SCR4x Projektarbeit SS 21

☒ **Gruppe 1** DI (FH) G. Horn-V., MSc

☐ **Gruppe 2** DI (FH) I. Krammer

Abgabetermin: Sonntag, 06.06.2021, 24:00 Uhr

Name: Andreas Wenzelhuemer

Aufwand (h): 15

Produktbewertungsportal mit PHP und MySQL

Zu entwickeln ist ein einfaches Produktbewertungsportal, in dem registrierte Benutzer Bewertungen für eine Reihe von Produkten abgeben können, um beliebigen Besuchern des Portals so z. B. bei Kaufentscheidungen zu helfen. Dabei sind die nachstehend beschriebenen Anforderungen zu berücksichtigen bzw. vollständig umzusetzen.

Funktionale Anforderungen

- Das Durchsuchen der Produkte sowie das Einsehen von abgegebenen Bewertungen muss anonym und somit auch ohne Anmeldung uneingeschränkt möglich sein.
- Für das Anlegen eines neuen Produkts oder das Abgeben einer Bewertung muss sich ein Benutzer zuvor registriert und angemeldet haben.
- Neue Benutzer können sich uneingeschränkt direkt über eine eigene Seite im Portal registrieren. Für jeden Benutzer sind dabei mindestens Benutzername (muss systemweit eindeutig sein) und Passwort über eine sinnvolle Benutzeroberfläche zu erfassen.
- Auf einer Übersichtsseite sollen alle im System gespeicherten Produkte aufgelistet werden. Für jedes Produkt muss mindestens sein Name, sein Hersteller, der Benutzer, der das Produkt angelegt hat, die Anzahl der abgegebenen Bewertungen und die durchschnittliche Bewertung (zwischen 1.0 und 5.0) für das Produkt angezeigt werden. Über ein entsprechendes Formular kann ein angemeldeter Benutzer ein neues Produkt anlegen.
- Durch Anklicken eines Produkts auf der Übersichtsseite gelangt man zu einer Detailansicht, in der alle Bewertungen zu einem Produkt chronologisch absteigend geordnet aufgelistet werden. Für jede Bewertung ist zumindest der Ersteller, das Erstellungsdatum, die eigentliche Bewertung (1, 2, 3, 4 oder 5 in Schulnoten) sowie ein möglicher Kommentar in Textform ansprechend darzustellen. Über ein entsprechendes Formular kann auf einfache Art und Weise eine weitere Bewertung für das Produkt erstellt werden.
- Auf einer Suchseite soll über ein Formular durch Eingabe eines Suchbegriffs (Freitext) gezielt nach Produkten im Forum gesucht werden können. Nach Absetzen einer Suchanfrage sollen alle Produkte angezeigt werden, deren Name oder Hersteller den eingegebenen Suchbegriff enthält. Durch das Anklicken eines Suchergebnisses kann direkt zur Detailseite des entsprechenden Produkts gesprungen werden.
- Ein angemeldeter Benutzer soll seine (und nur seine) abgegebenen Bewertungen im Nachhinein auch noch korrigieren bzw. löschen können.
- Ebenso soll der Ersteller eines Produkts (und nur dieser) die Produktdaten selbst auch nachträglich noch bearbeiten können. Dazu zählt aber natürlich nicht das Bearbeiten oder Löschen von Kommentaren anderer Benutzer… ;)

- Eine Gruppierung der Produkte in Kategorien wäre natürlich wünschenswert und auch sehr hilfreich für die Darstellung, ist aber nicht zwingend erforderlich.

Technische Anforderungen

- Die Webseite ist mit PHP und unter Verwendung des Model-View-Controller-Entwurfsmusters zu realisieren.
- Anwendungslogik, Präsentationslogik und Infrastrukturcode müssen sinnvoll und sauber voneinander entkoppelt werden.
- Alle zu speichernden Daten sollen in einer MySQL-Datenbank abgelegt werden. Entwickeln Sie dazu ein entsprechendes Datenbankmodell und dokumentieren Sie es ausführlich (Diagramme etc.).
- Passwörter dürfen nur in ausreichend gesicherter Form in der Datenbank abgelegt werden.
- Die Webseite soll ein einheitliches und ansprechendes Layout bieten und muss auch auf mobilen Endgeräten gut verwendbar sein. Für die Umsetzung dieser Anforderung kann ein entsprechendes UI-Framework wie z. B. Bootstrap verwendet werden.
- Die gesamte Anwendung muss möglichst robust implementiert werden. Diverse Sicherheitsprüfungen dürfen z. B. auch durch das Aufrufen von Skripts mit abgeänderten Parametern oder durch manuell abgesetzte HTTP-Anfragen nicht umgangen werden können.
- Die endgültige Lösung muss auf der in der Übung verwendeten Version von XAMPP betrieben werden können.

Organisatorische Anforderungen

- Die Projektarbeit ist in Einzelarbeit auszuführen.
- Für die Umsetzung sind nach Möglichkeit nur die in der Lehrveranstaltung vorgestellten Mittel zu verwenden.
- Die Projektarbeit ist spätestens bis zum oben aufgeführten Abgabetermin im Moodle-Kurs der SCR4-Übung hochzuladen und zum im Stundenplan vorgesehenen Termin in Form einer Live-Demonstration zu präsentieren.

Abzugebende Komponenten

- Ausführliche Systemdokumentation als PDF-Datei, mindestens mit folgendem Inhalt:
    - Allgemeine Lösungsidee
    - Datenmodell und Erstellungsskript für Datenbank
    - Architektur und Struktur der Webseite
    - Abgedruckter Code der Webseite (HTML-Seiten, PHP-Skripts, Stylesheets)
    - Testfälle inklusive Screenshots
- Ordner der entwickelten Webseite mit allen relevanten Daten (HTML-Seiten, PHP-Skripts, Stylesheets, andere Ressourcen wie Bilder etc.)
- Datenbankskript zur Erstellung der Datenbank als Textdatei
- Datenbankskript zur Erstellung von Testdaten in der Datenbank, mit denen auch bei der Entwicklung bereits getestet wurde, als Textdatei

SCR Project (Product Rating)

Table of Contents

1. Lösungsidee . . . 3
1.1. Allgemeines . . . 3
1.2. Anzeigen von Produkten . . . 3
1.3. Erstellen und Editieren von Produkten . . . 3
1.4. Produktdetailseite . . . 4
1.5. Fehlerseite bei ungültigem Controller oder Action . . . 5
2. Testfälle . . . 6
2.1. Testfall: Allgemein . . . 6
2.2. Testfall: Registrierung . . . 7
2.3. Testfall: Registrierung . . . 9
2.4. Testfall: Registrierung . . . 10
2.5. Testfall: Anmeldung . . . 11
2.6. Testfall: Anmeldung . . . 13
2.7. Testfall: Produktübersicht . . . 14
2.8. Testfall: Produktübersicht . . . 15
2.9. Testfall: Erstellen eines Produktes . . . 16
2.10. Testfall: Erstellen eines Produktes . . . 17
2.11. Testfall: Erstellen eines Produktes . . . 18
2.12. Testfall: Bearbeiten eines Produktes . . . 19
2.13. Testfall: Bearbeiten eines Produktes . . . 22
2.14. Testfall: Bearbeiten eines Produktes . . . 23
2.15. Testfall: Produktdetails . . . 24
2.16. Testfall: Produktdetails . . . 24
2.17. Testfall: Bewertungen . . . 26
2.18. Testfall: Bewertungen . . . 28
2.19. Testfall: Bewertungen . . . 30
3. Quellcode . . . 32
3.1. Application . . . 33
3.1.1. Commands . . . 34
3.1.2. Entities . . . 43
3.1.3. Interfaces . . . 47
3.1.4. Models . . . 49
3.1.5. Queries . . . 54

3.1.6. Services 59
3.2. Infrastructure 60
3.3. Presentation. 77
3.3.1. Controllers 77
3.4. Views. 89
3.4.1. Pages. 89
3.4.2. Partial views 94
3.5. Styles. 103
4. Datenbank 104
4.1. Datenbankmodell 104
4.2. Scripts. 105

1. Lösungsidee

1.1. Allgemeines

Zu Beginn wurde eine neue Datenbank erstellt, welches genau 3 Tabellen enthält, jeweils eine für Produkt, eine für Benutzer und eine für Bewertungen.

Die Tabellen wurden entsprechend mit Testdaten (in diesem Fall Bohrmaschinen) befüllt, dies wurde in ein eigenes SQL File ausgelagert.

Insgesamt wurden außerdem drei Controller erstellt, welche sich um die Anzeige, Speichern und Aktualisieren der unterschiedlichen Entitäten kümmern, nämlich:

- Users
- Products
- Ratings

1.2. Anzeigen von Produkten

Anschließend wurde eine Übersichtsseite für die Produkte erstellt, wo alle Produkte mit ihren Bewertungen und der Bewertungsanzahl angezeigt werden. Dazu gibt es eine Query, welche die Produkte mit deren Durchschnittsbewertungen und Anzahl der Bewertungen lädt.

1.3. Erstellen und Editieren von Produkten

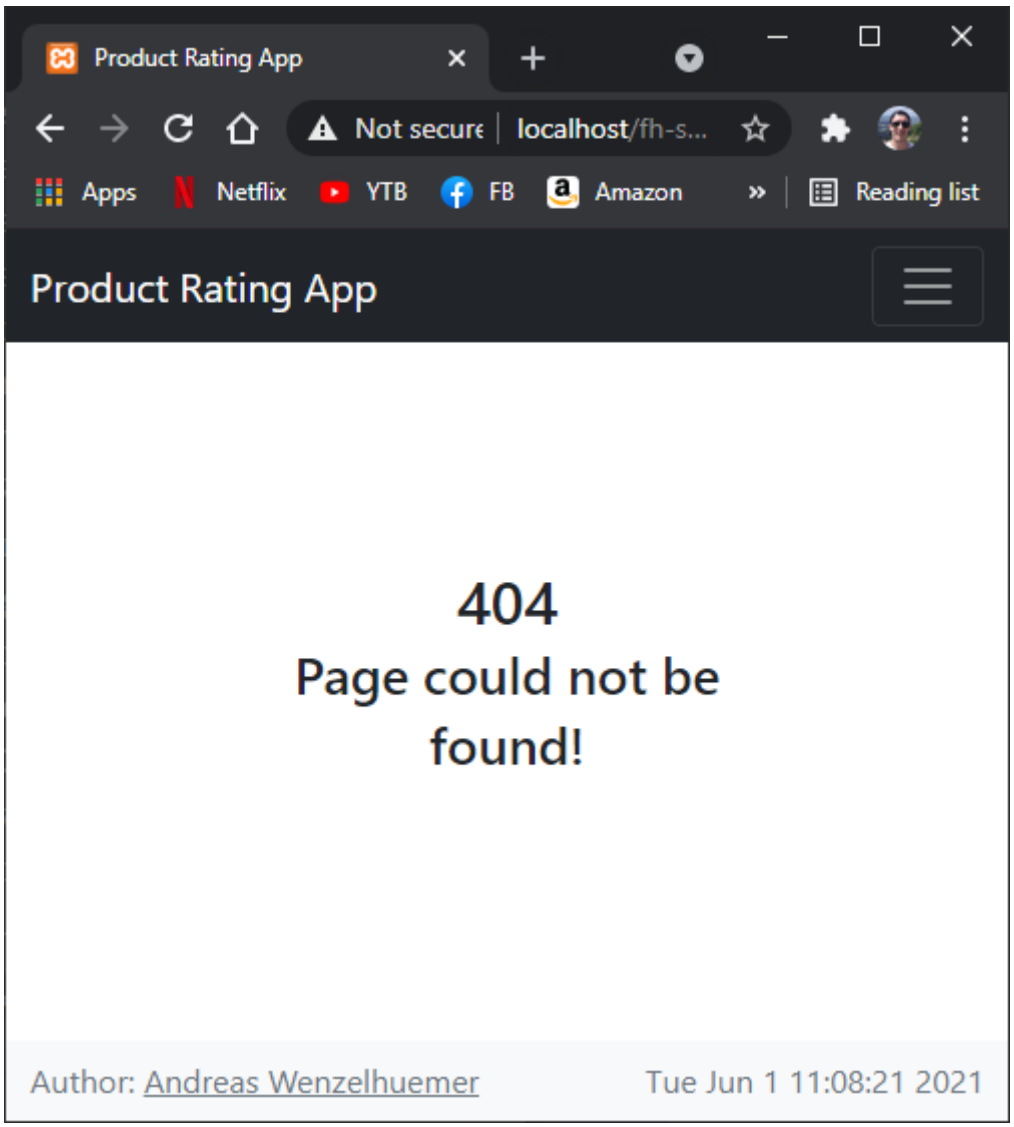
Für das Editieren und Erstellen wurden zwei eigene Seiten erstellt, wobei diese alle auf den ProductsController zugreifen. Hier wird auch entsprechend validiert, ob es sich um valide Daten handelt (keine leeren Felder) und beim Aktualisieren noch zusätzlich, ob der Nutzer auch tatsächlich der Ersteller ist. Dies wird rein über den Server überprüft, der User wird hier nicht mitgegeben. Wenn über die Url auf ein Produkt zugegriffen werden sollte, welches nicht existiert, wird dies in einer Fehlermeldung angezeigt.

1.4. Produktdetailseite

Wenn man den Link beim Produktnamen auswählt, kommt man auf eine Detailseite, wo Produktdetails, Durchschnittsbewertung und Anzahl aufgelistet wird. Darunter werden in einem Bootstrap Accordion alle Bewertungen aufgelistet. Zusätzlich gibt es immer an erster Stelle ein Formular, wo neue Bewertungen eingetragen werden können. Darunter werden die anderen Bewertungen aufgelistet und es können entsprechend Bewertungen (nur vom User erstellte) aktualisiert bzw. gelöscht werden. Ansonsten sind diese nur im Read-Only Modus verfügbar. Für die Produktdetailseite wurde eine eigene Query erstellt, da hier zusätzlich die Bewertungen geladen werden müssen. Daher gibt es eine ProduktDetailQuery und eine ProduktQuery. Wenn auf ein Produkt zugegriffen wird, welches nicht existiert, wird ein entsprechender Fehler angezeigt.

