durchführung

Testdatum: 2021-04-29

Getestet wurde auf folgendem Stand der jeweiligen Git Repositories.

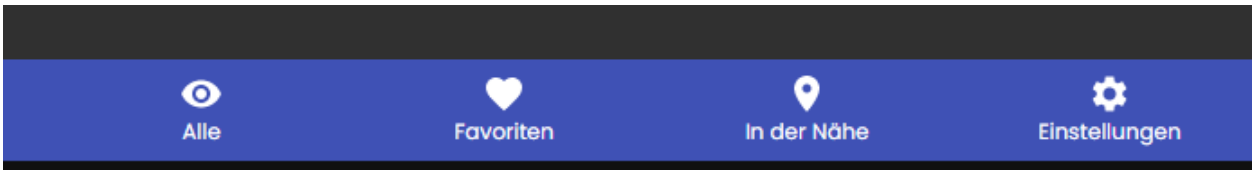
- **Frontend:** 6c603d3e
- **Backend:** a732acce

Um den Test durchzuführen wurde in beiden Repositories über Docker-Compose die Software gestartet.

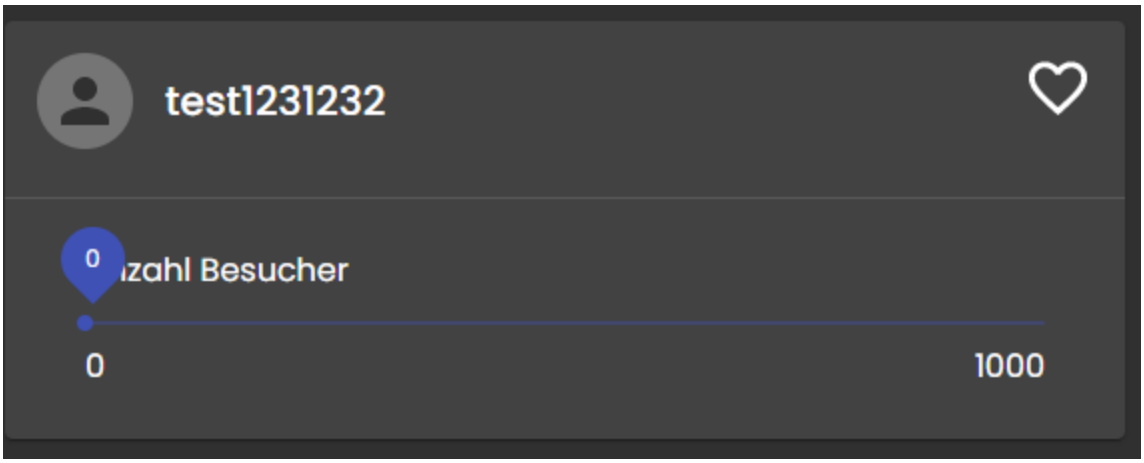
1.12.3 Protokoll

Anforderung	Implementiert	Kommentare	Status
AL-1-1	ja		erfüllt
AL-2-1	ja		erfüllt
AL-2-2	ja		erfüllt
AW-1-1	ja		erfüllt
AW-1-2	ja		erfüllt
AW-1-3	ja		erfüllt
AW-2-1	nein	Wurde nicht umgesetzt, weil Umfang zu gross	nicht getestet
AW-3-1	ja		erfüllt
AW-3-2	ja		erfüllt
AW-3-3	ja		erfüllt

1.12.4 Manuelle Frontend Tests



In der Navigation ist nicht ersichtlich, wo man sich aktuell befindet.

