



**REALOBJECTS**

Cross Platform Software

## **Implementierung einer Subscription-Management-Anwendung auf Basis von Amazon Web Services**

Auszubildender:	Thomas Eppers
Ausbildungsberuf:	Fachinformatiker/Anwendungsentwicklung
Ausbilder:	Dipl.-Ing. (FH), Dipl.-Wirt.-Ing. (FH) Michael Jung
Projektbetreuer:	Max Altmeyer, Fachinformatiker
Ausbildungsbetrieb:	RealObjects GmbH
Prüfungstermin:	Winter 2018/19

# Inhalt

<b>1. Einleitung</b>	1
1.1 Projektbeschreibung	1
1.2 Projektbegründung	1
1.3 Projektziel	2
1.4 Projektumfeld	2
1.5 Projektschnittstellen	2
1.6 Projektabgrenzung	2
<b>2. Projektplanung</b>	3
2.1 Ressourcenplanung	3
2.2 Zeitplanung	3
2.3 Entwicklungsprozess	3
<b>3. Analyse</b>	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
Projektkosten	4
Amortisationsdauer	5
3.3 Datenstrukturanalyse	5
3.4 Anwendungsfälle	6
3.5 Lastenheft	6
<b>4. Konzeption</b>	6
4.1 Zielplattform	6
4.2 Datenbankmodell	7
4.3 Ablaufmodell	7
4.4 Cloud-Modell	8
4.5 REST-API-Konzeption	8
4.6 Pflichtenheft	8
<b>5. Durchführung</b>	8
5.1 Implementierung der Datenstruktur	8
5.2 Implementierung des API-Gateways	8
5.3 Implementierung der Lambda-Funktionen	9
E-Commerce-Endpoint-Verarbeitung	9
Account- und Subscription-Management	10
5.4 Implementierung der Benutzeroberfläche	11
<b>6. Qualitätssicherung</b>	12
6.1 Modultests	12
Webservice	12
Benutzeroberfläche	13
6.2 Integrationstest	13
6.3 Datenintegrität	13
<b>7. Dokumentation</b>	13
7.1 Kundendokumentation	13
7.2 Entwicklerdokumentation	14
<b>8. Übergabe</b>	14
<b>9. Fazit</b>	14
9.1 Soll-/Ist-Vergleich	14

9.2 Ausblick .....	15
9.3 Subjektives Fazit .....	15
<b>Glossar</b> .....	I
<b>Quellenangaben</b> .....	III
<b>Anhang</b> .....	IV
A.1 Ressourcen .....	IV
A.2 Zeitplanung .....	V
A.3 Lastenheft .....	VI
A.4 Use-Case-Diagramm .....	IX
A.5 Purchase Model .....	X
A.6 Aktivitätsdiagramm .....	XI
A.7 Cloudmodell .....	XII
A.8 Datenbankschema .....	XIII
A.9 Pflichtenheft .....	XIII
A.10 Konfiguration für den Amazon Cognito Benutzerpool .....	XVI
A.11 AngularJS Modulkonfiguration .....	XVIII
A.12 AngularJS Cogntito-Service .....	XIX
A.13 Kundendokumentation .....	XX
A.14 Entwicklerdokumentation .....	XX
A.15 Ordnerstruktur der statischen HTML-Webseite .....	XXVIII
A.16 Benutzeroberfläche .....	XXIX

# Abkürzungsverzeichnis

**API** Application Programming Interface, Programmierschnittstelle

**AWS** Amazon Web Service

**CSS** Cascading Style Sheets

**ECD** E-Commerce-Dienstleister

**HTML** Hypertext Markup Language

**HTTP** Hypertext Transfer Protocol, Protokoll zur Übertragung von Daten

**JSON** JavaScript Object Notation

**MVC** Model View Controller

**PDF** Portable Document Format

**REST** Representational State Transfer, ein Programmierparadigma für Webanwendungen

**SaaS** Software-as-a-Service

**SAM** Serverless Application Model

**SVN** Subversion, freie Software zur zentralen Versionsverwaltung von Dateien und Verzeichnissen

**URL** Uniform Resource Locator

**XML** Extensible Markup Language

# 1. Einleitung

Die RealObjects GmbH befasst sich seit dem Jahr 2000 mit der Entwicklung und dem Vertrieb von plattformunabhängigen Softwareprodukten zum Editieren und Konvertieren von Webinhalten. Zu den Produkten zählen unter anderem „PDFreactor“, eine 2006 veröffentlichte Anwendung zur Konvertierung von HTML- und XML- in PDF-Dokumente, sowie der Webeditor „Nimbudocs Editor“. Die RealObjects GmbH verzeichnet heute einen Kundenbestand von über 3000 Unternehmen und Institutionen aus mehr als 40 Ländern, welche die Softwareprodukte der RealObjects GmbH nutzen. Derzeit beschäftigt die RealObjects GmbH 12 Mitarbeiter<sup>1</sup> und hat ihren Sitz in Saarbrücken.

## 1.1 Projektbeschreibung

Die RealObjects GmbH vertreibt ihre Produkte aktuell über den Verkauf von Lizenzen für ihre Softwareprodukte. Der Verkauf von Softwarelizenzen für PDFreactor wird unter anderem über einen externen E-Commerce-Dienstleister (ECD) abgewickelt, welcher die RealObjects GmbH nach erfolgreicher Bezahlung durch den Kunden per E-Mail benachrichtigt. Hiernach wird die Kunden- und Lizenzverwaltung von Mitarbeitern manuell durchgeführt.

Das Produktportfolio der RealObjects GmbH soll durch die Nutzung ihrer Produkte als "Software-as-a-Service" (SaaS) erweitert und die Geschäftsprozesse durch Automatisierung optimiert werden. In diesem Projekt soll ein System erstellt werden, das es ermöglicht, diesen Prozess auf ein Subscription<sup>2</sup>-Modell umzustellen und weitestgehend zu automatisieren.

## 1.2 Projektbegründung

Die Schwachstelle des momentanen Lizenzierungsprozesses ist das hohe Maß an manueller Arbeit, wodurch eine Umstellung auf einen agileren Geschäftsprozess (wie ein Subscription-Modell) nicht möglich ist. Dies beruht auf der Tatsache, dass der hierdurch ausgelöste zu erwartende, periodisch wiederkehrende Verwaltungsaufwand in erheblich größerem Maße anwächst, als es der bisherige Geschäftsprozess bezüglich der vorhandenen personellen Ressourcen zulässt.

Außerdem ist im bestehenden Prozess eine Erzeugung oder Änderung einer Lizenz nur zu Geschäftszeiten möglich, was die Anforderungen von Kunden bezüglich Verfügbarkeit (24 Stunden am Tag und sieben Tage die Woche) und Reaktionszeit eines SaaS-Produktes nicht erfüllt. Ebenso wird derzeit durch das manuelle Pflegen der Kundenkommunikation und des Kundenverwaltungssystems die Arbeitskraft mindestens eines Mitarbeiters erheblich gebunden.

---

<sup>1</sup> Kennzahl am Stichtag 20.11.2018

<sup>2</sup> Subscription (engl.) - Abonnement

## 1.3 Projektziel

Ziel des Projektes ist die Erstellung eines Subscription-Management-Systems, bei dem Kunden ihre Subscriptions, welche bei einem ECD erworben wurden, selbständig verwalten sowie den zur Subscription zugehörigen API-Schlüssel abfragen können.

Weiterhin soll eine Schnittstelle für ECD geboten werden, um Verkaufsdaten zu übermitteln sowie Daten abzufragen. Außerdem, sollen aus den übermittelten Daten Subscriptions erstellt und gegebenenfalls bekannten Kunden zugeordnet werden können.

## 1.4 Projektumfeld

Auftraggeber des Projektes ist die RealObjects GmbH selbst. Das Unternehmen ist daran interessiert, durch die Erweiterung des Produktportfolios um eine Lizenzierung ihrer Softwareprodukte über ein Subscription-Modell neue Kunden zu gewinnen sowie durch die Automatisierung des Lizenzvergabeprozesses die Geschäftsprozesse zu optimieren.

## 1.5 Projektschnittstellen

Um auf Daten zugreifen zu können, zu validieren und zu übermitteln wird ein RESTful Webservice in der Amazon Web Services (AWS) Umgebung mit Hilfe eines Amazon API Gateways realisiert.

Die Ressourcen des Webservices verweisen auf Funktionen in AWS Lambda, welche die Logik des Services beinhalten und sowohl auf eine Amazon DynamoDB Instanz als auch auf einen Amazon Cognito User Pool zugreifen, welche alle relevanten Daten beinhalten.

Die Web-Anwendung, welche es ermöglichen soll Subscriptions zu verwalten, soll via REST API ebenfalls auf die Datenbank und den Cognito User Pool zugreifen können. Das erstellte API Gateway, welches durch API- und Session-Schlüssel isoliert ist, dient somit als Bindeglied zwischen Endgerät und Web Service, der AWS-Dienst Cognito dient als Schlüsselgeber.

Als Vermittler zwischen allen Amazon Web Service Diensten wird der Dienst "IAM" von Amazon genutzt, mit dem der Integrator jedem Dienst eine Rolle und die damit verfügbaren Befugnisse zuweist.

Für eine bessere Übersicht der Komponenten findet sich im Anhang (Seite XII) ein Modell der Cloud-Komponenten zur Übersicht.

## 1.6 Projektabgrenzung

Die Generierung und Auswertung der API-Schlüssel, mit dem ein Produkt die lizenzierte Nutzung validiert, ist nicht Teil dieses Projektes, da dies den Umfang von 70 Stunden übersteigen würde. Weiterhin ist durch die Verwendung des Amazon "Serverless Application Models" das Einrichten und

Verwalten eines Servers nicht Teil des Projektes. Serverseitig liegt daher der Schwerpunkt auf der Implementierung der Logik und der Konfiguration der Dienste.

## 2. Projektplanung

### 2.1 Ressourcenplanung

In der Übersicht, welche sich im Anhang (Seite IV) befindet, sind alle Ressourcen aufgelistet, die für das Projekt eingesetzt wurden. Damit sind vor allem Hard- und Softwareressourcen gemeint. Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese nach Möglichkeit kostenfrei zur Verfügung steht oder die RealObjects GmbH bereits Lizenzen für diese besitzt. Dadurch sollten anfallende Projektkosten möglichst gering gehalten werden.

### 2.2 Zeitplanung

Zur Durchführung des Projektes standen 70 Stunden zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Die Zeitplanung als Übersicht wurde folgendermaßen festgelegt:

Teil	Zeitaufwand/h
Analyse	6
Konzeption	8
Entwicklung	36
Tests	10
Dokumentation	10
Gesamt	70

Eine detailliertere Auflistung findet sich im Anhang (Seite V).

### 2.3 Entwicklungsprozess

Vor der Realisierung des Projektes, musste sich der Autor für einen geeigneten Entwicklungsprozess entscheiden, welcher die Vorgehensweise definiert, nach der die Umsetzung erfolgt. Für das Abschlussprojekt wurde das Wasserfallmodell als Entwicklungsprozess gewählt, das ein lineares Vorgehensmodell in der Softwareentwicklung ist, bei dem der Softwareentwicklungsprozess in einzelnen, festen Teilen organisiert wird. Dabei gelten die Ergebnisse der einzelnen Teile immer als bindende Vorgaben für den jeweils nächsten. Jeder Teil muss komplett abgeschlossen sein, damit der nächste Teil beginnen kann.

Durch die logische Struktur des Modells können konzeptionelle Fehler häufig vermieden werden. Zuzüglich führt das Modell zu umfangreichen, technischen Dokumentationen, die für neue Entwickler eine Erleichterung darstellen und zudem in der Testphase hilfreich sind.

### 3. Analyse

#### 3.1 Ist-Analyse

Wie bereits im Abschnitt "Projektbeschreibung" (Seite 1) erläutert wurde, verläuft das derzeitige Lizenzmanagement größtenteils manuell. Nach der E-Mail-Benachrichtigung durch den ECD wird der Kunde, falls er derzeit dem System noch nicht bekannt ist, in einem internen Kunden-Lizenz-Management-System (CLM) von einem Mitarbeiter angelegt und die Lizenzdatei erzeugt. Dieser Lizenzschlüssel wird dem Kunden wiederum über das CLM per E-Mail zukommen gelassen.

