BACHELORARBEIT INFORMATIK

FREIE UNIVERSITÄT BERLIN

Für die Stromnetz Berlin GmbH

Testkonzept und Deployment-Optimierung für ein Webportal, in einem agilen Projektteam.

Christoph Cornelius

Gutachter:	Univ.-Prof. Dr. Lutz Prechelt
Zweitgutachter:	Prof. Dr. Jörn Eichler
Verfasser:	Christoph Cornelius
Matrikel-Nr.:	5344830
E-Mail:	christoc97@zedat.fu-berlin.de
Abgabetermin:	15. August 2021

Abstract

Um zukunftsfähig zu sein, entwickelt die Stromnetz Berlin eine Digitalisierungsstrategie, in diesem Rahmen wird ein neues Kundenportal umgesetzt.

Die Arbeit befasst sich mit der Konzeptionierung und Umsetzung eines Testkonzepts sowie einer technischen Optimierung der Deployment-Abläufe zu dem Projekt Kundenportal. Das Projekt wird von einem agilen, nach SCRUM arbeitenden Projektteam umgesetzt. Als Entwicklungsumgebung wird Azure DevOps genutzt.

Das Testkonzept soll dem agil arbeitenden Projektteam die Möglichkeit geben, den DevOps-Ansatz im Projekt bestmöglich umzusetzen. Im Rahmen der Arbeit wird ein Testkonzept entwickelt und exemplarisch umgesetzt.

Die Deployment-Optimierung wird für das als backend genutzte CRM-System vorgenommen, um die Bereitstellungsvorgänge schneller und sicherer zu ermöglichen. Die technische Optimierung wird durch eine Schnittstelle erreicht, deren Implementierung auch Teil dieser Arbeit ist.

Inhaltsverzeichnis

1 Einleitung

Die Stromnetz Berlin GmbH betreibt das Strom-Verteilungsnetz in Berlin und ist für Netzanschlüsse/-entwicklung und deren Abrechnung zuständig.

Um zukunftsfähig zu sein, entwickelt die Stromnetz Berlin GmbH eine Digitalisierungsstrategie, in deren Rahmen auch ein neues Kundenportal umgesetzt wird. Das Werkzeug, um dies zu erreichen, besteht in der Digitalisierung des Anfrage- und Angebotsprozesses für Netzprodukte.

Das Kundenportal dient dazu, allen Berliner*innen, auch vertreten durch Dritte, die Möglichkeit zu geben digital Dienstleistungen wie Hausanschlüsse und Anmeldungen von Einspeiseanlagen anzufragen.

Neben der Erfassung der Kunden*innenanfragen, welche zum Anfangszeitpunkt bereits implementiert ist, soll auch der Angebots- und weitere Bestellprozess digital implementiert werden.

1.1 Problemumfeld

Die Grundidee ist es, ein bereits genutztes CRM[G] System, inklusive dessen Standardprozesse, als Backend-System für das Web-Portal zu nutzen. In diesem Projekt handelt es sich beim CRM-System um Aurea CRM, nachfolgend auch xRM[G] genannt. Der Vorteil, der sich daraus ergibt, liegt darin, dass betriebliche Workflows nahtlos am Portal abgebildet werden können. Bereits digitale Prozesse im Unternehmen müssen nicht von Grund auf neu erdacht werden.

Das Frontend sowie das Web-Backend sind in der Azure Cloud durch die Verwendung von Angular und SpringBoot entstanden, sie können als Standard-Bausteine einer Web-Anwendung betrachtet werden. Das Web-Backend kommuniziert mittels einer separaten API mit dem xRM, welche als .Net Schnittstelle auf den xRM-Servern vorhanden ist. Diese abstrahiert die relationalen Datenbestände im xRM und gibt sie über einen REST Endpunkt an das Web-Backend aus.

Die Umsetzung des Projekts erfolgt in einem agilen Projektteam nach der SCRUM-Methodik. Die Arbeitsorganisation erfolgt nach dem DevOps-Prinzip[G], unter Nutzung von Azure DevOps als Plattform.

1.2 Rahmenbedingungen im Unternehmen

Das Projekt „Kundenportal“ der Stromnetz Berlin GmbH startete im Jahr 2020 im Rahmen einer vom damaligen Mutterkonzern, der Vattenfall Holding, angeregten Digitalisierungsstrategie.

Im Dezember 2020 wurde bekannt, dass das Land Berlin o.g. GmbH kaufen wird, das Datum des Betriebsübergangs ist der 01.07.2021, und liegt somit im Zeitraum der Erstellung dieser Arbeit.

[G] Siehe Glossar

Damit einher geht ein Carve-Out Prozess der IT-Systeme, aus der Vattenfall Infrastruktur, hin zu unternehmenseigenen Servern. Dabei ist ein Testkonzept, neben den Vorteilen, die es bei der Weiterentwicklung mit sich bringt, ein Teil der rechtlichen Anforderungen des Betriebsübergangs. Auch eine Automatisierung der Tests ist von Vorteil, um während des Übergangs die Funktionsfähigkeit des Gesamtsystems immer wieder validieren zu können.

Das Land Berlin, als Käufer der Stromnetz Berlin GmbH, erwartet das in Entwicklung befindliche Kundenportal als Baustein für das Projekt „Masterplan Solarcity“ nutzen zu können. Aus diesem Grund muss das Portal während der gesamten Migration nutzbar sein.

1.3 Zielsetzung

Ziel der Arbeit ist es, für diese konkrete Architektur und Implementierung ein Test- und Deployment-Konzept zu Entwickeln.

Ziel des Testkonzepts ist es den Entwicklungsprozess zu beschleunigen. Fehler sollen möglichst automatisch gefunden und lokalisiert werden. Die Entwickler sollen schnelles Feedback bekommen, ohne zeitintensive händische Tests in hohem Maße Ausführen zu müssen. Neue Funktionen sollen schnell und sicher auf Funktionstüchtigkeit getestet werden können, bestehende Funktionen sollen auf funktionalen Erhalt geprüft werden. All dies soll erfolgen, ohne den Entwicklern einen unverhältnismäßigen Mehraufwand zuzumuten.

Mit der Deployment-Optimierung sollen die Abläufe zum Bereitstellen von neuen Software-Artefakten definiert und teilweise implementiert werden. Der Fokus liegt dabei auf dem xRM-System, welches viele technische und organisatorische Einschränkungen mit sich bringt. Die technischen Einschränkungen sollen durch eine im Rahmen dieser Arbeit entwickelte Lösung reduziert werden.

2 Aktueller Stand des Projekts

Der folgende Abschnitt umfasst den Stand des Kundenportals zum Start-Zeitpunkt dieser Arbeit.

Das Kundenportal dient dazu Netzprodukte zu bestellen oder anzumelden. Dabei handelt es sich um Neuanschlüsse, Änderungen bestehender Anschlüsse, oder das Anmelden von besonderen Verbrauchs- und/oder Erzeugungseinheiten an das städtische Stromnetz. Diese Vorgänge werden in Form von bestellbaren Produkten auf einer Web-Oberfläche zugänglich gemacht, und bieten angepasste Formulare zur Erfassung der Kunden- und Anlagendaten.

2.1 Software-Komponenten des Kundenportals

In folgender Grafik sind die wesentlichen Komponenten des Portals dargestellt.

Die oberen Komponentengruppe befindet sich in der Azure Cloud, und kann somit auch außerhalb des Intranets erreicht werden. Die Konfiguration dieser erfolgt über Azure Services.

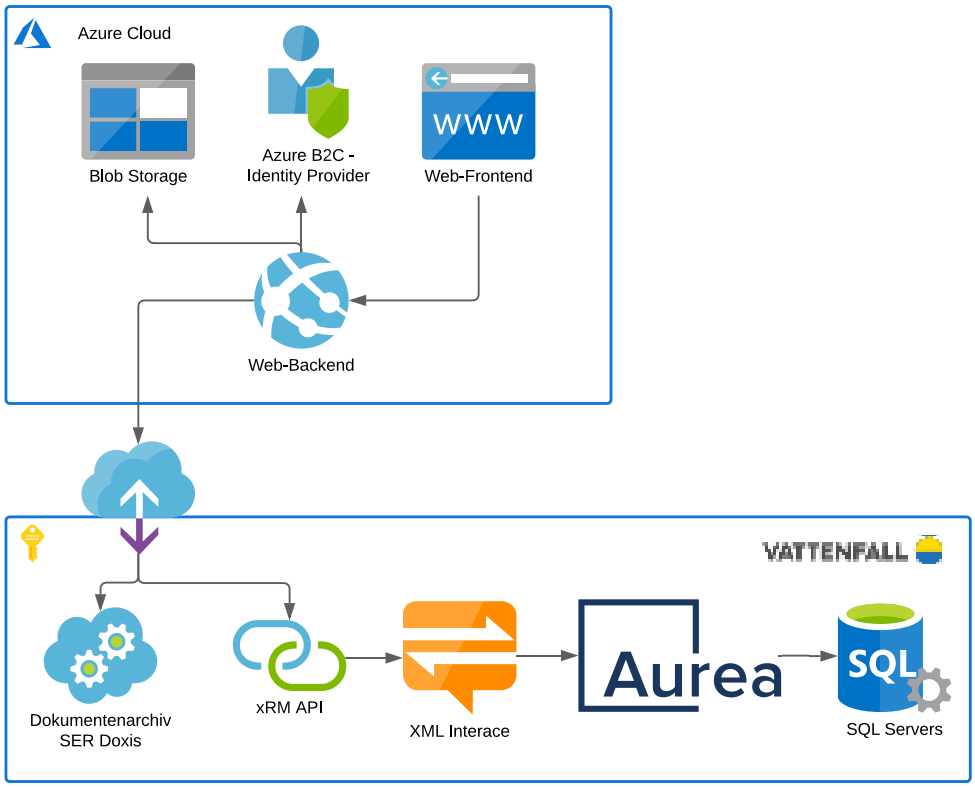


Abbildung 1 - Gesamtansicht der Komponenten des Kundenportals

Bei den Komponenten in der unteren Box handelt es sich um OnPremises[G] bereitgestellte Komponenten, diese sind nur im Intranet zugänglich. Dies ist von entscheidender Bedeutung, da ein Zugriff auf diese Komponenten nur über einen Web-Gateway (Wolken-Symbol), oder aus dem Intranet erfolgen kann.

2.1.1 Aurea CRM

Aurea CRM dient sowohl als Backend-System für das Kundenportal (WIN-Modul[G]), als auch als Frontend für die Sachbearbeiter*innen (WEB-Modul[G]).

