HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

COMPASS SECURITY

# Glockenemil on Steroids

## Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017


| | |
|---|---|
| Autor(en): | Patrick Steinhäusl |
| Betreuer: | Cyrill Brunschwiler |
| Projektpartner: | Compass Security Network Computing AG |

# 1. Inhaltsverzeichnis

# 2. Abstract

Die Firma Compass Security Network Computing AG nutzt für ihre Security Schulungen unter anderem die Plattform “Glocken Emil Shop”. Hierbei handelt es sich um keinen gewöhnlichen Webshop, denn dieser weist gezielt Sicherheitsschwachstellen auf. Diese Schwachstellen müssen durch die Anwender der Schulungen gefunden und ausgenutzt werden.

In den letzten Jahren gab es eine grosse Wandlung in den Open Web Application Security Project (OWASP) Top Ten Liste, sowie in den Technologien, welche im Web Engineering eingesetzt werden. Es werden vermehrt JavaScript basierte Frameworks eingesetzt, welche wiederum neue Schwachstellen aufweisen und durch die “Glocken Emil Shop” Plattform nicht mehr abgedeckt sind.

Aus diesem Grund war das primäre Ziel dieser Arbeit die Überarbeitung des Webshops mittels neuer Technologien inklusive der Implementation gezielter Schwachstellen. Die Plattform soll in Zukunft den Anwendern der Schulung einen realistischen Einblick in einen heutzutage angreifbaren Webshop bieten.

Das Projekt hat gezeigt, dass die neuen Technologien einen hohen Security Standard hatten und nicht von Grund auf angreifbar waren. Die Implementierung der Schwachstellen erwies sich aufgrund des hohen Security Standards der Technologien als schwierig und vor allem zeitintensiv.

# 3. Management Summary

Der “Glocken Emil Shop” muss gemäss Auftraggeber zwingend überarbeitet werden, um auch in Zukunft den Security Schulungen von der Compass Security Network Computing AG, den nötigen Nutzen als Schulungsplattform zu bringen. Die implementierten Schwachstellen haben sich in den letzten Jahren konzeptionell kaum geändert, desto mehr Veränderungen gab es in den Technologien.

In den Modulen CAS Front End Engineering und Web Engineering werden die aktuellsten Frameworks betrachtet. Die zwei JavaScript Frameworks NodeJS / ExpressJS und AngularJS stehen stark im Fokus dieser Vorlesungen. Aus diesem Grund wurde der “Glocken Emil Shop” durch ein AngularJS Frontend mit einem NodeJS/ExpressJS Backend umgesetzt. Das Backend greift auf eine MongoDB zu, welche die Daten als Dokumente hält.

Somit konnten gezielt Schwachstellen implementiert werden, welche in den aktuellen OWASP Top Ten aufgelistet sind oder spezifisch durch den Einsatz der neuen Technologien ermöglicht wurde.

Der neue Webshop bietet somit einen realistischen Einblick in einen zeitgemässen Onlineshop und die Möglichkeit weitere Schwachstellen einzubinden, welche noch nicht umgesetzt wurden.

# 4. Technischer Bericht der Arbeit

Dieses Kapitel soll einen Überblick über die geleistete Arbeit zeigen. Hierfür wird in der Einleitung die Ausgangslage der Arbeit genauer betrachtet. Anschliessend wird das Ziel der Arbeit beschrieben und zu guter Letzt die Umsetzung erklärt.

## Einleitung und Übersicht

Die Compass Security Network Computing AG setzt zurzeit für ihre Sicherheitsschulungen einen Webshop namens “Glocken Emil Shop” ein. Der Webshop weist gezielt Schwachstellen auf, welche aufgrund der Technologien vorhanden sind und andererseits solche, welche die OWASP Top Ten abdecken. Die Teilnehmer der Schulungen müssen diese Schwachstellen gezielt finden und diese ausnutzen.

## Technologien

Der alte “Glocken Emil Shop” besteht aus verschiedenen virtualisierten Servern und nutzt die nachfolgend beschriebenen Technologien.

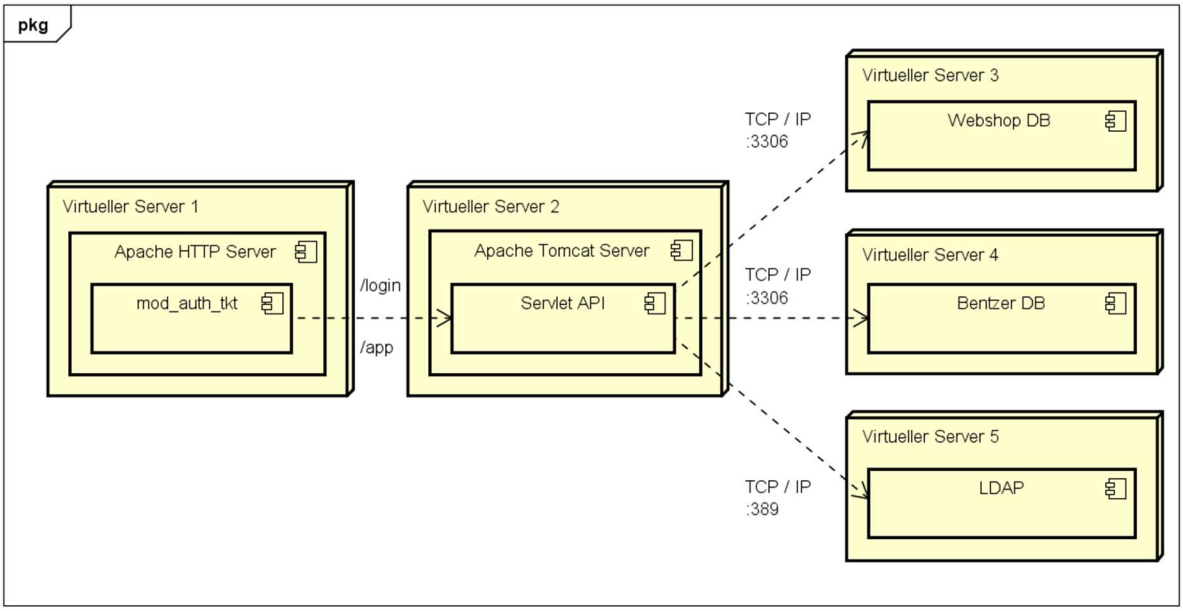


*Abbildung 1 Deployment-Diagramm vor Umsetzung*

### Apache Hypertext Transfer Protocol (HTTP) Server

Der Apache HTTP-Server ist mit dem Single-sign-on Authentifizierungsmodul mod_auth_tkt ausgestattet. Falls keine gültige Authentifizierung vorhanden ist, wird der Benutzer auf die Login Seite des Apache Tomcat Servers verwiesen und hat keinen Zugriff auf die App. Nach erfolgreicher Authentifizierung wird der Zugriff auf die App ermöglicht.

### Tomcat Server

Der Apache Tomcat Server stellt die Java Servlets zur Verfügung, welche als Frames dargestellt werden. Zudem wird im Hintergrund auf verschiedenen Datenbanken zugegriffen.

### MySQL Servers

Es werden zwei MySQL Datenbanken für den “Glocken Emil Shop” genötigt. Die Webshop-Datenbank beinhaltet zum Beispiel die Produkte, welche im Shop gekauft werden können. Die Benutzer-Datenbank enthält alle registrierten Benutzer für die Anwendung.

### Lightweight Directory Access Protocol (LDAP)

Der LDAP Server beinhaltet alle Benutzer und zugleich auch deren Benutzerinformationen. Er wird für die Authentifizierung benötigt. Aktuell werden die Benutzerdaten im LDAP und in der Benutzerdatenbank gepflegt, und sind somit dupliziert vorhanden.

## Benutzeroberfläche

Die Benutzeroberfläche des “Glocken Emil Shop” ist ein schlichter Webshop mit der Möglichkeit, Kuhglocken zu bestellen.

Für die Bestellung der Artikel ist eine Authentifizierung notwendig, welche mittels Benutzernamen und Passwort durchgeführt wird.

Darüber hinaus kann das Benutzerprofil bearbeitet werden, was auch die Änderung der Adresse und der Kreditkarteninformationen beinhaltet.


*Abbildung 2 Glocken Emil Shop vor Umsetzung*

## Schwachstellen

Der Webshop wurde vor einigen Jahren entwickelt. Aus diesem Grund deckt dieser die Schwachstellenliste der OWASP Top Ten aus dem Jahre 2007 ab.

| | |
|---|---|
| Cross Site Scripting | XSS-Fehler treten immer dann auf, wenn eine Anwendung vom Benutzer bereitgestellte Daten entgegennimmt und sie an einen Webbrowser sendet, ohne zuerst diesen Inhalt zu validieren oder zu codieren. XSS ermöglicht es Angreifern, Skripte im Browser des Opfers auszuführen, welche Benutzersitzungen stehlen, Websites verunstalten und Würmer usw. in das System einführen können. |
| Injection Flaws | Injection-Fehler, insbesondere SQL-Injection, sind in Web-Anwendungen üblich. Die Injection tritt auf, wenn vom Benutzer bereitgestellte Daten als Teil eines Befehls oder einer Abfrage an einen Interpreter gesendet werden. Die feindlichen Daten des Angreifers führen dazu, dass der Interpreter unbeabsichtigte Befehle ausführt oder Daten ändert. |
| Malicious File Execution | Code, der anfällig für Remote File Inclusion ist, erlaubt es Angreifern, feindlichen Code und Daten einzuschliessen, was zu verheerenden Angriffen führt, wie zum Beispiel der totalen Server-Kompromittierung. Solche bösartige Angriffe betreffen PHP, XML und jedes Framework, das Dateinamen oder Dateien von Benutzern akzeptiert. |
| Insecure Direct Object Reference | Eine direkte Objektreferenz tritt auf, wenn ein Entwickler einen Verweis auf ein internes Objekt, z. B. eine Datei, ein Verzeichnis, einen Datenbankeintrag oder einen Schlüssel, als Uniform Resource Locator (URL)- oder Formular-Parameter verfügbar macht. Angreifer können diese Verweise manipulieren, um ohne Autorisierung auf andere Objekte zuzugreifen. |
| Cross Site Request Forgery (CSRF) | Ein CSRF-Angriff zwingt den Browser eines angemeldeten Opfers, eine vorab authentifizierte Anfrage an eine anfällige Web-Anwendung zu senden, die dann den Browser des Opfers dazu zwingt, eine feindliche Aktion zum Nutzen des Angreifers durchzuführen. CSRF kann so mächtig sein wie die Webanwendung, die es angreift. |
| Information Leakage and Improper Error Handling | Anwendungen können unbeabsichtigt Informationen über ihre Konfiguration, interne Abläufe oder Verletzungen der Privatsphäre durch eine Vielzahl von Anwendungsproblemen verlieren. Angreifer nutzen diese Schwäche, um vertrauliche Daten zu stehlen oder gravierende Angriffe durchzuführen. |
| Broken Authentication and Session Management | Konto-Anmeldeinformationen und Sitzungstoken sind oft nicht ordnungsgemäss geschützt. Angreifer kompromittieren Kennwörter, Schlüssel oder Authentifizierungstoken, um die Identität anderer Benutzer anzunehmen. |
| Insecure Cryptographic Storage | Webanwendungen verwenden kryptographische Funktionen selten zum Schutz von Daten und Anmelde-Informationen. Angreifer verwenden schwach geschützte Daten, um Identitätsdiebstahl und andere Straftaten wie |

| | |
|---|---|
| | Kreditkartenbetrug durchzuführen. |
| Insecure Communications | Anwendungen verschlüsseln den Netzwerkverkehr häufig nicht, wenn sensible Daten übertragen werden. |
| Failure to Restrict URL Access | Häufig schützt eine Anwendung nur sensible Funktionen, indem die Anzeige von Links oder URLs für nicht autorisierte Benutzer verhindert wird. Angreifer können diese Schwachstelle verwenden, um auf nicht autorisierte Vorgänge zuzugreifen und sie auszuführen, indem sie direkt auf diese URLs zugreifen. |

*Tabelle 1 Liste der OWASP Top Ten 2007 (OWASP Foundation)*

# 5. Zielsetzung

In den letzten Jahren kamen viele neue Webframeworks in Umlauf, welche auf JavaScript basieren. Der Tumult um diese Frameworks ist mittlerweile so gross, dass die Compass Security Network AG sich entschieden hat, den “Glocken Emil Shop” zu aktualisieren und mit den neuen Technologien zu versehen.

Dies soll es ermöglichen, dass nicht nur OWASP Top Ten Schwachstellen in den Schulungen gezeigt werden können, sondern auch diejenigen, welche die neuen Technologien selbst betreffen.

Der Webshop soll somit mit den aktuell begehrtesten Technologien umgesetzt werden und die weiter unten aufgelisteten Schwachstellen aufweisen, welche nachfolgend beschrieben werden.

## Technologien

### AngularJS

Das Frontend soll mittels AngularJS, einem auf JavaScript basierenden Framework, umgesetzt werden. Aktuell hat es ein grosses Ansehen in der Web-Community. Viele topaktuelle Webseiten werden mit dieser Technologie umgesetzt, da AngularJS eine kleine Lernkurve hat und da es sich besonders für interaktive Apps eignet.

### NodeJS und ExpressJS

Für das Application Programming Interface (API) soll NodeJS mit darauf aufbauendem ExpressJS eingesetzt werden. Zurzeit ist ein grosser Hype, welcher mit diesen beiden Frameworks zusammenhängt, zu beobachten. Auch im Studium sind wir vermehrt mit diesem Framework in Kontakt gekommen und haben einen Einblick in diese Technologie erhalten, welcher nun in dieser Arbeit stark vertieft wurde.

### MongoDB

MongoDB ist eine dokumentenorientierte NoSQL-Datenbank. Durch die dokumentenorientierte Datenhaltung kann diese Datenbank Sammlungen von JSON ähnlichen Dokumenten verwalten. Die Daten werden in komplexe Hierarchien unterteilt und bleiben dabei jederzeit indizierbar und aufrufbar. Aus diesem Grunde ist es der Anwendung möglich, Daten auf natürliche Weise zu modifizieren. (Wikipedia)

Der Einsatz von MongoDB ist zurzeit sehr verbreitet, vor allem in Zusammenspiel mit NodeJS und ExpressJS.

### Docker

Docker bietet die Möglichkeit, Anwendungen in sogenannten Containers bereits zustellen. Container können aufeinander aufbauen und ermöglichen die Kommunikation zwischen diesen.

Ein Vorteil zu virtualisierten Maschinen ist, dass die Container zum Beispiel nicht das ganze Betriebssystem beinhalten müssen, sondern nur die relevanten Daten. Des Weiteren bietet es im Web Entwicklungsbereich den Vorteil, dass die Serverumgebungen exakt auf einem lokalen Client aufgebaut und später wieder einfacher ausgelagert werden können.

Das Docker Image soll in dieser Studienarbeit das Frontend und das Backend beinhalten.

## Benutzeroberfläche

Die Benutzeroberfläche des “Glocken Emil Shop” wird mittels AngularJS realisiert. Die neue Oberfläche soll modern sein, jedoch zugleich Ähnlichkeiten mit dem alten Shop aufweisen.

Nachfolgend ist die Produktübersicht des Shops angefügt. Die vollständigen Mockups sind im Anhang C ersichtlich.

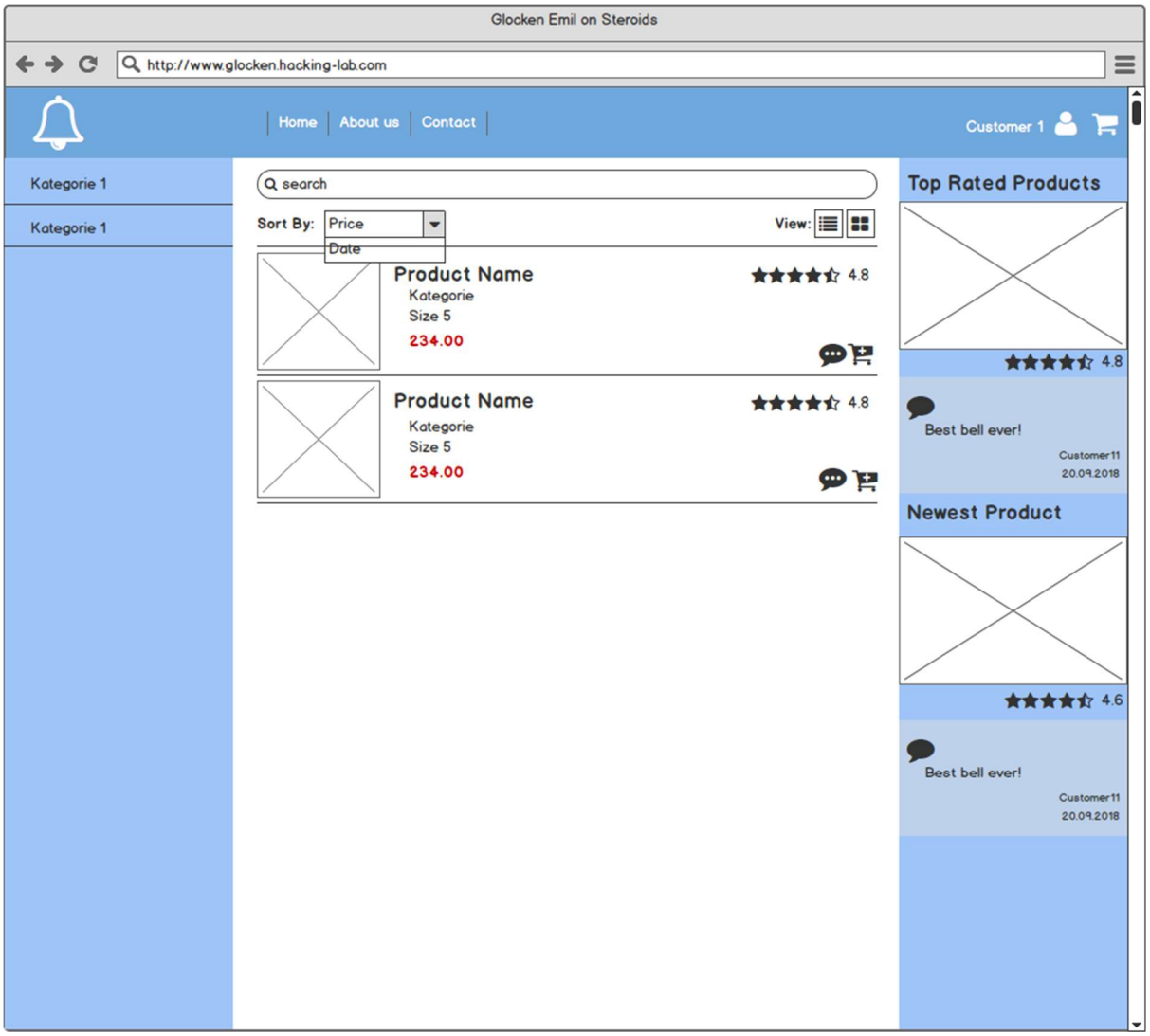


*Abbildung 3 Mockups der umgesetzten Lösung*

# Schwachstellen

Die Schwachstellen des alten “Glocken Emil Shop” waren auf die OWASP Top Ten und auf die davor eingesetzten Technologien zurückzuführen. Im neuen Shop sollen nachfolgende Schwachstellen implementiert werden.

