Projekt Obst vom Baum: Backend

Bachelorarbeit

im Studiengang Wirtschaftsinformatik

vorgelegt von

Flavian Thür
Rudolfstrasse 25, 8400 Winterthur
Matr.-Nr.: 16562274

Frühlingssemester 2020
Am 27. Mai 2020

an der ZHAW School of Management and Law

Betreut von

David Grünert

Management Summary

Aufgrund der Klimaerwärmung wird Nachhaltigkeit in allen Lebensbereichen immer wichtiger. Lebensmittelverschwendung bringt eine CO2-Emmisson mit sich, welche in vielen Fällen nicht nötigt wäre. Die vorliegende Arbeit ist ein Teil vom Projekt «Obst vom Baum». Das Projekt selbst hat zum Ziel die Verschwendung von Obst in der Schweiz zu vermindern. Das Endprodukt des Projekts «Obst vom Baum» soll eine Webapplikation sein, über welche nicht geerntetes Obst angeboten und nachgefragt werden kann. Die vorliegende Arbeit baut auf Bachelorarbeiten auf, welche als erster Teil des Projektes bereits verfasst wurden.

Ziel der vorliegenden Arbeit ist, den bestehenden Programmcode aus den vorgängigen Bachelorarbeiten so zusammenzuführen, dass ein automatisiertes Backend für den produktiven Einsatz der Webapplikation zur Verfügung steht. Ausgehend von den Analysen der letztjährigen Arbeiten, wird dabei ein Fokus auf die Performance der zu implementierenden Funktionalitäten gelegt.

Neben der vorliegenden Arbeit wird zeitgleich das Frontend der Webapplikation implementiert. Dies hat Einfluss auf das gesamte Vorgehen. Somit wird bei der Erarbeitung von Lösungen, stark auf die Anforderungen des Frontends eingegangen. Initial wird der Programmcode der letztjährigen Arbeiten, sowie die Erkenntnisse analysiert. Dies ist die Grundlage für die weitere Erarbeitung von Programmcode. Die Anforderungen werden in Zusammenarbeit mit dem Dozenten und dem Studierenden der Bachelorarbeit «Frontend» erarbeitet. Weiter wird für die Umsetzung des Programmcodes Internetrecherche betrieben oder Tests durchgeführt.

Der Programmcode ist so implementiert, dass die gestellten Anforderungen erfüllt wurden. Verglichen zu den letztjährigen Arbeiten, wurden diverse Anpassungen an der Datenbankstruktur und den Funktionalitäten des Backends gemacht. Am Framework der vorgängigen Bachelorarbeit, von JavaSpringBoot mit Anbindung an eine MongoDatenbank, wird weiterhin festgehalten.

Als Fazit steht, dass der implementierte Programmcode soweit für den produktiven Betrieb bereit ist, die gewünschten Funktionalitäten umgesetzt wurden und die durchgeführten Tests erfolgreich waren. Weiter können Standard-Aufgaben, welche durch einen Administrator durchzuführen sind, einfach abgearbeitet werden.

Für künftige Implementationen stehen mehrere Möglichkeiten zur Verfügung. Ein Ausbau der Testautomatisierung ist erstrebenswert. So können zukünftige Implementationen, einfacher getestet und somit auch schneller implementiert werden. Eine weitere Funktionalität, welche Sinn macht, sobald die Nutzerzahlen steigen, ist die Integration der Administrationsaufgaben in das Frontend. Dies würde auch Anpassungen im Backend bedeuten, da entsprechende Requests noch fehlen. Da die Bauern von den Bauernverbänden validiert werden und diese Arbeit für die Qualität der Daten sehr wichtig ist, sollte man ebenfalls allfällige Anforderungen der Bauernverbände berücksichtigen. Ebenfalls gibt es noch diverse kleinere Anpassungsmöglichkeiten, wie Programmcode-Refactoring, UI-Verbesserungen im E-Mail-Service und zusätzliche Tasks für die Datenbankbereinigung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Projekt «Obst vom Baum» hat zum Ziel eine Plattform zu entwickeln und zu lancieren, auf welcher Obst angeboten werden kann. Dabei wird der Fokus auf Ökologie sowie Einfachheit gelegt. Auf der Plattform soll ausschliesslich Obst angeboten werden, welches nicht kommerziell vertrieben werden kann oder gar nicht erst geerntet wurde. Dabei spielt es keine Rolle, ob das Obst von Privatpersonen oder von einem Bauern stammen. Die Plattform soll den Anbietern ermöglichen sein Obst so einfach wie möglich zu erfassen. Einem Nachfrager sollen nur Angebote angezeigt werden, welche ökologisch sinnvoll sind. Um die Plattform einfach zu halten, ist in einer ersten Version vorgesehen, dass Anbieter und Nachfrager direkt miteinander kommunizieren (über Telefon oder E-Mail) sobald diese untereinander vermittelt wurden.

1.1 Problemstellung

Die Verschwendung von Lebensmitteln (Foodwaste) hat einen enormen Einfluss auf das Klima (Rhyn, 2019). Nicht nur die verschwendeten Lebensmittel an sich, sondern auch die damit verbundenen Biodiversitätsverluste sowie Land- und Wasserverbrauch führen zu unnötigen CO2-Emmissionen (BAFU, 2019). Dabei sind 13 Prozent der in der Schweiz verschwendeten Lebensmittel auf die Landwirtschaft zurückzuführen (BAFU, 2019). Grund dafür ist, dass die Grossverteiler hohe Standards haben, was die Qualität des Obsts angelangt (Baier et al., 2016, S. 25). Daher kommt ein Teil des Obsts gar nicht in die Läden (Baier et al., 2016, S. 25).

Weiter zeigt der Erfolg von Projekten und Unternehmen, die sich gegen Foodwaste einsetzen, dass die Thematik immer mehr an gesellschaftlicher Bedeutung gewinnt. Am Beispiel der Wachstumszahlen der Anti-Foodwaste Applikation «TooGoodToGo», kann aufgezeigt werden, dass aus Unternehmens- *und* Konsumentensicht eine hohe Nachfrage besteht, wenn durch den Kauf von Lebensmitteln der Foodwaste reduziert werden kann (TooGoodToGo, 2020). Das Projekt «Obst vom Baum» will das zunehmende Bewusstsein der Bevölkerung nutzen, um die Problematik des Foodwaste in der Landwirtschaft zu bekämpfen.

1.2 Forschungsfragen

Da das Projekt «Obst vom Baum» nicht kommerziell betrieben werden soll, ist es wichtig, die Administrationsarbeiten, möglichst gering zu halten. Um dieses Ziel zu erreichen,

braucht es ein Backend, welches automatisiert ist und somit nur in Spezialfällen eine zusätzliche Interaktion von einem Menschen benötigt.

Aus der Motivation für das Backend leitet sich folgende Fragestellung ab:

- Wie kann das Backend so automatisiert werden, dass möglichst wenig Eingriffe eine manuelle Interaktion benötigen?

Dabei sollen nachfolgende Teilfragen beantwortet werden:

- Welches sind die relevanten Prozesse für das Backend?
- Wie sehen die Anforderungen an das Backend aus?
- Welche Prozesse müssen im Vergleich zur bestehenden Implementation verändert oder komplett umgebaut werden?
- Wie werden die Teilprojekte der letztjährigen Bachelorarbeiten am sinnvollsten zusammengeführt?
- Wie kann ohne manuelle Kontrolle eine hohe Datenqualität sichergestellt werden?
- Welche Funktionalitäten werden sinnvollerweise vom Frontend ins Backend verlagert?
- Bei welchen Prozessen macht es aufgrund der Nutzungshäufigkeit und der Implementationskomplexität keinen Sinn eine Automatisierung zu implementieren?
- Wie wird die Implementation weiterer Funktionalitäten am einfachsten durchgeführt?
- Wie werden die neu entwickelten Funktionalitäten getestet?
- Wie wird für künftige Implementationen ein Regressionstesting sichergestellt?

1.3 Beitrag der Arbeit

Am Ende dieser Arbeit soll der Programmcode vom Backend so weiterentwickelt sein, dass die definierten Prozesse funktionieren und somit die Plattform in einer ersten Version einsatzfähig ist.

2 Vorgehen und Methoden

Da die Bachelorarbeit zu einem grossen Teil aus dem Erstellen von Programmcode besteht, ist ein aktiver Austausch mit dem Dozenten nötig. Zeitgleich wird in einer zweiten Bachelorarbeit das Frontend der Applikation entwickelt. Daher besteht zu dieser Arbeit eine grosse Interdependenz. Daher werden wöchentliche Abstimmungsmeetings stattfinden, bei welchen die beiden Verfasser und der Dozent sich über den Stand der Arbeiten austauschen.

Im letzten Jahr wurden bereits vier Bachelorarbeit im Zusammenhang mit dem Projekt «Obst vom Baum» geschrieben. Die letztjährigen Bachelorarbeiten setzten sich mit folgenden Themen auseinander:

- User Interface & User Experience
- Ausarbeitung eines Geschäftsmodells, Implementierung und Lancierung einer ersten Version (Matching)
- Backend
- Datenvalidierung und rechtliche Abklärungen

In diesen Arbeiten wurde auch schon diverser Programmcode entwickelt, auf welchem nun aufgebaut werden kann. Daher wird am Anfang eine Analyse der letztjährigen Bachelorarbeiten und des vorhandenen Programmcodes durchgeführt. Der Fokus liegt vor allem auf den Programmcode der Bachelorarbeiten Backend, Matching und Datenvalidierung gelegt.

Das weitere Vorgehen und der Aufbau der Bachelorarbeit sind stark von der Aufgabenstellung getrieben und sind in die Teilschritte Prozesse, Anforderungen, Design, Implementation und Testing unterteilt. Nachfolgend werden diese Teilschritte und die Methoden, welche dabei angewandt werden, beschrieben. Die genannten fünf Schritte sind nicht strikte chronologisch einzuhalten. Vielmehr dienen sie einer groben Orientierung, wie weit die Implementierung der einzelnen Funktionalitäten und Prozesse ist.

2.1 Prozesse

In einem ersten Schritt werden die *Prozesse* aus der Sicht des End-Users analysiert. Dabei wird mit dem Dozenten und dem Studierenden, welcher die Bachelorarbeit für das Frontend schreibt, zusammengearbeitet, damit ein gemeinsames Verständnis für die Prozesse

geschaffen wird. Um diesen Schritt durchzuführen werden die einzelnen Prozesse modelliert, wobei auf die Schnittstelle zwischen Frontend und Backend geachtet wird. Weiter wird mittels Szenarien überprüft, ob die modellierten Prozesse vollständig oder überflüssig sind.

2.2 Anforderungen

In einem weiteren Schritt werden die *Anforderungen* an das Backend analysiert und festgehalten. Auch bei diesem Teilschritt sind die Anforderungen des Frontends, aber auch die nicht funktionalen Anforderungen zu berücksichtigen.

2.3 Software Design

Im Schritt *Design* wird analysiert welches Software-Design implementiert werden soll. Dabei werden einerseits die Anforderungen und andererseits die Komplexität für die Implementation analysiert. Bei der Analyse des zu implementierenden Software-Design wird einerseits Internet-Recherche berücksichtigt, aber auch Erfahrungen vom Studierenden als auch vom Dozenten miteinfliessen.

2.4 Implementation

In einem weiteren Schritt wird die *Implementation* beschrieben und der Programmcode auch implementiert. Für diesen Schritt wird einerseits stark auf den bereits bestehenden Programmcode zurückgegriffen, aber auch mittels Internetrecherchen neuer Programmcode implementiert. Auch hier fliessen Erfahrungen vom Dozenten und den Studierenden mit ein.

2.5 Testdesign

Der implementierte Programmcode wird im letzten Schritt, *Test*, auf Fehler überprüft. Das Testing soll so aufgebaut werden, dass dieses in Zukunft wiederverwendet werden kann. Beim Testing wird eine positive Teststrategie verfolgt. Die Testcases müssen nicht automatisiert, aber ohne grossen Aufwand und Vorkenntnisse manuell durchgeführt werden können. Auch in diesem Schritt wird Internet-Recherche betrieben, wie dieses Ziel am besten erreicht werden kann.

3 Ausganslage

In diesem Kapitel ist die Ausgangslage und somit die Analyse der letztjährigen Bachelorarbeiten des Projekts «Obst vom Baum» beschrieben. Zuerst wird die generelle Ausgangslage und somit die Systemlandschaft analysiert und beschrieben. In weiteren Unterkapiteln werden die genaueren Umsetzungen der einzelnen Bachelorarbeiten aufgezeigt. Die letztjährige Bachelorarbeit «User Interface & User Experience» wird dabei nicht berücksichtigt, da sie für die vorliegende Arbeit keine Relevanz hat.

3.1 Generelle Ausganslage

Die letztjährigen Bachelorarbeiten wurden als Applikationen umgesetzt, welche eigenständig funktionieren und mittels REST-API miteinander kommunizieren. Dazwischen ist ein Login-Server geschaltet, welcher die Requests initial abfängt, die Authentifizierung prüft und an den entsprechenden Server (Applikation) weiterleitet. Die jeweiligen Responses des Servers werden dann wiederum über den Login-Server weitergeleitet. Zur Verdeutlichung ist die Architektur aus den letztjährigen Arbeiten in Abbildung 1 aufgezeigt.

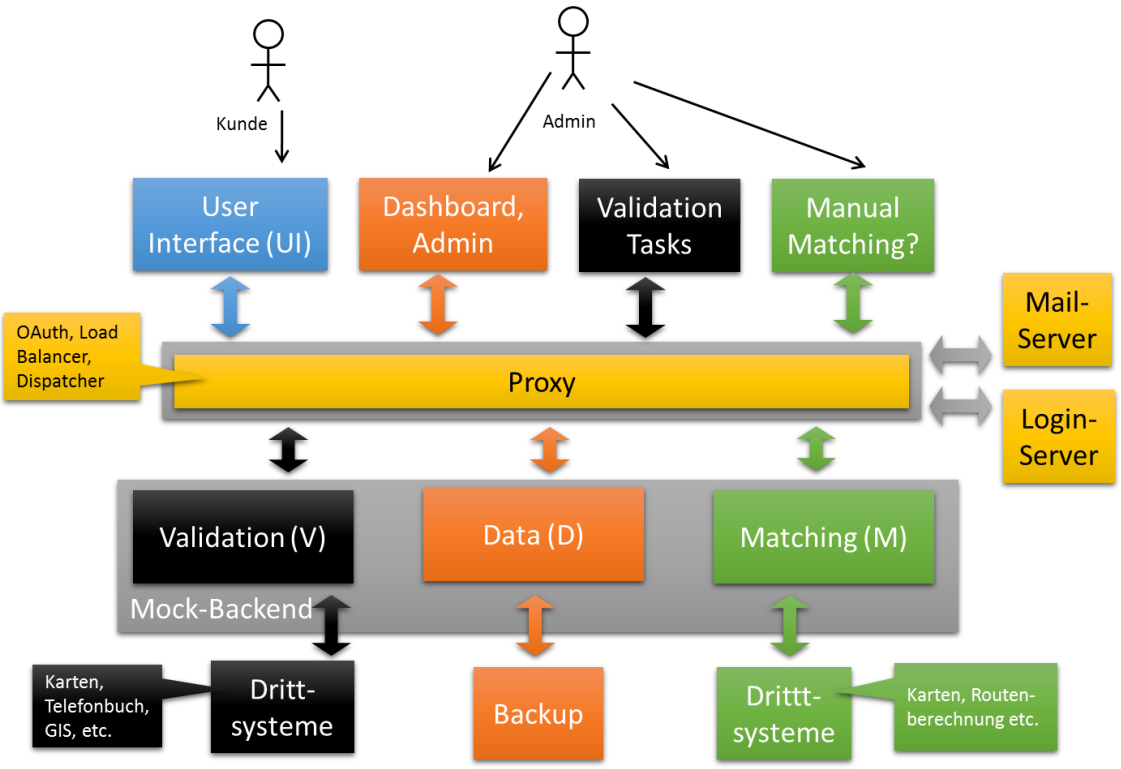


Abbildung 1: Ausganslage Systemarchitektur(Grünert, 2019, S. 1)

Der in Abbildung 1 dargestellte Login-Server wurde vom Dozenten entwickelt und liegt in seiner Verantwortung. Die einzelnen Systeme sind verantwortlich für die eigenen implementierten Funktionalitäten und dass diese, falls nötig, für die anderen Systeme mittels REST-API aufgerufen werden können. Um die jeweiligen Applikationen zu deployen wurden die PaaS-Dienstleistung von Jelastic Cloud benützt. Nachfolgend wird die Ausganslage in den einzelnen Applikationen beschrieben. Dabei wir ein Fokus auf den implementierten Programmcode gelegt.

3.2 Backend

Im Backend werden die Daten persistiert. Das Backend ist eine JavaSpring Boot Appliaction, welche durch das entsprechende Repository an einer No-SQL-Datenbank, genauer einer MongoDB, angeschlossen ist. Um die Abstraktion so einfach als möglich zu halten, sind die Entitäts-Klassen auf ein Minimum reduziert, was aber auch eine kleinere Flexibilität bei der Implementierung von Funktionalitäten bedeutet. Das Datenmodel für die REST-Aufrufe sind mehrere Controller Klassen umgesetzt. Um Daten in der MongoDB zu bearbeiten sind entsprechende Repositories für die einzelnen Java-Klassen aufgesetzt. Weiter ist ein Admin-Dashboard implementiert, über welches simple Requests durchgeführt werden können.

3.3 Matching

In der Applikation Matching ist die Logik implementiert, die definiert, welcher User, welches Angebot buchen kann. Dabei wird ein Angebot nur einem User zum Buchen bereitgestellt, wenn es CO2-neutral ist. Diese Implementation wird auch als Matching Service bezeichnet. Im Matching Service ist zudem ein Mail-Service implementiert, welcher in gewissen Fällen eine E-Mail an die User verschickt. Dabei wird ein Mailserver von Infomaniak genutzt. Weiter ist ein API zu Googlemaps implementiert, welche die Fahrtdistanz zwischen zwei Orten zurückgeben kann.

3.4 Validierung

Der Validierungs-Server stellt diverse Validierungsfunktionen zur Verfügung, sodass der Backend-Server diverse Datenmanipulationen mittels REST-Requests verifizieren kann. Dies dient dazu, dass die Daten, bevor diese in der Datenbank persistiert werden, auf ihre Qualität geprüft werden. Auch im Validierungs-Server ist ein Mail-Service angebunden. Dieser wird für den Versand eines E-Mails bei der Registration verwendet, um die E-

Mail-Adresse zu bestätigen. Weiter sind APIs zu Drittsystemen, wie zum Beispiel zu local.ch für die Validierung einer Adresse oder zu Clarifai für die Validierung eines Bildes umgesetzt.

4 Prozesse

In diesem Kapitel werden die Prozesse genauer beschrieben. Diese wurden in Zusammenarbeit mit dem Frontend übergreifend erarbeitet. Die gemeinsam erarbeiteten Prozesse sind die Grundlage für die in diesem Kapitel beschriebenen Backend-Prozesse. Dabei handelt es sich um Prozesse, welche vom User ausgeführt werden können. Prozesse, welche nicht manuell von einem User durchgeführt werden, sondern automatisch vom System angestossen werden, werden im Kapitel 7 beschrieben. Weiter gelten die nachfolgenden Prozesse als Happy-Case und es wird nicht jede Eventualität erwähnt. Sie sollen einen groben Überblick über die fachlichen Funktionalitäten geben. Für die einzelnen Prozesse gibt es jeweils ein Unterkapitel, worin ein entsprechendes Flussdiagramm und eine Prozessbeschreibung enthalten sind.

4.1 Registrierung Privatperson

Nachfolgend wird der Prozess beschrieben, wie sich eine Privatperson registrieren kann. Dabei spielt es keine Rolle, ob die Privatperson als Anbieter, Nachfrager oder beides agiert. In der Abbildung 2 ist der Prozess in einem Flussdiagramm aufgezeichnet, wobei eine Trennlinie zwischen Backend und dem Umsystem (Frontend / E-Mail / Useraktion) gemacht wird. Der Prozess an sich wird im Frontend ausgelöst, wo die relevanten Daten für die Registrierung erfasst werden.