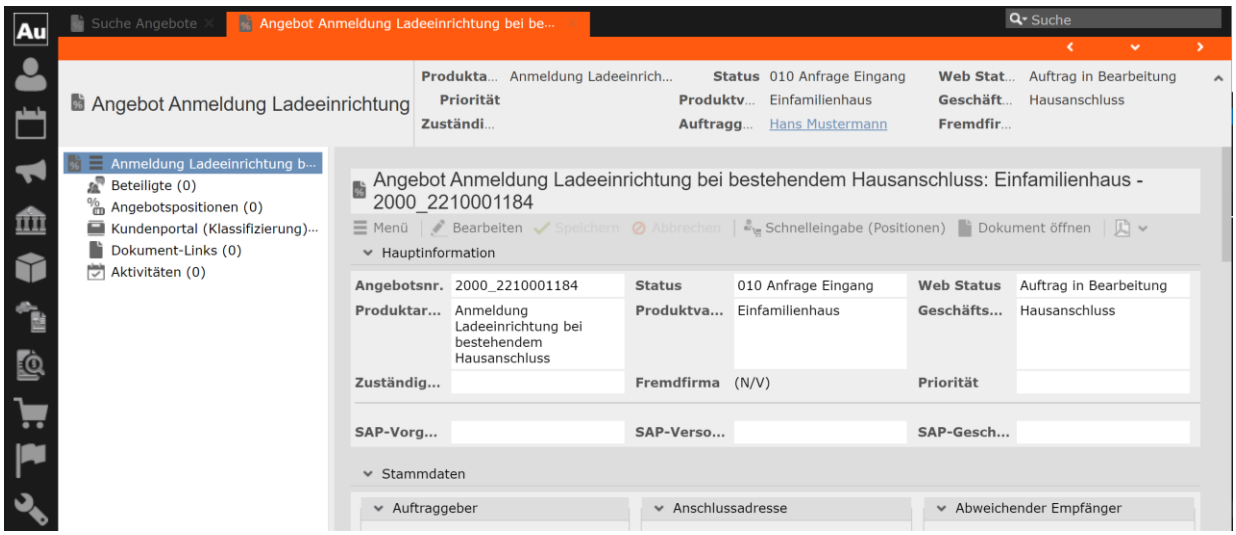


Abbildung 2 - WEB-Modul des xRM-Systems in der Angebotsansicht

Im CRM-System werden alle kundenbezogenen Daten gespeichert, mit Ausnahme von Dokumenten, für welche nur ein Link in ein Archivierungssystem angelegt wird.

Zusätzlich dient das CRM als Content Management System für das Kundenportal. Es werden Produkte und Klassifizierungen angelegt, welche ausgelesen werden können und als Web-Content im FE angezeigt werden. Diese Architektur ermöglicht es im CRM dynamisch neue Produkte hinzuzufügen, und bestehende zu ändern. Die Anpassung der Formulare im Web-Frontend kann und soll zukünftig durch die Fachbereiche selbst im xRM WEB-Modul erfolgen.

2.1.2 xRM-API

Die xRM-API greift über eine vom Hersteller bereitgestellte XML-Schnittstelle auf die im xRM-System liegenden Daten zu. Die XML-Schnittstelle bietet dabei ausschließlich eine Schnittstelle, um auf Datentabellen zuzugreifen. Die xRM-API bringt diese Daten in ein für das Kundenportal angepasstes Datenmodell.

Diese API ist OnPremises gehostet, und grundsätzlich außerhalb des Intranets nicht erreichbar. Der Zugriff des Web-Backends erfolgt über einen Web-Gateway.

2.1.3 Web-Backend

Das Web-Backend interagiert mit der xRM API, um Daten abzufragen oder anzulegen. Zusätzlich legt es Dokumente im Dokumentenarchiv ab.

Umgesetzt ist das Web-Backend als Springboot-API, welche in der Azure Cloud gehostet wird.

2.1.4 Web-Frontend

Das Web-Frontend ist der für den Endkunden sichtbare Bereich. Über das in der Grafik dargestellte Dashboard kann der Kunde ein Produkt auswählen und bestellen.

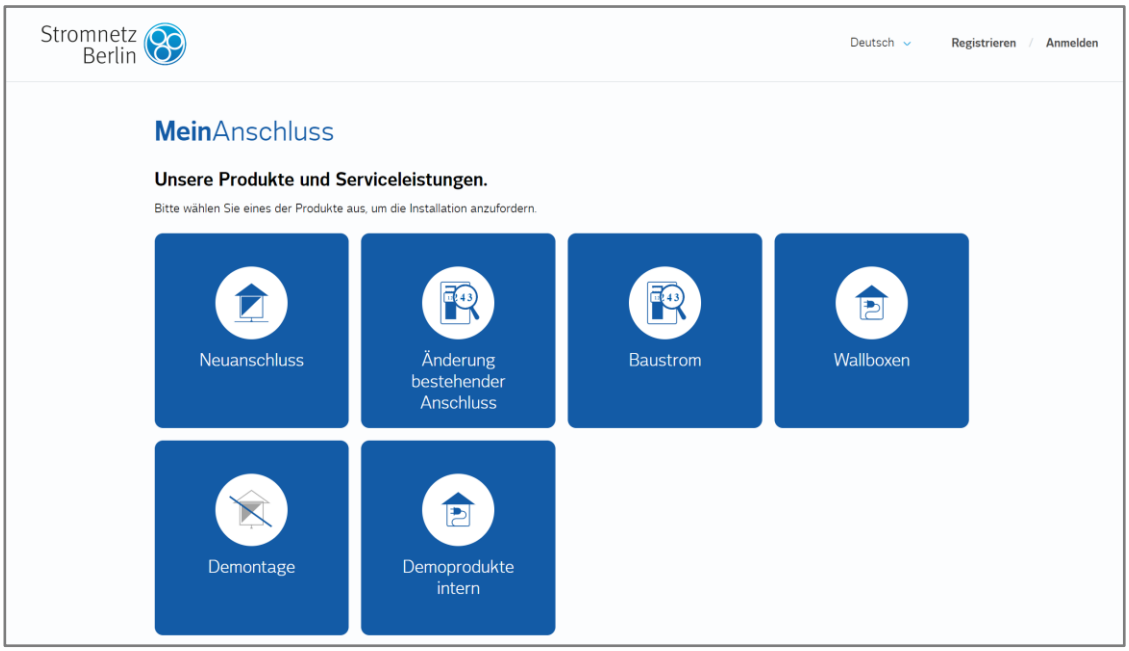


Abbildung 3 - Welcome Page des Kundenportals

Das Web-Frontend unterscheidet sich konzeptionell nicht von gängigen Implementierungen, und wird auch in der Azure Cloud gehostet. Daten bezieht das Web-Frontend vom Web-Backend.

3 Testkonzept

3.1 Ziele des Testkonzepts

Das Hauptziel des Testkonzepts ist es die Entwicklung des Kundenportals zu beschleunigen, ohne dabei Qualität und Zuverlässigkeit zu vernachlässigen. Zusätzlich ist für den Betriebsübergang der Stromnetz Berlin GmbH ein Testplan vertraglich vorgeschrieben. Der gesamte dafür nötige Testplan ist nicht Teil dieser Arbeit.

Die Beschleunigung der Entwicklung soll durch schnelles Test-Feedback an die Entwickler, sowie klar definierte Release-Abläufe erreicht werden.

Gleichzeitig soll im gesamten kein Mehraufwand für die Entwickler entstehen. Selbstverständlich stellt es im ersten Moment einen Mehraufwand dar, Tests zu implementieren. Langfristig reduzieren Sie aber den Zeitaufwand für exploratives Testen um ein Maß, welches den Zeitaufwand für die Testimplementierung übersteigt. Dieser zeitliche Vorteil wird dazu immer größer, je fortgeschrittener und größer das Projekt wird.

3.2 Rahmenbedingungen

3.2.1 Einordnung der Softwarekomponenten

Beim System Kundenportal handelt es sich um 4 vom Entwickler-Team implementierte- und/oder konfigurierte Softwarekomponenten[2]. Zusätzlich werden Schnittstellen zu Softwarekomponenten genutzt, welche von anderen Organisationseinheiten verwaltet werden.

Alle diese Komponenten lassen sich, aus Sicht der in diesem Projekt beschäftigten Entwickler, in drei Kategorien unterteilen. Diese sind in folgender Grafik dargestellt:

[2] Siehe 2.2.1 – 2.2.4

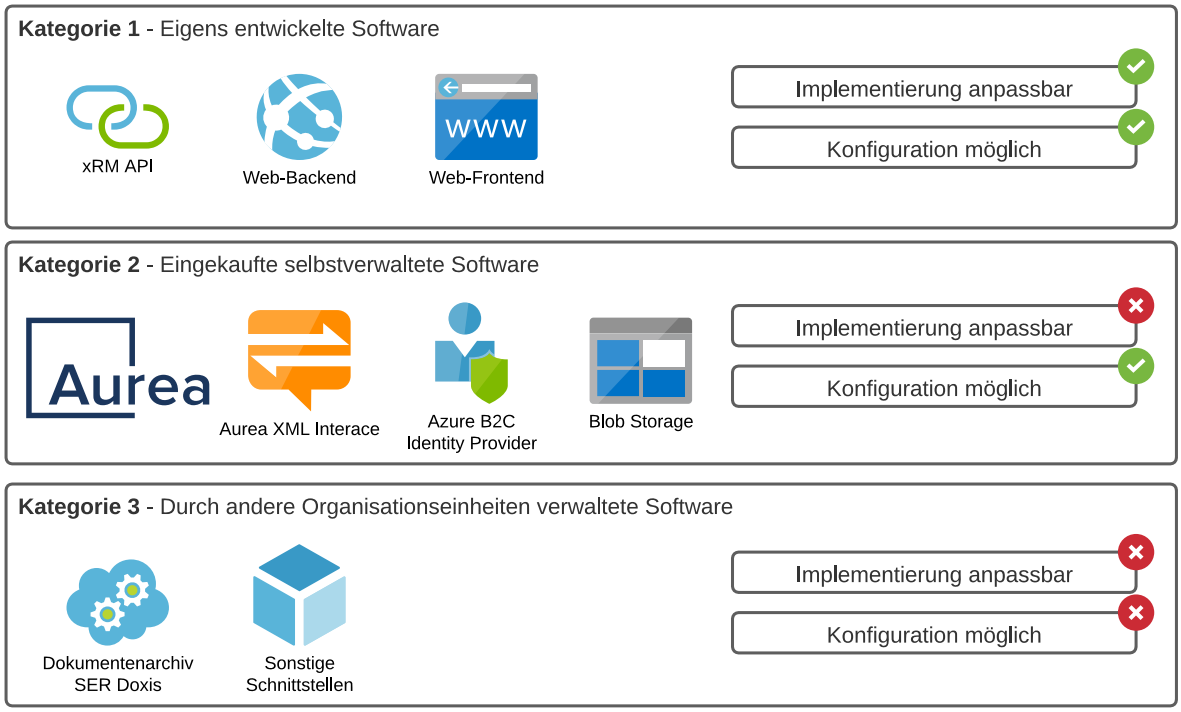


Abbildung 4 - Kategorisierung der Software-Komponenten aus Entwicklersicht

Für die 3 Software Komponenten der Kategorie 1 sollen Testpakete erstellt werden. Die Komponente Aurea CRM wird, obwohl Sie weiterentwickelt wird, nicht mit eigenem Testpaket getestet. Der Aufwand wäre unverhältnismäßig, da das System nur über aufwendige UI-Tests zu überprüfen ist. Alle Konfigurationsfehler, mit Einfluss auf das Projekt Kundenportal, würden zudem auch beim Testen der xRM-API auffallen.

Die übrigen Komponenten werden vom Entwicklerteam gar nicht weiterentwickelt, oder nur einmalig konfiguriert. Ein einmaliges, exploratives Testen dieser Komponenten kann daher als ausreichend angesehen werden. Die Konzeptionierung dieser Tests ist nicht Teil dieser Arbeit.

3.3 Entwurf des Konzepts

3.3.1 Ansatz

Im Folgenden werden einige, selbst gewählte, Ansätze vorgestellt und kurz begründet. Diese dienen als Grundlage für die Ausgestaltung des Testkonzepts.

Aufwandsersparnis durch Integrationstesten

Das Portal befindet sich aktuell in der ständigen Weiterentwicklung. Demzufolge müssen sich auch die Tests ständig an neue oder geänderte Implementierungen anpassen, da jede Komponente ständiger Veränderung unterliegt.

Um den Aufwand für die Wartung der Tests gering zu halten, wird auf das Testen von einzelnen, freigeschnittenen, Komponenten abgesehen. Diese Tests setzen ein Mock-Up der abhängigen Komponenten voraus, welche sich wie die Komponenten selbst, ständig verändern. Der Umstand, dass auf jeder Stage eine komplette Umgebung mit allen Schnittstellen existiert, begünstigt zusätzlich das Integrationstesten.

