



HSB

Hochschule Bremen
City University of Applied Sciences

Dokumentation

MICROSERVICE WARENWIRTSCHAFT MM / GO-TEAM

Hochschule Bremen – Fakultät für Elektrotechnik und Informatik
Studiengang Komplexe Softwaresysteme (M.Sc.)

Eingereicht von

MERLE LABUSCH und MARTIN MÜLLER

Dozenten

MATTHIAS STOCK UND RENZO KOTTMANN

Bremen, 30.06.2017

1 Microservice-Steckbrief

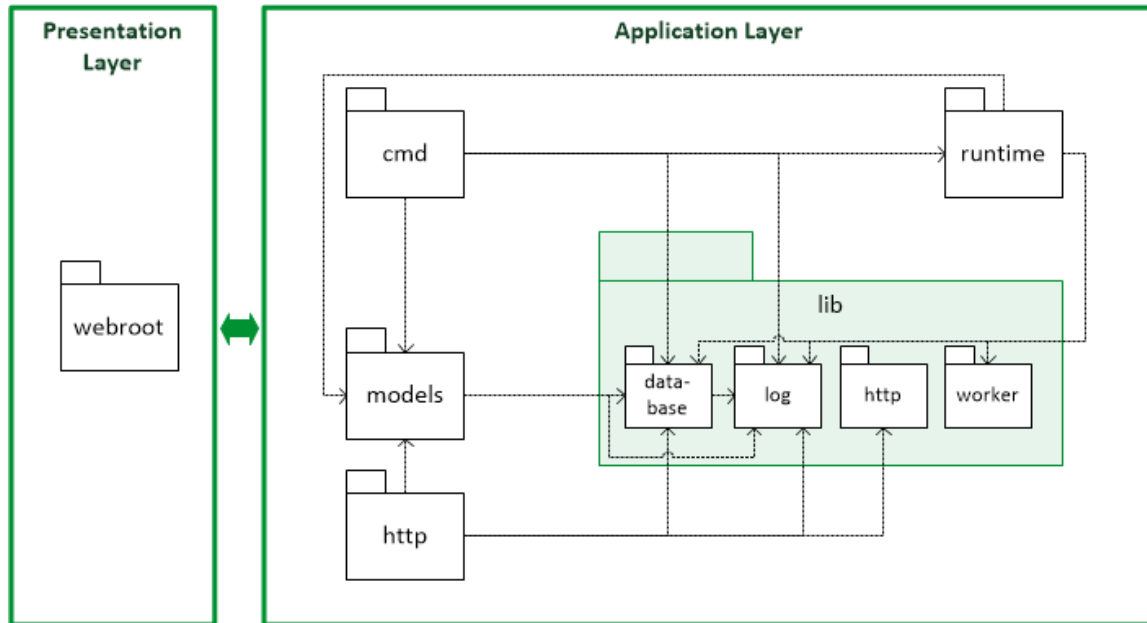


Abbildung 1: Paketdiagramm Microservice Warenwirtschaft

- Der Microservice Warenwirtschaft speichert die einzelnen Waren pro Produkt mit ihrem Lagerort und Ablaufdatum
- Das Admin-Frontend erlaubt das Hinzufügen sowie manuelle Löschen von Waren aus dem Warenbestand und zeigt zusätzlich eine Übersicht der Warenbestände
- In dem Kunden-Frontend wird der Warenbestand durch ein Ampelsystem dargestellt
- Der Microservice wurde in Go entwickelt, die Abbildung ?? gibt einen Überblick der Package-Struktur
- Der statische Inhalt der Webseite ist in dem Package `webroot` verordnet
- Die Hauptfunktionalitäten, die zentralen Structs sowie die notwendigen Hilfsfunktionen sind in den Packages `http` und `models` zu finden
- Eine SQL-Lite-Datenbank stellt den Persistant Layer

Inhaltsverzeichnis

2 Dokumentationsstruktur

Für die Dokumentation des Microservice Warenwirtschaft wurden eine Kombination aus drei Dokumenten gewählt. Zum einen beschreibt ein bebildertes Handout auf einer Seite die Funktionen des Admin-Frontends für dessen Benutzer. Diese sehr kurze Dokumentenform wurde gewählt, da Benutzer häufig nicht gewillt sind umfangreiche Anleitungen zu lesen um eine Anwendung nutzen zu können. Vielmehr wollen sie schnell einen Überblick der Kernfunktionalitäten erhalten. Aus diesem Grund wurde auch auf eine Benutzerdokumentation für die Kunden des Webshops Mosh verzichtet, da anzunehmen ist dass die Darstellung der Produktverfügbarkeit als Ampel sich selbst erklärt.

