27.4.2022

M150

E-Business-Applikation anpassen

Christopher O'Connor
BBBADEN

1 INHALTSVERZEICHNIS

2 BEWERTUNG

Phase	Bewertung	Gewichtung	Bewertete Punkte
Alle	Projektmaterial	1	• Gibt es ein zentrales Dokument das übersichtlich auf die Dokumentation verweist bzw. verlinkt? • Ist die Ordnerstruktur einleuchten? • Sind die Dokumente ansprechend gestaltet? • Ist der Projektauftrag ausgefüllt? • Gibt es Protokolle für alle Projektbesprechungen? • Ist die Rechtschreibung akzeptabel? • Abgabeformate eingehalten?
I	Ideen und Informationen	2	• Sind Besprechungsnotizen, Protokolle, Ideenskizzen vorhanden? • Wurden die richtigen Fragen erklärt und ist die Erklärung dokumentiert? • Wurde die Lernphase der benötigten Tools und Geräte dokumentiert?
P	Tätigkeitsliste	1	• Gibt es eine Tätigkeitsliste mit verantwortlichen Personen, Terminen, Abhängigkeiten? • Sind die Tätigkeiten genügend aufgeteilt, so dass nur selten mehrere Personen verantwortlich sind.
P	Terminplan	1	• Gibt es einen Terminplan? • Gibt es einen angepassten Terminplan über das Projekt und sind die Unterschiede analysiert? • Sind Verspätungen dokumentiert?
P	Situation	1	• Ist die Ausgangslage, der Ist- und der Sollzustand genau beschrieben? • Ist das eigentliche Ziel beschrieben?
P	Anforderung	2	• Sind die Anforderungen komplett? • Sind die Anforderungen verständlich?
P	Tests	2	• Gibt es Testfallspezifikationen? • Sind die Testfallspezifikationen klar formuliert? • Gibt es für jede Anforderung mindestens eine Testfallspezifikation? • Sind Eingabe und Ausgabe klar beschrieben?
E	Entscheidung	1	• Sind alle Varianten aufgeführt? • Sind alle Entscheidungen nachvollziehbar? • Es hat ein Fazit
R	Realisierung	2	• Sind Probleme und Lösungen dokumentiert? • War die Realisierung professionell (Kommentare im Code…) • Ist ersichtlich, wie die Arbeitspakete abgearbeitet wurden? • Sind alle Quellen von Texten und Bilder angegeben worden
K	Testausführung	1	• Gibt es ein Testprotokoll mit Datum, Resultaten, Bemerkungen und Namen der Tester? • Gibt es ein Testfazit zu den Tests?

3 AUFGABENSTELLUNG

3.1 AUSGANGSLAGE

Bei der Applikation handelt es sich um einen einfachen Webshop. Dieser soll nach Wunsch erweitert werden, sodass künftig bei einem neuen Kauf der Kunde eine E-Mail mit der Kaufbestätigung erhält.

3.2 AUFBAU DER APPLIKATION

Bei der Applikation handelt es sich um einen simplen Webshop, welcher mit Laravel realisiert wurde. Gäste können Produkte zu ihrem Warenkorb hinzufügen und mit fiktivem Geld bestellen.

3.3 APPLIKATIONSUMGEBUNG

Die Applikation befindet sich auf dem Entwicklersystem und wird mit Phpstorm realisiert.

3.4 RAHMENBEDINGUNGEN

Der simple Webshop wird vorab erstellt. Aus zeitlichen Gründen ist dieser auf minimale Features begrenzt.

3.5 VORGABE ANALYSIEREN

Zukünftig soll nach einer Bestellung der Kunde über den erfolgreichen Kauf via Bestätigungsmail informiert werden.

3.6 ÄNDERUNGSBEDARF

1) Die Laravel E-Mail API muss korrekt angepasst werden mit den nötigen Informationen des Maildiensts.
2) Es müssen Änderungen an einzelnen Entitäten der Datenbank sowie Datenmodels vorgenommen werden sowie die eigentliche Erweiterung in der Buisnesslogik hinzugefügt werden. Für die Realisierung gibt es dabei zwei Möglichkeiten:
   a) Eventlistener auf der Create-Order Methode
   b) Bestehende Create-Order Methode erweitern
3) E-Mail Template erstellen/anpassen

Die beiden Möglichkeiten von Teilschritt 2) werden vor der Realisierung genauer betrachtet und miteinander verglichen.

3.7 NICHTBESTANDTEIL

1. Aufsetzten eines neuen Mailservers.
2. Projektbesprechungen (Einzelprojekt)

3.8 ENTWICKLER

Christopher O'Connor

4 ORGANISATION DER ARBEITSERGEBNISSE

4.1 QUELLCODE

Der Quellcode wird in einem Git-Repository auf Github nach Erreichung von Meilensteinen / Teilzielen jeweils hochgeladen und somit versioniert.

https://github.com/chrisoco/m150

4.2 ENTWICKLUNGSPROZESS

Für dieses Projekt soll ein vereinfachter GitLab-Flow verwendet werden

Folgende Regeln gelten beim GitLab Flow:

- Es gibt keine normalen Commits auf den «master»-Branch.
- Für jede Entwicklung wird ein Eintrag im Issue-System erstellt.
- Für jede Entwicklung wird ein Seitenbranch erstellt, der mit der Issue-Nummer beginnt, also beispielsweise «15-require-a-password-to-change-it».
- Ist die Entwicklung abgeschlossen, so kann sie getestet und reviewt werden, bevor sie zurück zum «master» gemerged wird.
- Commit-Nachrichten beschreiben Absichten. Also nicht «Changed index.jsp» sondern «Changed background color of intro from gray to blue».
- Releases werden mit Tags realisiert, beispielsweise «2-3-stable».

4.3 DOKUMENTATION

Sämtliche Dokumente, welche nicht im Git abgelegt sind (Dokumentation, etc.), werden im OneDrive in der Cloud gespeichert. Zusätzlich wird am Ende des Projekts die Dokumentation dem Git-Verzeichnis hinzugefügt.

5 MITTEL UND METHODEN

5.1 TECHNOLOGIEN

TECHNOLOGIE	BESCHREIBUNG
Programmiersprachen	PHP (7.4.15), JS, (HTML5, CSS3)
Framework	Laravell (8.x)
Datenbank	MariaDB (10.4.17)
Object-Relational-Mapper (ORM)	Eloquent
Libraries	Bootstrap, jQuery (3.6)
Templating-Engine	Blade Templates

5.2 SOFTWARE

NAME	VERWENDUNGSZWECK
XAMPP (3.2.4)	Webserver & Datenbank
PhpStorm (2021.3.1)	IDE
Composer (2.2.5)	Laravel Packet-Manager
Git	Versionskontrollsystem
OneDrive	Dokumentversionskontrollsystem
MySQL Workbench	ERM Erstellung
phpMyAdmin	Datenbankmanagement
Microsoft Word	Dokumentation
Mailtrap.io	E-Mail Sandbox

5.3 LARAVEL LIBRARIES

NAME	VERWENDUNGSZWECK
Laravel Debugbar	Debugging
Eloquent Model Generator	Basis Models generieren

5.4 PROJEKTMANAGEMENT METHODE

IPERKA wird als Projektmanagement Methode per Vorgabe verwendet.