Die so ausgelieferten Lizenzschlüssel gelten für je einen Server mit einer bestimmten Anzahl von CPU-Kernen ohne zeitliche Begrenzung für die ihr jeweils zugeordnete Hauptversionsnummer von PDFreactor. Jegliche Anpassung dieser Lizenz muss nach Eingang einer entsprechenden Bestellung manuell von einem Mitarbeiter durchgeführt werden. Die Einführung eines Subscription-Modells ist demnach derzeit nicht praktikabel.

#### 3.2 Wirtschaftlichkeitsanalyse

##### Projektkosten

Im Folgenden werden die für das Projekt anfallenden Kosten kalkuliert. Berücksichtigt werden primär Personalkosten, da die benötigten Sachmittel, sowohl Hard- als auch Software, bereits unabhängig vom Projekt zur Verfügung standen.

Vorgang	Mitarbeiter	Zeit/h	Kosten/€	Bemerkungen
Entwicklung	1 × Auszubildender	70	417,90	Projektleiter
	1 × Mitarbeiter	70	994,00	Kompensation Wegfall Projektleiter
Datenstruktur-analyse	2 × Mitarbeiter	2	56,80	
Code-Review	1 × Mitarbeiter	2	28,40	
Abnahme	2 × Mitarbeiter	2	56,80	
<b>Gesamt</b>			<b>1.553,90</b>	



Hierbei wurden folgende Annahmen gemacht:

- 22 Arbeitstage pro Monat
- 8 Arbeitsstunden pro Arbeitstag
- Bruttolohn zzgl. Sozialleistungen pro Monat:
  - Auszubildende: 1.050,00 €
  - Mitarbeiter: 2.500,00 €

Die monatlichen Kosten für die Amazon Services sind aufgrund des geringen Datenvolumens sowie extrem niederfrequente HTTP-Anfragen so gering, dass sie hierbei vernachlässigt werden.

### Amortisationsdauer

Bei der Berechnung der Amortisationsdauer kann vorerst nur der Zeitgewinn des Mitarbeiters, der die Kunden- und Lizenzverwaltung übernimmt, herangezogen werden. Etwaige Rückflüsse in Form von zusätzlich generierten Kunden aufgrund der attraktiveren und agileren Lizenzierungsform werden vorerst nicht berücksichtigt.

Folgende Annahmen werden getroffen:

- Die Verwaltung einer Lizenz nimmt in der Regel ca. 15 Minuten in Anspruch.
- Durchschnittlich werden ca. 28 Verwaltungsvorgänge im Monat durchgeführt.
- Durch Einführung des Systems reduziert sich diese Zeit auf 0.

Somit wird davon ausgegangen, dass ein Mitarbeiter im Monat ca. sieben Stunden mit Verwaltungstätigkeit beschäftigt ist. Bei dem im Abschnitt Projektkosten (Seite 4) angenommenen Bruttolohn ergeben sich monatliche Kosten bzw. Einsparpotenzial von 99,40 €. Daraus folgt eine Amortisationszeit von:

$$\Delta = \frac{\text{Projektkosten}}{\text{Monatskosten}} = \frac{1553.90 \text{ €}}{99.40 \frac{\text{€}}{\text{Monat}}} = 15.63 \text{ Monate.}$$

Also wird sich das Projekt spätestens nach einem Jahr und vier Monaten amortisiert haben.

### 3.3 Datenstrukturanalyse

Aus vom ECD zur Verfügung gestellten Mock-Up-Benachrichtigungen sowie der Analyse der Geschäftsprozesse wurde auf ein Tupel von Daten geschlossen, welches zum reibungslosen Ablauf von Geschäftsbeziehungen unbedingt gebraucht wird. In Rücksprache mit Mitarbeitern und Geschäftsleitung wurde beschlossen, nur ein Minimum von Daten zu persistieren und vor allem die Verwaltung persönlicher Daten dem ECD zu überlassen und diese nur im Bedarfsfall abzufragen.

## 3.4 Anwendungsfälle

Um eine grobe Übersicht über die abzudeckenden Anwendungsfälle zu erhalten, wurde im Zuge der Analysephase ein Use-Case-Diagramm erstellt. Hierbei wurden die betroffenen Akteure identifiziert und deren Anforderungen an das Projekt ermittelt. Das Use-Case-Diagramm ist im Anhang zu finden (Seite IX). Hieraus kann auf die benötigten REST-Schnittstellen geschlossen werden.

## 3.5 Lastenheft

Am Ende der Analysephase wurde ein Lastenheft erstellt. Dieses umfasst alle Anforderungen des Auftraggebers an den zu erstellenden Service. Das Lastenheft befindet sich im Anhang (Seite VI) und dient als Grundlage für die Konzeption.

# 4. Konzeption

## 4.1 Zielplattform

Das Abschlussprojekt soll, wie bereits in Abschnitt Projektschnittstellen (Seite 2) erwähnt wurde, in der Amazon Web Services Umgebung entwickelt werden. Diese zeichnet sich durch eine sehr hohe Verfügbarkeit sowie Skalierbarkeit aus. Insbesondere soll das sogenannte "Serverless Application Model" (SAM) zum Einsatz kommen, was sich durch die weitestgehende Auslagerung der Server- und Infrastrukturverwaltung an Amazon auszeichnet.

Eine DynamoDB Instanz soll zum Einsatz kommen, da diese sich bereits im Unternehmen bewährt hat und die Integration mit anderen Amazon Diensten sehr intuitiv ist. SAM sieht als Back-End Engine AWS Lambda vor. Daher wurde ebendies für die Realisierung gewählt. Auch dieser Service zeichnet sich durch hohe Flexibilität und Skalierbarkeit aus.

Bei der Auswahl der Programmiersprache zur Realisierung des Services standen von Seiten Amazons .NET, Go, Java, Node.js (JavaScript) und Python zur Verfügung. Da die beiden meist verwendeten Sprachen im Unternehmensumfeld Java und JavaScript sind, wurde die Liste der möglichen Programmiersprachen auf diese beiden gekürzt. Die Wahl fiel auf Node.js (JavaScript), da diese in mehreren anderen internen Webservice-Projekten erfolgreich eingesetzt wird und sich somit bereits etabliert hat.

Zur Entwicklung der grafischen Oberfläche in HTML, CSS und JavaScript wurden die Frameworks jQuery und AngularJS zur logischen Verarbeitung von Eingaben, dynamischen Strukturierung des statischen HTMLs und Durchführung der REST Anfragen verwendet, sowie Bootstrap für das Design des Layouts. Auch diese Kombination wurde bereits bei mehreren internen Projekten verwendet und hat sich etabliert.

## 4.2 Datenbankmodell

Um einen Überblick über alle für den Validierungs- und Verwaltungsservice benötigten Daten zu erhalten wurde ein Modell erstellt, welches die Typen und etwaige Beziehungen veranschaulicht. Aufgrund der Nutzung einer nichtrelationalen Datenbank, wird nur eine einzige Tabelle "Subscriptions" erstellt, mit einem sog. Hash-Key "id" sowie zwei Indizes "user\_id" und "purchase\_id". Sowohl "id" als auch "purchase\_id" werden vom Reseller erstellt und übermittelt. Die einzige Relation zu einem anderen Datensatz stellt die "user\_id" dar, über welche der entsprechende Benutzer im Amazon Dienst Cognito gefunden und bearbeitet werden kann und als Zuordnung einer Subscription zu einem Kundenkonto dient.

Subscriptions
id : String (PK, Hash-Index)
purchase_id : String (Hash-Index)
user_id : String (Hash-Index)
status : String
quantity : Integer
cancellationUrl : String
changePaymentSubscriptionUrl : String

## 4.3 Ablaufmodell

Aus der Analysephase wird ein Modell für den Ablauf eines Subscription-Abschlusses erstellt. Nach dem Kauf einer Subscription wird die RealObjects GmbH vom ECD per REST API benachrichtigt.

Übergeben werden die Kaufdaten, die in einem vom ECD zur Verfügung gestellten Schema, welches sich auszugsweise im Anhang findet (Seite X), vorliegen. Hieraus werden die benötigten Daten extrahiert und überprüft, ob zu der übermittelten E-Mail Adresse des Lizenznehmers ein Benutzer vorliegt. Ist dies der Fall, wird die entsprechende Identifikationsnummer dem Datensatz hinzugefügt. Das so erstellte Tupel wird in die Datenbank eingetragen. Falls kein Nutzer zu der E-Mail Adresse gefunden wurde, wird aus der übermittelten "Purchase ID" und einem generierten Hashwert eine URL erstellt. Diese wird dem ECD zurückgegeben, der diese dann dem Kunden zum Aktivieren der Subscription übermittelt. Mit der erstellten URL kann sich ein Nutzer in der Benutzeroberfläche anmelden oder registrieren und die zugehörige Subscription seinem Nutzeraccount zuordnen.

Die Registrierung eines neuen Nutzers erfordert nach der Übermittlung der Daten eine Bestätigung der E-Mail-Adresse durch Besuch einer generierten Seite der Benutzeroberfläche. Der Nutzeraccount soll vor dieser Verifikation nicht genutzt werden können. Im Anhang findet sich das hierzu erstellte Aktivitätsdiagramm. (Seite XI)

## 4.4 Cloud-Modell

Um einen Überblick über die Dienste und ihre Wechselwirkung miteinander zu erstellen, wurde ein Modell mit dem kostenlosen Tool Cloudcraft erstellt und ist im Anhang (Seite XII) zu finden.

## 4.5 REST-API-Konzeption

Aufgrund des Anwendungsfalldiagramms aus dem Abschnitt "Anwendungsfälle" (Seite 6) wurde ein Modell für die benötigten Ressourcen und Methoden der REST API erstellt. Weiterhin wurde eruiert, welche generellen Funktionen die hierbei zugeordneten Lambdas erfüllen sollen, welche Werte übermittelt werden und welche Rückgaben zu erwarten sein sollen. Im Zuge dessen wurde für jede REST-Methode eine Autorisierungsrichtlinie festgelegt, um unberechtigte Zugriffe zu vermeiden.

## 4.6 Pflichtenheft

Am Ende der Entwurfsphase wurde ein Pflichtenheft erstellt. Dieses baut auf dem Lastenheft (Seite 6) auf und beschreibt wie und mit welchen Hilfsmitteln der Autor die an den Webservice und die Benutzeroberfläche gestellten Anforderungen umsetzen möchte. Das Pflichtenheft dient als Leitfaden zur Realisierung des Projektes und befindet sich im Anhang (Seite XIII).

# 5. Durchführung

## 5.1 Implementierung der Datenstruktur

Auf Basis der Datenstruktur, die bereits im "Datenmodell" für die zu erstellende Anwendung definiert worden ist, wurde die Implementierung der Datenbank durchgeführt. Zunächst wurde dafür ein Schema im JSON Format erstellt. Dieses findet sich im Anhang (Seite XIII). Anschließend wurde dieses Schema via AWS Command-Line-Interface per "create-table" Befehl übergeben. Die erstellte Datenbank entspricht der in der Entwurfsphase definierten Struktur und erfüllt so schon die nötigen Anforderungen.

Weiterhin wurde der benötigte User Pool in Amazon Cognito erstellt. Die angewandten Konfigurationen finden sich ebenfalls im Anhang (Seite XVI).

## 5.2 Implementierung des API-Gateways

Um die später in den Lambdas implementierten Funktionen nutzen zu können, wurde ein API Gateway erstellt, welches Anfragen an eine bestimmte URL

entgegennehmen und an die ihr jeweils zugeordnete Lambda-Funktion weitergeben kann.

Damit nur Anfragen aus berechtigten Quellen die Lambda-Funktionen erreichen, wurde das API Gateway so eingerichtet, dass Anfragen abgeblockt werden können, welche ohne API-Schlüssel oder Cognito-Identifikations-Header gestellt werden. Dies wurde vom Autor für Anfragen an Methoden, welche empfindliche Informationen liefern oder besondere Nutzerrechte erfordern, implementiert.

Im Falle der User-Pool-Autorisierung wurde als Autorisierungskriterium ein Header mit Namen "Authorization" festgelegt, welcher den von Cognito beim Start der Sitzung gelieferten Sitzungsschlüssel enthält. Auf diese Weise kann jeder Anfrage der Nutzer zugeordnet werden, der sie gestellt hat.