Auf der anderen Seite muss der Microservice auch für die Entwickler dokumentiert sein, hierfür wurde diese Dokumentation angelegt. Sie beginnt anstelle eines Abstract mit einem Steckbrief des Microservice, der dessen grundlegende Struktur und Funktionalität kurz beschreibt. Das eigentliche Dokument beschreibt zunächst die Anforderungen an den Microservice, da die Entwicklung sich primär an diesen orientiert. Weiter werden der Microservice mit seinem Aufbau, den Schnittstellen und der Anpassung des gegebenen Monolithen sowie Implementierungsregeln beschrieben. Dieses Dokument schließt mit einem *Getting Started* Guide. Auf Details wie ein Abkürzungs- oder Literaturverzeichnis wurde in dieser Dokumentation bewusst verzichtet, um sie kurz zu halten. Zitate und Verweise werden hier in Form von Fußnoten integriert.

Abschließend dokumentiert das Testprotokoll – als drittes Dokument – die für diesen Microservice angewendeten Black-Box-Testfälle. Das heißt es umfasst solche Tests die anhand der Anforderungen und aus Sicht des Benutzers durchgeführt wurden.

3 Definition der Anforderungen

Der Microservice Warenwirtschaft dient der Verwaltung der Warenbestände für den Webshop Mosh. Die nachfolgende Tabelle ?? definiert die hier verwendeten Begriffe, so wie sie in dem Code und innerhalb dieser Dokumentation genutzt werden.

Tabelle 1: Begriffsdefinition

Begriff	Englische Übersetzung	Bedeutung
Produkt	Product	Über den Webshop angebotene Früchte- oder Gemüseart, zum Beispiel Kiwis
Ware	Good	Einzelne Frucht oder einzelnes Gemüse pro Produkt (zum Beispiel eine Kiwi)
Warenbestand	Stock	Anzahl der einzelnen Waren pro Produkt, die sich im Lager befinden

Die übergeordnete Aufgabe dieses Microservice ist die Speicherung und Verwaltung der Waren mit ihrem Lagerort sowie einem Ablaufdatum, da es sich bei Obst und Gemüse um verderbliche Waren handelt. Nachfolgend sind die weiteren, detaillierten Anforderungen an diesen Microservice zusammengefasst.

– Funktionen des Admin-Frontends

- Hinzufügen neuer Waren zum Warenbestand
- Manuelles Entfernen von Waren aus dem Warenbestand, zum Beispiel wenn diese verdorben sind

– Funktionen für andere Microservice / dem Beispiel Warenkorb

- Entfernen von einzelnen Waren aus dem Warenbestand, wenn diese an einen Kunden versendet werden
- Blockieren von Waren in dem Warenbestand, wenn ein Kunde diese in seinen Warenkorb gelegt hat
- Automatische Freigabe von blockierten Waren, wenn diese nicht innerhalb einer Frist an den Versand überstellt werden

– Funktionen des Kunden-Frontends

- Anzeige des Warenbestands über ein Ampelsystem

– Optionale Zusatzfunktionen

- Admin-Frontend: Ausgabe einer Statistik, wie viele Waren sich gesamt und durchschnittlich im Warenbestand befinden
- Admin-Frontend: Ampeldarstellung pro Ware, die Anzeigt ob diese bereits ihr angegebenes Ablaufdatum erreicht oder überschritten hat

Die Angabe der Anzahl ist bei dem Hinzufügen neuer Waren zum Warenbestand verpflichtend, da ohne sie die Verwaltung neu eingetroffener Waren nicht möglich ist. Gleiches gilt für die Angabe des Ablaufdatums, diese ist speziell bei der Verwaltung von Lebensmitteln notwendig, um den Verkauf von verdorbenen Waren zu vermeiden. Die Datumsangabe erfolgt dabei im amerikanischen Format *Jahr-Monat-Tag*.

Im Gegensatz dazu sind die Angabe von Lagerplatz und Kommentar bei dem Hinzufügen neuer Waren optional, da diese Informationen für die Verwaltung des reinen Warenbestandes nicht essentiell notwendig sind. Diese beiden Angaben sind als Freitextfelder umzusetzen, um speziell bei dem Lagerort flexibel die Nutzung verschiedener Benennungsschema für Regale oder Lagerräume zu ermöglichen und den Benutzer hier nicht einzuschränken.