## NoSQL Injection

Datenbanken, welche mittels NoSQL aufgebaut sind, bieten geringere Konsistenz-Einschränkungen als SQL-Datenbanken. Aufgrund der wenigen relationalen Einschränkungen und Konsistenzprüfungen haben NoSQL-Datenbanken häufig Vorteile in der Leistung und Skalierung. Diese Datenbanken sind anfällig für Angriffe, auch wenn diese nicht den SQL-Syntax verwenden. Bei NoSQL Injection Angriffe sind die Auswirkungen meist grösser als bei gewöhnlichen SQL Injection, da die NoSQL in einer prozeduralen Sprache ausgeführt wird.

NoSQL Injection Angriffe können in verschiedenen Bereichen einer Anwendung ausgeführt werden, im Gegensatz zu der SQL Injection. Wenn eine SQL Injection in der Datenbank-Engine ausgeführt wird, können NoSQL-Varianten je nach verwendeter NoSQL-API und Datenmodell innerhalb der Anwendungsschicht oder der Datenbankebene ausgeführt werden. In der Regel werden NoSQL-Injection-Angriffe ausgeführt, indem die Eingabe analysiert, ausgewertet oder in einen NoSQL-API Aufruf verkettet wird. (OWASP Foundation)

## Hidden Functionality

Im API existieren gewisse Funktionalitäten, welche nicht von Aussen aufgerufen werden sollten, oder vergessen wurden zu entfernen, da diese nicht oder nicht mehr benötigt werden. Trotzdem ist es möglich, diese Funktionalitäten von Aussen aufzurufen und somit sensible Daten zu erlangen oder das System zu schädigen.

## Unprotected REST API

Diese Schwachstelle taucht auf, wenn ein API Aufruf nicht genügend oder gar nicht geschützt ist, obwohl dieses nur für autorisierte Benutzer zur Verfügung stehen sollte.

## *Scalable Vector Graphic* Injection

Die SVG-Injection basiert auf skalierbaren Vektorgrafiken, welche mit JavaScript Code ergänzt werden können. Falls eine solche SVG-Datei auf das Zielsystem hochgeladen werden kann, ohne den JavaScript Code zu unterbinden, ermöglicht es vollen Zugriff auf den Inhalt der Seite.

## Direct Object References

Dieser Angriff ist möglich, wenn Ressourcen über URL-Parameter direkt mittels eindeutiger Identifikation abgefragt werden können.

Wird zum Beispiel folgende URL aufgerufen:

http://example.com/api/account/j2l3j42l3j5l235loeqoivcxovixpcvoio

Und das API führt folgendes aus:

```
let accountId = req.params.accountId;
Account.find( { id: accountId } , function(err, acc) {
  if(err) return res.send(err);
  res.send(acc);
});
```

*Abbildung 4 Direct Object Reference Schwachstellen Beispiel*

In diesem Beispiel findet keine Authentifizierung statt, somit wäre es möglich, über eine Brute-Force-Attacke alle User Informationen zu erlangen.

## Cross Site Scripting (XSS)

Der XSS-Angriff ist eine Art Injection. Bei diesem Angriff wird ein schädliches Skript in eine ansonsten gutartige und vertrauenswürdige Website eingefügt. Dieses Script wird über den Server an einen anderen Anwender geschickt und bei diesem ausgeführt. Diese Attacke findet vor allem eine Angriffsfläche in Webanwendungen, welche Benutzereingaben vor dem Anzeigen nicht validieren oder zu codieren. Da dieses eingefügte Script nun vom vertrauenswürdigen Server kommt, kann der Browser des Opfers nicht feststellen, dass es sich um schädlichen Code handelt. Dieser wird beim Opfer somit ausgeführt und kann auf jegliche Cookies, Sitzungstoken oder andere vertrauliche Daten zugreifen. (OWASP Foundation)

# 6. Umsetzung

## Domain Model

Der alte “Glocken Emil Shop” hielt die Daten in verschiedenen Datenbanken und zudem teilweise dupliziert. Daher musste das Domain Model überarbeitet werden und sieht nun wie folgt aus:

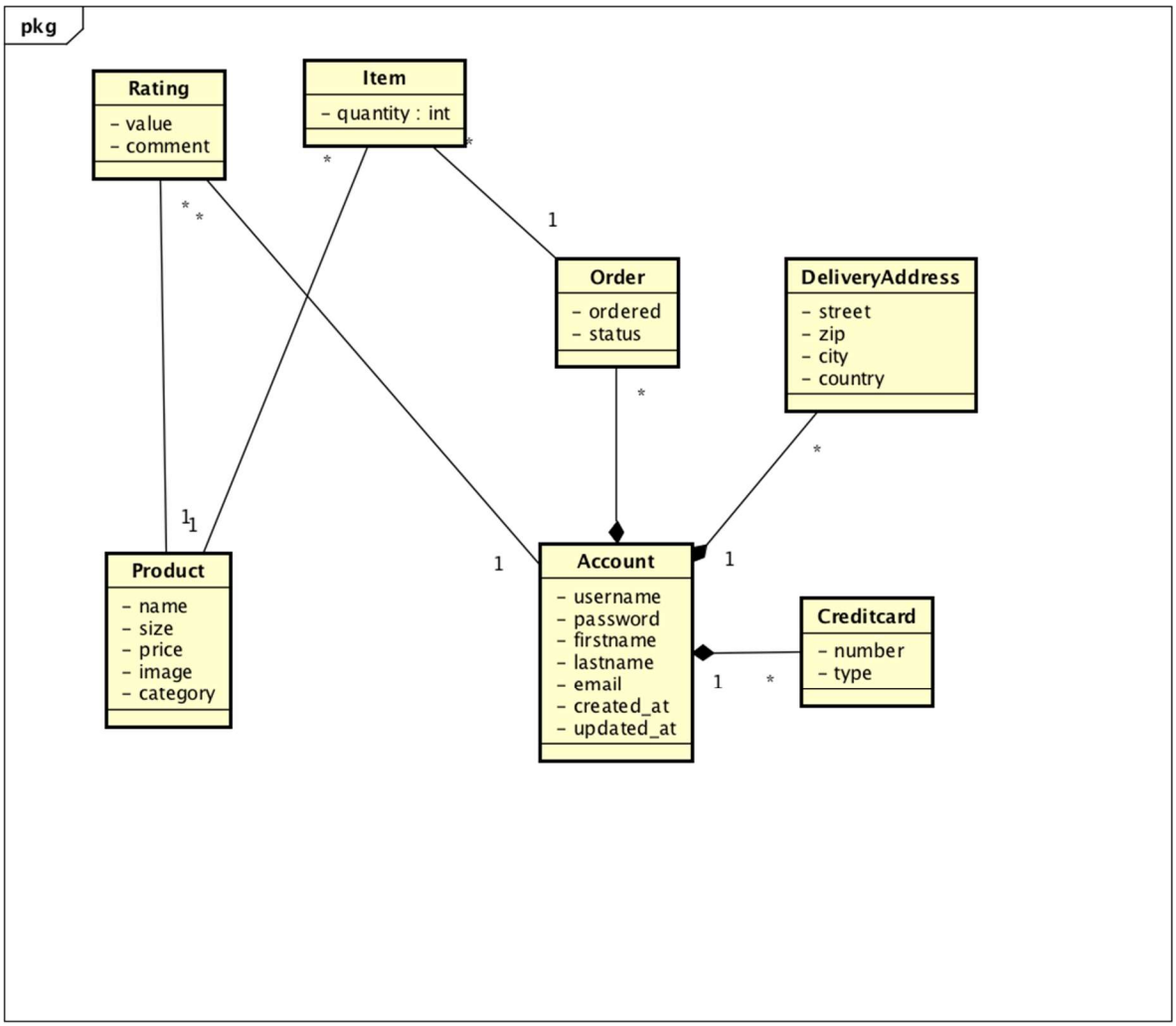


*Abbildung 5 Domain Modell der umgesetzten Lösung*

### Konto

Das Konto hält alle relevanten Benutzerinformationen, nämlich Benutzername, Passwort, Vorname, Nachname und Email-Adresse.

Ein Konto kann mehrere Kreditkarten, Lieferadressen oder Buchungen erstellen.

### Kreditkarte

Die Kreditkarte ist durch die Kartennummer und einen Typ gekennzeichnet.

### Lieferadresse

Die Lieferadresse hält die Felder Strasse, Postleitzahl, Stadt und Land.

### Buchung

Die Buchung enthält einen Status und das Datum inklusive Uhrzeit der Bestellung. Eine Buchung enthält einen oder mehrere Artikel.

### Artikel

Der Artikel enthält eine Referenz auf ein bestimmtes Produkt mit der Anzahl, wie oft dieses Produkt bestellt wurde.

### Produkt

Das Produkt entspricht den Glocken, welche im Shop als Produkte ersichtlich sind. Es ist durch einen Namen, Grösse, Preis, Bild und Kategorie gekennzeichnet. Jedes Produkt kann von einem Konto bewertet werden.

### Bewertung

Die Bewertung eines Produkts beinhaltet die Referenz auf das Produkt und das Konto, welches die Bewertung abgeschickt hat. Zudem enthält es den Bewertungswert und einen Kommentar des Benutzers.

## Deployment

In diesem Bereich wird auf das Deployment näher eingegangen.

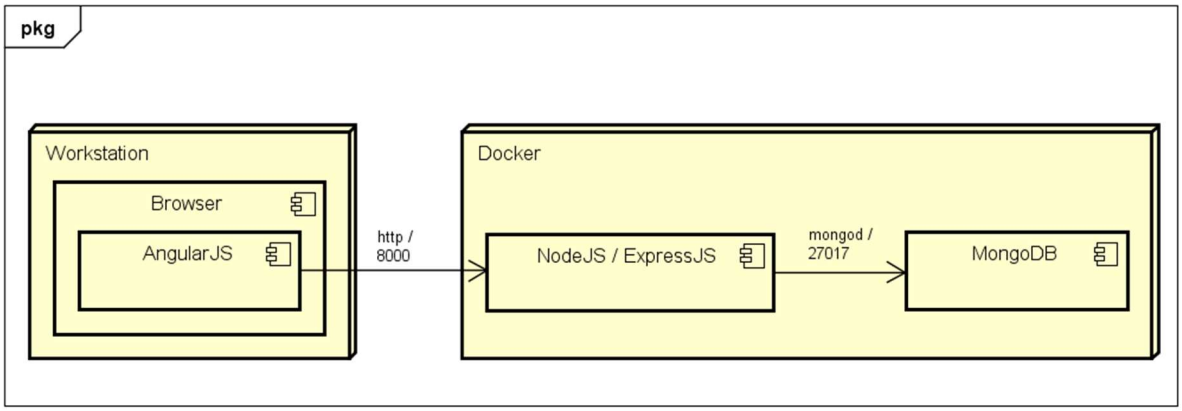


*Abbildung 6 Deployment Modell der umgesetzten Lösung*

### Workstation

Die Workstation beinhaltet die Browser Komponente, in welcher das AngularJS Frontend läuft.

### AngularJS

#### Views

Die Hypertext Markup Language (HTML) Views beinhalten die verschiedenen Views der einzelnen Komponenten und haben ihren Controller gemappt.

#### Controllers

Der Controller beinhaltet den Scope der View, das heisst bei Änderungen der Daten im Controller werden die Daten der View angepasst. Die aktualisierten Daten erhält er über den Service.

#### Services

Der Service kommuniziert mittels HTTP-Request mit dem REST API und gibt die Daten zurück an den Controller. Diese Verbindung findet über den Port 3000 statt.

## Docker

Das Docker-Image beinhaltet die Komponenten NodesJS / ExpressJS und MongoDB.

### NodeJS / ExpressJS

#### App.js

Zur Initialisierung des Servers wird das app.js File geladen, welches die Module definiert und inkludiert. Zudem werden die Route Files mit ihrem URL in der App eingebunden. Zusätzlich werden die Berechtigungen für den Aufruf der Routen festgehalten.

```
app.use(express.static(__dirname + '/public/app'));
app.use('/bower_components', express.static(__dirname + '/bower_components'));
app.use('/product-images', express.static(__dirname + '/assets/product-images'));
app.use('/post-images', express.static(__dirname + '/assets/post-images'));
app.use('/slider-images', express.static(__dirname + '/assets/slider-images'));
app.use('/favicon.ico', express.static(__dirname + '/assets/favicon.ico'));
app.use('/api/auth', auth);
// Injection Code Start - Unprotected REST API
app.use('/api/retailer', retailer);
// Injection Code End

app.use(expressJwt(GlobalConfig.auth.validateOptions).unless(GlobalConfig.auth.unprotectedRoutes));

app.use('/api/account', account);
app.use('/api/creditCard', creditCard);
app.use('/api/deliveryAddress', deliveryAddress);
app.use('/api/order', order);
app.use('/api/product', product);
app.use('/api/post', post);
```

*Abbildung 7 Die App-Datei des ExpressJS Backends*

#### Routes

Die Routes definieren, über welche URLs der Server erreichbar ist und gibt die Request an den entsprechenden Controller weiter.

*Produkt Route*

Als Beispiel hier die Routen, welche für das Produkt definiert wurden.

```
const express = require('express');
const router = express.Router();
const productController = require('../controllers/product');


router.get('/category', productController.getCategories);
router.get('/category/:categoryId', productController.getByCategoryId);
router.get('/searchValue/:searchValue', productController.getBySearchValue);
router.get('/toprated', productController.getTopRated);
router.get('/latest', productController.getLatest);
router.post('/rating', productController.insertRating);
router.get('/', productController.get);
router.get('/:productId', productController.getById);

module.exports = router;
```

*Abbildung 8 Produkt-Routen*

## Controllers

Der Controller nimmt die Request entgegen und bearbeitet den Request mit Hilfe des Services.

*Produkt Controller*

Auszug des Produkt Controllers, um den Ablauf zu veranschaulichen.

```
const ProductService = require('../services/product');

function get(req, res) {
  ProductService.get((error, result) => {
    if(error) return res.json(error);
    res.json(result);
  });
}

function getById(req, res) {
  let productId = req.params.productId;
  ProductService.getById(productId, (error, result) => {
    if(error) return res.json(error);
    res.json(result);
  });
}

function getTopRated(req, res) {
  ProductService.getTopRated((error, result) => {
    if(error) return res.json(error);
    res.json(result);
  });
}

function getLatest(req, res) {
  ProductService.getLatest((error, result) => {
    if(error) return res.json(error);
    res.json(result);
  });
}

function getByCategoryId(req, res) {
  let categoryId = req.params.categoryId;
```

*Abbildung 9 Produkt Controller*

### Services

Die Services machen die entsprechenden Datenbankabfragen über die definierten Models.

#### *Produkt Service*

Hier ein Beispiel des Produkt Services, wie er die Daten vom Modell abfragt.

```
const Product = require('../models/product').Product;
const ResponseUtil = require('../utils/response');

function get(callback) {
  Product.find({}, function(error, result) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());
    result = { 'products' : result };
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}

function getById(productId, callback) {
  if(!productId) return callback(ResponseUtil.createNotFoundResponse());
  Product.findById(productId, function(error, result) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());
    result = { 'product' : result };
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}

function getTopRated(callback) {
  Product.find({}, function(error, result) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());
    result = { 'products' : result };
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}

function getLatest(callback) {
  Product.find({}, function(error, result) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());
    result = { 'products' : result };
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}
```

*Abbildung 10 Produkt Service*

### Models

Die Models wurden anhand des erstellten Domain Models erstellt. Diese sind mittels Mongoose Modul im ExpressJS definiert und können durch die Services abgefragt werden. Die Kommunikation findet über den Port 27017 statt, welcher der Standard Port für MongoDB ist.