Einfache Tests zuerst implementieren

Bei API-Tests variiert der Implementierungs-Aufwand verschiedener Test-Cases deutlich. Während bei einer einfachen GET-Methode oft keine oder wenige Test-Parameter erforderlich sind, so sind in einer POST-Methode oft komplexe Anfrage-Inhalte zu entwerfen, um diese Routen testen zu können.

Bei der Implementierung der Test-Pakete wurden zuerst die einfachsten Tests umgesetzt, um möglichst schnell eine hohe Testabdeckung zu erreichen.

Unit Tests gehören den Entwicklern

Unit Tests sollen grundsätzlich den Großteil eines jeden Testplans ausmachen[3]. Dabei sind Sie Teil der jeweiligen Softwarekomponente, und somit in der Zuständigkeit der Entwickler. Dem Entwickler obliegt daher die Entscheidung, ob und welche Menge an Unit Tests er implementiert. Daher werden im Rahmen dieser Arbeit keine Unit-Tests implementiert. An dieser Stelle soll den Entwicklern aber empfohlen werden, zumindest Unit-Tests für komplexere Bausteine zu entwerfen, und vor jeder Änderung auszuführen.

3.3.2 Zusammenfassung der Komponenten in Test-Gruppen

Zur Vereinfachung werden die Komponenten des Kundenportals in 3 Gruppen eingeordnet, welche über die jeweilig vorhandene Schnittstelle getestet werden.

Diese Komponenten werden zusammen mit den jeweils Backend-Seitigen Komponenten integrativ getestet.

[3] Vgl. „How Google Tests Software", James Whittaker

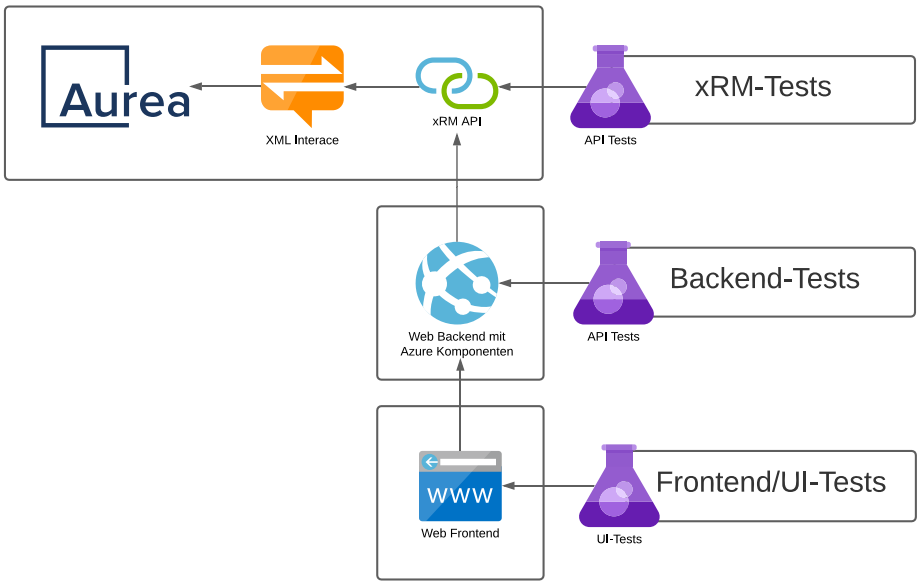


Abbildung 5 - Einordnung der Testpakete in die Architektur des Portals

3.3.3 Entwurf der Test-Pipeline

Bei Integrationstest besteht der Nachteil, dass die Fehler oft nicht genau lokalisiert werden können, da mehrere Softwarekomponenten für das Ergebnis eines Testfalls verantwortlich sind.

Dieser Nachteil soll ausgeglichen werden, indem die Testpakete in einer bestimmen Reihenfolge ausgeführt werden. Dabei werden zuerst die xRM-Tests ausgeführt werden, welche die geringste Anzahl an Abhängigkeiten besitzen. Bei Erfolg kommt es zum Ausführen der BE-Tests. Sollte bei den Backend-Tests ein Fehler auftreten, kann nahezu ausgeschlossen werden, dass dieser auf den zuvor getesteten Komponenten beruht. Analog dazu erfolgen im Anschluss die Frontend-Tests. Dieser Ablauf ist in der folgenden Grafik dargestellt:

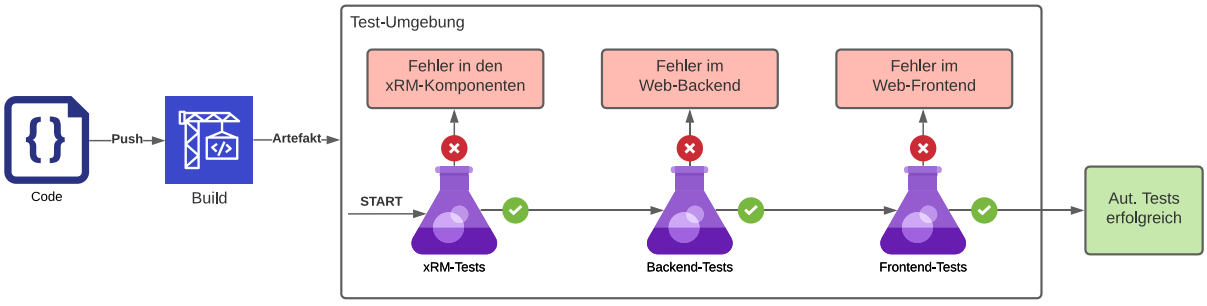


Abbildung 6 - Release-Pipeline Teil 1

Wenn alle Tests erfolgreich waren können die Software Artefakte auf der Akzeptanz-Umgebung bereitgestellt werden. Diese Umgebung dient dazu, den Produktverantwortlichen die Möglichkeit zu geben, manuelle Akzeptanztests durchzuführen.

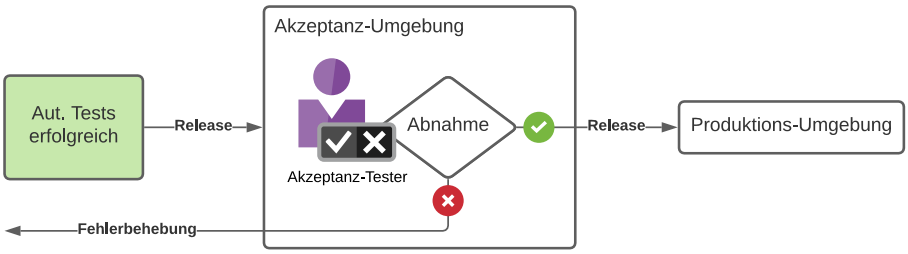


Abbildung 7 - Release-Pipeline Teil 2

Bei Abnahme kann das Release auf die Produktions-Umgebung vorgenommen werden. Die Entscheidung, wann dies zeitlich vorzunehmen ist, obliegt auch den Produktverantwortlichen, und erfolgt nicht automatisch[4].

3.3.4 Design der Testfälle

Zur Reduzierung des Aufwands bei der technischen Umsetzung der Testfälle, soll zuvor untersucht werden, wie Tests der beiden Kategorien (UI-Tests, API-Tests) mit dem besten Verhältnis aus Aufwand und Nutzen implementiert werden können.

Für Tests beider Kategorien gilt, dass die Anzahl der Testfälle, durch Parametrisierung, theoretisch beliebig gesteigert werden kann. Bei dem Testfall ‚User Registrierung' könnte der eingegebene Name parametrisiert werden, und somit eine nahezu unendliche Menge an Testfällen erzeugt werden. Bei welchen Test-Eingaben, eine Parametrisierung in welchem Maße zweckdienlich ist, muss je Testfall entschieden werden.

UI-/Frontend-Tests

UI-Tests sind sehr aufwändig zu implementieren, daher soll die Anzahl der Tests geringgehalten werden. Um dennoch eine hohe Testabdeckung zu erreichen sollen die Testfälle dementsprechend groß sein. Ein großer Testfall bringt das Problem mit sich, dass der Grund eines auftretenden Fehlers, anhand des Testergebnis, nur schwer lokalisiert werden kann. Da der Fehler aber im Frontend erscheint, kann er mittels des Browsers einfach reproduziert werden, sofern dem Entwickler die vorigen Interaktionen des „Klick-Roboters" bekannt sind. Um dem Entwickler diese Reproduktion möglichst einfach zu

[4] Gemäß des DevOps-Prinzips[G] wird dieses Vorgehen als Continuous Delivery bezeichnet

ermöglichen, sollen die Testfälle von Benamung und Test-Umfang immer möglichen User-Interaktionen im Portal entsprechen, bspw. ‚Ein Privatkunde bestellt das Produkt X'.

API-Tests

API-Tests sind schneller implementierbar als UI-Tests. Den geringsten Aufwand erreicht man bei diesen Tests, wenn jede verfügbare Route möglichst separat getestet wird. Durch Angabe der Route, sowie der Payload des Requests, bekommt der Entwickler alle Informationen, um den Fehler reproduzieren zu können.

Die API-Tests sind dabei in 3 Gruppen eingeteilt:

Status-Tests

Status-Tests sind einfach zu implementieren und schnell in der Ausführung. Dabei werden alle verfügbaren Routen, wenn nötig mit minimalen Parametern, aufgerufen. Viele Fehler im Code, welche erst zur Laufzeit auftreten, können so mit sehr geringem Aufwand aufgedeckt werden. Das Auftreten eines solchen Fehlers, der zu einer nicht behandelten Ausnahme im Programm führt, kann über den http-Status-Code 500 erkannt werden.

Contract-Tests

Unter Contract-Tests sind Testfälle zusammengefasst, welche die Antwort einer Route auf eine definierte Anfrage überprüfen. Dabei wird der Antwort-Inhalt auf ein definiertes Schema geprüft (Elemente und Datentypen der Antwort), und ob die Größe der Antwort (bspw. die Länge einer Liste von Produkten) in einem plausiblen Rahmen liegt.

Schematische Tests

Schematische Tests sind im Gegensatz zu den beiden obigen Kategorien nicht statisch. Sie benötigen mehrere Anfragen, und nehmen Änderungen in den Datenbanken des Backend vor. Die zweite Abfrage im Test wird zur Validierung dieser Änderung vorgenommen. Das Bestellen eines Produkts, oder eine User-Registrierung, sind Beispiele für schematische Tests.

3.4 Exemplarischer Testfall

In diesem Abschnitt wird exemplarisch für die Funktionalität „Produkt bestellen" eine Menge an Testfällen definiert. Es handelt sich, gemessen an der Anzahl der vom Web-Backend angesprochenen Routen, um einen sehr großen Testfall[5]. Es handelt sich sowohl um einen sehr großen Testfall als auch den Entscheidendsten, da das Bestellen von Produkten die Hauptfunktion des Kundenportals ist.

[5] Siehe Grafik in Abschnitt 8.4 (Anhang)

3.4.1 Zu betrachtende Szenarien

Wie im obigen Abschnitt ‚Design der Testfälle' erwähnt lassen sich Testfälle parametrisieren. Dies wird in diesem Abschnitt vorgenommen.

Verschiedene Account-Kategorien (Privatkunde, Installateur oder Errichter) können Produkte im Kundenportal bestellen.