Der Microservice ist in den bestehenden Monolithen Mosh zu integrieren, eine Kommunikation mit anderen Microservices wird jedoch nicht hergestellt.

4 Architektur des Microservice

Der Microservice Warenwirtschaft wurde in der Programmiersprache Go¹ entwickelt. Go-Anwendungen bestehen aus Packages, in denen die einzelnen Go-Files organisiert sind. Ohne Klassen zu besitzen ist Go objektorientiert, es besitzt Struct und Interface, die auch vererbt werden können. Des Weiteren wird durch Groß-/Kleinschreibung definiert, ob die Methode auch außerhalb des Struct oder Packages verwendet werden kann. Die Warenwirtschaft setzt sich aus den neun Packages zusammen, die die Abbildung ?? darstellt. In den nachfolgenden Unterkapiteln ?? und ?? werden die Packages und die darin enthaltenen Go-Files des Presentation sowie des Application Layers kurz vorgestellt.

Go-Files mit der Bezeichnung `«Name»_test.go` beinhalten Whitebox-Testfälle um die Funktionen der benannten Go-Files zu prüfen. Aus Gründen der Übersichtlichkeit werden diese Files hier nicht explizit aufgeführt. Die weiteren Unterkapitel beschreiben die Schnittstellen, den Persistent Layer sowie das Admin-Frontend und schließlich die Anpassung des Monolithen, um den Microservice Warenwirtschaft in diesen zu integrieren.

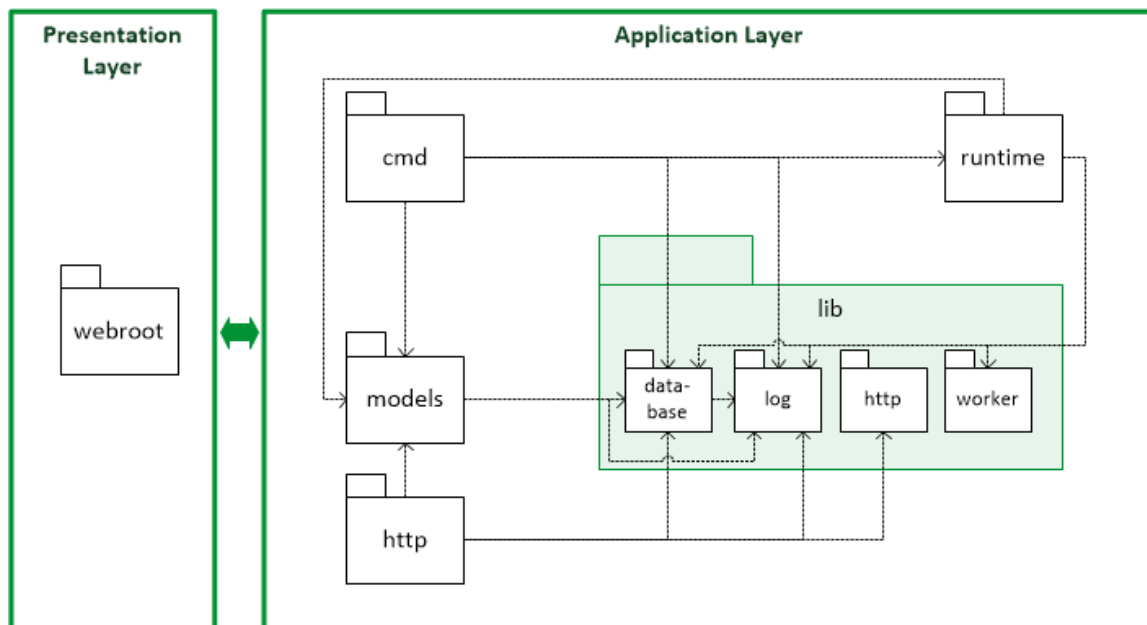


Abbildung 2: Paketdiagramm des Microservice

¹<https://golang.org/doc>

4.1 Schnittstellen zu anderen Microservices

Der Microservice Warenwirtschaft weist drei Schnittstellen² zu anderen Microservices auf. Zunächst greift der Microservice Warenwirtschaft für die Authentifizierung der Benutzer des Admin-Frontends vollständig auf einen Microservice Benutzerauthentifizierung zurück. Anstelle einer Login-Maske weist das Admin-Frontend deshalb nur einen Icon in Form eines Schlosses auf. Dieser symbolisiert, ob ein Benutzer die passende Berechtigung für das Admin-Front besitzt (Schloss geschlossen) oder nicht (Schloss geöffnet). Durch diesen Icon wird ein Session-Token gesetzt, welcher durch den gemocked Authentifizierten Microservice die benötigten Berechtigungen erlaubt.