#### *Konto*

Das Konto beinhaltet den Benutzernamen, Passwort, Vorname, Nachname und E-Mail-Adresse. Alle Felder sind zwingend und können nicht weggelassen werden.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const cryptoUtil = require('../utils/crypt');

let accountSchema = new Schema({
  username: { type: String, required: [true, 'Username is required'] },
  password: { type: String, required: [true, 'Password is required'] },
  firstname: { type: String, required: [true, 'Firstname is required'] },
  lastname: { type: String, required: [true, 'Lastname is required'] },
  email: { type: String, required: [true, 'Email is required'] }
}, {
  timestamps: {}
});

accountSchema.pre('save', function(callback) {
  let account = this;
  if (!account.isModified('password')) return callback();
  account.password = cryptoUtil.hashPwd(account.password);
  return callback();
});

accountSchema.path('password').validate(function(password, callback) {
  let regex = new RegExp(/^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{7,}$/);
  return callback(regex.test(password), 'Password: Require 7 characters, at least 1 letter and one number!');
}, 'An unexpected error occured');

accountSchema.path('email').validate(function(email, callback) {
  let regex = new RegExp(/(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b
  return callback(regex.test(email), 'Email: No valid email address');
}, 'An unexpected error occured');

let Account = mongoose.model('Account', accountSchema);

module.exports = Account;
```

*Abbildung 11 Modell des Kontos*

Das Modell hashed das Passwort mittels SHA 256 Hash Verfahren vor dem Speichern des Dokuments in der MongoDB.

### *Produktkategorie*

Das Produkt beinhaltet eine Kategorie, welche als einzelnes Modell ausgelagert wurde, um es über die ID eindeutig zu identifizieren.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let categorySchema = new Schema({
  name: { type: String, required: true }
}, {
  timestamps: {}
});

let Category = mongoose.model('Category', categorySchema);

module.exports = {
  Category,
  categorySchema
};
```

*Abbildung 12 Modell der Produktkategorie*

### *Kreditkarte*

Das Modell für die Kreditkarte beinhaltet die Nummer, einen Typ und das referenzierten Konto. Mit diesen Daten kann zurückverfolgt werden, wem eine Kreditkarte gehört.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let creditCardSchema = new Schema({
  number: { type: Number, required: [true, 'Number is required'] },
  type: { type: String, required: [true, 'Type is required'] },
  _account: { type: Schema.Types.ObjectId, ref: 'Account'}
}, {
  timestamps: {}
});

creditCardSchema.path('number').validate(function(number, callback) {
  let regex = new RegExp('(^4[0-9]{12}(?:[0-9]{3})?$|^(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}$)');
  return callback(regex.test(number), 'Number: No valid Mastercard or Visa number!');
}, 'An unexpected error occured');

creditCardSchema.path('type').validate(function(number, callback) {
  let regex = new RegExp('Mastercard|Visa');
  return callback(regex.test(number), 'Type: No valid type selected!');
}, 'An unexpected error occured');

let CreditCard = mongoose.model('CreditCard', creditCardSchema);

module.exports = {
  CreditCard,
  creditCardSchema
};
```

*Abbildung 13 Modell der Kreditkarte*

### *Lieferadresse*

Die Lieferadresse enthält die Strasse, Stadt, Postleitzahl, Land und wie auch schon zuvor beim Kreditkarten Model das Konto, zu welchem die Lieferadresse zugehörig ist.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let deliveryAddressSchema = new Schema({
  street: { type: String, required: true },
  city: { type: String, required: true },
  zip: { type: Number, required: true },
  country: { type: String, required: true },
  _account: { type: Schema.Types.ObjectId, ref: 'Account'}
}, {
  timestamps: {}
});

let DeliveryAddress = mongoose.model('DeliveryAddress', deliveryAddressSchema);

module.exports = {
  DeliveryAddress,
  deliveryAddressSchema
};
```

*Abbildung 14 Modell der Lieferadresse*

### *Artikel*

Im Warenkorb können Artikel hinzugefügt werden, welche eine Anzahl und das Produkt beinhaltet.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const productSchema = require('../models/product').productSchema;

let itemSchema = new Schema({
  quantity: { type: Number, required: true},
  product: productSchema
}, {
  timestamps: {}
});

let Item = mongoose.model('Item', itemSchema);

module.exports = {
  Item,
  itemSchema
};
```

*Abbildung 15 Modell des Artikels*

### *Order*

Die Buchung beinhaltet den ganzen Warenkorb, das heisst mehrere Items, Lieferadresse, Zahlungsmethode, Status der Buchung, einen totalen Preis und wiederum die Referenz zum Account, welchem diese Buchung gehört.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const itemSchema = require('../models/item').itemSchema;
const deliveryAddressSchema = require('../models/deliveryAddress').deliveryAddressSchema;
const creditCardSchema = require('../models/creditCard').creditCardSchema;

let orderSchema = new Schema({
  items: [itemSchema],
  deliveryAddress: deliveryAddressSchema,
  payment: {
    type: { type: String, required: true },
    creditCard: creditCardSchema,
  },
  status: { type: String, required: true },
  totalPrice: { type: Number, required: true },
  _account: { type: Schema.Types.ObjectId, ref: 'Account'}
}, {
  timestamps: {}
});

let Order = mongoose.model('Order', orderSchema);

orderSchema.pre('save', function(callback) {
  let order = this;
  if (!order.isModified('bill')) return callback();
  order.bill = false;
  return callback();
});

module.exports = Order;
```

*Abbildung 16 Modell der Buchung*

*Post*

Das Modell Post besteht aus dem Titel, einem Text, dem Pfad zum hochgeladenen Bild und wiederum einer Referenz zum Account, welcher den Post erstellt hat.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let postSchema = new Schema({
  title: { type: String, required: true },
  text: { type: String, required: true },
  image: { type: String, required: true },
  _account: { type: Schema.Types.ObjectId, ref: 'Account'}
}, {
  timestamps: {}
});

let Post = mongoose.model('Post', postSchema);

module.exports = Post;
```

*Abbildung 17 Modell des Posts*

### *Produkt*

Das Produkt ist mit einem Namen, Kategorie, Grösse, Preis und einem Link zu einem Bild versehen. Des Weiteren besitzt das Produkt ein summiertes Rating und die einzelnen Ratings der Benutzer.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const categorySchema = require('../models/category').categorySchema;
const ratingSchema = require('../models/rating').ratingSchema;

let productSchema = new Schema({
  name: { type: String, required: true },
  category: categorySchema,
  size: { type: Number, require: true },
  price: { type: Number, require: true },
  image: { type: String, require: true },
  ratings: [ratingSchema],
  rating: {}
}, {
  timestamps: {}
});

let Product = mongoose.model('Product', productSchema);

module.exports = {
  Product,
  productSchema
};
```

*Abbildung 18 Modell des Produkts*

### *Bewertung*

Das Bewertungsmodell besteht aus einem Wert, Kommentar und dem dazugehörenden Konto.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let ratingSchema = new Schema({
  value: { type: Number, required: true },
  comment: { type: String, required: true },
  _account: { type: Schema.Types.ObjectId, ref: 'Account'}
}, {
  timestamps: {}
});

let Rating = mongoose.model('Rating', ratingSchema);

module.exports = {
  Rating,
  ratingSchema
};
```

*Abbildung 19 Modell der Bewertung*

### Assets

Der Folder Assets beinhaltet statische Files, wie zum Beispiel Bilder. Diese statischen Daten sind dem Frontend als statische Ressource freigegeben.

```
app.use('/bower_components', express.static(__dirname + '/bower_components'));
app.use('/post-images', express.static(__dirname + '/assets/post-images'));
app.use('/slider-images', express.static(__dirname + '/assets/slider-images'));
app.use('/product-images', express.static(__dirname + '/assets/product-images'));
```

*Abbildung 20 Auszug aus der App.js Datei - Statischer Dateien*

### Configs

Es existiert eine globale Konfigurationsdatei, welche an verschiedenen Orten im Code verwendet wird um die Konfiguration zu vereinfachen. In dieser Datei werden die folgenden globalen Werte gelagert:

| | |
|---|---|
| **Server** | Hostname und Port |
| **Crypt** | Hash-Algorithmus und Secret Key |
| **Mongo** | Hostname, Port, Name der Datenbank und Connection String |
| **JWT** | Secret Key |
| **Auth** | **SignOptions** (Gültigkeitsdauer des Tokens, Audience und Issuer)<br>**ValidateOptions** (Wie das Token validiert werden soll) |

| | |
|---|---|
| | **UnprotectedRoutes** (Welche Routen keine Authentifizierung erfordern) |
| **Cors** | **CorsOptions** (Origin und Status-Code bei korrekter Origin) |
| **PostImage** | Upload-Verzeichnis für die Bilder |

*Tabelle 2 Liste der möglichen Konfigurationen am Backend*

### Utils

Es existieren die nachfolgend aufgelisteten vier Werkzeuge, welche in verschiedene Files immer wieder verwendet werden.

#### *Crypt*

Das Crypt Werkzeug wird zum Hashen und Verschlüsseln von beispielsweise den Passwörtern benötigt.

#### *Log*

Das Log Werkzeug erzeugt eine Logmeldung in einem Textfile und gibt die Nachricht zusätzlich an die Konsole weiter. Das Logfile befindet sich im Verzeichnis “/app/logs”.

#### *Mongo*

Das Mongo Werkzeug stellt die Datenbankverbindung her.

#### *Response*

Das Response Werkzeug erzeugt für Fehler und gültige Anfragen an den Server einen einheitlichen Response.

### MongoDB

Die Datenhaltung findet in der MongoDB statt, welche die Resultate der Abfrage an die NodeJS / ExpressJS Komponente zurückliefert.

## Benutzeroberfläche

Die Benutzeroberfläche wurde gemäss Vorlage der Mockups aus Anhang E erstellt.

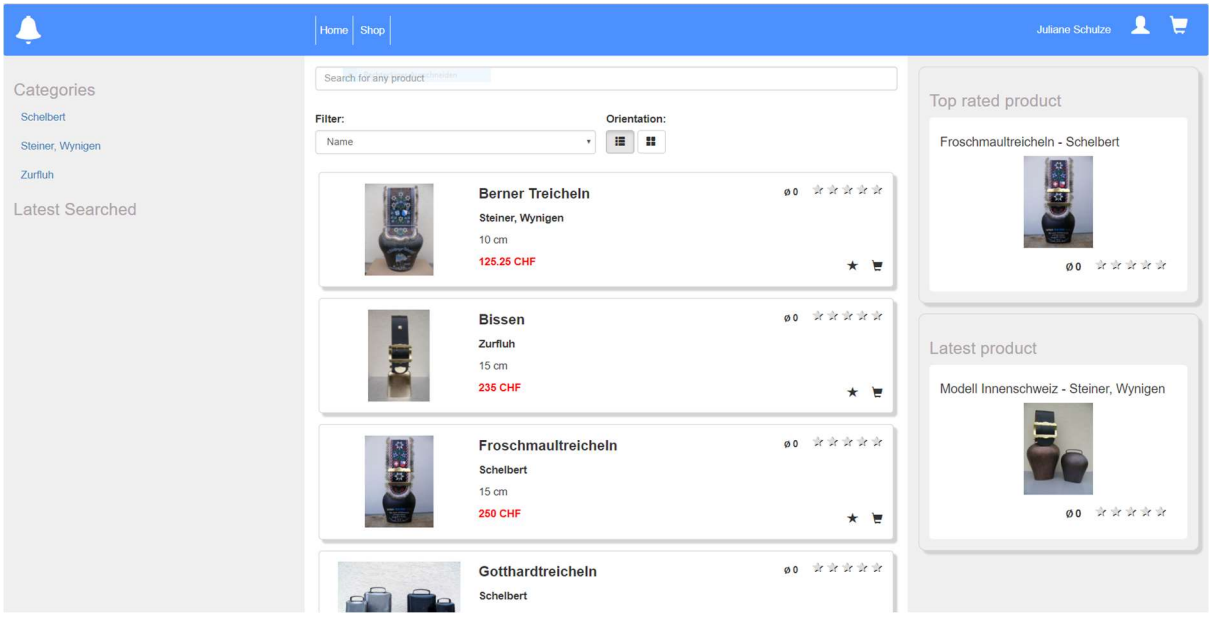


*Abbildung 21 Shop Ansicht der neuen Glocken Emil Applikation*

Es wurden nur geringfügige Anpassungen gemacht, welche nicht ganz mit den Mockups übereinstimmen. Ein Beispiel hierfür wäre der Warenkorb, welcher in den Mockups die Möglichkeit bietet, die Produktanzahl mittels Knöpfen anzupassen. Diese Feature wurde nicht umgesetzt, da es keinen funktionalen Nutzen auf die Anwendung gehabt hätte.


*Abbildung 22 Beispiel einer Abweichung bezüglich der Mockups*

## Schwachstellen

In den nachfolgenden Bereichen wird auf die umzusetzenden Schwachstellen näher eingegangen.

## NoSQL Injection - Bypassing Authentication

Der “Glocken Emil Shop” ermöglicht es, dem Nutzer das Login mittels einer NoSQL Injection Attacke zu umgehen. Der Angriff sieht wie folgt aus:

Statt mit einem registrierten Benutzer gezielt einzuloggen, können die Felder Benutzername und Passwort mit dem NoSQL Befehl { "$gt": “ “ } abgefüllt werden, was es dem Angreifer ermöglicht, mit dem ersten in der Datenbank gefundenen Benutzer einzuloggen. Dieses Szenario ist ausbaufähig, indem der Benutzername gezielt eingegeben wird und nur das Passwort Feld mittels NoSQL Befehl ausgetrickst wird.

### Implementierung

Im Authentifizierungs Service wird das Konto Modell mittels Benutzernamen und Passwort abgefragt. Beide Werte erhält der Service vom Authentifizierung Controller als String. Falls der NoSQL Befehl { "$gt": “ “ } in der Benutzeroberfläche als Benutzername oder Passwort eingetragen wird funktioniert die Injection nur, wenn es auch als Objekt erkannt wird. Hierfür musste der Benutzername und auch das Passwort versucht werden mittels JSON-Parse in ein Objekt zu verwandeln.

```
function login(username, password, callback) {
  if (!(username && password)) {
    return callback(null, ResponseUtil.createNotFoundResponse('Username or Password incorrect'));
  }

  // Injection Code Start - NoSQL Injection, Login bypass
  let hashedPassword = null;
  let usernameObj = null;

  try {
    hashedPassword = JSON.parse(password);
  } catch (exception) {
    hashedPassword = CryptoUtil.hashPwd(password);
  }

  try {
    usernameObj = JSON.parse(username);
  } catch (exception) {
    usernameObj = username;
  }
  // Injection Code End

  Account.findOne({
    username: usernameObj,
    password: hashedPassword,
  }, function(error, resAccount) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!resAccount) {
      return callback(null, ResponseUtil.createNotFoundResponse('Username or Password incorrect'));
    } else {
      let {_id, username, firstname, lastname, email} = resAccount;
      let user = {_id, username, firstname, lastname, email};
        CryptoUtil.createToken(user, GlobalConfig.jwt.secret, GlobalConfig.auth.signOptions, (error, token) => {
          if(error) return callback(ResponseUtil.createErrorResponse(error));
          let result = { 'user': user, 'token': token };
          return callback(null, ResponseUtil.createSuccessResponse(result, 'Login successfully'));
      });
    }
  });
}
```

*Abbildung 23 Umgesetzte Schwachstelle - NoSQL Injection*

## Hidden Functionality - Retailer Rabatt

Im Webshop ist es möglich einen Einblick in das Retailer API zu erhalten, welches darauf hinweist, dass man eine Buchung, welche nicht mittels Kreditkarte durchgeführt wurde, mit einem Preisnachlass zu versehen. Der Aufruf sieht wie folgt aus:

http://SERVER-IP:PORT/retailer/order/change/:orderId

Wenn die URL mit dem Parameter der Buchungsnummer ergänzt wird, wird diese Buchung mit einem Rabatt von 50% versehen. Der Angreifer hat somit eine Vergünstigung erhalten, weil die versteckte Funktionalität nicht ordnungsgemäss entfernt wurde.

### Implementierung

Im Backend Service für Retailer wurde die Rabatt Funktion zur Verfügung gestellt. Die Methode halbiert den totalen Preis der ursprünglichen Buchung und korrigiert auch dementsprechend jedes Item in der Buchung.

```
function change(orderId, callback) {
  if(!orderId) return callback(ResponseUtil.createNotFoundResponse());
  Order.findById(orderId, function(error, result) {
    if(error) return callback(ResponseUtil.createErrorResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());

    if(result.payment.type === 'bill') {
      result.totalPrice = round(result.totalPrice * 0.5, 0.05);
      result.items.forEach(function(item) {
        item.product.price = round(item.product.price * 0.5, 0.05);
      });
      result.save();
    }

    result = { 'order' : result };
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}

function round(value, step) {
  step || (step = 1.0);
  let inv = 1.0 / step;
  return Math.round(value * inv) / inv;
}
```

*Abbildung 24 Umgesetzte Schwachstelle - Hidden Functionality*

## Unprotected REST API - Retailer Rabatt

Die versteckte Funktionalität “Retailer Rabatt” benötigt keine Authentifizierung und ist somit für jeden zugänglich.

### Implementierung

Im Backend in der Datei app.js sind die Routen definiert. Für alle Routen, welche nach dem Einbinden des ExpressJS Moduls Express-JWT stehen ist ein gültiger Token notwendig. Um die Route /api/retailer für unautorisierte Benutzer zur Verfügung zu stellen musste diese Route vor dem Einbinden des Express-JWT Moduls definiert werden.

