Refactoring eines Newsletter Addons von ConcreteCMS

Inhaltsverzeichnis

[1 Abgabe Teil 1 6](#_Toc89072536)

[1.1 Theorie Teil 6](#_Toc89072537)

[1.2 Semesterarbeit 8](#_Toc89072538)

[1.2.1 Projektbeschreibung und Ziele 8](#_Toc89072539)

[1.2.2 Projektaufbau 8](#_Toc89072540)

[2 Abgabe Teil 2 10](#_Toc89072541)

[2.1 Semesterarbeit 11](#_Toc89072542)

[2.1.1 Service Architektur 11](#_Toc89072543)

[2.1.1.1 Entitäten 11](#_Toc89072544)

[2.1.1.2 Subscribe Service 13](#_Toc89072545)

[2.1.1.3 Schlussendlich eingesetzte Services 14](#_Toc89072546)

[2.1.2 API-Design 15](#_Toc89072547)

[2.1.2.1 Newsletter-Sets 15](#_Toc89072548)

[2.1.2.2 Newsletters 16](#_Toc89072549)

[2.1.2.3 Weitere HTTP-Response-Codes 16](#_Toc89072550)

[2.2 Abfrage-Beispiele 17](#_Toc89072551)

[2.2.1.1 Newsletter-Alben Liste 17](#_Toc89072552)

[2.2.1.2 Newsletter-Album Update 17](#_Toc89072553)

[2.2.1.3 Löschen eines Newsletters 18](#_Toc89072554)

[2.2.2 Filterung 18](#_Toc89072555)

[2.2.3 Versionierungskonzept 18](#_Toc89072556)

[3 Abgabe Teil 3 19](#_Toc89072557)

[3.1 Von Monolithen zu Microservices 19](#_Toc89072558)

[3.2 Deployment 19](#_Toc89072559)

[3.2.1 Branching Strategie 20](#_Toc89072560)

[3.2.2 SemVer 20](#_Toc89072561)

[3.2.3 Build 21](#_Toc89072562)

[3.2.4 Deploy 21](#_Toc89072563)

[3.3 Test 23](#_Toc89072564)

[3.3.1 Unittests 23](#_Toc89072565)

[3.3.2 API Integrationstests 23](#_Toc89072566)

[3.3.3 Statische Codetests 23](#_Toc89072567)

[3.3.4 End 2 End Tests 24](#_Toc89072568)

[3.3.5 Security Tests 24](#_Toc89072569)

[3.3.6 Performance und Betriebstests 24](#_Toc89072570)

[4 Abgabe Teil 4 26](#_Toc89072571)

[5 Abgabe Teil 5 27](#_Toc89072572)

[6 Literaturverzeichnis 28](#_Toc89072573)

[7 Abbildungsverzeichnis 28](#_Toc89072574)

[8 Tabellenverzeichnis 28](#_Toc89072575)

Einleitung

Das Content Management System (CMS) ConcreteCMS[[1]](#footnote-1) (PortlandLabs, 2021) ist ein Monolith, das in den Scriptsprachen PHP, MySQL und JavaScript geschrieben ist. Wie es bei solchen Systemen üblich ist, ist auch c5 durch Addons (Packages), erweiterbar. Diese Packages sind in der Regel so geschrieben, dass der ganze benötigte Code mitgeliefert wird, womit die Last der Server mit jedem installierten Addon zunimmt. Dies kann im schlimmsten Fall zu Ausfällen der Website führen.

Ein weiterer Nachteil kann sein, dass z.B. Konfigurationsdaten für den Zugriff auf eine externe Ressource auf jedem Hosting einzeln angepasst werden müssen. Im speziellen bietet sich hier das von Mesch betriebene Newsletter-Package an. Mit diesem Package können versionierte Newsletter erstellt und versendet werden, wobei der Versand über einen firmeneigenen E-Mail-Server abgewickelt wird. Wird nun eine Änderung des Ports nötig, über den die Versand-Aufträge von den Hostings auf E-Mail-Server laufen, so muss entweder direkt auf der Datenbank des jeweiligen Hostings oder aber im CMS-Dashboard die Änderung nachgetragen werden.

Weitere Nachteile solcher monolithischen Lösungen werden aus Architektonischer Sicht bei näherer Betrachtung ersichtlich:

* Ein schon riesiges System wird mit weiteren Funktionen angereichert, welche nicht zu ihren Kernkompetenzen (Domäne) gehören
* Vorhandene Daten Konstrukte werden vergewaltigt

Im Beispiel des Newsletter Packages werden, bezogen auf den obigen zweiten Punkt, Benutzergruppen dazu verwendet, ob jemand einen bestimmten Newsletter erhält oder nicht. Diese Gruppenzuweisungen werden vor allem aber auch dazu verwendet, ob ein Benutzer eine bestimmte Tätigkeit ausführen darf.

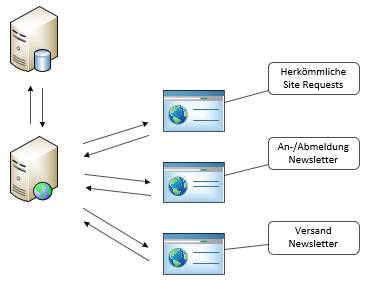


Abbildung 1: CMS wie ConcreteCMS bearbeiten sehr oft auch Anfragen, die Rechner intensiv sein können, aber mit den Kernaufgaben nichts zu tun haben.

Werden nun die Newsletterfunktionen in einen separaten Service ausgegliedert, so sendet, wie in Abbildung 2 zu sehen ist, der Hosting-Server weiterhin die Newsletter-Oberflächen, aber jegliche Funktionen, die die Newsletter betreffen, werden nun von einem anderen Server behandelt[[2]](#footnote-2).