Weiter benötigt der Microservice Warenwirtschaft Informationen darüber, ob ein Benutzer eine Ware in den Warenkorb gelegt hat und ob eine Bestellung abgeschlossen wurde. So können Waren im Warenkorb für die Bestellung durch andere Benutzer blockiert und die erfolgreich bestellten Waren aus dem Warenbestand gelöscht werden. Diese Funktionalitäten wird durch eine Schnittstelle zu dem Microservice Bestellung zur Verfügung gestellt. Für ein interaktiven Test, wurde ein kleiner Warenkorb Webseite entwickelt, welcher im localStorage des Webbrowsers benutzt.

Die dritte Schnittstelle besteht zu dem Microservice Produktkatalog, von welchem die Warenwirtschaft die angebotenen Produkte – die sich dementsprechend im Lager befinden können – abfragt. Das nachfolgende Listings zeigt die Daten, die von dem Microservices Produktkatalog im JSON-Format erwartet werden.

Listing 1: Datenabfrage aus dem Produktkatalog

```
{
  "id": <<Int>>,
  "name": "<<Produktname>>"
}
```


²Da es nicht Teil der übergeordneten Aufgabenstellung war, die Microservices der einzelnen Projektgruppen zu einem lauffähigen Webshop zusammenzufügen, greift die Warenwirtschaft an diesen Stellen auf Testdaten zurück

4.2 Presentation Layer – Admin-Frontend

Der Presentation Layer umfasst alle Packages, die sich mit der eigentlichen Darstellung der Warenwirtschaft aus der Sicht des Endbenutzers befassen. Im Detail ist dies das Package `webroot`, welches den statischen Inhalt der Frontends, wie zum Beispiel die HTML-Files und Bilder enthält.

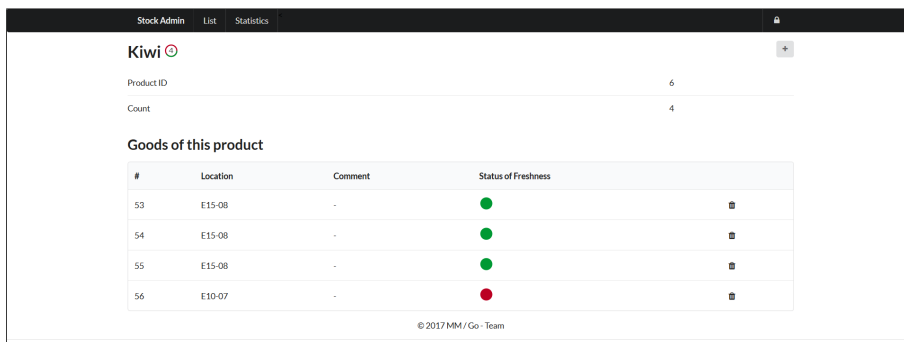
Die Startseite *List* des Admin-Frontends zeigt eine Übersicht aller vorhandenen Produkte mit ihrer Anzahl an Waren (Abbildung ??). Letztere wird mit einem Ampelsystem dargestellt, wobei ein vollkommen rot gefärbter Kreis einem Warenbestand von null entspricht und der Kreis sich mit zunehmender Anzahl an Waren immer mehr grün eingefärbt. Zu jedem Produkt können über den, mit einem Plus, gekennzeichneten Button Waren hinzugefügt werden. Ein Klick auf das jeweilige Produkt führt zu dessen Produktseite.

Die Produktseiten führen die ID, die Gesamtanzahl an Waren sowie die einzelnen Waren auf (Abbildung ??). Diese können jeweils über den Icon in Form eines Mülleimers manuell gelöscht werden. Auch auf den Produktseiten sind über einen, mit einem Plus gekennzeichneten, Button neue Waren hinzufügbare. Beim Hinzufügen von neuen Waren sind für diese ein Ablaufdatum, eine Lagerposition sowie ein Kommentar und die Anzahl anzugeben (Abbildung ??). Die Seite *Statistics* gibt letztendlich einen Überblick der gesamten und der durchschnittlichen Waren im Warenbestand.