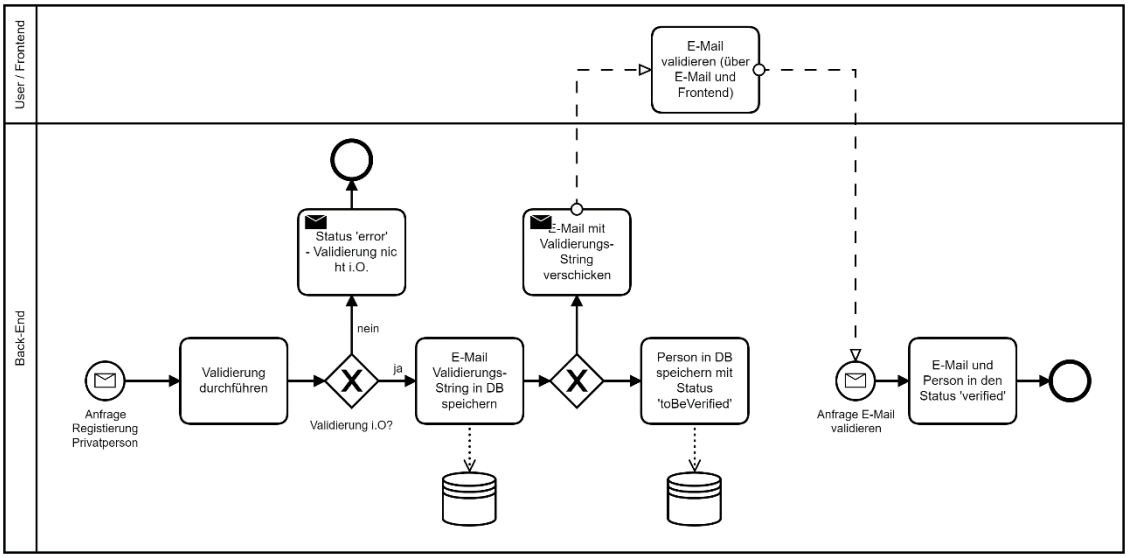


Abbildung 2: Flussdiagram Registrierung Privatperson

Das Backend wird vom Frontend initial mittels Request aufgerufen was in Abbildung 2 als Start-Event (Kreis mit Brief) auf der linken Seite des Diagramms dargestellt ist. Zu Beginn werden die Daten, wie z.B. E-Mail-Adresse, Name und Passwort *validiert*. Wenn die Validierung fehlschlägt, wird sogleich eine Rückmeldung an das Frontend in Form eines Error-Status gegeben. Ist die Validierung erfolgreich wird als nächster Prozessschritt ein Token (Validierungs-String) für die Validierung der E-Mail-Adresse generiert und in der Datenbank gespeichert. Der Token wird dann an die bei der Registrierung angegebene E-Mail-Adresse geschickt. Weiter wird die Person in einem temporären Status in der Datenbank gespeichert. Der User ruft über einen Link in der verschickten E-Mail die Validierung am Frontend auf. Diese löst wiederum einen Request ans Backend aus (zweiter Kreis mit Brief) welcher dann die E-Mail-Adresse und die Person validiert und in der Datenbank persistiert.

4.2 Registrierung Landwirtschaftsbetrieb

In diesem Kapitel ist der Prozess der Registrierung eines Landwirtschaftsbetriebs beschrieben. Im Gegensatz zur Privatperson, kann ein Landwirtschaftsbetrieb nur als Anbieter agieren und für das Obst einen Preis verlangen. Um sicherzustellen, dass hinter einer registrierenden Person auch wirklich ein Landwirtschaftsbetrieb steckt, gibt es in diesem Prozess noch zusätzliche Validierungen. In der Abbildung 3 ist das Flussdiagramm aufgezeigt, in welchem wieder eine Trennung zwischen Backend und den restlichen Aktionen gemacht wird. Auch dieser Prozess wird am Frontend ausgelöst.

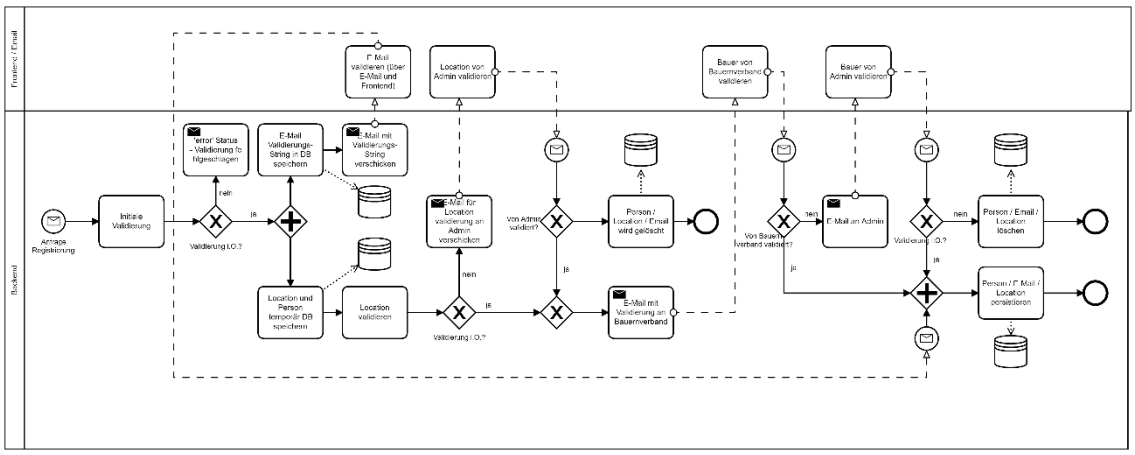


Abbildung 3: Flussdiagramm Registrierung Landwirtschaftsbetrieb

Der Prozess startet für das Backend mit einem Request und die Daten werden initial *validiert*. Fällt die Validierung negativ aus, wird ein Error-Status zurückgeschickt. Sofern

die initiale Validierung positiv ausfällt wird der Prozess aufgeteilt, was in der Abbildung 3 als Plus dargestellt ist. Einerseits wird ein Token für die E-Mail-Validierung generiert, in der Datenbank gespeichert und an den User verschickt. Andererseits wird die Adresse (Location) und die Person temporär in der Datenbank gespeichert. Nach der temporären Speicherung wird die Location genauer validiert. Wenn diese Prüfung negativ ausfällt muss eine manuelle Validierung von einem Admin ausgeführt werden. Sobald auch die Location automatisch oder manuell validiert wurde, wird eine E-Mail an den kantonalen Bauernverband des Domizilkantons des registrierenden Landwirtschaftsbetriebs geschickt. Im Idealfall läuft der Prozess bis zum Versand dieses E-Mails automatisch. Der kantonale Bauernverband prüft, ob der entsprechende Landwirtschaftsbetrieb Mitglied beim Bauernverband ist oder nicht. Sollte diese Prüfung negativ ausfallen, ist wiederum eine manuelle Prüfung durch einen Admin-User verlangt. In Abbildung 3 kann man beim rechten Plus erkennen, dass sobald diese Prüfungen positiv ausgefallen sind und der User seine E-Mail-Adresse validiert hat die Daten *persistent in der Datenbank gespeichert* werden.

4.3 Einloggen

Der Prozess des Logins beschreibt, wie sich ein User auf der Plattform einloggt. Da es sich dabei um einen sehr einfachen Prozess handelt, wird kein Flussdiagramm erstellt.

Vom Frontend wird eine Anfrage mit E-Mail-Adresse und Passwort an das Backend geschickt. Diese E-Mail/Passwort Kombination wird überprüft. Entweder wird eine Success-Meldung mit einem Authentifizierung-Token zurückgeschickt oder eine Error-Meldung, dass die Daten nicht authentifiziert werden konnten.

4.4 Passwort zurücksetzen

Dieser Prozess beschreibt, wie ein bereits registrierter User sein Passwort zurücksetzen kann. In Abbildung 4 ist das entsprechende Flussdiagramm ersichtlich. Um diesen Prozess auszulösen, muss die entsprechende E-Mail-Adresse in der Datenbank bei einem User hinterlegt sein. Der Prozess wird am Frontend ausgelöst.

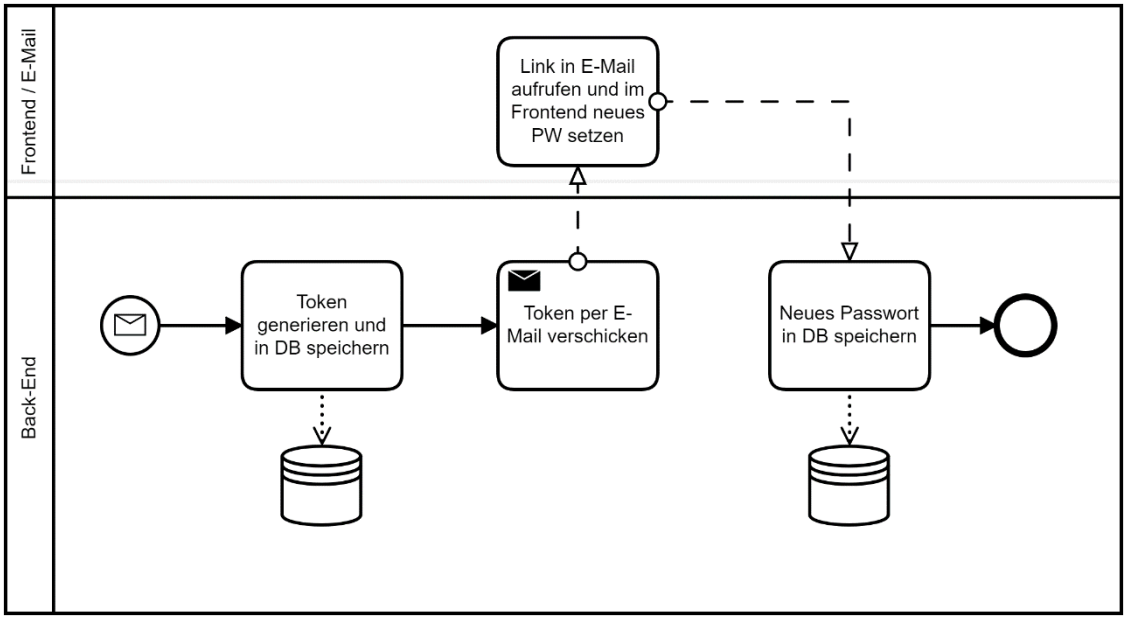


Abbildung 4: Flussdiagramm Passwort zurücksetzen

Aus der Sicht vom Backend beginnt der Prozess mit den Request. Der Input ist nur die E-Mail-Adresse des Users. Als erstes wird ein Token generiert und in der Datenbank gespeichert. Dieser Token wird an die angegebene E-Mail-Adresse verschickt, mit welchem dann über das Frontend ein neues Passwort eingeben werden kann. Vom Frontend wird ein Request, mit dem neuen Passwort und dem vorher generierten Token ausgelöst. Durch diesen Request wird abschliessend für den entsprechenden User das neue Passwort in der Datenbank gespeichert.

4.5 Person / User bearbeiten

Dieser Prozess beschreibt, wie ein bestehender User seine Daten bearbeiten kann. Obwohl gewisse Validierungen und Attribute unterschiedlich sind, wird im grundsätzlichen Prozess nicht zwischen Privatperson und Landwirtschaftsbetrieb unterschieden. In Abbildung 5 ist der Prozess in einem Flussdiagramm abgebildet. Der Prozess wird wiederum vom Frontend heraus ausgelöst.

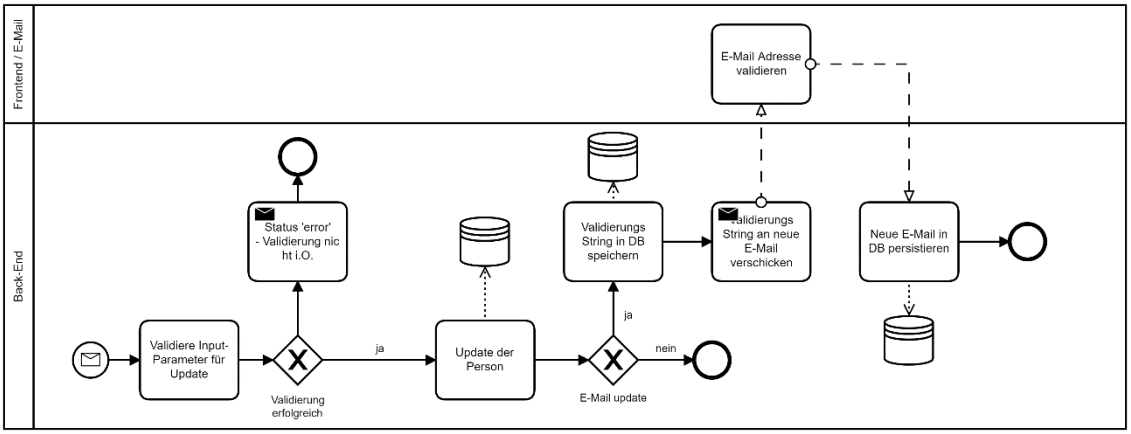


Abbildung 5: Flussdiagramm Person / User bearbeiten

Der Prozess für das Backend beginnt mit einem eingehenden Request für die Aktualisierung der Personendaten. Zuerst werden die Input-Daten *validiert* und sofern diese nicht in Ordnung sind, wird ein entsprechender Error-Status als Antwort zurückgeschickt. Wenn die Validierung in Ordnung ist werden die Daten in der Datenbank gespeichert. In der Mitte der Abbildung 5 kann man erkennen, dass der Prozess abgeschlossen ist, sofer die E-Mail-Adresse nicht aktualisiert wird. Wird die E-Mail-Adresse jedoch aktualisiert, so geht der Prozess noch weiter. Ähnlich wie bei der initialen Registrierung wird ein Token generiert, welcher an die neue E-Mail-Adresse zur Validierung verschickt wird. Wenn der User mit einem Link das Frontend aufruft wird ein Request ausgelöst, welcher die neue E-Mail-Adresse final in der Datenbank speichert.

4.6 Produkt und Angebot erfassen und ändern

Dieser Prozess beschreibt, wie ein potenzieller Anbieter sein Obst in die Plattform stellen kann. Voraussetzung ist, dass es sich um einen komplett validierten User handelt. Dabei spielt es keine Rolle, ob es sich um eine Privatperson oder um einen Landwirtschaftsbetrieb handelt. Da der Prozess sehr einfach gehalten werden kann, wird kein entsprechendes Flussdiagramm erstellt. Der Prozess beginnt im Frontend, wo der User sein Obst und dessen Abholmöglichkeiten erfasst.

Auslöser für das Backend ist ein Request mit den Input-Daten, welches Obst zu welchem Zeitpunkt, in welcher Form abgeholt werden kann. Dabei werden die Daten zuerst validiert. Entweder wird eine Error-Meldung zurückgeschickt oder die Daten werden erfolgreich in der Datenbank gespeichert.

4.7 Angebote anzeigen

In diesem Prozess wird aus Sicht des Nachfragers beschrieben wie die Angebote angeschaut werden können. Dieser Prozess kann nur von einer Privatperson durchgeführt werden, da Landwirtschaftsbetriebe keine Angebote buchen können und ihnen daher auch nicht angezeigt werden müssen. Dabei kann der User die grundsätzlichen Daten, wie z.B. den Ort des Angebots, einsehen. So kann sich der User ein Bild darüber machen, welches Angebot er buchen möchte. Weiter hat er die Möglichkeit gewisse Filterkriterien zu setzten, sodass nicht alle Angebote angezeigt werden, sondern nur die von ihm gefilterten. Das Flussdiagramm in Abbildung 6 zeigt den entsprechenden Prozess. Der Prozess wird am Frontend ausgelöst, indem ein User die Suchresultate seiner Angebote anzeigen lässt.

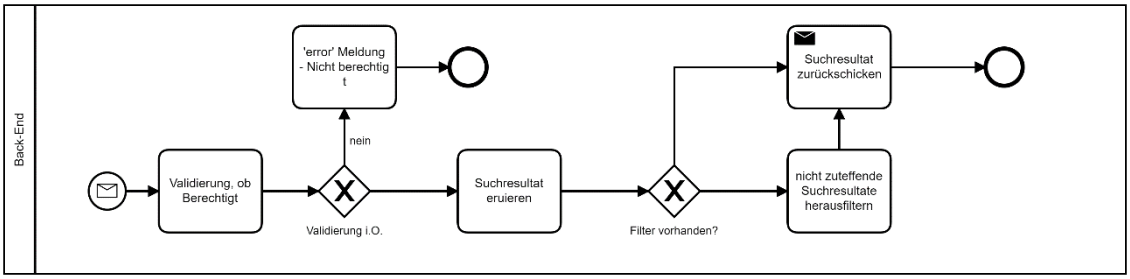


Abbildung 6: Flussdiagramm Angebote anzeigen

Ist der User berechtigt, so wird das Suchresultat des eingeloggten Users eruiert. Wurde initial ein Filter mitgegeben (z. B. nur Äpfel anzuzeigen), so werden alle Suchresultate, welche die Filterkriterien nicht erfüllen herausgefiltert. Zum Schluss wird das (gefilterte) Suchresultat als Meldung zurückgeschickt.

4.8 Angebot buchen

Dieser Prozess behandelt die Buchung eines Angebots, ausgeführt von einer Privatperson. Dabei muss es sich um eine verifizierte Person handeln. Der User gibt an, welche Menge er buchen möchte, wonach ihm die Buchung zugeteilt wird. Der Prozess ist im Flussdiagramm in Abbildung 7 dargestellt. Der Prozess wird am Frontend ausgelöst.

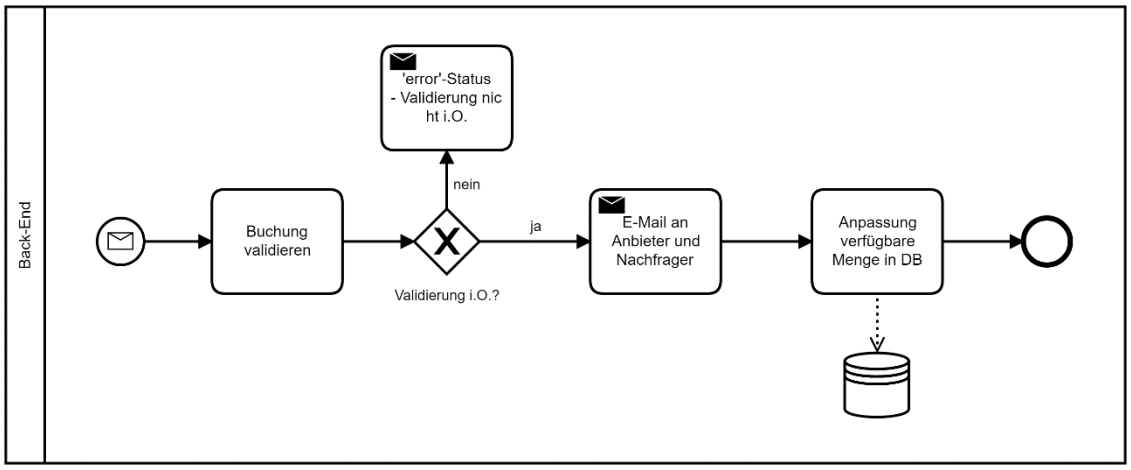


Abbildung 7: Flussdiagramm Angebot buchen

Das Backend erhält einen Request, um eine Buchung auf einem Produkt vorzunehmen. Zuerst wird *validiert*, ob der entsprechende User das Produkt buchen darf und ob die gewünschte Menge noch verfügbar ist. Falls diese Bedingungen erfüllt sind, kann das Produkt gebucht werden. Dann wird eine E-Mail an den Besitzer sowie an die buchende Person ausgelöst. In dieser E-Mail sind die Details zur Buchung einsehbar. Auf dem Produkt in der Datenbank wird die verfügbare Menge angepasst.

4.9 Grundsätzliches

Die meisten Prozesse sind in der Grundstruktur sehr ähnlich oder sogar analog. Grundsätzlich wird jeweils vom Frontend ein Request ausgelöst, welcher initial im Backend validiert wird und jeweils entweder Daten zurückgibt (klassischer GET-Request) oder eine Manipulation in der Datenbank auslöst (klassischer POST-Request). Bei den erwähnten Prozessen handelt es sich ausschliesslich um die zentralen Prozesse für das Backend, welche von einem User ausgelöst werden. Es stellt keine abschliessende Liste von Prozessen dar. Speziell werden in den nachfolgenden Kapiteln noch Prozesse und Anforderungen erwähnt, welche ausschliesslich technisch sind und nicht von einem User ausgelöst werden.

5 Anforderungen

Aufbauend auf die im Kapitel 3 beschriebene Ausgangslage und die definierten Prozesse im Kapitel 4 werden in diesem Kapitel die Anforderungen an den Backend-Server beschrieben. Die Fragestellung der vorliegenden Arbeit hat ebenfalls einen grossen Einfluss auf die Anforderungen an die Applikation. Dabei werden Anforderungen beschrieben, welche nicht vom User selbst ausgeführt, sondern automatisch vom Backend ausgelöst werden. Neben diesen funktionalen Anforderungen bestehen noch sogenannte nicht funktionale Anforderungen. Dabei handelt es sich um Anforderungen, welche keine Funktion ausführen, sondern zum Beispiel, dass eine gewisse Antwortzeit nicht überschritten wird (Böhm, 2002, S. 85).

Eine übergreifende Anforderung, welche sich aus der Fragestellung der Arbeit ableitet, ist die automatisierte Abwicklung. Bei jeder spezifischen Anforderung wird darauf geachtet, wie diese übergreifende Anforderung erfüllt werden und ob allenfalls zusätzliche Anforderungen aufkommen, sodass so wenig manuelle Eingriffe wie möglich nötig sind. Die Anforderungen werden gegliedert nach funktionalen und nicht funktionalen Anforderungen.

5.1 Funktionale Anforderungen

In diesem Kapitel sind die funktionalen Anforderungen an das Backend beschrieben. Daraus soll Programmcode resultieren, welcher eine Funktion ausführt (wie zum Beispiel eine Datenbankmanipulation in der Datenbank oder eine E-Mail versenden).