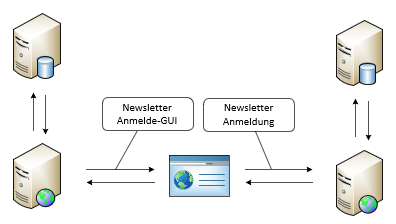


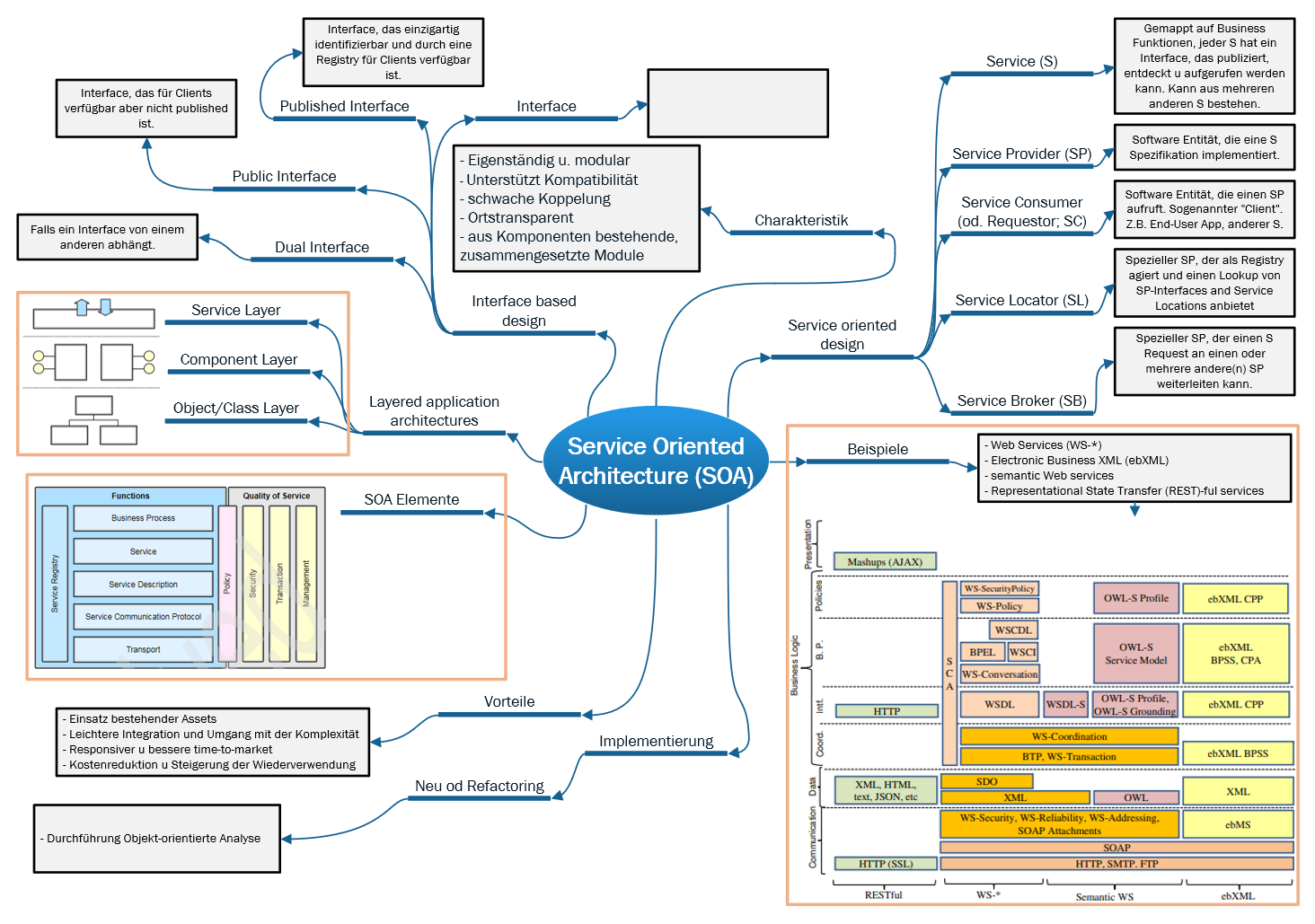
Abbildung 2: Ausgliederung der Newsletter Funktionen in einen eigenen Service, wobei links der Hosting- und rechts der Service-Server ist.

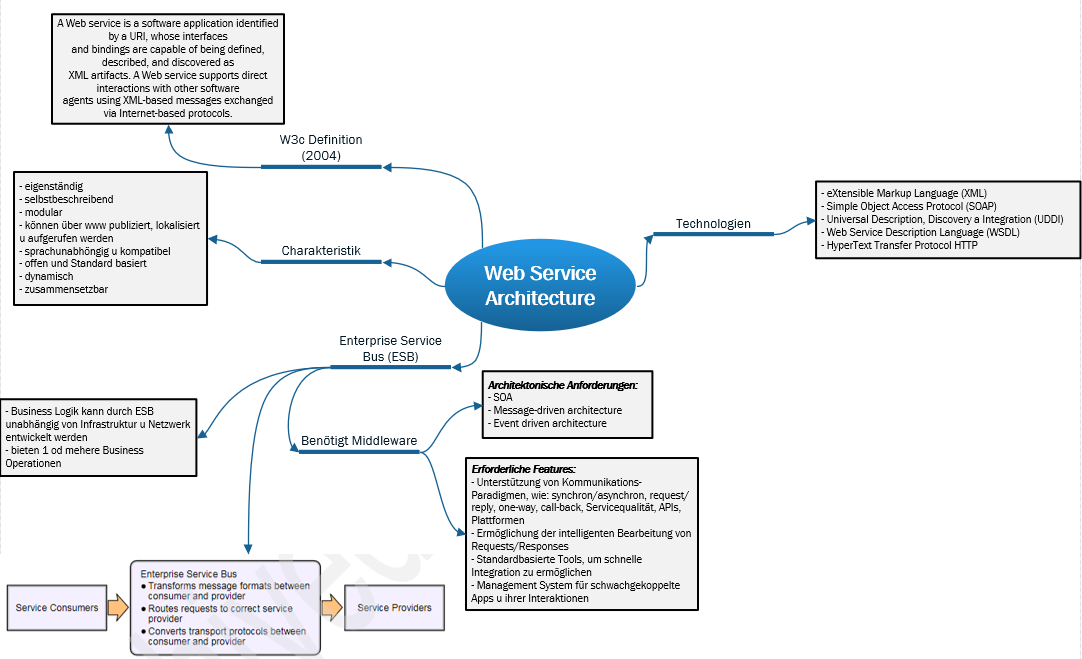
Diese Arbeit behandelt das Refactoring des zuvor vorgestellten Newsletter Packages. Dabei wird ein Dienst gemäss SOA geplant.

Pro Hauptkapitel, die sich den periodischen Abgaben nach den PVAs widmen, befindet sich einerseits ein Theorieteil, der entsprechende Themen behandelt, andererseits die eigentliche Semesterarbeit.

# Abgabe Teil 1

## Theorie Teil





## Semesterarbeit

### Projektbeschreibung und Ziele

Ziel dieses Projekts ist es, die Business Logik des Newsletter Packages von Mesch für c5 in einen Service auszulagern, der den bekannten SOA Pattern folgt. Das umgebaute Package soll mittels RESTful-API Requests auf diesen Service zugreifen[[3]](#footnote-3). Eine neu zu entwickelnder Oberfläche soll das zentrale Konfigurieren die Services ermöglichen, wobei deren Zugriff ebenfalls mittels RESTful API Requests ablaufen muss.

|  |  |  |
| --- | --- | --- |
| Komponente | Neu | Tätigkeiten |
| Newsletter Package | Nein | * Auslagerung der Business-Logik * Entfernung der Konfigurationsseite * Versand neu mittels RESTful API Requests * Statistik neu über RESTful API Requests |
| An-/Abmelde Package | Nein | * Infos der Newsletter mittels REST Requests * An-/Abmeldungen über REST Requests |
| Newsletter Microservice | Ja | * Aufbau der RESTful API-Schnittstelle * Handling Newsletters pro Hosting * Handling Newsletter Versand über ein Queue-System * Handling Konfigurationsanfragen |
| Newsletter Konfigurations-UI | Ja | * Leicht aufsetzbares Hosting * GUI für die Konfiguration des Newsletter Versands * Queue-Administratoren Aufgaben |

Das Projekt wird nach dem agilen Vorgehensmodell durchgeführt, wobei einige Tools von SCRUM (z.B. User Stories, Story Board und Backlog) verwendet werden. Die Versionskontrolle findet auf GitHub statt (Meister & Plüss, 1stthomas / ffhs-soa-work, 2021), wo auch ein vereinfachtes Projektmanagement betrieben wird, (Meister & Plüss, PVA 1 - KickOff, 2021).