#	Productname	Amount
1	Apple	0
2	Pear	0
3	Strawberry	0
4	Blueberry	0
5	Kaki	0
6	Kiwi	0

Abbildung 3: Admin-Frontend – List



#	Location	Comment	Status of Freshness
53	E15-08	-	Green
54	E15-08	-	Green
55	E15-08	-	Green
56	E10-07	-	Red

Abbildung 4: Admin-Frontend – Produktseite



The screenshot shows a web interface for an admin panel. At the top, there are navigation tabs: 'Stock Admin', 'List', and 'Statistics'. The main content area is titled 'Apple' with a small icon. Below the title, there is a form with the following fields:

- Expiration Date:** A text input field with a placeholder 'Expired at date (e.g. 2017-06-30)'.
- Position:** A text input field with a placeholder 'Location in Store'.
- Comment:** A text input field with a placeholder 'Comment for this good'.
- Count:** A text input field with the value '1' and a small icon to its right.

At the bottom of the form is a 'Submit' button. Below the form, there is a copyright notice: '© 2017 MM / Go-Team'.

Abbildung 5: Admin-Frontend – Hinzufügen von Waren

4.3 Application Layer

Die Packages und Go-Files des Application Layers umfassen die Logik des Microservice Warenwirtschaft. Sie werden nachfolgend aufgelistet und kurz beschrieben.

cmd: Go-File `main.go`, welches die Applikation letztendlich ausführt und alle Angaben zu den Config-Files der Applikation enthält

http: Go-Files, die die Anwendungslogik (Funktionen) und die API-Routen beinhalten

- `bindapi.go`: Funktionen, die für das Binden der URL-Pfade notwendig sind
- `good.go`: Funktionen für das Hinzufügen von Waren zum Warenbestand und die Anzeige, ob eine Ware abgelaufen ist
- `good_lock.go`: Funktionen für das Blockieren von Waren, die sich im Warenkorb befinden
- `good_show.go`: Funktionen für die Auflistung und Zählung der vorhandenen Waren sowie die Feststellung ihrer Verfügbarkeit
- `good_temp.go`: Hilfsfunktionen, die für die Darstellung des Warenbestandes als Ampel im Kunden-Frontend benötigt werden
- `status.go`: Funktion, die den Status des Microservice abfragt

models: Go-Files, die die zentralen Structs und zugehörige Hilfsfunktionen (hauptsächlich statischen Inhalt des Microservice) beinhalten

- `config.go`: Structs mit den Informationen zur Konfiguration des Webservers, der Datenbank und dem Cache-Management sowie Hilfsfunktionen zum Lesen von Config-Files
- `duration.go`: Structs und Hilfsfunktionen zur Definition eines Typs für Zeitangaben
- `good.go`: Structs und Hilfsfunktionen zur Darstellung von Waren, hier werden auch die geforderten Funktionalitäten wie das Blockieren von Waren umgesetzt

runtime: Go-Files mit weiteren Hilfsfunktionen

- `auth.go`: Hilfsfunktionen zur Prüfung, ob eine Berechtigung für den Zugriff vorliegt
- `cache_worker.go`: Hilfsfunktionen für das Löschen und Anlegen von Cache-Workers
- `good_fouled.go`: Hilfsfunktion, um abgelaufene Waren automatisch aus dem Warenbestand zu entfernen
- `good_release.go`: Hilfsfunktionen zum Blockieren und Entsperren von Waren
- `productcache.go`: Hilfsfunktionen zum Anlegen eines Caches für Produkte

lib: fasst die vier Packages `database`, `http`, `log` und `worker` zusammen

- `database`: Go-File `database.go` mit Funktionen für das Öffnen und Schließen der Datenbank
- `http`: Go-Files, die die Webserverlogik umgesetzt
- `io.go`: Funktionen zum Lesen und Schreiben von JSON aus beziehungsweise in HTTP-Pakete
- `permission.go`: Funktionen zur Prüfung der Berechtigung für den Zugriff
- `log`: Go-File `log.go`, das den Logger startet und initiiert
- `worker`: Go-File `worker.go`, das Funktionen für regelmäßige Aktivitäten bereitstellt. (z.B. das Entsperren von Waren nach einer vorgegeben Zeit)