Der Backend-Server soll drei zentrale Anforderungen erfüllen: Datenhaltung, Datenvalidierung und Durchführen von Geschäftslogik. Mit *Datenhaltung* ist das persistente Speichern von Daten in einer Datenbank gemeint. Unter *Datenvalidierung* ist eine Prüfung der Daten, bevor diese persistiert werden gemeint. Mit *Durchführen von Geschäftslogik* sind zum Beispiel Workflows oder das Auslösen von E-Mails gemeint. Die Funktionalitäten des Backends sollen über eine Schnittstelle, vorzugsweise mittels REST-API angesteuert werden können. In der Tabelle 1 sind die funktionalen Anforderungen an das Backend ersichtlich. Dabei wird zwischen initialen Anforderungen und Zusatzanforderungen unterschieden. Zu den initialen Anforderungen werden ursprünglich Anforderungen gezählt. Diese waren in der ersten Phase des Projekts bekannt und wurden entsprechend an das Backend gestellt. Zu den Zusatzanforderungen gehören Anforderungen, welche

erst in der späteren Projektphase aufgetreten sind. Dabei wird nicht unterschieden, ob die Zusatzanforderung später aufgetreten ist und explizit gewünscht wurde oder ob diese bei der Implementation des Programmcodes aufgetreten ist und daher implementiert wurde. In Tabelle 1 sind die initialen Anforderungen mit dem Präfix «A» und die Zusatzanforderungen mit dem Präfix «Z» beschrieben.

Tabelle 1: Funktionale Anforderungen

Nr.	**Beschreibung**
A-0	Der Backend-Server soll mittels REST-Schnittstelle aufgerufen werden können.
A-1	Requests, welche etwas in der Datenbank speichern (POST-Requests), sollen, auch wenn diese neuen Einträge in mehreren Entities erstellen, mit einem Request durchgeführt werden können.
A-2	Antworten auf einen Request sollen mit einem Response-Code versehen werden, so dass im Frontend ein allfälliger Fehler evaluiert werden kann.
A-3	Die Prozesse gemäss Kapitel 4 sollen im Backend abgebildet und durchführbar sein.
A-4	Vom Backend wir geprüft, ob der User berechtigt ist den Request durchzuführen.
A-5	Der Admin-User soll zu den manuellen Prozessen, welche standardmässig durchgeführt werden müssen, mittels E-Mail aufgefordert werden.
A-5.1	Der Admin-User soll den Prozess gleich über die erhaltene E-Mail ansteuern können.
A-6	Die Daten werden vor der Speicherung in die Datenbank auf ihre Qualität überprüft.
A-7	Der Matching-Service läuft im Hintergrund und ohne manuelle Eingriffe

A-8	Es müssen Requests zur Verfügung gestellt werde, durch die Daten endgültig aus der Datenbank gelöscht werden können.
A-9	Ein E-Mail-Service wird angebunden, über den E-Mails verschickt werden können.
ZA-1	Es gib einen Garbage-Collector, welche nicht mehr benötigte Daten aufräumt
ZA-2	Die Security und das E-Mail/Passwort Management wird im Backend abgebildet und gespeichert

5.2 Nicht funktionale Anforderungen

In diesem Kapitel sind die nicht funktionalen Anforderungen festgehalten. Aus diesem Teil resultiert zum Beispiel die Wahl eines Frameworks oder Optimierungen für die Ausführungen des Programmcodes, jedoch keine eigentlichen Funktionalitäten. In der Tabelle 2 sind diese nicht funktionalen Anforderungen aufgelistet.

Tabelle 2: Nicht funktionale Anforderungen

Nr.	**Beschreibung**
NFA-0	GET-Requests die nur ein Java-Objekt zurückgeben und mit einer UUID durchgeführt werden sollen durchschnittlich in unter 200 Millisekunden eine Antwort liefern
NFA-0.1	GET-Requets, die mehrere (bis zu 50) Java-Objekte oder komplexere Konstrukte, wie zum Beispiel mehrere in sich verschachtelte Objekte liefern, sollen durchschnittlich in unter 500 Millisekunden eine Antwort liefern.
NFA-0.2	GET-Request, Searchresult muss so optimiert sein, dass dieser eine grosse Anzahl von Resultaten (bis zu 500) in durchschnittlich unter 1000 Millisekunden zurückgibt.

NFA-1	POST-Requests, welche ein normales Update oder Insert in die Datenbank durchführen, ohne weitere Validierungen über externe APIs, sollen innerhalb von 300 Millisekunden eine Antwort liefern.
NFA-2	Der Matching-Engine soll so optimiert sein, dass dieser alle fünf Minuten laufen kann und standardmässig (Annahme 15 neue Angebote, fünf gelöschte Angebote, 1000 Nachfrager) in durchschnittlich unter 30 Sekunden durchläuft.
NFA-2.1	Der Matching-Engine auf Personenebene (nach der Validierung der Person) soll bei 1000 Angeboten in durchschnittlich unter 30 Sekunden durchlaufen.

6 Design

In diesem Kapitel wird das grundsätzliche, prozessunabhängige Design des Backend beschrieben. Dabei wird einerseits auf die Wahl des Frameworks und andererseits auf die Struktur des Programmcodes eingegangen.

6.1 Wahl des Frameworks

Die Wahl des Frameworks stützt sich stark auf die in Kapitel 3 beschriebene Ausganslage. Da sich die Fragestellung der vorliegenden Arbeit nicht auf die Anpassung des Frameworks bezieht und sich das Framework im Projekt soweit bewährt hat, wurde dies als Basis genommen. Somit erfolgt die Weiterentwicklung auf einer Java-Spring Boot Applikation mit einer Anbindung an eine MongoDB. Die Infrastruktur ist vom Dozenten vorgegeben. Die Plattform-as-a-Service-Plattform namens Jelastic-Cloud wird als Server benutzt. Die Übernahme des Frameworks der letztjährigen Bachelorarbeit «Backend» hat den Vorteil, dass der Programmcode an sich übernommen werden kann. Jedoch besteht ein wesentlicher Teil der vorliegenden Arbeit daraus, den Programmcode aus allen drei Bachelorarbeiten (Backend, Validierung und Matching) zusammenzuführen. Dafür muss der Programmcode von allen drei Projekten hinterfragt werden und kann auch nicht immer wiederverwendet werden.

6.2 Struktur des Programmcodes

In der Abbildung 8 ist die grobe Ordnerstruktur des Programmcodes abgebildet. Bei der einzigen Java-Klasse, welche hier sichtbar ist (Appliaction.java), handelt es sich um die Main-Class, welche die Springboot-Applikation ausführt.

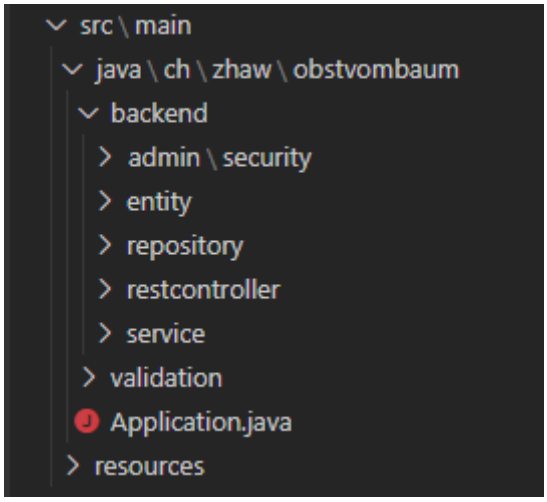


Abbildung 8: Grobe Ordnerstruktur Programmcode

Nachfolgend sind die einzelnen Ordner beschrieben und allfällige Unterstrukturen erwähnt.

6.2.1 Admin / Security

In diesem Ordner sind die Java-Klassen abgelegt, welche die Requests initial steuern und prüfen, ob eine Authentifizierung benötigt wird. Der Ordner hat noch diverse weitere Unterordner. Das zugrundeliegende Framework wird in Kapitel 6.6 beschrieben. Auf die technische Implementation wird im Kapitel 7.8 eingegangen.

6.2.2 Entity

In diesem Ordner sind die einzelnen Java-Klassen zu finden, welche mittels entsprechendem Repository in der Datenbank als Document gespeichert werden können. Weiter sind auch Java-Klassen gespeichert, welche als Java-Objekt für einen Request benötigt werden.

6.2.3 Repository

Dieser Ordner enthält die Repository-Klassen für die einzelnen in Kapitel 6.2.2 erwähnten Java-Objekte, welche in der Datenbank gespeichert werden können. Diese Repository-Klassen agieren als Schnittstelle zwischen den Java-Objekten und den Documents in der MongoDB.

6.2.4 Restcontroller

Dieser Ordner enthält die Java-Klassen, welche als Schnittstelle zwischen den eingehenden Requests (JSON-Notation) und den Java-Objekten agiert. In den einzelnen Klassen sind die entsprechenden Requests mit den jeweiligen Pfaden enthalten. Die Requests für den Registrierungsprozess, stellen eine Ausnahme dar, da sich diese im Ordner Security (Kapitel 6.2.1) befinden.

6.2.5 Service

Im Ordner Service befinden sich diverse Java-Klassen, welche entweder Logik im Hintergrund ausführen (zum Beispiel Job-Services) aber auch Helper-Klassen, welche durch mehrere anderen Klassen gebraucht werden, sodass dieselben Funktionen und Prozeduren nicht redundant entwickelt, respektive gewartet werden müssen.

6.2.6 Validation

In diesem Ordner befinden sich die Java-Klassen, in denen die Logik für die Validierungen abgebildet ist. Gewisse einfache Validierungslogiken sind direkt in den jeweiligen Restcontroller-Klassen (s. Kapitel 6.2.4) abgebildet. Weiter sind auch APIs zu finden, welche für Validierungen benötigt werden.

6.2.7 Ressourcen

Im Ordner Ressourcen befinden sich zwei sehr zentrale Files. Einerseits das `config.properties` File, in welchem diverse Globale-Variablen definiert sind. Weiter ist das File `appliaction.propierties` zu finden, in welchem Einstellungen zur MongoDB und zum Mail-Server enthalten sind.

6.3 Datenbankstruktur

Die Datenbankstruktur richtet sich stark an die im Kapitel 3.2 beschriebene Ausgangslage. Dabei wurden jedoch noch signifikante Änderungen gemacht, da die ursprüngliche Datenbankstruktur nicht in allen Hinsichten den aktuellen Anforderungen entsprach. Um ein Verständnis zu erhalten, ist in Abbildung 9 die Ausgangslage der Datenstruktur aufgezeigt.

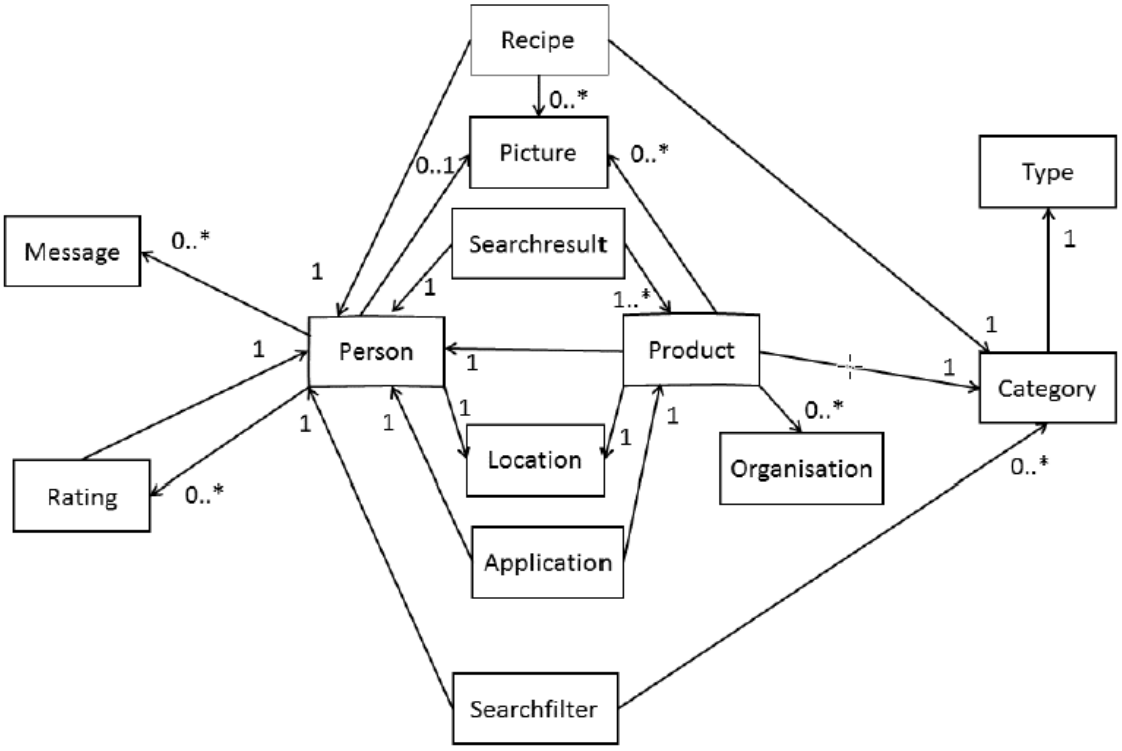


Abbildung 9: Ausgangslage Datenbankstruktur (Graf, 2019, S. 27)

Da die neue Datenstruktur stark an die Ausgangslage angelehnt ist, sind nachfolgend ausschliesslich Anpassungen erwähnt. In der Abbildung 10 ist die angepasste Datenstruktur abgebildet. Da es sich um eine NoSQL-MongoDB handelt repräsentieren die Entities keine Tabellen, sondern sogenannte Collections (MogoDB, 2008a). Somit wird nachfolgend für eine Entität jeweils die Formulierung Collections verwendet.

Um den Prozess zu vereinfachen gibt es keine Bewerbung mehr, sondern ein Angebot wird direkt gebucht. Folglich fällt die Collection Application weg. Weiter wird in einer ersten Version auf das Rating und den Nachrichtenaustausch über die Plattform verzichtet. Somit fallen auch die Collections Rating und Message weg. Ebenfalls wird in einer ersten Version auf das Speichern von Rezepten verzichtet, wodurch die Collections Reciept wegfällt. Da die Erstellung und Verwaltung der eigenen Produkte überarbeitet wurde, war es zwingend, die ursprüngliche Collections Product aufzuteilen. Neu kann ein Produkt mehrere Offers haben. Der gesamte Matching-Prozess und somit die Suchresultate (Collection Searchresult) wie auch die Angabe des Ortes (Collection Location) haben neu eine Relation zur Collection Offer.

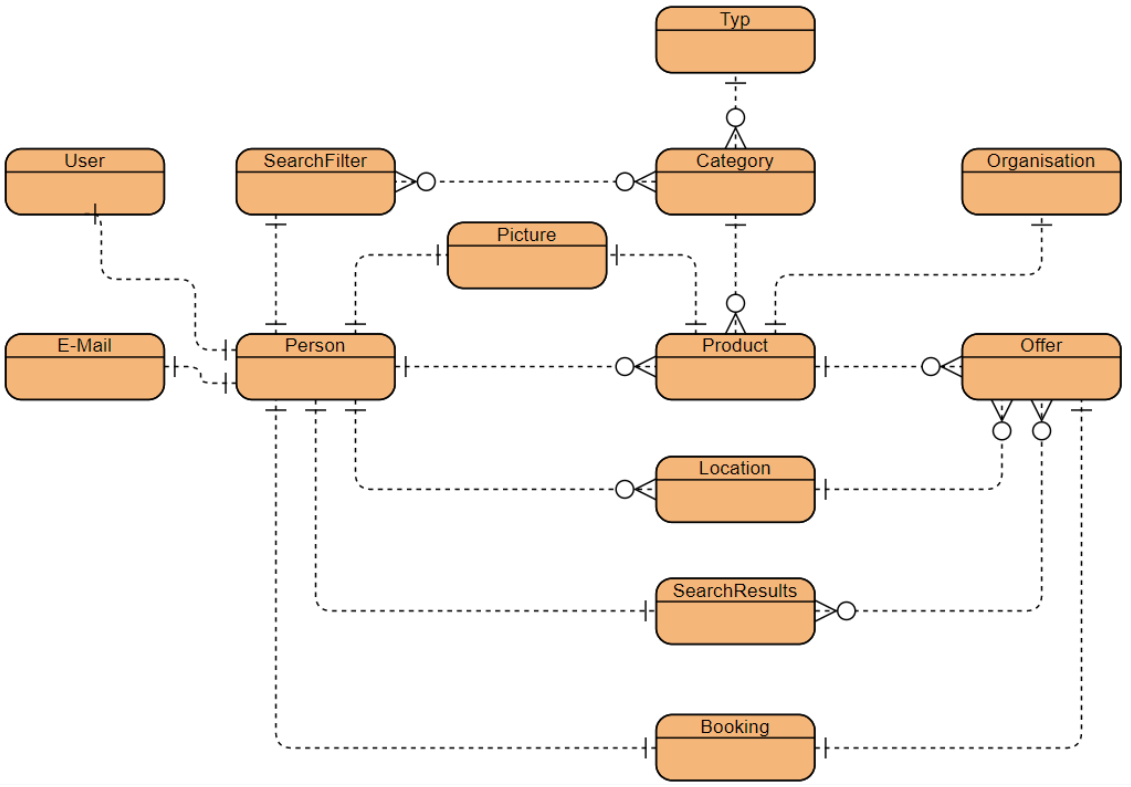


Abbildung 10: Datenbankstruktur

Die Abstraktion zur Realität wurde in der Datenbankstruktur auf ein Minimum reduziert. Nichtsdestotrotz sind nachfolgend die einzelnen Collections genauer erläutert. Es wird beschrieben, welche Daten genau in der Collection gespeichert und wie die jeweiligen Relationen aufgebaut sind. Ein einzelner Eintrag in einer Collection wird in der MongoDB Document genannt (MogoDB, 2008a). Nachfolgend wird daher der Begriff Document verwendet.

6.3.1 Person

In der Collection Person bildet jedes Document eine eigene Person ab. Dabei spielt es keine Rolle, ob es sich um eine private Person oder um einen Landwirtschaftsbetrieb handelt. Es werden beide in derselben Collection Person gespeichert. Weiter werden auf dem Personen-Document personenspezifische Daten wie Anrede, Name, Telefonnummer, Registrierungsdatum, eine zweite E-Mail-Adresse, der Personentyp sowie Status gespeichert. Ebenfalls werden zwei Boolean-Attribute gespeichert, welche Informationen für die Anzeige am Frontend enthalten. Das erste Attribut enthält die Information, ob die entsprechende Person ein Nachfrager ist, also bereits etwas gebucht hat. Das zweite Attribut hält fest, ob die Person ein Anbieter ist, also bereits ein Angebot auf die Plattform

gestellt hat. Weiter werden Referenzen zu E-Mail, Domizil-Location, Offer-Location und Picture (Profilbild) gemacht.

6.3.2 E-Mail

In der Collection E-Mail repräsentiert jedes Document eine E-Mail-Adresse. Die Referenz zur jeweiligen Person ist auf dem Personen-Document zu finden. Auf dem E-Mail Document sind die Attribute E-Mail und Status.

6.3.3 User

In dieser Collection wird die E-Mail / UserId / Passwort-Kombination gespeichert. Weiter wird gespeichert, welche Rolle der entsprechende User hat. Diese Documents haben keine reale Relation zu den Documents in der Collection Person. Es wird nur die jeweilige Personen-Uuid als Username gespeichert.

6.3.4 Picture

In dieser Collection werden Bilder gespeichert. Dabei spielt es keine Rolle, ob es sich um ein Bild von einem Produkt oder einer Person handelt. Der entsprechende Typ wird ebenfalls im jeweiligen Docuement gespeichert. Die Referenz auf ein Bild wird jeweils in der Collection Person respektive Product gemacht.

6.3.5 Product

In der Product-Collection werden Produkte gespeichert, welche der jeweilige User anbietet (zum Beispiel Äpfel). Dabei wird eine Referenz zu einer Organisation gemacht, an die der User den Ertrag spenden möchte, oder man kann einen Preis erfassen. Ebenfalls besteht eine Relation zu einer Kategorie (Collection Category) sowie zum Besitzer des Produkts (Collection Person). Weiter wird Relation zu den einzelnen Angeboten (Collection Offers) in einer ArrayList gespeichert.

6.3.6 Category

In dieser Collection werden die verschiedenen Typen von Obst gespeichert (zum Beispiel: Äpfel, Birnen, Erdbeeren). Diese Collection soll nur von einem Admin-User administriert werden können. Darauf sind Attribute, wie der aktuell durchschnittliche Marktpreis, sowie die CO2-Emmisson für die Ernte von einem Kilogramm des jeweiligen Obsts, gespeichert. Auf diese Informationen wird wiederum in gewissen Prozessen zugegriffen.

6.3.7 Typ

In der Collection Typ wird ein Überbegriff für eine Kategorie gespeichert. Im vorliegenden Beispiel gib es nur ein Typ (Obst). Diese Collection ist für den Ausbau der Applikation angedacht, wenn zum Beispiel zusätzlich Gemüse angeboten werden soll. So besteht bereits eine Collection, in welcher diese Logik abgebildet ist.

6.3.8 Searchfilter

In dieser Collection kann ein User ein Suchabo speichern. Darin kann er eine oder mehrere Kategorien (Category) angeben. Es ist ebenfalls möglich keine Kategorie anzugeben. Dann gilt das Suchabo für alle Kategorien. Wenn ein oder mehrere neue Angebote von dieser Kategorie vorhanden sind, wird eine E-Mail an den User ausgelöst.

6.3.9 Offer

In dieser Collection werden die Angebote eines Prodcuts gespeichert. So kann ein Anbieter sein Produkt an mehreren Stellen anbieten (zum Beispiel auf dem Markt bereits gepflückt und auf dem Bauernhof zum selbst pflücken). In einem Offer-Document werden verfügbare Menge, Zeitspanne, und eine Beschreibung gespeichert. Weiter wird eine Relation zu einer Location gemacht. Im Hintergrund wird die ID des Products gespeichert, sodass dieses performant aufgerufen werden kann.