6 INFORMIEREN

6.1 LÖSUNGSMÖGLICHKEITEN

Für das Auslösen der Bestätigungs-Email gibt es zwei mögliche Lösungsvarianten:

1) Eventlistener auf dem Purchase Model beim Erstellen eines neuen Purchases.
2) Bestehende Create-Order Methode erweitern

6.1.1 1. Möglichkeit

Mittels einem Observer kann, wenn ein neuer Kauf erstellt wird eine Notifikation abgefeuert werden.

Vorteil:

- Die bestehende Businesslogik muss nicht angepasst werden.

Nachteil:

- Für nicht Laravel-Kenner ist dies unter Umständen nicht Nachvollziehbar wo der Aufruf geschieht und erschwert die Leserlichkeit des Codes.

6.1.2 2. Möglichkeit

Bestehende Businesslogik erweitern mit einem manuellen Aufruf der Notifikation.

Vorteil:

- Lesbarer Code
- Einfachere Implementierung

Nachteil:

- Die Methode, welche für das Erstellen eines neuen Kaufs zuständig ist, ist nicht mehr nur für das Erstellen, sondern auch für das Senden der Notifikation zuständig.

6.2 LERNPHASE

6.2.1 Mailtrap

Mailtrap ist ein E-Mail-Sandbox Service mit welchem, ausgehende E-Mails getestet werden können ohne einen Mailserver verwenden zu müssen. Ein Konto wird benötigt um ein API-Username sowie Passwort zu erhalten.

Um Mailtrap verwenden zu können muss lediglich die «.env» Datei mit folgenden Einträgen ergänzt werden:

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=<API_USERNAME>
MAIL_PASSWORD=<API_PASSWORT>
MAIL_ENCRYPTION=tls
```

Anschliessend können Mails wie für Laravel üblich versendet werden und auf mailtrap.io in der Inbox analysiert werden.

7 PLANEN

7.1 IST-ZUSTAND

Beim System handelt es sich um eine PHP-Webapplikation, welche das Framework Laravel verwendet und befindet sich auf der lokalen Entwicklungsumgebung des Entwicklers. Als relationale Datenbank wird MariaDB vom Programm XAMPP verwendet sowie dessen Apache Web Server. E-Mail-Notifikationen werden mittels Mailtrap.io abgefangen und überprüft. Bei der Applikation handelt es sich um einen einfachen Webshop in welchem Artikel einem Warenkorb hinzugefügt werden und gekauft werden können.

7.2 SOLL-ZUSTAND

Künftig soll nach dem Kauf eine Bestätigungsemail an den Kunden versendet werden mit einer Übersicht der gekauften Artikel. Zusätzlich sollen die Benutzer aufgefordert werden ihre E-Mail-Adresse mittels einem link zu bestätigen.

7.3 AUFTRAGSDIVISION

Die zu realisierenden Teilaufgaben werden für die Planung Stichwortartig chronologisch aufgelistet:

1) Purchase-History
   a) Datenbank erweitern
   b) Datensatz beim Kauf erstellen
2) E-Mail Notifikation
   a) Mailtrap.io einrichten
   b) Notifikation erstellen
   c) E-Mail Blade-Templates erstellen
   d) E-Mail versenden
3) E-Mail Bestätigung
   a) E-Mail-Adresse muss nach Registrierung bestätigt werden

7.3.1 Tätigkeitsliste

Auftrag	Abhängigkeit	Verantwortlich
#1a	-	COC
#1b	#1a	COC
#2a	-	COC
#2b	-	COC
#2c	#2b	COC
#2d	#2a, #2b, #2c	COC
#3a	#2	COC

Die Auflistung erleichtert den Überblick der zu erledigenden Aufgaben im teil Realisierung. Beachtet wurde dabei eine Sinnvolle Reihenfolge einzuhalten durch mögliche Abhängigkeiten.

7.4 TERMINPLAN

Projektphase	Geplanter Termin	Beendet am
Informieren	02.02.22	02.02.22
Planen	23.02.22	23.02.22
Entscheiden	02.03.22	02.03.22
Realisieren	23.03.22	21.04.22
Kontrollieren	30.03.22	26.04.22
Auswerten	06.04.22	27.04.22

7.5 PROJEKTDATENSTRUKTUR

Dieses Projekt wird mit dem Framework Laravel umgesetzt. Entsprechend werden die Normen und Strukturen von Laravel eingehalten.

7.6 MOCKUP

Ein Mockup für die E-Mail Notifikation wurde auf Papier erstellt um während der Realisierung Bezug dazu nehmen zu können:

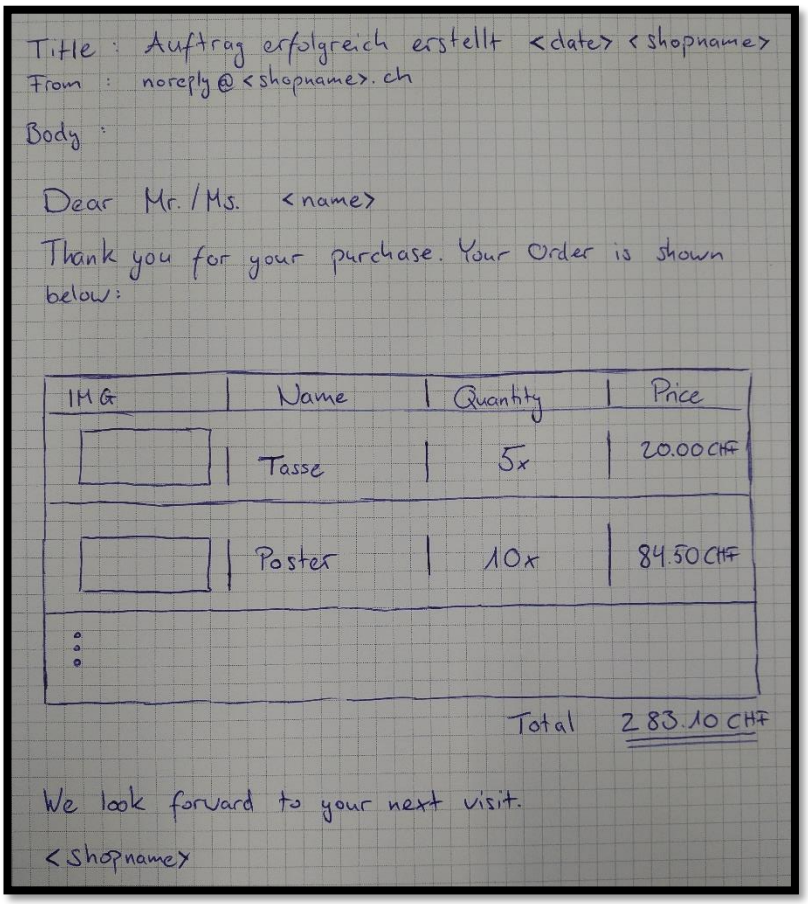


Abbildung 1 Bestätigungs-E-Mail Mockup

7.7 ERD Datenbankmodell

Die Datenbank muss erweitert werden für eine Purchase-History. Aktuell besteht die Datenbank nur aus den Users sowie den Items welche gekauft werden können.

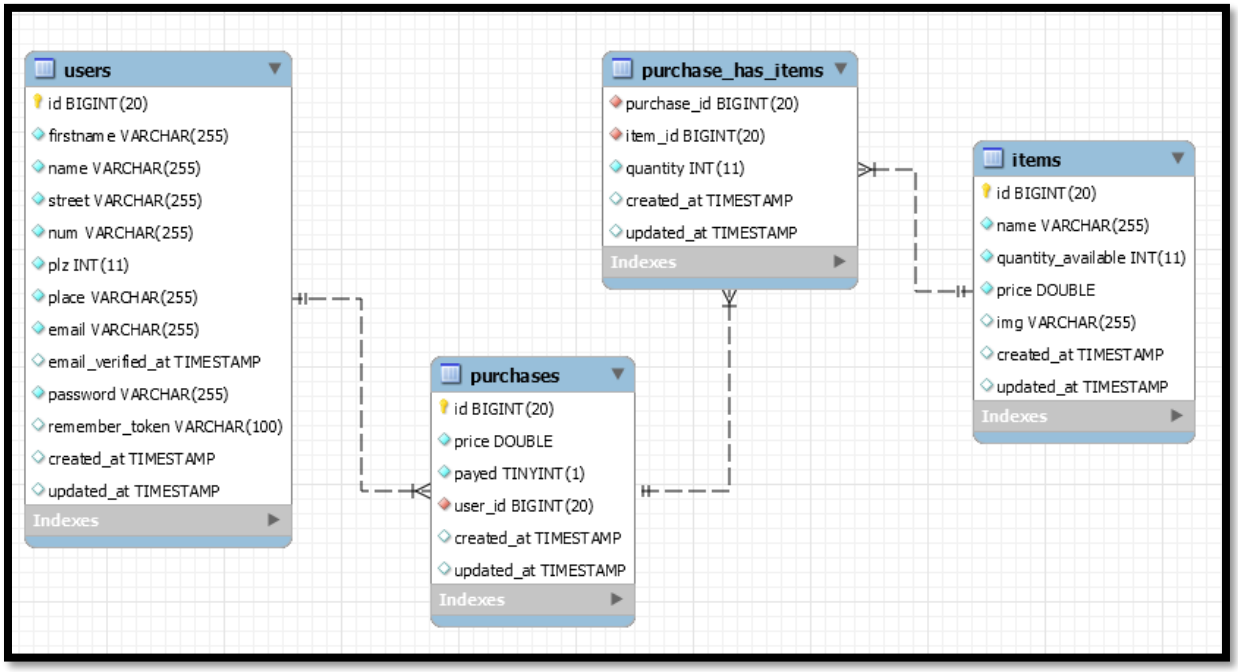


Abbildung 2 ERD Datenbankmodel

Ein Einkauf wurde hinzugefügt. Dieser beinhaltet den Kaufzeitpunkt, den Preis sowie ob die Transaktion der Bezahlung erfolgreich war. Dem Einkauf werden alle gekauften Items hinzugefügt durch die n:m Verbindungstabelle «purchase_has_items» inklusive der Quantität.

7.8 Erkannte Risiken

Die erkannten Risiken werden nachfolgend aufgeführt mit entsprechenden Lösungsvorschlägen.

7.8.1 Mailtrap

Mailtrap ist ein ukrainisches Unternehmen. Mit der aktuellen russischen Invasion könnte der Dienst allenfalls ausfallen und nicht mehr verfügbar sein.

7.8.1.1 Mögliche Alternative

Ein aus einem anderen Projekt verwendeter Mailserver kann nötigenfalls für Testzwecke verwendet werden. Dazu muss lediglich in der «.env» Datei die Maileinstellungen angepasst werden.

7.9 TESTKONZEPT

Das zu testende System ist eine PHP-Webapplikation, welche das Framework Laravel verwendet und befindet sich auf der lokalen Entwicklungsumgebung des Entwicklers. Bei der Applikation handelt es sich um einen einfachen Webshop. Dieser soll nach Wunsch erweitert werden, sodass künftig bei einem neuen Kauf der Kunde eine E-Mail mit der Kaufbestätigung erhält. Als relationale Datenbank wird MariaDB vom Programm XAMPP verwendet sowie dessen Apache Web Server. E-Mail-Notifikationen werden mittels Mailtrap.io abgefangen und überprüft.

7.9.1 Überprüfung

Alle Testfälle werden manuell vom Entwickler der Reihe nach durchgeführt.

7.9.2 Testmittel

Das relevante Testmittel besteht aus der lokalen Entwicklungsumgebung mit einer bestehenden Internetverbindung und einer neu erstellter sowie geseedeter MariaDB Datenbank. Der Apache Webserver sowie die Datenbank werden vom Programm XAMPP zur Verfügung gestellt. Getestet wird mit dem Google Chrome Web-Browser v.99.

7.9.3 Teststufen

Die nachfolgenden Teststufen werden als Black-Box getestet:

1. Unittests:
   Testen kleinst-möglicher testbarer Funktionalitäten isoliert von anderen.
2. Integrationstests:
   Test der Funktionalität bei der Zusammenarbeit voneinander abhängiger Komponenten.

7.9.4 Allgemeine Testvoraussetzungen

1. Der Source code der Applikation befindet sich im Verzeichnis «C:\xampp\htdocs»
2. XAMPP ist gestartet sowie der Apache und MySQL dienst sind am Laufen.
3. Mailtrap Key und Secret sind in der Datei «.env» korrekt angegeben.

7.9.5 Testfallspezifikation