1.5. Fehlerseite bei ungültigem Controller oder Action

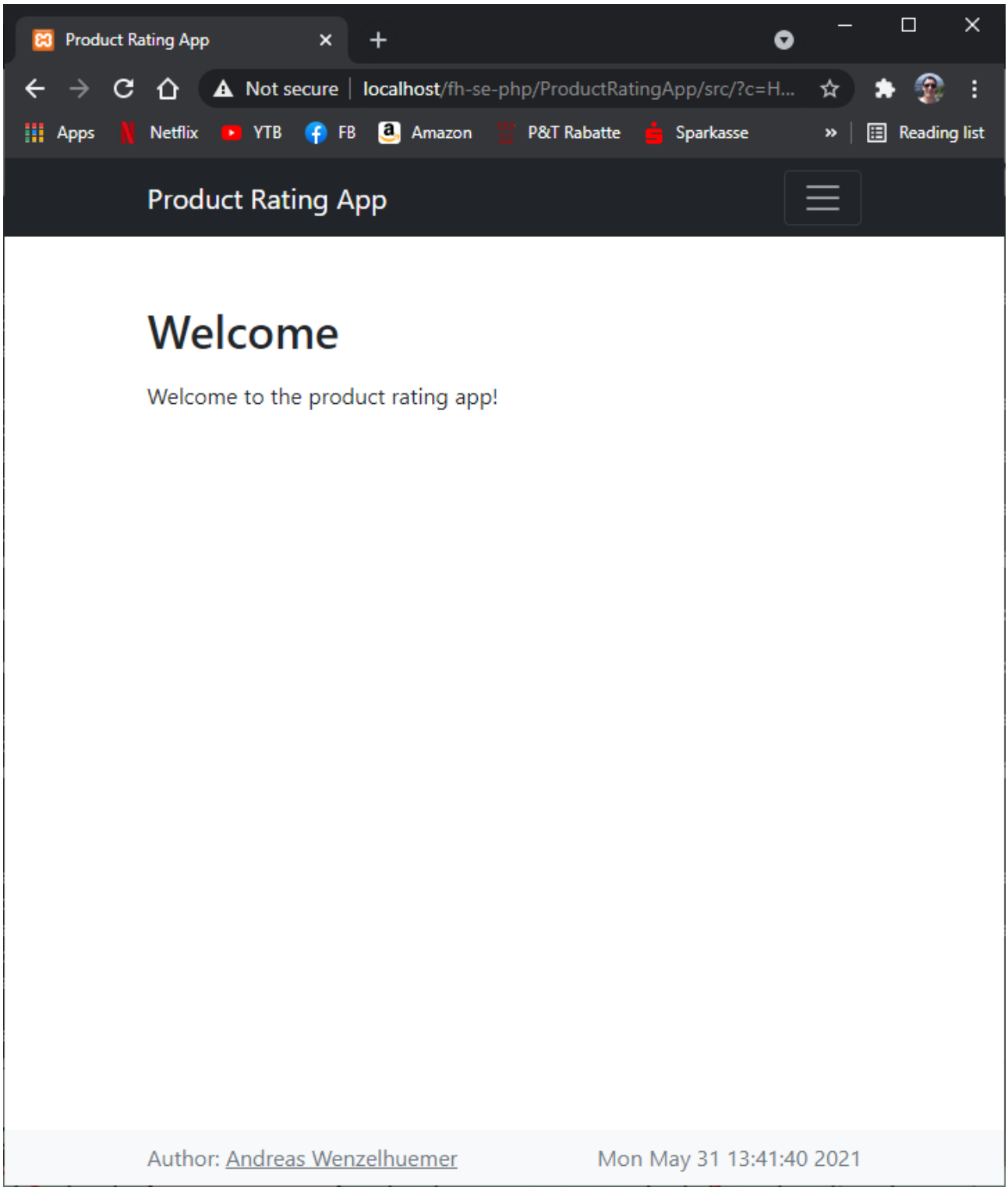
Wenn ein ungültiger Controllername beziehungsweise eine ungültige Action eingegeben wird, wird automatisch auf eine 404 Page weitergeleitet. Dazu wurde die Methode handleRequest in der Klasse MVC.php erweitert.



2. Testfälle

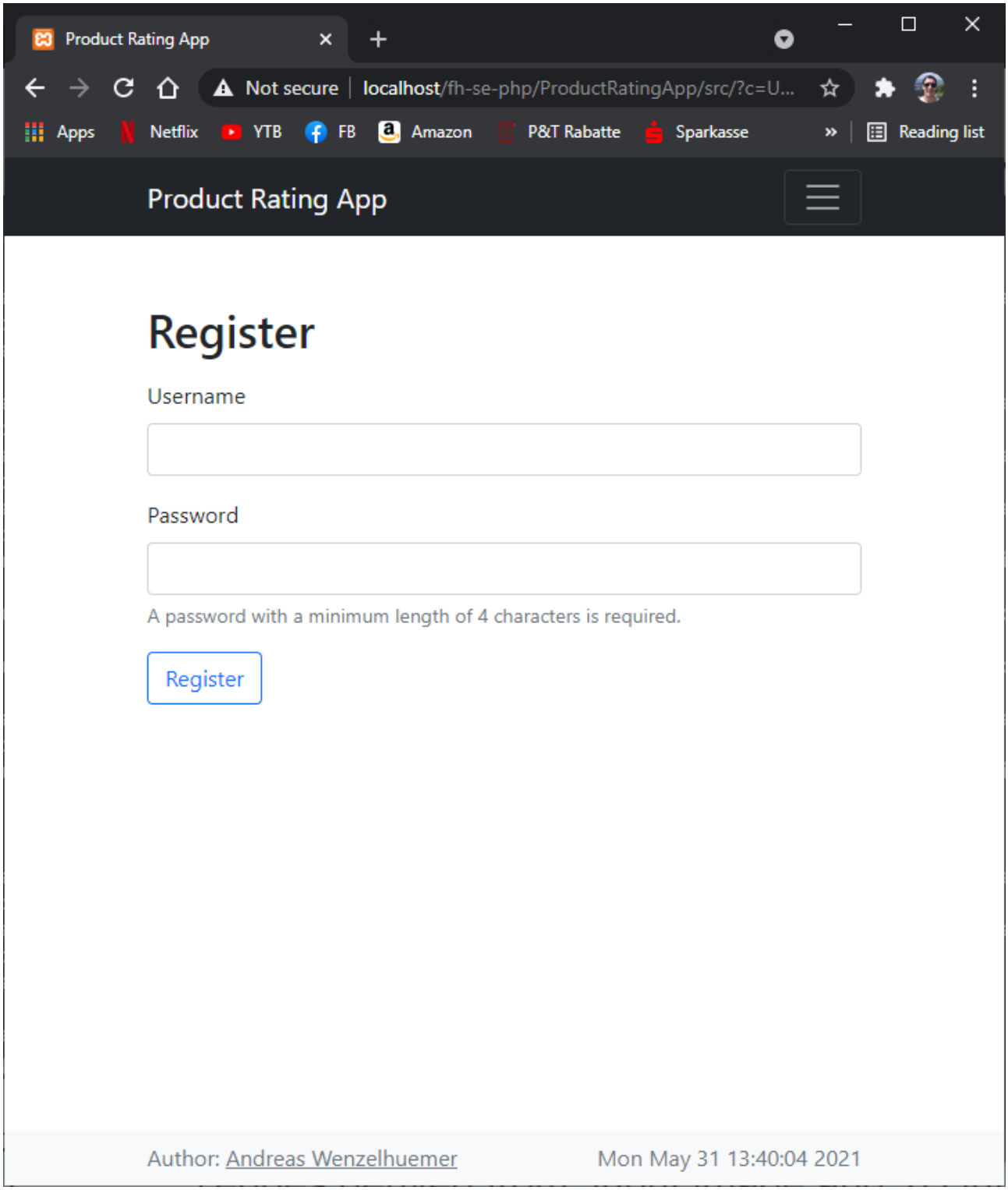
2.1. Testfall: Allgemein

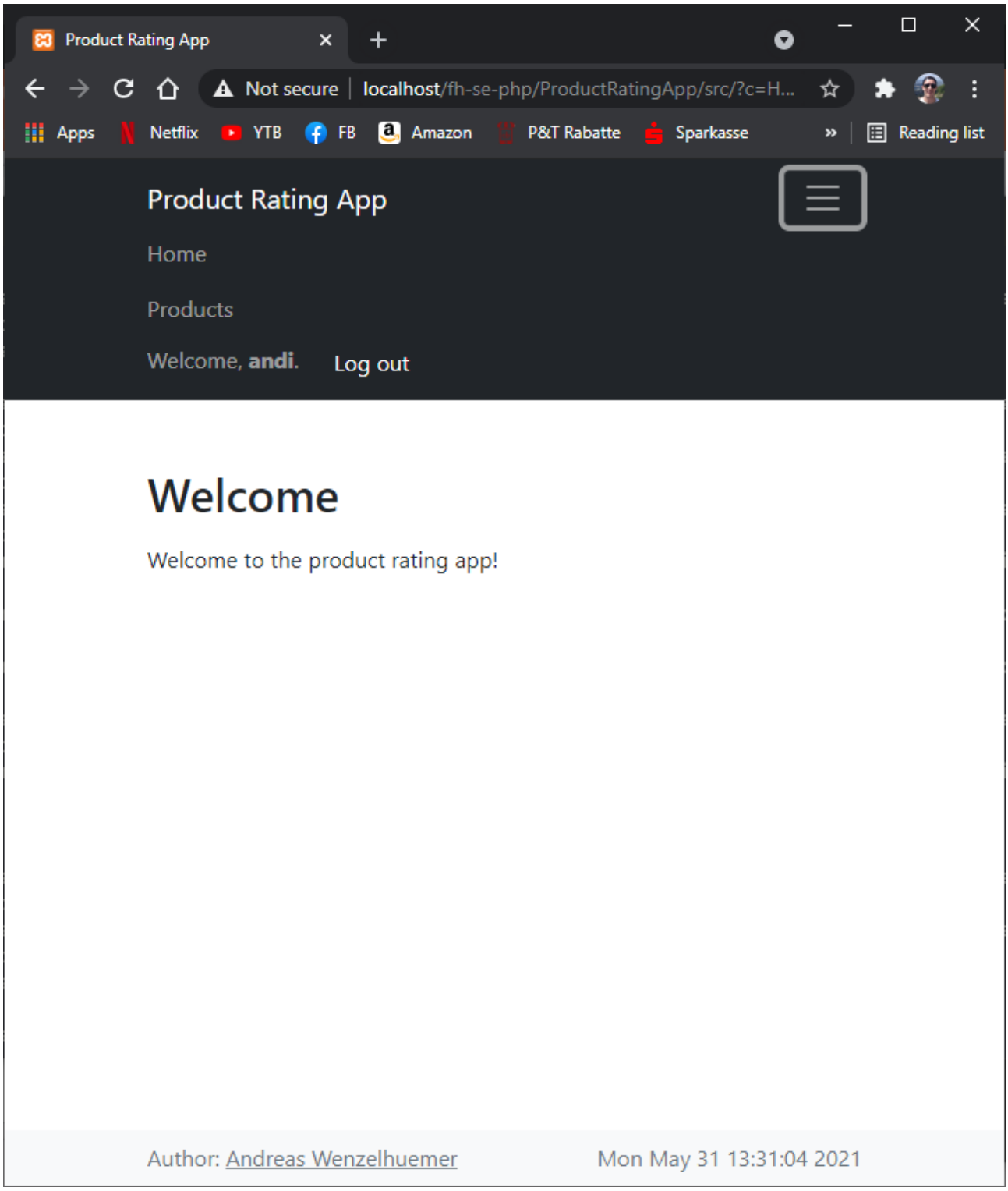
Website lässt sich ausführen und liefert keine Fehler.



2.2. Testfall: Registrierung

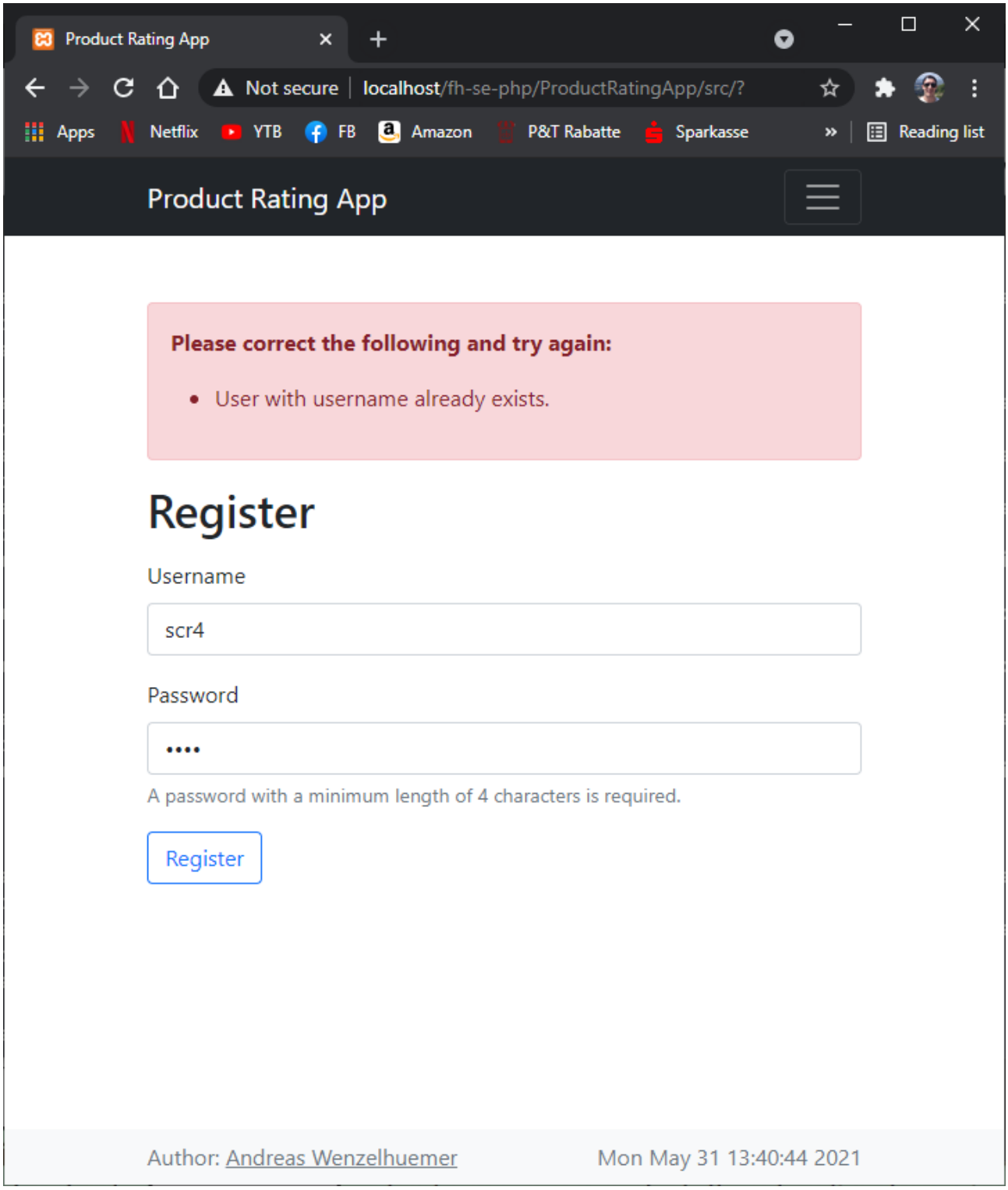
Neuer Benutzer kann sich registrieren und wird angemeldet.




Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=H...
Apps
Netflix
YTB
FB
Amazon
P&T Rabatte
Sparkasse
Reading list
Product Rating App
Home
Products
Welcome, andi.
Log out
Welcome
Welcome to the product rating app!
Author: Andreas Wenzelhuemer
Mon May 31 13:31:04 2021

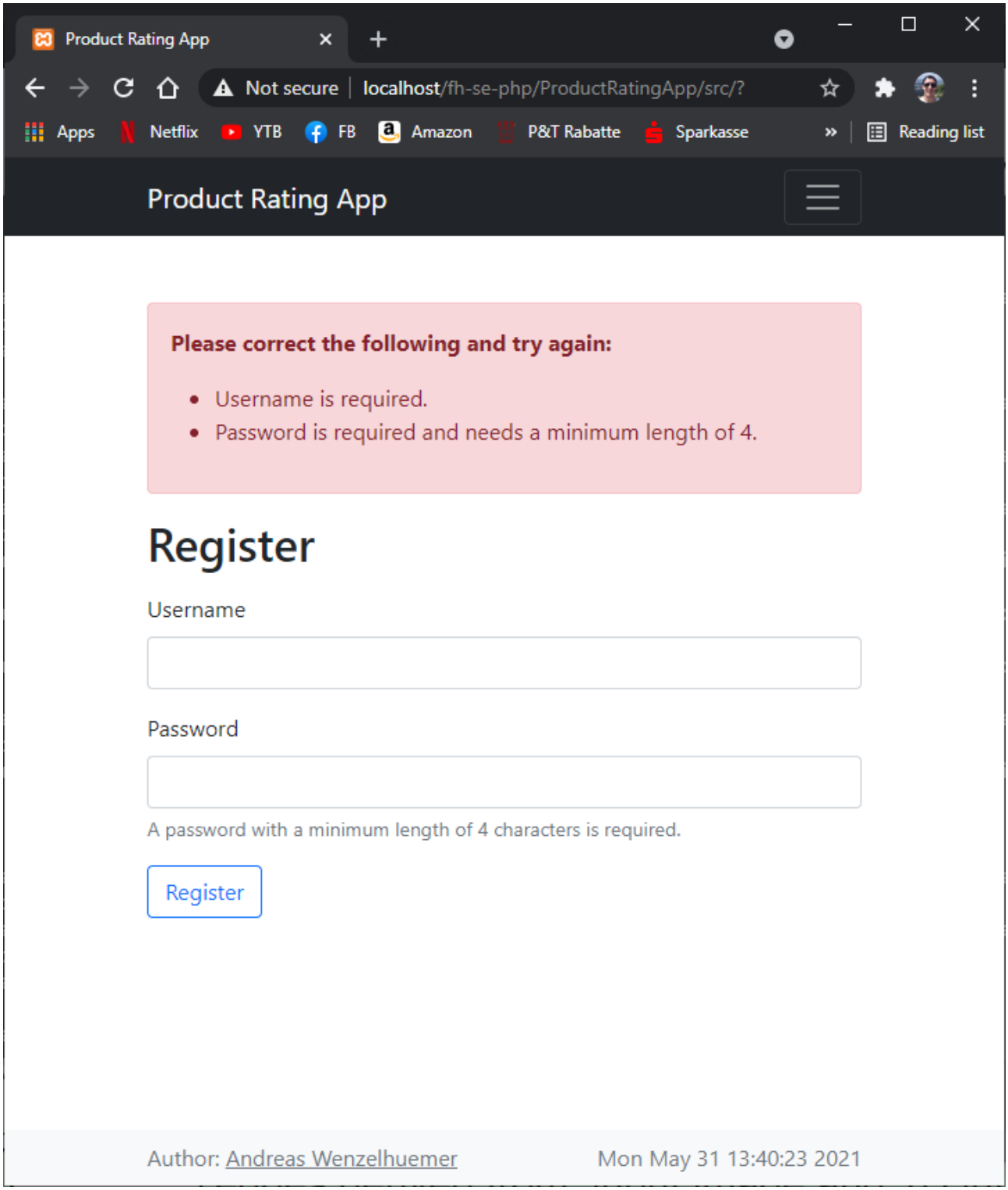
2.3. Testfall: Registrierung

Registrierung mit bereits im System vorhandenem Benutzernamen.liefert Valideriungsfehler



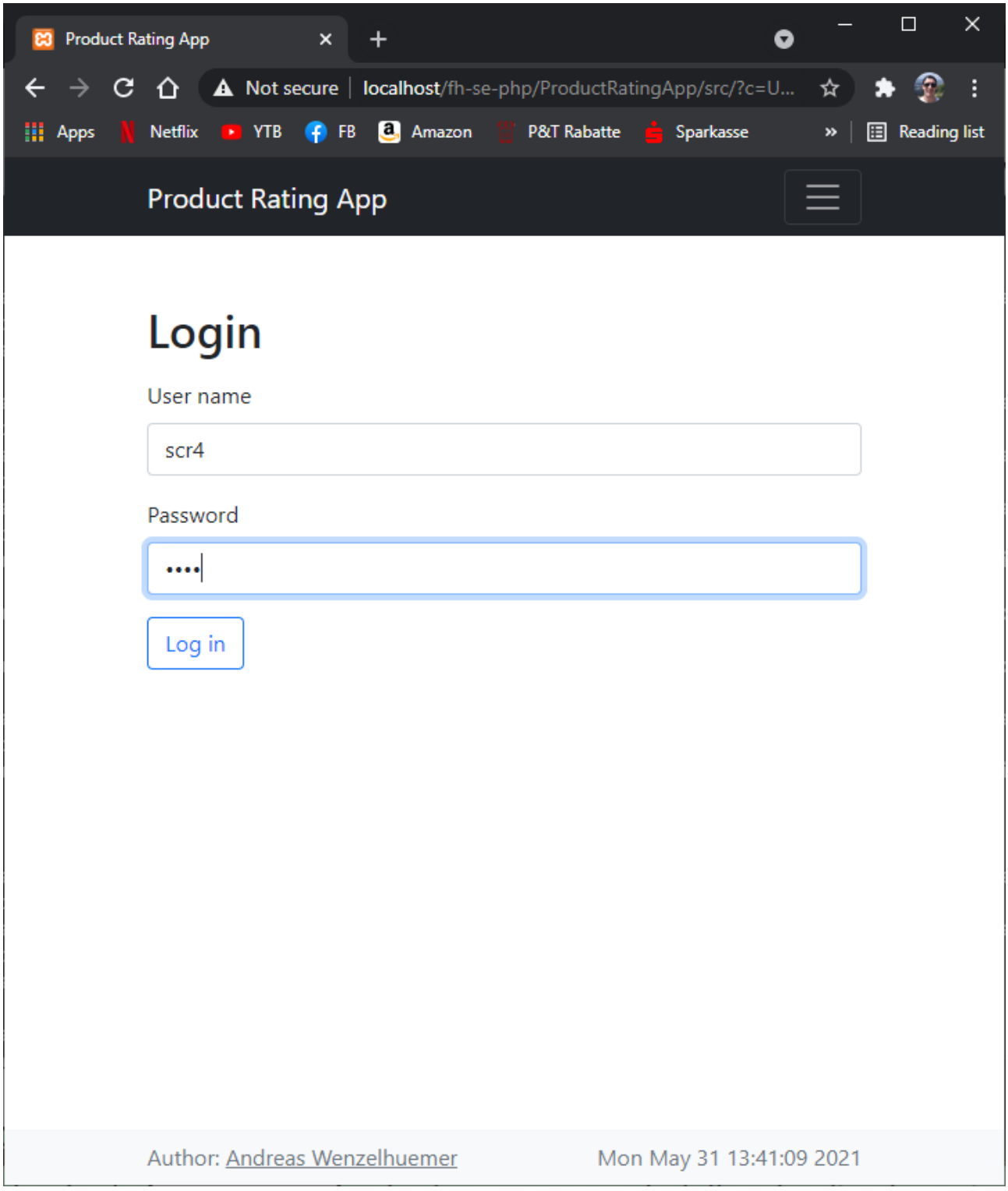
2.4. Testfall: Registrierung

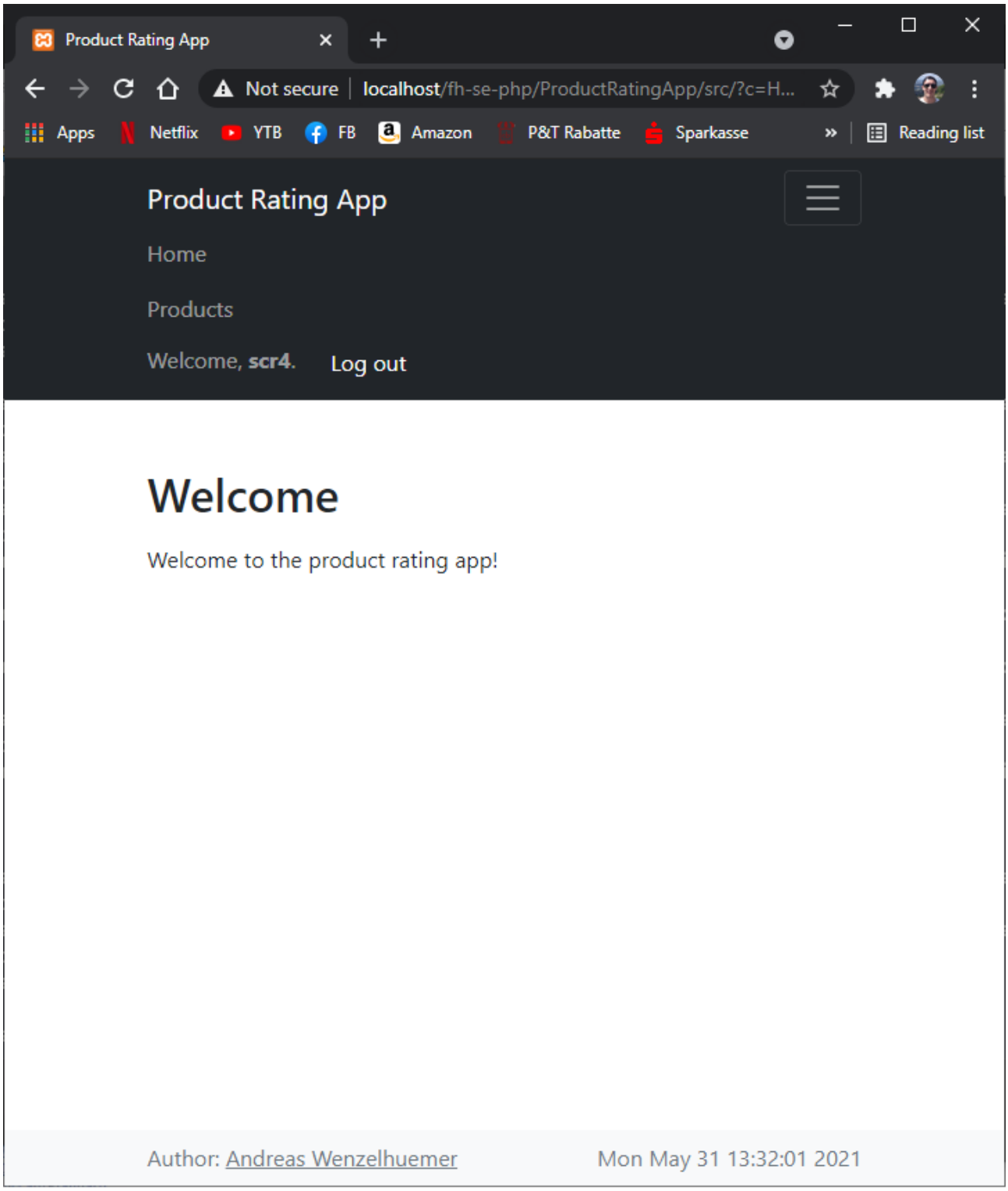
Registrierung mit invaliden Eingaben liefert Validierungsfehler.



2.5. Testfall: Anmeldung

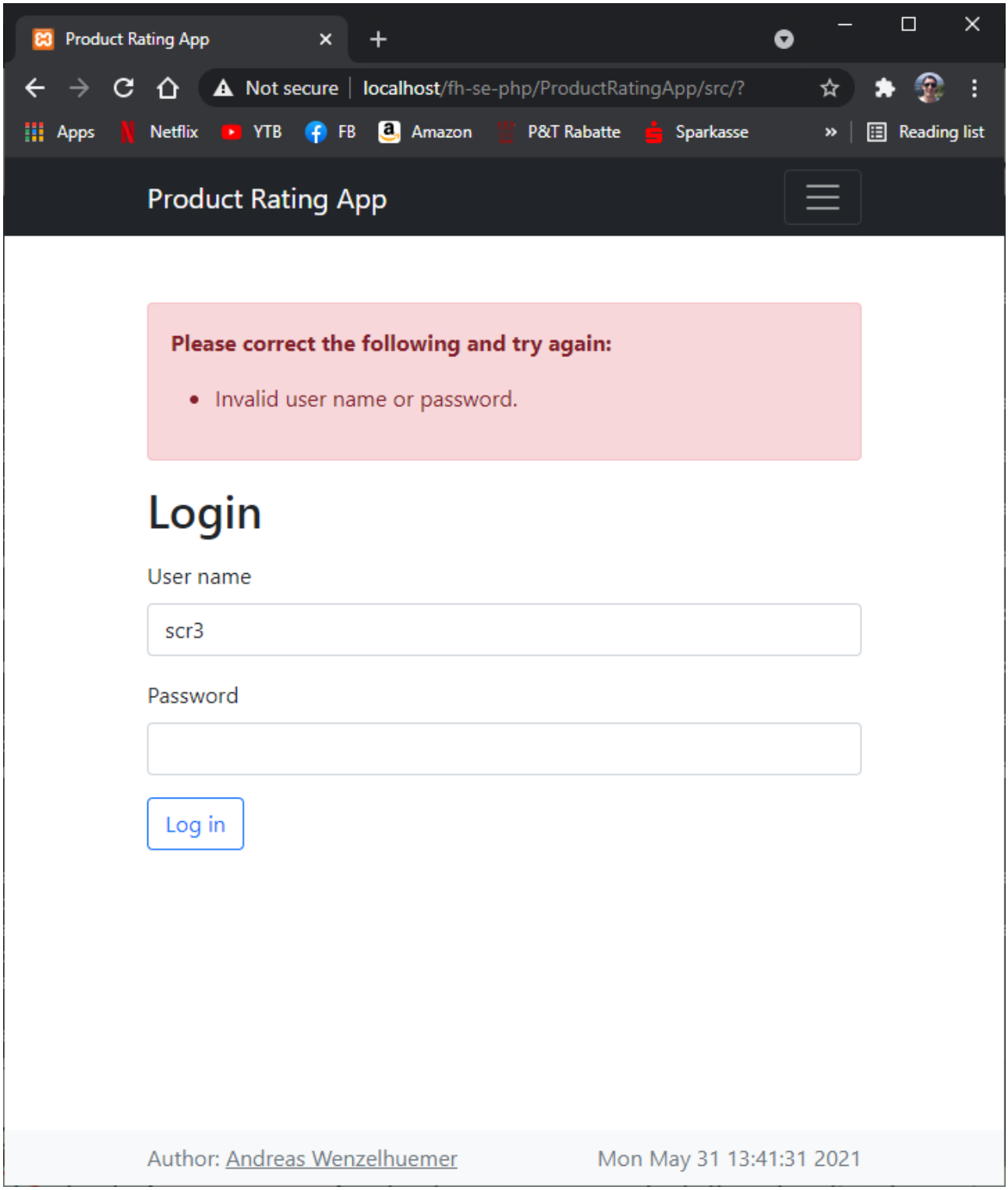
Anmeldung mit validen Eingaben führt zu Anmeldung.




Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=H...
Apps
Netflix
YTB
FB
Amazon
P&T Rabatte
Sparkasse
Reading list
Product Rating App
Home
Products
Welcome, scr4.
Log out
Welcome
Welcome to the product rating app!
Author: Andreas Wenzelhuemer
Mon May 31 13:32:01 2021

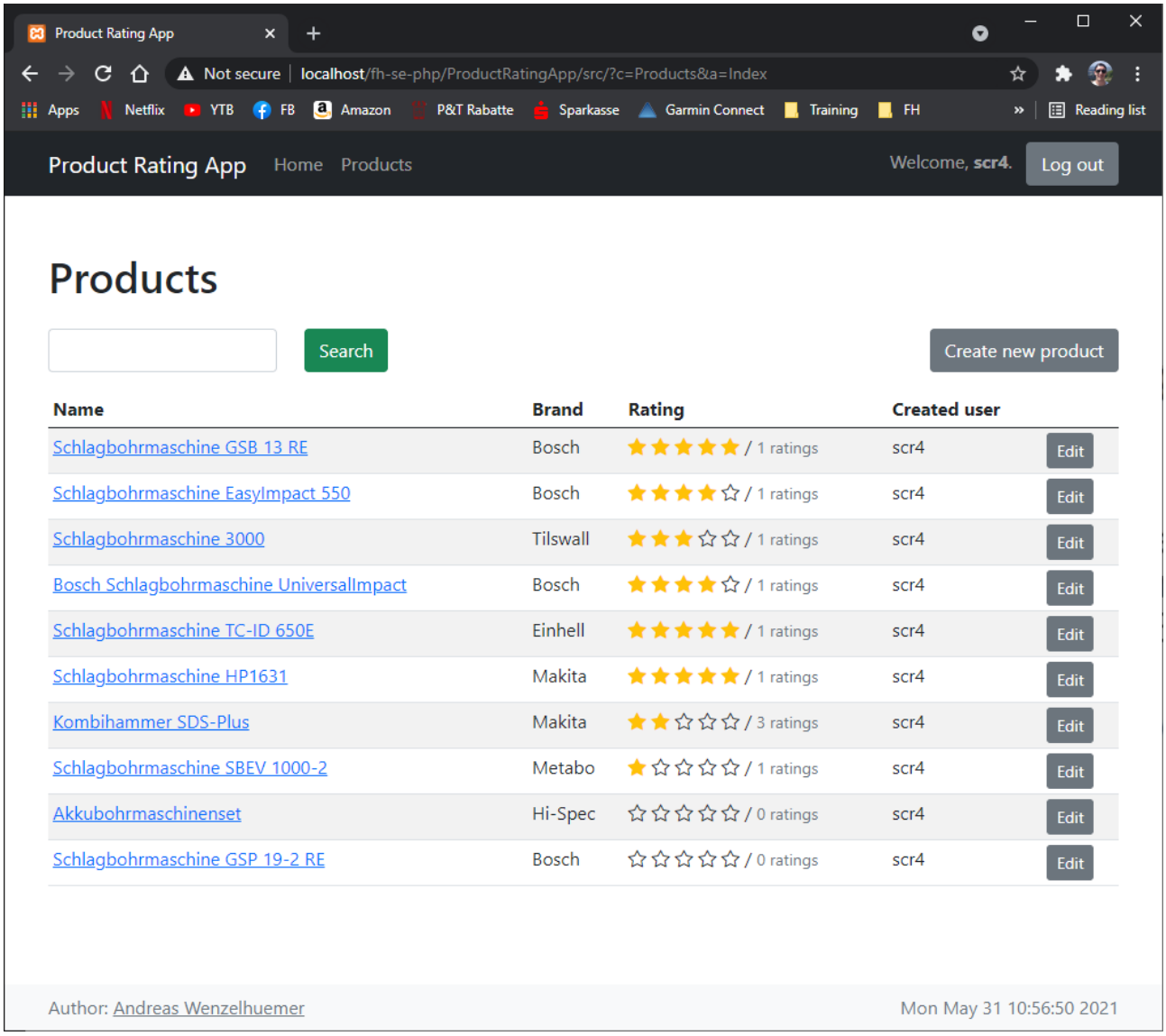
2.6. Testfall: Anmeldung

Anmeldung mit invaliden Eingaben führt zu Fehlermeldungen.



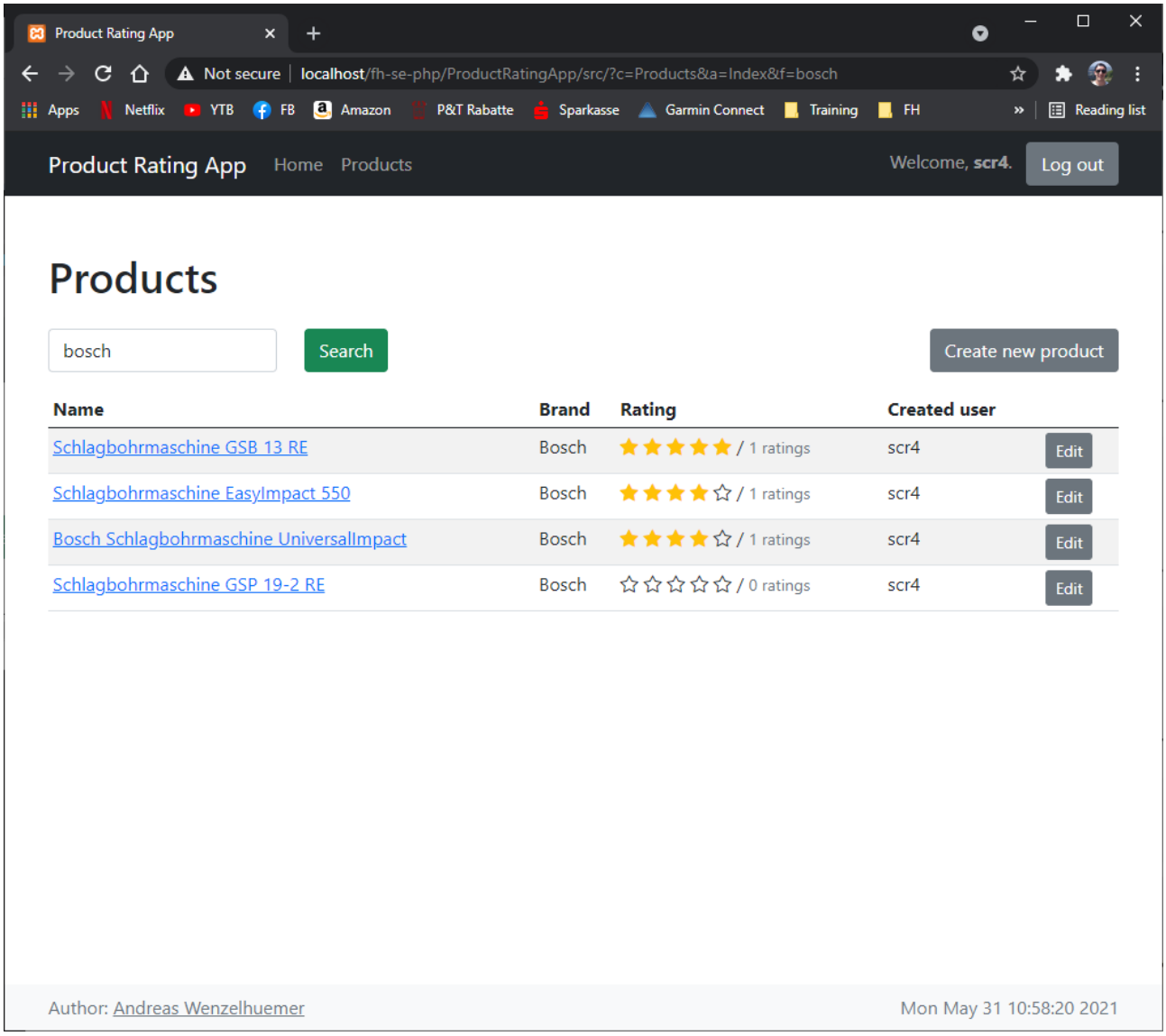
2.7. Testfall: Produktübersicht

Produkte aus der Datenbank werden entsprechend angezeigt.



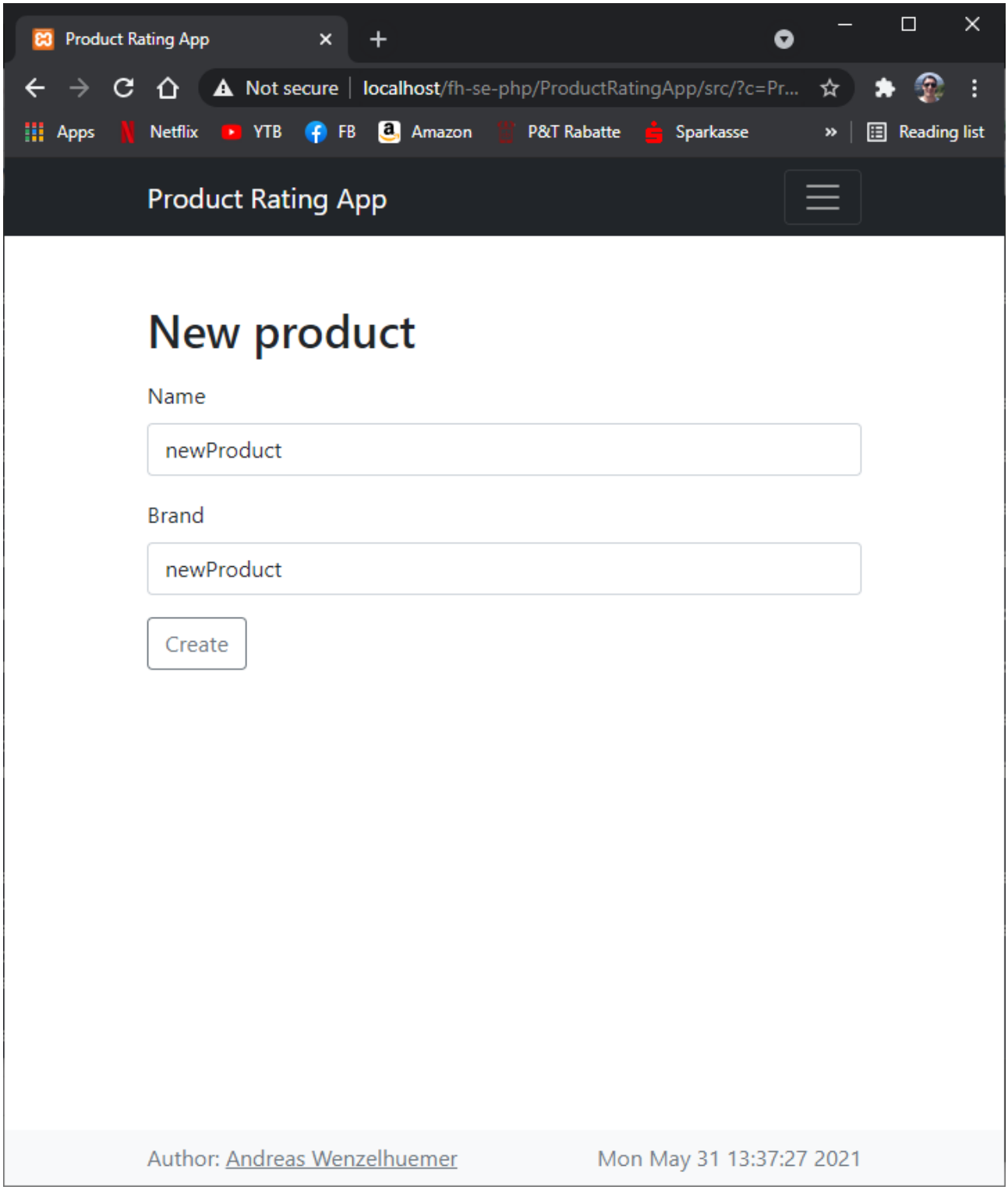
2.8. Testfall: Produktübersicht

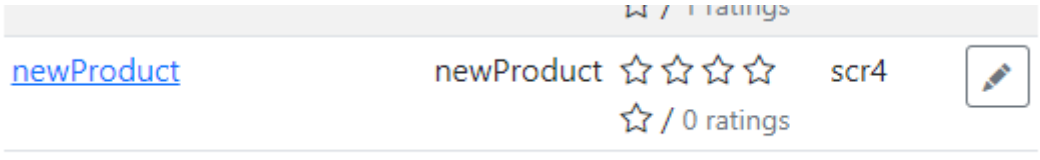
Produkte werden entsprechend anhand des Namens bzw. Hersteller gefiltert.