```
app.use('/slider-images', express.static(__dirname + '/assets/slider-images'));
app.use('/favicon.ico', express.static(__dirname + '/assets/favicon.ico'));
app.use('/api/auth', auth);
// Injection Code Start - Unprotected REST API
app.use('/api/retailer', retailer);
// Injection Code End

app.use(expressJwt(GlobalConfig.auth.validateOptions).unless(GlobalConfig.auth.unprotectedRoutes));

app.use('/api/account', account);
app.use('/api/creditCard', creditCard);
app.use('/api/deliveryAddress', deliveryAddress);
app.use('/api/order', order);
```

*Abbildung 25 Umgesetzte Schwachstelle - Unprotected REST API*

## SVG-Injection - Community Post

Wird das folgende Bild mittels File Upload im Bereich Community dem Post beigefügt, so wird dieser auf der Home Seite wie ein gewöhnlicher Post inklusive Bild angezeigt. Jedoch wird der JavaScript Code, welche im SVG Bild inkludiert ist mit ausgeführt. In diesem Fall wird ein JavaScript Alert ausgeführt mit der Meldung “!SVG’s are dangerous!”.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
        "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <circle cx="250" cy="250" r="50" fill="red" />
    <script type="text/javascript"><![CDATA[
    alert("!SVG's are dangerous!");
  ]]></script>
</svg>
```

*Abbildung 26 Beispiel eines SVG Bildes mit JavaScript Code*

### Implementierung

Im Frontend mussten hier folgende Codezeilen angepasst werden.

```
    });
  };

  self.init();
}])
// Injection Code Start - SVG Injection
.filter('trustAsResourceUrl', ['$sce', function ($sce) {
  return function(postImage) {
    let serverUrl = 'http://localhost:3000/post-images/';
    return $sce.trustAs($sce.RESOURCE_URL, serverUrl + postImage);
  };
}])
// Injection Code End
.filter('extension', function() {
```

*Abbildung 27 Umgesetzte Schwachstelle - SVG Injection*

Es wurde der Filter “trustAsRessourceUrl” im Home Controller ergänzt, welcher das Modul Sanitize nutzt und somit ermöglicht externen Inhalt als vertrauenswürdig zu definieren.

In der View musst dann der Filter für diese Ressource angewandt werden.

```
<div class="row">
  <!-- Injection Code Start - SVG Injection -->
  <img class="post-image" ng-src="{{ post.image | trustAsResourceUrl }}"
       ng-if="(post.image | extension) !== 'svg'" />
  <object ng-attr-id="post-image-{{ $index }}"
          ng-attr-data="{{ post.image | trustAsResourceUrl }}"
          type="image/svg+xml"
          ng-if="(post.image | extension) === 'svg'"></object>
  <!-- Injection Code End -->
</div>
```

*Abbildung 28 Umgesetzte Schwachstelle - SVG Injection – View*

## Direct Object Reference - Kreditkarten Manipulation

Im Webshop ist die Bearbeitung der Kreditkarte mittels direkter Objekt Referenz möglich.

http://SERVER-IP:PORT/creditcard/:creditCardNumber

Die Überprüfung, ob der Aufrufer der Kreditkarte zugleich der Inhaber der Kreditkarte ist, wurde deaktiviert. Somit steht dem Angreifer nichts im Weg, um gezielt Kreditkarten im System zu manipulieren oder auszulesen.

### Implementierung

Im Backend wurde im Kreditkarten Controller die Überprüfung, ob der aufrufende Benutzer auch der Inhaber der Kreditkarte ist weggelassen. Zudem kann die Kreditkarte mit der Nummer der Karte abgerufen werden.

```
function getByNumber(creditCardNumber, callback) {
  if(!creditCardNumber) return callback(ResponseUtil.createNotFoundResponse());
  CreditCard.findOne({ number: creditCardNumber }, function(error, result) {
    if(error) return callback(ResponseUtil.createNotFoundResponse(error));
    if(!result) return callback(ResponseUtil.createNotFoundResponse());
    result = { 'creditCard' : result};
    return callback(null, ResponseUtil.createSuccessResponse(result));
  });
}
```

*Abbildung 29 Umgesetzte Schwachstelle - Direct Object Reference*

## XSS - Produkt bewerten

Der Webshop bietet die Möglichkeit, die Produkte im Shop mit einem Kommentar und einer Wertung zu versehen. Dieses Kommentarfeld wird nicht validiert und eröffnet die Gelegenheit ein schädliches Script hochzuladen, welches dann im rechten Sidebar unter “Top Rated Product”

angezeigt wird und bei den restlichen Benutzern ausgeführt wird und somit die Möglichkeit bietet die Lokalen Benutzerdaten inklusive Token auszulesen.

### Implementierung

Im Frontend musste im Shop Controller der Filter “trustAsHtml” implementiert werden. Dieser Filter ermöglicht es mittels Modul Sanitize den Inhalt als vertrauenswürdige HTML Ressource zu definieren.

```
  };
})
// Injection Code Start - XSS
.filter('trustAsHTML', ['$sce', function ($sce) {
  return function(comment) {
    return $sce.trustAs($sce.HTML, comment);
  };
}]);
// Injection Code End
```

*Abbildung 30 Umgesetzte Schwachstelle - XSS*

Des Weiteren musste in den beiden Views “topRatedProduct” und “latestProduct” der Filter eingebunden werden.

```
<!-- Injection Code Start - XSS -->
<div ng-bind-html="latestProductRating.comment | trustAsHTML"></div>
<!-- Injection Code End -->
```

*Abbildung 31 Umgesetzte Schwachstelle - XSS – View latestProduct*

```
<!-- Injection Code Start - XSS -->
<div ng-bind-html="topRatedProductRating.comment | trustAsHTML"></div>
<!-- Injection Code End -->
```

*Abbildung 32 Umgesetze Schwachstelle - XSS - View topRatedProduct*

# Probleme

## AngularJS

Ein erstes Problem tauchte auf, als der Authentifizierungs Controller im AngularJS zweimal genutzt werden sollte. Durch den Einsatz zweier Instanzen des Controllers hatten beide Controller einen unterschiedlichen Scope. Dies führte dazu, dass die angezeigten Daten der Controller nicht übereingestimmt haben.

Die Lösung des Problems bot sich mit dem Einsatz eines Shared Services, welcher die Daten vom REST API bezieht und die Daten an den Controller zurückgibt.

Ein weiteres Problem, welches sich kurz nach der ersten Construction Phase bemerkbar machte war, dass die Implementation der Webshop Funktionalität aufwendiger war als erwartet. Daraus resultierte, dass ich meine Planung ändern musste und den Meilenstein für die Fertigstellung der Webshop Funktionalität um eine Woche nach hinten schieben musste.

Des Weiteren gab es ein Problem mit der Same Origin Policy, da das Backend und das Frontend einzeln gehostet waren. Das Frontend konnte somit nicht auf Inhalte, welche auf das Backend geladen wurden zugreifen.

Das Hosten des Frontends über das ExpressJS Backend bot hier Abhilfe und löste das Problem.

Zu guter Letzt entstand noch das Problem, dass beim Einbau der XSS- und SVG-Schwachstelle der JavaScript Code nach dem Absenden des Formulars nicht auf dem AngularJS ausgeführt wurde.

Hier half das Sanitize Modul, welches es ermöglicht HTML Code inklusive dem JavaScript zu trauen und somit die Ausführung dieses zu ermöglichen.

## Testing

Das Testen zeigte sich als sehr aufwendig. Nur schon das Schreiben eines einzigen Testfalls dauerte Ewigkeiten. Somit wurde der Fokus der Arbeit auf die Funktion des Webshops gelegt und weniger auf die Tests.

# Fazit

Trotz der zuvor beschriebenen Probleme, welche zum Teil auf fehlendes Know How zurückzuführen sind konnte rasch eine Lösung gefunden werden. Die JavaScript Frameworks waren sehr gut geeignet für die Umsetzung dieser Webanwendung. Auch die NoSQL Datenbank MongoDB konnte leicht in das Backend implementiert werden. Das ExpressJS Modul Mongoose war hier sehr vorteilhaft, da zugleich auch Validierungen auf den Objekten erstellt werden konnten.

# 7. Anhang A: Aufgabenstellung

## Auftraggeber und Betreuer

Diese Studienarbeit bildet den neuen Grundstock für die Schulung im Bereich Web Security für die Compass Security Network Computing AG.

## Betreuer

- Cyrill Brunschwiler, Compass Security Network Computing AG, cyrill.brunschwiler@compass-security.com

## Autor

- Patrick Steinhäusl, Hochschule für Technik Rapperswil, psteinha@hsr.ch

## Ausgangslage

Die Firma Compass Security Network Computing AG nutzt zu Schulungszwecken eine Webanwendung namens “Glocken Emil Shop”, welcher 2007 erstellt wurde. Dieser Webshop behandelt die Problematiken der OWASP Top Ten, welche sich in den letzten 10 Jahren kaum geändert haben. Die Technologien, welche heute im Web Engineering eingesetzt werden, haben jedoch sich jedoch stark gewandelt. Der etwas verstaubte Webshop soll somit in dieser Arbeit mit neuen Technologien neu aufgebaut werden und die Schwachstellen der OWASP Top Ten, sowie einige Spezialthemen der aktuellen einzusetzenden Technologien beinhalten.

## Ziele und Aufgabenstellung

Das Ziel dieser Studienarbeit ist es, die Webanwendung “Glocken Emil Shop” mit den folgenden Technologien umzusetzen:

- **Server API:** NodeJS und ExpressJS
- **Client:** AngularJS
- **Datenbank:** MongoDB
- **Image:** Docker

In die Webanwendung müssen folgenden Schwachstellen eingebaut werden:

- **NoSQL Injection**
- **Hidden Functionality**
- **SVG Injection**
- **Unprotected REST API (forgotten auth)**
- **Direct Object References**

Die Software muss anschliessend auf einem Docker Image lauffähig und für die Schulung nutzbar sein. Für die Schulung müssen die Verwundbarkeiten in eine Aufgabe eingebunden und mit Musterlösung versehen werden.

## Durchführung

Mit dem Betreuer der Arbeit ist wöchentlich am Donnerstag um 9:00 Uhr in der Compass Security Network Computing AG ein Meeting mit der Dauer von 1 Stunde angesetzt.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Es soll ein kontinuierlicher und sichtbarer Arbeitsfortschritt ersichtlich sein. Die Meilensteine im Projektplan reflektieren die Arbeitsresultate. Eine definitive Beurteilung erfolgt bei der finalen Abgabe und basiert auf den abgegebenen Dokumenten.

## Dokumentation

Die Arbeit muss dokumentiert und die erstellten Dokumente müssen im Projektplan ersichtlich sein. Während des Projekts sollen alle Dokumente auf dem aktuellsten Stand vorhanden sein. In der Abgabe müssen alle Dokumente vollständig auf CD/DVD/USB in 3 Exemplaren abgegeben werden. Auf Wunsch des Auftraggebers oder Betreuers ist eine gedruckte Version zu erstellen.

## Termine

Folgende Termine gelten für diese Studienarbeit:

| | |
|---|---|
| 15.09.17 | Start der Studienarbeit mit Ausgabe der Aufgabenstellung durch den Betreuer |
| 19.12.17 | Abgabe des Abstracts im Abstract Online Tool zur Überprüfung und Freigabe des Betreuers. |
| 22.12.17 | Der Betreuer gibt das vollständige Dokument mit dem korrekten und vervollständigten Abstract an das Studiensekretariat weiter. |
| 22.12.17 | Abgabe des Berichts an den Betreuer bis 17:00 Uhr |

*Tabelle 3 Termine der Studienarbeit*

## Beurteilung

Die erfolgreiche Studienarbeit erhält 8 ECTS-Punkte, wobei 1 ECTS-Punkt einer Arbeitsleistung von ca. 25 Stunden entspricht. Die Modulbeschreibung der Studienarbeit ist unter folgendem Link zu finden

http://studien.hsr.ch/allModules/24386_M_SAI14.html

# 8. Anhang B: Zeitauswertung

Die nachfolgende Grafik zeigt die Zeitauswertung über das ganze Projekt.

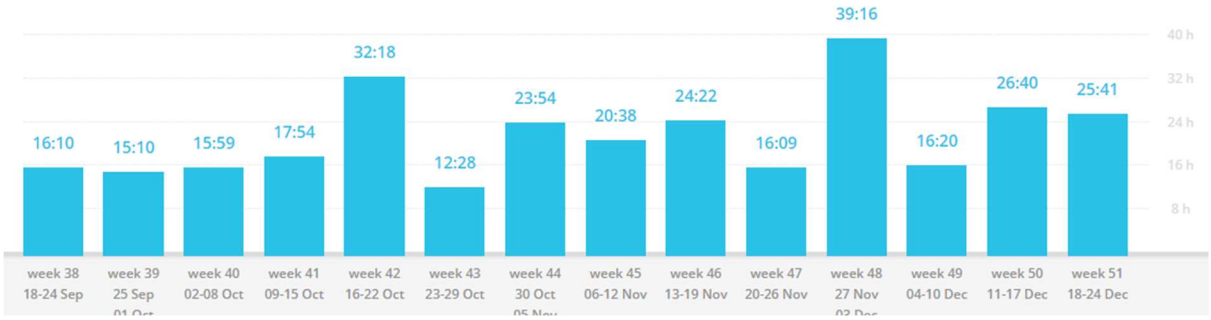


*Abbildung 33 Zeitauswertung des gesamten Projektes*

Unten befindet sich die detaillierte Zeitauswertung der Inception Phase.

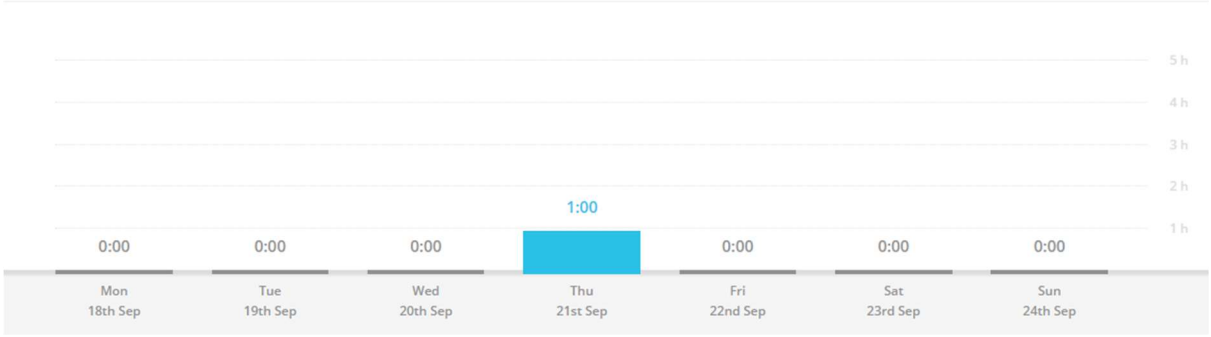


*Abbildung 34 Zeitauswertung der Inception Phase*

Nun folgt die detaillierte Zeitauswertung der Elaboration Phase.

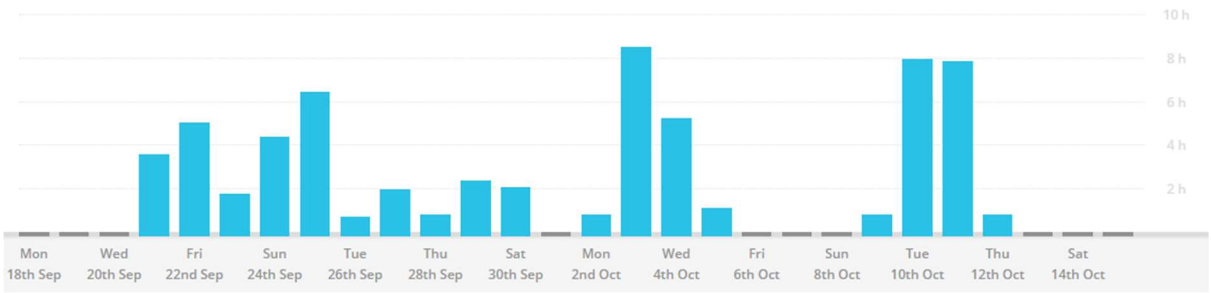


*Abbildung 35 Zeitauswertung der Elaboration Phase*

Unten ist die detaillierte Auflistung der Construction Phase.

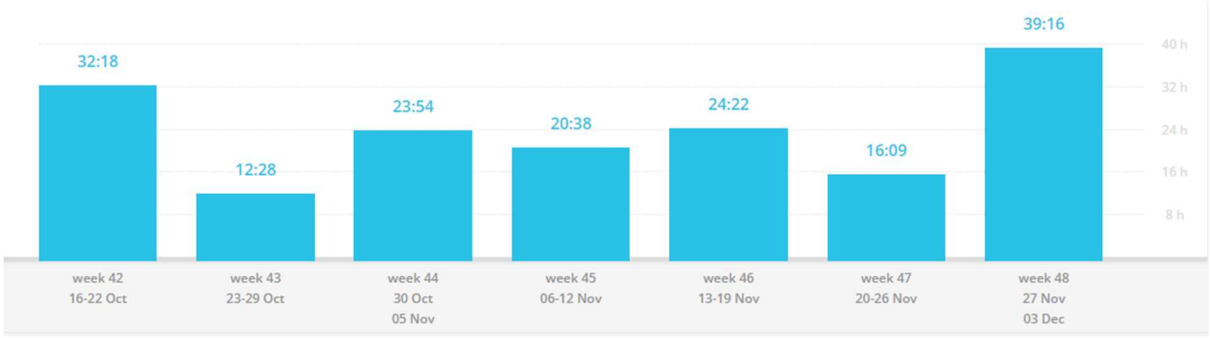


*Abbildung 36 Zeitauswertung der Construction Phase*

Zuletzt die genaue Auflistung der Transition Phase.