Um Zugriff auf erweiterte Daten der Anfrage zu erhalten, wurde für alle Integrationen die sogenannte "Lambda Proxy Integration" verwendet, welche die Lambda-Funktion mit einem Objekt aufruft, das diese Informationen enthält. Hierbei werden, falls als Autorisierungsmaßnahme der Cognito-Identifikations-Header gewählt wurde, auch die Daten des Nutzers mitgeliefert, zu welchem dieser Header gehört. Dies wurde genutzt, um beispielsweise die dazugehörigen Subscription-Daten korrekt zurückzugeben und Nutzerberechtigungen zuzugestehen.

### 5.3 Implementierung der Lambda-Funktionen

Die beiden wichtigsten Aufgaben des Webservice sind zum einen die Verarbeitung der POST-Anfragen auf die REST-API-Ressource zur Übermittlung der Kaufdaten durch den ECD und Rückgabe von Schlüsselattributen an den ECD, welcher dieser dann in die Bestätigungsmail des Kunden einfügt, zum anderen die Benutzerkonten- und Subscription-Verwaltung als Backend für die Benutzeroberfläche (siehe unten). Für den Zugriff auf Datenbank und Benutzerpool wurde in jedem Script das Node.js Modul "AWS-SDK" integriert, welches Funktionen und Objekte für Dokumente, Nutzer und Benutzerpools zur Verfügung stellt.

#### E-Commerce-Endpoint-Verarbeitung

Die Verarbeitung der Kaufdaten erfolgt direkt nach Übermittlung durch den ECD per POST Request an die zuvor bereitgestellte REST API Ressource. Hierfür wurde ein Node.js Script erstellt, welches einen Handler exportiert, der von der entsprechenden REST API Gateway Ressource mit den Lambda Proxy Objekten aufgerufen wird.

Um Zugriff auf Datenbank und Benutzerpool zu erhalten wurde der Lambda-Instanz eine IAM-Role zugeordnet, welche vollen Zugriff auf diese Services hat. Für die Erstellung des Hashwertes wurde die Verschlüsselungsmethode SHA-512 ausgewählt und neben der "Purchase ID" ein weiterer Bestandteil der Subscription-Daten sowie zwei 20-stellige geheime Schlüssel als Quelldaten. Die

Hasherzeugung wurde mit dem Modul "crypto", welches bereits von Node.js zur Verfügung gestellt wird, implementiert.

## Account- und Subscription-Management

Zur Verarbeitung und Bereitstellung der Daten der Benutzeroberfläche wurde für jede verfügbare REST API Methode ein Lambda mit einem Node.js Script erstellt, welches die übergebenen Daten prüft und bearbeitet. Außerdem wurde ein weiteres Lambda erstellt, dessen Funktion einen Link mit Hashwert erzeugt, welcher bei der Registrierung im Cognito Benutzer-Pool mitgesendet wird.

Um die Validität eines Purchase-ID-Hashwert-Paares zu überprüfen wurde der GET-Methode der variablen Ressource `/purchase/verify/{id}/{key}` ein Script zugeordnet, das anhand der übergebenen `{id}`, welche der `purchase_id` der Subscription entspricht, den passenden Eintrag in der Datenbank ausliest, die im Abschnitt "E-Commerce-Endpoint-Verarbeitung" (Seite 9) beschriebene Hasherzeugung erneut durchführt und das Ergebnis mit dem angegebenen `{key}` vergleicht. Stimmen diese beiden überein, gibt die Methode `true` zurück, ansonsten `false`.

Zur Zuordnung einer Subscription zu einem Benutzer-Account wurde der POST-Methode der API Ressource `/purchase/assign` eine Lambda-Funktion zugeordnet, welche im Body der Anfrage die Purchase-ID sowie den dazugehörigen Hashwert erwartet. Durch Hinzufügen der Autorisierungsmethode `per User-Pool` wird dem der Lambda-Funktion übergebenen "event" Objekt durch das API Gateway unter anderem die ID des Nutzers hinzugefügt. Wie zuvor wird die Validität des Schlüssels überprüft und im Validierungsfalle die ID des Nutzers der Subscription hinzugefügt.

Die Informationen über alle einem Nutzer zugeordneten Subscriptions wurden per GET-Methode via `/subscriptions` zugänglich gemacht. Auch hier wurde die Methode `per User-Pool` Authorisierung abgesichert und das zugeordnete Script wurde so implementiert, dass es über die mitgelieferte Nutzer-ID die zugehörigen Subscriptions aus der Datenbank ausliest und zurückgibt.

Da der Dienst Cognito bei Registrierung neuer Nutzer selbständig eine E-Mail mit einem Verifizierungscode versendet, wurde eine Lambda-Funktion implementiert, welche einen entsprechenden E-Mail Inhalt generiert und zurückgibt. Für diesen Inhalt wurde wiederum eine URL erzeugt, welche Benutzername und einen generierten Hashwert enthält. Quelle für den Hashwert ist der Benutzername in Verbindung mit einem zwanzigstelligen geheimen String. Auch hierfür wird das von Node.js mitgelieferte Modul "crypto" verwendet. In der Benutzeroberfläche soll später dieser Link aufgelöst werden und so eine Verifikation des Benutzer-Accounts erfolgen.

Die zur Verifikation der Benutzer-Accounts benötigte Ressource wurde der POST-Methode der `/user/verify` Ressource zugeordnet. Das zugehörige Lambda erwartet einen Benutzernamen und den zuvor erwähnten Hashwert zur Verifi-



zierung der E-Mail Adresse. Auch hier wird der Benutzername in Verbindung mit dem selben zwanzigstelligen geheimen String zu einem Hashwert verarbeitet und mit dem mitgesendeten Schlüssel verglichen. Bei Gleichheit wird der Nutzer im Cognito User-Pool als verifiziert gekennzeichnet.

## 5.4 Implementierung der Benutzeroberfläche

Mit Hilfe von Bootstrap wurde das Layout des Webinterfaces erstellt, welches eine Seitenleiste zum Navigieren zwischen verschiedenen Ansichten enthält sowie eine Kopfleiste zur Darstellung des Firmenlogos und ein Dropdown zum Verwalten des Nutzer-Accounts (bspw. zum Logout). Hierfür wurde ein passendes Bootstrap Template<sup>1</sup> gefunden und auf die Bedürfnisse des Kunden angepasst.

Für die Navigation zwischen den einzelnen Ansichten wurde die AngularJS Routing Funktion genutzt, welche HTML-Templates in ein dementsprechend durch ein Attribut "ng-route" gekennzeichnetes Block Element lädt. Für jedes so angelegte Template wird eine HTTP-Ressource in dem Hash-Parameter der URL erzeugt. Beim Laden des Templates werden darin vorkommende AngularJS-Variablen, in diesem Fall gekennzeichnet durch den Einschluss in doppelten eckigen Klammern, durch die entsprechende Variable im Controller ersetzt bevor der HTML-Ausschnitt eingefügt wird.

Unter anderem wurde ein Login-Template erstellt, um Nutzern zu ermöglichen sich einzuloggen. In den Routing-Mechanismus wurde weiterhin eine Direktive eingefügt, dieses Template anzuzeigen, falls derzeit kein Benutzer angemeldet ist. Diese Konfigurationen werden dem AngularJS-Modul per "config()" Methode übergeben. Im Anhang findet sich der hierzu passende Codeausschnitt (Seite XVIII), die angelegte Ordnerstruktur (Seite XXVIII) sowie ein Screenshot der Benutzeroberfläche (Seite XXIX).

Die Benutzeroberfläche wurde als statische HTML-Website in einem Amazon S3 Bucket hinterlegt, welcher eingerichtet wurde, um die die statischen Ressourcen für die Webanwendung zu hosten. Jegliche dynamische Funktionalität wird so auf Clientseite ausgelagert oder via JavaScript über RESTful APIs, die mit AWS Lambda und Amazon API Gateway erstellt wurden, aufgerufen. Die Entscheidung, alle HTML-Ressourcen statisch zur Verfügung zu stellen und nur die Datenabfrage dynamisch zu gestalten wurde vom Autor getroffen, um Datenvolumen und Abfragefrequenz für das API Gateway möglichst gering zu halten.

Die dynamische Verwaltung von Daten und das Behandeln spezieller Funktionen werden in AngularJS über sogenannte Controller gesteuert und ausgeführt. Diese Controller werden dem Modul hinzugefügt und bestimmten HTML-Elementen per "data-ng-controller" Attribut zugeordnet. Beim Laden des Elements werden der zugehörige Controller aufgerufen, Daten zugeordnet und

---

<sup>1</sup> Siehe hierzu <https://startbootstrap.com/template-overviews/sb-admin/>

die entsprechenden Variablen im HTML ersetzt bzw. eingefügt. Somit folgt die Implementierung dem MVC-Prinzip (Model View Controller).

Der Autor hat für die Anwendung einen Root-Controller erstellt, welcher allgemeine und oft benötigte Daten und Funktionen zur Verfügung stellt und beim Laden der Seite aufgerufen wird. Weiterhin wurde für jedes Template ein Controller geschrieben, der unter anderem bestimmte Style-Änderungen auslöst. Zum Beispiel wird sowohl beim Aufruf der Login- als auch der Registrierungsseite die Seitennavigationsleiste ausgeblendet.

Zur Verwaltung der Benutzersitzung wurde "amazon-cognito-identity-js", als Teil der JavaScript-Bibliothek AWS-Amplify<sup>1</sup> benutzt. Zur Bereitstellung der Funktionen dieser Bibliothek in den AngularJS Controllern wurde vom Autor ein entsprechender AngularJS Service implementiert, welcher später in Controllern, die Sitzungsverwaltung benötigen, injiziert wurde.

Der Service deckt die Funktionen Registrierung, Einloggen, Ausloggen und Erneutes zusenden der Verifikations-Email ab und bietet Methoden zum Zugriff auf das von AWS-Amplify zur Verfügung gestellte CognitoSession-Objekt, welches Authentifizierungsschlüssel und Nutzerdaten enthält. Dieses Objekt, sowie das CognitoUser-Objekt greifen dabei auf Sitzungsdaten zurück, die im lokalen Speicher des Browsers vorgehalten werden. Der Codeausschnitt für diesen Service findet sich im Anhang (Seite XIX).

Bei Anfragen an das REST API Gateway, welche Ressourcen betreffen, die per User-Pool-Autorisierung geschützt sind, musste hier darauf geachtet werden, den entsprechenden Token für die Sessionauthentifizierung als Header "Authorization" mitzusenden.

## 6. Qualitätssicherung

### 6.1 Modultests

Aufgrund der Beschaffenheit und dem hohen Maß an Isolation zwischen den einzelnen Programmteilen hat sich der Autor dazu entschlossen, Modultests durchzuführen. Das bedeutet, dass die Anwendung in Teile untergliedert wird und jeder Teil einer Reihe von Tests unterzogen wird. Die Unterteilung wird wie folgt vorgenommen:

#### Webservice

Hierzu zählen die einzelnen Lambda-Komponenten mit ihren jeweiligen REST API Methoden. Zu diesem Zweck wurde die Autorisierung im API Gateway abgeschaltet, Dummy-Daten übermittelt und Konsolenausgaben erzeugt, welche im Amazon Dienst CloudWatch überprüft werden können. Zum Übermitteln der Anfragen wird die Anwendung Postman von Google benutzt. Die Methode jeder API Ressource wird einzeln auf Korrektheit der Konsolenaus-

---

<sup>1</sup> Siehe hierzu <https://github.com/aws-amplify/amplify-js/tree/master/packages/amazon-cognito-identity-js>



gaben sowie der erwarteten Antwort (sowohl Status, als auch Inhalt) für verschieden geartete Anfragen überprüft. Des Weiteren wird bei Lambdas, welche Daten in die Datenbank schreiben die Korrektheit der geschriebenen Daten überprüft.

## Benutzeroberfläche

Hierzu zählen die einzelnen statischen HTML-Ressourcen sowie die AngularJS Controller. Jedes Template wurde auf jede Funktion getestet, welche durch den Nutzer ausgeführt werden kann. Hierzu wurden die API-Gateway Methoden so abgeändert, dass sie Mock-up Antworten zurückgeben und dementsprechend das erwartete Verhalten der Funktionen überprüft. Außerdem wurden Konsolenausgaben eingefügt und die erhaltenen Ausgaben auf Übereinstimmung mit dem erwarteten Verhalten abgeglichen.