### Projektaufbau

Vom Auftrag vordefinierte Meilensteine geben die Projektstruktur vor. Sie sind in der Tabelle 1 aufgeführt.

Tabelle 1: Meilensteine des Projekts.

|  |  |  |
| --- | --- | --- |
| Meilenstein | Auftrag | Datum |
| 1 | Projektstruktur | 05.09.2021 |
| 2 | Service Architektur | 03.10.2021 |
| 3 | Deployment- und Testkonzept | 31.10.2021 |
| 4 | Monitoring- und Security-Konzept | 26.11.2021 |
| 5 | Projektabgabe | 24.12.2021 |

# Abgabe Teil 2

## Semesterarbeit

### Service Architektur

#### Entitäten

Eine Analyse der Geschäftsprozesse ergibt die in Tabelle 2 dargestellten Hauptentitäten.

Tabelle 2: Hauptentitäten der Newsletter-Anwendung.

|  |  |  |
| --- | --- | --- |
| # | Entität | Bemerkung |
| E1 | NewsletterSet | Zusammengesetzte Entität |
| E2 | Newsletter | Komplexere, zusammengesetzte Entität |
| E3 | User | Einfache Entität, v.a. E-Mail |
| E4 | Account | Zusammengesetzte Entität |
| E5 | Log | Einfache Entität |
| E6 | Queue | Zusammengesetzte Entität |

Die Beziehung der Entitäten zueinander ist in Abbildung 3 dargestellt.

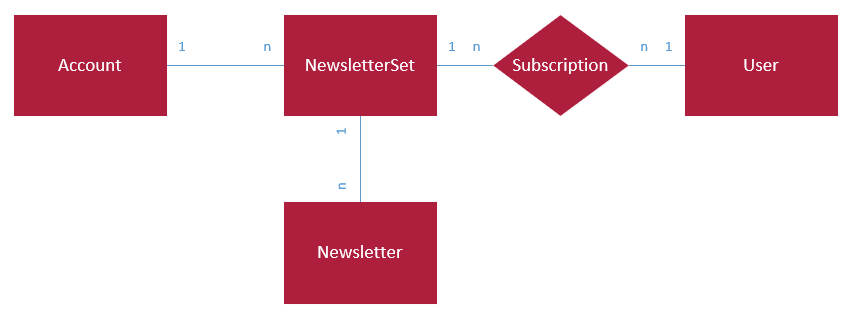


Abbildung 3: Die Hauptentitäten der Newsletter-Anwendung und ihre Beziehungen zueinander. Die Log Entität findet sich hier nicht, da sie keine Beziehung zu einer anderen hat.

Werden die Anwendungsfälle der Newsletter-Anwendung als Nanoservice betrachtet und ihnen die Service-Typen gegenübergestellt (s. Tabelle 3), kann dies als Grundlage für die Einteilung in einzelne Services verwendet werden.

Tabelle 3: Einzelnen Tätitgkeiten der Newsletter-Anwendung.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Anwendungsfall | Service-Typ | Entität | Bemerkung |
| UC1 | Anmelden an NewsletterSet | Entity Service | Subscription |  |
| UC2 | Abmelden von NewsletterSet | Entity Service | Subscription |  |
| UC3 | Newsletterset erstellen | Entity Service | NewsletterSet |  |
| UC4 | Newsletterset bearbeiten | Entity Service | NewsletterSet |  |
| UC5 | NewsletterSet löschen | Entity Service | NewsletterSet |  |
| UC6 | Newsletter erstellen | Entity Service | Newsletter |  |
| UC7 | Newsletter bearbeiten | Entity Service | Newsletter |  |
| UC8 | Newsletter löschen | Entity Service | Newsletter |  |
| UC9 | Versandauftrag in Queue | Utility Service | Subscription |  |
| UC10 | Abarbeiten Queue | Task Service | Subscription |  |
| UC11 | Newsletter Config | Entity Service | Account |  |
| UC12 | Newsletter Statistik | Utility Service | Account, Subscription |  |
| UC13 | Konto Verwaltung | Entity Service | User, Subscription | Div. Konto-Typen |
| UC14 | Services Config | Entity Service | Service |  |
| UC15 | Newsletter Logging | Utility Service | Account, Subscription |  |
| UC16 | Queue Admin | Utility Service |  |  |

Die einfachste Lösung wäre nun, Services anhand der Typen zu definieren. Dann wäre jedoch einerseits eine Vermischung der Domänen zu gross und andererseits wäre mit dem Task Service ein Nanoservice, was gemäss (Rotem-Gal-Oz, 2012) ein Antipattern darstellt.

#### Subscribe Service

Somit werden die Grenzen der Typen durchbrochen werden müssen. Eine Möglichkeit stellt ein Subscribe Service dar, der nur für An- und Abmeldungen zuständig ist. Eine Zuteilung, die diese Art von Service verwendet, findet sich in der nachfolgenden Tabelle.

Tabelle 4: Einteilung unter Verwendung eines Subscribe Services.

|  |  |  |
| --- | --- | --- |
| Service Name | # |  |
| Subscribe | UC1 | Anmelden an NewsletterSet |
| UC2 | Abmelden von NewsletterSet |
| Admin | UC3 | Newsletterset erstellen |
| UC4 | Newsletterset bearbeiten |
| UC5 | NewsletterSet löschen |
| UC6 | Newsletter erstellen |
| UC7 | Newsletter bearbeiten |
| UC8 | Newsletter löschen |
| UC11 | Newsletter Config |
| UC13 | Konto Verwaltung |
| UC14 | Services Config |
| Utility | UC9 | Versandauftrag in Queue |
| UC12 | Newsletter Statistik |
| UC15 | Newsletter Logging |
| UC16 | Queue Admin |

Diese Architektur bietet sich vor allem an, wenn ein grosses Aufkommen an An- und Abmeldungen zu erwarten ist und eine sehr gute Skalierung gewährleistet werden muss. Dieses Bedürfnis kann einerseits von einer hohen Verwendung dieser Newsletter-Anwendung oder von einer weiteren Verwendung für andere Anwendungen kommen. Da beide Punkte bei diesem Projekt in nächster Zeit nicht eintreten dürften, wird eine Aufteilung, wie in Tabelle 5 zu sehen, eher den Bedürfnissen gerecht.