Die Testfälle sind nicht nach den Teststufen gegliedert, sondern nach dem Applikationsdurchlauf und bauen aufeinander auf. Diesbezüglich müssen die Testfälle der Reihe nach durchgegangen werden. Kommt es zu einem schwerwiegenden Fehler, muss der Testdurchlauf abgebrochen, evaluiert und allenfalls nötige Änderungen vorgenommen werden. Anschliessend muss der Testdurchlauf von vorne neu begonnen werden.

Testfall-Nr.	**#1**
Anforderung	Login
Beschreibung	Als Käufer will ich mich Anmelden können um meinen Warenkorb speichern zu können und einen kauf tätigen können.
Testvoraussetzung	1. Nicht angemeldet sein 2. URL: http://localhost/m150/app/public/ aufrufen
Eingabe	1. E-Mail: admin@appli.ch 2. Passwort: 123 3. Button «Login» klicken
Erwartetes Resultat	Der Benutzer wird eingeloggt und auf die URL: http://localhost/m150/app/public/ weitergeleitet.

Testfall-Nr.	#2
Anforderung	Logout
Beschreibung	Ein Benutzer kann sich von der Applikation abmelden.
Testvoraussetzung	1. Angemeldet sein
Eingabe	1. Button «Logout» in der Navigationsleiste klicken
Erwartetes Resultat	User wird Ausgeloggt und auf http://localhost/m150/app/public/ weitergeleitet.

Testfall-Nr.	#3
Anforderung	Login: Falsche Angaben
Beschreibung	Login mit falschen Angaben darf nicht möglich sein.
Testvoraussetzung	1. Nicht angemeldet sein 2. URL: http://localhost/m150/app/public/ aufrufen
Eingabe	1. Anmeldeformular Eingabe: a. E-Mail: admin@appli.ch b. Password: asdf 2. Button «Login» klicken
Erwartetes Resultat	Fehlermeldung «E-Mail oder Passwort falsch» erscheint.

Testfall-Nr.	#4
Anforderung	Gegenstand Warenkorb hinzufügen
Beschreibung	Gegenstände können dem Warenkorb hinzugefügt werden.
Testvoraussetzung	1. Angemeldet sein 2. URL: http://localhost/m150/app/public/ aufrufen
Eingabe	1. Stückzahl eingeben 2. Button «Add to cart» klicken
Erwartetes Resultat	Gegenstand wird dem Warenkorb auf der rechten Seite hinzugefügt.

Testfall-Nr.	#5
Anforderung	Gegenstand Warenkorb hinzufügen mit ungültiger Stückzahl
Beschreibung	Ungültige Stückzahl (mehr als vorhanden) darf nicht möglich sein.
Testvoraussetzung	1. Angemeldet sein 2. URL: http://localhost/m150/app/public/ aufrufen
Eingabe	1. Stückzahl eingeben = 300 2. Button «Add to cart» klicken
Erwartetes Resultat	Fehlermeldung erscheint und der Gegenstand wird dem Warenkorb **nicht** hinzugefügt.

Testfall-Nr.	#6
Anforderung	Check-Out
Beschreibung	Übersicht des aktuellen Warenkorbs mit allen Gegenständen sowie Stückzahl.
Testvoraussetzung	1. Angemeldet sein 2. URL: http://localhost/m150/app/public/ aufrufen
Eingabe	1. Button «Check-Out» klicken unterhalb des Carts
Erwartetes Resultat	Weiterleitung auf http://localhost/m150/app/public/checkout. Alle zuvor hinzugefügten Gegenstände sind aufgelistet mit der richten Anzahl Stück.

Testfall-Nr.	#7
Anforderung	Kauf
Beschreibung	Alle Gegenstände im Warenkorb werden gekauft.
Testvoraussetzung	1. Angemeldet sein 2. URL: http://localhost/m150/app/public/checkout aufrufen 3. Gegenstände sind im Cart vorhanden
Eingabe	1. Button «Check-Out» klicken
Erwartetes Resultat	Der Einkauf wird getätigt und ein Bestätigungs-E-Mail wird versendet in welcher die gekauften Gegenstände aufgelistet sind.

Testfall-Nr.	#8
Anforderung	E-Mail muss verifiziert sein
Beschreibung	E-Mail Adresse muss vom User verifiziert worden sein bevor er einen Kauf tätigen kann.
Testvoraussetzung	1. Angemeldet sein ohne E-Mail verifizierung 2. URL: http://localhost/m150/app/public/checkout aufrufen
Eingabe	-
Erwartetes Resultat	Weiterleitung auf http://localhost/m150/app/public/email/verify.

8 ENTSCHEIDEN

8.1 LÖSUNGSMÖGLICHKEIT NOTIFIKATION

Es wurde entschieden, die 2. Lösungsmöglichkeit zu implementieren. Durch die Erweiterung der bisherigen Businesslogik wird der Code viel Lesbarer und der Ablauf ist nachvollziehbarer.

8.2 MASSNAHMEN DER RISIKEN

8.2.1 Mailtrap

Mailtrap wird weiterführend verwendet. Der Maildienst wird nur während dem Modul benötigt, um den Mailverkehr zu Testen und allenfalls zu Demonstrieren.

8.3 FAZIT

Durch die Erweiterung einer bestehenden Applikation sind die Technologien vorgegeben und werden weitergeführt. Das System wird mittels der E-Mail-Notifikationen, welche durch die 2. Lösungsmöglichkeit beschrieben, in der bestehenden Businesslogik erweitert. Aktuell können die ausgehenden E-Mails mittels Mailtrap abgefangen und analysiert werden. Es kann mit der Implementierung der Änderungen fortgefahren werden.

9 REALISIEREN

9.1 PURCHASE-HISTORY

9.1.1 Datenbank erweitern

Damit die Einkäufe getrackt werden können, müssen diese in der Datenbank gespeichert werden. Dazu werden zwei neue Tabellen erstellt mittels Laravel Migrations.

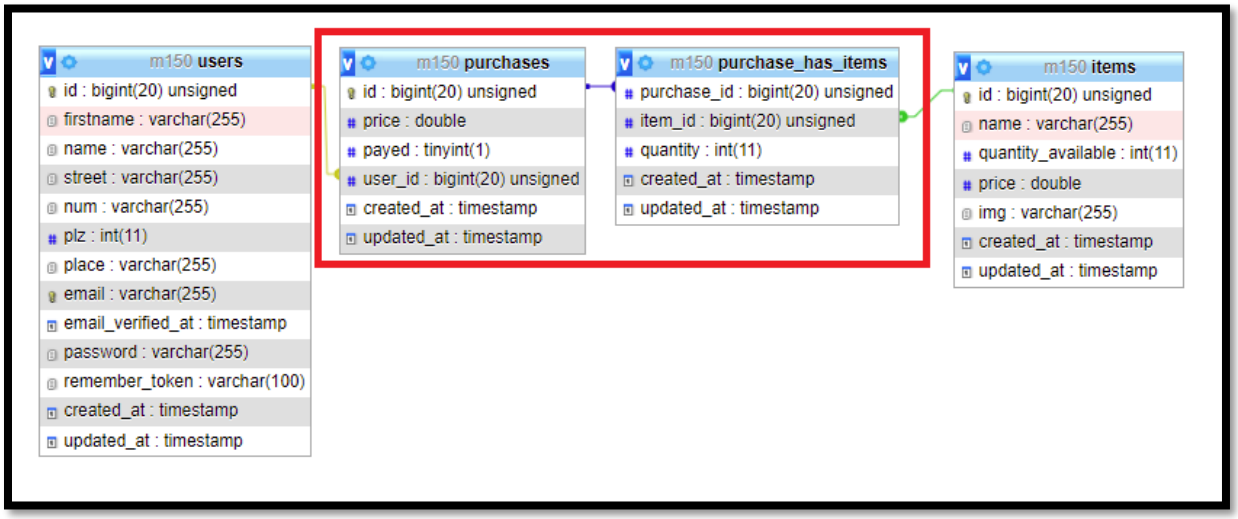


Abbildung 3 ERD Datenbankmodel

Die Tabelle «purchases» beinhaltet die Informationen zu einem Kauf und die n:m Tabelle «purchase_has_items» enthält alle Artikel des Einkaufes. In Laravel werden Änderungen in der Datenbank mittels Migrations erstellt. Diese sehen beispielsweise wie nach folglich aus:

```php
public function up()
{
    Schema::create( table: 'purchases', function (Blueprint $table) {
        $table->bigIncrements( column: 'id');
        $table->double( column: 'price');
        $table->boolean( column: 'payed')->default( value: false);
        $table->unsignedBigInteger( column: 'user_id');
        $table->foreign( columns: 'user_id')->references( columns: 'id')->on( table: 'users');
        $table->timestamps();
    });
}
```

Abbildung 4 Laravel Migration Tabelle purchases

Für die Beiden Tabellen wurden zusätzlich die Entsprechenden Models erstellt. Diese beinhalten die einzelnen Spalten sowie Relations:

```php
class Purchase extends Model
{
    use HasFactory;
    /**
     * The "type" of the auto-incrementing ID.
     *
     * @var string
     */
    protected $keyType = 'integer';

    /**
     * @var array
     */
    protected $fillable = ['user_id', 'price', 'payed', 'created_at', 'updated_at'];

    /**
     * @return \Illuminate\Database\Eloquent\Relations\BelongsTo
     */
    public function user()
    {
        return $this->belongsTo( related: 'App\Models\User');
    }

    /**
     * @return \Illuminate\Database\Eloquent\Relations\HasMany
     */
    public function purchaseHasItems()
    {
        return $this->hasMany( related: 'App\Models\PurchaseHasItems');
    }
}
```

Abbildung 5 Purchases Model

9.1.2 Datensatz beim Kauf erstellen

Um einen neuen Eintrag zu erstellen für einen Kauf, wird direkt in der «buy()» Methode ein neuer Purchase erstellt:

```php
public function buy()
{

    $failed = [];

    if(session( key: 'c') != null && count(session( key: 'c')) > 0) {

        $newPurchase = Purchase::create([
            'price'   => '0',
            'payed'   => '0',
            'user_id' => auth()->id(),
        ]);
```

Abbildung 6 Purchase erstellen

Anschliessend wird jeder Artikel im Warenkorb Iteriert und dem Purchase hinzugefügt sowie der Gesamtpreis errechnet:

```php
// CalcPriceTotal
$total = 0;
foreach (session( key: 'c') as $key => $value) {

    $item = Item::find($key);

    if(Item::find($key)->buy($value)) {

        $total += $item->calcPrice($value);

        PurchaseHasItems::create([
            'purchase_id' => $newPurchase->id,
            'item_id'     => $item->id,
            'quantity'    => $value,
        ]);

    } else {
        $failed[] = Item::find($key);
    }

}
```

Abbildung 7 Errechnung des Gesamtpreises

Zu Letzt wird der Totalpreis vom Purchase aktualisiert sowie der Bezahlstatus auf 1 gesetzt:

```php
$newPurchase->update([
    'price'    => $total,
    'payed'    => '1',
]);
```

Abbildung 8 Preis vom Purchase anpassen

9.2 E-Mail Notifikation

E-Mails können im Laravel auf verschiedene Arten erstellt und versendet werden. In dieser Applikation werden Notifikationen verwendet. Diese können direkt von einem User-Objekt aus Versendet werden.

9.2.1 Notifikation erstellen

Eine neue Notifikation kann mittels dem Konsolenbefehl «php artisan make:notification <name>» erstellt werden. Diese muss nun folglich angepasst werden:

Damit die Notification die Informationen aus dem Purchase kennt, muss dieser übergeben werden. Diesbezüglich wird im Konstruktor der Purchase erwartet:

```php
class PurchaseNotification extends Notification implements ShouldQueue
{
    use Queueable;

    private $purchase;

    /**
     * Create a new notification instance.
     *
     * @return void
     */
    public function __construct(Purchase $purchase)
    {
        $this->purchase = $purchase;
    }
```

Abbildung 9 E-Mail Notification Konstruktor

In der vordefinierten Methode toMail wird die Mail Nachricht erstellt. Es sind auch wiederum hier mehrere Möglichkeiten verfügbar. Damit die Emails auch für das Auge attractiv sind, wird eine Bladeview zurückgegeben. Diese View kann mittels CSS gestylt werden und wird im E-Mail Inbox auch angezeigt. Zusätzlich kann in der BladeView auch php code ausgeführt werden und diesbezüglich die E-Mail dynamischen content enthalten:

```php
public function toMail($notifiable)
{
    return (new MailMessage)
        ->markdown( view: 'vendor.notifications.purchase', ['purchase' => $this->purchase]);
}
```

Abbildung 10 Notification Kanal

9.2.2 E-Mail Blade-Templates erstellen

Die E-Mail Templates müssen aus dem Vendor Ordner vom Laravel zuerst Publiziert werden um diese zu bearbeiten können. Dies kann mittels dem nachkommendem Befehlen erreicht werden:

«php artisan vendor:publish --tag=laravel-notifications»

«php artisan vendor:publish --tag=laravel-mail»

Für die PurchaseNotifikation wird ein neues Bladefile im Ordner «resources/views/vendor/notifications/» erstellt. In diesem wird die E-Mail Nachricht dynamisch hinterlegt:

```
@component('mail::message')


Dear Mr./Ms. {{ $purchase->user->name }}

Thank you for your purchase. Your Order is shown below:


<table style="...">
    <tr>
        <th>IMG</th>
        <th>Name</th>
        <th>Quantity</th>
        <th>Single Price</th>
    </tr>
    @foreach($purchase->purchaseHasItems as $hasItem)
        <tr>
            <td style="..."><img src="{{ url('/img/'.$hasItem->item->img) }}" class="mx-auto d-block" style="..."></td>
            <td style="...">{{ $hasItem->item->name }}</td>
            <td style="...">{{ $hasItem->quantity }}x</td>
            <td style="...">{{ $hasItem->item->price }}</td>
        </tr>
    @endforeach
</table>
<br><br>
<div><b>Total: <span style="...">{{ $purchase->price }} CHF</span></b></div>


<br><br>
We look forward to your next visit,<br>
{{ config('app.name') }}

@endcomponent
```

Abbildung 11 Notification Template

9.2.3 E-Mail versenden

Um E-Mails vom einem User aus versenden zu können muss dem User-Model das Trait Notifiable hinzugefügt werden:

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
```

Abbildung 12 Notifiable Trait hinzufügen

Um E-Mails als Notifikationen versenden zu können muss nun lediglich die Businesslogik bei der «buy()» Methode erweitert werden mit dem Aufruf der Notifikation:

```
auth()->user()->notify(new PurchaseNotification($newPurchase));
```

Abbildung 13 Notification versenden

9.3 E-Mail Bestätigung

Damit die E-Mail-Adresse nach einer Registrierung bestätigt werden muss, muss auf dem User-Model das Interface «MustVerifyEmail» hinzugefügt werden:

```
class User extends Authenticatable implements MustVerifyEmail
{
    use HasApiTokens, HasFactory, Notifiable;
```

Abbildung 14 MustVerifyEmail Interface

Ein neuer VerificationController wird ausserdem zusätzlich benötigt:

```
class VerificationController extends Controller
{
    /*
    |--------------------------------------------------------------------------
    | Email Verification Controller
    |--------------------------------------------------------------------------
    |
    | This controller is responsible for handling email verification for any
    | user that recently registered with the application. Emails may also
    | be re-sent if the user didn't receive the original email message.
    |
    */

    use VerifiesEmails;

    /**
     * Where to redirect users after verification.
     *
     * @var string
     */
    protected $redirectTo = RouteServiceProvider::HOME;

    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware( middleware: 'auth');
        $this->middleware( middleware: 'signed')->only( methods: 'verify');
        $this->middleware( middleware: 'throttle:6,1')->only( methods: 'verify', 'resend');
    }
}
```

Abbildung 15 Verification Controller

Alle Routes können nun zusätzlich mit der Middleware «verified» umschlossen werden, bei welchen ein User die E-Mail-Adresse bestätigt haben muss:

```
Route::group(['middleware' => ['auth', 'verified']], function() {

    Route::get( uri: '/checkout', [App\Http\Controllers\ShopController::class, 'checkout'])->name( name: 'checkout');
    Route::get( uri: '/buy', [App\Http\Controllers\ShopController::class, 'buy'])->name( name: 'buy');

});
```

Abbildung 16 Route Middleware verified

Um die E-Mail-Adresse zu bestätigen nach dem Registrieren wird die Registriermethode erweitert im Auth/RegisterController:

```
protected function create(array $data)
{
    $user = User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);

    event(new Registered($user));

    return $user;
}
```

Abbildung 17 Verify E-Mail versenden beim Registrieren

Mittels des neuen Events wird beim Registrieren eine neue E-Mail via Notification an den erstellen Benutzer versendet. Im E-Mail ist ein Link zur Verifizierung enthalten.