4.4 Persistent Layer

Der Persistent Layer umfasst beliebige SQL-Datenbanken, deren Tabellen-Struktur beim Starten automatisch angelegt. Die nachfolgende Abbildung ?? zeigt den grundsätzlichen Aufbau der Datenbank. Sie speichert den Warenbestand (stock). Zur Referenzierung wird vom Produkt die ID verwendet, die aus dem Produktkatalog bezogen und bei Anfrage in einem Cache zwischengespeichert werden. Vom Microservice werden nur die Waren (good) gespeichert und verwaltet, die eine ID, ein Ablaufdatum, eine Lagerposition und einen Kommentar besitzen. Dabei kann eine Ware nur zu einem Produkt gehören. Die Datenbank kann über die Konfigurationsdatei `config_example.conf`, deren relevanter Ausschnitt nachfolgend dargestellt wird, flexibel angepasst werden. Zum Testen wurde eine SQL-Lite-Datenbank vorkonfiguriert, die im Cache gehalten wird.

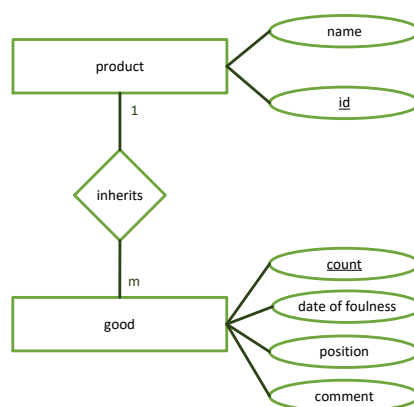


Abbildung 6: Datenbankmodell des Microservice

Listing 2: Datenbankeinstellungen in der Konfigurationsdatei

```
[database]
type = "sqlite3"
# logging = true
```

```
connection = "file::memory:?mode=memory&cache=shared"  
# For Master-Slave cluster  
# read_connection = ""
```

4.5 Integrierte Tests

Neben den bisherigen Packages, die bereits Whitebox-Tests umfassen, ist in dem Package **test** ein weiteres Go-File (`testrest.go`) enthalten. Dieses setzt eine Testumgebung um, bei dem auf Testdaten eines Produktkataloges zurückgegriffen wird. Mit Hilfe der integrierten Tests wird in der hier beschriebenen Version eine Code-Coverage von 100% erreicht, das heißt jedes Stück Code wird mindestens einmal zur Ausführung gebracht. Dies bedeutet nicht, dass dieser Microservice fehlerfrei ist, sondern lediglich das aus Entwickler-Sicht, alles getan wurde.

4.6 Anpassung des Monolithen

Damit der Microservice Warenwirtschaft durch den bestehenden Monolithen des Webshops Mosh genutzt werden kann, wurden hier die unter dem Anhang ?? aufgeführten Änderungen vorgenommen.

5 Implementierungsregeln

Die folgende Aufzählung gibt einige Regeln für die Implementierung des Microservice Warenwirtschaft vor. Diese sollen im Rahmen einer Weiterentwicklung eingehalten werden, um die Konsistenz des Codes aufrecht zu erhalten.

1. Packages werden eindeutig und sprechend benannt
2. Go-Files werden eindeutig und sprechend benannt
3. Wenn ein Package nur ein Go-File enthält, erhält dieses den Namen seines Packages
4. Vor jedem Package steht ein ein- bis zweizeiliger, beschreibender Kommentar, der die Hauptfunktionalitäten wiedergibt
5. Vor jeder Funktion steht ein ein- bis zweizeiliger, beschreibender Kommentar, der die Hauptfunktionalitäten wiedergibt

6 Getting Started

6.1 Installation von Go

In Abhängigkeit von dem Betriebssystem, gibt es verschiedene Ansätze, die Programmiersprache Go auf einem System zu installieren. Eine Anleitung für die Installation unter Linux, Windows und Mac ist unter dem Link <https://golang.org/doc/install> zu finden. Damit alle Abhängigkeiten des Microservices Warenwirtschaft auch auf dem System bereitstehen, sind diese – über die Ausführung der folgenden Befehlszeile im Root-Verzeichnis des Microservices – zu laden.

Listing 3: Laden der Abhängigkeiten

```
go get ./...
```

6.2 Start des Microservice

Um den Microservice Warenwirtschaft zu starten, ist die folgende Befehlszeile unter dem Root-Verzeichnis des Microservice auszuführen. Anschließend wird der Microservice unter <http://localhost:8080/> bereitgestellt. Zusätzlich wird der Microservice durch Continuous Integration unter der URL <https://stock.pub.warehouse.de/> bereitgestellt. Unter der URL https://stock.pub.warehouse.de/dummy_cart/ steht zudem ein rudimentärer Dummy-Warenkorb bereit, mit dem das Blockieren und Freigeben von Waren für den bereitgestellt Microservice getestet werden kann.