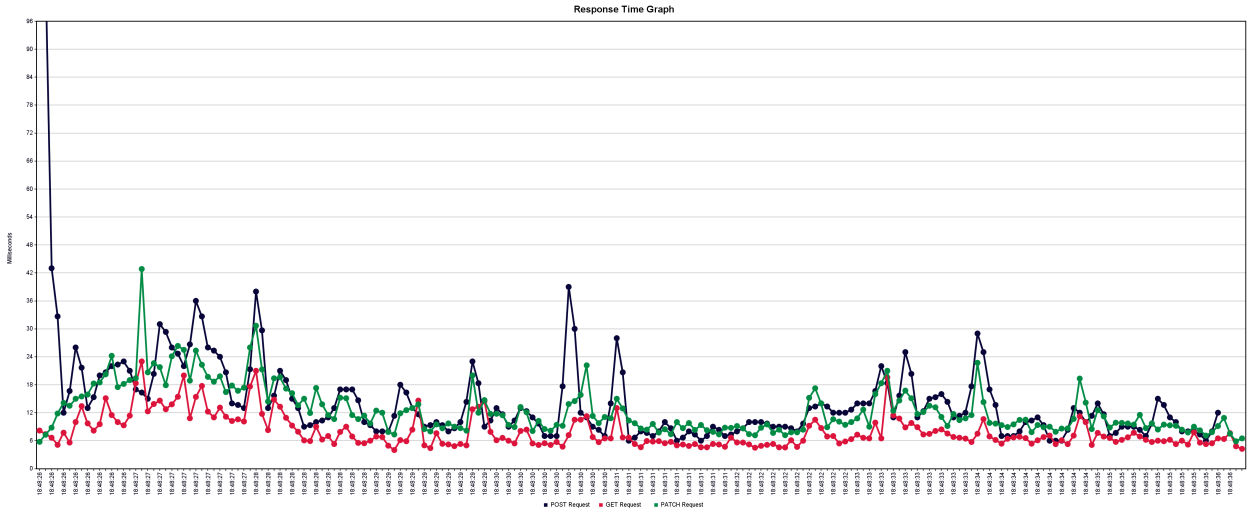


Wenn die Anzahl aktueller Besucher zu tief ist, überlappt der Picker den Titel.

1.12.5 Test der NF-Anforderungen, Performance- und Lasttests

Soweit möglich und sinnvoll wurden die nicht funktionalen Anforderungen getestet. Das Backend wurde mit JMeter getestet. Die Ergebnisse sind mit Vorsicht zu betrachten, da nicht auf einem isolierten System getestet wurde. Trotzdem kann man sagen, dass wir fürs erste nicht mit Performanceproblemen rechnen müssen.

Anforderung	Getestet	Kommentar	Erfüllt
NF-1	nein	Bis jetzt haben wir noch keine Benutzerumfragen gemacht.	nicht getestet
NF-2	ja	Komplettes Laden der Seite mit 50 Stores 1.15s	ja
NF-3	ja	Lasttest mit 100 Benutzern. Antwortzeiten immer unter 50ms.	ja
NF-4	ja	Reaktionszeit der Seite mit 50 Stores war schnell	ja
NF-5	nein	Aktuell werden keine DSG relevanten Daten verwaltet.	nicht getestet
NF-6	ja	Gefahren wurden im Threat Model neu evaluiert.	95%
NF-7	nein	Konnte nicht getestet werden	nicht getestet



1.12.6 Verbesserungsmöglichkeiten

Der Umfang wurde unterschätzt und aufgrund von sehr vielen Problemen mit der CI/CD Pipeline, wurde auf eine Funktionalität verzichtet. Mit einer detaillierteren Planung wollen wir dieses Problem im nächsten Sprint vermeiden.

Bekannte Einschränkungen

Eine MUSS-Anforderung, die Suche, wurde aus Zeitgründen erst im Backend umgesetzt und wird im nächsten Sprint im Frontend implementiert.

1.13 Systemtest-Protokoll 20. Mai 2021

1.13.1 Einführung

Zweck

Dieses Dokument dient dazu, eine Übersicht zu geben welche Systemtests wann durchgeführt wurden und was das Resultat war.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

Referenzen

Systemtestspezifikation

1.13.2 Angaben zur Durchführung

Testdatum: 2021-05-20

Getestet wurde auf folgendem Stand der jeweiligen Git Repositories.

- **Frontend:** 358cafa9
- **Backend:** 6d39dd0c

Um den Test durchzuführen wurde in beiden Repositories über Docker-Compose die Software gestartet.

1.13.3 Protokoll

Anforderung	Implementiert	Kommentare	Status
AL-1-1	ja		erfüllt
AL-2-1	ja		erfüllt
AL-2-2	ja		erfüllt
AW-1-1	ja		erfüllt
AW-1-2	ja		erfüllt
AW-1-3	ja		erfüllt
AW-2-1	nein		erfüllt
AW-3-1	ja		erfüllt
AW-3-2	ja		erfüllt
AW-3-3	ja		erfüllt