#### Schlussendlich eingesetzte Services

Tabelle 5: Die endgültige Version der Services und ihrer Use Cases.

|  |  |  |
| --- | --- | --- |
| Service Name | # |  |
| Newsletter-Operations | UC1 | Anmelden an NewsletterSet |
| UC2 | Abmelden von NewsletterSet |
| UC3 | Newsletterset erstellen |
| UC4 | Newsletterset bearbeiten |
| UC5 | Newsletterset löschen |
| UC6 | Newsletter erstellen |
| UC7 | Newsletter bearbeiten |
| UC8 | Newsletter löschen |
| UC9 | Versandauftrag in Queue |
| Newsletter-Admin | UC11 | Newsletter Config |
| UC13 | Konto Verwaltung |
| UC14 | Services Config |
| UC16 | Queue Admin |
| UC10 | Abarbeiten Queue |
| Newsletter-Utility | UC12 | Newsletter Statistik |
| UC15 | Newsletter Logging |

Mit dieser Einteilung der Anwendungsfälle sind nun folgende Punkte sichergestellt

* der Utility Service ist nur für lesende Zugriffe zuständig
* der Newsletter-Operations Service ist der Einzige, der die Newsletter und Newsletterset Entitäten liest und schreibt
* Alle weiteren Anwendungsfälle können als Administrationsaufgabe aufgefasst werden und finden sich im passenden Admin Service wieder
* Versandaufträge werden in die Queue gelegt, der Newsletter Admin Service arbeitet diese Queue anschliessend ab, um die Newsletter Mails zu versenden.

Die Zuordnung Service <-> Entität ist in Abbildung 4 zu sehen. Wie ersichtlich ist, greift jeder Service nur auf seine ihm zugeordneten Entitäten zu.

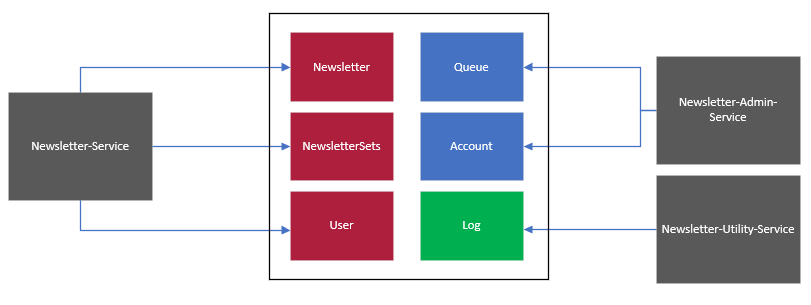


Abbildung 4: Zuordnung der Entitäten zu den Services. Die grauen Kästen sind die Services, die farbigen die Entitäten.

### API-Design

Im Folgenden werden die Designs für einzelne Entitäten aufgeführt. Dabei bedeutet die Aktion create ein Erstellen einer Entität, ein update ein Aktualisieren, ein get entweder der Erhalt einer oder mehrerer Entitäten. Schlussendlich wird mit einem delete eine Entität gelöscht. Mit {id} wird jeweils angedeutet, dass der Identifier der gewünschten Entität mitgegeben werden muss.

#### Newsletter-Sets

Mit diesen Abfragen werden die Newsletter-Alben betreut.

Tabelle 6: URI-Design für die Newsletter-Alben Entität.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http-Methode | URI-Design | Aktion | Response OK | Response nOK |
| POST | /v1/newsletter-sets/ | create | 201 | 422 |
| PUT | /v1/newsletter-sets/{id} | update | 200 | 422 |
| GET | /v1/newsletter-sets/{id} | get | 200 | 404 |
| GET | /v1/newsletter-sets/ | get | 200 | 204 |
| DELETE | /v1/newsletter-sets/{id} | delete | 200 | 404 |

Es wird darauf verzichtet, bei Anfragen einer id, für die keine Berechtigung besteht, einen 403 Response Code zurückzugeben. Dies um keine unnötige (oder je nachdem gefährliche) Informationen mitzuteilen.

#### Newsletters

Mit diesen Abfragen werden die Newsletter-Alben betreut.

Tabelle 7: URI-Design für die Newsletter Entität.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http-Methode | URI-Design | Aktion | Response OK | Response nOK |
| POST | /v1/newsletter/ | create | 201 | 422 |
| PUT | /v1/newsletter/{id} | update | 200 | 422 |
| GET | /v1/newsletter/{id} | get | 200 | 404 |
| GET | /v1/newsletter/ | get | 200 | 204 |
| DELETE | /v1/newsletter/{id} | delete | 200 | 404 |

#### Weitere HTTP-Response-Codes

Im Folgenden die Response-Codes für ausgewählte (fehlerhafte) Abfragen.

|  |  |
| --- | --- |
| HTTP Response Code | Wert |
| 400 | Falsche Abfragedaten |
| 401 | Falls eine Person auf einen nicht autorisierten Service zugreift |
| 403 | Abfragen eines anderen Accounts, auf den keine Zugriffsberechtigungen bestehen |
| 404 | Abfragen z.B. mit fehlerhaften Pfaden |
| 405 | Falls für einen Request eine falsche HTTP-Methode verwendet wird |
| 408 | Falls der Server die Anfrage nicht innerhalb einer definierten Zeitspanne beantworten kann |
| 409 | Wenn vom aktuellen Status zum Zielstatus ein Konflikt entstanden ist |
| 410 | Das Objekt existiert nicht mehr |
| 422 | Fehler beim Erstellen oder aktualisieren einer Entität |
| 429 | Zu viele Requests gesendet |
| 500 | Allgemeiner interner Serverfehler |
| 501 | Falls ein Filter angefragt wird, der für die betroffene Entität nicht implementiert ist |
| 502 | Falls externe Ressourcen einen invaliden Response senden |
| 503 | Maintenance |
| 505 | Falls Abfrage mit HTTP1.0 oder HTTP1.1 eintreffen sollten |

## Abfrage-Beispiele

#### Newsletter-Alben Liste

Mit der in diesem Beispiel gezeigten Abfrage werden alle Newsletter-Alben des verbundenen Accounts aufgelistet.

Tabelle 8: Request Informationen für die Newsletter-Album Auflistung.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Protokoll | HTTP |
| HTTP-Methode | GET |
| Bsp.-URL | https://domain.ch/v1/newsletter-sets/ |

Tabelle 9: Response Informationen für die Newsletter-Album Auflistung.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Datentyp | JSON |
| Wert | {  "newsletter\_sets":  [  {  "id": 1,  "name": "Set 1",  "createdAt":"2021-06-13 04:02:04",  "updatedAt": "2021-10-02 01:25:07",  },  {…}  ]  } |
| HTTP Response Code | 200 |

#### Newsletter-Album Update

Mit diesem Beispiel wird das Newsletter-Album mit der id 1 von «Set 1» auf «wonderful set» umbenannt.

Tabelle 10: Request Informationen für das Newsletter-Album Update.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Protokoll | HTTP |
| HTTP-Methode | PUT |
| Bsp.-URL | https://domain.ch/v1/newsletter-sets/1 |
| Formular-Daten | [name="wonderful set"] |

Tabelle 11: Response Informationen für das Newsletter-Album Update.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Datentyp | JSON |
| HTTP Response Code | 200 |

#### Löschen eines Newsletters

Die Abfrage aus diesem Beispiel löscht den Newsletter mit der id 1 mit all seinen Verbindungen.