6.3.10 Location

In dieser Collection werden Orte gespeichert. Ein Ort kann entweder eine Domiziladresse einer Person sein und/oder eine Adresse eines Offers. Auf einer Location werden Strasse, Postleitzahl und Stadt- respektive Dorfname angegeben. Weiter werden im Hintergrund die Koordinaten auf der Location gespeichert, welche für die Distanzberechnung benötigt werden. Ebenfalls im Hintergrund wird der Kanton gesetzt, welcher für den allfälligen Versand eines E-Mail an einen kantonalen Bauernverband nötig ist.

6.3.11 Serachresult

In der Collection Searchresult werden erfolgreiche Matching-Resultate aus dem Matching-Run gespeichert. In dieser Collection wird eine Referenz zur Person gemacht, für die das Suchresultat ist. Weiter werden diverse Informationen vom Offer und Product auf das Searchresult propagiert (Begründung: siehe Kapitel 6.4.2). Zusätzlich wird die Distanz zwischen Offer-Location und Domizil-Location des Nachfragers sowie die Mindesternte pro Matching-Resultat gespeichert. Um diese Informationen sauber auf dem

Searchresult abzulegen, wurde eine eigenes Java-Objekt umgesetzt, welche einen einzelnen Match repräsentiert. Die Collection Searchresult enthält eine ArrayList dieser Java-Klasse.

6.3.12 Booking

In dieser Collection wird ein gebuchtes Angebot abgebildet. Es hat eine Referenz zum entsprechenden Offer sowie zum User, welcher das Angebot gebucht hat. Weiter wird die gebuchte Menge gespeichert.

6.4 Performance

Um den Performance-Anforderungen gemäss Kapitel 5.2 gerecht zu werden, wird die Performance im Software-Design berücksichtigt. Grundsätzlich wurden zwei kritische Punkte analysiert. Erstens, der *Matching-Service*, welcher über alle verifizieren Angebote und Orte loopen muss, um zu prüfen, ob die entsprechenden Angebote einem Nachfrager ins Searchresult geschrieben werden sollen. Zweitens, stellt der GET-Request auf Searchresult ein weiterer kritischer Punkt dar, da hier mehrere 100 Angebote als Antwort kommen können und dies performant sein muss, da der User direkt auf eine Antwort wartet. Nachfolgend ist das Kapitel in diese beiden Teile aufgeteilt.

6.4.1 Matching-Service

Dieses Unterkapitel behandelt das Performance-Problem, welches im Matching-Service aufkommen kann, sobald die Userzahl grösser wird. Die Logik des Matching-Service selbst sowie die Implementation, ist im Kapitel 7.3 genauer beschrieben.

Das Performance-Problem im Matching-Service besteht aus zwei Teilproblemen. Das eigentliche Problem lässt sich wie folgt formulieren: *Jede Nachfrage/Angebots-Paarung, welche im System besteht, muss auf ein potenzielles Matching überprüft werden*. Somit gilt folgende Formel

$$Durchzuführende\ Matchingprüfungen = Antebote * Locations$$

Damit aus dieser Formel eine aussagekräftige Zahl entsteht, wird die durchschnittliche Dauer eines Matching-Runs für eine einzelne Nachfrage/Angebots-Paarung gemessen. Dafür wurde am Ende des Programmcodes vom Matching-Run folgendes Debug-Statement eingebaut:

```
System.out.println("Matching done. Duration with "
```

```
+ numOffers
+ " Offers ms: "
+ ((end.getTime()-start.getTime()))
);
```

Die Variable `numOffers` wurde dabei mit der Anzahl Offers befüllt, welche im jeweiligen Matching-Run neu zugeteilt wurden. Weiter wurde am Anfang vom Matching-Run die Zeit in der Variable `start` gespeichert und am Schluss in der Variabel `end`. So kann die Differenz berechnet werden. Auf der Testumgebung wurde ein Nachfrager und 1000 Angebote erstellt. Der Matching-Run wurde dann drei Mal ausgelöst. In der Abbildung 11 wird deutlich, dass der Matching-Run beim ersten Mal 18931 Millisekunden, beim zweiten Mal 18563 Millisekunden und beim dritten Mal 18041 Millisekunden gedauert hat.

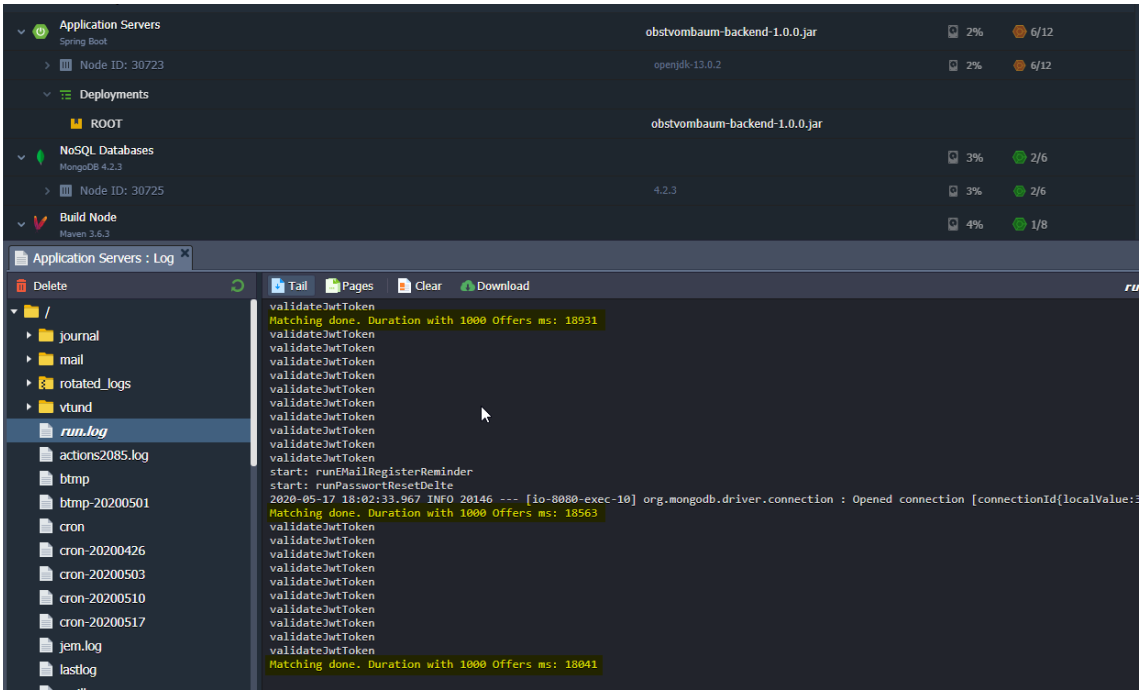


Abbildung 11: Messung Matching-Run

Der Durchschnittswert liegt gerundet bei 18512 Millisekunden. Dadurch lässt sich eine gerundete Durchschnittszeit von 18.52 Millisekunden pro Nachfrage/Angebots-Paarung errechnen. Ausgehend von dieser Durchschnittszeit, kann folgende Formel für die Durchlaufzeit eines kompletten Matching-Runs aufstellt werden:

$$Durchlaufszeit\ in\ Millisekunden = Antebote * Locations * 18.52$$

Wenn mit 1000 privaten Usern und 3000 verifizierten Angeboten auf der Plattform gerechnet wird, so ergibt dies eine durchschnittliche Laufzeit von 55’560’000 Millisekunden (≈ 15.43 Stunden) pro Matching-Run. Da die Suchresultate jedoch jeweils möglichst aktuell sein sollen und die Daten aufgrund der nachfolgenden Erläuterungen im Kapitel 6.4.2 nicht direkt über die jeweiligen Repositories geholten werden können, muss der Matching-Run alle fünf Minuten laufen (s. Anforderung NFA 2.1, Kapitel 5.2). Um die Durchlaufszeit zu vertiefen muss also etwas an den Variablen (Angebote / Locations / Durchlaufszeit pro Paarung) heruntergeschraubt werden. Eine Möglichkeit besteht darin, dass jeweils nur die neusten Angebote berücksichtigt werden und alle bestehenden Matchings so belassen werden wie diese sind. Daraus ergibt sich aber das nächste Teilproblem, dass *ein bestehender Offer angepasst oder gelöscht werden kann*. Somit muss auch bei einem Update oder einer Löschung eines Offers die Information möglichst schnell in das Suchresultat kommen. Damit dies gewährleistet ist wurde der in Abbildung 12 illustrierte Ansatz verfolgt.

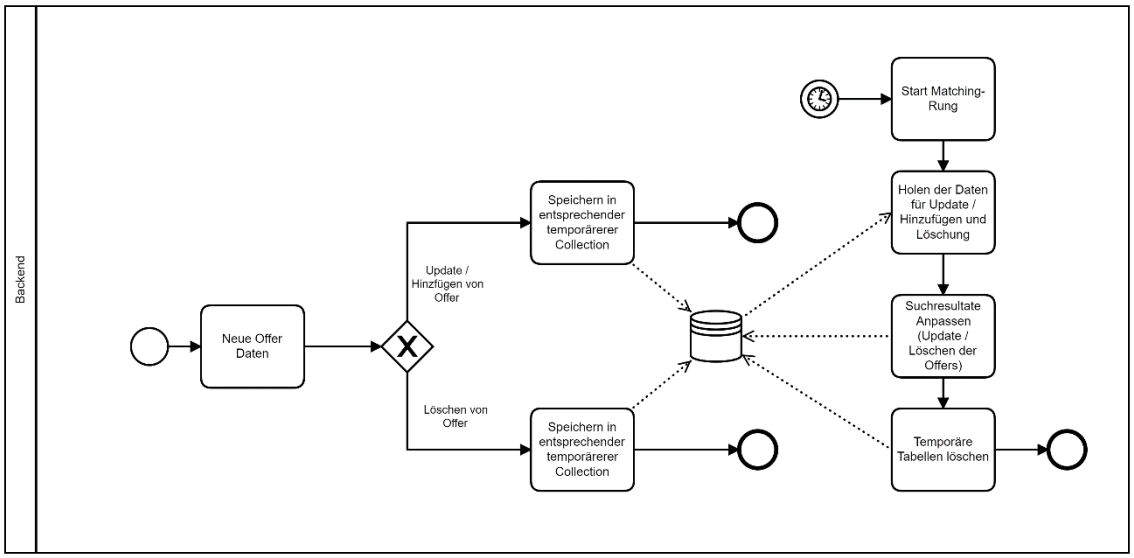


Abbildung 12: Daten von Offer in Searchresult

Bei jeder Erstellung, jedem Update und jeder Löschung eines Angebots wird ein Document in einer temporären Collection erstellt. Sobald der Matching-Run läuft, wird diese temporäre Collection als Inputparameter verwendet. Somit werden im Matching-Run nur Offers berücksichtig, welche auch eine Relevanz haben. Eine Ausnahme gilt für neu erstellte Offers. Diese bleiben 48 Stunden in der temporären Collection. Dies hat einerseits den Grund, dass davon ausgegangen wird, dass ein neues Angebot dynamischer ist (es wird mehr gebucht und somit pass sich die verfügbare Menge oft an). Andererseits

hat dies mit der Logik des Matching-Service zu tun, da in den ersten 48 Stunden unterschiedlich gematcht wird (siehe Kapitel 7.3).

6.4.2 Suchresultat (GET-Request auf Searchresult)

Ein Suchresultat bildet eine Sammlung von Angeboten ab. In einem Suchresultat sind somit alle Angebote, welche von einem User gebucht werden können, enthalten. In einem produktiven Betrieb ist es realistisch, dass mehrere hundert Angebote in einem Suchresultat enthalten sind. Da der Suchrequest am GUI ausgeführt wird, soll der End-User schnell eine Response erhalten. Daher muss sichergestellt sein, dass auch im Falle von mehreren hundert Angeboten der GET-Request auf ein einzelnes Suchresultat performant durchgeführt werden kann. Im Kapitel 5.2 ist die nicht Funktionale Anforderung (NFA-0.2) genauer beschrieben.

Damit ein GET-Request performant ist, muss sichergestellt sein, dass die Informationen nicht über mehrere Collections und somit Repositories gesammelt werden. Alternativ müssen alle Informationen in einer Collection respektive in einem Document enthalten sein. Die Problematik beim Searchresult ist, dass neben den Daten der Searchresult -Collection auch noch Daten der Product-Collection und der Offer-Collection benötigt werden. Nachfolgend ein Verglich von zwei GET-Requests, welche an sich dieselben Informationen zurückliefen. Beim ersten Request sind jedoch alle Informationen auf einem Document gespeichert, beim zweiten Request ist nur eine Uuid vom Offer auf dem ersten Document gespeichert und damit wird innerhalb des Requests mittels `offerRepository` und `productRepository` die Information vom Offer und dem Product geholt. Im Programmcode wurde also zusätzlich folgender Stelle eingebaut, um ein Aufruf der beiden Repsitories zu simulieren, in welchen die benötigten Daten eigentlich gespeichert sind:

```
// this part is normal code
ArrayList<OfferMinHarvestDistanceMatrix> offerMatrix =
  searchResult.getOfferMinHarvestDistanceMatrixs()
;
offerMatrix.sort((o1, o2) ->
  o1.getDistance().compareTo(o2.getDistance()))
;

// only for test
for (OfferMinHarvestDistanceMatrix matrix: offerMatrix){
```

```
   Optional<Offer> offerOptional =
      offerRepository.findById(matrix.getOfferUuid())
   ;
   if (offerOptional.isPresent()){
      // get some data from offer
      Optional<ProductPrivate> product =
         productRepository.findById(
            offerOptional.get().getProductUuid()
         )
      ;
      if (product.isPresent()){
         // get some date from product
         String productName = product.get().getDescription();
      }
   }
}
```

In der Abbildung 13 wurde der Request nur auf einem Document der Collection Searchresult gemacht. In der Abbildung 14 war der obengenannte Programmcodeteil eingebaut, also wurden jeweils die Product und Offer-Repositories aufgerufen.

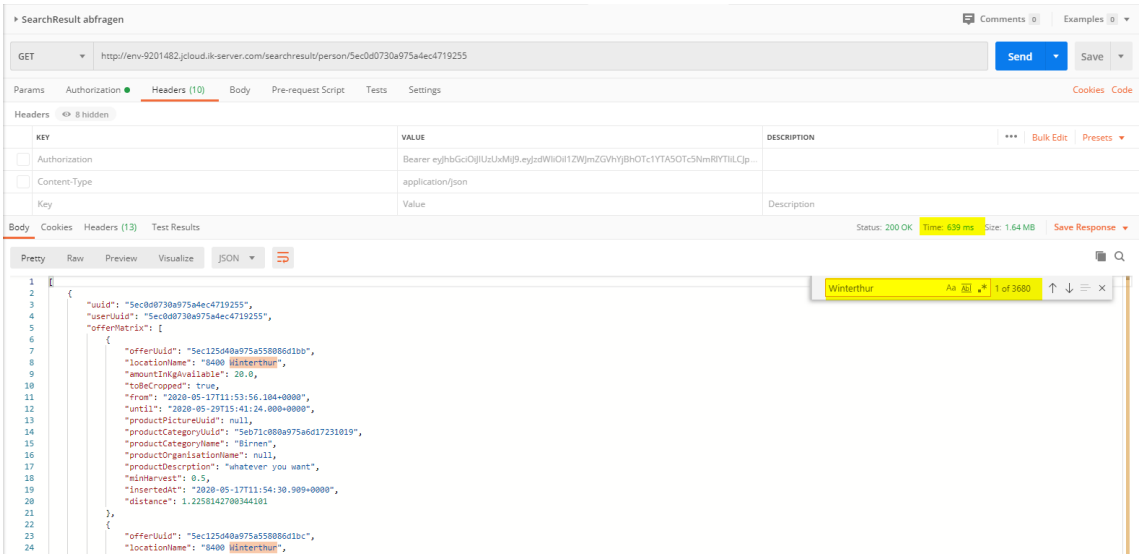


Abbildung 13: Request ohne Aufruf von zusätzlichem Repository

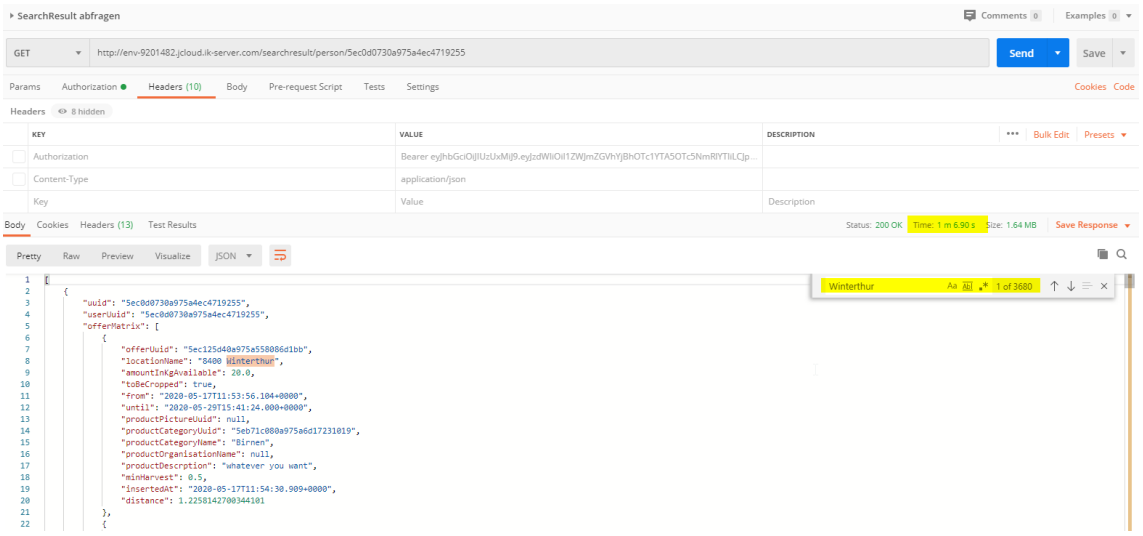


Abbildung 14: Request mit Aufruf von zwei zusätzlichen Repositories

Dieses Beispiel zeigt klar auf, dass ein Aufruf von weiteren Repositories in solchen Fällen aus Performance-Sicht nicht akzeptabel ist. Somit müssen alle Daten, welche in einem Searchresult-Request angezeigt werden, auch in den Documents der Collection Searchresult gespeichert werden.

6.5 Job Schedule

Es gibt mehrere Prozeduren im System, welche immer wieder und ohne User-Interaktion laufen müssen. Der zentralste ist der Matching-Run. JavaSpringBoot bietet mit der @Scheduled Annotation eine entsprechende Funktionalität (Paraschiv, 2020). Diese wird auch für die Implementation in diesem Projekt gewählt.

6.6 Security

Die Security war zu Beginn keine Anforderung an das Backend, da diese gemäss ursprünglicher Anforderung (siehe Kapitel 3.1) über einen Login-Server hätte laufen sollen. Im Projektverlauf wurde jedoch aus verschiedenen Gründen festgestellt, dass dies zu anderen Problemen führen würde. Einerseits in das Backend Master über die Personendaten. Daher werden auch die Uuids vom Backend vergeben. Weiter sind Prozesse wie *Passwort zurücksetzen* und *Registrierung* nur unschön umsetzbar, da diese Requests immer vom Login-Server angepasst oder im Login-Server ein zusätzlicher Mail-Server angebunden werden müsste. Somit wurde im Projektverlauf entschieden, dass die Security im Backend als Zusatzanforderung umgesetzt wird.

Für die Umsetzung wurde ein entsprechendes Framework gesucht, welches gleichzeitig für die bestehenden Umsetzungen im Frontend passt. Als Grundlage wird die Spring Security genommen, welche mit verhältnismässig kleinem Aufwand in eine bestehende Spring Boot Application eingebunden werden kann. Grundsätzlich wird zwischen drei zentralen Funktionalitäten unterschieden: *Registrierung* (Erstellung einer E-Mail / Passwort Kombination), *Login* (Validierung der E-Mail / Passwort Kombination) und dem *Zugriff auf geschützte Requests*.