Ein Installateur oder Errichter bestellt dabei immer für einen Privatkunden, welcher zum Zeitpunkt der Anfrage noch nicht registriert sein muss. Ein Privatkunde kann Produkte für sich selbst bestellen, und hat die Möglichkeit direkt seinen Installateur im Anfrageprozess mit Anzugeben. Auch dieser muss zum Zeitpunkt der Anfrage noch nicht im Portal registriert sein. Diese Szenarien sind in folgender Grafik illustriert:

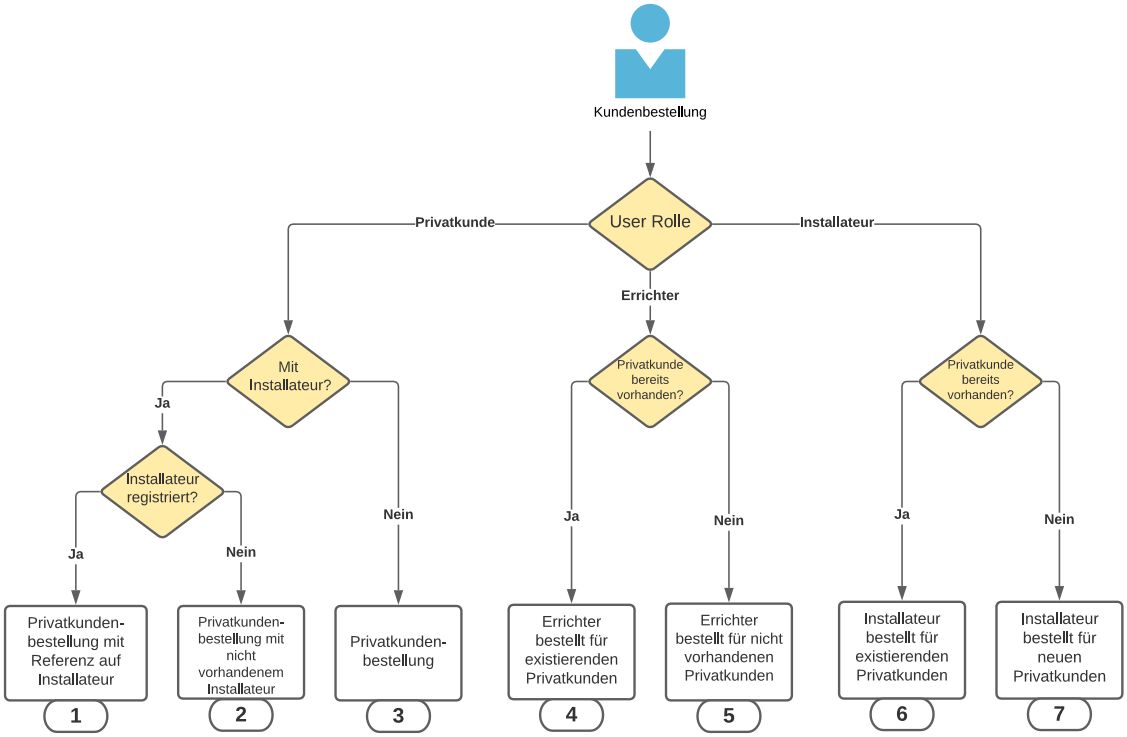


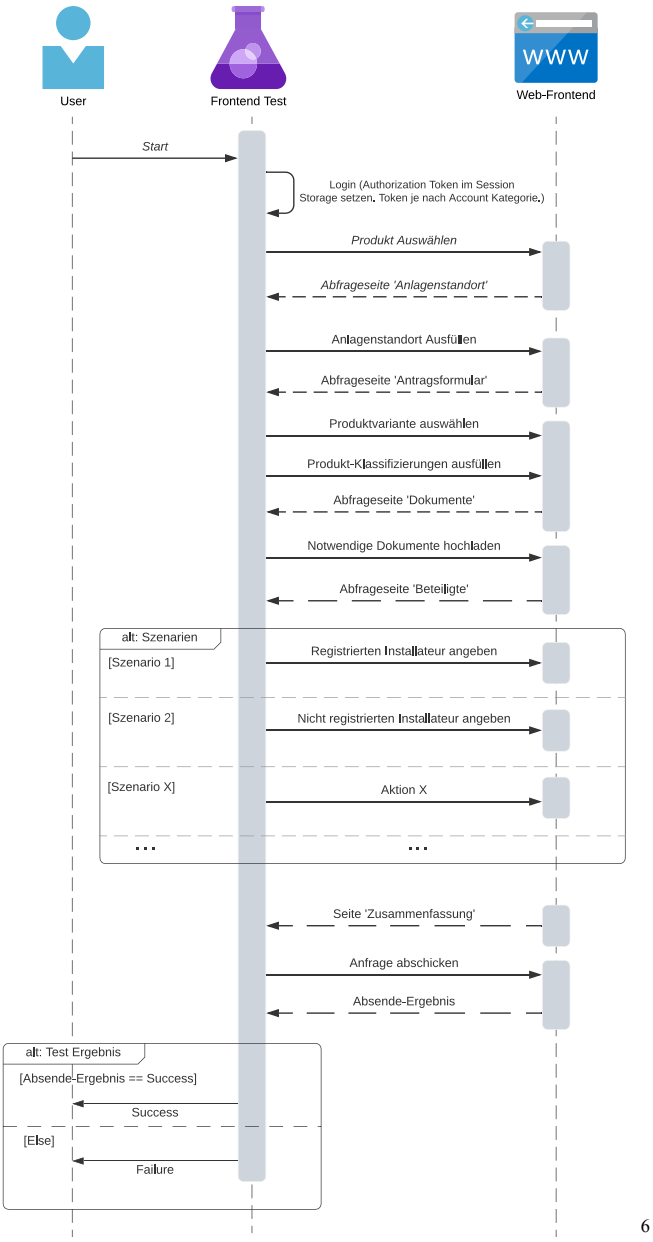
Abbildung 8 - Fallunterscheidung im Test-Szenario Produktbestellung

Es ergeben sich somit sieben verschiedene Bestell-Szenarien.

Nun wäre es naheliegend, die Parametrisierung nicht auf diesen sieben Szenarien zu belassen, sondern bspw. für jedes bestellbare Produkt alle dieser Szenarien zu Testen. Dies soll aber nicht vorgenommen werden, da sonst die Abhängigkeit der Testfälle zu der Produktkonfiguration zu hoch wäre. Das Verhältnis aus Mehraufwand in der Wartung der Tests, und gewonnener Testabdeckung wäre nicht zufriedenstellend.

3.4.2 Testfälle für das Frontend

Die Testfälle für das FE sollen wie oben beschrieben immer komplette Kundenaktionen abbilden. Nach dem Login, mit der jeweiligen User-Rolle, wird ein Produkt bestellt. Die Parameter für diese Bestellung gehen aus der obigen Grafik hervor. Die Erfolgsmeldung des Backends, dargestellt im Frontend, validiert schlussendlich die Funktionalität. Dieser Ablauf ist in folgendem Sequenzdiagramm vereinfacht dargestellt:



6

[6] Die Darstellung, sowie die folgende, erfolgte gemäß „UML 2 – Das umfassende Handbuch“

Abbildung 9- Sequenzdiagramm Backend-Test Produktbestellung

3.4.3 Testfälle für das Web-Backend

Im Web-Backend soll, im Gegensatz zum Frontend, genau die Funktionalität: Bestelle ein Produkt getestet werden. Dazu ist nur die Route *POST ../offer* erforderlich.

Da eine Änderung in der Datenbank des Backends erforderlich ist, sowie eine Validierung mittels eines zweiten Aufrufs, ist dieser Testfall in die Kategorie der Schematischen Tests einzuordnen.

Jede Bestellung enthält Produkt-Parameter, sog. Klassifizierungen. Das korreket Anlegen dieser Parameter wird nach Anlage der Bestellung durch einen GET-Aufruf mit der zuvor gesendeten Payload verglichen.

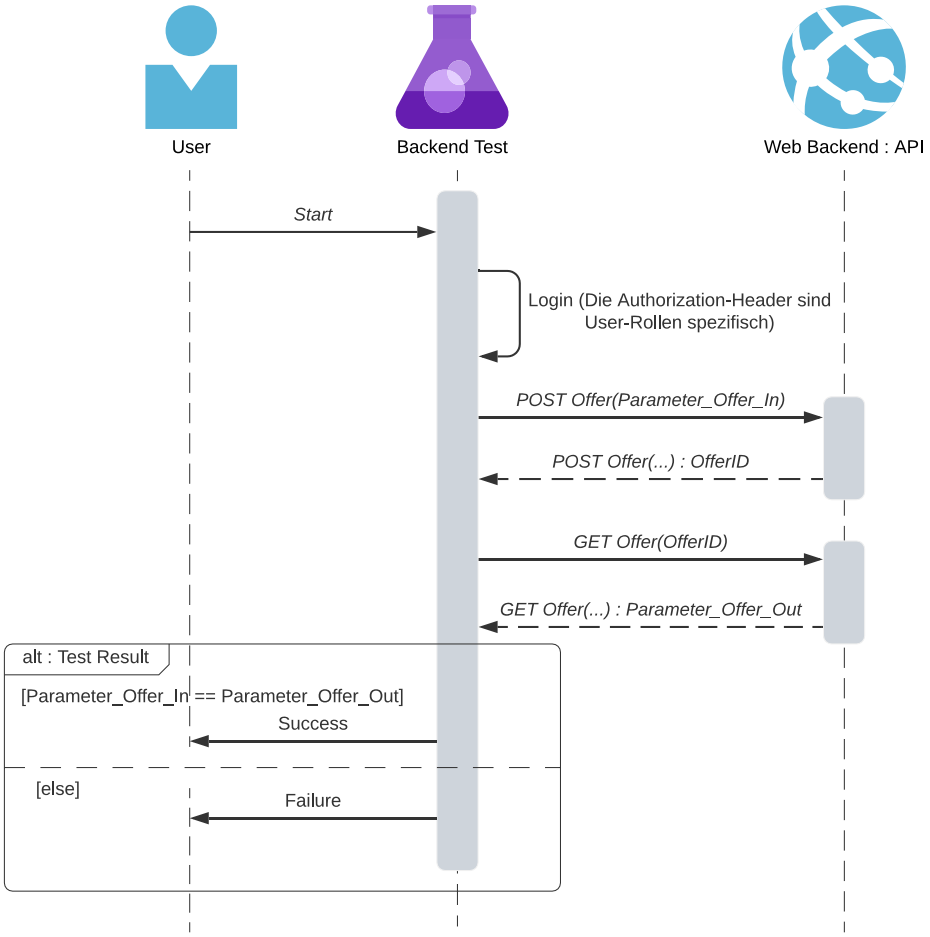


Abbildung 10 - Sequenzdiagramm Backend-Test Produktbestellung

Die beschriebenen 7 verschiedenen Varianten nicht alle berücksichtigt werden. Nur für jede User-Rolle soll ein Bestellprozess getestet werden. Die Gründe dafür liegen darin, dass die anderen Szenarien weitere Routen aufrufen. Diese werden in separaten Testfällen für das Web-Backend behandelt. Somit ergeben

sich nur 3 Szenarien, welche sich im Diagramm nur durch unterschiedliche Authorization Header unterscheiden.

3.4.4 Testfälle für die xRM-API

Der Testablauf für die xRM-API erfolgen ist analog zu dem im Web-Backend, mit Ausnahme des Logins, dieser entfällt. Die Information welcher Akteur das Produkt bestellt ist Teil des Anfragekörpers.

3.5 Test-Werkzeuge

Alle Testpakete werden in Python 3 unter Nutzung des des pytest-Frameworks implementiert. Zur Implementierung der UI-Tests wird zusätzlich Selenium als Frontend-Werkzeug genutzt.

Alle Tests befinden sich zusammen in einem von den Komponenten separaten Repository[7].

Die Struktur des Repositories geht aus der README-Datei hervor.

[7] https://git.imp.fu-berlin.de/christoc97/testpaket-ba-cornelius

4 Deployment-Optimierung

In diesem Abschnitt wird zuerst auf die technischen und organisatorischen Rahmenbedingungen eingegangen, aus denen dann die Deployment-Optimierung entworfen wird. Danach wird die technische Umsetzung der Optimierung vorgestellt. Zum Abschluss wird geprüft, ob das Ergebnis einer Optimierung entspricht.