Tabelle 12: Request Informationen für das Löschen eines Newsletters.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Protokoll | HTTP |
| HTTP-Methode | DELETE |
| Bsp.-URL | https://domain.ch/v1/newsletters /1 |

Tabelle 13: Response Informationen für das Newsletter-Album Update.

|  |  |
| --- | --- |
| Bezeichnung | Wert |
| Datentyp | JSON |
| HTTP Response Code | 200 |

### Filterung

Jede Abfrage ist auf eine Rückgabe von 1000 Elementen limitiert. Dies heisst wenn eine Abfrage mehr als 1000 Elemente liefert, kommen trotzdem nur 1000 zurück.

Wenn eine Abfrage trotzdem mehr Elemente liefern soll, kann dieser durch den Parameter ?limit=2000 erweitert oder durch ?limit=0 deaktiviert werden.

### Versionierungskonzept

Wie im Kapitel 2.1.2 bei den URI-Designs zu sehen ist, werden den Pfaden /v1 vorangestellt, womit die jeweilige API-Version festgelegt wird. Damit die Versionen zentral verwaltet werden können, werden alle Requests, bevor sie auf den jeweiligen Service treffen, über ein API Gateway geroutet.

# Abgabe Teil 3

## Von Monolithen zu Microservices

Wie bereits in der Einleitung beschrieben, ist die Ausgangslage des Monolithen Concrete5, ein modulbasiertes CMS. In dieser Semesterarbeit liegt der Scope nur bei einem Modul, dem Newslettermodul. Dieses wird, wie bereits in Kapitel 2.1.1 beschrieben, in Services zerlegt. Dies macht in dem Fall sicher Sinn.

DB

Newsletter Modul

Concrete

Abbildung 5: Monolith Concrete 5 CMS. Im bisherigen Anwendungsfall ist das Newslettermodul eng mit dem CMS gekoppelt. Dies gilt es mit dieser Arbeit konzeptuell zu lösen.

In der Theorie gibt es jedoch noch viele andere Möglichkeiten wie man Monolithen zerlegen kann. Jedoch müssen diese zum Anwendungsfall passen.

Mögliche Vorgehensweisen und Varianten sind:

* Strangler Pattern (Teilerneuerung)
* Komplette Neuerstellung vom Code

## Deployment

Das Deployment wird mit Hilfe einer Pipeline realisiert. Die nachfolgenden Stages werden bei jedem Durchlauf der Pipeline durchgeführt. Als Tool für die Pipeline wird Jenkins verwendet.

Abbildung 6: Die vier Pipline Stages des Deployments.

Tabelle 14: Übersicht der CI / CD Stages

|  |  |
| --- | --- |
| Stage | Beschreibung |
| Git Pull Request | Ein Git Pullrequest triggert die Pipeline. Mittels SemVer wird versioniert. So kann gesteuert werden, welche Releases produktiv werden. |
| Build | Im Build Stage wird der Code zusammen mit den notwendigen Artifacts in Container kopiert und das Container Setup anschliessend gestartet. |
| Test | Im Teststage werden statische Codetests, UnitTests sowie die API Integrationstests ausgeführt. |
| Deploy | Erst wenn alle Tests erfolgreich durchgelaufen sind, werden die Container mittels Blue / Green Deployment in die produktive Umgebung deployed. |

### Branching Strategie

Bei diesem Projekt wird ein Git-Repository mit nur einem «statischen» Branch verwendet. Es wird nach dem GitFlow Prinzip gearbeitet. Dies bedeutet, jeder Entwickler erstellt seine eigenen Feature Branches und merged diese dann in den Main Branch. Zusätzlich hat jeder Entwickler die Möglichkeit sich aus seinem Feature Branch mittels SemVer eine Testumgebung seines aktuellen Codes deployen zu lassen. Dabei werden alle Stages ausgeführt, auch alle Tests. Der Unterschied zur Produktion ist jedoch, dass auch bei fehlerhaften Tests deployed wird.

Tabelle 15: Übersicht der Branches

|  |  |
| --- | --- |
| Branchname | Beschreibung |
| Main | Git-Branch für die produktiven Versionen. |
| Dev-feature-xy | Beginnt ein Branch mit dev, wird das komplette Setup deployed, jedoch ohne Abbruch bei fehlerhaften Tests. |

### SemVer

Mittels Semantic Versioning wird gesteuert, welche Pull Requests auf die Produktion deployed werden und welche Releases nur Beta oder Alpha Releases werden, um interne Tests wie z.B. UAT durchzuführen.

Tabelle 16: SemVer Keyword Übersicht

|  |  |
| --- | --- |
| SemVer Keyword | Beschreibung |
| PROD   * Major * Minor * Patch | Beginnt ein Pullrequest mit Prod und wird der Main Branch als Ziel angegeben, werden die Änderungen in die Produktion eingespielt. Dabei wird auch die Versionierung mittels Tags automatisiert vorgenommen.  Zusätzlich werden die Keywords Major, Minor und Patch berücksichtigt. Siehe dazu unten bei Versionierung. |
| DEV | Beginnt ein Commit mit Dev, so wird eine Betaversion erzeugt und die Pipeline angestossen. Dabei wird die Pipeline wie in der Produktion ausgeführt, jedoch auch wenn die Tests nicht sauber durchlaufen. |

Die Versionierung ist in diesem Projekt so definiert, dass Versionen der Produktiven Umgebung anders gehandhabt werden, als Versionen der «eigenen» Dev Umgebung.

Tabelle 17: Versionierunsschema

|  |  |  |
| --- | --- | --- |
| Branch | Versionierungsschema | Beschreibung |
| Main | 1.2.3 | Beim Beispiel links steht die 1 für Major, die 2 für Minor und die 3 für Patch. Je nach dem was ein Entwickler in den Pull-Request schreibt, wird die entsprechende Zahl um eins erhöht. |
| Dev | 1.2.3-beta-<zahl> | Bei den Dev Branches wird die jeweils aktuelle Version gewählt, danach ein -beta- und eine Zahl beginnend bei eins gewählt. Die Zahl wird jeweils bei jedem Commit um 1 erhöht. |

### Build

Da die Codebasis der vorliegenden Webapplikation in PHP geschrieben ist, ist kein eigentlicher Build im Sinne von kompilieren nötig. Stattdessen werden in diesem Stage alle externen Pakete heruntergeladen und im Artefakt Repository gespeichert. Mindestens die letzten 5 Versionen der Pakete bleiben dort verfügbar. Jedoch mindestens so lange, wie sie von einem Build verwendet werden.

Dazu gehören auch die Docker Container, in welche der Sourcecode und die entsprechenden Pakete gepackt werden.

### Deploy

Das Deployment wird automatisch auf einem Kubernetes Cluster vorgenommen.

Jede Version, welche deployed wird, erstellt einen eigenen Pod. Dies ermöglicht den Entwicklern ihre Dev (Beta) Versionen auch mit anderen Entwicklern zu teilen.

Und Transaktionen, welche noch auf der alten Instanz laufen, können ohne Unterbruch zu Ende geführt werden. Beim Deployment auf die Produktion wird mittels Blue / Green deployment gearbeitet. Das heisst, dass es während dem Deployment ein Blue deployment sowie ein Green deployment gibt. Beide stehen hinter einem Loadbalancer.