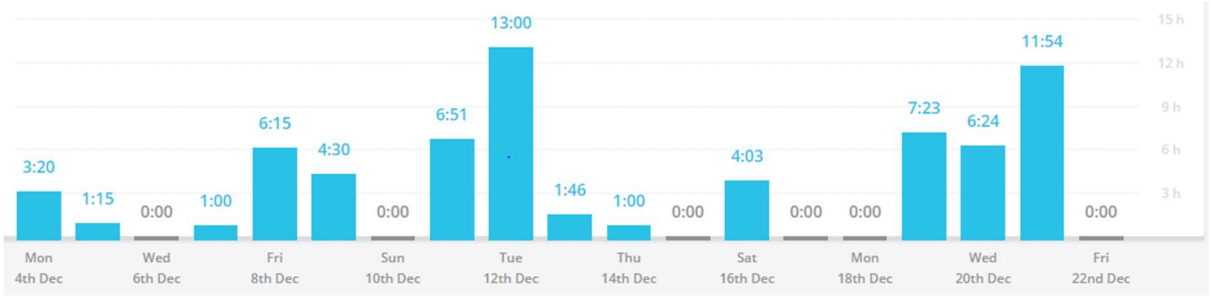


*Abbildung 37 Zeitauswertung der Transition Phase*

# 10. Anhang C: Mockups

## Home Ansicht

Die Home Ansicht zeigt die Startseite des neuen Webshops.

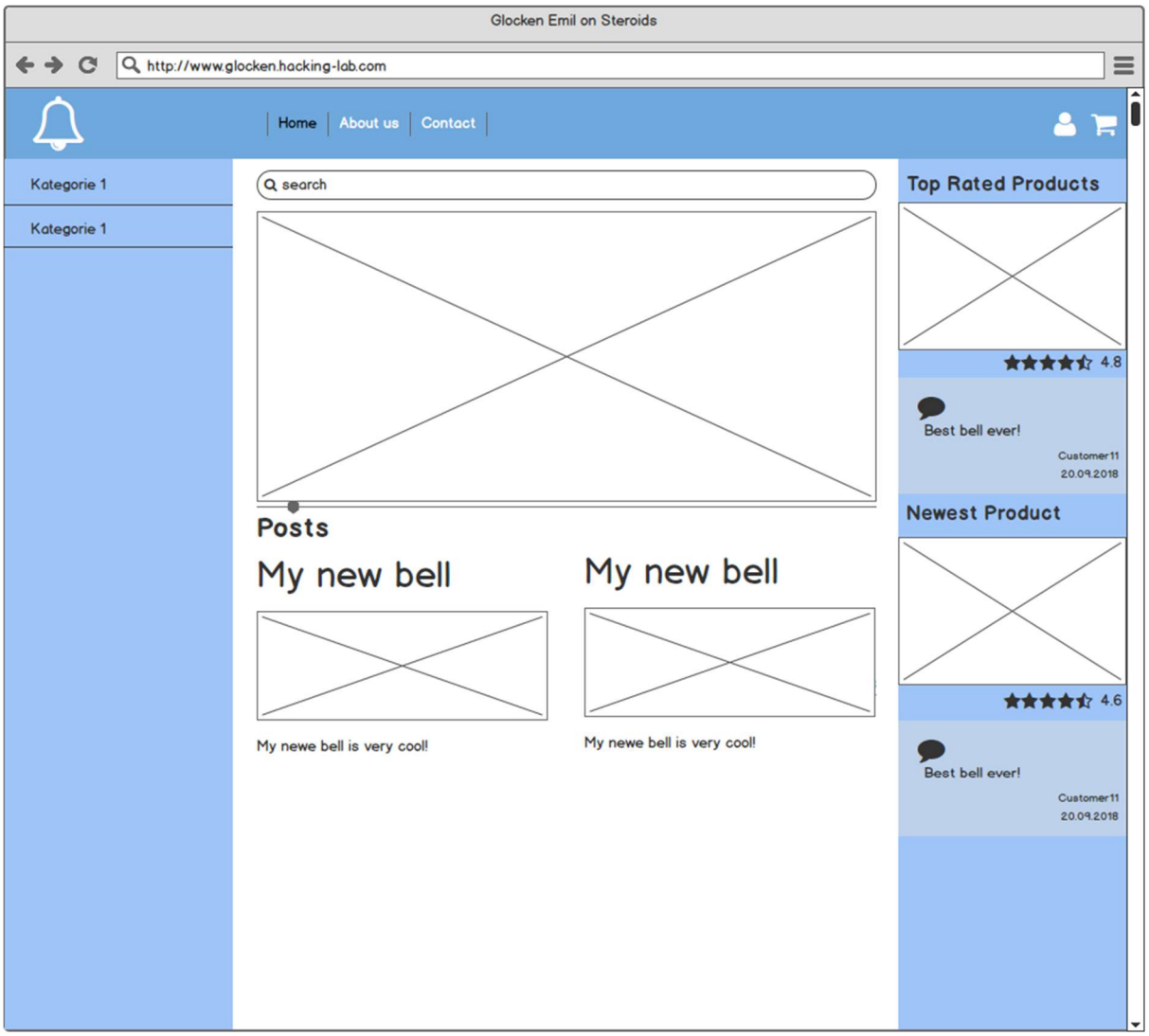


*Abbildung 38 Mockup - Home Ansicht*

# Shop Ansicht - Einspaltig

Hier wird der Übersicht der Produkte des Shops als einspaltige Ansicht gezeigt.

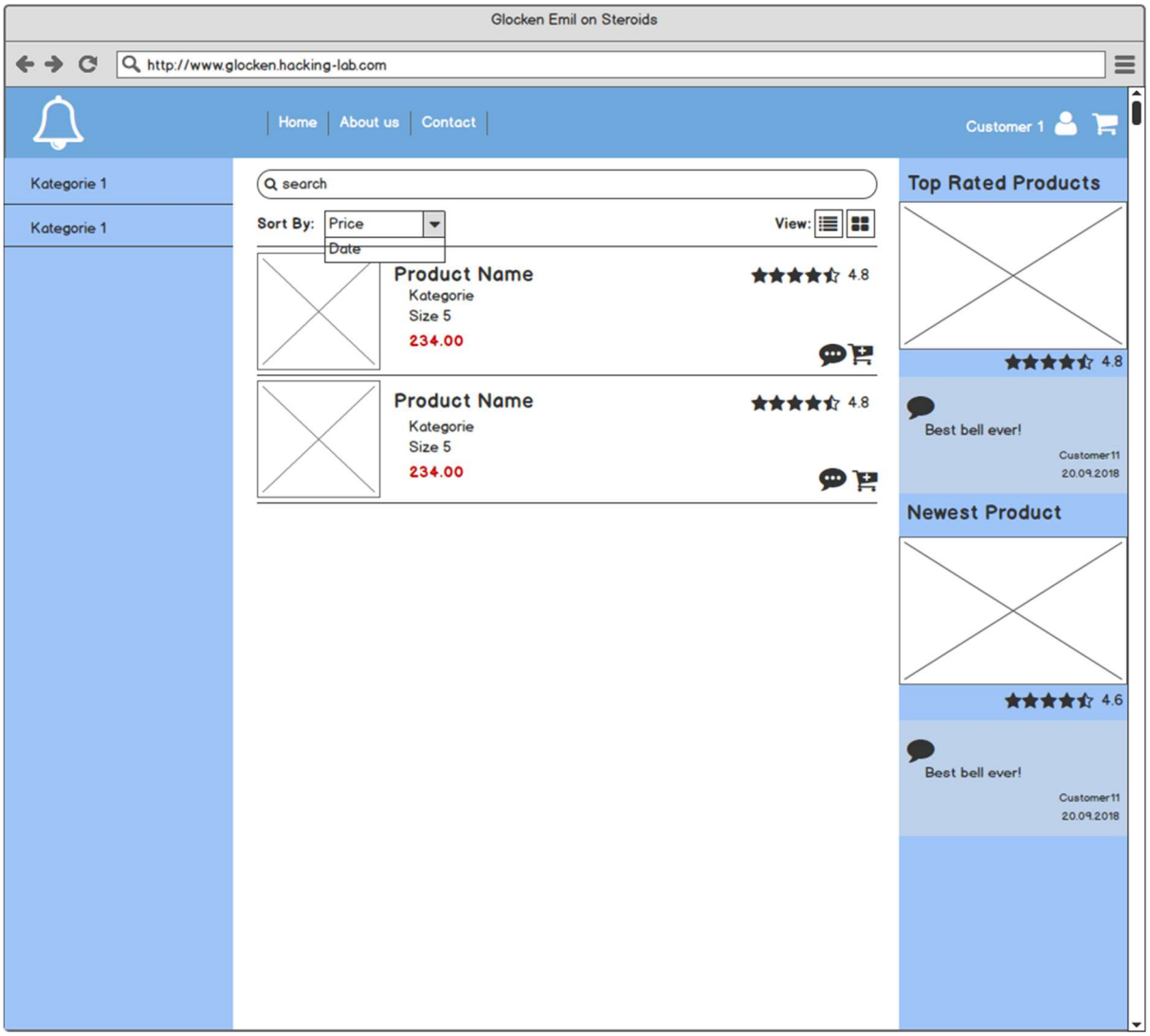


*Abbildung 39 Mockup - Shop Ansicht einspaltig*

# Shop Ansicht - Zweispaltig

Hier wird der Übersicht der Produkte des Shops als zweispaltige Ansicht gezeigt.

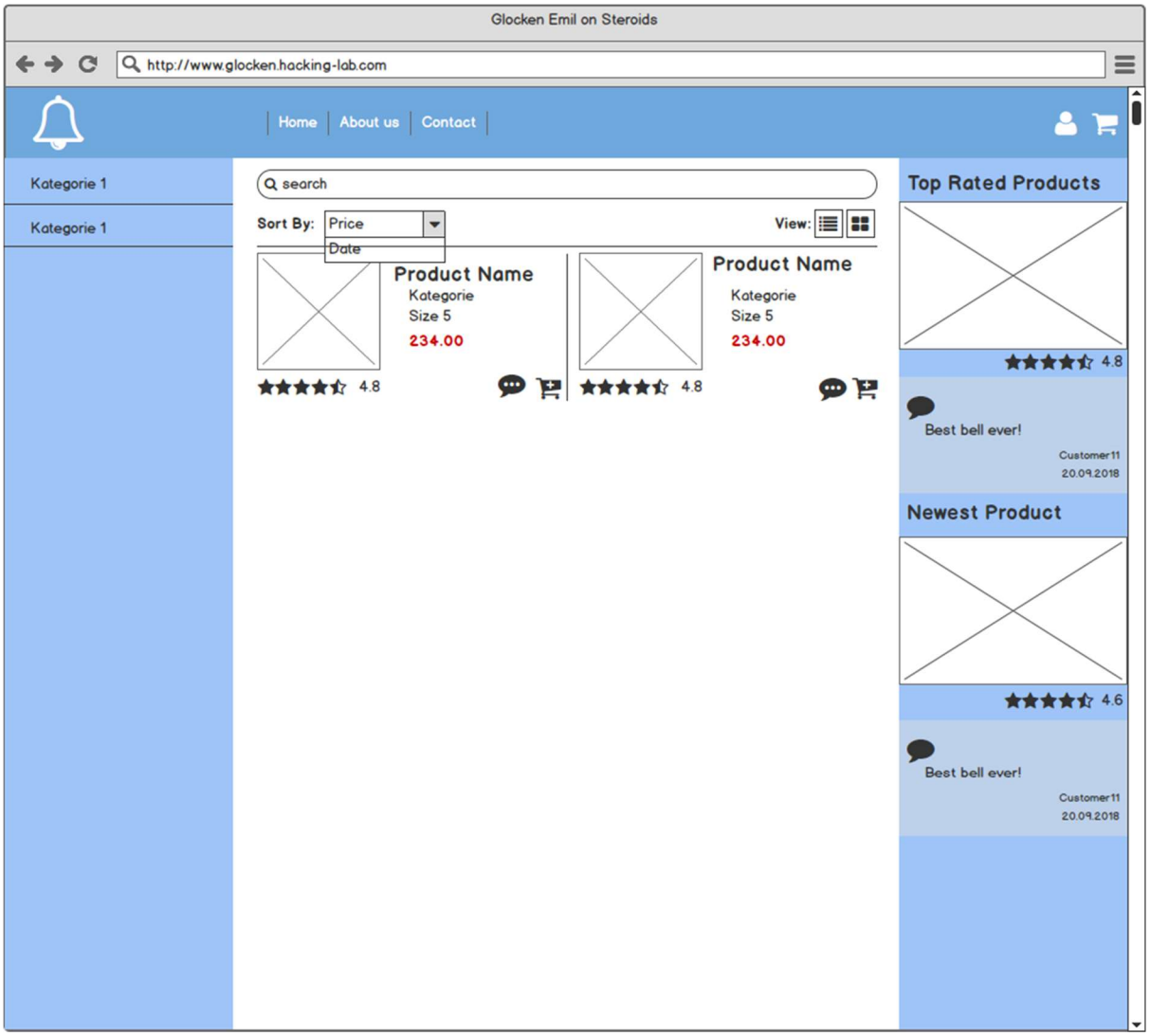


*Abbildung 40 Mockup - Shop Ansicht zweispaltig*

# Shop Ansicht - Benutzermenü

In diesem Bild ist das Benutzermenü ersichtlich.

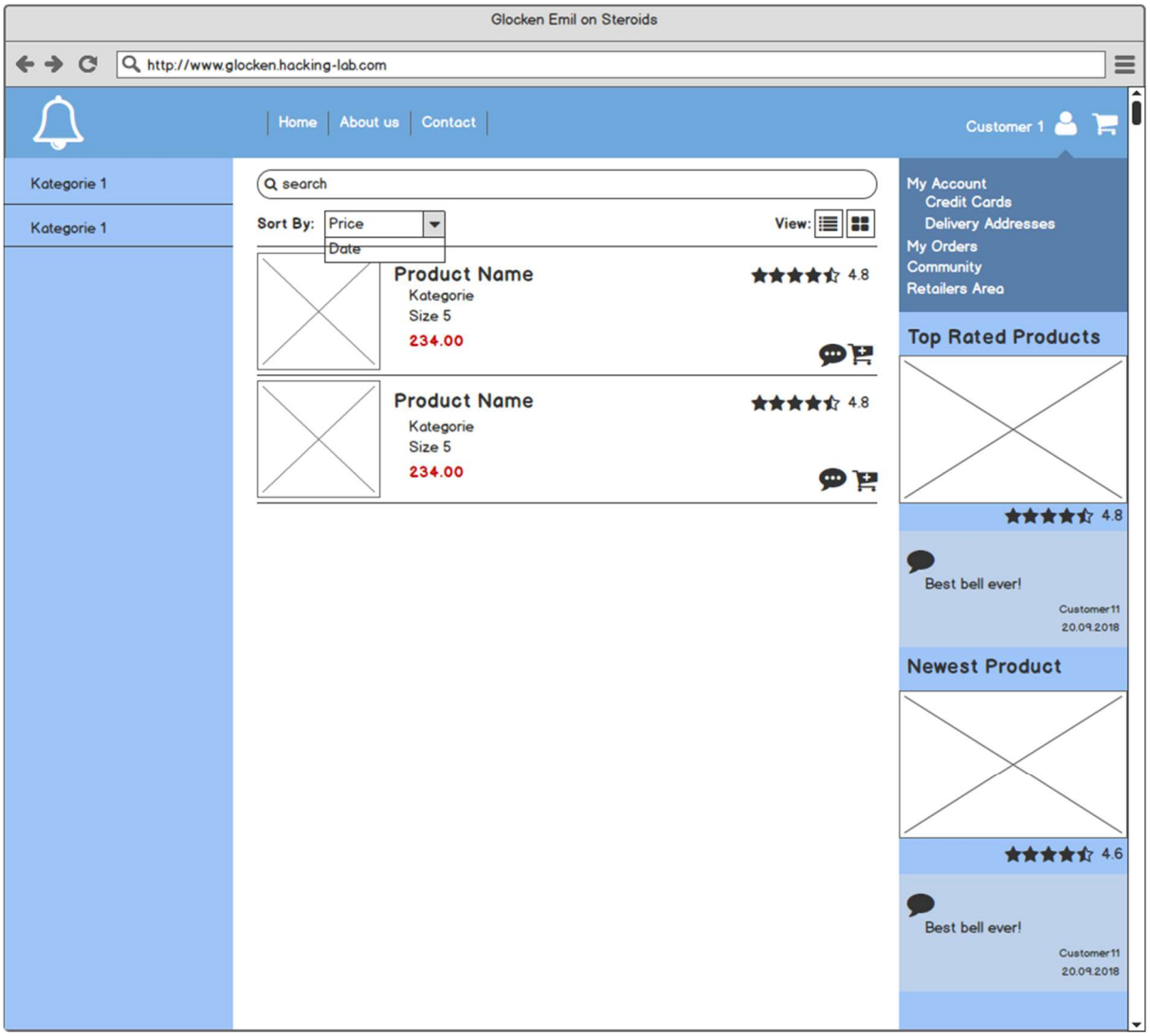


*Abbildung 41 Mockup - Shop Ansicht mit Benutzermenü*

# Shop Ansicht - Zuletzt gesuchte Produkte

In dieser Ansicht ist ersichtlich, dass die zuletzt gesuchten Produkte in der linken Navigation angezeigt werden.