Weiterhin wurde die Benutzeroberfläche in verschiedenen Browsern getestet, um die Kompatibilität und Bedienbarkeit und Funktionalität sicherzustellen. Bei Tests aller Views und Funktionen in den gängigen Browsern Chrome (Google), Firefox (Mozilla Foundation), Edge (Microsoft) und Safari (Apple) konnte keine Inkompatibilität festgestellt werden.

## 6.2 Integrationstest

Nachdem die korrekte Funktionalität der einzelnen Module überprüft wurde, wurde ein Integrationstest durchgeführt, um das Zusammenspiel der Komponenten untereinander zu überprüfen. Hierbei wurden die Ressourcen und Methoden des API Gateways, welche im vorherigen Schritt verändert wurden, wiederhergestellt und der Arbeitsablauf für verschiedene Szenarien durchgespielt. Im Anschluss wurden die aufgezeichneten Konsolenausgaben mit dem erwarteten Verhalten verglichen.

Hierbei wurde ebenfalls durch bewusste Eingaben von Fehldaten die Robustheit des Systems getestet. Auch hierbei entsprachen die Konsolenausgaben und das Verhalten dem erwarteten.

## 6.3 Datenintegrität

Aufgrund der Beschränkung der Datenbanktabellen auf eine einzige ist die Integrität der Daten in der Datenbank gewährleistet. Im Falle einer Abmeldung eines Benutzers vom Managementsystem wird der entsprechende Eintrag weiter in der Datenbank geführt und nur als "Deaktiviert" markiert. Das heißt also, dass der Lizenznehmer in jedem Falle noch nachvollziehbar ist.

# 7. Dokumentation

## 7.1 Kundendokumentation

Es wurde eine Kundendokumentation erstellt, welche eine Beschreibung der allgemeinen Bedienung des Services aus Endnutzersicht darstellt. Sie erklärt den Vorgang der Registrierung, die Zuordnung der Subscriptions zu einem

Account sowie die Möglichkeiten der Account- und Subscription-Verwaltung und findet sich im Anhang (Seite XX).

## 7.2 Entwicklerdokumentation

In der Entwicklerdokumentation wird der Code der AngularJS Applikation der Benutzeroberfläche dokumentiert und eine Anleitung zum Erweitern der Oberfläche um Funktionalitäten und Templates gegeben. Des Weiteren werden die Konfigurationen der einzelnen Amazon Services dokumentiert, die Änderungsmöglichkeiten sowie eine Anleitung zum Erweitern des Webservices um weitere API Ressourcen und den dazugehörigen Lambda-Funktionen.

Außerdem wird die Wechselwirkung der Services untereinander näher beschrieben und die mögliche Einflussnahme darauf. Schlussendlich wurde das Deployment und die Konfiguration der Services für den Einsatz in einer Produktivumgebung dokumentiert, da zum Zeitpunkt des Projektabschlusses noch kein Deployment stattfinden konnte.

## 8. Übergabe

Im Rahmen einer Präsentation wurde das fertige Produkt am 16.11.2018 den Verantwortlichen sowie den Mitarbeitern mit kaufmännischem Handlungsfeld vorgestellt und die korrekte Ausführung aller Funktionen vorgeführt. Nachdem alle Beteiligten mit dem Ergebnis zufrieden waren und verschiedene Rückfragen zur Nutzung der Amazon Services geklärt wurden, wurde ein Code-Review durchgeführt, um die Wartbarkeit und Qualität des Codes an sich sicherzustellen.

Derzeit wird das Produkt noch nicht in den Live-Betrieb übergehen, da zum Zeitpunkt der Fertigstellung des Produkts die entsprechenden Verträge mit dem ECD noch nicht abgeschlossen waren. Das Subscription und Account Managementsystem als Serverlose Anwendung (kurz SAMSA) wurde als Projekt im firmeninternen SVN eingchecked und galt somit als übergeben.

## 9. Fazit

### 9.1 Soll-/Ist-Vergleich

Bei einer rückblickenden Betrachtung des IHK-Abschlussprojektes, kann festgehalten werden, dass alle zuvor festgelegten Anforderungen gemäß dem Pflichtenheft erfüllt wurden. Der zuvor aufgestellte Projektplan konnte eingehalten werden.

In untenstehender Tabelle findet sich eine Gegenüberstellung der für die einzelnen Phasen eingeplanten mit der tatsächlichen benötigten Zeit. Die entstandenen Differenzen konnten in darauffolgenden Projektphasen kompensiert werden. Insbesondere die geschickte Auswahl eines passenden Bootstrap Layout Templates konnte Zeit in der Entwicklung der Benutzeroberfläche einsparen.

Bei der Implementierung der Datenbank konnten weitere Zeiteinsparungen erwirkt werden, aufgrund der geringen Komplexität der Datenstrukturen. Dies bedurfte wiederum einem leichten Mehraufwand in der Konzeptionsphase bedurfte. Somit konnte die für das Projekt zur Verfügung gestellte Zeit von 70 Stunden eingehalten werden.

Projektteil	Soll	Ist	$\Delta$
Analyse	6	7	+ 1
Konzeption	8	10	+ 2
Entwicklung	36	33	-3
Tests	10	10	0
Dokumentation	10	10	0
<b>Gesamt</b>	<b>70</b>	<b>70</b>	<b>0</b>

## 9.2 Ausblick

Die Erzeugung und Validierung von konkreten API-Schlüsseln, mit denen Produkte den Status der zugehörigen Subscription abfragen können, wurde diesem Projekt nachgelagert, da dies tiefgreifende Änderungen in momentan vertriebenen Produkten bedeutet. Derzeit wurde die Planung für eine solche Funktionalität für die Programme zwar schon begonnen, allerdings ist die konkrete Form und Generierungsart der API-Schlüssel noch nicht festgelegt. Daher wird eine nachgelagerte Aufgabe sein, diese Funktionalität nachzurüsten, sobald der Entwicklungsprozess auf Softwareseite abgeschlossen ist.

## 9.3 Subjektives Fazit

Im Zuge des Projektes konnte der Autor viele Erfahrungen über die Arbeit an einem vollständigen Projekt sammeln. Hierbei wurde deutlich, dass großer Wert auf die Analyse und den Entwurf eines Projektes gelegt werden muss. Durch das Projekt wurden fachliche Kompetenzen im Bezug auf Amazon Web Services erworben und die Automatisierung eines Geschäftsprozesses erwies sich als spannendes und lehrreiches Unterfangen, bei dem das korrekte Einhalten der Schnittstellen und Schemata eine hohe Priorität hat. Insbesondere im Bereich der Cloud-basierten Software ist dieser Aspekt unabdingbar.

# Glossar

**Amazon Web Service (AWS)** Cloud-Computing-Anbieter, Tochterunternehmen des weltweit agierenden Unternehmens Amazon

**Amazon API Gateway** Vollständig verwalteter Service, der das Erstellen, Veröffentlichen, Warten, Überwachen und Sichern von APIs für Entwickler vereinfacht

**Amazon CloudWatch** Service zur Überwachung von Ressourcen und Anwendungen, die in der AWS-Umgebung ausgeführt werden

**Amazon Command Line Interface** Open-Source-Tool für Python, das Befehle für die Interaktion mit Services in der AWS-Umgebung bereitstellt

**AWS Lambda** AWS Lambda ist ein Datenverarbeitungsservice, mit dem Code ausgeführt werden kann, ohne Server bereitstellen oder verwalten zu müssen. AWS Lambda führt Code nur bei Bedarf aus und skaliert automatisch

**Amazon DynamoDB** Vollständig verwalteter NoSQL-Datenbankservice, der schnelle und planbare Leistung mit guter Skalierbarkeit bietet

**Amazon Cognito** Webservice, der Authentifizierung, Autorisierung und Benutzerverwaltung für Web- und Mobil-Apps bietet

**Amazon IAM** Webservice, der dabei hilft, Zugriff auf AWS-Ressourcen zu steuern, indem Rollen definiert werden, die Anwendungen zugewiesen werden

**Amazon S3 (Simple Storage Service)** Cloud-Speicher-Service, der darauf ausgelegt ist, Cloud Computing für Entwickler zu erleichtern; S3 besitzt eine einfache Web-Service-Schnittstelle zum Speichern und Abrufen einer beliebigen Datenmenge

**Amazon Serverless Application Model (SAM)** Modell, um serverlose Anwendungen von AWS zu definieren; Grundsätzlich gilt für alle Anwendungen in diesem Modell, dass sie das Entwickeln eines Cloud-Service ermöglichen, ohne dass der Entwickler Server bereitstellen oder verwalten muss

**AWS-SDK** Bibliothek, die ein API für Amazon-Dienste bietet; Wird für verschiedene Programmiersprachen angeboten

**AngularJS** Clientseitiges JavaScript-Webframework zur Erstellung von Single-Page-Webanwendungen

**Bootstrap** Kostenloses CSS-Framework mit optionaler JavaScript-Erweiterung

**Cloudcraft** Visualisierungswerkzeug für Cloud-Architekturen, spezialisiert auf AWS

**jQuery** Freie JavaScript-Bibliothek, die Funktionen zur DOM-Navigation und -Manipulation zur Verfügung stellt

**MVC-Prinzip** Unterteilung einer Anwendung in Modell (model), Präsentation (view) und Steuerung (controller), wobei das Modell von Präsentation und Steuerung unabhängig sein soll

**Node.js** Serverseitige Plattform in der Softwareentwicklung zum Betrieb von Netzerkanwendungen, welche in der JavaScript-Laufzeitumgebung V8 ausgeführt wird

**Software-as-a-Service** Modell aus dem Bereich des Cloud-Computing; folgt dem Grundsatz, dass Software und Infrastruktur bei einem Dienstleister verbleibt und der Kunde für die Nutzung zahlt.

**REST** Programmierparadigma für verteilte Systeme, dessen Schwerpunkt auf der Kommunikation von Maschine zu Maschine liegt und das den Prinzipien von Roy Fielding folgt

## Quellenangaben

- [https://docs.aws.amazon.com/de\\_de/lambda/latest/dg/lambda-dg.pdf](https://docs.aws.amazon.com/de_de/lambda/latest/dg/lambda-dg.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/AmazonS3/latest/dev/s3-dg.pdf](https://docs.aws.amazon.com/de_de/AmazonS3/latest/dev/s3-dg.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/IAM/latest/UserGuide/iam-ug.pdf](https://docs.aws.amazon.com/de_de/IAM/latest/UserGuide/iam-ug.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/cognito/latest/developerguide/cognito-dg.pdf](https://docs.aws.amazon.com/de_de/cognito/latest/developerguide/cognito-dg.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/AmazonCloudWatch/latest/monitoring/acw-ug.pdf](https://docs.aws.amazon.com/de_de/AmazonCloudWatch/latest/monitoring/acw-ug.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/amazondynamodb/latest/developerguide/dynamodb-dg.pdf](https://docs.aws.amazon.com/de_de/amazondynamodb/latest/developerguide/dynamodb-dg.pdf) (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/apigateway/latest/developerguide/welcome.html](https://docs.aws.amazon.com/de_de/apigateway/latest/developerguide/welcome.html) (abgerufen am 23.11.2018)
- <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (abgerufen am 23.11.2018)
- [https://docs.aws.amazon.com/de\\_de/AmazonCloudWatch/latest/monitoring/acw-ug.pdf](https://docs.aws.amazon.com/de_de/AmazonCloudWatch/latest/monitoring/acw-ug.pdf) (abgerufen am 23.11.2018)

# Anhang

## A.1 Ressourcen

### Sachmittel

- Büroarbeitsplatz mit Tower PC

### Software zum Entwickeln

- Betriebssystem: Windows 10 (Microsoft)
- Code-Editor: Visual Studio Code (Microsoft)
- NPM Version 6.4.1
- Node.js Version 8.10
- Google Chrome 70.0.3538 (Google)
- AWS Command Line Interface 1.15.75 (Amazon)
- AWS services (Amazon)
  - AWS Lambda (Verwendet mit Node.js Version 8.10)
  - Amazon S3 (Simple Storage Service)
  - Amazon DynamoDB
  - Amazon API Gateway
  - Amazon Cognito
  - AWS Identity and Access Management (IAM)
  - Amazon CloudWatch

Die lokale Entwicklung der Lambda Funktionen wurde mit einer Node.js Umgebung durchgeführt. Mit dem Framework Express.js wurde ein lokaler HTTP-Server implementiert, der das Verhalten des API-Gateway simuliert hat.