Graphical user interface, application

Description automatically generated

Abbildung 7: Blue green Deployment

Die neue Version wird parallel deployed und sobald diese lauffähig ist, leitet der Loadbalancer neue Sessions auf die neue Version um. So kann sichergestellt werden, dass alle Jobs und Prozesse noch vollständig zu Ende geführt werden und kein Datenverlust entsteht.

Der Prozess des Deployment ist in Abbildung 8 definiert.

Diagram

Description automatically generated

Abbildung 8: Deployment Prozess

Pro Service wird eine Code Repository und eine Pipeline bereitgestellt. Als Git Repository (Coderepo) wird Github verwendet. Für die Bereitstellung der Pipelines wird Jenkins verwendet. Es werden pro Coderepo zwei Pipelines konfiguriert um das Blue-green deployment abzubilden.

Tabelle 18: Code Repo und Pipelines

|  |  |  |  |
| --- | --- | --- | --- |
| Service | Coderepo | Pipeline | Branch |
| Newsletter-Service | **newsletter/service** | **newsletter/service-prod1**  **newsletter/service-prod2** | **main**  **main-pre** |
| Newsletter-Admin-Service | **newsletter/admin** | **newsletter/admin-prod1**  **newsletter/admin-prod2** | **main**  **main-pre** |
| Utility Service | **newsletter/utility** | **newsletter/utility-prod1**  **newsletter/utility-prod2** | **main**  **main-pre** |

## Test

In diesem Projekt werden drei Arten von Tests durchgeführt. Nachfolgend sind diese in der Testpyramide abgebildet (Abbildung 9). Je höher sich ein Test in der Pyramide befindet, desto höher ist der Aufwand.

E2E Tests

API & Integrationstests

Unit Tests / Statische codetests

Abbildung 9: Testpyramide

### Unittests

Das Ziel ist eine 100% Codeabdeckung mit Unit Tests. Die Unittest Abdeckung wird vom Tool Sonarqube kontrolliert. Sinkt die Abdeckung unter 95%, bricht die Pipeline ab. Dies zwingt die Entwickler aktiv an einer 100% Abdeckung zu arbeiten. Als Unit- und Integrationstestframework wird PHPUnit verwendet. Die folgenden Testfälle sind aufgrund des Umfangs nicht abschliessend beschrieben. Als Beispiel wird der UC13 (Konto Verwaltung) des Newsletter-Admin Service verwendet.

Tabelle 19: Testliste der Unittests

|  |  |
| --- | --- |
| Unittest | Beschreibung |
| createUser | Es wird geprüft, ob die Methode createUser korrekt funktioniert, die Attribute korrekt übergibt. |
| getUser | Es wird geprüft, ob die Methode getUser korrekt funktioniert, die Attribute korrekt übergibt. |
| updateUser | Es wird geprüft, ob die Methode updateUser korrekt funktioniert, die Attribute korrekt übergibt. |
| removeUser | Es wird geprüft, ob die Methode removeUser korrekt funktioniert, die Attribute korrekt übergibt. |

### API Integrationstests

Für die API Integrationstest steht eine separate Datenbank mit Beispielen zur Verfügung. Mit diesen Beispielen können alle REST Funktionen getestet werden. Bei den API-integrationstests werden alle API Endpunkte getestet

Der Ablauf der Tests läuft für jeden Service und jede Funktion grösstenteils gleich ab.

Tabelle 20: Testablauf REST-API Integrationstests

|  |  |  |
| --- | --- | --- |
| No | Test | Beschreibung |
| 1 | POST | Die Daten aus der DB werden via REST-API call (POST) ins System erfasst |
| 2 | GET | Die erfassten Daten werden via REST-API call (GET) abgefragt. Es wird dabei geprüft, ob das korrekte Resultat zurückkommt. |
| 3 | PUT | Die Daten werden mittels (PUT) modifiziert. Anschliessend mit (GET) abgefragt ob die Änderung angepasst wurde. |
| 4 | GET | Die geänderten Daten werden via REST-API call (GET) abgefragt. Es wird dabei geprüft, ob das korrekte geänderte Resultat zurückkommt. |
| 5 | DELETE | Einige Daten werden mit (DELETE) gelöscht. Anschliessend wird mit (GET) geprüft, ob die Daten auch gelöscht wurden. |

### Statische Codetests

Für die Codetests wird Sonarqube verwendet. Sonarqube arbeitet mit sogenannten Quality Gates. Diese können frei konfiguriert werden. Sinkt die Qualität eines Quality Gates unter einen bestimmten Wert, bricht die Pipeline ebenfalls ab, was die Behebung der jeweiligen Probleme im Code erfordert.

### End 2 End Tests

Für End2End Tests wird das Robot Framework eingesetzt. Das Robot Framework bietet eine Art Abstraktionslayer für End-2-End Tools. Mit dem Robot Framework können mit der gleichen Syntax die wichtigsten End-2-End Tools angesprochen werden. Zum Einsatz kommen folgende Tests:

Tabelle 18: Testliste der End 2 End Tests

|  |  |
| --- | --- |
| Test | Beschreibung |
| Admin Login | Kann sich ein Admin anmelden und stehen ihm die Admin Menüpunkte zur Verfügung. Anschliessendes Log-out. |
| Operator Login | Kann sich ein Operator anmelden und stehen im die Operator Menüpunkte zur Verfügung. Anschliessendes Log-out. |
| Newsletter Versand | Der Test versendet einen Newsletter an eine Mailbox und prüft, ob dieser dort ankommt. |
| API | Login auf die API-Dokumentation und prüfen, ob Inhalt vorhanden ist. |

Natürlich könnte diese Liste noch endlos weitergetrieben werden, dies würde jedoch Rahmen dieses Projekts sprengen.

### Security Tests

Für die Security Tests wird der Apache Zapp Proxy eingesetzt. Mit dem Zapp Proxy werden hauptsächlich die OWASP Top 10 getestet (OWASP, 2021).

### Performance und Betriebstests

Die REST-API jedes Services exponiert zusätzlich auch interne Performance Metriken des eigenen Services. Diese Metriken werden anschliessend durch Prometheus abgeholt und in Grafana dargestellt.

Tabelle 19: Abfragemetriken zur Überwachung der Applikation

|  |  |
| --- | --- |
| Metrik | Beschreibung |
| CPU | Die CPU-Auslastung jedes einzelnen Containers |
| Memory | Die Memory Auslastung jedes einzelnen Containers |
| Disk I/O | Die Anzahl der Disk I/O pro Container |
| Anfragen | Die Anzahl der Anfragen pro Sekunde an den Service |
| Fehlerhafte Anfragen | Anfragen, welche fehlerhaft sind. Diese sind insofern wichtig, um Probleme nachzuvollziehen und zu isolieren |
| Traffic | Wie viel Traffic die Requests der Applikation generieren |
| Versandte Mails | Die Anzahl der versandten Mails |
| Fehlerhafte Mails | Um Probleme beim Mailversand zu erkennen |
| Service Status | Der Status der verschiedenen Services |
| Zeit | Die aktuelle Zeit jedes Services, um Zeitunterschiede oder Probleme bei der Zeitsynchronisierung zu erkennen |
| LoadBalancer | Welches deployment auf dem Loadbalancer aktiv ist |
| Sessions | Wie viele Sessions vorhanden sind |
| Fehlerhafte Logins | Um zu erkennen, wenn Bruteforce Attacken stattfinden. Oder zum Nachvollziehen, wenn sich ein Benutzer falsch anmeldet |

TODO: Service Instanzen / Kapazitäten / Placement

# Abgabe Teil 4

Die Überwachung und Security einer Applikation wird oft unterschätzt. Die beiden Themen sind jedoch essenziell für einen sicheren Betrieb einer Applikation. Mittels einer sauberen Umsetzung können Störungen früh erkannt und böswillige Angriffe auf die Applikation verhindert werden.