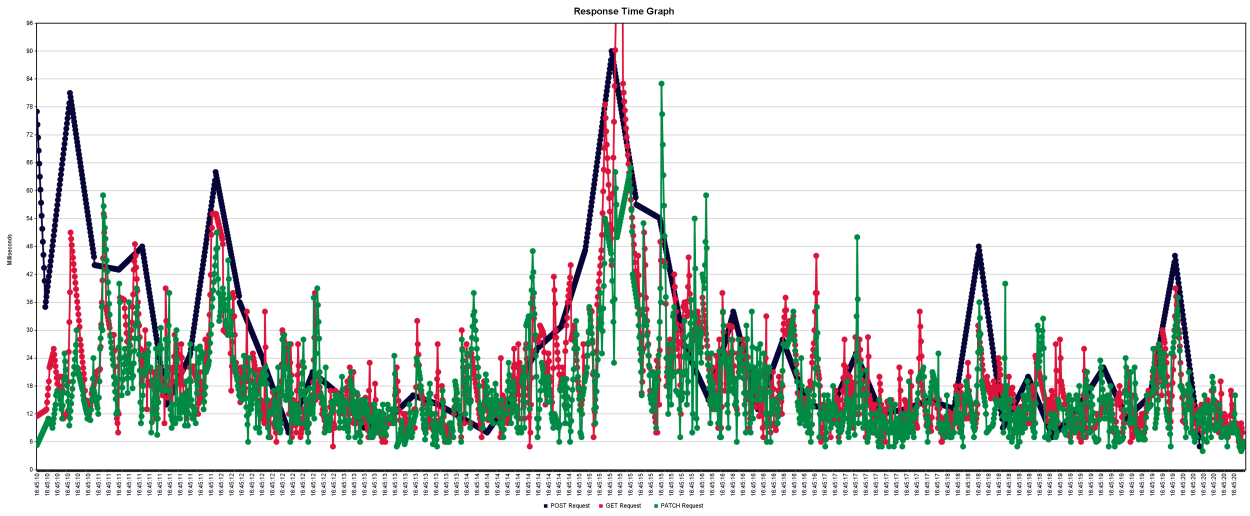
1.13.4 Manuelle Frontend Tests

Es wurden keine Fehler entdeckt. Alles funktioniert mit diesem Funktionsumfang einwandfrei.

1.13.5 Test der NF-Anforderungen, Performance- und Lasttests

Soweit möglich und sinnvoll wurden die nicht funktionalen Anforderungen getestet. Das Backend wurde mit JMeter getestet. Die Ergebnisse sind mit Vorsicht zu betrachten, da nicht auf einem isolierten System getestet wurde. Trotzdem kann man sagen, dass wir fürs Erste nicht mit Performanceproblemen rechnen müssen.

Anforderung	Getestet	Kommentar	Erfüllt
NF-1	nein	Bis jetzt haben wir noch keine Benutzerumfragen gemacht.	nicht getestet
NF-2	ja	Komplettes Laden der Seite mit 50 Stores 1.15s	ja
NF-3	ja	Lasttest mit 100 Benutzern. Antwortzeiten immer unter 50ms.	ja
NF-4	ja	Reaktionszeit der Seite mit 50 Stores war schnell	ja
NF-5	nein	Aktuell werden keine DSG relevanten Daten verwaltet.	nicht getestet
NF-6	ja	Gefahren wurden im Threat Model neu evaluiert.	95%
NF-7	nein	Konnte nicht getestet werden	nicht getestet



1.13.6 Usability Tests

Es wurden Usability Tests mit zwei Personen durchgeführt. Dabei lag der Fokus auf der Benutzerführung und Anwenderfreundlichkeit. Dabei wurden folgende Punkte festgestellt/bemängelt:

- Die Testpersonen finden die angezeigten Informationen übersichtlich dargestellt und finden sich im Frontend sehr schnell zurecht.
- Der Slider welcher das aktuelle Besucheraufkommen anzeigt wird vom User als Eingabeelement wahrgenommen. Als Alternative wäre hier eine Progressbar geeignet.
- Die nicht implementierten Navigationspunkte („In der Nähe“ und „Einstellungen“) geben dem User kein Feedback. Hier wäre es ratsam eine Infopage anzuzeigen oder mit einem Popup darauf hinzuweisen, dass diese Funktionen noch nicht zur Verfügung stehen.
- Auf der Desktopversion sind die Navigationspunkte am unteren Rand des Browsers, diese werden vom Benutzer somit nicht sofort erkannt. Die App wurde bewusst nach dem Prinzip Mobile-first entwickelt, dies sollte aber bei einer Optimierung für Desktop beachtet werden.
- Ebenfalls in der Desktopversion werden die Cards unterschiedlich gross dargestellt, sofern eine Card einen viel längeren Namen hat. Dies sollte bei einer Optimierung für Desktop ebenfalls beachtet werden.

1.14 Schlussbericht

1.14.1 Einführung

Zweck

Dieses Dokument fasst unsere Erfahrungen und Ergebnisse des Projektes CapWatch zusammen.

Gültigkeitsbereich

Dieses Dokument ist gültig für das Engineering Projekt CapWatch, welches im Frühlingssemester 2021 an der Fachhochschule OST Rapperswil-Jona durchgeführt wurde. Es ist für die Betreuer und Entwickler des Projekts ausgelegt.

1.14.2 Zielerreichung

Durch das Erreichen aller Minimalanforderungen, konnten wir das Projekt erfolgreich abschliessen. Da unser Fokus auf Qualität gerichtet war, lag es nicht mehr in der Zeit, optionale Anforderungen vollständig umzusetzen. Sofern man eine Geschäftsstelle findet, welche Daten anliefert, kann man die Informationen über CapWatch abfragen. Das Design sieht sehr ansprechend aus, sowohl auf Mobile, wie auch auf Desktopsystemen. Durch die gute Softwarearchitektur und vielen qualitätssichernden Massnahmen erreichten wir eine qualitativ sehr hochwertige Software.