4.1 Rahmenbedingungen

Folgend werden die Rahmenbedingungen für die einzelnen Softwarekomponenten erläutert.

4.1.1 xRM-System

Das xRM-System besitzt neben den später erläuterten technischen Begrenzungen, bezüglich des Staging, auch organisatorische Einschränkungen. Die organisatorischen Einschränkungen begründen sich damit, dass es nicht explizit der Entwicklung des Kundenportals zur Verfügung steht. Aktuell werden viele Geschäftsprozesse bereits über das xRM-System abgebildet. Diese sollen erhalten bleiben, und teilweise auch weiterentwickelt werden. Gleichzeitig soll der Aufwand des Entwicklerteams, bei der gleichzeitigen Bedienung dieser Fachbereichsbedürfnisse, nicht unangemessen hoch sein.

Um die organisatorischen und technischen Rahmenbedingungen des xRM zu erläutern, wird zuerst erklärt, welche Komponenten des xRM weiterentwickelt, und somit immer wieder bereitgestellt werden müssen.

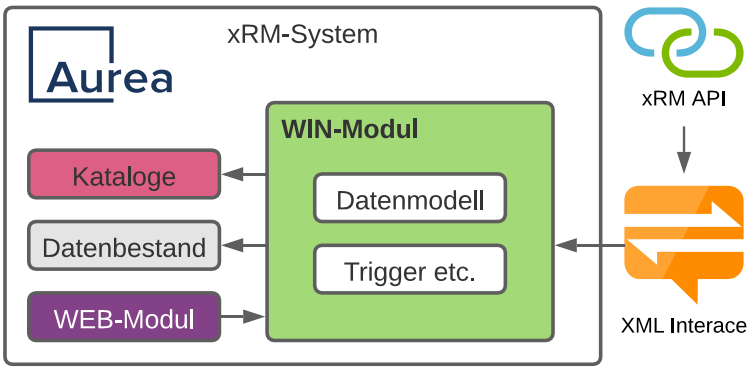


Abbildung 11 - Komponentenansicht des xRM-Systems

In der obigen Grafik sind die relevanten Komponenten des xRM-Systems, sowie deren Abhängigkeiten dargestellt. Die Pfeilrichtung gibt dabei die Zugriffsrichtung an. Diese Komponenten werden auf der folgenden Seite einzeln erklärt.

WIN-Modul

Das WIN-Modul ist die Kern-Komponente des xRM. In ihm können Logiken programmiert sowie das Datenmodell angepasst und erweitert werden. Die Logiken bestehen aus Triggern, Workflows und Prozessen[G]. Das Datenmodell beschreibt die Datenbankstruktur, welche das WIN-Modul auf einem separaten Datenbankserver verwaltet.

Das WIN-Modul kann nur vom Entwicklungsserver ausgehend, der sogenannten „Zentrale“, an alle anderen Server mittels eines WIN-Transports[G] übertragen werden.

WEB-Modul

Das WEB-Modul ist die für den Sachbearbeiter sichtbare Oberfläche. In ihm können die Kundendaten sowie Web-Anfragen verwaltet werden. Die Konfiguration des WEB-Moduls hat keinen Einfluss auf das vom Kunden genutzte Frontend, es kann als separates Frontend betrachtet werden. Dennoch müssen bei manchen Änderungen die Views für den Sachbearbeiter angepasst werden, damit dieser bspw. neue Felder im Datenmodell auch auf Web-Oberfläche sehen kann. Das WEB-Modul wird über den Transportweg des Web-Transports[G], ausgehend von der Zentrale, übertragen.

Kataloge

Bei Katalogen handelt es sich um Wertelisten. Kataloge müssen bei Änderungen über einen separaten Transportweg zwischen den Systemen synchronisiert werden. Die Synchronisierung bei Neuanlage von Katalogen. erfolgt abweichend zur Änderung von Katalogen, mit dem WIN-Transport[G].

Tabellenwerte / Datenbestand

Unter Tabellenwerte sind Einträge in Datenbanken zu verstehen. Diese werden in den CRM-Systemen der Stromnetz Berlin üblicherweise nicht zwischen Systemen übertragen. Durch den Umstand, dass Datenbestände Produkte im Kundenportal beschreiben, ist dies in diesem Projekt dennoch erforderlich.

4.1.2 Organisatorische Einschränkungen im xRM

Beziehen von Katalogen aus der Produktivumgebung

Das xRM wird in der Produktivumgebung von vielen Fachbereichen der Stromnetz Berlin benutzt. Diese nehmen Änderungen in Wertelisten und Konfigurationen vor, welche teilweise auch Auswirkungen auf die Webshop-Funktionalitäten haben. Daher ist es notwendig diese Änderungen bei Weiterentwicklung zu berücksichtigen.

Daraus folgt, dass bevor mit Weiterentwicklungen auf dem Entwicklungssystem gestartet werden kann, Änderungen auf dem Produktionssystem in der Entwicklungsumgebung zu übernehmen sind.

Produkt-Entwicklungen außerhalb der Entwicklungsumgebung

Die vom Kunden bestellbaren Produkte werden als Datenbestände auf dem Akzeptanz-System, statt auf dem Entwicklungssystem angelegt. Das Akzeptanz-System dient als sog. Redaktionssystem für diese Datenbestände, und stellt immer den aktuellen Stand der Produktkonfigurationen dar.

Eine falsche Konfiguration der Produkte kann zum Ausfall des Kundenportals führen, daher sind die Produkte in diesem Kontext genauso kritisch zu betrachten wie programmierte Komponenten.

Die Gründe für die Entscheidung, die Produktentwicklung in der Akzeptanz-Umgebung zu vollziehen, sind folgende:

1. Die Fachbereiche sollen in Zukunft selbst die Produktpflege übernehmen. Diese wissen am besten, welche Nutzereingaben Sie von den Kunden haben möchten, und wie die Produkte im FE ausgestaltet werden sollen. Um ihre Änderungen selbst validieren zu können, müssen Sie diese auch auf der Website selbst betrachten können.
2. Beim Anlegen neuer Produkte möchte ich deren Erscheinung und Funktionalität in einer möglichst Produktionsnahen Umgebung prüfen. Diese Anforderung erfüllt die Akzeptanz-Umgebung zu jedem Zeitpunkt, da es immer den (zukünftigen) Entwicklungsstand der Produktion hat, sowie einen Datenbestand, der der Produktionsumgebung sehr ähnlich ist.

4.1.3 Technische Einschränkungen im xRM

Wie erwähnt lässt sich das xRM als „ganzes“ weder technisch noch organisatorisch von einer auf die andere Umgebung übertragen. Es gibt verschiedene Transport bzw. Kommunikationswege, um Entwicklungen zwischen den Servern zu übertragen.

Es existiert vom Hersteller kein automatischer Transport für diese Wege. Zum Transport der Entwicklungen lassen sich folgende Anforderungen an den Entwickler definieren:

- Administrator-Zugang auf allen xRM Servern
- Kenntnis über den technischen Ablauf der verschiedenen Transportvorgänge

Über alle oben genannten Anforderungen verfügen im Entwicklerteam nur 2 Personen. Zudem sind die Abläufe vergleichsweise zeitintensiv, Sie nehmen durchschnittlich 10min in Anspruch.

4.1.4 Alle weiteren Software-Komponenten

Die xRM-API sowie das Web-Back-/Frontend sind als Git-Repositorys vorhanden und können so optimal versioniert werden. Beliebige Softwareartefakte können auf die betreffenden Systeme jederzeit eingespielt werden. Es ergeben sich keine technischen Einschränkungen.

Es existieren auch keine organisatorischen Einschränkungen, da diese Komponenten momentan ausschließlich für das Kundenportal genutzt werden.

4.2 Entwurf der Optimierung

Aus dem letzten Abschnitt ergibt sich das Ziel, nicht das theoretisch bestmögliche Deployment-Konzept zu erdenken, sondern mit den für das xRM technisch- und organisatorisch durchführbaren Transportwegen, die bestmögliche Gesamt-Plattform für die (Weiter-)Entwicklung des Kundenportals zu entwerfen.

Das xRM besitzt Werkzeuge zum Übertragen von Entwicklungen, welche sich allerdings nicht an gängigen Software-Deployment Strategien orientiert, sondern an Anwendungsfällen in der Domäne eines CRM-Systems[G].

Daraus ergibt sich die Fragestellung:

Wie lässt sich die Anwendungsdomäne des CRM-Systems, in einen für Software-Entwicklungen üblichen Staging-Ansatz übersetzen, und welche technischen Voraussetzungen sind dafür zu schaffen?

Um diese Frage zu beantworten, werden nun die aktuellen Transport-Routinen vorgestellt, welche aufgrund der o.g. Einschränkungen nicht veränderbar sind.

4.2.1 Aktuelle Versionierung des xRM

In folgender Grafik sind die aktuellen Übertragungsvorgänge der einzelnen xRM Systeme dargestellt.

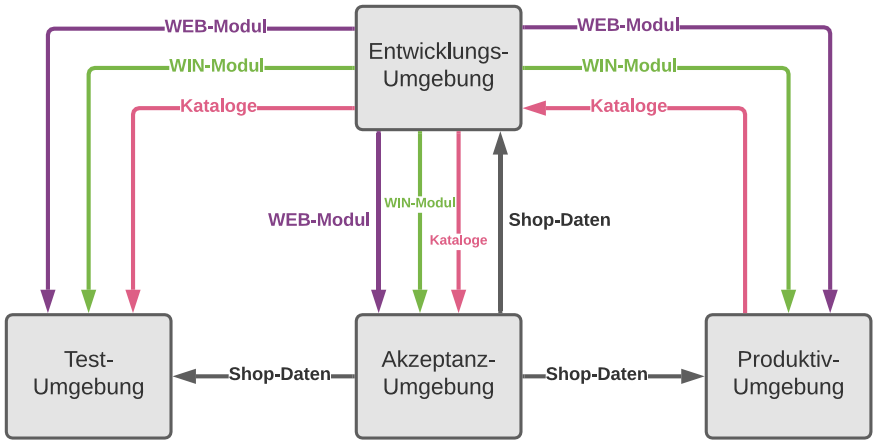


Abbildung 12 - Transportwege des xRM-Systems

Die ausgehenden Pfeile aus der Entwicklungs-Umgebung, an alle anderen Systeme, erklären sich durch das Zentrale/Außenstationen-Konzept des CRM-Systems.

Dabei stellen die Pfeile immer ein Delta an Konfigurationen dar, welches zwischen den Systemen kommuniziert wird. Jede Station weiß, welche Informationen das andere System bereits vom Ausgehenden erhalten hat, und überträgt nur Informationen über Änderungen. Das Zielsystem informiert das Quellsystem mittels einer Notifikation über die eingespielten Änderungen. Diese muss separat, nach jedem Übertragungs-Vorgang, in umgekehrter Pfeilrichtung, ebenso manuell übertragen werden.

Die ausgehenden Pfeile des Akzeptanz-Systems erklären sich durch die o.g. organisatorische Einschränkung, gemäß der die Produkte auf diesem System gepflegt werden. Auf dem Akzeptanz-System befindet sich der neueste Stand der Produktkonfiguration, und auch dort wird die Konfiguration von den jeweiligen Fachbereichen im Unternehmen überprüft.

Der Umstand, dass die Übertragung der Katalogwerte vom Produktionssystem an die Systeme Test und Akzeptanz über die Entwicklungs-Station erfolgt, hat weder technische noch organisatorische Gründe. Sie könnte auch direkt erfolgen. Eine Änderung dieses Weges ist mit Aufwand verbunden, welcher aktuell keine absehbaren Vor- oder Nachteile mit sich bringt. Es ist immer eine vollständige Synchronisation dieser Werte angestrebt, über welchen Weg diese verläuft ist nicht entscheidend.