2.9. Testfall: Erstellen eines Produktes

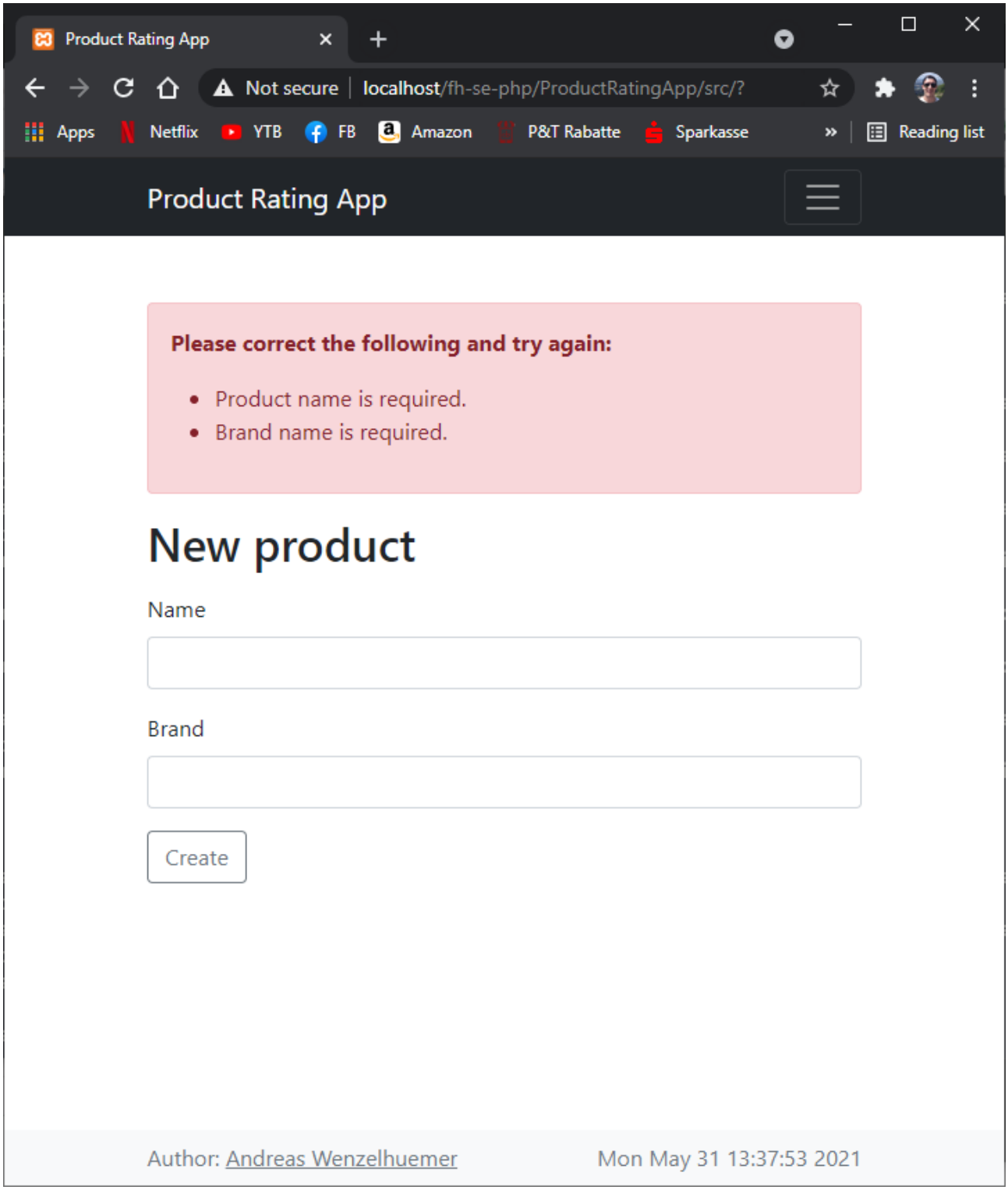
Produkt mit entsprechender Eingabe wird erstellt.





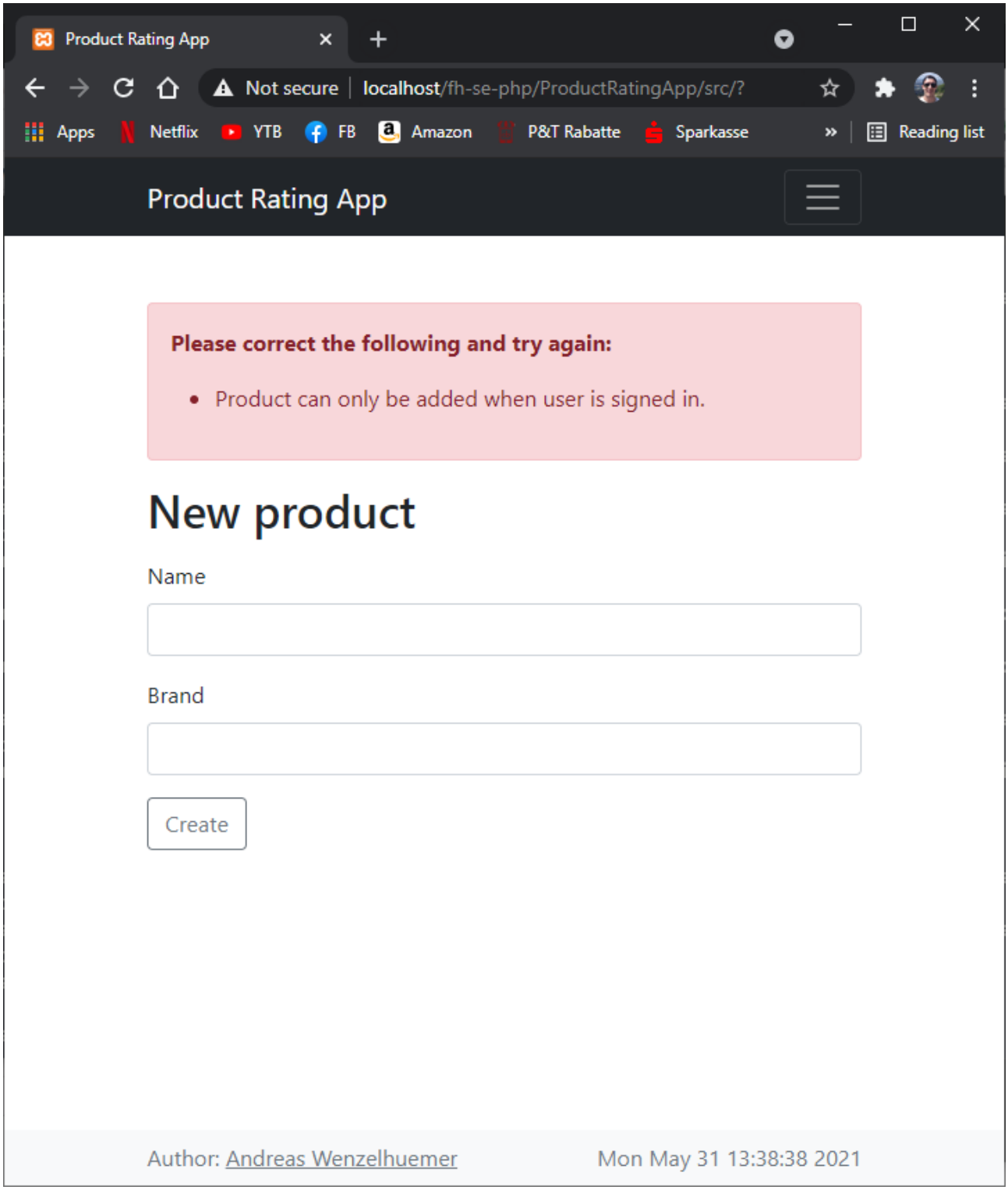
2.10. Testfall: Erstellen eines Produktes

Produkt mit ungültigen Eingaben liefert Validierungsfehler.



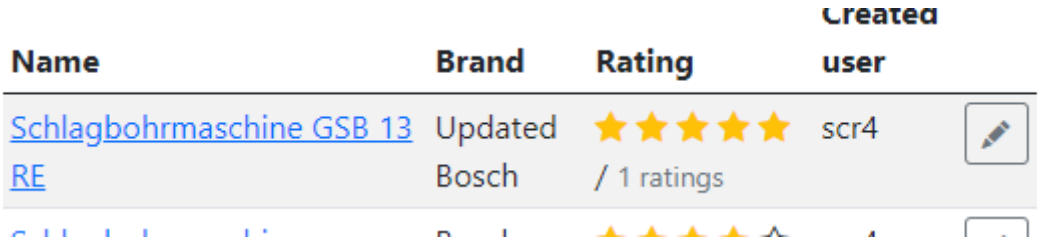
2.11. Testfall: Erstellen eines Produktes

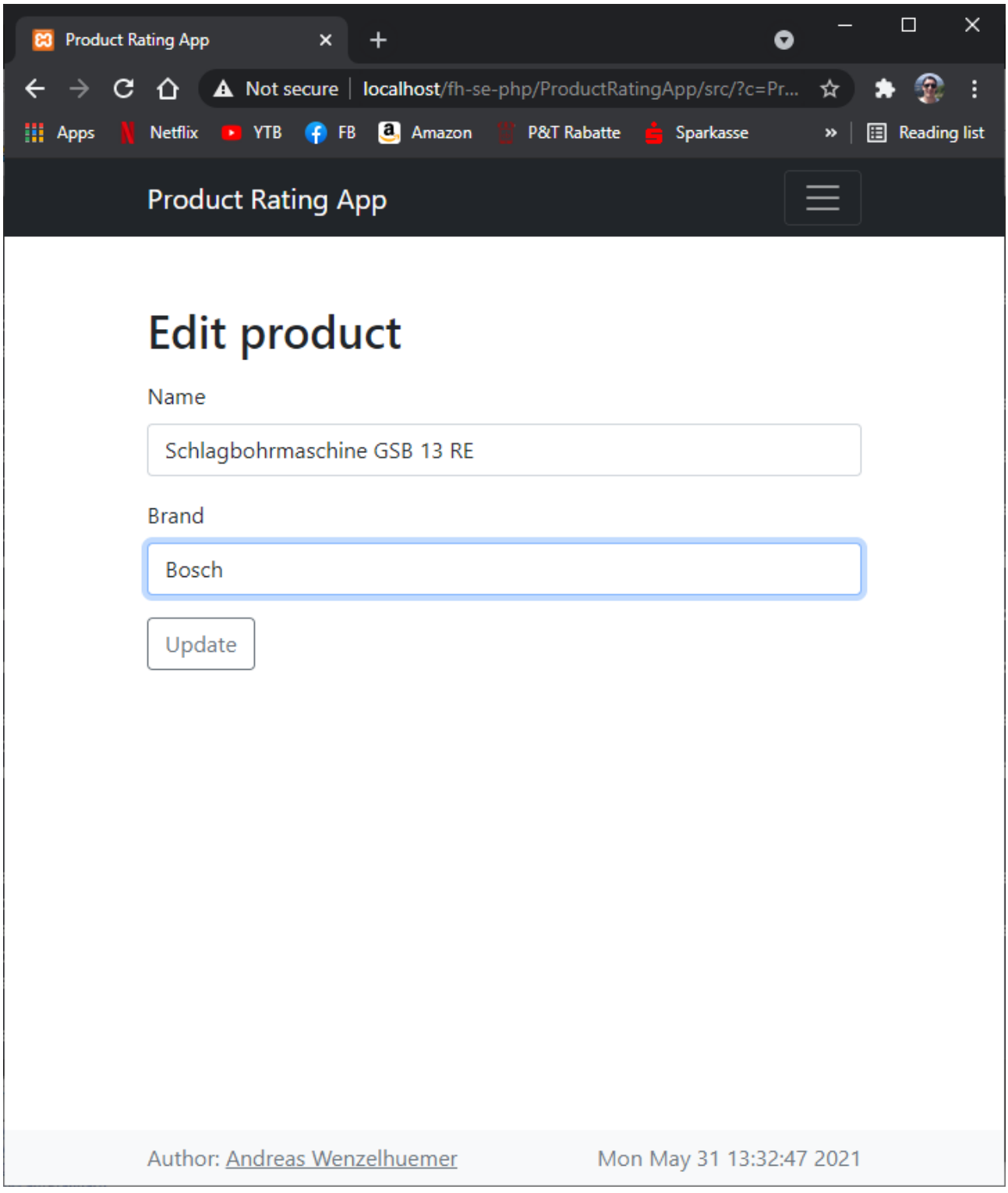
Produkterstellung ohne Anmeldung ist nicht möglich.

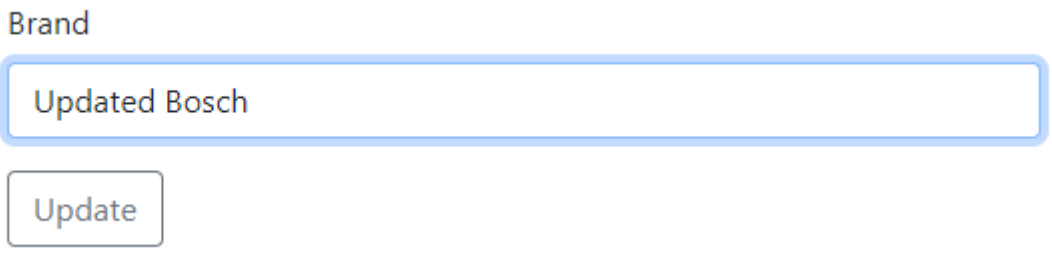


2.12. Testfall: Bearbeiten eines Produktes

Produkt wird entsprechend bearbeitet



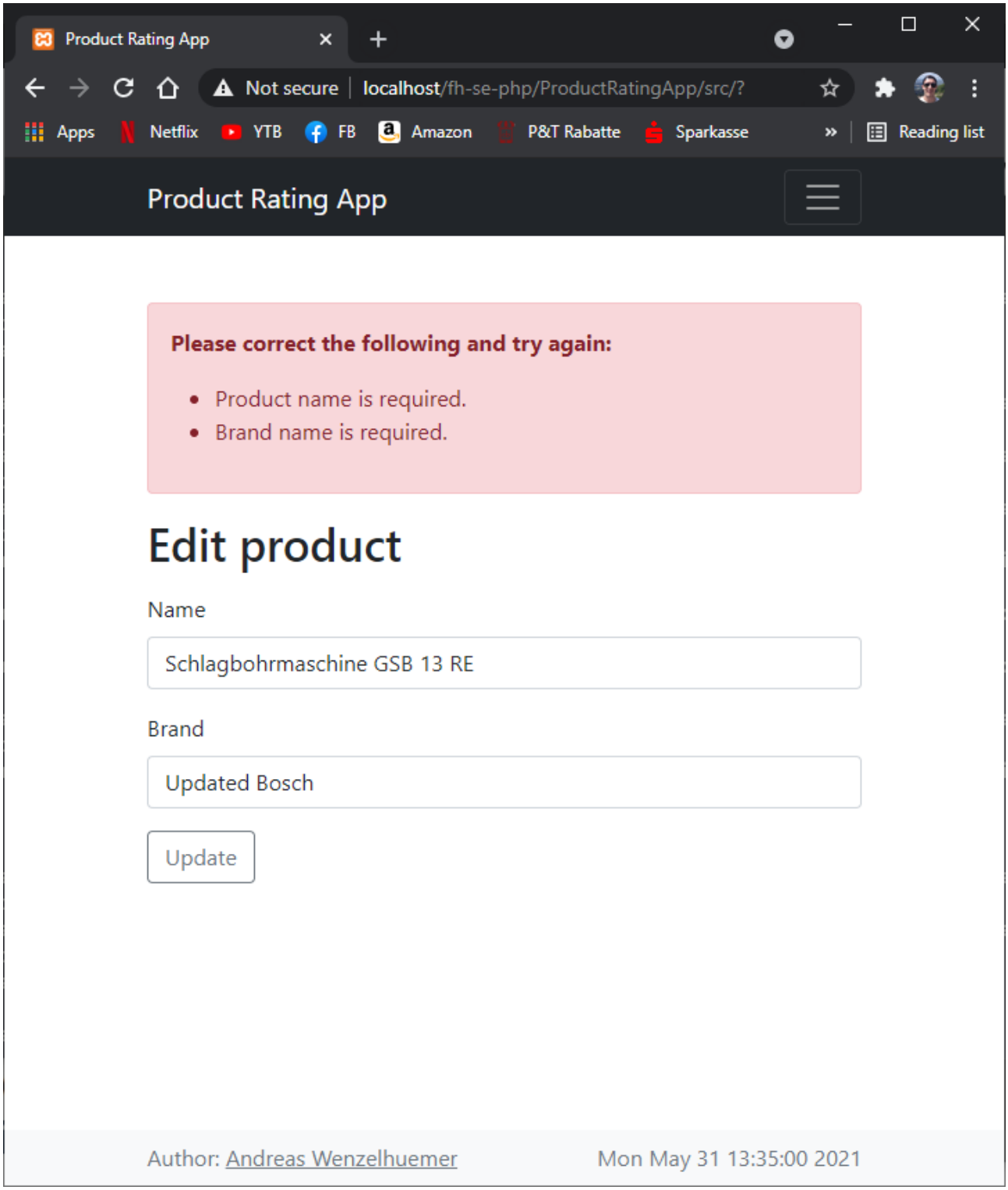

Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=Pr...
Apps Netflix YTB FB Amazon P&T Rabatte Sparkasse Reading list
Product Rating App
Edit product
Name
Schlagbohrmaschine GSB 13 RE
Brand
Bosch
Update
Author: Andreas Wenzelhuemer
Mon May 31 13:32:47 2021