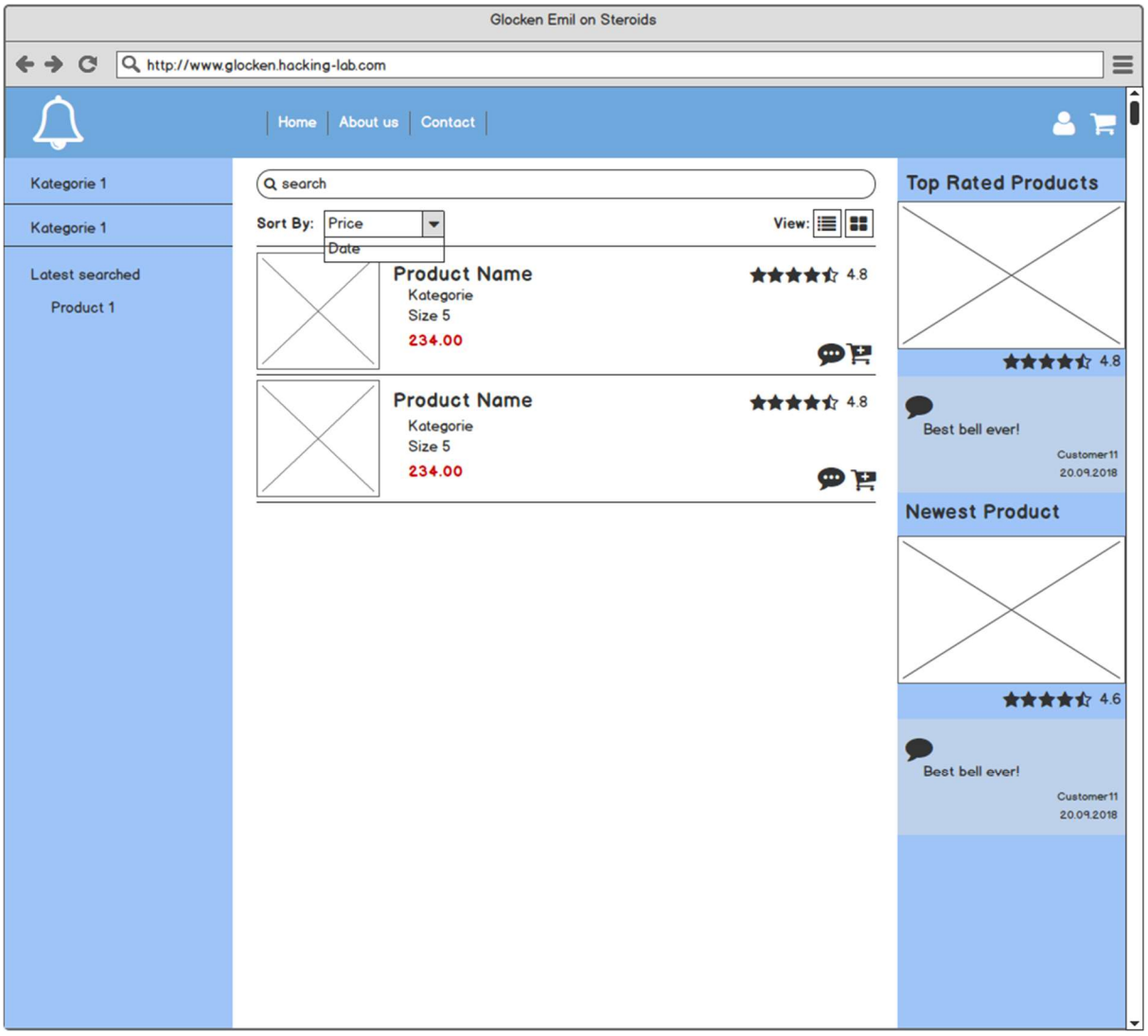


*Abbildung 42 Mockup - Shop Ansicht inklusive zuletzt gesuchten Produkt*

# Shop Ansicht - Produkt bewerten

Jedes Produkt im Shop kann bewertet werden.

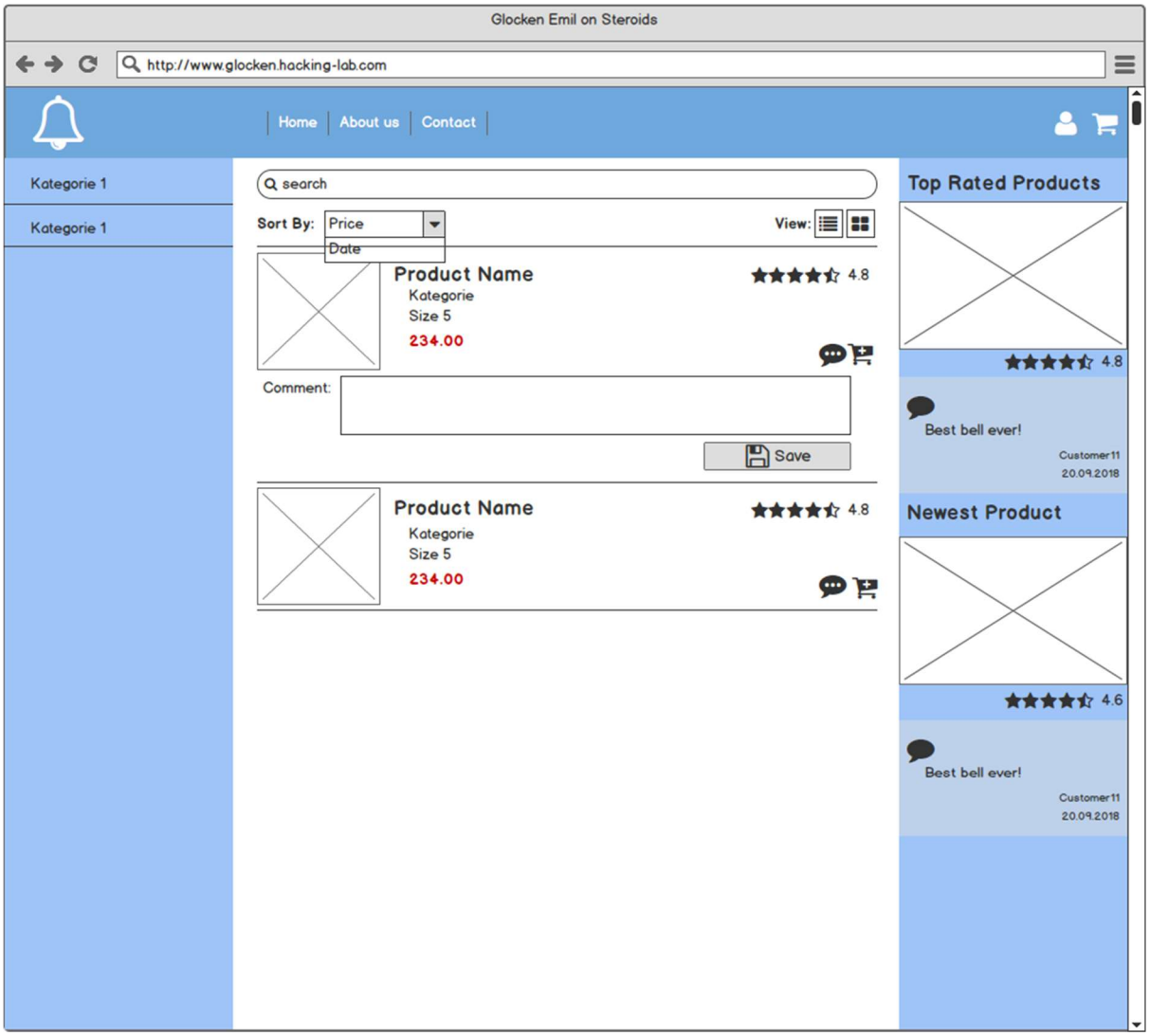


*Abbildung 43 Mockup - Shop Ansicht mit Produktbewertung*

# Shop Ansicht - Login

Die Login Maske wird hier nachfolgend gezeigt.

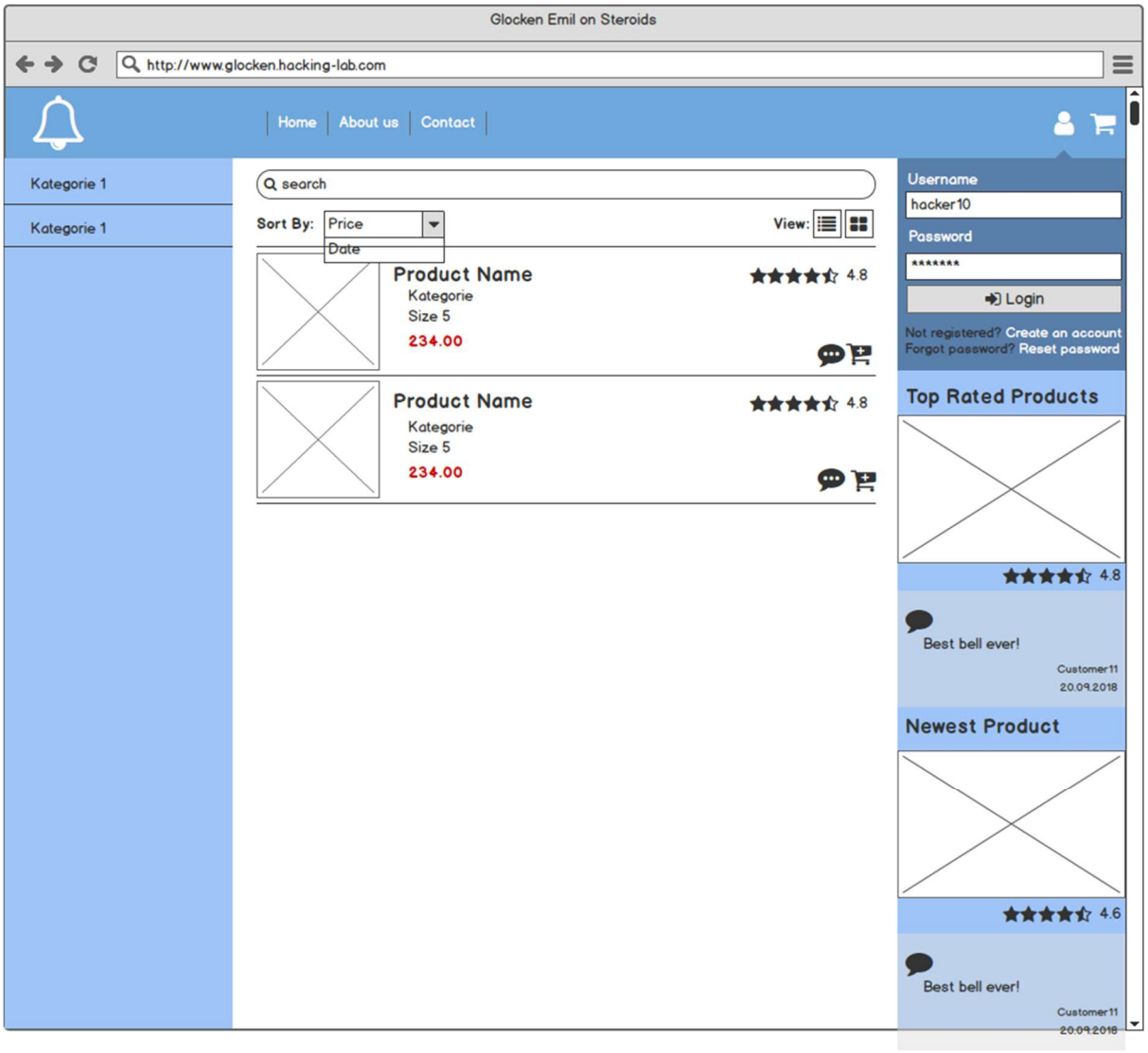


*Abbildung 44 Mockup - Shop Ansicht mit Loginmaske*

# Shop Ansicht - Warenkorb

In diesem Mockup ist der Warenkorb ersichtlich.

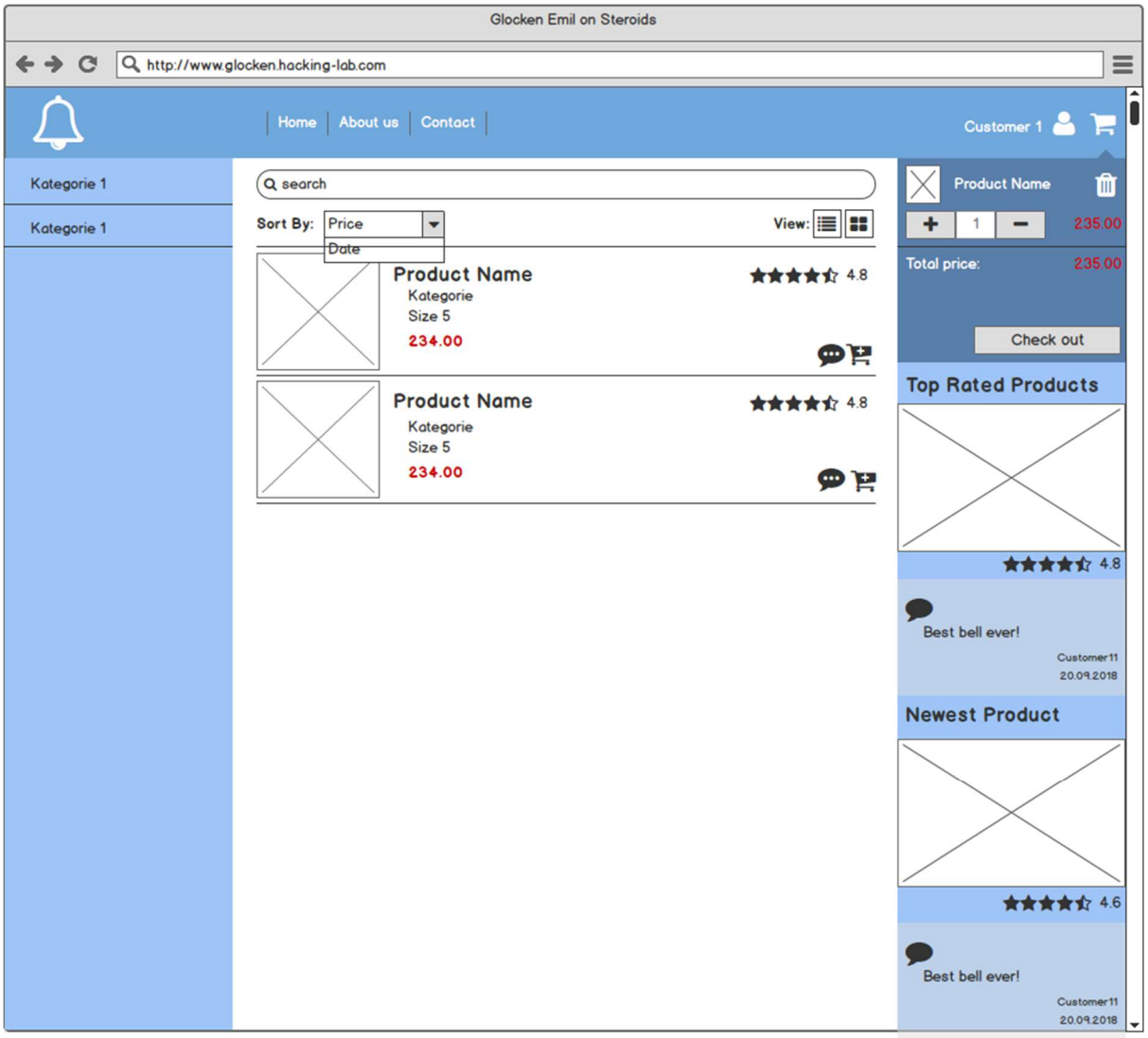


*Abbildung 45 Mockup - Shop Ansicht mit Warenkorbmenü*

# Order Ansicht - Übersicht

Hier ist eine Übersicht der Artikel des Warenkorbs ersichtlich, nachdem der Bestellprozess gestartet wird.

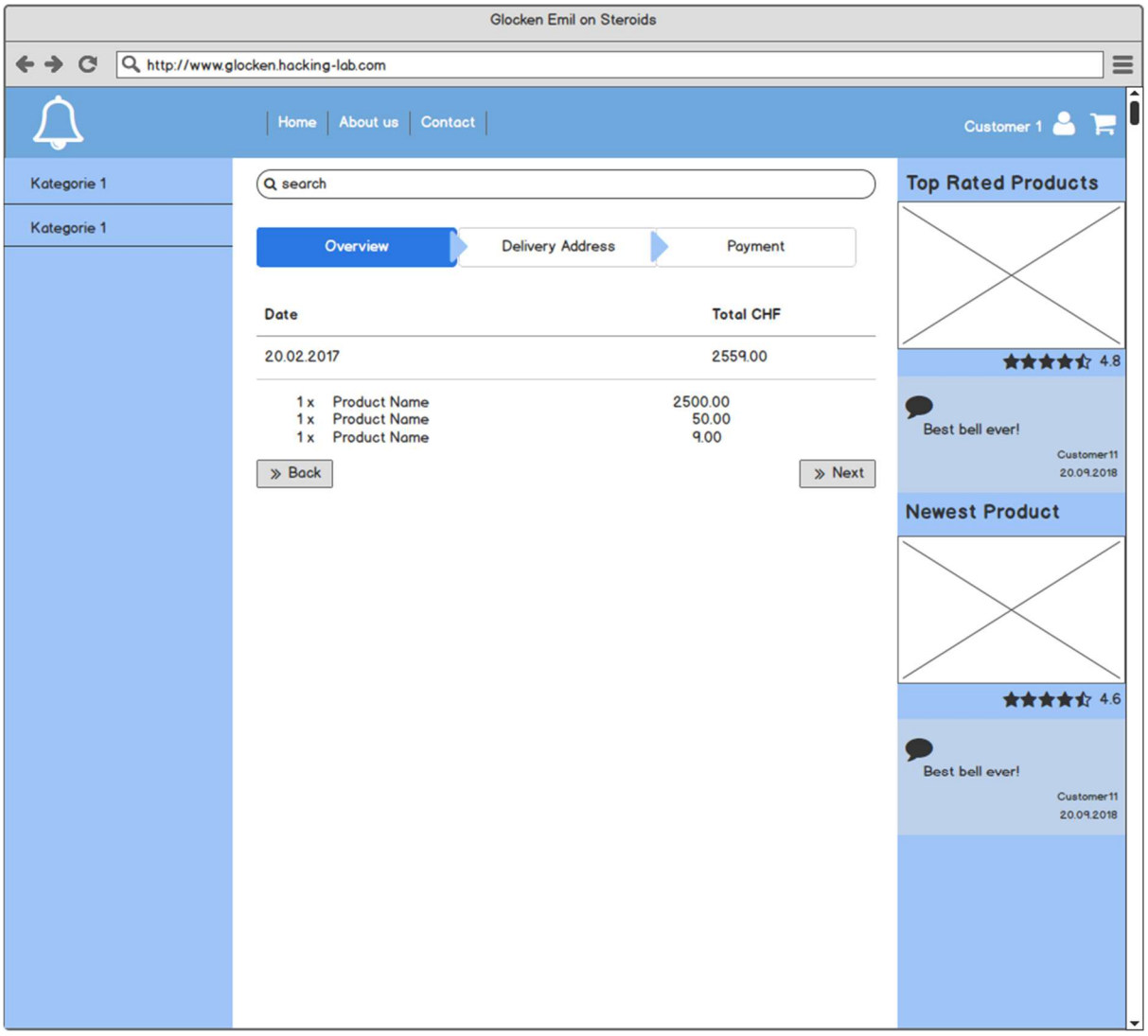


*Abbildung 46 Mockup - Übersicht der Artikel während Bestellprozess*

## Order Ansicht - Lieferadressen

Der nächste Schritt nach der Übersicht der Warenkorb Artikeln ist die Auswahl der Lieferadresse im Bestellprozess.