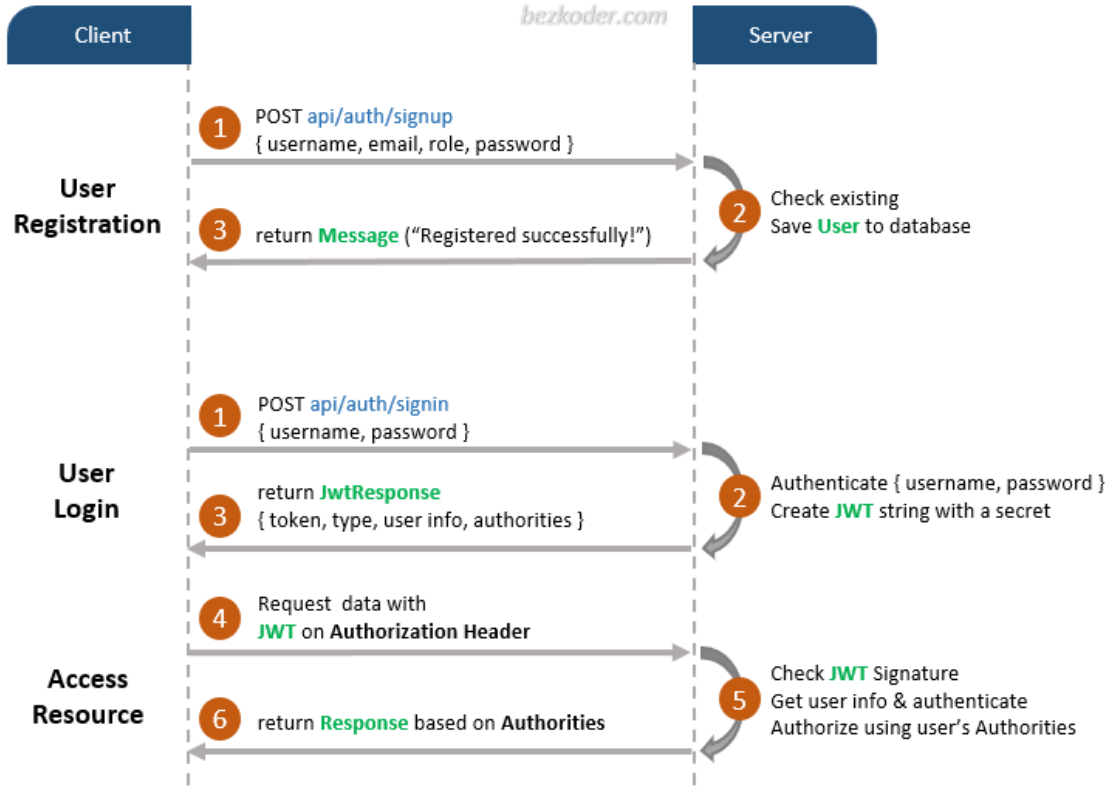


Abbildung 15: Ablauf Authentifizierung (Bezkoder, 2020)

In der Abbildung 15 ist der Ablauf einer Authentifizierung inklusive der Erstellung eines neuen Users aus Security-Sicht abgebildet. Am Anfang steht die Erstellung eines neuen Users und somit einer E-Mail / Passwort Kombination. Dieser Prozess wird implizit bei einer Registrierung einmalig pro User durchgeführt. Um auf eine geschützte Ressource zuzugreifen (Access Resource: Schritt 4 - 6) wird eine sogenannte JWT-Signature benötigt. Diese wird bei einem Login (User Login: Schritt 1 - 3) generiert, mittels Response ans Frontend geliefert und dort im Cache gespeichert werden. Für jeden Request auf eine geschützte Ressource sollte dann vom Frontend die JWT-Signature mitgegeben werden.

7 Implementation

In diesem Kapitel wird die genaue Implementation des Programmcodes beschrieben. Die Implementation leitet sich stark von den Anforderungen in Kapitel 5 und den Software-Design Evaluationen im Kapitel 6 ab.

7.1 Entity und Documents

Die einzelnen Entities repektive Collections werden in Kapitel 6.3 bereits genauer erläutert. Somit wird in diesem Kapitel nur noch auf die Implementationen eingegangen.

7.1.1 Spring Boot Annotationen

In den verschiedenen Entities wird SpringBoot Annotationen verwendet, welche in diesem Kapitel genauer beschrieben ist.

`@Document` wird jeweils oberhalb der `public class` annotiert (Docs Spring, o.J.). So wird definiert, dass es sich bei der jeweiligen Klasse um ein Document handelt, welches in der MongoDB gespeichert wird (Docs Spring, o.J.). Mit dieser Annotation kann definiert werden, in welcher Collection ein solches Java-Objekt gespeichert werden soll (Docs Spring, o.J.).

`@Id` wird jeweils oberhalb des jeweiligen Objekt-Attributs annotier, welches als Identifier-Attribut in der Collection agiert (Docs Spring, o.J.). Mit dieser Id kann das Objekt dann auch über die entsprechenden Repositories gefunden werden (Docs Spring, o.J.). In diesem Projekt wurde die ID (oder auch Uuid) immer als String implementiert und die Ids werden in den meisten Fällen automatisch von der MongoDB vergeben (Docs Spring, o.J.).

Mit `@DBRef` kann eine Referenz zu einem Document in einer anderen Collection (respektive einem anderen Java-Objekt) gemacht werden (Docs Spring, o.J.). Die Referenz wird immer mittels der `@Id` des referenzierten Documents gemacht (Docs Spring, o.J.). Das Document, welches referenziert ist muss bereits in der MongoDB bestehen und nicht nur als Java-Objekt, sonst gibt es einen Runtime-Error (Docs Spring, o.J.).

`@NotNull` wird über den Attributen verwendet, welche nicht null sein dürfen.

`@Indexed(unique = true)` wird für Attribute verwendet, welche in der entsprechenden Collection einen unique-constraint haben müssen (MongDB, 2008c).

7.1.2 Searchresult

Im Searchresult müssen diverse Informationen propagiert gespeichert werden. Dazu wurde ein eigenes Java-Objekt erstellt, in welchem alle diese Informationen vorhanden sind. Rein aus Datenbank-Sicht agiert dieses Objekt als Relation zwischen Searchresult und Offer. Auf dem `SearchResult.java` Objekt ist eine entsprechende ArrayList.

```
private ArrayList<OfferMinHarvestDistanceMatrix>
offerMinHarvestDistanceMatrixs;
```

Dies ermöglicht einen performanten Zugriff auf die Daten pro Angebot.

7.2 Repository

Die Repositories wurden von der letztjährigen Bachelorarbeiten übernommen. Daher wird nicht mehr genauer darauf eingegangen, sondern nur der Grundsatz aufgezeigt. Die jeweiligen Entitäten haben je drei Repository-Klassen (Graf, 2019, S. 21). In der Standard-Klasse wird das MongoRepository geerbt (extended) (Graf, 2019, S. 21). Somit kann auf alle Methoden in der MongoRepository zugegriffen werden (Graf, 2019, S. 21). In den selbst geschriebenen Repositories ist jeweils pro Collection ein Interface implementiert, über welches die Möglichkeit besteht, eine Liste von Objekten zu erhalten, welche gewisse Filterkriterien erfüllen (getObjectByFilter). Somit kann als Inputparameter zum Beispiel ein Personenstatus mitgegeben werden und das Repository gibt alle entsprechenden Objekte in diesem Status zurück (Graf, 2019, S. 21).

7.3 Matching Service

Unter dem Matching-Service wird der Algorithmus verstanden, welcher einen Offer (Angebot) in ein Searchresult schreibt. Das Searchresult ist jeweils immer einer privaten Person zugeordnet. Vereinfacht formuliert, evaluiert der Matching-Service, welche Angebote einer Privatperson zugeteilt werden. Damit ein Angebot einer Privatperson zugeteilt wird, gibt es mehrere Faktoren. Einerseits soll die Ernte *ökologisch* sein, andererseits sollen zusätzlich *kurze Distanzen* bevorzugt werden.

Für die Berechnung des Faktors Ökologie stellt Spirig (2019, S. 83) folgende Formel für die Berechnung der Mindesternte auf:

$$\frac{CO2 - Emissionen\ Transportmittel\ in\ g\ /\ km\ *\ Distanz}{CO2 - Emissionen\ Fruchtart\ in\ g\ /\ kg}$$

Weiter wird zwischen den Transportmitteln Auto, öffentlicher Verkehr und Fahrrad unterschieden, wobei nur dem Auto ein Emissionswert pro Kilometer zugeteilt ist (Spirig, 2019). Daraus lässt sich schliessen, dass mit den Transportmitteln öffentlicher Verkehr und Fahrrad keine Mindesternte besteht. Die Zahlen für die CO2-Emissison pro Fruchtart sind in der Tabelle 3 abgebildet (Spirig, 2019, S. 84).

Tabelle 3: CO2-Emission nach Fruchtart (Spirig, 2019, S. 84)

Fruchtart	**CO2-Emissionen / 100 Gramm Frucht**	**CO2-Emissionen / Kilogramm Frucht**
Apfel	46g	460g
Birne	63g	630g
Kirsche	48g	480g
Zwetschge	54g	540g
Aprikose	32g	320g

Die Daten aus Tabelle 3 müssen in der Datenbank in der jeweiligen Category auf dem Attribut `private Double co2Use` gespeichert werden. Falls auf diesem Attribut keine Daten vorhanden sind, ist ein Fallback von 400 Gramm pro Kilo im config.properties File hinterlegt.

Da der Matching-Service performant sein muss, kann keine API aufgerufen werden, welche die Distanz für eine Strecke zurückgibt. Somit wird folgender Ansatz gewählt: Für die Berechnung der Distanz wird zuerst die Luftlinie der beiden Orte anhand der Koordinaten direkt im System berechnet. Die Grundlage für die Berechnung basiert auf dem Beitrag von Bajaj (o. J.). Die Luftdistanz wird dann mit 1.4 multipliziert. Die Variabel 1.4 ist eine theoretische Nummer und nicht wissenschaftlich belegt, da keine Quelle für einen realistischen Multiplikator gefunden werden konnte. Die Zahl basiert auf Foreneinträge und eigenen Stichproben auf GoogleMaps (OpenStreetMap Forum, 2009). Die Implementation dieser Methode, wurde mit dem Dozenten abgesprochen. Grund dafür war, dass das API, welches in der vorgängigen Bachelorarbeit umgesetzt wurde, produktiv nicht realistisch hätte eingesetzt werden können. Es wurde ein API zu GoogleMaps umgesetzt, welches die Routendistanz zwischen zwei Adressen berechnet. Dieses API hätte

jedoch für jede Angebots-Location-Paarung aufgerufen werden müssen. Da dieses API kostenpflichtig ist, wurde entschieden die erstgenannte pragmatische Lösung zu implementieren.

Für den zweiten Faktor *kurze Distanz* wurde folgende Logik implementiert:

1. Es gibt kein Matching, wenn die Luftlinie zwischen den beiden Orten (Wohnort Nachfrager / Ort des Angebots) mehr als 25 Kilometer beträgt.
2. In den ersten 24 Stunden nachdem ein Angebot verifiziert wurde, wird dieses nur innerhalb von zehn Kilometern Luftlinie gematcht.
3. In den ersten 48 Stunden nachdem ein Angebot verifiziert wurde, wird dieses nur innerhalb von 17.5 Kilometer Luftlinie gematcht.

Somit wird aus dem Matching-Service heraus die Parameter *Distanz* und *Mindesternte* berechnet. Diese Parameter werden bei einem erfolgreichen Matching in einem Document in der Collection Searchresult gespeichert.

7.4 Datenservice

Unter Datenservice werden Java-Klassen verstanden, welche Funktionen und Prozeduren enthalten, die bei Datenmanipulationen Daten ergänzen oder Standard-Methoden anbieten um Datenmanipulationen durchzuführen. Weiter sind auch sonstige Standardfunktionen enthalten, welche immer wieder benötigt werden.

Einer dieser Service ist eine API, welche bei Erstellung einer Location die entsprechenden *Koordinaten* und den *Kanton* aufruft und direkt auf der Location speichert (Driss, 2015). Für den Aufruf wurde das API von Mapquest.com genutzt. Beim GET-Request muss ein Authentication-Key sowie die Adresse (Strasse, Strassennummer, PLZ, Stadt/Dorf) als Path-Variable im Link mitgegeben werden (Mapquest, o. J.). Als Response kommt ein JSON-Objekt zurück, in welchem der Kanton und die Koordinaten als Attribut enthalten sind (Mapqeust, o. J.). Monatlich sind 15'000 Requests gratis (Mapquest, o. J.). Da der Request nur bei der Erstellung einer neuen Location gemacht wird, sollte dies in einer ersten Version der Applikation ausreichen.

Bei der Manipulation eines Offers muss diese Information auch immer auf das Searchreuslt propagiert werden. Dafür gibt es zwei weitere Collections offerDeleteSearchResult sowie offerUpdateSearchresult. Diese beiden Collections agieren als temporäre Collections, welche nach dem Lauf des Matching-Service wieder zurückgesetzt werden.

Da die Logik für die Befüllung und Löschung der temporären Collections an verschiedenen Stellen gebraucht wird, ist diese Logik zentral im Datenservice implementiert.

Wenn Documents aus einer Collection gelöscht werden, ist es gut möglich, dass diese noch eine Abhängigkeit zu anderen Documents in der Datenbank haben (siehe Kapitel 6.3). Mit dem *Consistency-Service* können Inkonsistenzen in der Datenbank verhindert werden. Beim Löschen eines Documents wird sichergestellt, dass auch abhängige Documents gelöscht werden oder dass eine Validierung erscheint, dass das entsprechende Document nicht gelöscht werden kann. Das Framework des Consistency-Services wurde aus der letztjährigen Bachelorarbeit «Backend» übernommen und erweitert.

Der E-Mail Service ist ein weiterer zentraler Service. Über diesen wird der E-Mail Server angesprochen und es können verschiedene E-Mails versendet werden. Die Java-Klasse HTMLGenerator.java generiert den Inhalt des E-Mails in HTML-Notation und gibt dies als String zurück. Die E-Mail selbst wird jeweils in der EMailService.java Klasse erstellt und versendet.

7.5 Prozesse

Um die Prozesse gemäss Kapitel 4 im Backend abzubilden, sind diverse Funktionalitäten implementiert. Eine zentrale Rolle spielt dabei das *Statuskonzept* der jeweiligen Collections, die Generierung von *Tokens*, und der *Versand von Validierungs-E-Mails*.

7.5.1 Status

Damit pro Document klar ist, in welchem Status sich dies befindet gibt es auf den meisten Collections ein Attribut Status (state). Für die Registrierung sind die Collections Person, E-Mail und Location zentral. Das Statuskonzept ist sehr einfach gehalten. Die Person ist Master, was heisst, wenn die Person verifiziert ist, müssen die Location und die E-Mail auch verifiziert sein. Maximal müssen drei Verifikationen durchgeführt werden (Ort, E-Mail, Bauernverband). Daher kann eine Person folgende drei Status haben *toBeVerified1*, *toBeVerified2* und *toBeVerified3*. Die Nummer steht dabei für die Anzahl Verifikationen, welche noch gemacht werden müssen. Bei einer Verifizierung wird nachfolgender Programmcode ausgeführt, um den neuen Status zu setzen.

```
switch (person.getState()) {
    case "toBeVerified3":
        person.setState("toBeVerified2");
        break;
```

```
    case "toBeVerified2":
        person.setState("toBeVerified1");
        break;
    case "toBeVerified1":
        person.setState("verified");
        default:
        break;
}
```

Weiter wird auch der Status des jeweiligen anderen Documents aktualisiert (Beispiel: Person im Status *verified2*, E-Mai im Status *toBeVerified* ➔ E-Mail-Validierung durchgeführt ➔ Neue Status: Person: *toBeVerified1* / E-Mail: *verified*).

Dieselbe Logik wird auch bei den Angebot-Location Paarungen angewendet. Hier muss zwar nur eine Validierung durchgeführt werden, aber es ist möglich, dass ein Angebot erstellt werden kann ohne, dass die dazugehörige Location verifiziert ist. Dann kommt auch das Angebot in einen temporären Status. Sobald dann die Location von einem Admin-User verifiziert wurde, wird auch automatisch das dazugehörige Angebot verifiziert.

7.5.2 Token

Für alle Prozesse in denen eine Verifizierung von einem User, Admin-User oder vom Bauernverband durchgeführt werden muss, wird ein Token generiert. Mit diesem Token wird ein Document in der jeweiligen Collection erstellt. Dafür gibt es die Collections personVerification (Bauernverband oder Admin-User), locationVerification (Admin-User) und emailVerification (End-User). Der gespeicherte Token wird einem Link angehängt und dem User mittels E-Mail verschickt. Mit diesem Link kann das Frontend aufgerufen werden, wodurch ein entsprechender Request an das Backend mit dem Token ausgelöst wird. Da der entsprechende Token in der Datenbank gespeichert ist kann die Verbindung zum entsprechenden Document hergestellt werden und der entsprechende Validierungsprozess wird ausgelöst.

7.5.3 E-Mail

Der E-Mail-Service spielt eine zentrale Rolle, da dies die Schnittstelle zum User darstellt. Die in Kapitel 7.5.2 erwähnten Tokens werden mittels E-Mail verschickt. Für den Versand an den Bauernverband kann in der Datei `config.properties` pro Kanton eine entsprechende E-Mail-Adresse hinterlegt werden.

7.6 Requests

In diesem Kapitel werden die einzelnen Requests genauer beschrieben, welche vom Frontend an das Backend gesendet werden. Die Requests bestehen aus einer URL, einem Header und einem Body. Grundsätzlich werden zwischen prozessspezifischen Requests und generischen Requests unterschieden. Bei einem prozessspezifischen Request handelt es sich um einen Request, welcher aufgrund seiner Funktionalität ausschliesslich im entsprechenden Prozess gebraucht werden kann. Ein generischer Request hingegen kann auch wiederverwendet werden. Auf Standard-Requests wird nicht genauer eingegangen, sondern nur die Requests, welche prozessspezifisch sind.

7.6.1 BookingController

Diese Java-Klasse enthält die Requests für die Collection respektive das Java-Objekt Booking. Dabei sind folgende Requests umgesetzt.

7.6.1.1 Neues Booking erstellen

Mit diesem Request kann ein neues *Booking* erstellt werden. Dabei werden zuerst entsprechende Validierungen ausgeführt. Weiter wird ein E-Mail an den Nachfrager und ein oder zwei E-Mails an den Anbieter verschickt sowie die gebuchte Menge beim entsprechenden Offer auf dem Attribut amountInKgAvailable abgezogen. Somit ist auf der Entity Offer immer klar wie viel davon noch verfügbar ist.

7.6.1.2 Bookings als Nachfrager / Anbieter abfragen

Diese Requets liefern Buchungen zurück. Da ein User Anbieter und Nachfrager sein kann werden zwei verschiedene Requests umgesetzt. Einmal aus Sicht Nachfrager (requester) und einmal aus Sicht Anbieter (supplier). Aus Sicht des Nachfragers sind alle gebuchten Angebote relevant, also die selbst gebuchten *Bookings.* Hingegen aus Sicht des Anbieters, sind die Buchungen, welche für eines seiner Produkte gemacht wurden, relevant.

7.6.1.3 Einzelnes Booking abfragen (Standard GET-Request)

Hierbei handelt es sich um einen Standard GET-Request. Darauf wird nicht genauer eingegangen. Abgesehen von der Berechtigung wird nichts Weiters überprüft.

7.6.2 CategoryController

In diesem Controller sind nur Standard-Requests vorhanden.

7.6.3 ContactFormRequestController

In diesem Controller gibt es nur Standard-Requets. Der Request löst eine E-Mail aus und keine Datenbankmanipulationen. Für diesen Request wird keine Authentifizierung benötigt.

7.6.4 LocationController

Für die Collection Location sind mehrere prozessspezifische Requets vorhanden.

7.6.4.1 Offer-Location hinzufügen

Mit diesem Request kann einer Person eine Offer-Location hinzugefügt werden. Diese Location kann danach bei der Erstellung eines Angebots hinzugefügt werden. Neben der Erstellung der Location selbst wird das Document auf dem Attribut offerLocation auf dem Person-Document hinterlegt.

7.6.4.2 Offer-Location pro Person abfragen

Mit diesem Request können alle Locations der entsprechenden Person abgefragt werden. Es werden Domizil-Locations wie auch Offer-Location zurückgegeben. Der Status der Location spielt dabei keine Rolle.

7.6.4.3 Location validieren

Mit diesem Request kann ohne zusätzliche Validierung mit einem gültigen Token im Link eine Location validiert werden. Dieser Request wird normalerweise vom Frontend aufgerufen und von einem Admin-User ausgelöst.

7.6.5 OfferController

Für die Collection Angebot (Offer) sind diverse Standard-Reqeuts implementiert. Ein spezieller Request ist der GET-Offer mit der Person-Id. Dabei werden alle Angebote zurückgegeben, welche vom entsprechenden User erstellt wurden.

7.6.6 OrganisationController

Der Organisation Controller hat nur Standard-Requests, welche von einem Admin-User ausgeführt werden.

7.6.7 PersonController

Der PersonController enthält nur den Reqeust für das Update einer bestehenden Person. Neben der Person selbst, kann die Location und die E-Mail mitgegeben werden. Die Location kann jedoch in einer ersten Version nicht angepasst werden.

7.6.8 PictureController

In diesem Controller sind nur Standard-Requests enthalten.

7.6.9 ProductController

In diesem Controller ist ein Request für die Erstellung oder Anpassung eines Produkts (Product) und den dazugehörigen Angeboten (Offer) implementiert. Pro Request können mehrere Angebote mitgegeben werden, da es sich um eine ArrayList handelt. Weiter sind noch Standard-Requests umgesetzt.

7.6.10 SchedulerController

Dieser Controller enthält ausschliesslich Requests für das Testing. Der Controller enthält Requests, welche Prozeduren auslöst, welche normalerweise vom Job-Engine ausgelöst werden.

7.6.11 SearchFilterController

In diesem Controller sind ausschliesslich Standard-Requests enthalten.

7.6.12 SearchResultController

In diesem Controller ist der GET-Request für das Suchresultat umgesetzt. Im Request muss eine Personen-Id mitgegeben werden. Damit können alle Suchresultate einer Person abgerufen werden. Der Request ist auf eine schnelle Antwortzeit optimiert.

7.6.13 TypeController

In diesem Controller sind nur Standard-Requests für den Admin-User vorhanden.