### Software für Qualitätssicherung

- Postman (Postdot Technologies)
- Google Chrome 70.0.3538 (Google)
- Microsoft Edge 42.17134.1.0 (Microsoft)
- Mozilla Firefox 63.0.1 (Mozilla Foundation)
- Apple Safari 12 (Apple)

## A.2 Zeitplanung

Im folgenden findet sich eine genauere Aufschlüsselung der Zeitplanung aus dem gleichnamigen Kapitel (Seite 3).

<b>Analyse</b>	<b>6h</b>
Analyse der Verkaufsplattformdaten	2h
Analyse der REST-Schnittstellen	2h
Analyse der benötigten Funktionen und Berechtigungen	2h
<b>Konzeption</b>	<b>8h</b>
Entwurf des Datenbankmodells	2h
Konzeption der Funktionalitäten und REST-Schnittstellen	3h
Konzeption der Benutzeroberfläche	3h
<b>Entwicklung</b>	<b>36h</b>
Aufsetzen der Datenbank	1h
Erstellen der Lambda Funktionen	7h
Herstellen des API-Interface	2h
Umsetzung der Benutzeroberfläche	26h
Erstellung der Basis-Website mit Navigationsleiste und Startseite auf Basis von AngularJS	5h
Anpassung der Stylesheets entsprechend Designvorgaben	5h
Erstellung der Templates und Eingliederung in die Webseitenstruktur	3h
Implementierung des Registrierungs-Prozesses für User	6h
Einrichtung der Benutzer-Authentifizierung	3h
Implementierung der Abfragen der Subscription-Daten	4h
<b>Tests</b>	<b>10h</b>
Integritätstests der Datenbank	1h
Funktionalitätstest REST-Schnittstellen und Lambda-Funktionen	5h
Funktionalitätstest der Benutzeroberfläche	1h
Kompatibilitätstest der Benutzeroberfläche	1h
Robustheitstest der Benutzeroberfläche	2h
<b>Dokumentation</b>	<b>10h</b>
Kundendokumentation	2h
Entwicklerdokumentation	8h
<b>Gesamt</b>	<b>70h</b>



## **A.3 Lastenheft**

Im folgenden Lastenheft werden die Anforderungen definiert, welche die zu entwickelnde Web-Anwendung erfüllen soll. (Siehe Folgeseiten)



# **Subscription und Account Management in einer Serverlosen Anwendung (SAMSA)**

## **Lastenheft**

### **Allgemeines**

#### **Zweck und Ziel**

Dieses Lastenheft enthält die Zusammenstellung aller Anforderungen des Auftraggebers hinsichtlich des Liefer- und Leistungsumfangs. Es beschreibt die fachlichen Basisanforderungen aus Anwendersicht einschließlich aller Randbedingungen und definiert damit, welche Aufgaben zu lösen sind, allerdings nicht, wie diese Lösung aussehen soll.

#### **Hintergrund**

Die Entwicklung des Produktes geschieht vor dem Hintergrund der IHK-Abschlussprüfung im Winter 2018 für Fachinformatiker für Anwendungsentwicklung. Da dem Projekt eine ausführliche Dokumentation beiliegt, beschränkt sich dieses Lastenheft nur auf die konkreten Anforderungen an das zu entwickelnde Produkt und entbehrt jeglichen weiteren Ansprüchen an ein Lastenheft.

### **Anforderungen**

#### **Musskriterien**

- Back-End
  - Die Anwendung muss eine ReSTful Methode anbieten, womit ein Reseller Verkaufsdaten übermitteln kann.
  - Die Anwendung muss eine hohe Verfügbarkeit gewährleisten.
  - Die Anwendung muss eine geeignete Teilmenge der übermittelten Informationen persistieren, die notwendig ist, um eine sichere Abfrage des Status einer Subscription korrekt beantworten zu können.
  - Die Integrität der übermittelten Daten muss gewährleistet sein.

- Die Anwendung muss Subscriptions anlegen und editieren können.
- Die Anwendung muss Informationen über Subscriptions benutzerspezifisch anzeigen können und vor Fremdzugriff geschützt sein.
- Die Anwendung muss für den Auftraggeber sowohl erweiterbar als auch wartbar sein.
- Benutzeroberfläche
  - Die Benutzeroberfläche muss per Webbrowser erreichbar sein, d.h. als HTML-Seite gehostet werden.
  - Es muss ein Sitzungsmanagement für die Benutzeroberfläche implementiert werden
  - Nutzer müssen sich ein- und ausloggen, sowie selbständig registrieren können.
  - Zur Verifikation der Nutzerdaten muss ein Bestätigungsmechanismus für die E-Mail-Adresse implementiert werden.
  - Nutzer sollen ihre erworbenen Subscriptions ansehen sowie beenden können.
  - Nutzer müssen sich ihre erworbenen Subscriptions via Benutzeroberfläche selbständig zuordnen können.

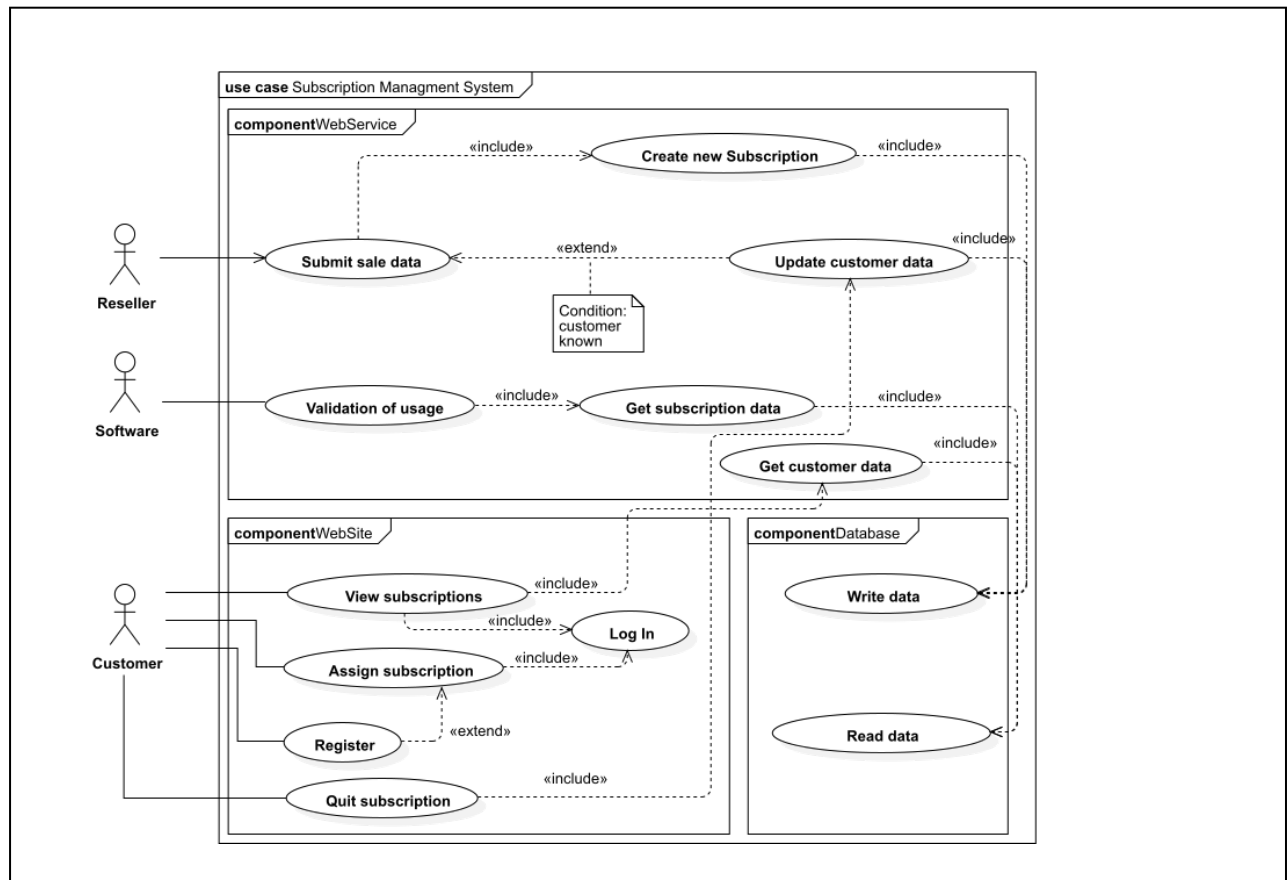
#### **Wunschkriterien**

- Es soll für eine entsprechende Skalierbarkeit Sorge getragen werden.
- Es soll für eine leichte Erweiterbarkeit der Anwendung Sorge getragen werden.
- Es soll Kunden möglich sein, sich bei einer verpassten Verifikations-Email diese erneut zusenden zu lassen.
- Das Design der Benutzeroberfläche soll übersichtlich und ansprechend sein.

#### **Abgrenzungskriterien**

- Die Anwendung soll das vorhandene Kunden-Lizenz-Management nicht ersetzen.
- Es ist nicht nötig, Kontodaten oder Daten für den Schriftverkehr aufnehmen oder verwalten zu können.
- Die Verifikation der Aktivität einer Subscription per REST-Methode wird nachgelagert.

## A.4 Use-Case-Diagramm



## A.5 Purchase Model

Aus Platzgründen wurde das Model stark gekürzt.

```
"PurchaseItemNotificationModel": {
  "type": "object",
  "properties": {
    "runningNumber": [gekürzt],
    "productId": {
      "type": "integer"
    },
    "productName": {
      "type": "string"
    },
    "productNameExtension": {
      "type": "string"
    },
    "clientId": {
      "type": "integer"
    },
    "quantity": {
      "type": "integer"
    },
    "recurringBilling": {
      "type": "object",
      "properties": {
        "subscriptionId": {
          "type": "string"
        },
        "status": {
          "type": "string"
        },
        "itemStatusId": {
          "type": "string"
        },
        "itemStatus": {
          "type": "string"
        },
        "nextBillingDate": {
          "type": "string"
        },
        "cancellationUrl": {
          "type": "string"
        },
        "changePaymentSubscriptionUrl": {
          "type": "string"
        }
      }
    }
  }
}
```