1.14.3 Allgemeiner Erfahrungsbericht

Projektmanagement

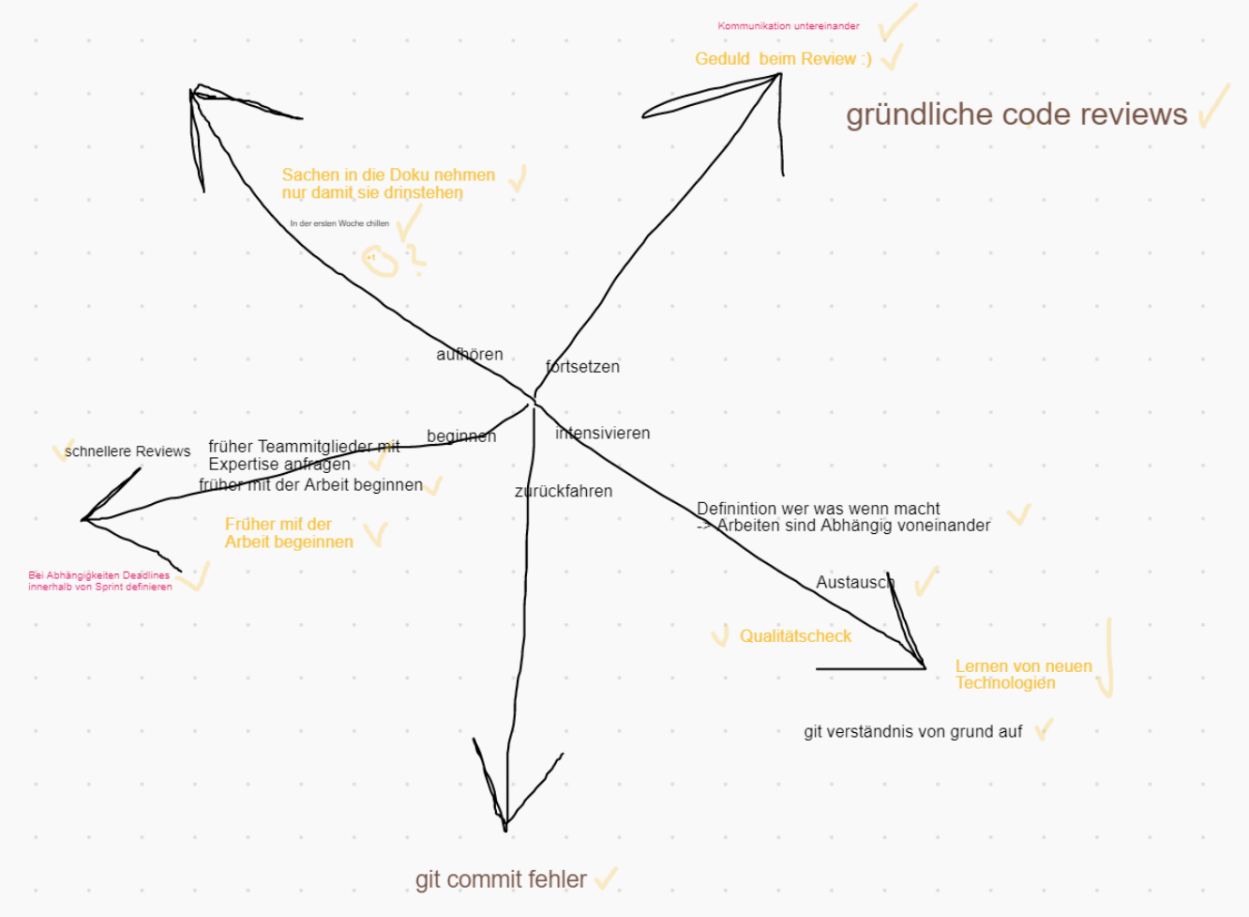
Die Verwendung von SCRUM hat uns im Entwicklungsprozess ideal unterstützt. Unsere Schätzgenauigkeit war bei unseren zweiwöchigen Sprints, schon von Beginn weg, erstaunlich genau. Wir haben jeweils pro Sprint ein Backlog Refinement, Sprint Planning, Sprint Review und als Abschluss eine Sprint Retrospektive durchgeführt. Auf die Daily Standup Meetings haben wir verzichtet, weil es für uns nicht möglich war mit unserem Teilzeitpensum einen täglichen Termin einzuplanen.Die Abhängigkeiten von einzelnen Tickets wurden von uns oft zu spät erkannt. Deshalb führten wir im Projektverlauf ein zusätzliches Meeting ein, welches wir jeweils am ersten Donnerstag des Sprints durchführten. Bei den Retrospektiven haben wir verschiedene Formen ausprobiert. Diese haben uns geholfen die Problem in der Zusammenarbeit und dem Projektfortschritt früh anzusprechen und so jeden Sprint besser zu werden. Da wir uns nicht physisch treffen konnten, haben wir uns auf einer virtuellen Kollaborationsplattform ausgetauscht. Nachfolgend zwei der verschiedenen Formen der Retrospektiven, die wir ausprobiert haben:

Seestern

Zu Beginn der Retrospektive zeichnet der Scrum Master einen „Seestern“ an das Whiteboard oder auf den Flipchart.