Diese, auf den ersten Blick unübersichtliche, Menge von Transportwegen beruht somit vollständig auf organisatorischen und technischen Umständen. Diese organisatorischen Umstände lassen sich teils konzeptionell umgehen, aber aktuell nicht umsetzen. Der aktuelle Betriebsübergang der Stromnetz Berlin erlaubt grundsätzliche jegliche Änderungen außerhalb des Projekts Kundenportal nicht.

Aufwand und Zuverlässigkeit dieser Übertragungsvorgänge können aber durch technische Optimierung reduziert werden.

Um einen technisch geeigneten Ansatz finden zu können, wird nun überprüft wo die größten Abweichungen zu den „Best practices“ des Software-Deployens zu finden sind. Anschließend werden diese Abweichungen durch technische Optimierungen reduziert.

4.2.2 Abweichung vom „richtigen“ Pfad bei der aktuellen Integration

Für das Deployment von Software existieren zahlreiche Grundsätze und Handlungsempfehlungen[8]. Die aktuellen Deployment-Abläufe des xRM-Systems weichen von diesen Empfehlungen teils deutlich ab. Die wesentlichsten Abweichungen werden in diesem Abschnitt vorgestellt:

[8] Aus „Continuous Delivery“, Part 1, Chapter 1

„Automatisiere den Deployment Prozess“

Grundsätzlich sollen Deployment Prozesse automatisiert werden. Die aktuellen Deployments laufen nahezu vollständig manuell ab. Dazu werden Neuentwicklungen händisch zwischen Servern kopiert und mittels Skripten in die Systemkonfigurationen übertragen.

„Schnelle Deployments“

Deployments sollen wenig Zeit in Anspruch nehmen. Im xRM-System nimmt ein Deployment unter Berücksichtigung aller xRM-Komponenten[9] etwa 15 Minuten in Anspruch, welche den Entwickler dauerhaft aktiv beschäftigen. Die tatsächliche ‘Rechenzeit’ des Deployments nimmt dabei nur einen Bruchteil der Zeit ein. Gleichzeitig ist der manuelle Weg fehleranfällig, und auf wenige Personen beschränkt.

„Baue nur einmal“ - Nur einmalige Erstellung von Software-Artefakten

Software-Artefakte sollen für alle Umgebungen nur einmal gebaut werden. Etwaige umgebungs-spezifischen Konfigurationen sollen zur Deploy-Zeit gesetzt werden.

Das Kommunikationskonzept von Aurea Servern erlaubt im Herstellerumfang diese Art des Deployments nicht. Es werden keine Builds übertragen, sondern Änderungen in der Konfiguration, mittels vom System selbst definierter Protokolle. Daraus ergibt sich folgende akute Problematik:

Angenommen alle Systeme haben den gleichen Stand.

Nun werden auf dem Entwicklungssystem Änderungen vorgenommen, welche an das Test-System übertragen werden. Aufgrund der Größe der Änderungen werden diese erst nach 3 Tagen auf dem Testsystem abgenommen, und genau diese Änderungen sollen nun an die Akzeptanz-Umgebung übertragen werden. Ab dem Zeitpunkt der Übertragung auf das Testsystem, vor 3 Tagen, wurden aber bereits neue Funktionen auf dem Entwicklungssystem getätigt. Der alte Stand des Entwicklungssystems, welcher auf dem Test System vorhanden ist, kann somit nicht mehr an das Akzeptanz-System übertragen werden, da Entwicklungen immer nur ausgehend vom Entwicklungssystem übertragen werden können.

Diese Einschränkung verhindert das Deployment von einzelnen Entwicklungen an das Akzeptanz-System. Erst wenn auf dem Entwicklungssystem keine Änderungen mehr gemacht werden, und alle diese Änderungen auf dem Test-System verifiziert wurden, kann der komplette Stand an das Akzeptanz- und Produktiv-System übertragen werden.

[9] Siehe Komponenten des xRM-Systems

„Rollback Strategie“

Ein Rollback ist aktuell nur für einige Komponenten des xRM möglich, und auch dort mit erheblichem Aufwand verbunden. Grundsätzlich werden keine Systemabbilde erstellt, welche für ein Rollback erforderlich wären.

Die aktuell übliche Lösung für ein Rollback besteht darin, die Änderungen in der Konfiguration wieder zu entfernen, und erneut an die anderen xRM-Systeme zu übertragen. Komplette Rollback-Systemabbilde werden nicht verwendet.

4.2.3 Technische Optimierung des Deployment-Prozesses vom xRM System

Zur Automatisierung der Deployment Prozesse des xRM wurde im Rahmen dieser Arbeit eine API implementiert[10], und auf allen xRM-Servern installiert, welche über definierte Routen Interaktionen mit dem Filesystem, sowie den aus/einspiel-Programmen erlaubt. Dadurch ist es möglich mittels einer HTTP-Schnittstelle Änderungen zwischen den Systemen zu übertragen.

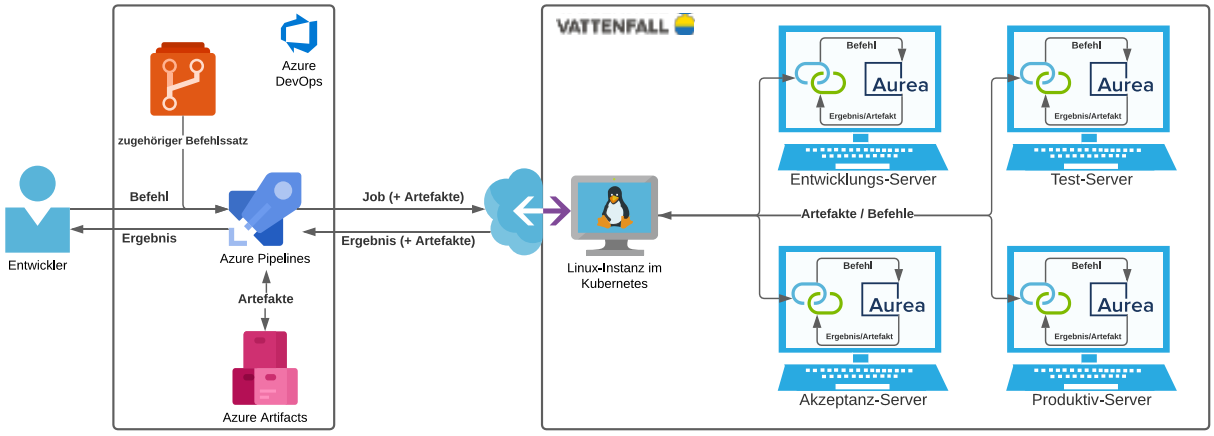


Abbildung 13 - Funktionsweise der Deployment-Optimierung

Die Aufrufe auf diesen APIs, sowie Datei Up- und Downloads, werden von einer Kubernetes Instanz ausgeführt, welche OnPremises gehostet ist und der mittels Azure Pipelines “Jobs” zugewiesen werden können. Diese “Jobs” beinhalten ein Programm zum Aufrufen der Schnittstellen, und zur Weitergabe der Dateien an die anderen Systeme. Mithilfe eines Jobs kann beispielweise eine Änderung von einem Server abgerufen, und auf dem anderen Server eingespielt werden.

Daneben besteht die Möglichkeit, die zwischen den Servern zu kommunizierenden Dateien in der DevOps Komponente „Azure Artifacts“ zwischenzuspeichern. Bei der Übertragung an das Test-System können zusätzlich die Änderungs-Pakete für den Akzeptanz- und Produktivserver erstellt werden. Diese können automatisiert in der Artifacts Komponente gespeichert, und erst später eingespielt werden. Dieses spätere

[10] https://git.imp.fu-berlin.de/christoc97/deployment-optimierung-ba-cornelius

Einspielen würde schrittweise erfolgen, wenn die Voraussetzungen gemäß des Test-Konzepts gegeben sind[11]. Diese Vorgehensweise löst die im Punkt „Baue nur einmal“ beschriebene Problematik.

4.2.4 Erreichte Optimierung

Die implementierte Schnittstelle, mit zugehöriger Befehlssatz-Logik für die Linux-Instanz, ermöglicht es außerhalb des Vattenfall Intranets, und ohne Administrator Zugang zu den einzelnen Servern, die Entwicklungsstände zwischen den Systemen automatisiert zu übertragen.

Dabei wurden im Rahmen dieser Arbeit nicht alle Transport-Möglichkeiten implementiert. Welche Routen bereits implementiert sind geht ausfolgender Grafik hervor.

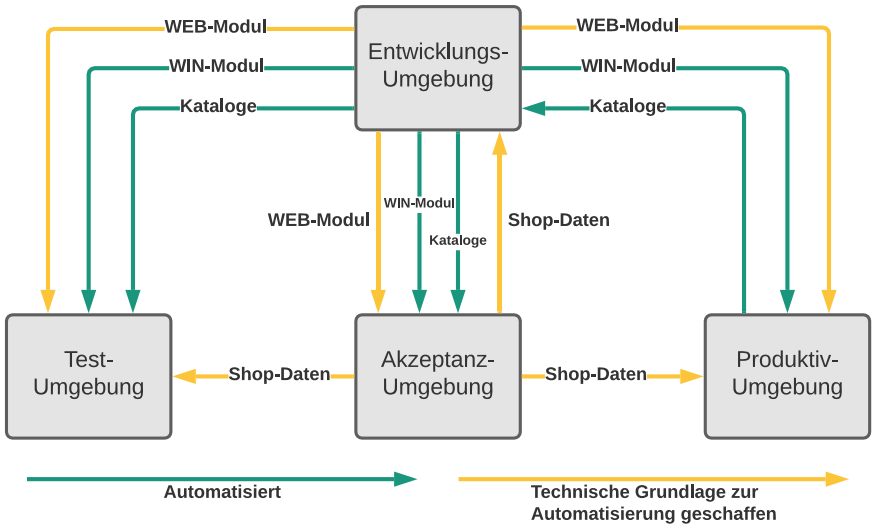


Abbildung 14 - Automatisierungsstand der Transportwege des xRM

Die Berechtigungen für diese Bereitstellungsvorgänge können, für jeden Job und User spezifisch, über Azure DevOps eingestellt werden. Daraus folgt eine einfachere Zugänglichkeit zu Deployment-Triggern, als auch ein feiner definierbares Rechtekonzept, über die Benutzer-spezifischen Berechtigungseinstellungen im Azure DevOps.

Das Prinzip “Automatisiere den Deployment Prozess” kann durch diese technischen Schnittstellen als erfüllt angesehen werden, da durch wenige Klicks ein Deployment ausgelöst werden kann.

Die Ausführungszeiten liegen für alle Arten von Deployments jetzt zwischen 20 und 50 Sekunden, daher kann der Punkt „Schnelle Deployments“ ebenfalls als erfüllt angesehen werden.

[11] Siehe 3.3.2 - Teststrategie

Das Prinzip „Baue nur einmal“ ist nicht vollumfänglich erfüllt. Es wurde aber ein Lösungsweg vorgestellt, welcher ein Problem bei der aktuellen Versionierung löst.

Eine abweichende Rollback-Strategie wurde im Rahmen der Optimierung nicht entwickelt. Die implementierte technische Optimierung beschleunigt und vereinfach aber die aktuellen Handlungsabläufe zum Widerherstellen von früheren Systemzuständen des xRM-Systems.

Da es sich bei den zu automatisierenden Deployments offensichtlich nur um einfache Datei-Operationen, und das Ausführen von Skripten handelt, wären auch andere, technisch einfachere, Lösungen denkbar. Zwei weitere Lösungsansätze werden nun kurz vorgestellt, und es wird erklärt, warum diese nicht gleichwertig zu der implementieren Lösung sind bzw. sein können.