Brand
Updated Bosch
Update

Name	Brand	Rating	Created user	
Schlagbohrmaschine GSB 13 RE	Updated Bosch	★★★★★ / 1 ratings	scr4	

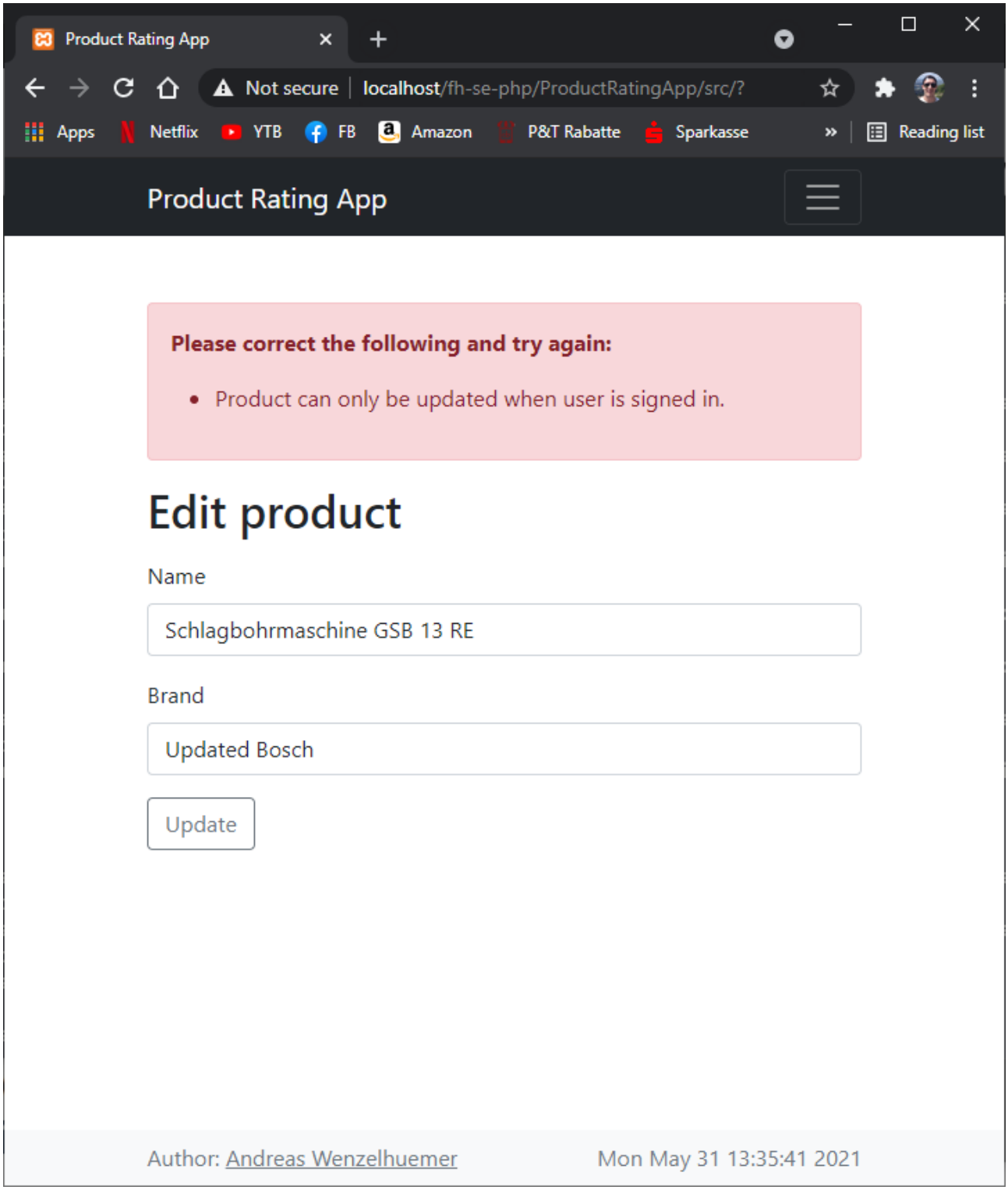
2.13. Testfall: Bearbeiten eines Produktes

Produkt mit ungültigen Eingaben liefert Validierungsfehler.



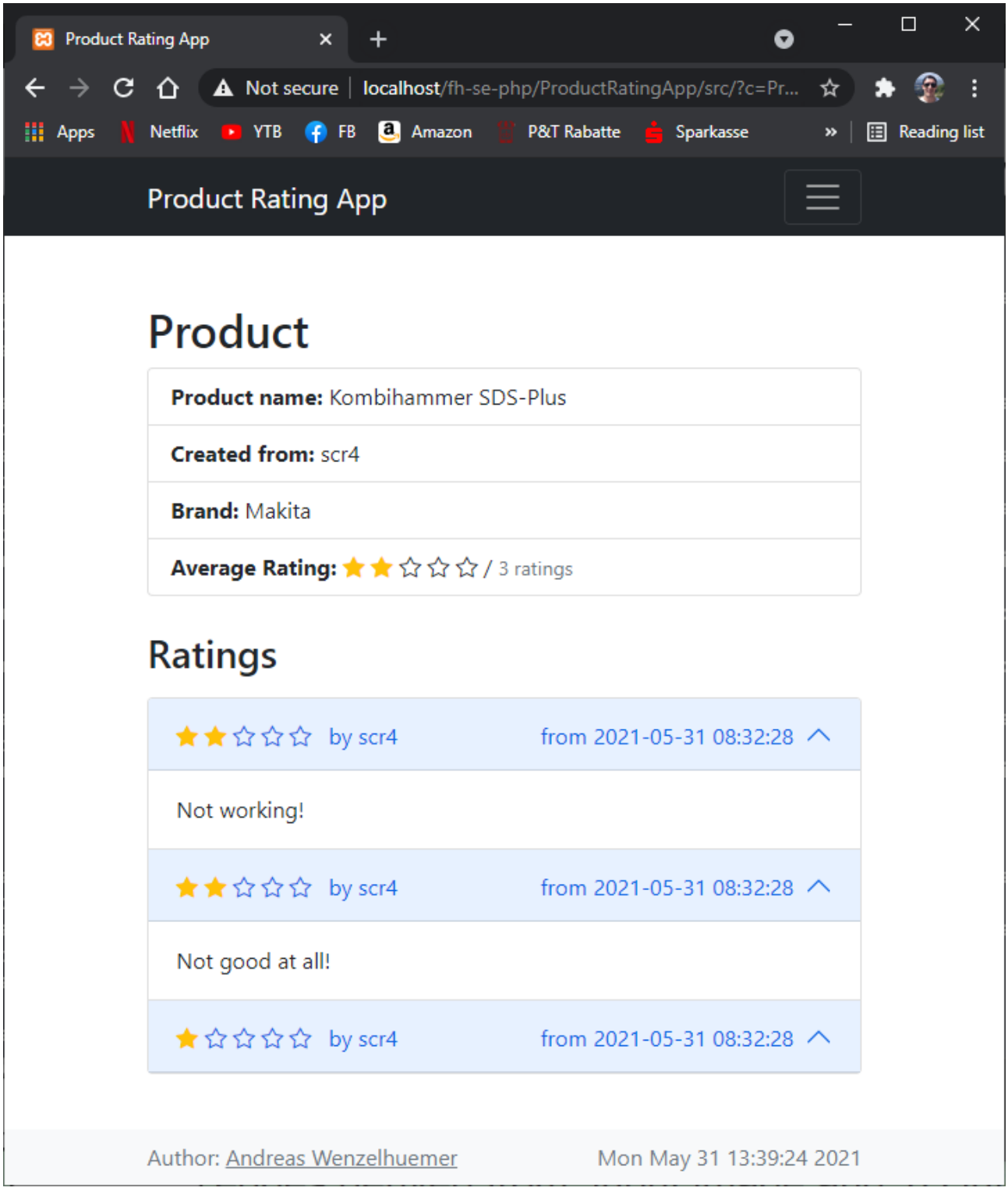
2.14. Testfall: Bearbeiten eines Produktes

Bearbeitung mit Nutzer != Ersteller ist nicht möglich.



2.15. Testfall: Produktdetails

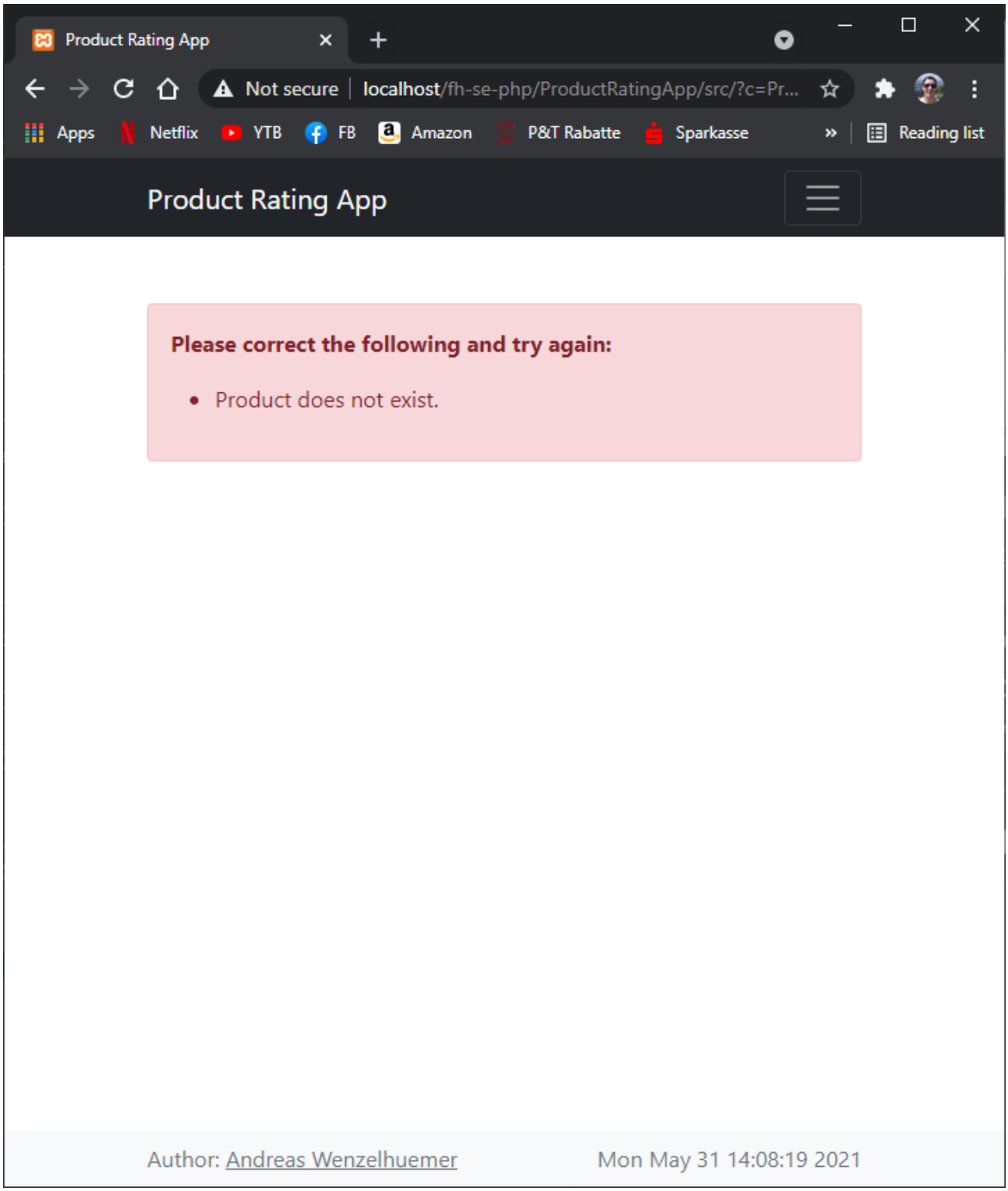
Anzeige der Produktdetails inklusive Bewertungen.