## Monitoring

Dieses Thema hat einen engen Zusammenhang mit Performance Daten, welche im in den Kapiteln 3.3.5 und 3.3.6 beschrieben. Als Monitoring Lösung und Tool Set wird eine Kombination aus fünf unterschiedlichen Systemen eingesetzt.

Tabelle 23: Monitoring Systeme

|  |  |
| --- | --- |
| System | Zweck |
| CheckMK | Systemmonitoring und Übersicht  In CheckMK werden alle Dashboards und die Gesamtübersicht aus allen 4 Tools integriert. |
| Prometheus | Applikationsmonitoring zur Überwachung der Applikation und auswertung der Performance Counter. |
| Graylog | Logging Management für Syslog und Eventmeldungen |
| Robot Framework | End 2 End Monitoring mit direkter Integration in CheckMK. Konfiguration und Deployment der Checks direkt im CheckMK möglich. |
| Zapp Proxy | Security Tests und Penetration |

### CheckMK als Drehscheibe

CheckMK bietet die Möglichkeit, die Umsysteme wie Prometheus, Graylog oder Robot Framework zu integrieren. Auch die Pipeline von Jenkins oder das Coderepo kann mittels CheckMK überwacht werden und bietet deshalb eine perfekte Übersicht über die ganze «System und App Landschaft». Um Probleme tiefer zu debuggen, eignet es sich nicht, dazu sind dann Umsysteme wie Prometheus oder Graylog, welche detailliertere Möglichkeiten bieten. CheckMK bietet lediglich den Gesamtüberblick.

A screenshot of a computer

Description automatically generated with medium confidence

Abbildung 10: CheckMK Dashboard

### Prometheus

Mit Prometheus gelingt es, Daten aus der Applikation abzufragen. Dazu werden diese in der Applikation zur Verfügung gestellt, um eine Abfrage «Scraping» durch Prometheus zu ermöglichen.

### Graphical user interface Description automatically generated

Abbildung 11: Prometheus Dashboard

Folgende Messwerte werden mindestens abgefragt. Zum Teil ergänzend zu Tabelle 19: Testliste der Unittests.

**System (Container)**

* CPU usage
* Average system load
* Disk usage
* Memory usage
* Network traffic

**Webserver**

* Requests / Responses
* Traffic
* Threads
* Errors
* Memory Buffer

**Mailserver**

* Sent mails
* Received mails
* Mails with errors
* Returned mails
* Mail error code
* Uptime
* Job counter
* Mail Blacklist checks

**Applikation generell**

* Uptime
* http Requests
* Status (mit Mapping auf healthy / unhealthy)
* Log counter (Ahzahl Logs pro Event)
* API Calls (pro Endpunkt ein Counter)
* Scrape duration

**Weiterführende Metriken (CheckMK oder Prometheus)**

* Container Status
* Container Host / Status / Cluster Auslastung
* Pipeline Status
* Container orchestrierung Status
* Netzwerk verfügbarkeit
* Service Verfügbarkeiten
* Loadbalancer status

### Graylog

Mit Graylog werden Lognachrichten gesammelt und können später ausgewertet und korreliert werden. Bei Graylog und Log management geht dies auch bereits in Richtung Security. Bei der Konfiguration von Graylog muss beachtet werden, dass nicht Daten, welche in Prometheus gesammelt werden in Graylog doppelt konfiguriert werden.

Graphical user interface, application

Description automatically generated

Abbildung 12: Graylog Dashboard

### Robot Framework

Mit dem Robot Framework werden End 2 End Tests (später E2E genannt) durchgeführt. Diese können bequem mittels YAML beschrieben werden. Das Robot Framework bietet die Möglichkeit, dass die bekanntesten Library’s für E2E Testing (Selenium / Sikuli / Appium / Browser uvm. ) mittels eines Frameworks und somit einem einheitlichen Syntax angesprochen werden können.

Für die E2E Tests werden Roboter (Client VM’s oder Fatclients) bereitgestellt. Um die Tests auszuführen, wird die nötige Software darauf installiert.

Da E2E Tests zeitaufwendig und eher teuer sind, werden jeweils nur die Funktionen getestet, welche am businesskritischsten sind.

Folgende E2E Tests sind in diesem Projekt vorgesehen:

Tabelle 24: End 2 End Tests Übersicht

|  |  |
| --- | --- |
| Testkategorie | Beschreibung |
| Authentisierung | Die Authentisierung wird mittels End2End Web UI Test automatisiert geprüft. Dieser Check prüft, ob ein Benutzer sich an der Applikation sowie an der API authentisieren kann. |
| Autorisierung | Diese Tests prüfen, ob ein Benutzer mit der Rolle Admin im Admin Service Berechtigungen erhält und entsprechende Menüpunkte im GUI sowie auf der API aufrufen kann. Im Umkehrschluss, ob ein normaler Benutzer ohne Admin Rolle dies nicht kann. |
| API | Die API wird bereits mittels Integrationstests geprüft. Mit diesem E2E Test wird geprüft, ob die API-Dokumentation aufgerufen und korrekt angezeigt wird. |
| Mailversand | Diese End2End Tests versenden E-Mails an einen bestimmten Absender und prüfen im Postfach, ob genau diese Mails ankommen. |
| Admin Service | Diese Tests prüfen, ob die wichtigsten Funktionen des Admin Services funktionieren und führen diese im Web UI automatisiert durch. |
| Utility Service | Diese Tests prüfen, ob die wichtigsten Funktionen des Utility Services funktionieren und führen diese im Web UI automatisiert durch. |
| Newsletter Service | Diese Tests prüfen, ob die wichtigsten Funktionen des Newsletter Services funktionieren und führen diese im Web UI automatisiert durch. |

## Security

Die Sicherheit hat bei einer Applikation, wie hier beschrieben einen sehr hohen Stellenwert. Gründe dafür sind:

* Die Applikation befindet sich direkt im Internet und bietet deshalb sehr viel Angriffsfläche
* Die Applikation beinhaltet Kunden- und Endkundendaten

### Zapp Proxy

Mit dem Zapp Proxy werden automatisiert Sicherheitstests der Applikation durchgeführt. Hierbei werden mindestens die OWASP Top 10 automatisiert überprüft.