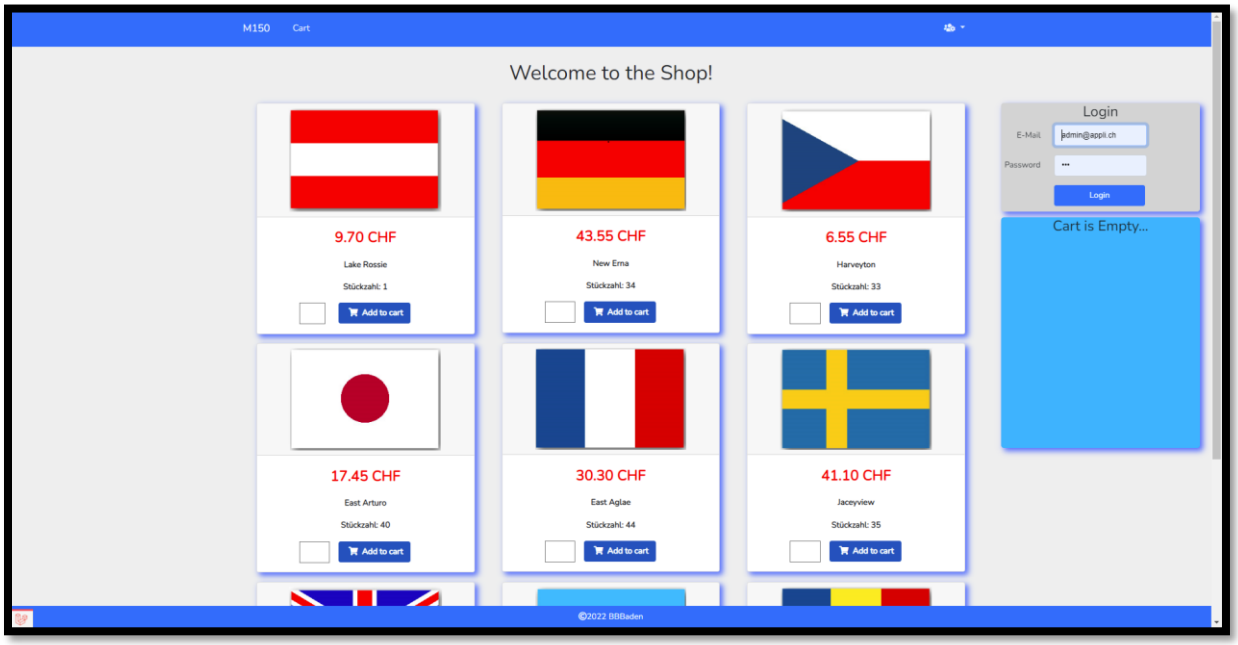
Damit die Verifizierungen im Backend an den richtigen Ort gelangen müssen letztlich folgende drei Routes hinzugefügt werden:

```
Route::get( uri: '/email/verify', action: 'VerificationController@show')->name( name: 'verification.notice');
Route::get( uri: '/email/verify/{id}/{hash}', action: 'VerificationController@verify')->name( name: 'verification.verify')->middleware(['signed']);
Route::post( uri: '/email/resend', action: 'VerificationController@resend')->name( name: 'verification.resend');
```