Die Mitglieder des Scrum-Teams notieren auf Post-its eine Antwort zu jeder der folgenden Fragen:

- Was müssen wir fortsetzen?
- Was müssen wir intensivieren?
- Was müssen wir zurückfahren?
- Womit müssen wir beginnen?
- Womit müssen wir aufhören?



K.A.L.M.

K.A.L.M. steht für:

- Keep – Etwas, das vom Team gut gemacht wird und fortgesetzt werden muss
- Add – Eine neue Idee, die der Verbesserung dient
- Less – Etwas, das gegenwärtig gemacht wird, jedoch nicht zum Erreichen des Ziels beiträgt
- More – Etwas, das gegenwärtig gemacht wird und zum Erreichen des Ziels beitragen kann, wenn es intensiviert wird.

Wir hatten die Projektrollen Scrum Master und Product Owner. Unser Scrum Master hat ebenfalls die Rolle des Projektleiters wahrgenommen. Diese zentrale Rolle war sehr wichtig, da sie einerseits die zentrale Ansprechstelle für unser Team war und andererseits den Fortschritt des Sprints genau verfolgte und auf einzelne Teammitglieder zuging, wenn noch nicht so viel umgesetzt worden war. Zusätzlich moderierte er die Meetings und schaute, dass wir keine Zeit verschwenden. Dadurch konnten wir den Umfang des Sprints bis auf kleine Ausnahmen jeweils umsetzen.

Unsere teaminterne Dokumentation wurde in OneNote geführt. In den Meeting Protokollen wurden Entscheidungen und Besprechungsdetails festgehalten. Zusätzlich sind hier unsere Definition of Done, ausführlichen Richtlinien und Links strukturiert abgelegt.

Wir verwendeten die Kommunikationskanäle WhatsApp und Microsoft Teams. Ankündigungen und Informationen wurden über unseren WhatsApp Chat kommuniziert. Für Meetings und bilaterale Besprechungen trafen wir uns auf Teams. Zu Beginn hatte der Informationsfluss zwischen den Teammitgliedern noch Verbesserungspotential. Durch die Rückmeldungen aus den Retrospektiven konnten wir die Kommunikation jedoch fortlaufend verbessern.

Wir haben am Anfang GitLab als Ticket System und für die Zeiterfassung benutzt. Verschiedene Gründe, welche wir im Dokument Projektplan genauer beschrieben haben, bewogen uns zur Umstellung auf YouTrack. GitLab verwendeten wir nur noch für die Quellcode Verwaltung, Generierung der Dokumente und Container über die Pipeline. Die Verfügbarkeit von GitLab und der Pipeline liess sehr zu wünschen übrig und legte uns einige Steine in den Weg zum absoluten Projekterfolg. Dies war zeitweise sehr frustrierend, sodass wir auch hier analyisierten , die Plattform komplett zu wechseln.

Dokumentation

Die Dokumentationsvorlage war für uns zuerst irreführend. Wir hatten es so verstanden, dass man für alle aufgeführten Vorgaben etwas dokumentieren muss. Viele Vorgaben in der Vorlage brachten für die Grösse unseres Projektes keinen Mehrwert. Trotzdem dokumentierten wir zu Beginn für alle Vorgaben. Diese Teile der Dokumentation wurden aber zu einem späteren Zeitpunkt wieder entfernt und dadurch wurde unsere Zeit nicht effizient genutzt.

Da Markdown-Tabellen nicht korrekt zu PDF konvertiert werden können, mussten alle Markdown-Tabellen in Restructured Text-Tabellen umgewandelt werden. Dies hat uns auch wieder viel Zeit gekostet und machte das Bearbeiten von Tabellen bemerkbar aufwendiger.

Die Checklisten sind nicht mehr marktorientiert bzw. zeitgemäss und brauchen eine Überarbeitung. Gerade das Definieren von fully dressed Use Cases ist für kleinere Projekte viel zu aufwendig und bringt keinen sichtlichen Mehrwert. In der Anleitung des EPJ Handbooks waren wichtige Details nicht korrekt, was uns mehrere Stunden an Aufwand beschert hat mit fruchtlosen Fehlersuchen, welche mit einer korrekten Anleitung nicht angefallen wären.

Es war vor allem zu Beginn schwierig, die Arbeit aufzuteilen, da alle an den gleichen zwei Dokumenten arbeiten mussten. Dieses Problem hat sich später gemindert, da wir mehr unterschiedliche Arbeiten hatten.