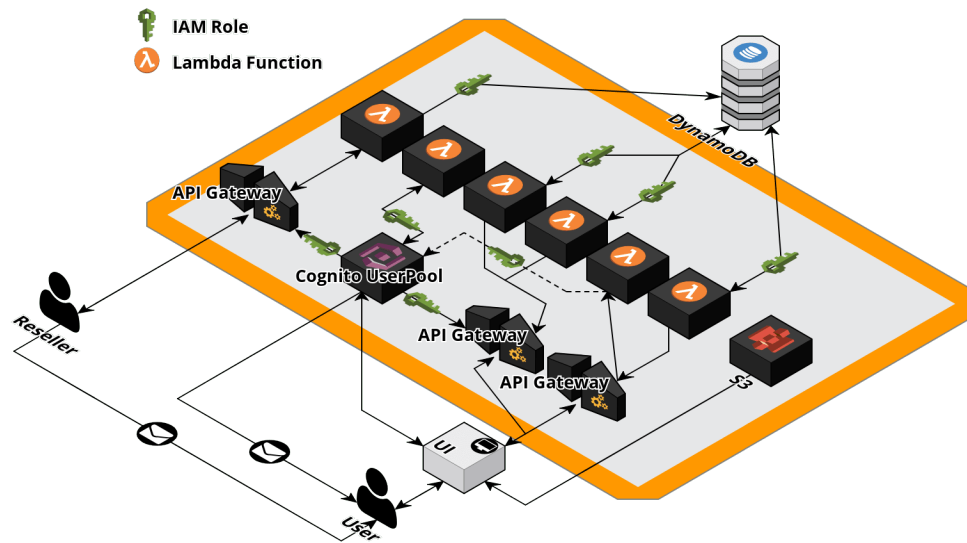
```

    graph TD
        subgraph Kunde
            Start(( )) --> Subscribe
            Subscribe --> Reseller
            Reseller --> ResellerKey[Frage Key ab]
            ResellerKey --> Samsa
            Samsa --> SamsaCreate[Erstelle Subscription]
            SamsaCreate --> SamsaCheck[Prüfe ob Nutzer bekannt]
            SamsaCheck --> SamsaJoin1(( ))
            SamsaJoin1 -- "[Nutzer bekannt]" --> SamsaAssign[Ordne Subscriptions Kundenaccount zu]
            SamsaAssign --> SamsaEnd1(( ))
            SamsaJoin1 --> SamsaJoin2(( ))
            SamsaJoin2 --> SamsaKey[Generiere Key-Text]
            SamsaKey --> Reseller
            Reseller --> ResellerMail[Sende Mail]
            ResellerMail --> Kunde
            Kunde --> KundeMail[Empfange Mail]
            KundeMail --> KundeClick[Klicke Link]
            KundeClick --> Samsa
            Samsa --> SamsaValid1[Validiere Activation-Link]
            SamsaValid1 --> SamsaShow[Zeige Zuordnungsseite]
            SamsaShow --> Kunde
            Kunde --> KundeJoin1(( ))
            KundeJoin1 -- "[Kunde hat Account]" --> KundeCreate[Create account]
            KundeCreate --> KundeData[Übermittle Accountdaten]
            KundeData --> Samsa
            Samsa --> SamsaAssign2[Lege Account an]
            SamsaAssign2 --> SamsaLink[Generiere Activation-Link]
            SamsaLink --> SamsaSend[Versende Activation-Link]
            SamsaSend --> Kunde
            Kunde --> KundeMail2[Empfange Mail]
            KundeMail2 --> KundeClick2[Klicke Link]
            KundeClick2 --> Samsa
            Samsa --> SamsaValid2[Validiere Activation-Link]
            SamsaValid2 --> SamsaActivate[Aktiviere Account]
            SamsaActivate --> Kunde
            Kunde --> KundeKey[Rufe Key ab]
            KundeKey --> Samsa
            Samsa --> SamsaJoin3(( ))
            SamsaJoin3 -- "[Subscription bereits zugeordnet]" --> SamsaAssign3[Ordne Subscriptions Kundenaccount zu]
            SamsaAssign3 --> SamsaJoin4(( ))
            SamsaJoin4 --> SamsaNotify[Benachrichtige dass Subscriptions hinzugefügt]
            SamsaNotify --> KundeKey
        end

        subgraph Samsa
            SamsaCreate
            SamsaCheck
            SamsaJoin1
            SamsaAssign
            SamsaEnd1
            SamsaJoin2
            SamsaKey
            SamsaValid1
            SamsaShow
            SamsaAssign2
            SamsaLink
            SamsaSend
            SamsaValid2
            SamsaActivate
            SamsaJoin3
            SamsaAssign3
            SamsaJoin4
            SamsaNotify
        end

        subgraph Reseller
            ResellerKey
            ResellerMail
        end
    
```

## A.7 Cloudmodell



## A.8 Datenbankschema

```
1 {
2   "params": {
3     "TableName": "Subscriptions",
4     "AttributeDefinitions": [{
5       "AttributeName": "id",
6       "AttributeType": "S"
7     }],
8     {
9       "AttributeName": "purchase_id",
10      "AttributeType": "S"
11    },
12    {
13      "AttributeName": "user_id",
14      "AttributeType": "S"
15    }
16  ],
17  "KeySchema": [{
18    "KeyType": "HASH",
19    "AttributeName": "id"
20  }],
21  "GlobalSecondaryIndexes": [{
22    "IndexName": "user_id",
23    "KeySchema": [{
24      "AttributeName": "user_id",
25      "KeyType": "HASH"
26    }],
27    "Projection": {
28      "ProjectionType": "ALL"
29    },
30    "ProvisionedThroughput": {
31      "WriteCapacityUnits": 1,
32      "ReadCapacityUnits": 1
33    }
34  },
35  {
36    "IndexName": "purchase_id",
37    "KeySchema": [{
38      "AttributeName": "purchase_id",
39      "KeyType": "HASH"
40    }],
41    "Projection": {
42      "ProjectionType": "ALL"
43    },
44    "ProvisionedThroughput": {
45      "WriteCapacityUnits": 1,
46      "ReadCapacityUnits": 1
47    }
48  }
49  ],
50  "ProvisionedThroughput": {
51    "WriteCapacityUnits": 1,
52    "ReadCapacityUnits": 1
53  }
54 }
```

## A.9 Pflichtenheft

Siehe Folgeseiten.





## **Subscription und Account Management in einer Serverlosen Anwendung (SAMSA)**

### **Pflichtenheft**

#### **Allgemeines**

##### **Zweck und Ziel**

Dieses Pflichtenheft enthält die Zusammenstellung aller Realisierungsbestrebungen des Entwicklers. Es beschreibt, wie die fachlichen Anforderungen aus Entwicklersicht einschließlich aller Randbedingungen zu erfüllen sind.

##### **Hintergrund**

Die Entwicklung des Produktes geschieht vor dem Hintergrund der IHK-Abschlussprüfung im Winter 2018 für Fachinformatiker für Anwendungsentwicklung. Da dem Projekt eine ausführliche Dokumentation beiliegt, beschränkt sich dieses Pflichtenheft nur auf die konkreten Anforderungserfüllungen und entbehrt jeglichen weiteren Ansprüchen auf Vollständigkeit an ein Pflichtenheft.

#### **Umsetzungen**

##### **Musskriterien**

- Back-End
  - Die Anwendung wird in der Amazon Web Service Umgebung implementiert, wodurch die hohe Verfügbarkeit gewährleistet wird
  - Es wird eine ReSTful Methode mittels Amazon API Gateway erstellt, welche als Ziel der Datenübermittlung des Resellers dient und die übermittelten Daten an eine Funktion, welche im Amazon Lambda Kontext aufgerufen wird, weitergibt. Diese Funktion filtert die Daten und schreibt sie in eine dafür angelegte Tabelle einer Amazon Dynamo Datenbank. Die Filterung der Daten geschieht nach vorheriger Absprache mit dem Auftraggeber.
  - Die verwendete Programmiersprache für alle Funktionen des Lambda Dienstes wird JavaScript in einer Node.js-Umgebung, da dies bei dem Auftraggeber eine bereits etablierte Sprache ist und so die Wartbarkeit gewährleistet wird. Durch

den modularen Aufbau in Amazon Lambda ist die Anwendung außerdem sehr einfach um weitere Methoden zu erweitern.

- Durch ein sorgfältig konzipiertes Datenbankmodell und eine nachhaltige Datenpolitik wird die Integrität der Datenbank sichergestellt.
- Um einen benutzerspezifischen Zugang zu den Daten und Befugnisse zum Ändern dieser zu gewährleisten wird eine Benutzerkontenverwaltung auf Basis von Amazon Cognito erstellt. Für die Anwendung wird ein neuer sog. Userpool angelegt, welcher die Zugangsdaten der Accounts aufbewahrt und verwaltet.
- Mittels der Konfiguration des API-Gateways wird sichergestellt, dass Rest API Methoden, welche mit sensiblen Daten umgehen, durch API Schlüssel oder Hash-Schlüssel, welche von Amazon Cognito user- und sitzungsspezifisch generiert werden, gegen unbefugte Nutzung geschützt werden.
- Um die Nutzungsrechte der Funktionen für den Zugriff auf Dynamo und Cognito zu steuern werden in Amazon IAM sogenannte Rollen erstellt, welche den Diensten, denen sie zugewiesen werden die entsprechenden Rechte gewährt oder verweigert.
- Benutzeroberfläche
  - Die Benutzeroberfläche wird als HTML-Seite konzipiert, welche im Simple Storage Service (S3) von Amazon gehostet wird. Es wird ein MVC-Modell auf Basis von AngularJS implementiert sowie das freie Frontend-CSS-Framework Bootstrap zum Design der Seiten verwendet.
  - Mittels Amazon Cognito und der Javascript Bibliothek AWS Amplify wird eine Benutzerkontensteuerung auf Basis von Hash-Schlüsseln implementiert.
  - Es wird ein Login- sowie Registrierungsmechanismus auf Basis des zuvor genannten Systems eingerichtet.
  - Mit Amazon Cognito wird eine E-Mail-Verifikation eingerichtet.
  - Es wird für den eingeloggten Nutzer eine Ansicht für die Subscriptions implementiert, in welcher er diese auch ändern oder beenden kann.
  - Durch den E-Mail-Dienst des Resellers wird eine Zuordnungsfunktion auf Basis von generierten Hash-Schlüsseln implementiert, welche durch den Besuch einer Seite der Benutzeroberfläche und nach einem Login die Subscription dem entsprechenden Nutzer zuordnet.

### **Wunschkriterien**

- Mittels Amazon Cognito wird die Funktion zugänglich gemacht, welche die E-Mail mit dem Verifikationslink erneut zusendet.
- Durch die Verwendung von Bootstrap entspricht die Benutzeroberfläche dem neuesten Designstandard.

## A.10 Konfiguration für den Amazon Cognito Benutzerpool

```
{
  "UserPool": {
    "SmsVerificationMessage": "Your verification code is {####}. ",
    "SchemaAttributes": [{
      "Name": "sub",
      "StringAttributeConstraints": {
        "MinLength": "1",
        "MaxLength": "2048"
      },
      "DeveloperOnlyAttribute": false,
      "Required": true,
      "AttributeDataType": "String",
      "Mutable": false
    },
    {
      "Name": "name",
      "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
      },
      "DeveloperOnlyAttribute": false,
      "Required": false,
      "AttributeDataType": "String",
      "Mutable": true
    },
    {
      "Name": "email",
      "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
      },
      "DeveloperOnlyAttribute": false,
      "Required": true,
      "AttributeDataType": "String",
      "Mutable": true
    },
    {
      "AttributeDataType": "Boolean",
      "DeveloperOnlyAttribute": false,
      "Required": false,
      "Name": "email_verified",
      "Mutable": true
    }
  ],
  ["..."],
  "EmailVerificationSubject": "Your verification code",
  "MfaConfiguration": "OFF",
  "Name": "RealObjects",
  "EmailVerificationMessage": "Your verification code is {####}. ",
  "VerificationMessageTemplate": {
    "EmailMessageByLink":
      "Please click the link below to verify your email address. {##Verify Email##} ",
    "EmailSubjectByLink": "Your verification link",
    "DefaultEmailOption": "CONFIRM_WITH_CODE",
    "EmailMessage": "Your verification code is {####}. ",
  }
},
["..."]
],
"EmailVerificationSubject": "Your verification code",
"MfaConfiguration": "OFF",
"Name": "RealObjects",
"EmailVerificationMessage": "Your verification code is {####}. ",
"VerificationMessageTemplate": {
  "EmailMessageByLink":
    "Please click the link below to verify your email address. {##Verify Email##} ",
  "EmailSubjectByLink": "Your verification link",
  "DefaultEmailOption": "CONFIRM_WITH_CODE",
  "EmailMessage": "Your verification code is {####}. ",
}
```

```

    "EmailSubject": "Your verification code",
    "SmsMessage": "Your verification code is {####}. "
  },
  "SmsAuthenticationMessage": "Your authentication code is {####}. ",
  "AdminCreateUserConfig": {
    "InviteMessageTemplate": {
      "EmailMessage": "Your username is {username} and temporary password is {####}. ",
      "EmailSubject": "Your temporary password",
      "SMSMessage": "Your username is {username} and temporary password is {####}. "
    },
    "UnusedAccountValidityDays": 7,
    "AllowAdminCreateUserOnly": false
  },
  "EmailConfiguration": {},
  "AutoVerifiedAttributes": [
    "email"
  ],
  "Policies": {
    "PasswordPolicy": {
      "RequireLowercase": true,
      "RequireSymbols": false,
      "RequireNumbers": true,
      "MinimumLength": 8,
      "RequireUppercase": true
    }
  },
  "UserPoolTags": {},
  "UsernameAttributes": [
    "email"
  ],
  "Id": "XXX",
  "Arn": "XXX",
  "LambdaConfig": {
    "CustomMessage": "arn:aws:lambda:eu-central-1:XXXXXXXXXXXX:function:SAMSA_user"
  }
}