2.16. Testfall: Produktdetails

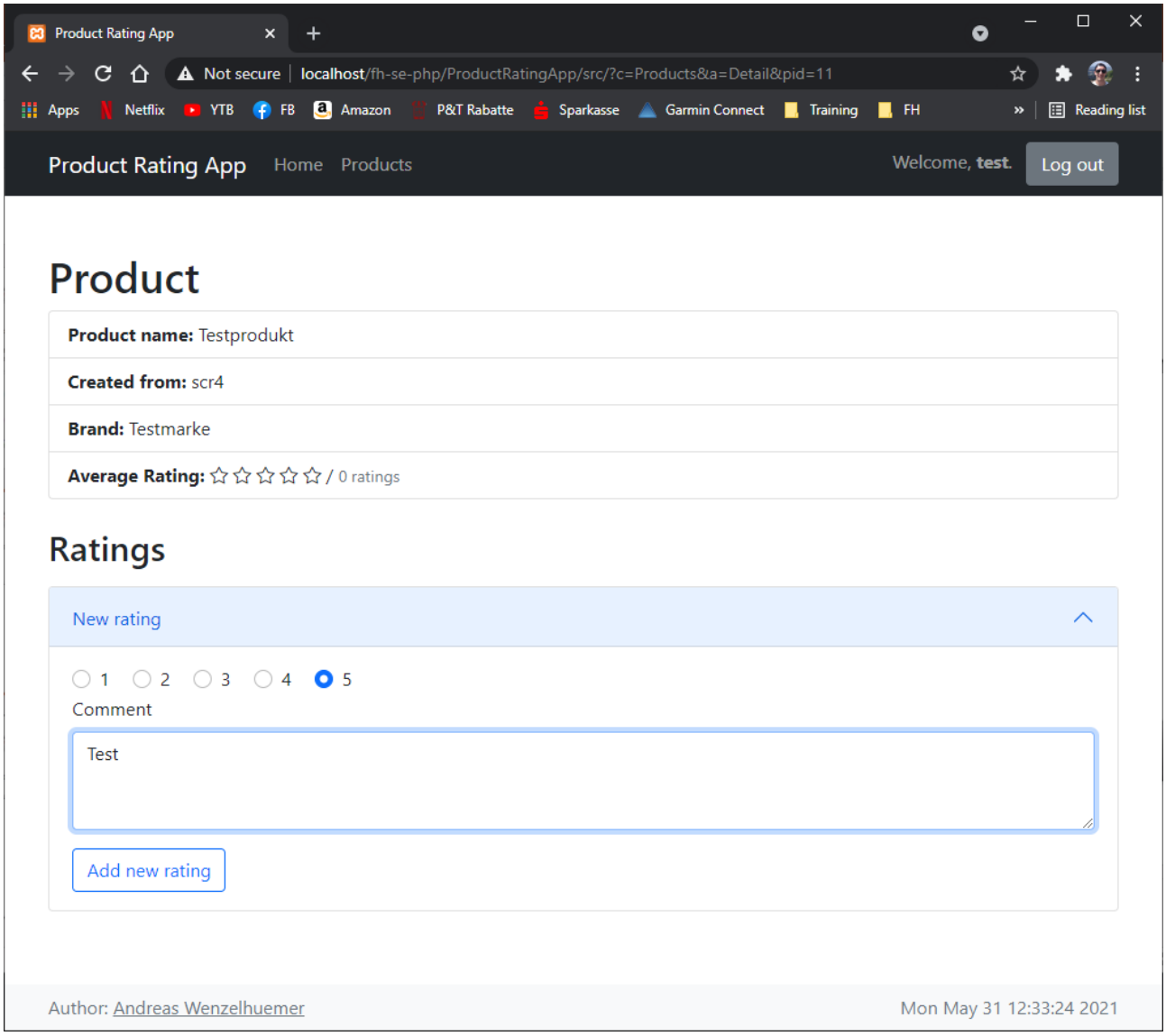
Anzeige eines Produkts, welches nicht existiert, weil es gelöscht wurde oder eine

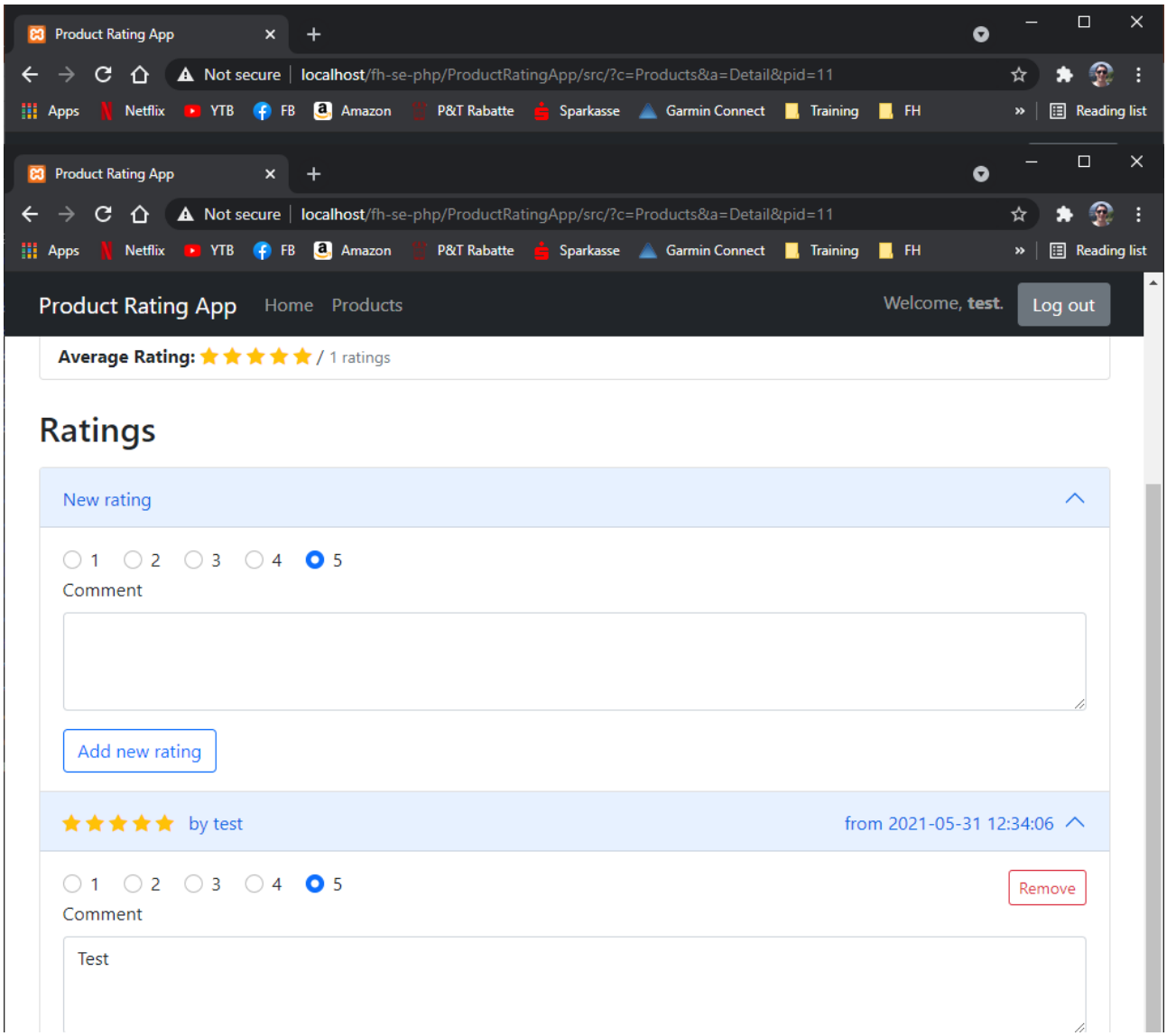
ungültige Id ausgewählt wurde.



2.17. Testfall: Bewertungen

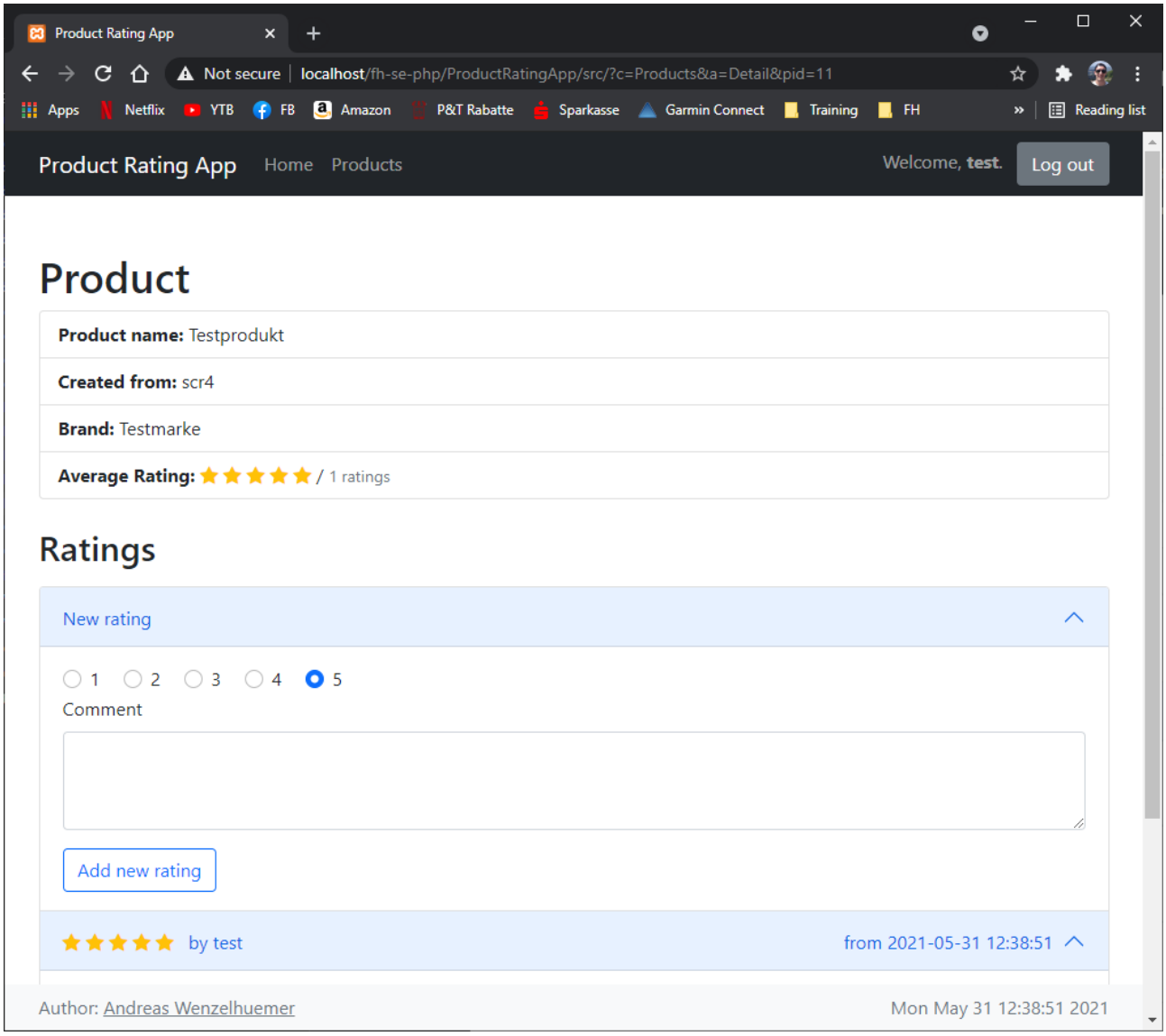
Hinzufügen einer neuen Bewertung.

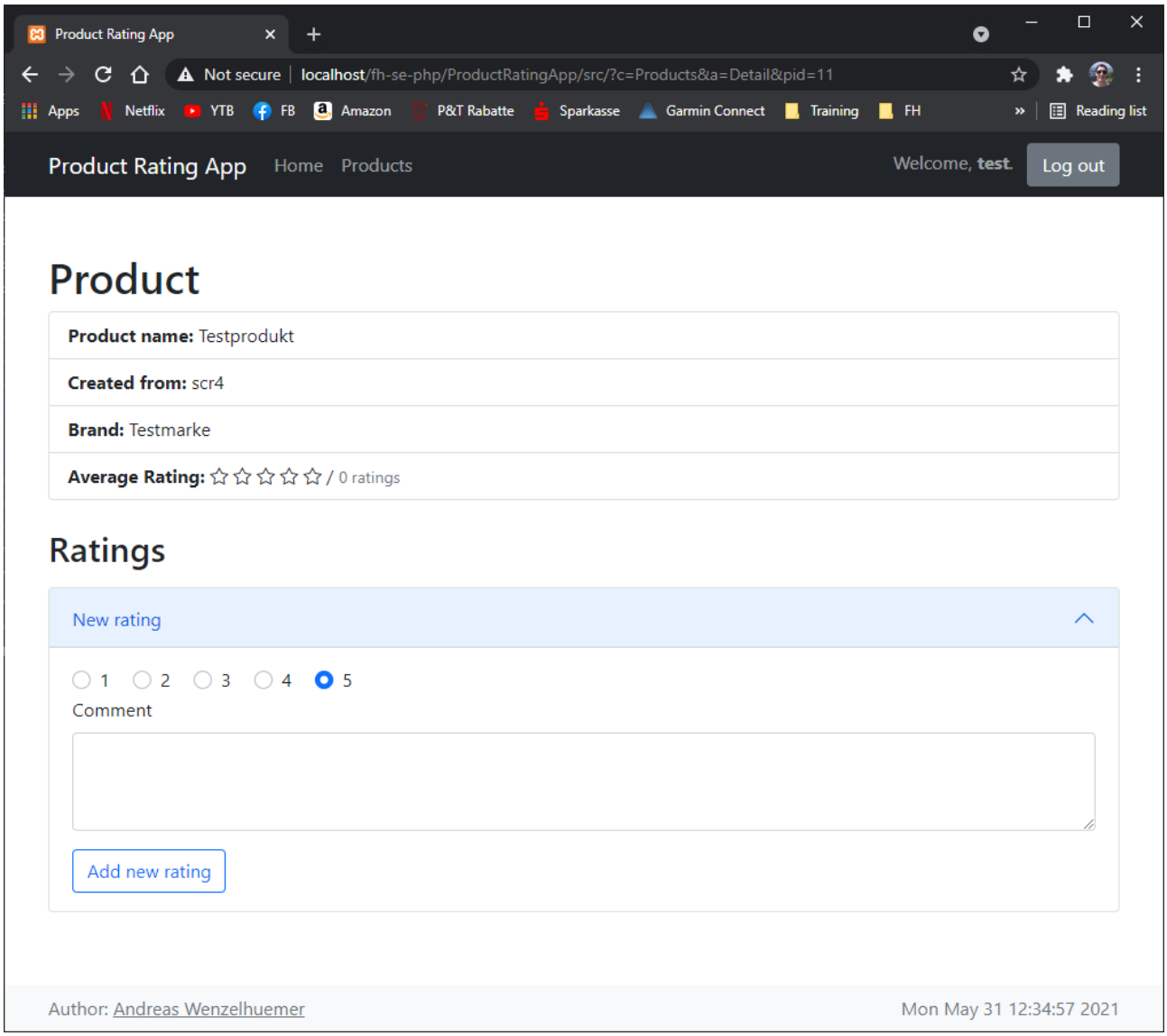



Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=Products&a=Detail&pid=11
Apps Netflix YTB FB Amazon P&T Rabatte Sparkasse Garmin Connect Training FH Reading list
Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=Products&a=Detail&pid=11
Apps Netflix YTB FB Amazon P&T Rabatte Sparkasse Garmin Connect Training FH Reading list
Product Rating App Home Products
Welcome, test. Log out
Average Rating: / 1 ratings
Ratings
New rating
1 2 3 4 5
Comment
Add new rating
by test
from 2021-05-31 12:34:06
1 2 3 4 5
Remove
Comment
Test