7.6.14 AuthController

Im AuthController sind mehrere Requests im Zusammenhang mit der Registrierung und dem Login enthalten. Speziell ist, dass keiner dieser Requests eine Authentifizierung benötigt.

7.6.14.1 Privatperson registrieren

Mit dem Request Privatperson registrieren, kann das Frontend mittels einem Request eine neue Person, eine neue damit verbundene Location sowie eine neue E-Mail-Adresse an das Backend senden. Das Backend meldet dann einen Status zurück. Im Falle eines erfolgreichen Requests werden die entsprechenden Entitäten in der Mongo-Datenbank abgelegt.

7.6.14.2 Kommerzielle Person registrieren

Mit dem Request kommerzielle Person registrieren, kann das Frontend mittels einem Request eine neue Person, eine neue damit verbundene Location sowie eine neue E-Mail-Adresse an das Backend senden. Das Backend meldet dann einen Status zurück. Im Unterschied zum Request Privatperson registrieren kann zusätzlich das Feld companyName mitgegeben werden, was für den Namen des Unternehmens steht. Im Falle eines erfolgreichen Requests werden die entsprechenden Entitäten in der Mongo-Datenbank abgelegt.

7.6.14.3 E-Mail validieren

Dieser Request kann mit Hilfe eines Tokens, welcher bei der Registrierung generiert wird, eine E-Mail-Adresse validieren. Sofern der Token nicht in der Datenbank gefunden werden kann oder abgelaufen ist wird eine entsprechende Meldung zurückgegeben.

7.6.14.4 Einloggen

Dieser Request führ eine Authentifizierungsanfrage aus. Dafür werden E-Mail-Adresse und Passwort benötigt, welche im Body mitgeschickt werden müssen. Als Rückmeldung kommt ein Error oder eine OK-Meldung. Im Falle der OK-Meldung wird ein BearerToken mitgeschickt, durch den man dann auf geschützte Requests zugegriffen kann.

7.6.14.5 Passwort zurücksetzen

Hierfür wurden drei Requests umgesetzt. Der speichert einen Token in der Datenbank und löst eine E-Mail an den User aus, mit dem generierten Token. Der zweite ist ein GET-Request, welcher anhand eines Tokens die E-Mail-Adresse zurückliefert. Dies Request wäre grundsätzlich nicht zwingend nötigt, wurde aber umgesetzt, dass am Frontend nochmals die E-Mail-Adresse angezeigt werden kann, für welche das Passwort geändert wird.

Der dritte Request ändert das Passwort der entsprechenden E-Mail-Adresse. Im Body müssen Token, E-Mail-Adresse (aus dem zweiten Request) und das neue Passwort mitgegeben werden.

7.7 Validierungen

Um eine hohe Datenqualität zu gewährleisten werden diverse Validierungen durchgeführt. Jeder Request, welcher ans Backend gesendet wird und eine Datenbankmanipulation auslöst wird einer zusätzlichen Validierung unterzogen. Da eine zentrale Anforderung ist, die Applikation ohne grossen Aufwand betrieben zu können, muss einerseits die Datenqualität hoch sein und andererseits müssen die Daten automatisch validiert werden. In den nachfolgenden Unterkapiteln ist die Datenvalidierung der einzelnen Collections genauer beschrieben.

7.7.1 E-Mail

Bei der Erstellung einer E-Mail werden zwei Validierungen direkt durchgeführt. Die erste Validierung prüft, ob es sich bei dem Input-Parameter auch um einen E-Mail String handelt. Dabei wird das Pattern mit einer Regex-Expression geprüft (GeeksforGeeks, o. J.). Die zweite Validierung prüft, ob die entsprechende E-Mail-Adresse bereits in der der Datenbank einem User zugeteilt ist. Damit soll vermieden werden, dass dieselbe E-Mail-Adresse von zwei User benützt wird. Grund dafür ist, dass die E-Mail-Adresse einerseits für das Login und andererseits für die Kommunikation verwendet wird. Um zu prüfen, ob die E-Mail bereits in der Datenbank existiert, wurde ein Index auf dem Attribut-E-Mail in der Collection Email erstellt (Spring.io, o. J.). Die Indexierung wurde so erstellt, dass diese als unique markiert wurde. Somit wird von der Datenbank selbst verhindert, dass mehrere Documents mit derselben E-Mail-Adresse erstellt werden können (MongoDB, 2008b).

Auf der E-Mail-Adresse wird eine weitere Validierung durchgeführt, jedoch nicht sofort bei der Erstellung. Dabei wird eine E-Mail mit Validierungstoken an die E-Mail-Adresse geschickt. Erst wenn der User den Validierungstoken bestätigt hat wird die E-Mail-Adresse definitiv validiert. So kann sichergestellt werden, dass der User, welcher sich registriert auch Zugriff auf die angegebene E-Mail-Adresse hat.

7.7.2 Location

Bei der Erstellung der Location werden ebenfalls mehrere Validierungen durchgeführt. Initial wird geprüft, ob eine Postleitzahl, eine Stadt oder ein Dorf und eine Strasse angegeben sind. Danach wird eine API von local.ch aufgerufen, welche prüft, ob es sich um eine gültige Schweizer Adresse handelt. Dieser Ansatz wurde von der Ausgangslage des letztjährigen Bachelorarbeit «Validierung» übernommen (siehe Kapitel 3.4). Die API sucht mit den entsprechenden Angaben die Adresse. Der Validierungscheck gilt als erfolgreich, wenn das API ein Resultat zurückliefert. Die ursprüngliche Implementation sah vor, dass es ein Rating-System gibt, welches Punkte vergibt und so die Validierung durchgeführt wird (Blöchlinger, 2019). Um die Komplexität zu reduzieren, gilt die Location entweder als verifiziert oder nicht. Je nach Usecase kann die Location vom Admin-User validiert werden oder sie kann gar nicht erstellt werden.

7.7.3 Person (Privat)

Auf der Person werden alle Felder geprüft, welche am Frontend ausgefüllt werden können. Beim Namen wird überprüft, ob es sich dabei um einen gültigen String handelt und ob Vor- und Nachname abgefüllt sind. Diese Überprüfung wird wiederum mittels Regex-Patterns gemacht. Weiter wird geprüft, dass die Anrede nicht null ist. Die Telefonnummer wird ebenfalls überprüft. Die Logik der Telefonnummernprüfung wurde selbst entwickelt und ist somit relativ einfach gehalten. Zuerst werden alle Vorwahlen (+41 / 0041) mit 0 ersetzt und danach Klammern und Abstände entfernt. Wenn danach der String nur noch aus Nummern besteht und genau zehn Zeichen hat ist die Prüfung erfolgreich.

7.7.4 Person (Bauer)

Für die Bauern wurde ein zusätzlicher Verifikationsprozess implementiert. Zusätzlich zu den in Kapitel 7.7.3 genannten Validierungen, wird noch der Name des Hofs validiert. Neben der normalen E-Mail-Verifikation wird eine E-Mail an den entsprechenden kantonalen Bauernverband versendet (sofern eine Abmachung besteht, sonst wird die E-Mail an den Admin-User versendet). Diese E-Mail enthält ein Validierungslink, mit dem der Bauer entweder vom Bauernverband oder vom Admin-User verifiziert oder abgelehnt werden kann. Im Falle einer Ablehnung wird noch eine manuelle Verifizierung vom Admin-User gemacht. Bei einer Ablehnung vom Admin-User werden die entsprechenden Documents aus der Datenbank gelöscht.

7.8 Security

Für die Implementation der Security wurde das in Kapitel 6.6 beschriebene Framework verwendet. Die Security wurde so implementiert, dass in Zukunft auch Admin und Moderator-Rollen im Frontend eingebaut werden können. Die jeweiligen Requests müssten dann angepasst werden, aber die Rollen bestehen bereits.

Bei der Implementation der Security wurden gewisse kleinere Anpassungen am Framework und zwei zentrale für diese Applikation wichtige Anpassungen gemacht. Einerseits wurde eine *Runtime-Hashmap* mit den gültigen Validierungs-Keys erstellt. Weiter wird bei Incoming-Requets der Token im Header in die entsprechende *UserId* des eingeloggten Users *umgewandelt*.

7.8.1 Generelle Implementation

Zentral für die Implementation der Security ist die Java-Klasse WebSecurityConfig.java. Dort ist definiert welche Pfade wie berechtigt sind. Im nachfolgenden Programmcode ist die Steuerung der geschützten und ungeschützten Pfade definiert.

```
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
            .exceptionHandling().authenticationEntryPoint(unauthorized
Handler).and()
            .sessionManagement().sessionCreationPolicy(SessionCreation
Policy.STATELESS).and()
            .authorizeRequests().antMatchers("/auth/**").permitAll()

            .anyRequest().authenticated();
            // comment the above line and uncomment the line below for
 testing without auth
            // .anyRequest().permitAll(); // permit all - for testing

        http.addFilterBefore(authenticationJwtTokenFilter(), UsernameP
asswordAuthenticationFilter.class);
    }
```

Die Zeile `.anyRequest().authenticated();` definiert, dass grundsätzlich alle Requests eine Authentifizierung benötigen. Aufgrund der Codezeile `.authorizeRequests().antMatchers("/auth/**").permitAll()` sind Reqeusts, welche mit dem Pfad /auth/ beginnen, davon ausgenommen.

In der Java-Klasse wird der jeweilige Jwt-Token generiert (Bezkoder, 2020). Über die, im File `application.properties`, definierten Variablen wird dann entsprechend ein Token generiert (Bezkoder, 2020). Einer dieser Variablen ist die Gültigkeitsdauer des Tokens, welche aktuell auf 24 Stunden eingestellt ist.

7.8.2 Runtime-Hashmap

Damit im produktiven Betrieb nicht bei jedem Request über die Standard-Funktionalität, die Gültigkeit des mitgegebenen Tokens überprüft werden muss, wurde eine Runtime-Hashmap erstellt. Diese Hashmap wird initial beim Start der Applikation erstellt und wird dann bei einem Login mit dem gültigen Token als Key und einem Datum als Insert-Datum befüllt. Die Befüllung dieser Hashmap wird direkt im RestController des Login-Requests mit der nachfolgenden Programmcode-Stelle ausgeführt.

```
        // save token in map
        GlobalConfigService.getInstance().getTokeMap().put(jwt, oneDay
.getTime());
```

Bei einem geschützten Request wird dann jeweils zuerst geprüft, ob der entsprechende Token in der Run-Time Hashmap enthalten ist. Falls dies der Fall ist, gilt der Token als validiert. Dies konnte mit einer zusätzlichen Bedingung bei den Incoming-Requests implementiert werden.

```
          if (GlobalConfigService.getInstance().getTokeMap().containsK
ey(jwt)
            ||(jwtUtils.validateJwtToken(jwt))) {
               // is authenticated
             }
```

Der markierte Programmcode wurde zusätzlich implementiert. Der Java-Compiler wertet bei einer logischen Oder-Bedingung immer zuerst den ersten Teil aus und wenn dieser True zurückgibt, wird der zweiten Teil ignoriert (Harold, 1999). So kann eine Reduktion des Funktionsaufrufs `jwt.Utils.validateJwtToken()` erzielt werden.

7.8.3 Umwandlung Token in UserId

Da von normalen Requests (abgesehen von denen, die keine Autorisierung benötigen), im Header eine UserId erwarten wird, muss diese bei Incoming-Requets hinzugefügt werden. Dazu wird die UserId mit dem Jwt-Token, ausgelesen und als zusätzlicher Parameter

im Header des Requests hinzugefügt bevor dieser weitergeleitet wird. Um das Hinzufügen der UserId im Header zu gewährleisten, wird der Incoming-Request jeweils in einem Java-Objket vom Typ HeaderMapRequestWrapper gewrappt (Stackoverflow, 2019). Dann wird dem gewrappten Request ein Header hinzugefügt und dieser wird mit der Prozdur `filterChain.doFilter(requestWrapper, response);` weitergeleitet.

7.8.4 Userspezifische Security

Um eine userspezifische Security auf gewissen Requests einzubauen muss jeweils bekannt sein, von welchem User ein Request abgesetzt wurde. Dafür wird der übergebene Token in die UserId umgewandelt (siehe Kapitel 7.8.3). Mit der UserId kann überprüft werden, dass der entsprechende User diesen Request auf dieses spezifische Document durchführen darf. Zum Beispiel darf ein User nur Angebote für seinen eigenen User erstellen und löschen. Daher wird überprüft, ob die UserId des Owners eines neuen Angebots (oder eines zu löschenden Angebots) identisch mit der UserId im Header ist. Falls dies nicht der Fall ist, wird eine entsprechende Response geliefert. Diese spezifische Security ist für POST- / GET- und DELTE-Requests umgesetzt. Die Logik der Überprüfung ist in vielen Fällen in der zentralen Java-Klasse AuthorizationService.java umgesetzt.

7.9 Job-Scheule

Es gibt verschiedene Funktionen, welche regelmässig automatisch angestossen werden müssen. Dazu wird, wie in Kapitel 6.5 beschrieben, die @Scheduled Annotation gebraucht. Dabei wird zwischen *fixedRate* und *cron* (Paraschiv, 2020) unterschieden. Unter *fixedRate* wird eine Anzahl in Millisekunden angegeben. Dies ist der Zeitraum zwischen der letzten Beendigung der Ausführung und des Starts der nächsten Ausführung (Paraschiv, 2020). Mit *cron* kann eine fixe Zeit (zum Beispiel jede volle Stunde oder täglich um 17 Uhr) angegeben werden (Paraschiv, 2020).

7.9.1 Matching-Service

Der Matching-Service wird durch einen @Scheduled Task ausgeführt. Der Matching-Service selbst ist in Kapitel 7.3 genauer beschrieben und wird alle fünf Minuten ausgelöst.

7.9.2 Garbage Collector

Unter einem Garbage Collector werden Prozeduren verstanden, welche nicht mehr benötigte Daten, in einer Datenbank oder Runtime-Daten löschen (Stackify, 2017). Nicht mehr

benötigten Daten sind abgelaufen oder nicht mehr gültig. Im Kontext zur vorliegenden Arbeit wurden konkret vier Garbage Collectors umgesetzt.

Der erste setzt *abgelaufene Offers* in den Status expired sobald diese das Ablaufdatum erreicht haben. Nach 30 Tagen werden diese auf der Datenbank gelöscht. Dieser Job läuft täglich um 19 Uhr. Alle Angebote, die ein Gültigkeitsdatum vor Mitternacht haben werden dann auf expired gestellt und aus den Suchresultaten herausgelöscht. Angebote im Status expried mit einem Gültigkeitsdatum, welches mehr als 29 Tage in der Vergangenheit liegt, werden gelöscht.

Ein weiterer Garbage Collector hat auch einen Sicherheitsaspekt. Die Prozedur löscht Tokens in der Runtime-Hashmap, welche im Kapitel 7.8.2 beschrieben ist. Die Bedingung, dass ein Token aus der Hashmap gelöscht wird, ist in der Hashmap selbst enthalten. Zusätzlich neben dem Token wird in der Hashmap der Ablaufzeitpunkt des Tokens gespeichert. Aus Sicherheitsgründen wird eine Stunde abgezogen, sodass d Dieser Task läuft alle 30 Minuten.

Ein weiterer Garbage Collector, welcher ebenfalls einen Sicherheitsaspekt hat, ist die Löschung der Tokens, welche für das Zurücksetzten des Passworts genutzt werden. Wenn ein Passwort zurückgesetzt wird, wird der entsprechende Token in der Datenbank gespeichert (siehe Kapitel 7.6.14.5). Wenn dieser Token abgelaufen ist, wird er aus der Datenbank gelöscht.

Es gibt ebenfalls einen Garbage Collector für Registrierungen, bei denen die *E-Mail-Adresse nicht bestätigt* wird. Wenn eine E-Mail-Adresse nicht innerhalb von sieben Tagen bestätigt wird, so wird eine E-Mail an die E-Mail-Adresse mit dem Validierungstoken verschickt. Falls der User den Token nicht innerhalb von 30 weiteren Tagen bestätigt, so wird die E-Mail-Adresse mit den ganzen weiteren Userdaten (Collection Person, Location, User, E-Mail) gelöscht. So soll verhindert werden, dass Datenleichen in der Datenbank entstehen.

7.9.3 E-Mail Versand (Suchabo)

Wenn ein neues Angebot für einen User mit entsprechenden Anforderungen in seinem Suchabonnement (siehe Kapitel 6.3.8) aufgeschaltet wird, muss er darüber informiert werden. Dafür wurde eine Prozedur umgesetzt, welche jeweils um 21 Uhr läuft.

8 Testdesign

In diesem Kapitel wird das Testdesign beschrieben. Von der letztjährigen Bachelorarbeiten konnten initial Postman-Requests übernommen werden. Nachfolgend wird analysiert, welche Teststrategien in Frage kommen und werden Testcases dementsprechend aufgebaut werden. Da diverse Prozesse teilweise mehrere Tage benötigen, muss geprüft werden wie solche Funktionalitäten sinnvoll getestet werden können.

8.1 Analyse

Um genauer zu analysieren, welches Testframework in Zukunft sinnvoll ist, muss zuerst Klarheit geschaffen werden, auf welchen Teststufen ein Testing durchführt werden soll. Dabei werden folgende Teststufen analysiert: Unit Testing, Integration Testing, System Testing und Acceptance Testing (Luo, 2001, S. 2). In der Webapplikation «Obst vom Baum» macht es aufgrund des hohen Umsetzungsaufwands initial keinen Sinn, alle Teststufen komplett abzudecken. Trotzdem macht es Sinn zwei Teststufen zu definieren. Vor allem wenn man davon ausgeht, dass die Applikation produktiv eingesetzt wird und dann auch entsprechend Updates gemacht werden. Als erste Teststufe, werden Unit-Tests entwickelt, welche direkt von einem Softwareentwickler durchgeführt werden sollten nachdem ein spezifischer Request oder eine Funktionalität angepasst wird. Dafür wird auf den Postman-Requests aufgebaut, welche von der letztjährigen Bachelorarbeit umgesetzt wurden. Die Requests müssen jedoch angepasst werden, da sich die meisten verändert haben. Die zweite Teststufe soll auf Ebene Systemtest respekitve Acceptance Testing sein. Auch dort wird eine Postman-Collection erstellt. Diese wird jedoch komplett an einem Stück sein und Prozesslogik komplett abbilden.

Das Ziel der ersten Teststufe ist, dass entsprechende Postman-Requests zur Verfügung stehen, welchen ein Softwareentwickler durchführen kann, wenn er eine Anpassung vornimmt. Für die zweite Teststufe ist das Ziel, dass diese möglichst automatisiert durchgeführt werden kann und dass auch entsprechende Testdaten erstellt werden, welche für weitere manuelle Tests benötigt werden können.

Es gibt eine Variable (devMode) im `config.properties` File, welche beim Testen auf `true` gestellt werden soll. Dadurch werden die verschiedenen E-Mails an eine für das Testing dezidierte E-Mail-Adresse verschickt, welche ebenfalls im `config.properties`

File definiert ist (emailTesting). Weiter sind noch Requests umgesetzt, welche ausschliesslich für das Testing genutzt werden. So kann zum Beispiel mit einer UserId auf ein Validierungstoken zugegriffen werden. Dadurch muss nicht manuell im E-Mail auf den Validierungslink geklickt werden, sondern kann mittels der API den Validierungstoken von der Datenbank holen und diesen in den Reqeust für die Validierung einfügen. So kann das Testing automatisiert werden.

8.2 Unit-Testing

Für alle umgesetzten Requests in den REST-Controller sollen entsprechende Postman-Requests vorhanden sein, so dass ein Softwareentwickler diese bei Anpassungen nicht selbst schreiben muss. Dabei wird davon ausgegangen, dass jeder Test in sich unabhängig von den anderen Tests ist und immer wieder wiederholt werden kann. Die einzige Ausnahme ist der Prozess von der Registrierung und dem Login. Als Vorbedingung für jeden Test müssen zwei User (einmal privat und einmal landwirtschaftlich) erstellt und validiert sein. Jedoch wurden auch für diese beiden Prozesse entsprechende Requests implementiert. Über den Login-Requests werden dann für beide User entsprechende Bearer Tokens abgerufen und im Postman in entsprechenden Variablen gespeichert. Dadurch ist es möglich das Login eines Users zu simulieren und auch die Requests so zu simulieren, wie diese vom Frontend ausgelöst werden. Sobald die Requests Login Privat und Login Landwirtschaft durchgeführt sind, soll es möglich sein die restlichen Reqests unabhängig voneinander durchzuführen. Die voneinander abhängigen Requests sind jeweils in sogenannten Collections zusammengefasst.