4.2.5 Alternative technische Lösungswege

Vorab zu erwähnen ist, dass während der Bearbeitungszeit dieser Arbeit, der, gemäß Hersteller, finale Major-Release (Version 13) der Aurea CRM-Software auf den Servern installiert wurde. Die implementierte Schnittstelle ist auch mit dieser neuen Version kompatibel. Weitere Anpassungen an der Software Schnittstelle sind aufgrund der finalen Version mit hoher Wahrscheinlichkeit nie wieder erforderlich.

Variante 1 – Power Shell Script:

Mithilfe eines Power Shell Scripts können auf Windows-Servern Datei-Operationen sowie Skripte automatisch gestartet werden. Durch den Umstand, dass der Entwickler, als Administrator für alle beteiligten Systeme, diese Skripte ausführt ist es auch möglich Dateien direkt auf andere Server zu verschieben und dort die jeweiligen Routinen auszulösen.

Somit scheint die Lösung mithilfe eines Power Shell Scripts sowohl technisch möglich zu sein als auch wartungsintensive Software zu vermeiden. Dagegensprechen aber folgende Aspekte:

1. Die Vorgänge sind nur durch Server-Administratoren auslösbar.
2. Es existiert kein Ausführungsprotokoll der Vorgänge, dass für das gesamte Entwicklerteam einsehbar und verständlich ist.
3. Das Testen der Skripte ist fast ausgeschlossen, es muss am “live” System erfolgen. Falsche Befehle im Skript können das Gesamtsystem stark beeinträchtigen, da diese Skripte als Administrator ausgeführt werden müssen.
4. Die Definition der Abläufe wäre aufgrund der Anzahl der Server und Transportroutinen sehr redundant zu implementieren. PowerShell Skripte erlauben keine Vererbung oder Generalisierung zwischen Skripten, es würde viel Logik mehrfach in Skripten existieren.

5. Die Artifacts Komponente, welche Azure DevOps zur Speicherung von Software Artefakten anbietet, kann durch die fehlende Anbindung nicht genutzt werden. Etwaiges vorhalten von Artefakten müsste manuell oder durch separate Implementierung erfolgen.

Variante 2 - Zugriff über SSH auf den xRM-Server

Anstatt der HTTP Schnittstelle könnte die Kubernetes Instanz direkt über ssh auf die Server zugreifen, dies erspart die programmierte Schnittstelle auf den xRM-Servern.

Die SSH Ports der Server sind nicht geöffnet werden, und sollen auch nicht geöffnet werden. Daher scheidet diese Lösung aus.

5 Fazit

5.1 Testkonzept

Ausgehend von der komplexen technischen Umgebung, in der das Kundenportal entwickelt wird, wurde ein möglichst einfaches Testkonzept entwickelt. Es wurde viel Wert darauf gelegt den zukünftigen Aufwand bei der Implementierung und Wartung der Tests gering zu halten, und gleichzeitig eine akzeptable Testabdeckung zu gewährleisten.

Im Zuge des Testkonzepts wurde auch eine Release Pipeline definiert. Diese basiert größtenteils auf allgemeinen Handlungsempfehlungen, und wurde an die konkrete technische und organisatorische Umgebung im Projekt Kundenportal angepasst.

Das Testkonzept bietet, bei vollständiger Umsetzung, großes Potential die Projektentwicklung weiter in Richtung des DevOps-Prinzips zu verlagern.

5.2 Deployment-Optimierung

Im Zuge der Deployment Optimierung wurde erkannt, dass nur beim xRM-System Handlungsbedarf besteht.

Anschließend wurden die einzelnen Bereitstellungsschritte bzw. Routen des xRM-Systems vorgestellt. Es stellte sich heraus, dass organisatorische Änderungen an den komplexen bestehenden Bereitstellungs-Routen organisatorische Änderungen mit sich bringen. Aufgrund der aktuellen betrieblichen Umstände ließen sich diese nicht verändern.

Daher wurde eine technische Optimierung entwickelt. Diese umgeht organisatorische Änderungen, indem Sie bestehende Routen nutzt, und diese automatisiert. Durch die Automatisierung wurde das Vornehmen von Deployments auf den xRM-Systemen auf eine Weise vereinfacht und beschleunigt, die die Weiterentwicklung des Kundenportals unterstützt.

6 Glossar

Begriff	Bedeutung
CRM	Steht für Customer-Relationship-Management. Es bezeichnet für eine Software zum Pflegen von Kundendaten u.v.m..
Domäne eines CRM-Systems	Gemeint ist der Umstand, dass Entwicklungen nur ausgehend von einer zentralen Station an andere „Außenstationen“ übertragen werden können. Dies begründet sich in dem Anwendungsfall, das bspw. ein Vertriebsunternehmen auf einem Zentralen System (Zentrale) neue Funktionen entwickelt, und dann auf die mobilen Arbeitsgeräte (Außenstationen) der im Vertrieb tätigen Personen überspielt.
DevOps-Prinzip	Mit „DevOps“ ist ein Prozessverbesserungsansatz gemeint, der oft in der Softwareentwicklung zur Anwendung kommt. DevOps setzt sich aus den Wörtern „Development“ und „IT Operations“ zusammensetzt.
Ex-/Import	Beschreibt das Exportieren von beliebigen Datenbeständen von einer xRM-Instanz, und den Import dieser Daten in eine andere.
OnPremises	Bezeichnet eine Software(-Komponente) welche sich innerhalb des Vattenfall Intranets befindet. Diese kann, innerhalb ihrer Schicht aufwärts, auf Schnittstellen im Intranet zugreifen, und über einen Web Proxy auf das Internet. Ein Zugriff von außen ist nur über Gateways möglich.
Trigger, Workflows, Prozesse	Trigger sind konfigurierbare Automatismen im xRM WIN-Modul, bspw.: ‚bei Neuanlage eines Angebots, ändere folgende Datenfelder‘. Eine Kombination mehrerer solcher Trigger ergibt einen Workflow oder Prozess.
WEB-Transport	Beschreibt den Transport der Konfiguration der Web-Oberfläche des xRM-Systems auf eine andere xRM-Instanz.
WIN-Transport	Ein WIN-Transport ist die Übertragung der Konfiguration des WIN-Moduls, auf eine andere xRM-Instanz.
xRM	Steht für Extended-Relationship-Management. Es ist als Erweiterung eines CRM-Systems zu sehen, das ‚x‘ steht für beliebige Akteure, welche mit dem Unternehmen in Beziehung stehen.

7 Literaturverzeichnis

James Whittaker, Jason Arbon, Jeff Carollo. *How Google Tests Software.* Addison-Wesley Pearson Education, 2012.

Jez Humble, David Farley. *Coninuous Delivery.* n.d.

Kecher, Christoph. *UML 2 - Das umfassende Handbuch 3. Auflage.* Bonn: Galileo Computing, 2009.

8 Anhang

8.1 Code-Repositories

8.1.1 Deployment API xRM

Programmiersprache:	C#
Genutzte Frameworks:	.Net Core 3
Link:	https://git.imp.fu-berlin.de/christoc97/deployment-optimierung-ba-cornelius
Commit SHA:	1b7ec7e1488a1ca0a6234a1d2be6250d78162ac9

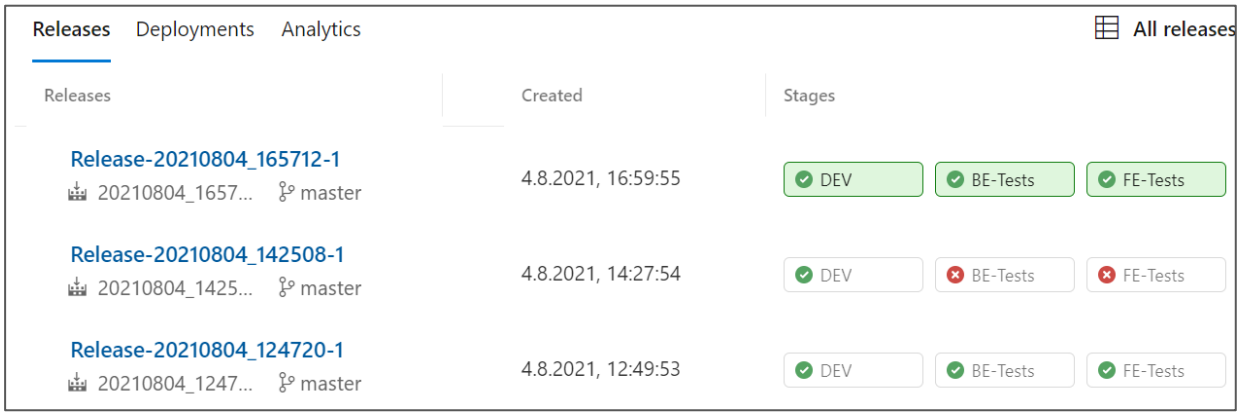
8.1.2 Testpaket

Programmiersprache:	Python3
Genutzte Frameworks:	Pytest
Link zum Repository:	https://git.imp.fu-berlin.de/christoc97/testpaket-ba-cornelius
Commit SHA:	38459b2edca07ec676b4af4760f1789511400871

8.2 Integration der Tests in die Entwicklungsumgebung

In diesem Abschnitt wird kurz gezeigt, wie die Tests in das Entwicklungsportal „Azure DevOps“ integriert sind.

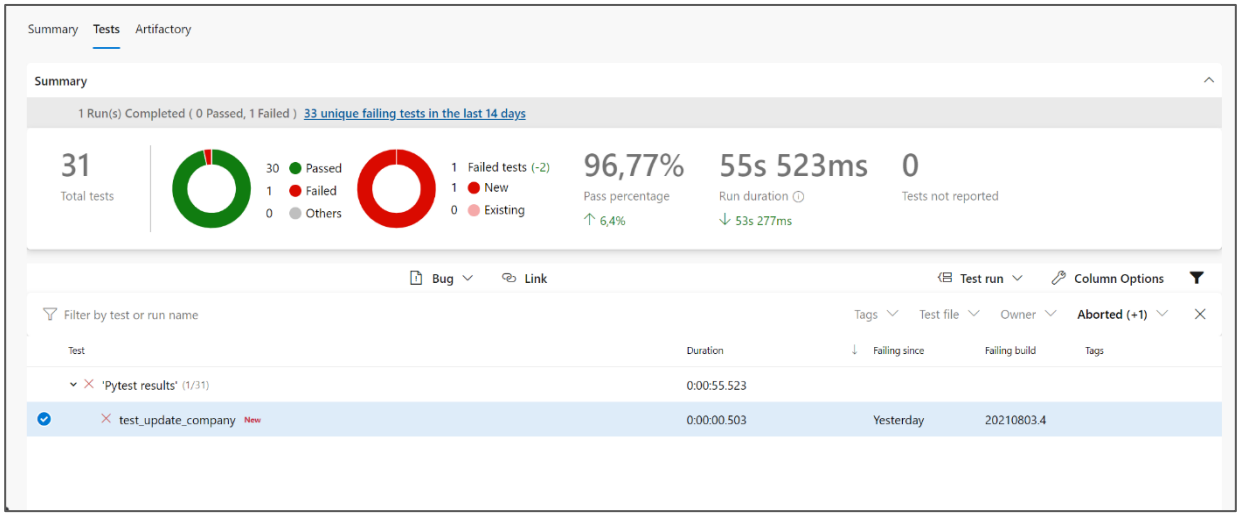
8.2.1 Tests werden nach jedem Release auf Test ausgeführt