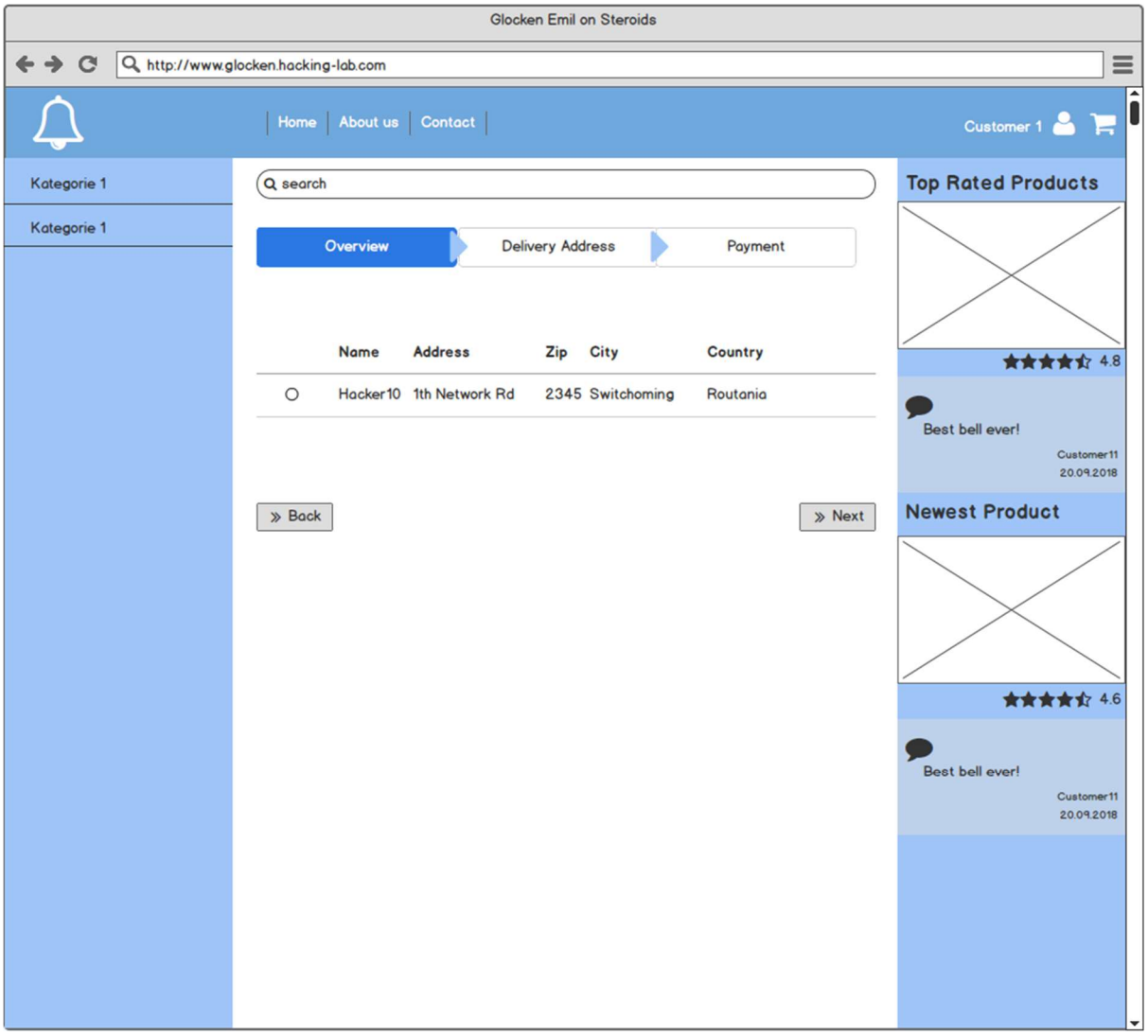


*Abbildung 47 Mockup - Lieferadressenauswahl während Bestellprozess*

# Order Ansicht - Zahlungsmethode

Der letzte Schritt des Bestellprozesses beinhaltet die Auswahl der Zahlungsmethode.

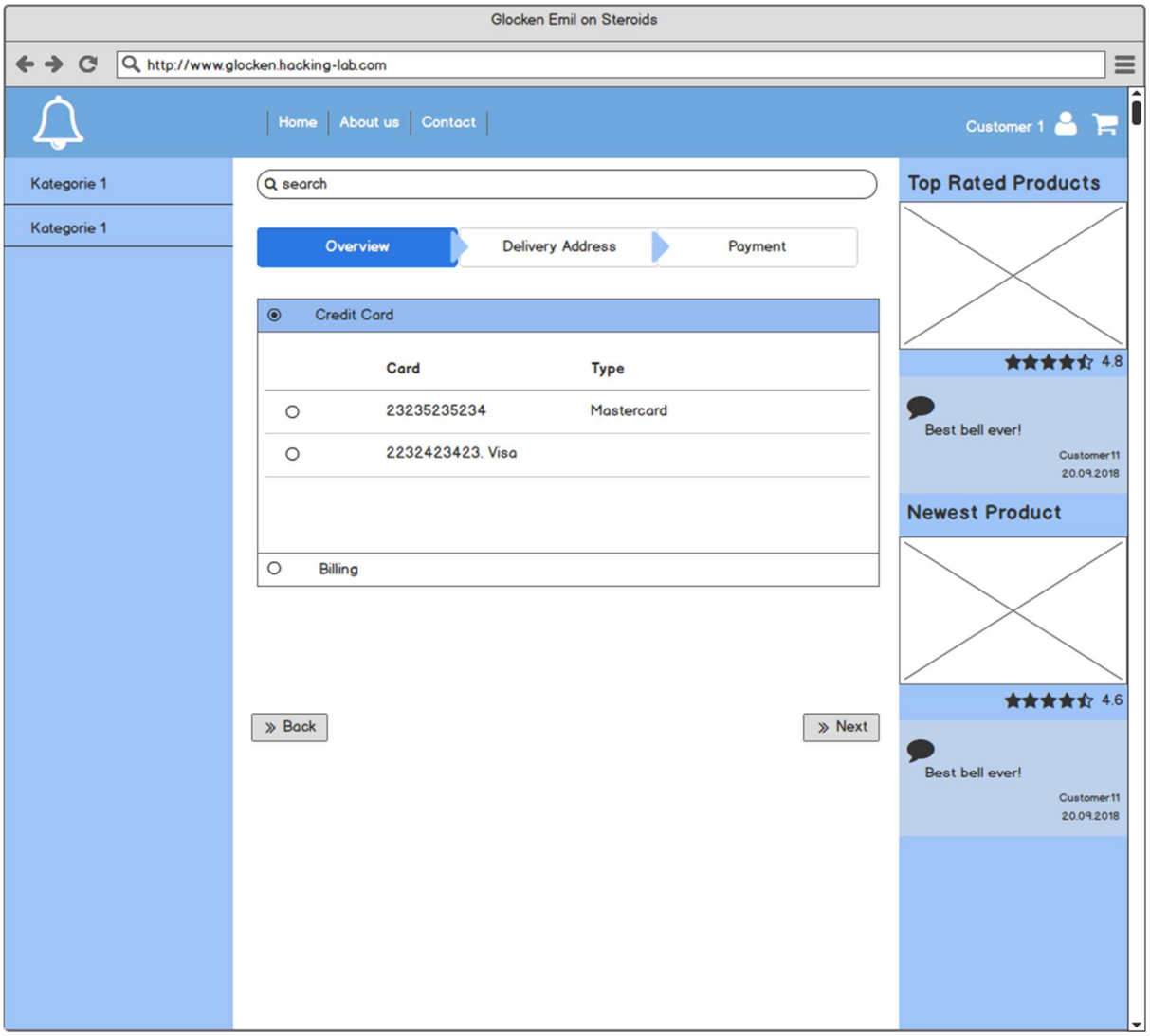


*Abbildung 48 Mockup - Zahlungsmethodenauswahl während Bestellprozess*

# Konto Ansicht

Das nachfolgende Bild zeigt das Konto mit den erfassten Daten.

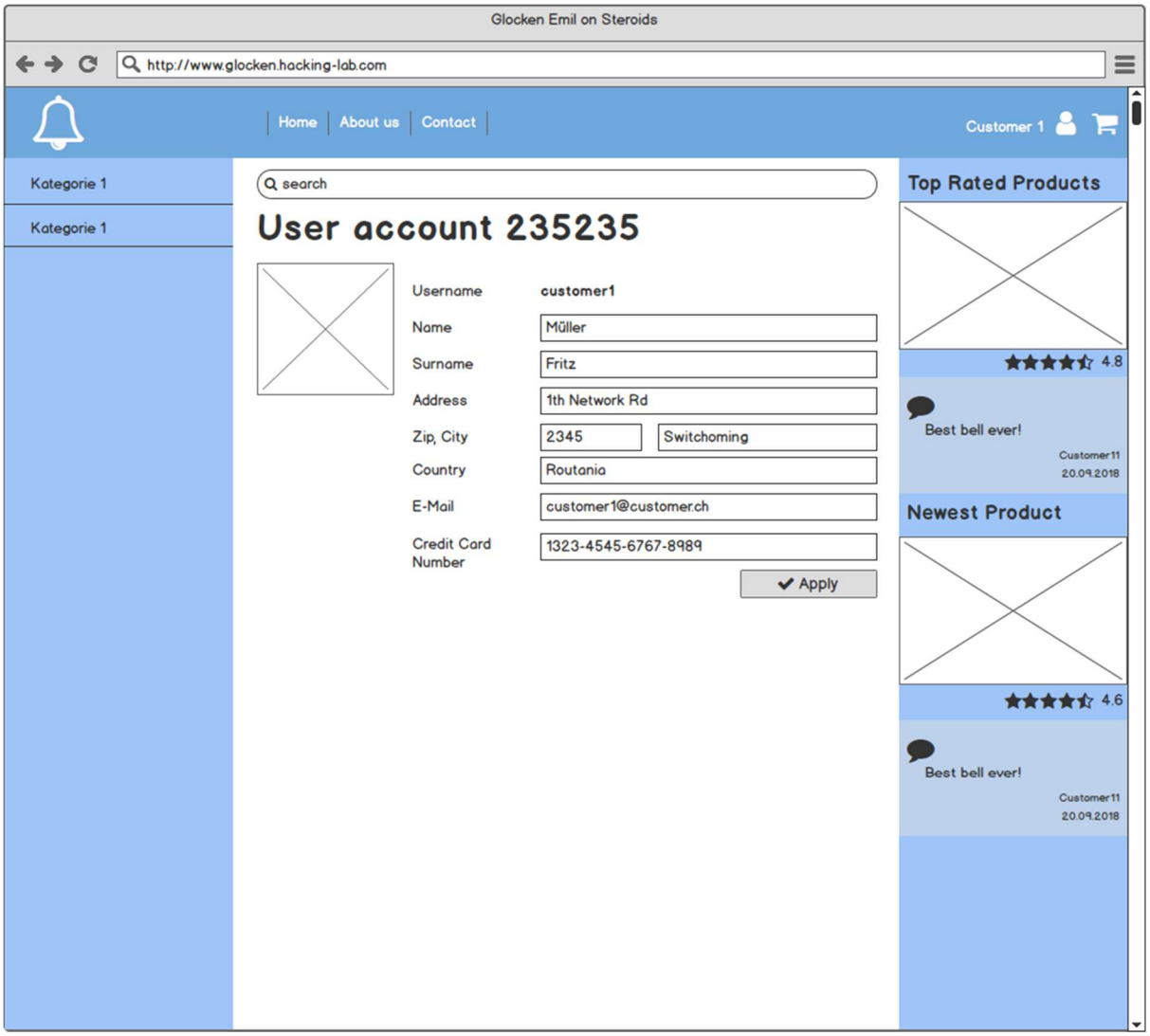


*Abbildung 49 Mockup - Konto Ansicht*

# Konto Ansicht - Kreditkarten

Hier wird die Kreditkarten Übersicht gezeigt.

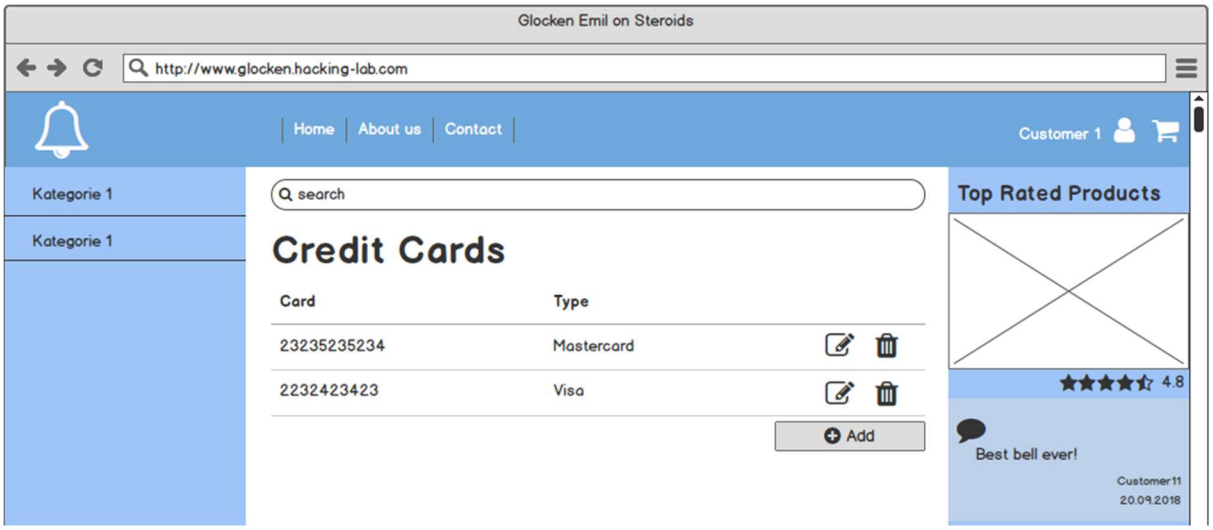


*Abbildung 50 Mockup - Überblick der Kreditkarten des Kontos*

## Konto Ansicht - Lieferadressen

Im nachfolgenden Bild ist die Übersicht über die erfassten Lieferadressen des Accounts ersichtlich.

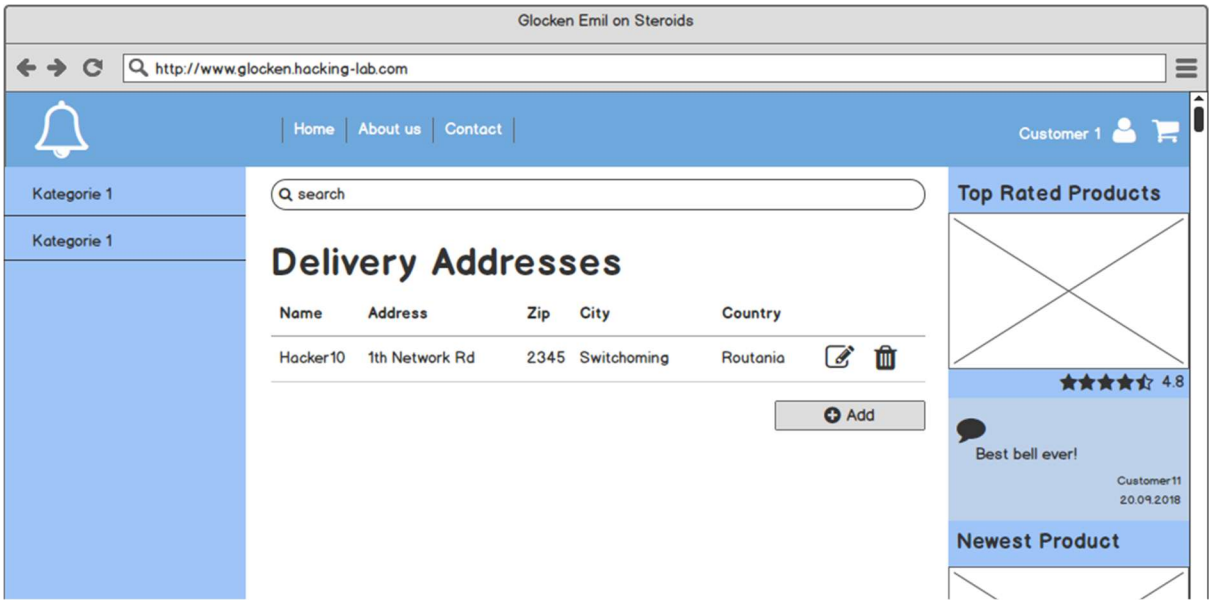


*Abbildung 51 Mockup - Überblick der Lieferadressen des Kontos*

## Post Ansicht - Hinzufügen

Es können Posts im Community Bereich erstellt werden, welche dann in der Home Ansicht angezeigt werden.