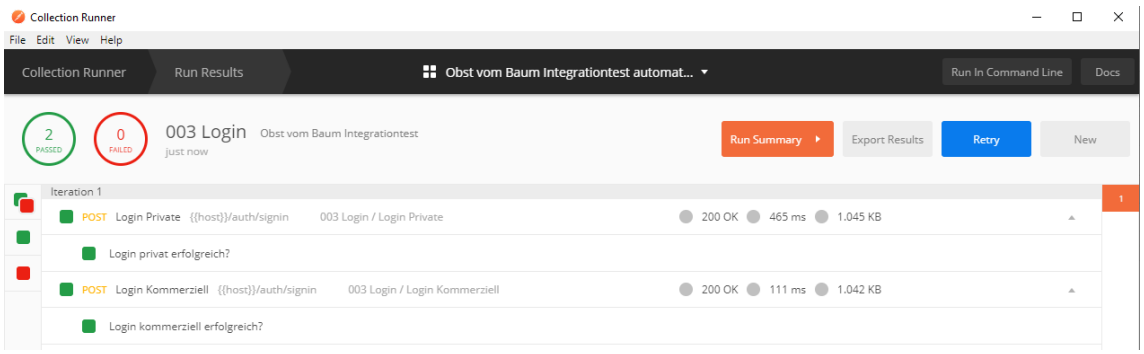


Abbildung 16: Beispiel Unit-Test Login

In der Abbildung 16 ist erkennbar, dass die beiden Requets direkt nacheinander durchgeführt wurden und beide erfolgreich waren. Damit ausgewertet werden kann, ob ein durchgeführter Request erfolgreich war, wurde für jeden Request ein entsprechendes Test-

Skript implementiert. Passt ein Softwareentwickler nun in Zukunft den Login-Request an, kann er mit der Collection *003 Login* überprüfen, ob die Requests noch so funktionieren, wie gewollt. Die vollständigen Unit-Tests werden im Anhang C als ZIP-File mitgeliefert. Weiter ist die Durchführung der Unit-Tests im Anhang E protokolliert.

8.3 System-Testing

Da das Systemtesting automatisiert durchgeführt werden soll, muss eine entsprechende Postman-Collection erstellt werden. Diese beinhaltet alle Requests von der Registierung bis zur Buchung. Weiter werden im Systemtesting die Daten so in der Datenbank angelegt, dass diese für weiteres manuelles Testing gebraucht werden können. Es gibt dezidierte Requests, welche extra für das Testing aufgesetzt wurden und gewisse Daten aus der Datenbank holen oder entsprechende Tasks in der Datenbank anstossen. So können auch die Prozesse abschliessend durchgeführt werden.

9 Rechtsform

In diesem Kapitel wird analysiert, welche Rechtsform bei der Gründung ideal ist und welche Vor- und Nachteile diese mit sich bringt. Der Fokus liegt einerseits bei der Einfachheit zur Gründung und andererseits darauf, dass möglichst wenig Administrationsarbeiten, wie zum Beispiel die Abrechnung der Mehrwertsteuer, anfallen. Zum Schluss wird eine Empfehlung abgegeben.

9.1.1 Analyse

In der ersten Phase der Analyse werden die Rechtsformen eruiert, welche für das Projekt «Obst vom Baum» ausgeschlossen werden können. Dabei wird der Rechtsformenvergleich des Staatssekretariats für Wirtschaft (SECO) beigezogen. Darin werden die Rechtsformen Einzelunternehmen, Kollektivgesellschaft, Gesellschaft mit beschränkter Haftung (GmbH) und Aktiengesellschaft (AG) verglichen (Staatssekretariat für Wirtschaft (SECO), 2019). Zusätzlich zu diesen vier Rechtsformen wird für die Analyse der Verein beigezogen.

Für die Gründung einer GmbH oder einer Aktiengesellschaft wird ein Mindestkapital von CHF 20'000 respektive CHF 100'000 benötigt (Staatssekretariat für Wirtschaft (SECO), 2019). Die Verwaltungskosten, wie auch doppelte Gewinnbesteuerungen zählen zu den Nachteilen (Staatssekretariat für Wirtschaft (SECO), 2019). Aufgrund dieser Nachteile werden GmbH und Aktiengesellschaft für die weitere Analyse ausgeschlossen.

Um eine genauere Übersicht über die Rechtsformen Einzelunternehmen, Kollektivgesellschaft und Verein zu erhalten, werden deren genaueren Eigenschaften in der Tabelle 4 genauer verglichen.

Tabelle 4: Vergleich Rechtsformen (Kanton Zürich Steueramt, 2018) / (Staatssekretariat für Wirtschaft (SECO), 2019

	Einzelunternehmen	**Kollektivgesellschaft**	**Verein**
Besteuerung	Einkommen Inhaber berufliches und privates Vermögen	Anteil an den Erträgen und am Vermögen der Firma	Bund: CHF 5’000 Freigrenze

			Kanton (Zürich): CHF 10’000 Freigrenze
Buchführungspflicht	Jahresumsatz weniger als CHF 500’000: Pflicht zur vereinfachten Buchführung	Jahresumsatz weniger als CHF 500’000: Pflicht zur vereinfachten Buchführung	Keine Buchführungspflicht, sofern gemäss Art. 61, Abs II. ZGB, kein HR-Eintrag nötig.
Mehrwertsteuerabrechnung	Sofern der Umsatz unter CHF 100'000 ist, besteht keine Mehrwertsteuerpflicht	Sofern der Umsatz unter CHF 100'000 ist, besteht keine Mehrwertsteuerpflicht	Sofern gemeinnützig bis zu einem Umsatz von CHF 150'000 keine Mehrwertsteuerpflicht
Gründungsaufwand	Handelsregister obligatorisch ab CHF 100’000 Jahreseinkommen; Anmeldung bei der AHV	durch die Eintragung im Handelsregister und den Abschluss eines Gesellschaftsvertrags (dieser ist freiwillig, aber sehr empfehlenswert)	Da kein Eintrag im Handelsregister nötigt, ohne grossen Aufwand. Es werden nur Statuten benötig, für welche es Vorlagen gibt.
Steuererklärung	Eine Steuererklärung muss immer ausgefüllt werden	Eine Steuererklärung muss immer ausgefüllt werden	Eine Steuererklärung muss immer ausgefüllt werden

In der Tabelle 4 ist ersichtlich, dass es bezüglich Gründung und Verwaltungsaufwand gewisse Unterschiede bei diesen drei Rechtsformen gibt. Dabei fällt auf, dass die Gründung eines Vereines sehr einfach ist. Weiter kommt ein Verein oder eine Kollektivgesellschaft dem Zweck des Projekts «Obst am Baum» näher als ein Einzelunternehmen.

9.1.2 Empfehlung

Wie aus dem Kapitel 9.1.1 Analyse hervorgeht, ist die Gründung eines Vereins, verglichen mit einem Kollektivgesellschaft und einem Einzelunternehmen um einiges einfacher. Weiter profitiert ein Verein von Steuerbefreiung bis zu einem gewissen Gewinnbetrag. Daher wird empfohlen für die Umsetzung des Projekts initial einen Verein zu gründen.

10 Schlussteil

Dieses Kapitel enthält die Erkenntnisse, welche bei der Entwicklung des Programmcodes sowie beim Verfassen der Bachelorarbeit gesammelt wurden. Das Kapitel fasst die gesamte Arbeit prägnant zusammen und zeigt Erkenntnisse auf, welche für zukünftige Implementationen in Frage kommen.

Aus der vorgegebenen Ausgangslage der letztjährigen Analysen und des darin entwickelten Programmcodes, konnte in der vorliegenden Bachelorarbeit aufgebaut werden. Der Fokus lag darauf, den Programmcode aus den bestehenden Teilprojekten und zusammenzuführen sowie das Backend für den produktiven Einsatz fertigzustellen. Wichtig für die Implementation war, dass der Prozess End-to-End (also von der Registrierung bis zur Buchung von Obst) durchgeführt werden kann. Dabei konnte nicht immer alle Funktionalitäten aus den bestehenden Programmcodes übernommen werden. Ein weiterer zentraler Punkt war die Überarbeitung der Datenbankstruktur, dass alle Anforderungen erfüllt werden können sowie die komplette Verlagerung der Security vom Login-Server in den Backend-Server.

In der vorliegenden Bachelorarbeit konnte der entsprechende Programmcode erfolgreich implementiert werden, welcher für den produktiven Betrieb der Plattform «Obst vom Baum» verwendet werden kann. Die Requests wurden dahingehend optimiert, dass diese performant ausgeführt werden können. Ein Resultat ist, dass die Suchresultate aggregiert gespeichert werden, sodass die Daten nicht bei einem Request selbst aufgerufen werden müssen. Weiter wurden die POST-Requets dahingehend implementiert, dass diese prozessspezifisch mittels einem Request durchgeführt werden können, auch wenn die Datenmanipulationen über mehrere Collections laufen. Weiter konnten mehrere automatisierte Prozesse implementiert werden, welche die Administration der Plattform vereinfachen. Die Durchführung der erstellten Testfälle validierten den implementierten Programmcode.

Die Arbeit legt die Basis für den produktiven Betrieb der Plattform «Obst vom Baum». Nichtsdestotrotz gibt es noch Möglichkeiten für eine Weiterentwicklung der Plattform. Ein Teil ist der *Ausbau der Testautomatisierung*. Die in Kapitel 8 beschriebenen Testcases sind initial sicherlich ausreichend, jedoch muss das Testframework wieder überprüft werden, sobald produktive Daten vorhanden sind. Durch produktive Daten muss man nicht nur die Funktionalitäten an sich testen, sondern auch ob die neu implementierten

Funktionalitäten auch noch mit den produktiven Daten funktionieren. Dieser Testaspekt muss zusätzlich berücksichtig werden. Eine weitere Implementation, welche mit zunehmender Nutzung der Plattform Sinn macht, ist die Implementierung eines *Admin-Dashboard* ins Frontend. Dabei ist die Grundlage im Rollenkonzept bereits gelegt. Jedoch müssen im Frontend, wie auch im Backend, entsprechende Funktionalitäten erweitert werden. Zum Beispiel müssen entsprechende Requests aufgesetzt werden und für die richtigen Rollen autorisiert werden. Weiter wurde in der Kommunikation mit den kantonalen Bauernverbänden festgestellt, dass diese verschiedene Anforderung an den Validierungsprozess haben. Da die kantonalen Bauernverbände einen entscheidenden Beitrag zur Datenqualität der Plattform beitragen, müssen diesen Anforderungen ebenfalls nachkommen werden, sobald sich die Nutzerzahlen der Plattform erhöhen. Weiter sollte der Code im aktuellen Status an gewissen Stellen ein Refactoring erhalten, da gewisse Funktionalitäten redundant umgesetzt wurden. Ebenfalls gibt es noch Verbesserungspotenzial in den Tasks für die Datenbereinigung, wie auch an den E-Mails, welche an den User verschickt werden.

Als Fazit wird festgehalten, dass die Arbeit die Implementation des Programmcodes beschreibt, welcher für das Backend der Plattform «Obst vom Baum» verwendet werden kann. Das implementierte Framework bietet, auch nachdem die Applikation in Betrieb genommen wird, stets noch entsprechende Flexibilität für Anpassungen. Weiter konnte die Implementation dahingehend gestaltet werden, dass das Backend eine grössere Userzahl von mehreren tausend Usern verwalten kann.

11 Literaturverzeichnis

Bajaj T. (o. J.). Program for distance between two points on earth. GeeksforGeeks. Abgerufen von https://www.geeksforgeeks.org/program-distance-two-points-earth/

Bezkoder. (15. April 2020). Spring Boot, MongoDB: JWT Authentication with Spring Security. Abgerufen von https://bezkoder.com/spring-boot-jwt-auth-mongodb/

Böhm R. (2002). *System-Entwicklung in der Wirtschaftsinformatik: Systems Engineering*. *(5. Aufl.).* doi: 10.3218/3668-8

Blöchlinger S. (2019). *Plattform «Obst vom Baum»: Datenvalidierung und rechtliche Abklärungen* (Bachelorarbeit). Zürcher Hochschule für Angewandte Wissenschaften, Winterthur.

Bundesamt für Umwelt. (2019). *Lebensmittelabfälle.* Abgerufen von https://www.bafu.admin.ch/bafu/de/home/themen/abfall/abfallwegweiser-a-z/biogene-abfaelle/abfallarten/lebensmittelabfaelle.html abgerufen

Docs Spring (o. J.). Spring Data MongoDB - Reference Documentation. Abgerufen von https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#reference

Driss A. (18. August 2015). Build a location based API with Spring Data MongoDB and GeoJSON. Abgerufen von https://drissamri.be/blog/2015/08/18/build-a-location-api-with-spring-data-mongodb-and-geojson/

GeeksforGeeks (o. J.). Check if email address valid or not in Java. Abgerufen von https://www.geeksforgeeks.org/check-email-address-valid-not-java/

Graf T. (23. Mai 2019). *Plattform «Obst vom Baum»: Backend.* (Bachelorarbeit). Zürcher Hochschule für Angewandte Wissenschaften, Winterthur.

Grünert D. (2019). *Schnittstellen.* Zürcher Hochschule für Angewandte Wissenschaften, Winterthur.

Harold E.-R. (3. Februar 1999). The Order of Evaluation of Logic Operators. Abgerufen von http://www.cafeaulait.org/course/week2/46.html

Kanton Zürich Steueramt. (2018). *Wegleitung für Vereine und Stiftungen.* Abgerufen von https://www.steueramt.zh.ch/internet/finanzdirektion/ksta/de/steuererklaerung/formulare-merkblaetter/_jcr_content/contentPar/form_1/formitems/wegleitung_vereine_u/download.spooler.download.1544004708634.pdf/515+Wegleitung+Vereine-Stiftungen+ZH+2018+HA+Vers-1.pdf

Luo L. (2001). *Software Testing Techniques.* School of Computer Science Carnegie Mellon University. Abgerufen von http://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf

Mapquest (o. J.). Open Geocoding API. Abgerufen von https://developer.mapquest.com/documentation/open/geocoding-api/address/get/

MongoDB. (2008a). MongoDB Manual. Databases and Collections. Abgerufen von https://docs.mongodb.com/manual/core/databases-and-collections/

MongoDB. (2008b). MongoDB Manual. db.collection.createIndex(). Abgerufen von https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/

MongoDB. (2008c). MongoDB Manual. Unique Indexes. Abgerufen von https://docs.mongodb.com/manual/core/index-unique/

OpenStreetMap Forum (9. Juli 2009). Näherung für berechnete Luftlinien-Entfernung. Abgerufen von https://forum.openstreetmap.org/viewtopic.php?id=3941

Paraschiv E. (2020). The @Scheduled Annotation in Spring. Abgerufen von https://www.baeldung.com/spring-scheduled-tasks

Spirig J. (22. Mai 2019). *Plattform «Obst vom Baum»: Ausarbeitung eines Geschäftsmodells, Implementierung und Lancierung einer ersten Version* (Bachelorarbeit). Zürcher Hochschule für Angewandte Wissenschaften, Winterthur.

Spring.io (o. J.). Annotation Type Indexed. Abgerufen von https://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/core/index/Indexed.html

Staatssekretariat für Wirtschaft (SECO) (2019). *Rechtsformenvergleich: Vorteile und Nachteile.* Abgerufen von https://www.kmu.admin.ch/kmu/de/home/praktisches-wissen/kmu-gruenden/uebersicht-rechtsformen/rechtsformenvergleich.html

Stackify (2017). What is Java Garbage Collection? How It Works, Best Practices, Tutorials, and More. Abgerufen von https://stackify.com/what-is-java-garbage-collection/

Stackoverflow (2019). Spring Boot RestController: Intercept incoming requests. Abgerufen von https://stackoverflow.com/questions/55852142/spring-boot-restcontroller-intercept-incoming-requests

Anhänge

A Programmcode

Der Programmcode wurde mittels Github-Repository eingereicht. Das Github-Repository ist folgendes: https://github.com/zhaw-iwi/obstvombaum-backend. Der Code soll bis zum Commit mit dem Namen finalcommit2 (Id 94af13ccd8cae294feb46a61da3782ad39e4c24e) vom 27. Mai 2020 berücksichtigt werden.

B Flussdiagramme als Camunda-Datei

Damit die enthaltenen Flussdiagramme weiterverwendet und angepasst werden können, werden diese ebenfalls als Bestandteil der Bachelorarbeit eingereicht. Diese sind als Anhang auf Molena geladen (Dateiname: BSC_FS20_FlavianThuer_Projekt_Obst_Vom_Baum_Backend_Flussdiagramme.zip)

C Unittest – Postman-Requests

Die Postman-Requests für das Unittesting wurden als Anhang auf Molena geladen (Dateiname: BSC_FS20_FlavianThuer_Projekt_Obst_Vom_Baum_Backend_IT.zip)

D Systemtest – Postman-Requests

Die Postman-Requests für das Systemtesting wurden als Anhang auf Molena geladen (Dateiname: BSC_FS20_FlavianThuer_Projekt_Obst_Vom_Baum_Backend_ST.zip)

E Unittest – Durchführung

Nachfolgend die die Nachweise für die erfolgreiche Ausführung des Unit-Testings aufgeführt. Es wurde jeweils ein Screenshot von den ‘passed’ Testfällen pro Kategorie gemacht.