Entwicklung

Der Einsatz von Lintern, Formattern und Coding Conventions hat sich sehr bewährt und konnte eine konstant hohe Codequalität sicherstellen. Von den Werkzeugen, die Git zur Verfügung stellt, haben wir regen Gebrauch gemacht. Wir haben mit Git Flow, Branching, Conventional Commits und anschliessenden Merge Requests eine gute Struktur erstellt, in welcher Code Reviews eine zentrale Rolle spielten. Im Frontend wurden sogar die Commit Messages geprüft und Commits nur bei keinen Errors vom Linting erlaubt.Allgemein lief die Entwicklung im Backend relativ flüssig, da schon viel Wissen vorhanden war. Mit MongoDB und C# gab es zu Beginn ein paar Hürden.Die SonarQube Konfiguration war mit .Net um einiges komplizierter als mit Node.js. Im Frontend war am Anfang noch nicht so viel Wissen vorhanden. Dadurch musste vieles im Pairprogramming entwickelt werden und weitere Zeit ausserhalb des Projektes für den Wissensaufbau aufgewendet werden. Dies hat einige Issues weit über die geschätzte Zeit gebracht. Dies wurde im Verlauf des Projektes jedoch auch besser, nachdem wir uns mehr Wissen und Erfahrung angeeignet hatten.

DevOps

Der Einsatz der GitLab Pipelines für CI/CD hat sich gut bewährt und die Produktivität des Teams spürbar erhöht, da sich die einzelnen Teammitglieder ganz auf das entwickeln von Software konzentrieren konnten und sich nicht um das bauen und integrieren des neuen Codes sorgen mussten. Der Einsatz von SonarQube in der Pipeline hat uns ausserdem stets mit aktuellen Codequalitäts-Metriken versorgt und die Qualität des Codes insgesamt nochmals weiter verbessert.Im Backend wurden die geschriebenen Unit Tests durch die Pipelinies ausgeführt und ein Merge eines Feature-Branches in den Develop-Branch konnte nur vorgenommen werden, wenn die Unit Tests erfolgreich ausgeführt werden konnten und die SonarQube Metriken erfüllt wurden.

Es gab aber auch Probleme im DevOps Bereich, vor allem mit der Infrastruktur (siehe unten) und dem CI/CD Teil. Das Thema Continuous Deployment konnten wir am Ende leider gar nicht mehr umsetzen, da die Infrastruktur in der letzten Projektphase sehr häufig nicht erreichbar und generell sehr langsam war. Da die OST IT uns ausserdem von den GitLab Runners her keinen Zugriff via SSH (Port 22) auf unseren Deploymentserver geben wollte, hätten wir aber ohnehin kein vollständiges CD implementieren können in hoher Qualität.

Pipeline und Infrastruktur

Die von der OST zur Verfügung gestellte Infrastruktur war oft von Problemen und Ausfällen betroffen. Dies war vor allem in den letzten Wochen des Projekts immer häufiger ein Problem. Diese Schwierigkeiten haben zum Teil starke Verzögerungen verursacht und am Ende haben sie dann auch dazu geführt, dass wir das Deployment auf unseren Server nicht korrekt fertigstellen konnten. Wir mussten schliesslich ein manuelles lokales Deployment vornehmen, um das Projektdeployment demonstrieren zu können.

Die Pipeline selber war generell langsam und eher schwerfällig. Vermutlich lag das daran, dass allen Projekten zusammen nur ein einzelner GitLab Runner zugeteilt war. Leider konnten wir das nicht weiter beeinflussen. Es hat aber dazu geführt, dass wir bei der Arbeit oft unnötig ausgebremst wurden, weil lange Wartezeiten entstanden.

1.14.4 Persönliche Erfahrungen

Pascal Schlumpf

Für mich war das Projekt sehr lehrreich, da ich das erste Mal ein Projekt von Anfang bis Ende durchgeführt habe. Auch kam ich in Kontakt mit neuen Technologien, Methoden und Arbeitsweisen, die ich so in meinem Arbeitsalltag noch nie verwendet hatte. Ich brauchte am Anfang eine Weile, bis ich die MongoDB in C# integriert hatte. Als dies dann funktionierte, setzte ich auch andere Bereiche im Backend um. Am Ende konnte ich auch noch ein paar Aufträge im Frontend umsetzen, so dass ich alles einmal gesehen habe. Meine Rolle als Product Owner war für mich zu Beginn sehr ungewohnt und ich hatte auch nicht wirklich eine Wahl, was das Bestimmen der umzusetzenden Tasks anging. Es

war auch schwierig alle nötigen Issues zu erfassen, da nicht alle Anforderungen über die Checkliste und die Vorlage abgedeckt waren. Für mich als Berufsbegleitender Student war die zusätzliche Belastung durch das EPJ Projekt sehr gross, so dass ich für viele Module nur das Nötigste tun konnte. Nichts desto trotz verlief das Projekt erfolgreich und es war eine grosse Teamleistung.