```

## A.11 AngularJS Modulkonfiguration

```
1 Ui.app.config(function ($interpolateProvider, $routeProvider, $sceDelegateProvider) {
2     $interpolateProvider.startSymbol('[[');
3     $interpolateProvider.endSymbol(']]');
4     $routeProvider
5         .when('/', {
6             templateUrl: function () {
7                 let currentUser = Ui.userPool.getCurrentUser();
8                 if (!currentUser) return 'views/login.html';
9                 return 'views/start.html';
10            }
11        })
12        .when('/:temp', {
13            templateUrl: function (params) {
14                if (!Ui.userPool.getCurrentUser() && params.temp != "signup") {
15                    return 'views/login.html';
16                }
17                return 'views/' + params.temp + '.html';
18            }
19        })
20        .when('/:temp/:id/:key', {
21            templateUrl: function (params) {
22                return 'views/' + params.temp + '.html';
23            }
24        }).otherwise({
25            redirectTo: "/"
26        });
27     $sceDelegateProvider.resourceUrlWhitelist([
28         // Allow same origin resource loads.
29         'self',
30         // Allow loading from our assets domain.
31         'http://*.*.*/**'
32     ]);
33 });
34
```

## A.12 AngularJS Cognito-Service

```
1 Ui.app.service("cognito", function ($rootScope, config) {
2   let userPool = Ui.userPool;//new AmazonCognitoIdentity.CognitoUserPool(poolData);
3   if (typeof AWS !== 'undefined') {
4     AWS.config.region = config.cognito.region;
5   }
6
7   let signOut = function signOut() {
8     userPool.getCurrentUser().signOut();
9   };
10
11   let authToken = new Promise(function fetchCurrentAuthToken(resolve, reject) {
12     var cognitoUser = userPool.getCurrentUser();
13     if (cognitoUser) {
14       cognitoUser.getSession(function sessionCallback(err, session) {
15         if (err) {
16           reject(err);
17         } else if (!session.isValid()) {
18           resolve(null);
19         } else {
20           resolve(session.getIdToken().getJwtToken());
21         }
22       });
23     } else {
24       resolve(null);
25     }
26   });
27
28   function register(email, password, onFailure, onSuccess) {
29     var dataEmail = {
30       Name: 'email',
31       Value: email
32     };
33     var attributeEmail = new AmazonCognitoIdentity.CognitoUserAttribute(dataEmail);
34
35     userPool.signUp(email, password, [attributeEmail], null,
36       function(err, result) {
37         if (!err) {
38           onSuccess(result);
39         } else {
40           onFailure(err);
41         }
42       }
43     );
44   }
45
46   function signin(email, password, onSuccess, onFailure) {
47     let authenticationDetails = new AmazonCognitoIdentity.AuthenticationDetails({
48       Username: email,
49       Password: password
50     });
51
52     let cognitoUser = createCognitoUser(email);
53     cognitoUser.authenticateUser(authenticationDetails, {
54       onSuccess: onSuccess,
55       onFailure: onFailure
56     });
57   }
58
59   function resend(user) {
60     let cognitoUser = createCognitoUser(user);
61     cognitoUser.resendConfirmationCode(function(err, result) {
62       if (err) {
63         console.log(err);
64         return;
65       }
66     });
67   }
68
69   function createCognitoUser(email) {
70     return new AmazonCognitoIdentity.CognitoUser({
71       Username: email,
72       Pool: userPool
73     });
74   }
75 }
```

```

74 |
75 |     function getSession() {
76 |         return new Promise((resolve, reject) => {
77 |             let user = Ui.userPool.getCurrentUser();
78 |             if(!user) resolve(null);
79 |             user.getSession((err, ses) => {
80 |                 if(err) reject(err);
81 |                 else resolve(ses);
82 |             });
83 |         });
84 |     }
85 |
86 |     return {
87 |         signup: register,
88 |         login: signin,
89 |         logout: signOut,
90 |         authToken: authToken,
91 |         resend: resend,
92 |         session: getSession
93 |     };
94 | })
95 |

```

## A.13 Kundendokumentation

Die Kundendokumentation wurde aufgrund des weltweiten Vertriebs von PDFReactor in englischer Sprache verfasst. Aufgrund der ausgedehnten Entwicklerdokumentation wird diese an dieser Stelle sehr kurz gehalten und auf den Folgeseiten zu finden.

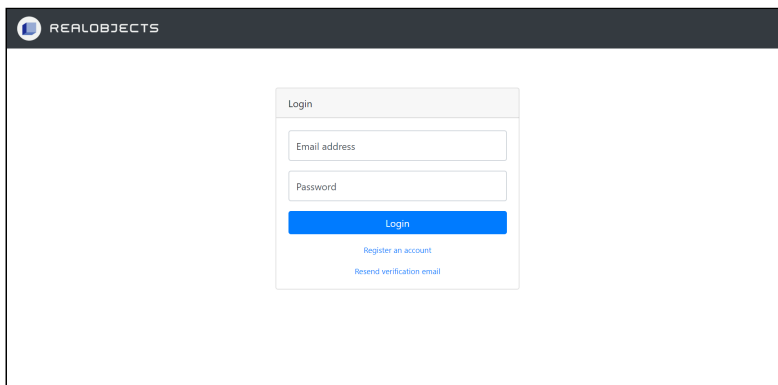
# SAMSA

## About

SAMSA (Subscription and Account Managementsystem in a Serverless Application) is a web-application for our customers to manage their subscriptions for RealObjects products.

## How to log in

Just visit our website <http://realobjects.samsa.com>. Visiting for the first time you will immediately be redirected to the log-in-view as seen below.



Just type your username and choose your password to get access to the management dashboard. If you don't have an account yet, just click "Register an account" right below the input fields in the log-in-panel. You will then be redirected to our sign-up-view. Please read below for further progression.

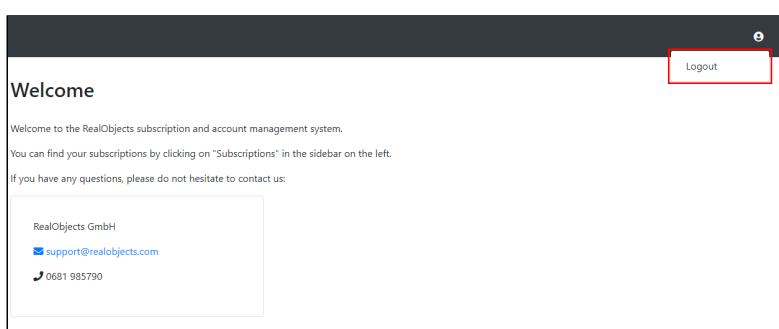
## How to register an account

Visit <http://realobjects.samsa.com#!/signup> or visit our log-in-view as seen above and click "Register an account". You will then be redirected to our sign-up-view. Enter your email-address (which has to be valid) and choose a password. After clicking "Register" you will receive a verification-email which contains a link which you have to visit in order to be able to log in. This ensures your email is valid and we are able to reach out to you.



## How to log out

Everywhere in SAMSA the user account symbol in the upper right corner is visible. By clicking this icon a dropdown menu will appear which contains "logout". After clicking it and confirming your choice to log out you will be logged out and redirected to the log-in-view.

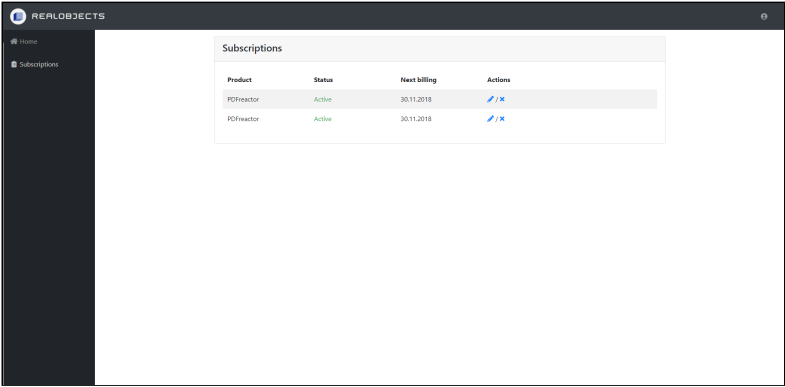






## How to get a subscription

In order to be able to use RealObjects products via subscription you have to purchase it. For example to purchase a PDFReactor subscription, visit <https://www.pdfreactor.com/buy/> and choose the required subscription. After your purchase, you will receive an email from us which contains a link to our website. By visiting our website by this link, you will be able to apply this subscription to your account by logging in or creating a new account.

## How to manage subscriptions

If you are logged into SAMSA, click on "Subscriptions" in the side-bar to the left which leads you directly to the subscription dashboard where all your subscriptions are listed like you can see below. In the column "Actions" are several buttons that will give you the ability to change or end your subscription.

A screenshot of a web application interface for 'REAROBJECTS'. The top header bar is dark grey with the 'REAROBJECTS' logo on the left and a user profile icon on the right. A dark sidebar on the left contains a 'Home' link and a 'Subscriptions' link, which is currently selected. The main content area displays a table titled 'Subscriptions'. The table has four columns: 'Product', 'Status', 'Next billing', and 'Actions'. It contains two rows of data, both for 'POreactor' products with an 'Active' status and a 'Next billing' date of '30.11.2018'. The 'Actions' column for each row contains two blue icons: a pencil (edit) and a trash can (delete).

Product	Status	Next billing	Actions
POreactor	Active	30.11.2018	 
POreactor	Active	30.11.2018	 

## **A.14 Entwicklerdokumentation**

Im Folgenden soll ein Auszug aus der Entwicklerdokumentation gezeigt werden.

# **SAMSA**

## **Produktbeschreibung**

### **1. Über**

SAMSA (Subscription and Account Management in a Serverless Application) ist eine Webanwendung mit welcher Kunden, die eine Subscription für eines der Produkte der RealObjects GmbH abgeschlossen haben, ihre Subscriptions verwalten können. Die Anwendung wurde vollständig in der AWS (Amazon Webservice) Umgebungen implementiert.

### **2. Struktur**

Die Webanwendung gliedert sich in drei Module:

1. Frontend; ein UI im Webseitenformat
2. Backend; eine Aggregation von verschiedenen Amazon-Services, deren Komposition den Webservice bildet, um u.a. das Frontend dynamisch mit Daten zu versorgen und die POST-Requests des E-Commerce-Dienstleisters empfängt.
3. Datenbank; Persistiert und liefert die Daten, welche vom Backend bearbeitet und abgerufen werden können und müssen

Jedes dieser Module ist wiederum in kleinere Module unterteilt, die im Folgenden spezifischer erklärt werden.

### **3. Frontend**

#### **3.1 Konfiguration**

Alle Dateien sind in einem Amazon S3 Bucket abgelegt, welcher diese als Webseite hosten soll. Um einen Bucket so zu konfigurieren, dass dieser eine statische Webseite hostet sind folgende Schritte notwendig (Anm.: hier wird davon ausgegangen, dass der wert Leser bereits in der AWS Konsole angemeldet ist und alle Berechtigungen besitzt):

1. Aktivieren des Website-Hostings
  - 1.1. Auswahl des Buckets, der für die gehostete Seite verwendet werden soll
  - 1.2. Öffnen der Registerkarte "Properties"
  - 1.3. Öffnen von "Static website hosting" und Auswahl von "Use this bucket to host a website"
  - 1.4. Eingabe von "index.html" in die Eingabe-Box "Index document"
  - 1.5. Speichern durch klicken auf "Save"
2. Erteilen der erforderlichen Berechtigungen für den Website-Zugriff
  - 2.1. Auswahl des Buckets, der die Seite hosten soll
  - 2.2. Öffnen der Registerkarte "Permissions"
  - 2.3. Auswahl von "Public Access Settings"
  - 2.4. Klicken auf "Edit"
  - 2.5. Entfernen der Häkchen unter "Manage public bucket policies for this bucket"

2.6. Speichern durch klicken auf "Save" und bestätigen

2.7. Auswahl von "Bucket Policy"

2.8. Einfügen von

```
{  
  "Version": "2012-10-17",  
  "Statement": [{  
    "Effect": "Allow",  
    "Principal": "*",  
    "Action": "s3:GetObject",  
    "Resource": "arn:aws:s3:::[YOUR_BUCKET_NAME]/*"  
  }]  
}
```

in den "Bucket policy editor", wobei "[YOUR\_BUCKET\_NAME]" durch den Namen des Buckets ersetzt werden muss.

2.9. Speichern durch klicken auf "Save"

In diesem Bucket sollten alle Dateien von SAMSA hochgeladen sein.