### Pentests

Bei jedem Minor Release werden zusätzlich noch einen manuellen Pentest durchgeführt. Die Erkenntnisse aus dem Pentest werden anschliessend in einem Zapp Proxy Test konfiguriert und abgebildet. Somit wachsen die Zapp Proxy Konfiguration und dessen Tests stetig auch mit dem Produkt.

### Andere Sicherheitsmassnahmen

Zusätzlich zu den automatisierten Security Tests werden andere Massnahmen wie eine sichere Mailkonfiguration oder auch Massnahmen zur sicheren Veröffentlichung der Applikation vorgesehen.

#### Mail Konfiguration

Beim Massenversand von Newslettern sind einige Dinge zu beachten, um nicht auf eine Black List zu gelangen. Z.B. dass der Mailversand gestaffelt veranlasst wird. (100 Mails pro Minute)

Auch sollen die DNS-Konfiguration für Mail (DMARC / MX Records und SOA) nach best practise konfiguriert werden, um einen stabilen Betrieb sicherzustellen. Um die Applikation resp. die Testsysteme vor eingehender Traffic (SPAM) zu schützen, wird ein SPAM-Filter (Mail Gateway mit SPAM-Filter) vor den Mailserver gestellt. Dieser verhindert unnötigen Mailtraffic.

#### WAF (Web Application Firewall)

Da die Applikation direkt im Internet steht wird nebst einem Loadbalancer noch eine WAF vorgesehen. Diese schützt die Applikation vor diversen Angriff Szenarien z.B. OWASP Top 10, ohne dass die Applikation einen internen Schutz gegen diese Angriffe verfügt.

#### DDOS Schutz

Dort wo die Applikation betrieben wird, verfügt über einen DDOS Schutz. Um bei DDOS Attacken den Service weiterhin zu gewährleisten.

#### Scaleout

SOA Applikationen bieten nicht per se, dass sie skalierbar sind. Jedoch ist die beschriebene Applikation, dank der REST APIs skalierbar bleiben.

Dies heisst, je mehr der Service belastet wird, skaliert die Applikation automatisch.

Auch ist es möglich, automatisch zusätzliche Mailserver bereitzustellen, falls dies nötig ist.

#### Hochverfügbarkeit

Die Applikation könnte per Design hochverfügbar gemacht werden. Dies heisst, jeder Service ist mindestens doppelt vorhanden. Sollte ein Teil des Service ausfallen, übernimmt ein anderer Teil des Services ohne dass der Endbenutzer etwas davon bemerkt.

# Abgabe Teil 5

# Literaturverzeichnis

Meister, S., & Plüss, T. (05. 09 2021). *1stthomas / ffhs-soa-work*. Abgerufen am 05. 09 2021 von GitHub: https://github.com/1stthomas/ffhs-soa-work

Meister, S., & Plüss, T. (05. 09 2021). *PVA 1 - KickOff*. Abgerufen am 05. 09 2021 von GitHub: https://github.com/1stthomas/ffhs-soa-work/projects/1

PortlandLabs. (2. 9 2021). *Open Source Content Management System for Teams*. (PortlandLabs, Herausgeber) Abgerufen am 2. 9 2021 von concretecms.com: https://www.concretecms.com/

Rotem-Gal-Oz, A. (2012). Nanoservice antipattern. In A. Rotem-Gal-Oz, *SOA Patterns* (S. 195-202). Shelter Island, NY 11964: Manning. Abgerufen am 1. 10 2021

# Abbildungsverzeichnis

[Abbildung 1: CMS wie ConcreteCMS bearbeiten sehr oft auch Anfragen, die Rechner intensiv sein können, aber mit den Kernaufgaben nichts zu tun haben. 4](#_Toc86527132)

[Abbildung 2: Ausgliederung der Newsletter Funktionen in einen eigenen Service, wobei links der Hosting- und rechts der Service-Server ist. 5](#_Toc86527133)

[Abbildung 3: Die Hauptentitäten der Newsletter-Anwendung und ihre Beziehungen zueinander. Die Log Entität findet sich hier nicht, da sie keine Beziehung zu einer anderen hat. 11](#_Toc86527134)

[Abbildung 4: Zuordnung der Entitäten zu den Services. Die grauen Kästen sind die Services, die farbigen die Entitäten. 15](#_Toc86527135)

[Abbildung 5: Monolith Concrete 5 CMS 19](#_Toc86527136)

[Abbildung 6: Pipline stages 19](#_Toc86527137)

[Abbildung 7: Blue green Deployment 22](#_Toc86527138)

[Abbildung 8: Deployment Prozess 22](#_Toc86527139)

[Abbildung 9: Testpyramide 23](#_Toc86527140)

# Tabellenverzeichnis

[Tabelle 1: Meilensteine des Projekts. 9](#_Toc86527141)

[Tabelle 2: Hauptentitäten der Newsletter-Anwendung. 11](#_Toc86527142)

[Tabelle 3: Einzelnen Tätitgkeiten der Newsletter-Anwendung. 12](#_Toc86527143)

[Tabelle 4: Einteilung unter Verwendung eines Subscribe Services. 13](#_Toc86527144)

[Tabelle 5: Die endgültige Version der Services und ihrer Use Cases. 14](#_Toc86527145)

[Tabelle 6: URI-Design für die Newsletter-Alben Entität. 15](#_Toc86527146)

[Tabelle 7: URI-Design für die Newsletter Entität. 16](#_Toc86527147)

[Tabelle 8: Request Informationen für die Newsletter-Album Auflistung. 17](#_Toc86527148)

[Tabelle 9: Response Informationen für die Newsletter-Album Auflistung. 17](#_Toc86527149)

[Tabelle 10: Request Informationen für das Newsletter-Album Update. 17](#_Toc86527150)

[Tabelle 11: Response Informationen für das Newsletter-Album Update. 18](#_Toc86527151)

[Tabelle 12: Request Informationen für das Löschen eines Newsletters. 18](#_Toc86527152)

[Tabelle 13: Response Informationen für das Newsletter-Album Update. 18](#_Toc86527153)

[Tabelle 14: Übersicht der CI / CD Stages 20](#_Toc86527154)

[Tabelle 15: Übersicht der Branches 20](#_Toc86527155)

[Tabelle 16: SemVer Keyword Übersicht 21](#_Toc86527156)

[Tabelle 17: Versionierunsschema 21](#_Toc86527157)

[Tabelle 18: Testliste der End 2 End Tests 25](#_Toc86527158)

[Tabelle 19: Abfragemetriken zur Überwachung der Applikation 26](#_Toc86527159)

1. Vor 2021 hiess das CMS concrete5, weshalb im weiteren Verlauf des Dokuments die weitläufig bekannte Abkürzung c5 verwendet wird. [↑](#footnote-ref-1)
2. Die vom zusätzlichen Service benötigte Oberfläche kann auch, anders als hier im Beispiel dargestellt, vom Service selbst angeboten werden. [↑](#footnote-ref-2)
3. In dieser Arbeit wird nur die RESTful API-Technologie verwendet, weshalb im weiteren Verlauf der Arbeit nur noch von REST Requests geschrieben wird. [↑](#footnote-ref-3)