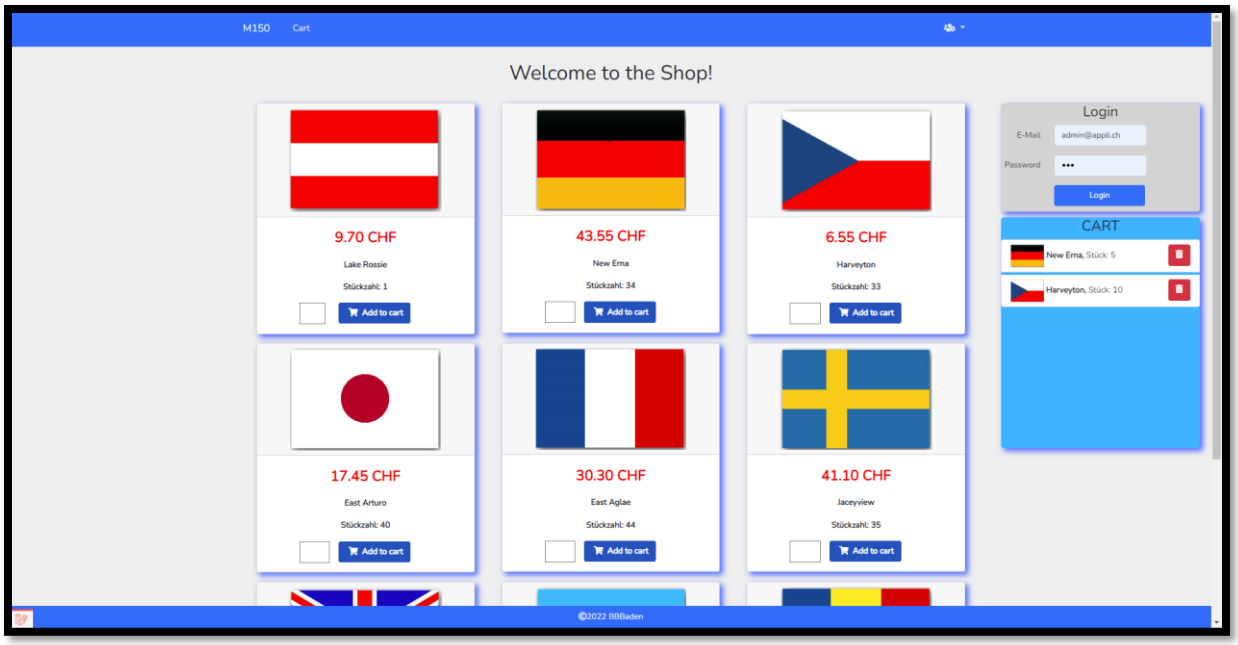
Abbildung 18 E-Mail verifizieren Routes

9.4 ERGEBNIS

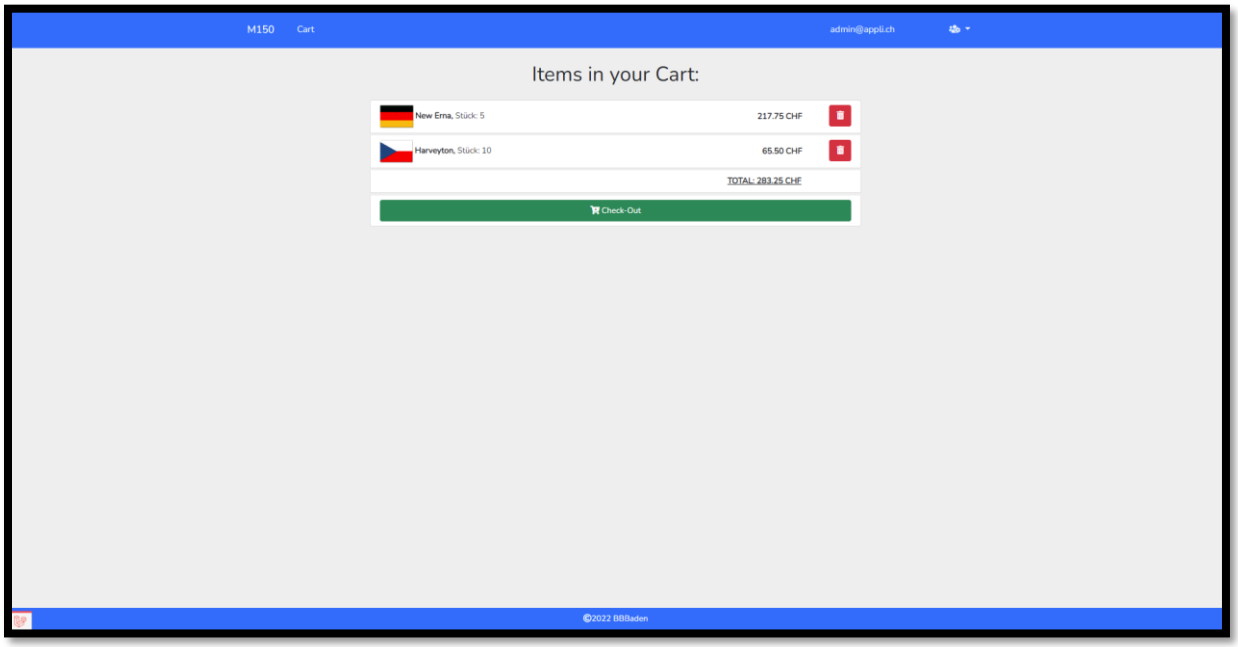
9.4.1 Index



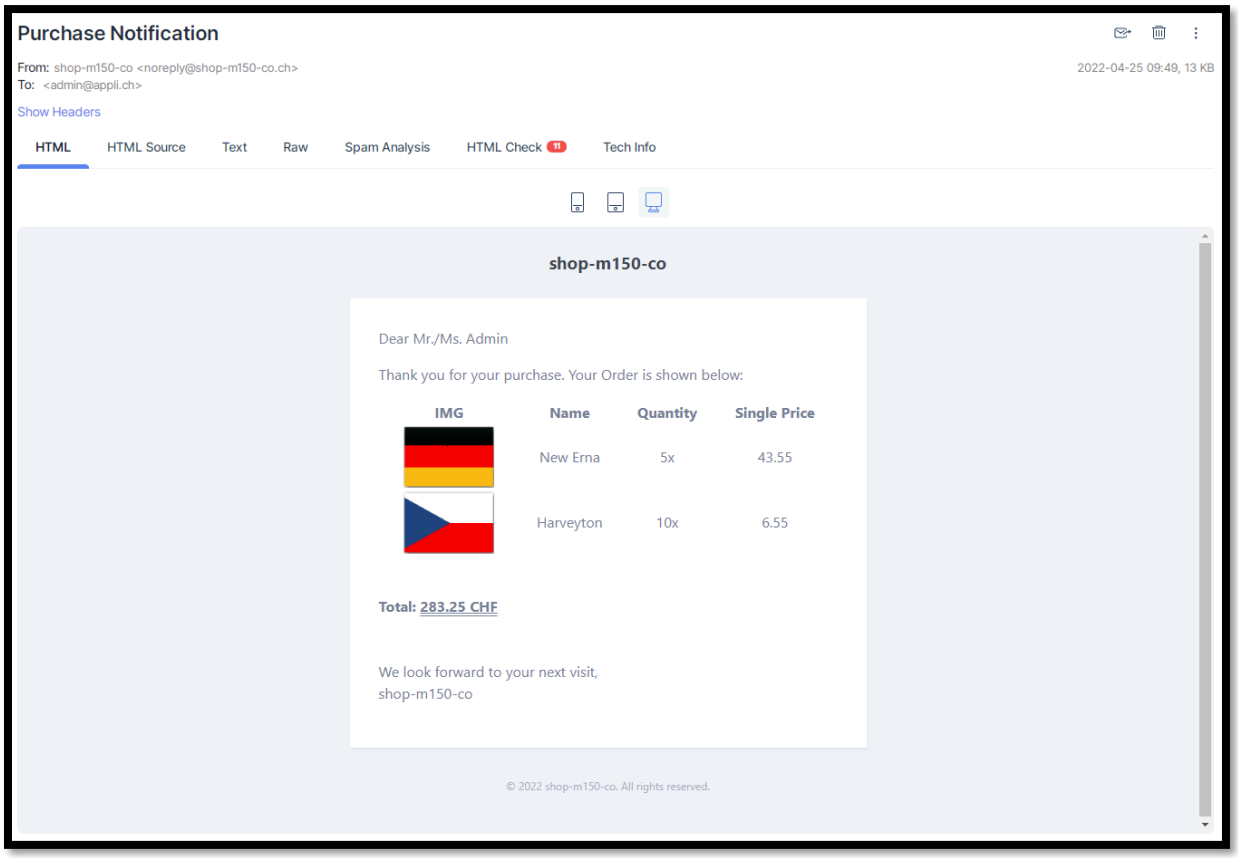
9.4.2 Cart



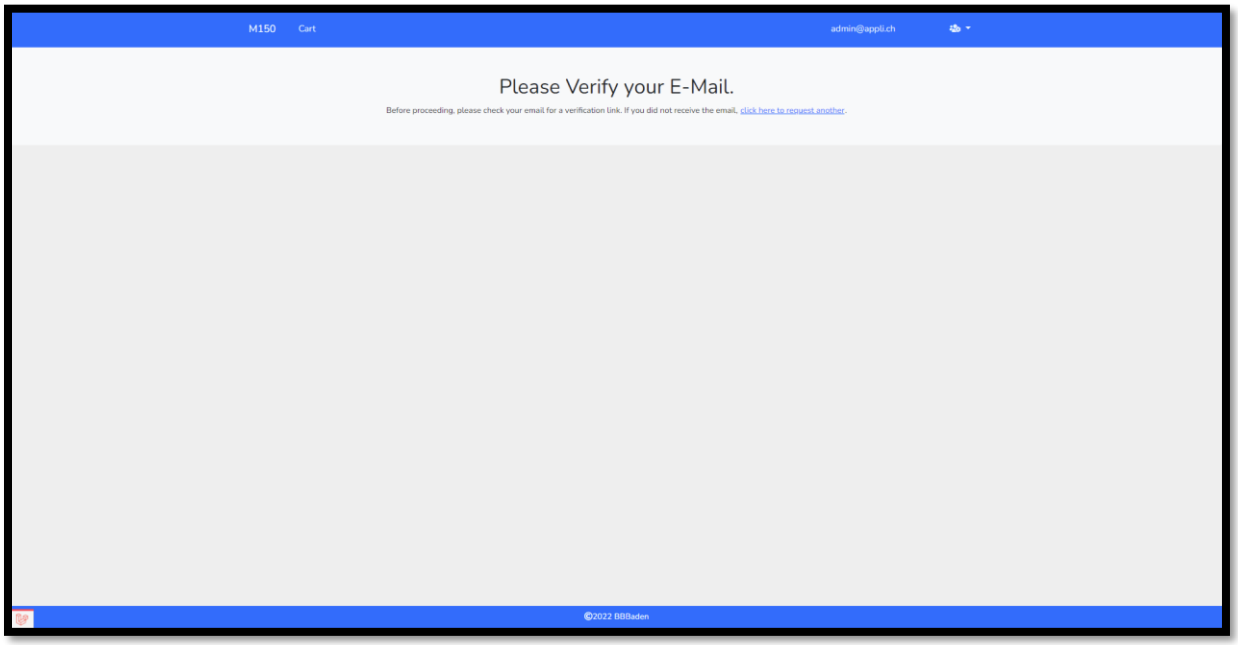
9.4.3 Checkout



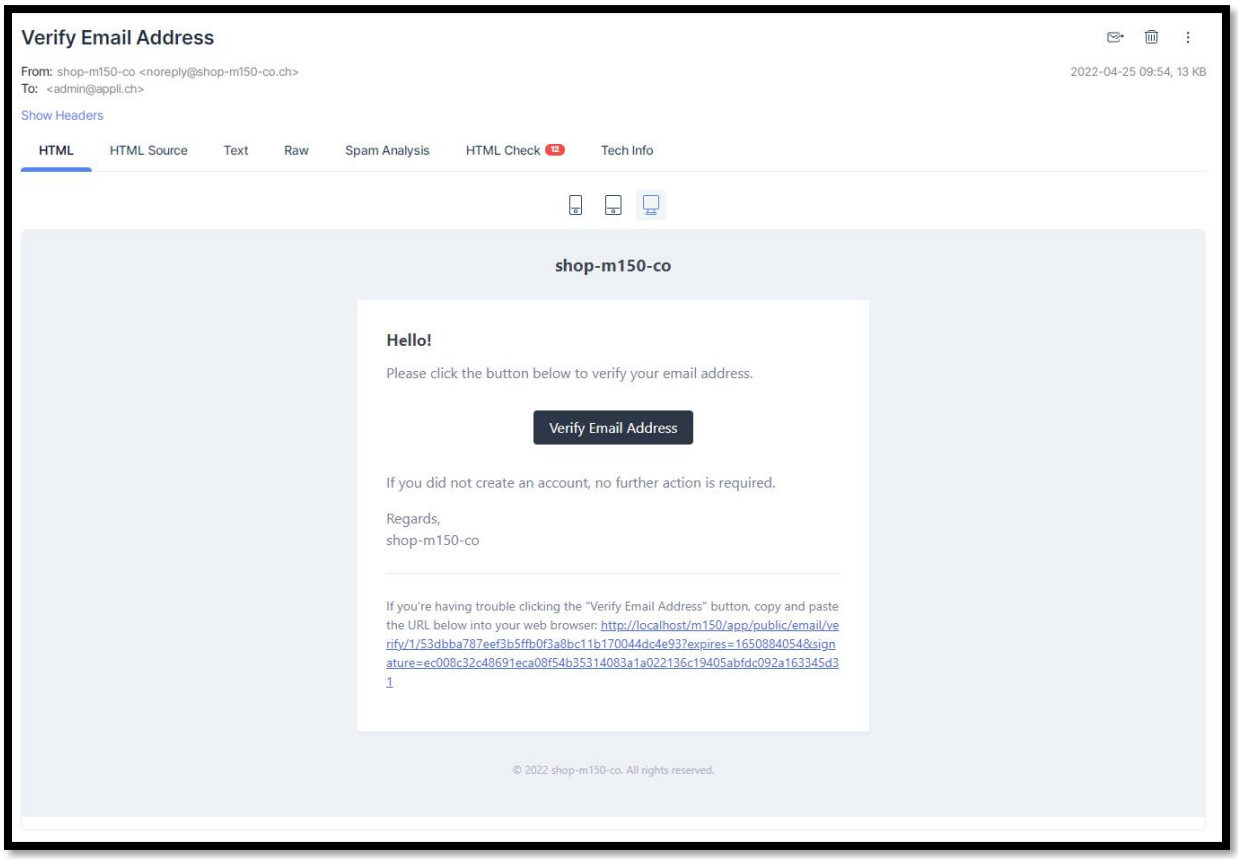
9.4.4 Purchase Bestätigungs-E-Mail



9.4.5 E-Mail noch nicht verifiziert



9.4.6 Verifikation E-Mail



10 KONTROLLIEREN

10.1 TESTPROTOKOLL 01

Getestet wurde am 26.04.2022 vom Entwickler Christopher O'Connor.

Testfall Nr.	Resultat	Bemerkung	Unterschrift
#1	Erfüllt	-	COC
#2	Erfüllt	-	COC
#3	Erfüllt	-	COC
#4	Erfüllt	-	COC
#5	Erfüllt	-	COC
#6	Erfüllt	-	COC
#7	Erfüllt	-	COC
#8	Erfüllt	-	COC

10.1.1 Massnahmen

Das erwartete und tatsächliche Resultat aller Testfälle stimmte überein. Es müssen keine Massnahmen getroffen werden.

10.2 TESTFAZIT

Durch ausgiebiges Testen während dem realisieren konnten Fehler frühzeitig erkannt werden und diesbezüglich gab es keine unerwarteten Fehler während dem Testen.