Christoph Scheiwiller

Das EPJ war für mich eine super Erfahrung. Da das Projekt auf einer grünen Wiese startete, konnten wir interessante Technologien und moderne Tools einsetzen, welche uns ideal im Entwicklungsprozess unterstützten. Da ich in meiner täglichen Arbeit SVN verwende, kämpfte ich am Anfang stark mit der Verwendung von GitLab, GitFlow und Conventional Commits. Auch wenn ich diese Tools und Prozesse noch nicht vollständig beherrsche sehe ich viele Vorteile und möchte Git weiterhin verwenden. Gefordert war ich bei der Entwicklung des Frontends, da ich React noch nicht kannte. Das React Framework ist sehr umfangreich und fordert deshalb eine längere Einarbeitungszeit. Im Team hatten wir eine gute Zusammenarbeit, damit konnten wir jede Herausforderung gemeinsam meistern. Dabei halfen uns auch unsere Berufserfahrung und Interessen in verschiedenen Fachbereichen, mit welchen wir uns ideal ergänzten. Als Verantwortlicher für die Code Qualität war ich nicht sehr gefordert, da sich alle Teammitglieder an die vereinbarten Richtlinien hielten. Viele Unschönheiten wurden bereits durch die eingesetzten Tools und den Reviewprozess erkannt und behoben. Insgesamt ziehe ich für mich ein positives Feedback und kann viele Erfahrungen für die Studienarbeit und die Bachelor-Arbeit mitnehmen.

Rafael Fuhrer

Das Engineering Projekt war für mich eine besondere Herausforderung. Ich habe bis zu diesem Projekt, trotz Berufstätigkeit neben dem Studium, nie an einem richtigen Softwareprojekt mitgearbeitet. Entsprechend gab es hier für mich viel zu lernen. Vor allem was die Themen Git, Git flow, Clean Code, Best Practices und Debugging angeht, habe ich in den letzten 3 Monaten sehr viel dazu gelernt. Ich durfte aber auch Aufgaben übernehmen, in denen ich um einiges routinierter bin, wie z.B. das Schreiben von Docker & Docker-Compose Files und das Setup der CI-Pipelines. Leider waren diese Tasks in diesem Projekt mit vielen Schwierigkeiten verbunden. Dies lag vor allem an der unzuverlässigen Infrastruktur und den Vorlagen, die uns die OST bereitstellte. Aber auch der Einsatz von C# im Zusammenhang mit Open Source Tools wie SonarQube und Renovate war am Ende eine grössere Herausforderung als ich ursprünglich gedacht hätte. Das Projekt hat mir aber trotzdem meistens viel Spass gemacht, da ich all diese interessanten neuen Dinge gelernt habe, die ich später sicher einmal gebrauchen kann. Alles in allem bin ich sehr froh, dass wir trotz all dieser Probleme unsere Projektziele erreicht und das Projekt erfolgreich abgeschlossen haben.

Jonas Hauser

Ich habe mich persönlich sehr gefreut, dass ich bei der ersten grösseren Gruppenarbeit an der OST direkt die Funktion als Projektleiter und Scrum Master übernehmen durfte. Meine Passion ist es, Projekte zu organisieren und zu leiten und darum habe ich sehr viel Engagement und Erfahrungen in das Engineering Projekt gesteckt. Dies hat sich soweit wir als Team das beurteilen können, sehr gelohnt. Jeder Sprint ist nahezu problemlos von statten gegangen und die Resultate sind allesamt auf sehr hohem Niveau und im Vergleich zu Projekten auf dem Markt wahrscheinlich überdurchschnittlich. Dennoch gab es einige Schwierigkeiten zu meistern. Mitunter einer der grössten Dorne im Auge war der stetige Zeitdruck, ausgelöst durch die sehr kurze Zeitspanne, in der am Projekt gearbeitet werden sollte. Viel Zeit wurde für die Dokumentation, dem allgemeinen Projektmanagement und der mangelnden Infrastruktur der OST aufgewendet, wodurch ich immer versucht habe, quantitativ voranzukommen, aber dennoch in möglichst hoher Qualität. Mir hat das Projekt viel Freude bereitet und ich habe persönlich viele praktische Erfahrungen dabei sammeln können.

Pascal Schneider

Das Engineering Projekt war eine interessante Erfahrung. Neben Arbeit, restlichem Studium und sonstigen Verpflichtungen war es aber zeitweise auch sehr anstrengend. Die Zusammenarbeit im Team hat zwischenmenschlich sehr gut funktioniert und die Sitzungen waren immer sehr positiv und motivierend. Einen negativen Nachgeschmack hinterlassen hat hingegen die Infrastruktur der OST, welche uns des Öfteren Probleme bereitet hat. Die Unzulänglichkeiten von GitLab bezüglich Issue Tracking und Zeiterfassung konnten wir glücklicherweise früh mit YouTrack umgehen, die mangelnde Qualität / Anzahl GitLab Runner welche vor allem gegen Ende des Projekts bemerkbar wurde kostete hingegen sehr viel Zeit und Nerven. Im Backend war für mich vor allem das Implementieren der hexagonalen Architektur spannend, auch wenn diese für unseren Projektumfang etwas übertrieben ist. Auf der negativen Seite ist das Zusammenspiel zwischen .NET und den open-source Werkzeugen, welche wir eingesetzt haben, aufgefallen. Da wir auf der Arbeit Team Foundation Server im Einsatz haben war mir nicht bewusst wie viele Stolpersteine hier existieren und wie oft .NET anders integriert werden muss als andere Technologien. Für zukünftige Projekte nehme ich definitive mit, dass ich bei der Auswahl von Tools besser recherchieren muss, wie gut sich diese mit .NET integrieren lassen. Alles in Allem war das EPJ ein lehrreiches Projekt mit positiven und negativen Höhepunkten.