Listing 4: Start des Go-Microservice

```
go run main.go
```

6.3 Start des Monolithen

Der angepasste Monolith wird entsprechend der Anleitung unter <https://gitlab.com/matthiasstock/monolith> gestartet.

A Änderungen am Monolithen

```
From 0754b05118df8af98963428fe32baf7a81920cb7 Mon Sep 17 00:00:00 2001
```

```
Date: Thu, 22 Jun 2017 20:15:07 +0200
```

```
Subject: [PATCH] [TASK] patch for microservice stock
```

```
---
.../monolith/domain/DataTransferObjectFactory.java      |  2 ++
.../de/mstock/monolith/domain/ProductRepository.java    |  5 +++++
.../java/de/mstock/monolith/service/ShopService.java   | 13 ++++++++
.../java/de/mstock/monolith/web/HomepageController.java | 17
+++++++
.../java/de/mstock/monolith/web/ProductController.java | 15
+++++++
src/main/java/de/mstock/monolith/web/ProductDTO.java    | 10 ++++++++
src/main/resources/templates/product.html                |  4 ++++
7 files changed, 64 insertions(+), 2 deletions(-)
```

```
diff --git a/src/main/java/de/mstock/monolith/domain/
    DataTransferObjectFactory.java b/src/main/java/de/mstock/monolith/domain
    /DataTransferObjectFactory.java
```

```
index 914d1ae..b026020 100644
```

```
--- a/src/main/java/de/mstock/monolith/domain/DataTransferObjectFactory.
    java
```

```
+++ b/src/main/java/de/mstock/monolith/domain/DataTransferObjectFactory.
    java
```

```
@@ -86,6 +86,8 @@ private ProductDTO createProductWithoutReviewsDTO(Product
    product, Locale locale
```

```
    ProductI18n i18n = product.getI18n().get(locale.getLanguage());
```

```
    String price = numberFormat.format(i18n.getPrice());
```

```
    ProductDTO productDTO = new ProductDTO();
```

```
+ // Addition: productDTO.setID()
```

```
+ productDTO.setId(product.getId());
```

```
    productDTO.setItemNumber(product.getItemNumber());
```

```
    productDTO.setUnit(product.getUnit());
```

```
    productDTO.setName(i18n.getName());
```

```
diff --git a/src/main/java/de/mstock/monolith/domain/ProductRepository.java
    b/src/main/java/de/mstock/monolith/domain/ProductRepository.java
```

```
index e811855..d72fb53 100644
```

```
--- a/src/main/java/de/mstock/monolith/domain/ProductRepository.java
```

```
+++ b/src/main/java/de/mstock/monolith/domain/ProductRepository.java
```

```
@@ -1,5 +1,7 @@
```

```
package de.mstock.monolith.domain;
```

```
+import java.util.List;
```

```
+
```

```
import org.springframework.data.jpa.repository.Query;
```

```
import org.springframework.data.repository.Repository;
```



```

@@ -9,4 +11,7 @@
     + "where key(i) = ?1 and lower(i.name) = ?2")
     Product findByI18nName(String language, String name);

+   @Query("select p from Product p")
+   List<Product> findAll();
+
+   }
diff --git a/src/main/java/de/mstock/monolith/service/ShopService.java b/
    src/main/java/de/mstock/monolith/service/ShopService.java
index c230e45..c96f266 100644
--- a/src/main/java/de/mstock/monolith/service/ShopService.java
+++ b/src/main/java/de/mstock/monolith/service/ShopService.java
@@ -59,6 +59,19 @@
     }

    /**
+   * Gets all products in the current language.
+   *
+   * @return A simplified Data Transfer Object.
+   */
+   public List<ProductDTO> getAllProducts(Locale locale) {
+       String language = locale.getLanguage();
+       List<Product> products = productRepository.findAll();
+       List<ProductDTO> productsDTO =
+           dtoFactory.createProductWithoutReviewsDTOs(products, locale);
+       return Collections.unmodifiableList(productsDTO);
+   }
+
+   /**
    * Gets a product in the current language.
    *
    * @return A simplified Data Transfer Object.
diff --git a/src/main/java/de/mstock/monolith/web/HomepageController.java b
    /src/main/java/de/mstock/monolith/web/HomepageController.java
index 90a0acc..4395a42 100644
--- a/src/main/java/de/mstock/monolith/web/HomepageController.java
+++ b/src/main/java/de/mstock/monolith/web/HomepageController.java
@@ -17,11 +17,24 @@
    @Autowired
    private ShopService shopService;
+   // Addition: contant with the address of the stock microservice
+   adminfrontend
+   private final String STOCKADMINFRONTENDTEMPLATE = "https://stock.pub.
+       warehost.de/index.html";

    /**
-   * Homepage
-   *