### 3.2 Ordnerstruktur

Die Ordnerstruktur von SAMSA sieht folgendermaßen aus:

```

S3
|--+ controllers
|   `-- _rootController.js
|
|--+ css
|   `-- style.css
|
|--+ img
|   `-- ...
|
|--+ vendors
|   |--+ angular
|   |   `-- ...
|   |
|   |--+ aws
|   |   `-- ...
|   |
|   |--+ bootstrap
|   |   `-- ...
|   |
|   `--+ jquery
|       `-- ...
|
|--+ views
|   |-- login.html
|   |-- logout.html
|   |-- purchase.html
|   |-- signin.html
|   |-- signout.html
|   |-- start.html
|   |-- subscriptions.html
|   `-- user.html
|
`-- index.html

```

Dabei beschreibt von oben nach unten:

controller	Ordner, in welchem die AngularJS Controller- Dateien angelegt werden
_rootController.js	Datei, in der sowohl der rootController als auch die Controller für einzelne Views zu finden sind
css	Ordner, in dem die Stylesheets angelegt wer- den
style.css	Das Stylesheet für die Be- nutzeroberfläche
img	Ordner, in dem Bilder und Icons zu finden sind
vendors	Ordner mit Bibliotheken für alle Frameworks
views	Ordner für alle verwen- deten Templates
index.html	Datei, welche den Rahmen für Templates, und Con- troller bildet

Alle angelegten Dateien und Ordner sollten möglichst verbos benannt werden, so dass später hinzukommende Entwickler intuitiv die korrekten Ordner und Dateien finden können.

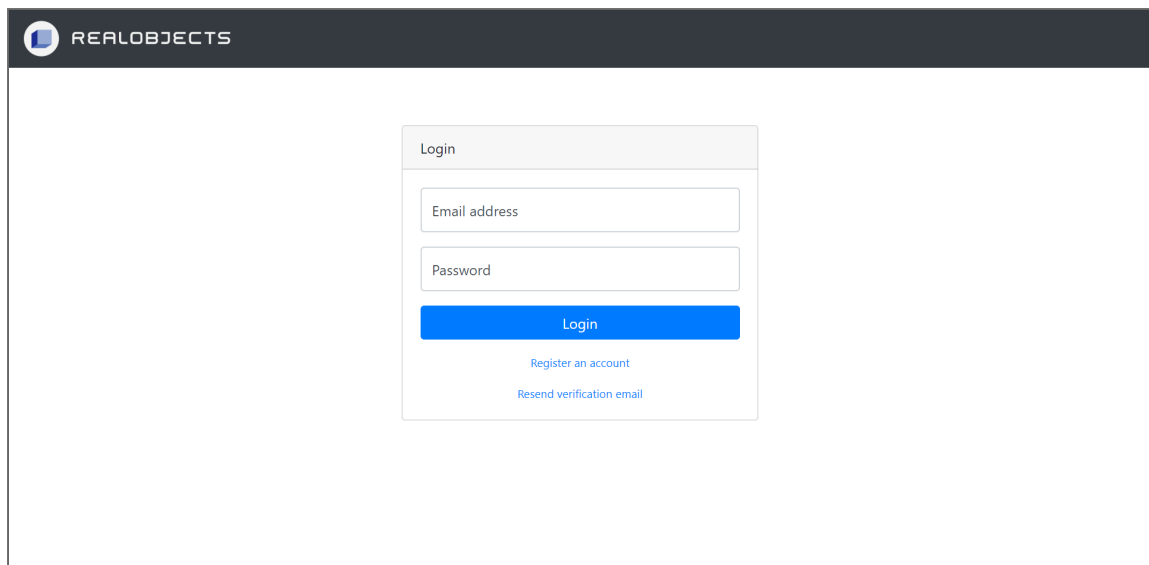
### 3.3 Routing

Für das Routing der Views wird "ng-route" von AngularJS verwendet. Alle Dateien, die in "views" abgelegt werden und die Endung ".html" besitzen sind via Dateinamen über die erzeugte Route erreichbar. Hierbei wird eine Pfadstruktur simuliert, welche der URL als Hashparameter mit vorangestelltem "!" angehängt wird. Das heißt, das Template, das in "test.html" gespeichert wurde, ist dann unter "#!/test" erreichbar.

## A.15 Ordnerstruktur der statischen HTML-Webseite

```
S3
| -+ controllers
|   \- _rootController.js
|
| -+ css
|   \- style.css
|
| -+ img
|   \- ...
|
| -+ vendors
|   | -+ angular
|   |   \- ...
|   |
|   | -+ aws
|   |   \- ...
|   |
|   | -+ bootstrap
|   |   \- ...
|   |
|   \+ jQuery
|       \- ...
|
| -+ views
|   |-- login.html
|   |-- logout.html
|   |-- purchase.html
|   |-- signin.html
|   |-- signout.html
|   |-- start.html
|   |-- subscriptions.html
|   \- user.html
|
\ - index.html
```

## A.16 Benutzeroberfläche



The image shows a login form for the REALOBJECTS website. The form is centered on a white background with a dark header bar at the top containing the REALOBJECTS logo. The form itself has a light gray border and a title 'Login' at the top. It contains two input fields: 'Email address' and 'Password'. Below these fields is a blue 'Login' button. At the bottom of the form, there are two links: 'Register an account' and 'Resend verification email'.

REALOBJECTS

Login

Email address

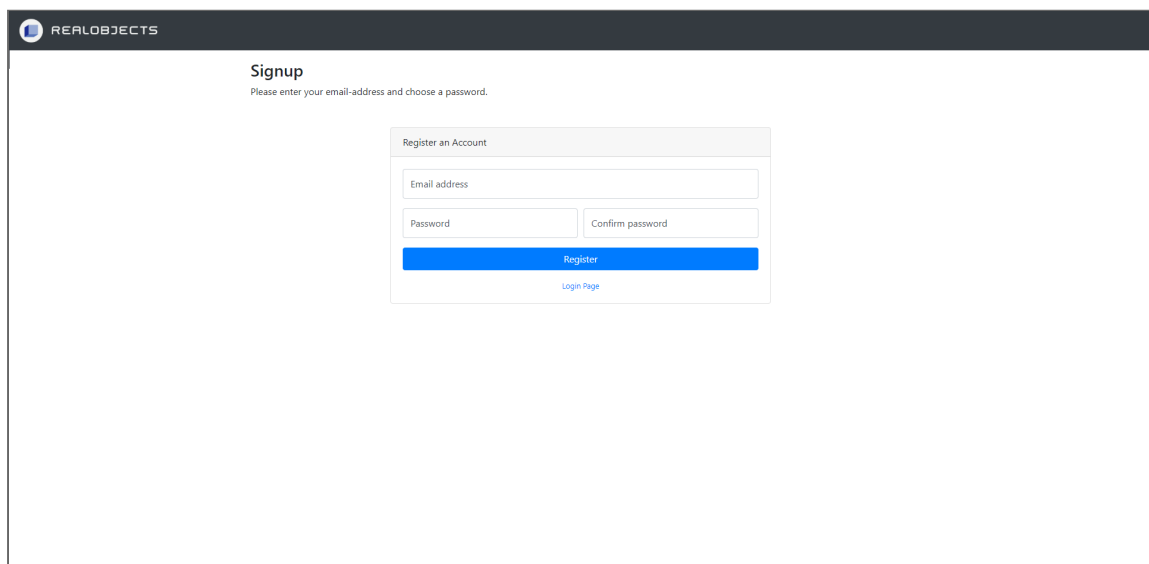
Password

Login

[Register an account](#)

[Resend verification email](#)

Abb.: Login-Ansicht



The image shows a signup form for the REALOBJECTS website. The form is centered on a white background with a dark header bar at the top containing the REALOBJECTS logo. The form has a light gray border and a title 'Signup' at the top, followed by the instruction 'Please enter your email-address and choose a password.' Below this is a sub-form titled 'Register an Account' which contains three input fields: 'Email address', 'Password', and 'Confirm password'. At the bottom of this sub-form is a blue 'Register' button. Below the sub-form, there is a link 'Login Page'.

REALOBJECTS

**Signup**  
Please enter your email-address and choose a password.

Register an Account

Email address

Password

Confirm password

Register

[Login Page](#)

Abb.: Registrierungs-Ansicht



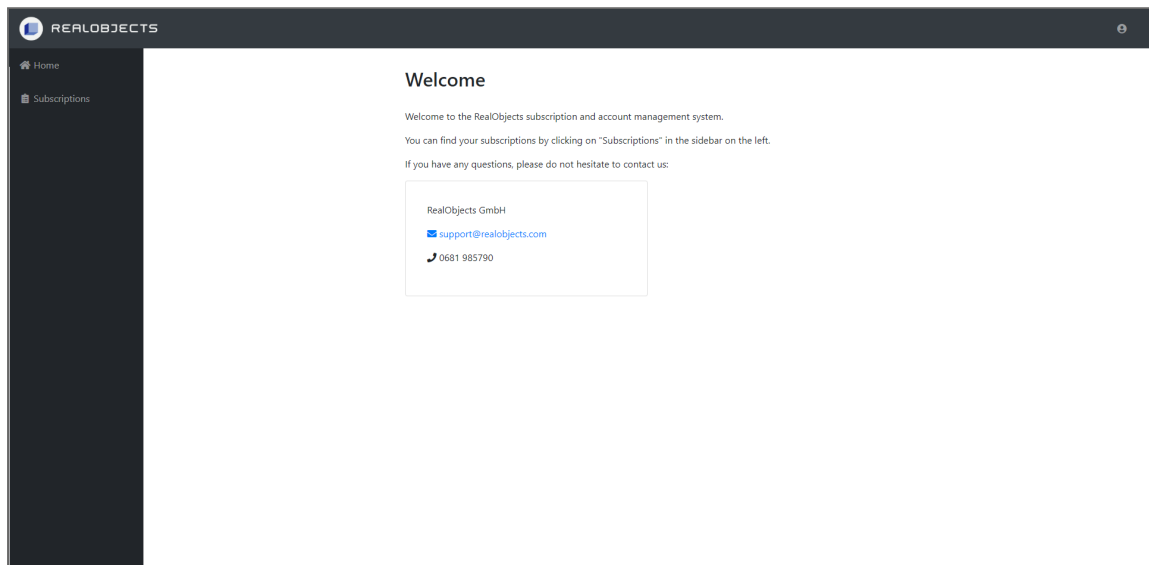


Abb.: Willkommensseite

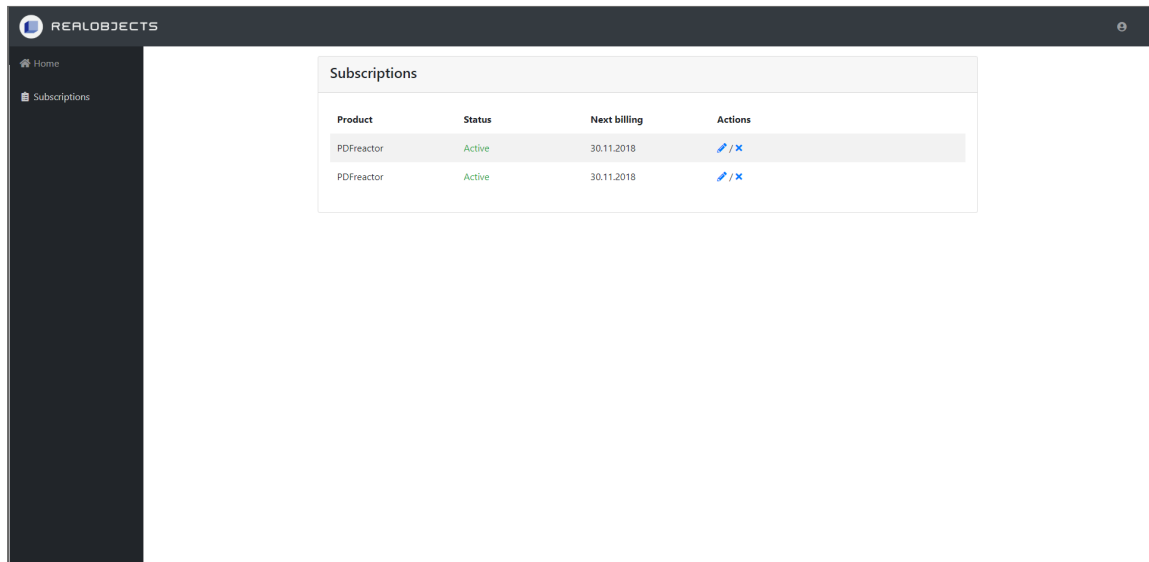


Abb.: Subscription-Dashboard