1.15 Eigenständigkeitserklärung

Erklärung zum Engineering Projekt bezüglich Eigenständigkeit der Arbeit:

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit erwähnt ist,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben und
- dass die Urheberschaft aller Arbeitsergebnisse korrekt angegeben ist.

Ort, Datum: Name, Unterschrift:

Bronschhofen, 23. Mai 2021Christoph Scheiwiller

Winterthur, 23. Mai 2021Pascal Schneider

Schwerzenbach, 23. Mai 2021Rafael Fuhrer

Rapperswil, 24. Mai 2021Pascal Schlumpf

Rorschach, 25. Mai 2021Jonas Hauser

1.16 Zeitauswertung

Letztes Update: 25. Mai 2021, 21:15

1.16.1 Total

- **Total geschätzt**: 401h 30m*
- **Total gearbeitet**: 560h 40m

1.16.2 Pro Projektteilnehmer

Projektteilnehmer	Zeit
Christoph Scheiwiller	111h 10m
Jonas Hauser	118h 00m
Pascal Schlumpf	103h 50m
Pascal Schneider	113h 55m
Rafael Fuhrer	113h 45m

1.16.3 Aufgewendete Zeit

Zeiterfassung auf YouTrack

YouTrack Zeitauswertung Export

Um die Arbeitszeit jederzeit nachweisen zu können, werden am Ende jedes Meilensteins die Auswertungen `Time Report per Milestone`, `Time Report`, `Time Report per User` und `Arbeitszeittabelle` erstellt und in der Dokumentation abgelegt. Die Zeitauflösung in den Auswertungen ist in Minuten angegeben.

1.16.4 Anmerkungen zur Zeitauswertung

*Aufgrund der Umstellung auf YouTrack in der dritten Projektwoche, sind im Time Report bis zu diesem Zeitpunkt keine Schätzungen vorhanden, jedoch wurde der Aufwand in YouTrack nachgeführt. Aus diesem Grund ist die geschätzte Zeit tiefer. Eine grobe Übersicht dazu liefert die Zeitplanung mit RUP im Dokument Projektplan.

KAPITEL 2

Glossar

Anbieter Unternehmen welche Daten an CapWatch liefern.

BDUF Big Design Up Front

C4-Modell Software Architekturmodell: Context, Containers, Components und Code

Capacity Aktuelle oder maximale Personenanzahl in einem Store.

CapWatch Produktname und Abkürzung für Capacity Watcher.

DRY Don't Repeat Yourself

DoD Definition of Done

EPJ Abkürzung für Engineering Projekt.

KISS Keep It Simple, Stupid

Konsument Benutzer der Applikation CapWatch und Kunde der in CapWatch registrierten Unternehmen.

MVP Minimal Viable Product

OWASP Open Web Application Security Project

Secret Dient zur Identifikation der Registrierten Unternehmen.

Store Fachlich: Ein Geschäftsstelle, über welche wir mit Daten beliefert werden. Technisch: Objekt welches eine Geschäftsstelle repräsentiert.

S-O-L-I-D S - Single-responsibility Principle O - Open-closed Principle L - Liskov Substitution Principle I - Interface Segregation Principle D - Dependency Inversion Principle

YAGNI You Aren't Gonna Need It

KAPITEL 3

Review Meilensteine

Dieses Dokument enthält eine Auflistung der Meilensteine mit den Links zu den zugehörigen Dokumenten und Checklisten.

3.1 M1: Review Projektplanung

- Dokumente
  - *Projektplan*
  - *Risikoanalyse*
  - Zeitauswertung

3.2 M2: Review Requirements Analyse

- Dokumente
  - *Anforderungsspezifikation*
  - *Domainanalyse*
  - Zeitauswertung

3.3 M3: Review End of Elaboration

- Dokumente
  - *API-Dokumentation*
  - Zeitauswertung

3.4 M4: Review Architekturdesign

- Dokumente
  - *Softwarearchitektur*
  - Zeitauswertung

3.5 M5: Review Qualitätsmassnahmen

- Dokumente
  - *Qualitätssicherung*
  - *Systemtestspezifikation*
  - Systemtestprotokoll
  - Zeitauswertung

3.6 M6: Beta Version

- Dokumente
  - *Schlussbericht*
  - *Eigenständigkeitserklärung*
  - Zeitauswertung

3.7 M7.1: Schlussabgabe

Siehe Moodle

3.8 M7.2: Schlusspräsentation

Siehe Moodle

Stichwortverzeichnis