```

```

+ * Redirect to stock admin frontend
+ *
+ * @param model Template model
+ * @return The constant template name for the stock admin frontend.
+ */
+ @RequestMapping(value = "/stockadmin", method = RequestMethod.GET)
+ public String redirect(Model model) {
+     return "redirect:" + this.STOCKADMINFRONTENDTEMPLATE;
+ }
+
+ /**
+ * Homepage
+ *
+ * @param model Template model
+ * @param locale Current locale
+ * @return The template's name.
+ */
diff --git a/src/main/java/de/mstock/monolith/web/ProductController.java b/
    src/main/java/de/mstock/monolith/web/ProductController.java
index 52f1ed3..5fbcacf 100644
--- a/src/main/java/de/mstock/monolith/web/ProductController.java
+++ b/src/main/java/de/mstock/monolith/web/ProductController.java
@@ -1,6 +1,7 @@
    package de.mstock.monolith.web;

    import java.util.Locale;
+import java.util.List;

    import javax.validation.Valid;

@@ -11,6 +12,7 @@
    import org.springframework.web.bind.annotation.PathVariable;
    import org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RequestMethod;
+import org.springframework.web.bind.annotation.ResponseBody;

    import de.mstock.monolith.service.ReviewService;
    import de.mstock.monolith.service.ShopService;
@@ -66,4 +68,17 @@ public String post(@Valid ReviewForm reviewForm,
        BindingResult bindingResult,
        model.addAttribute("product", shopService.getProduct(locale,
            prettyUrlFragment));
        return TEMPLATE;
    }
+
+ @RequestMapping(value = "/products/{prettyUrlFragment:[\\w-]+}.json",
+     method = RequestMethod.GET)
+ @ResponseBody
+ public ProductDTO productJson(@PathVariable String prettyUrlFragment,
+     Locale locale) {

```

```
+     return shopService.getProduct(locale, prettyUrlFragment);
+ }
+
+ @RequestMapping(value = "/products.json", method = RequestMethod.GET)
+ @ResponseBody
+ public List<ProductDTO> allProductJson(Locale locale) {
+     return shopService.getAllProducts(locale);
+ }
+
+ }
diff --git a/src/main/java/de/mstock/monolith/web/ProductDTO.java b/src/
    main/java/de/mstock/monolith/web/ProductDTO.java
index a5b76b6..a7d3245 100644
--- a/src/main/java/de/mstock/monolith/web/ProductDTO.java
+++ b/src/main/java/de/mstock/monolith/web/ProductDTO.java
@@ -6,6 +6,7 @@

    public class ProductDTO {

+ private int id;
    private String itemNumber;
    private ProductWeightUnit unit;
    private String name;
@@ -14,6 +15,15 @@
    private String description;
    private List<ReviewDTO> reviews;

+ // Addition: int id, getId() and setID()
+ public int getId() {
+     return id;
+ }
+
+ public void setId(int id) {
+     this.id = id;
+ }
+
    public String getItemNumber() {
        return itemNumber;
    }
diff --git a/src/main/resources/templates/product.html b/src/main/resources
    /templates/product.html
index 0d7bc31..9b2d0bf 100644
--- a/src/main/resources/templates/product.html
+++ b/src/main/resources/templates/product.html
@@ -31,6 +31,10 @@
        </div>
        <div class="col-md-8">
            <h2 th:text="{product.name}">Product Name</h2>
+

```

```
+      <!-- Addition: traffic light food labeling system of the stock
+      microservice -->
+      
+
+      <p class="text-info text-uppercase" th:text="{product.price
+      }">0,00 Euro</p>
+      <p class="lead" th:text="{product.description}">Description.</p>
+      <div th:replace="fragments/reviews :: reviews"></div>
```