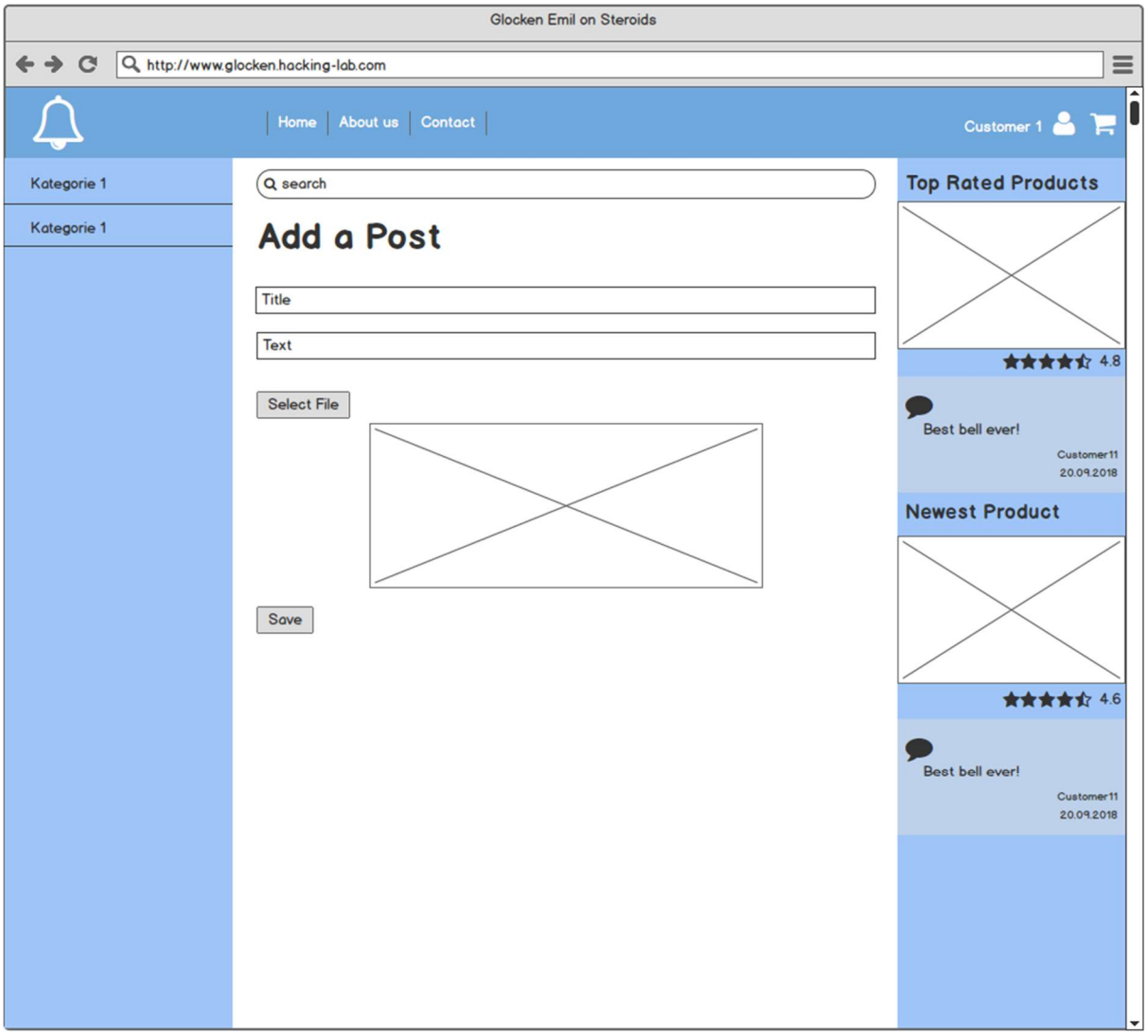


*Abbildung 52 Mockup - Erfassung eines Posts*

# Buchungen Ansicht

Nach Abschluss einer Buchung ist es möglich die erstellten Bestellungen anzuzeigen.

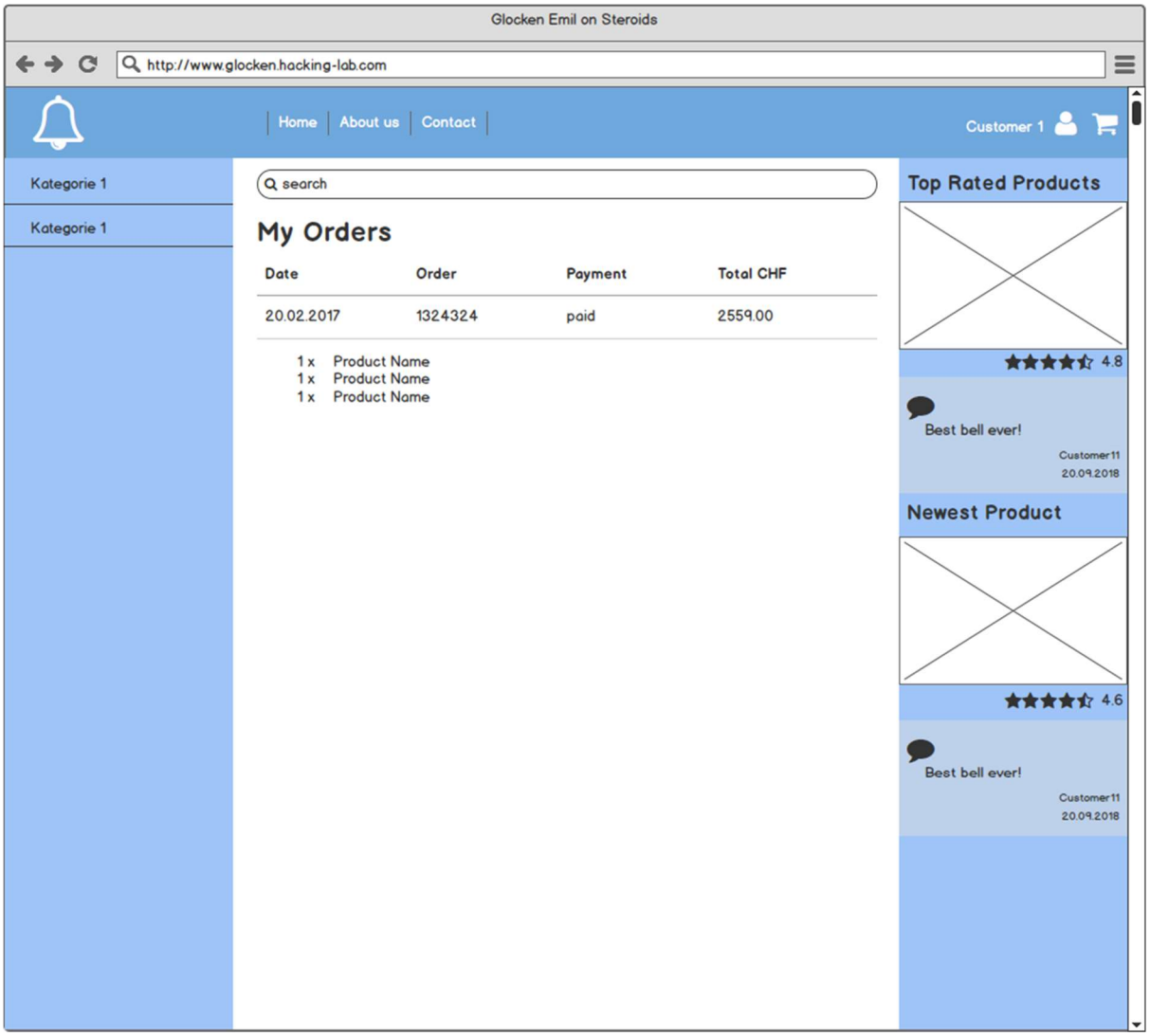


*Abbildung 53 Mockup - Übersicht über alle Buchungen*

## Retailer Ansicht

Die Retailer Ansicht, bietet einen Einblick in das API und zeigt, wie dieses aufgerufen werden kann.

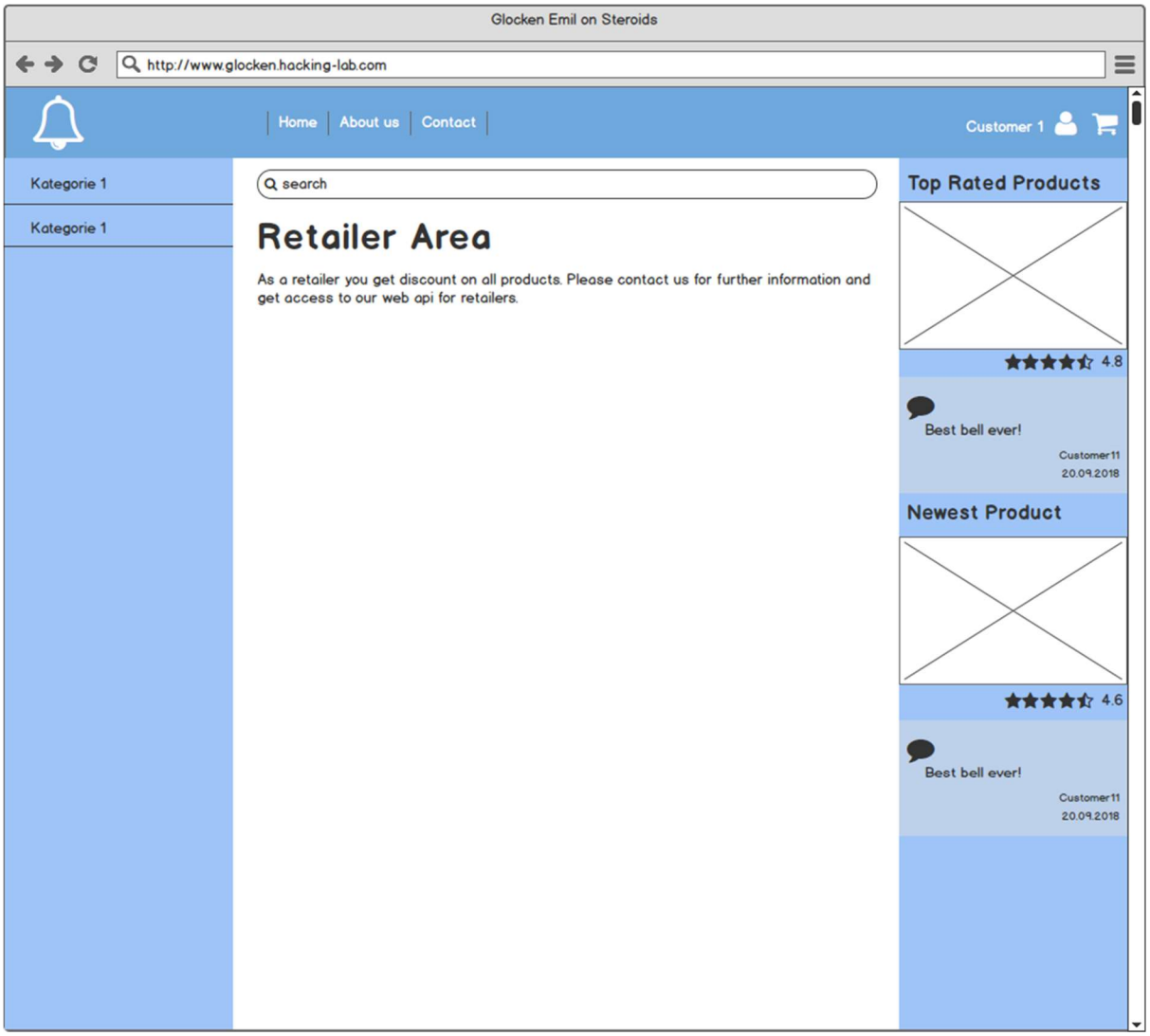


*Abbildung 54 Mockup - Retailer Area*

# 11. Anhang E: Personal Summary Patrick Steinhäusl

Aktuell bin ich bereits im Endspurt dieser Studienarbeit. Es war eine sehr spannende und auch sehr fordernde Arbeit. Trotz dieser Herausforderung habe ich mich mit vollem Einsatz in diese Arbeit gehangen. Ich hatte eher das Problem, dass ich nicht zu viel Zeit investiere, da ich ein Mensch bin, wenn mich etwas interessiert vergesse ich die Zeit. In der Planung für die Umsetzung des Webshop habe ich mich in der Zeitplanung leicht verspekuliert und somit musste ich den Meilenstein betreffend der Fertigstellung des Webshops um eine Woche verschieben. Ich kann somit sagen, dass ich für zukünftige Projekte mehr Zeit einplanen werde.

Zu Beginn der Projektarbeit hatte ich bedenken, ob ich dieser Aufgabe gewachsen bin. Ich hatte bereits einige Webanwendungen gemacht in dem Studium sowohl auch Privat. Auch die entsprechenden Module für Web Engineering und Web Security habe ich bestanden. Am Anfang hatte ich zudem Bedenken, weil ich diese Arbeit allein angehen muss, doch mein Betreuer war sehr zuvorkommend und bot jederzeit seine Hilfe an. So war ich von Zeit zu Zeit immer zuversichtlicher, dass ich diese Arbeit erledigen kann.

Ich konnte einen fast vollständigen Webshop bauen, welche nun bereits einige Schwachstellen aufweist.

Nun habe ich einen tiefen Einblick in die Materie und mein Interesse ist weiterhin voll und ganz da. Vielleicht liegt meine zukünftige Orientierung sogar in dieser Richtung.

# 12. Anhang F – Glossar

API
: Application Programming Interface — 10, 12, 13, 16, 29, 30, 31, 34, 35, 55

CSRF
: Cross Site Request Forgery — 9

HTML
: Hypertext Markup Language — 16, 33, 34

HTTP
: Hypertext Transfer Protocol — 6, 16

LDAP
: Lightweight Directory Access Protocol — 7

OWASP
: Open Web Application Security Project — 4, 5, 6, 8, 9, 10, 12, 35

REST
: Restpresentational State Transfer — 16, 30, 31, 34, 35

SVG
: Scalable Vector Graphic — 12, 31, 32, 34, 35

URL
: Unified Resource Locator — 9, 12, 13, 16, 17, 29

URLs
: Unified Resource Locator — 9

XSS
: Cross Site Scripting — 13, 32, 33, 34

# 14. Anhang G – Verzeichnisse

## Abbildungsverzeichnis

## Tabellenverzeichnis



## Quellenverzeichnis

OWASP Foundation. *OWASP*. 2007. 15. Dezember 2017. <https://www.owasp.org/index.php/Top_10_2007>.

Wikipedia. *Wikipedia*. 7. Dezember 2017. 16. Dezember 2017. <https://de.wikipedia.org/wiki/MongoDB>.

OWASP Foundation. *OWASP*. 20. Juni 2017. 5. Dezember 2017. <https://www.owasp.org/index.php/Testing_for_NoSQL_injection>.

—. *OWASP*. 4. Juni 2016. 12. Dezember 2017. <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>.

# 16. Anhang H - Urheber- und Nutzungsrechte

## Vereinbarung

## 1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit Glockenemil on Steroids von Patrick Steinhäusl unter der Betreuung von Cyrill Brunschwiler geregelt.

## 2. Urheberrecht

Die Urheberrechte stehen dem Studenten zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von dem Studenten, von der HSR wie von der Compass Security Network Computing AG nach Abschluss der Arbeit verwendet und weiterentwickelt werden.

Rapperswil, den …… ……..

Der Student, Patrick Steinhäusl

Rapperswil, den …… ……..

Der Betreuer der Studienarbeit, Cyrill Brunschwiler

# 18. Anhang I - Eigenständigkeitserklärung

## Erklärung

Ich erkläre hiermit,

- Dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welcher explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- Dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- Das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, den …… ……..

Der Student, Patrick Steinhäusl

# 20. Anhang J - Einverständniserklärung

## Einverständniserklärung Publikation auf eprints.hsr.ch

☒ SA

☐ BA

Titel der Arbeit: Glockenemil on Steroids

Team: Patrick Steinhäusl

Betreuer: Cyrill Brunschwiler

Ich bin mit der Publikation meiner Arbeit auf eprints.hsr.ch einverstanden, sofern für diese Arbeit keine Geheimhaltungsvereinbarung unterzeichnet wurde.

Nach Bekanntgabe der Note habe ich die Möglichkeit innert 14 Tagen Einsprache zu erheben und das Einverständnis zur Publikation der Arbeit auf eprints.hsr.ch zurückzuziehen. In diesem Falle wird nur der Abstract publiziert.

Rapperswil, den …… ……..

Der Student, Patrick Steinhäusl