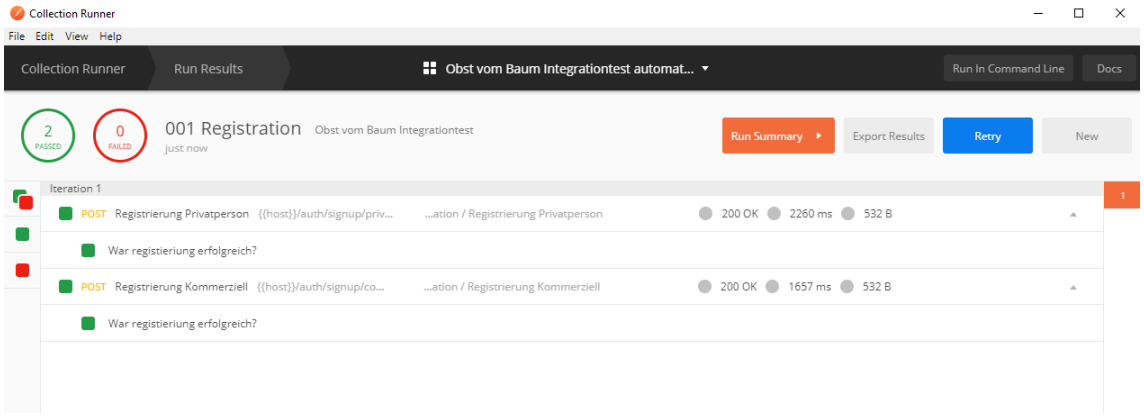


Abbildung 17: Unit-Testing - Testevidenz - Registrierung

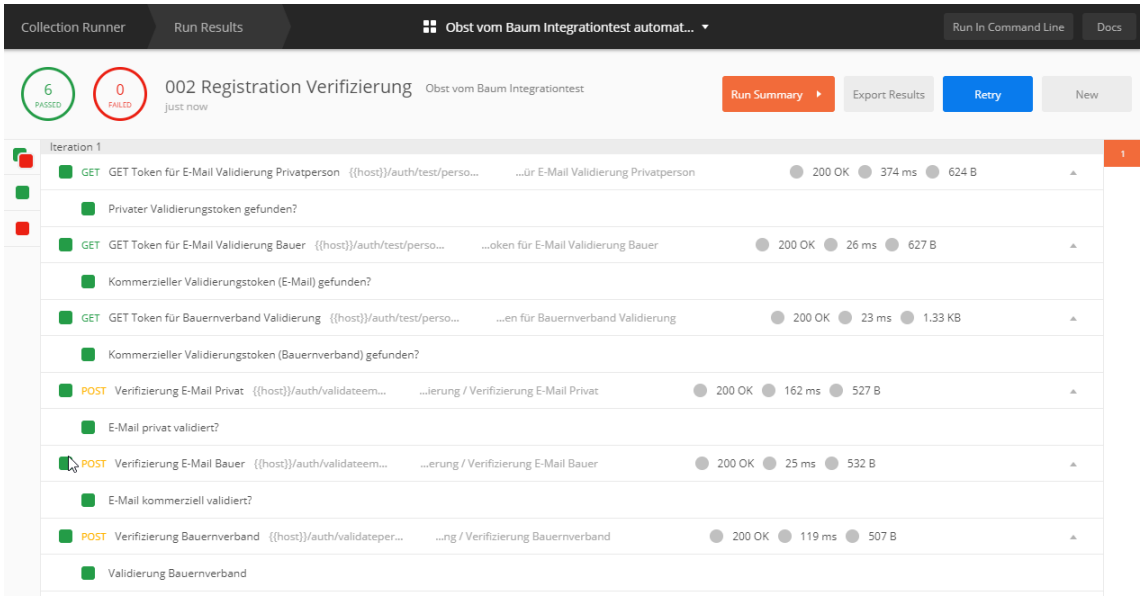


Abbildung 18: Unit-Testing - Testevidenz - Registrierung Verifikation

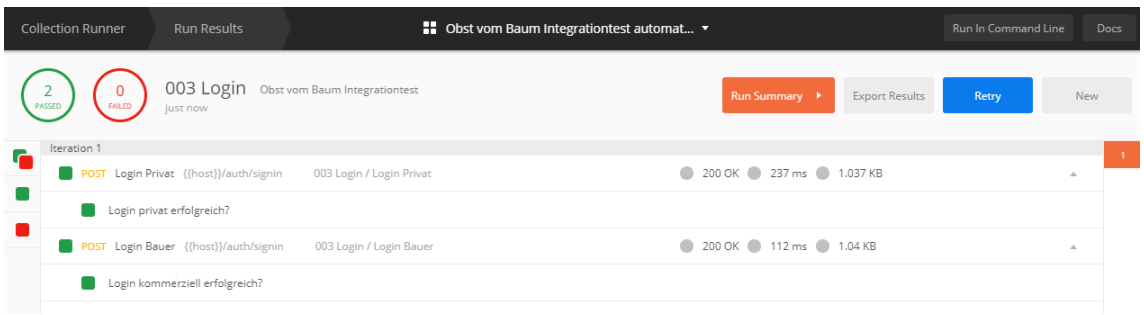


Abbildung 19: Unit-Testing - Testevidenz - Login

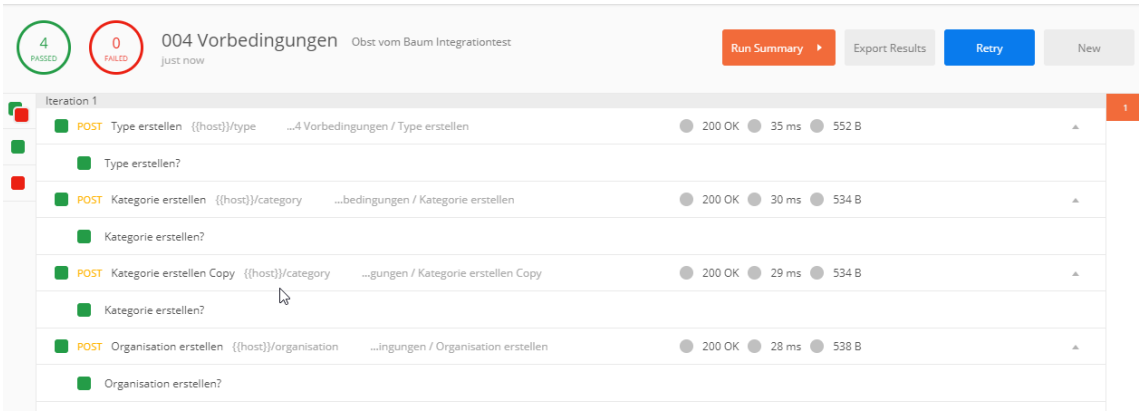


Abbildung 20: Unit-Testing - Testevidenz - Vorbedingungen

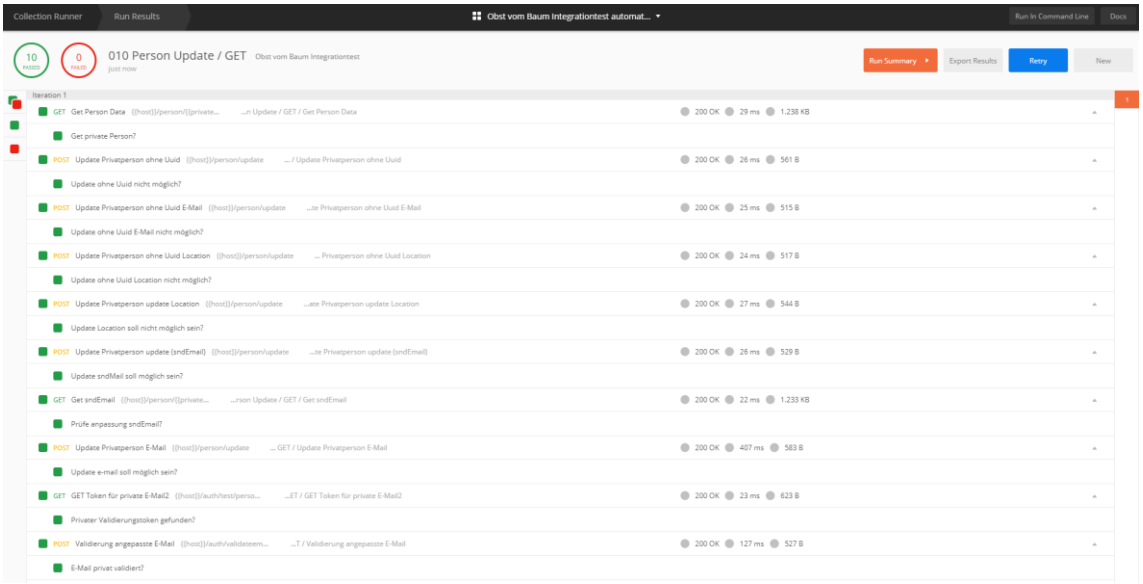


Abbildung 21: Unit-Testing - Testevidenz - Person Update / Get

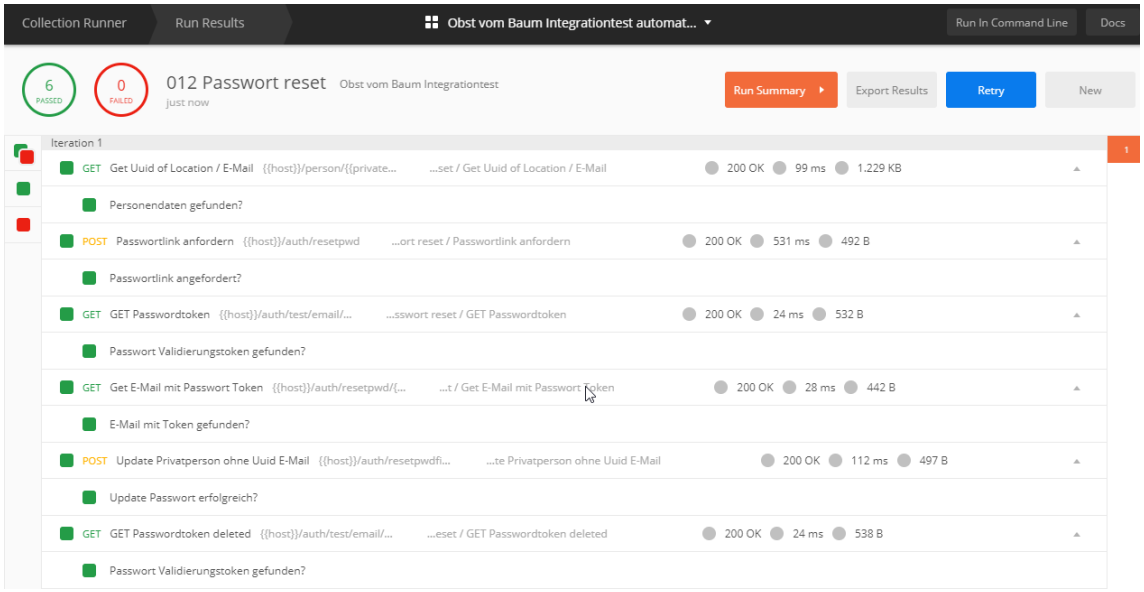


Abbildung 22: Unit-Testing - Testevidenz - Passwort zurücksetzten

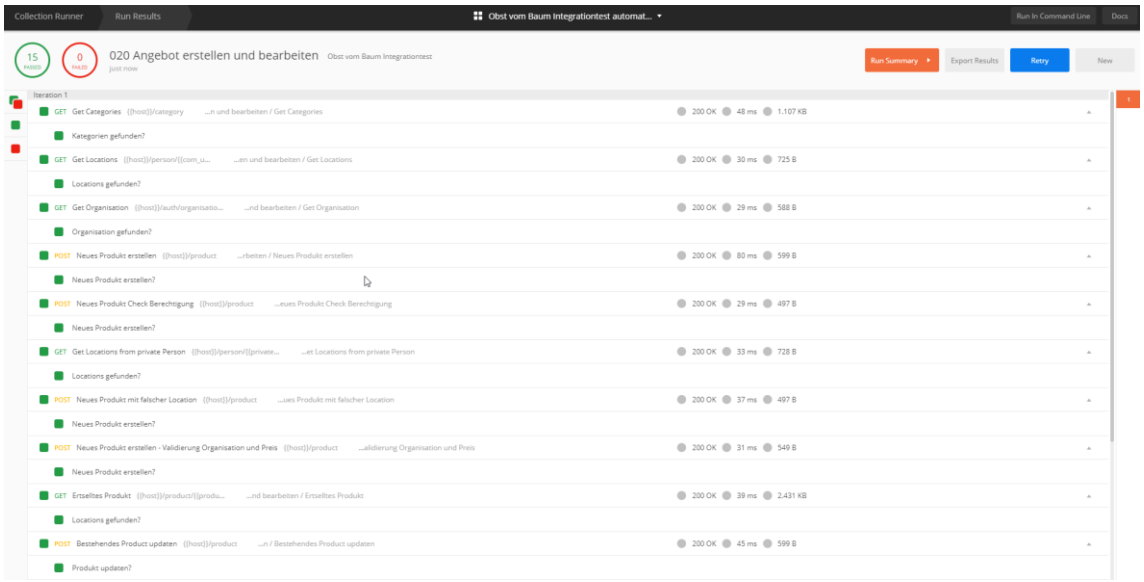


Abbildung 23: Unit-Testing - Testevidenz - Angebote erstellen und bearbeiten

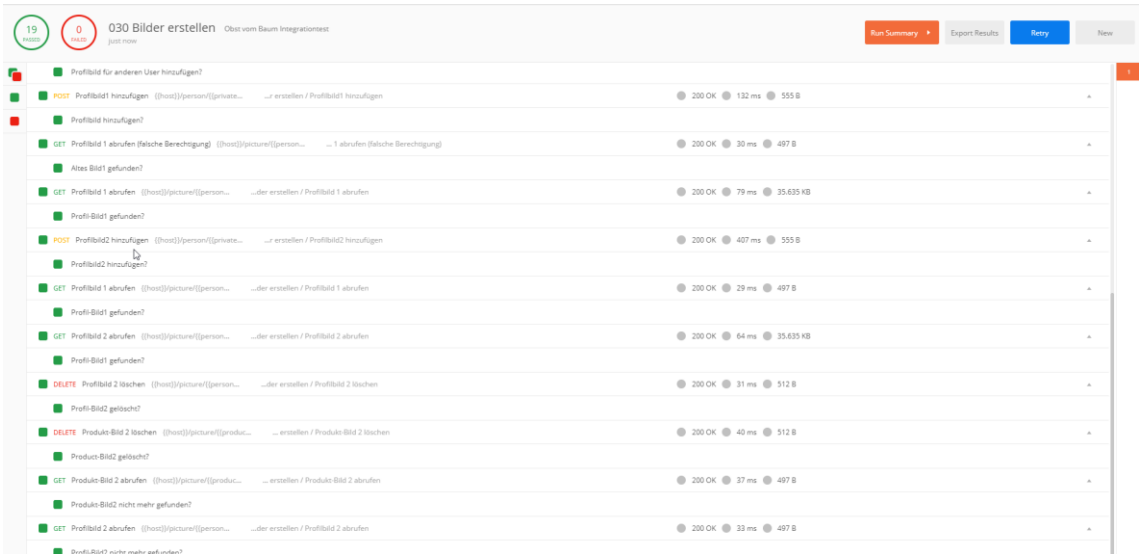


Abbildung 24: Unit-Testing - Testevidenz - Bilder erstellen

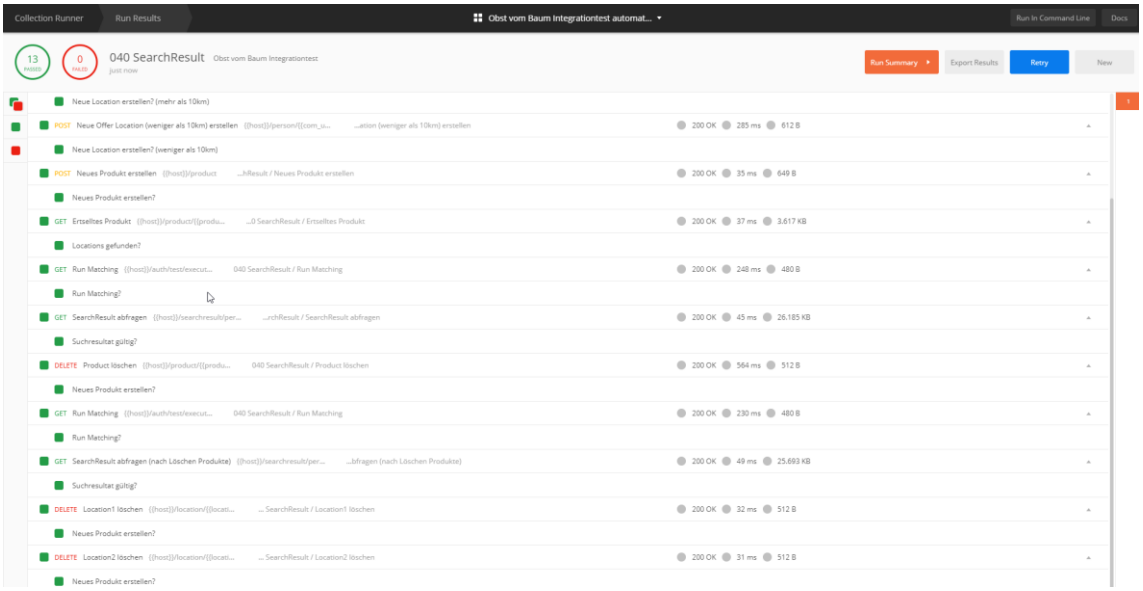


Abbildung 25: Unit-Testing - Testevidenz - SearchResult

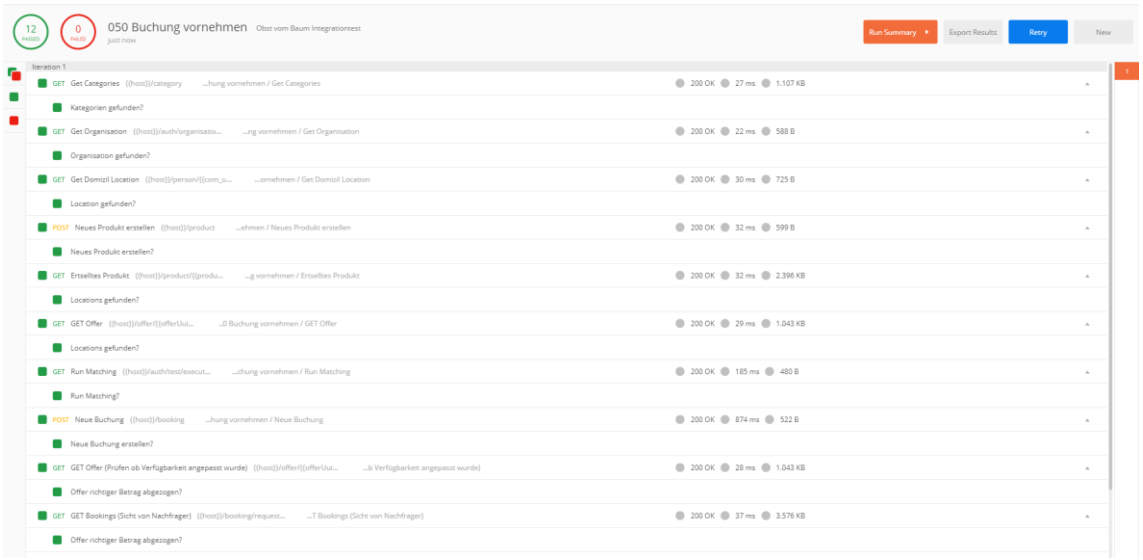


Abbildung 26: Unit-Testing - Testevidenz - Buchung vornehmen

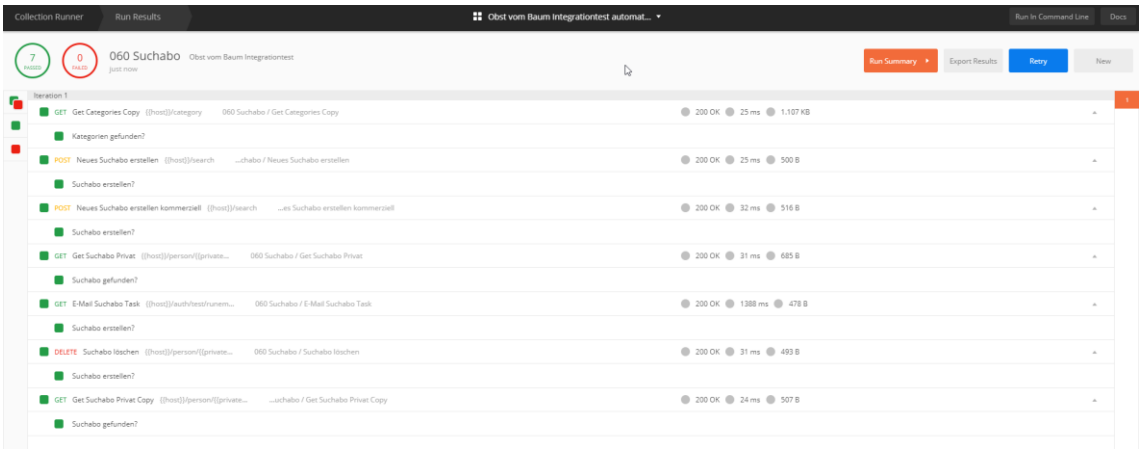
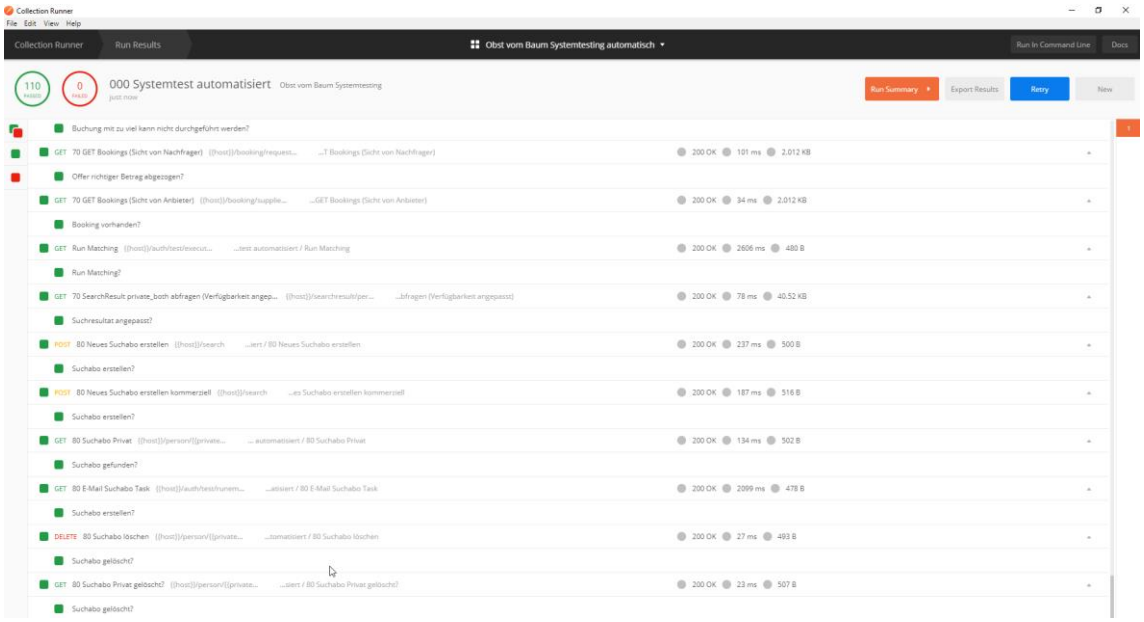


Abbildung 27: Unit-Testing - Testevidenz - Suchabo

F Systemtest – Durchführung

Nachfolgend die die Nachweise für die erfolgreiche Ausführung des System-Testings aufgeführt. Es wurde ein Screenshot der ‘passed’ Testfälle gemacht.


Collection Runner
File Edit View Help
Collection Runner
Run Results
Obst vom Baum Systemtesting automatisch
Run In Command Line
Docs
110
PASS
0
FAIL
000 Systemtest automatisiert
Obst vom Baum Systemtesting
just now
Run Summary
Export Results
Retry
New
Buchung mit zu viel kann nicht durchgeführt werden?
GET 70 GET Bookings (Sicht von Nachfrager) {{host}}/booking/request... ...T Bookings (Sicht von Nachfrager) 200 OK 101 ms 2.012 KB
Offer richtiger Betrag abgezogen?
GET 70 GET Bookings (Sicht von Anbieter) {{host}}/booking/supplie... ...GET Bookings (Sicht von Anbieter) 200 OK 94 ms 2.012 KB
Booking vorhanden?
GET Run Matching {{host}}/auth/test/execut... ...test automatisiert / Run Matching 200 OK 2606 ms 480 B
Run Matching?
GET 70 SearchResult private_both abfragen (Verfügbarkeit angep... {{host}}/searchresult/per... ...bfragen (Verfügbarkeit angepasst) 200 OK 78 ms 40.52 KB
Suchresultat angepasst?
POST 80 Neues Suchabo erstellen {{host}}/search ...iert / 80 Neues Suchabo erstellen 200 OK 237 ms 500 B
Suchabo erstellen?
POST 80 Neues Suchabo erstellen kommerziell {{host}}/search ...es Suchabo erstellen kommerziell 200 OK 187 ms 516 B
Suchabo erstellen?
GET 80 Suchabo Privat {{host}}/person/{{private... ... automatisiert / 80 Suchabo Privat 200 OK 134 ms 502 B
Suchabo gefunden?
GET 80 E-Mail Suchabo Task {{host}}/auth/test/runem... ...atisiert / 80 E-Mail Suchabo Task 200 OK 2099 ms 478 B
Suchabo erstellen?
DELETE 80 Suchabo löschen {{host}}/person/{{private... ...tomatisiert / 80 Suchabo löschen 200 OK 27 ms 493 B
Suchabo gelöscht?
GET 80 Suchabo Privat gelöscht? {{host}}/person/{{private... ...siert / 80 Suchabo Privat gelöscht? 200 OK 23 ms 507 B
Suchabo gelöscht?