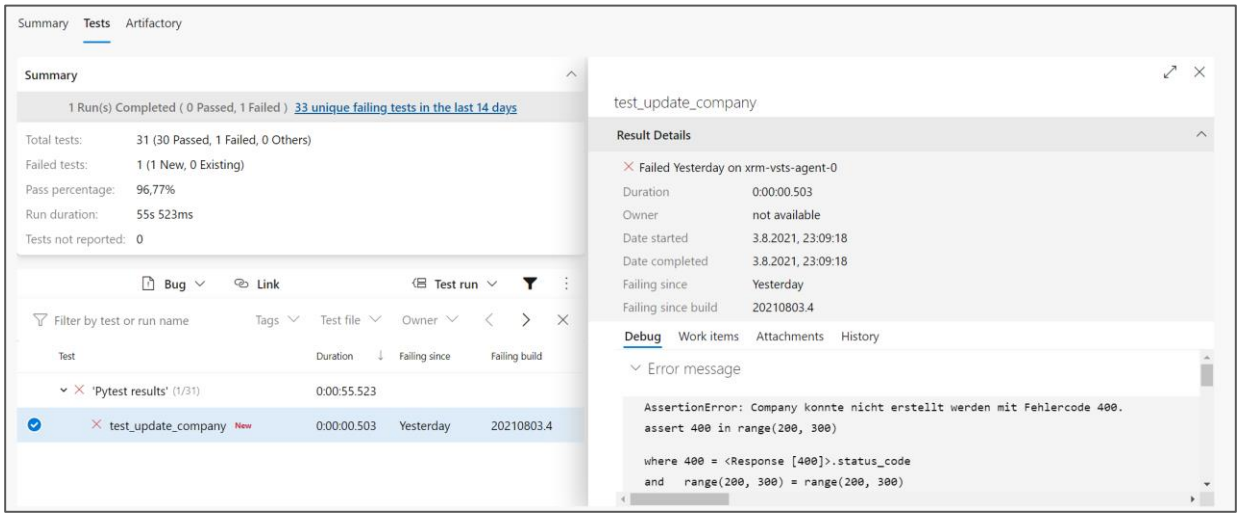
Anmerkung: Bei der Stage namens „DEV“ handelt es sich um die Testumgebung.

Nach einem Release auf die Testumgebung werden die Tests automatisch ausgeführt. Bei fehlgeschlagenen Tests wird der Entwickler informiert.

8.2.2 Ansicht der Testergebnisse

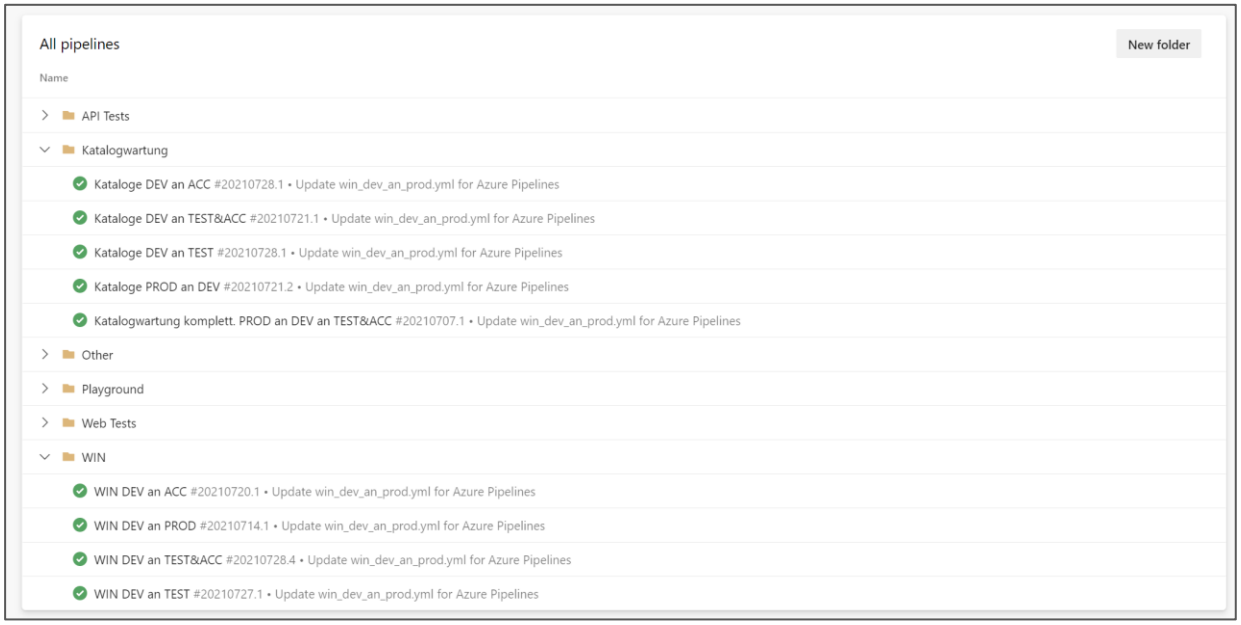


Fehlgeschlagene Tests werden von Azure DevOps erkannt und grafisch dargestellt. Beim Auswählen eines fehlgeschlagenen Testfalls wird eine Beschreibung angezeigt:

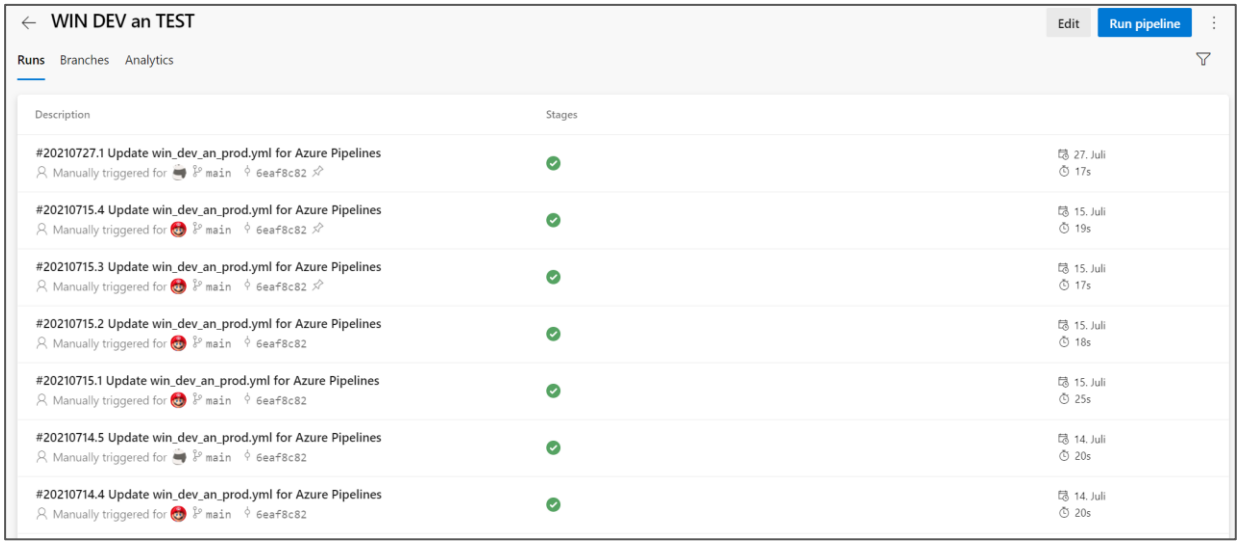


8.3 Integration der xRM-Deployments in die Entwicklungsumgebung

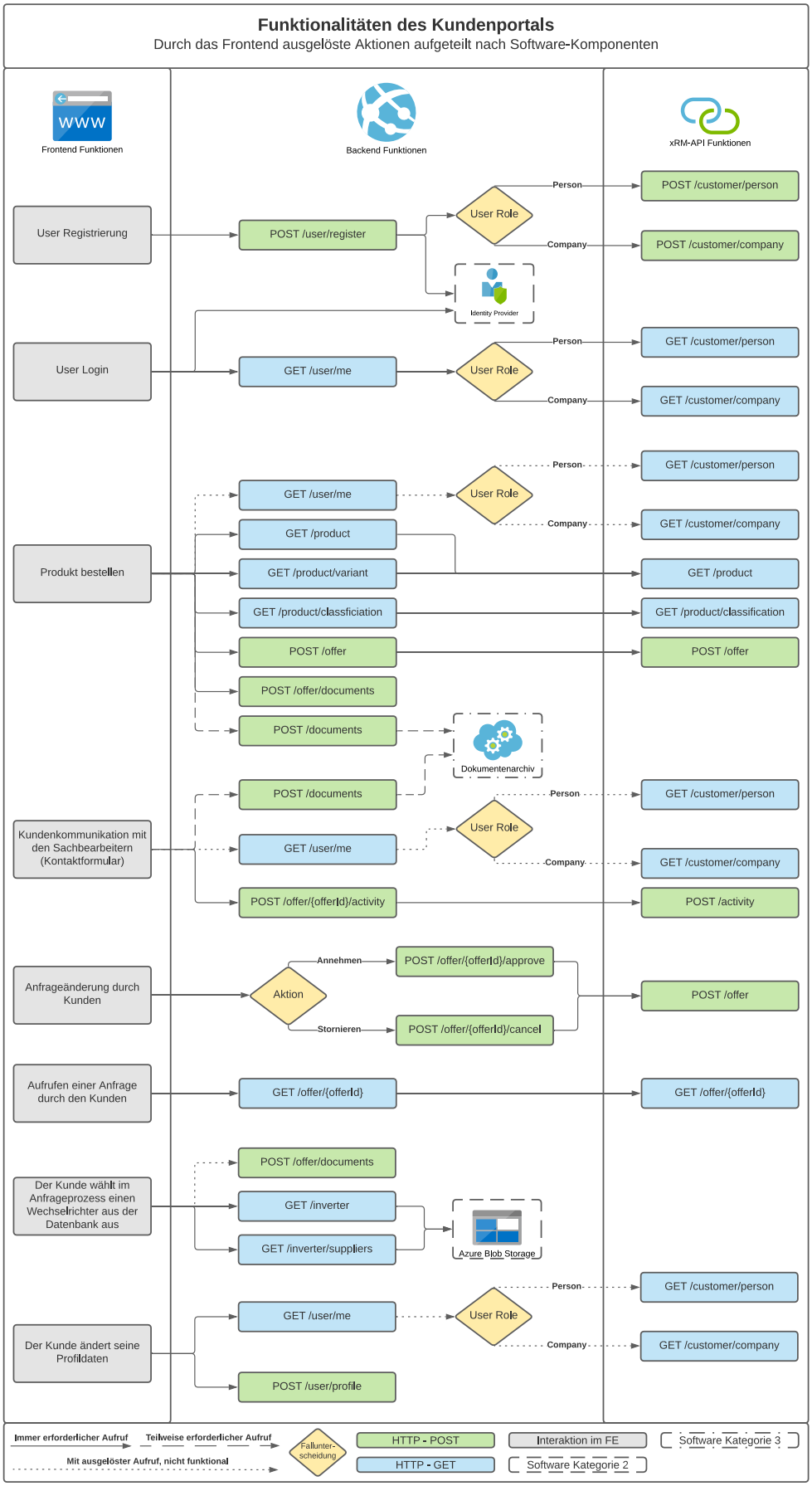
Die Deployments können jetzt durch starten der Pipelines (2 Mausklicks) ausgeführt werden. Das Badge informiert über den Erfolg der letzten Übertragung.



Über „Runs“ ist eine Ausführungshistorie einfach einsehbar:



8.4 Weitere Testszenarien



Freie Universität Berlin

Fachbereich Mathematik, Informatik und Physik

SELBSTSTÄNDIGKEITSERKLÄRUNG

Name:	Cornelius	(BITTE nur Block- oder Maschinenschrift verwenden.)
Vorname(n):	Christoph Peter	
Studiengang:	Bachelor Informatik	
Matr. Nr.:	5344830	

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Datum: 15.08.2021 Unterschrift: C. Cornelius