Aufgrund der erfolgreichen Testfallüberprüfung kann ausgesagt werden, dass die Applikation und die durchgeführten Änderungen alle Erwartungen erfüllen und eingesetzt werden kann.

11 AUSWERTEN

11.1 PLANUNGSDISKREPANZ

Die Planung der Projektphasen wurde vor der Bekanntgabe des IPA-Durchführungszeitraums eingeplant. Die Phasen Informieren, Planen und Entscheiden konnten innert den geplanten Terminen abgeschlossen werden. Die restlichen Phasen wurden verschoben und wurden nach der IPA nachgeholt um genügend Zeit für die IPA sowie andere in diesem Zeitraum angestandene wichtige Prüfungen zu finden.

11.2 FAZIT

Durch Zeitliche Knappheit musste die Durchführung dieses Moduls nach Priorisierung etwas verschoben werden. Die Verschiebung hat sich jedoch meiner Meinung nach gelohnt und war zudem auch geplant um den Abgabetermin nicht zu verpassen. Ich habe mich nach den Kriterien Orientiert und sozusagen ein «Kriteriendriven-development» verfolgt um mich auf das tatsächlich Wichtige zu beschränken. Bei zukünftiger zeitlicher Knappheit werde ich dies wieder tun. Allen in allem bin ich zufrieden mit meiner Arbeit und dem beenden des letzten Moduls dieser Lehre.

12 ABBILDUNGSVERZEICHNIS