2.18. Testfall: Bewertungen

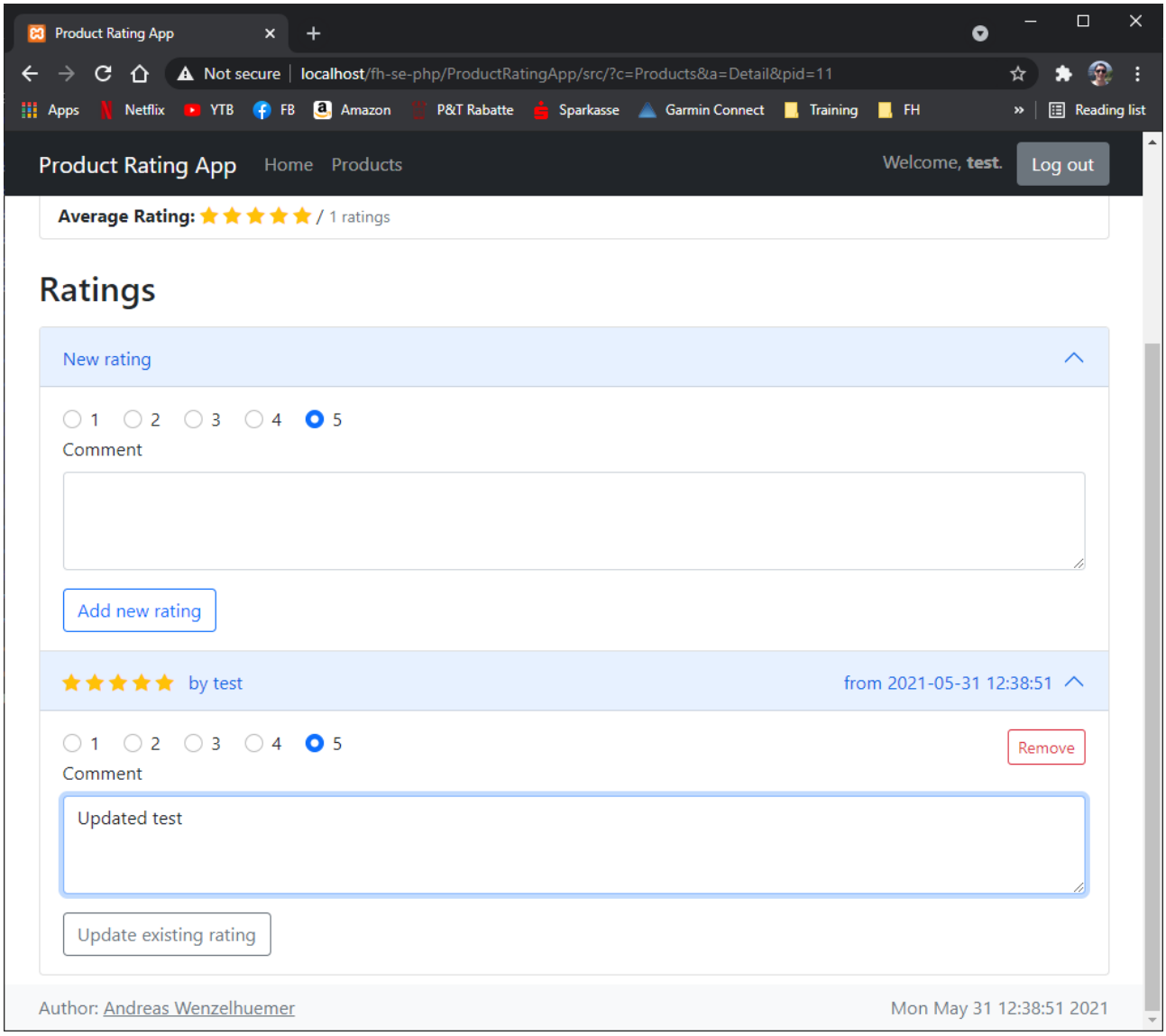
Bewertung entfernen

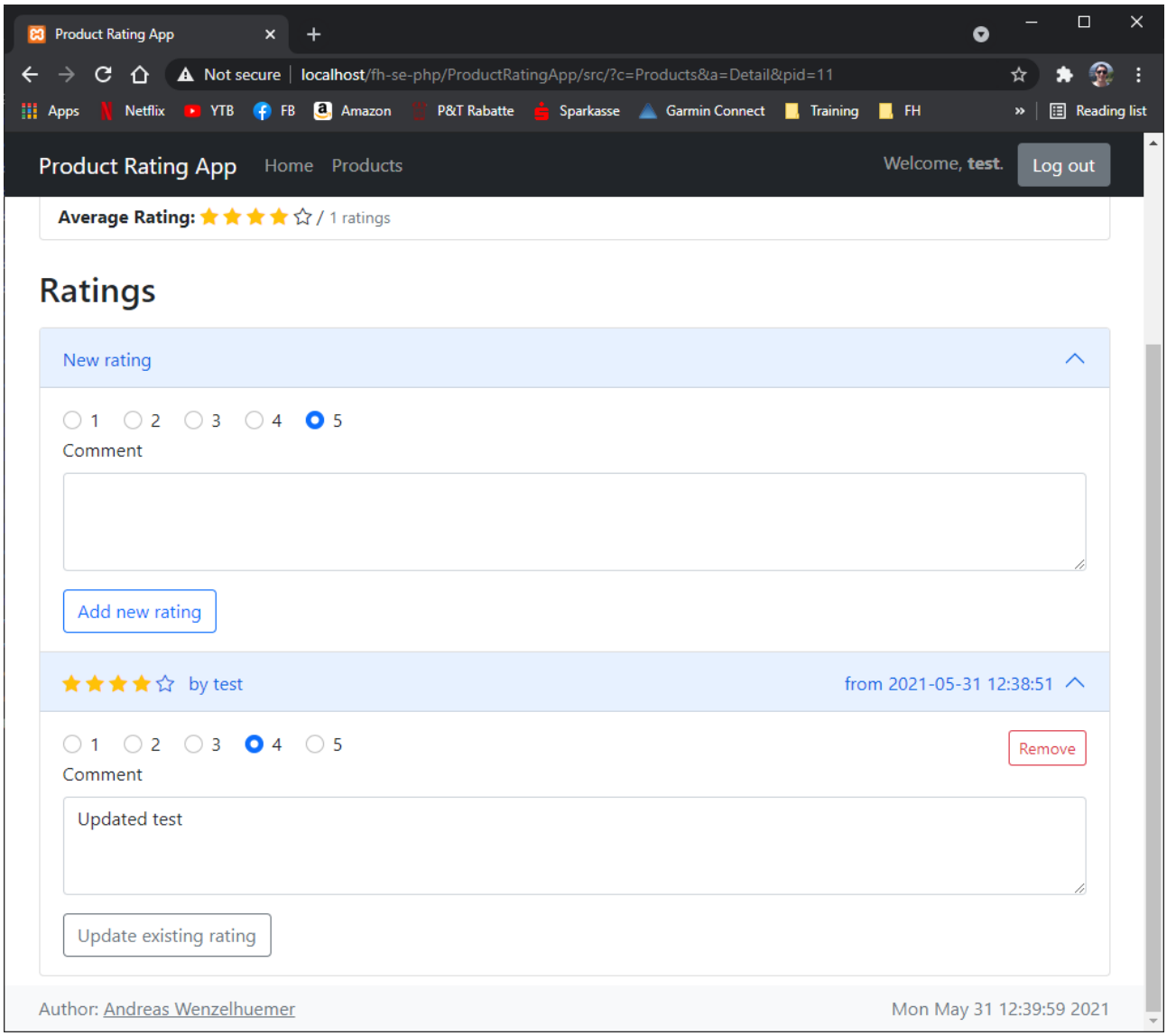



Product Rating App
Not secure | localhost/fh-se-php/ProductRatingApp/src/?c=Products&a=Detail&pid=11
Apps Netflix YTB FB Amazon P&T Rabatte Sparkasse Garmin Connect Training FH Reading list
Product Rating App Home Products
Welcome, test. Log out
Product
Product name: Testprodukt
Created from: scr4
Brand: Testmarke
Average Rating: / 0 ratings
Ratings
New rating
1 2 3 4 5
Comment
Add new rating
Author: Andreas Wenzelhuemer
Mon May 31 12:34:57 2021

2.19. Testfall: Bewertungen

Bewertung aktualisieren




Product Rating App
Not secure
localhost/fh-se-php/ProductRatingApp/src/?c=Products&a=Detail&pid=11
Apps
Netflix
YTB
FB
Amazon
P&T Rabatte
Sparkasse
Garmin Connect
Training
FH
Reading list
Product Rating App
Home
Products
Welcome, test.
Log out
Average Rating: / 1 ratings
Ratings
New rating
1 2 3 4 5
Comment
Add new rating
by test
from 2021-05-31 12:38:51
1 2 3 4 5
Remove
Comment
Updated test
Update existing rating
Author: Andreas Wenzelhuemer
Mon May 31 12:39:59 2021

3. Quellcode

Listing 1. Index.php

```
<?php

spl_autoload_register(function ($class) {
    $file = __DIR__ . '/src/' . str_replace('\\', '/', $class) .
'.php';
    if (file_exists($file)) {
        require_once($file);
    }
});

function registerCommandsAndQueries(\ServiceProvider $sp): void {
      // queries
      $sp->register(\Application\Queries\SignedInUserQuery::class);
      $sp->register(\Application\Queries\ProductsQuery::class);
      $sp->register(\Application\Queries\ProductQuery::class);
      $sp->register(\Application\Queries\ProductDetailQuery::class);

      // commands
      $sp->register(\Application\Commands\AddProductCommand::class);
      $sp->register(\Application\Commands\EditProductCommand::class);
      $sp->register(\Application\Commands\SignInCommand::class);
      $sp->register(\Application\Commands\SignOutCommand::class);
      $sp->register(\Application\Commands\RegisterCommand::class);
      $sp->register(\Application\Commands\AddRatingCommand::class);
      $sp->register(\Application\Commands\RemoveRatingCommand::class);
      $sp->register(\Application\Commands\EditRatingCommand::class);
}

function registerServices(\ServiceProvider $sp): void {
    $sp->register(\Application\Services\AuthenticationService::class);
}

function registerRepositories(\ServiceProvider $sp): void {
    $sp->register(\Infrastructure\FakeRepository::class, isSingleton:
true);
    $sp->register(\Infrastructure\Repository::class, function() {
return new \Infrastructure\Repository("localhost", "root", "",
"productrating");});
    $sp->register(\Application\Interfaces\UserRepository::class,
\Infrastructure\Repository::class, isSingleton: true);
```

```
    $sp->register(\Application\Interfaces\ProductRepository::class,
\Infrastructure\Repository::class, isSingleton: true);
    $sp->register(\Application\Interfaces\RatingRepository::class,
\Infrastructure\Repository::class, isSingleton: true);
}

function registerControllers(\ServiceProvider $sp): void {
    $sp->register(\Presentation\MVC\MVC::class, function () {
        return new \Presentation\MVC\MVC();
    }, isSingleton: true);

    $sp->register(\Presentation\Controllers\Error404::class);
    $sp->register(\Presentation\Controllers\Home::class);
    $sp->register(\Presentation\Controllers\User::class);
    $sp->register(\Presentation\Controllers\Products::class);
    $sp->register(\Presentation\Controllers\Ratings::class);
}

// === register services
$sp = new \ServiceProvider();

// --- Application
registerCommandsAndQueries($sp);

// --- Services
registerServices($sp);

// --- Infrastructure
  $sp->register(\Infrastructure\Session::class, isSingleton: true);
  $sp->register(\Application\Interfaces\Session::class,
\Infrastructure\Session::class);
registerRepositories($sp);

// --- Presentation
registerControllers($sp);

// === handle requests
$sp->resolve(\Presentation\MVC\MVC::class)->handleRequest($sp);
```

3.1. Application

3.1.1. Commands

Listing 2. AddProductCommand.php

```php
<?php

namespace Application\Commands;

class AddProductCommand
{
    const Error_NotAuthenticated = 0x01; // 1
    const Error_InvalidName = 0x02; // 2
    const Error_InvalidProducer = 0x04; // 4
    const Error_CreateProductFailed = 0x08; // 8

    public function __construct(
        private \Application\Services\AuthenticationService
$authenticationService,
        private \Application\Interfaces\ProductRepository
$productRepository
    ) {

    }

    public function execute(string $producer, string $name): int
    {
        $errors = 0;
        $name = trim($name);
        $producer = trim($producer);

        $userId = $this->authenticationService->getUserId();

        // check for authenticated user
        if($userId === null) {
            $errors |= self::Error_NotAuthenticated;
        }

        if(strlen($name) == 0) {
            $errors |= self::Error_InvalidName;
        }

        if(strlen($producer) == 0) {
            $errors |= self::Error_InvalidProducer;
        }
```

```
        if(!$errors) {
            $productId = $this->productRepository->addProduct(
$producer, $userId, $name);

            if($productId === null) {
                $errors |= self::Error_CreateProductFailed;
            }
        }

        return $errors;
    }
}
```

Listing 3. AddRatingCommand.php

```
<?php

namespace Application\Commands;

use Application\Interfaces\RatingRepository;
use Application\Services\AuthenticationService;

class AddRatingCommand {

    const Error_NotAuthenticated = 0x01; // 1
    const Error_CreateRatingFailed = 0x2; // 2

    public function __construct(
        private AuthenticationService $authenticationService,
        private RatingRepository $ratingRepository
    ) { }

    public function execute(int $productId, int $rating, ?string
$comment): int
    {
        $userId = $this->authenticationService->getUserId();
        $errors = 0;

        $comment = trim($comment);
        $comment = $comment === "" ? null : $comment;

        // check for authenticated user
        if($userId === null) {
```

```
            $errors |= self::Error_NotAuthenticated;
        }

        if(!$errors) {
            // try to create new rating
            $ratingId = $this->ratingRepository->addRating(
                $userId,
                $productId,
                $rating,
                $comment
            );

            if($ratingId === null) {
                $errors |= self::Error_CreateRatingFailed;
            }
        }

        return $errors;
    }
}
```

Listing 4. EditProductCommand.php

```
<?php

namespace Application\Commands;

class EditProductCommand
{
    const Error_NotAuthenticated = 0x01; // 1
    const Error_InvalidName = 0x02; // 2
    const Error_InvalidProducer = 0x04; // 4

    public function __construct(
        private \Application\Services\AuthenticationService
$authenticationService,
        private \Application\Interfaces\ProductRepository
$productRepository
    ) {

    }

    public function execute(int $id, string $producer, string $name):
int
```

```
    {
        $errors = 0;
        $name = trim($name);
        $producer = trim($producer);

        $userId = $this->authenticationService->getUserId();

        // check for authenticated user
        if($userId === null  || $this->productRepository-
>canEditProduct($id, $userId) == 0) {
            $errors |= self::Error_NotAuthenticated;
        }

        if(strlen($name) == 0) {
            $errors |= self::Error_InvalidName;
        }

        if(strlen($producer) == 0) {
            $errors |= self::Error_InvalidProducer;
        }

        if(!$errors) {
            $this->productRepository->editProduct($id, $producer,
$name);
        }

        return $errors;
    }
}
```

Listing 5. EditRatingCommand.php

```
<?php

namespace Application\Commands;

use Application\Interfaces\RatingRepository;
use Application\Services\AuthenticationService;

class EditRatingCommand {

    const Error_NotAuthenticated = 0x01; // 1

    public function __construct(
```

```
        private AuthenticationService $authenticationService,
        private RatingRepository $ratingRepository
    ) { }

    public function execute(int $ratingId, int $productId, int $rating,
?string $comment): int
    {
        $userId = $this->authenticationService->getUserId();
        $errors = 0;

        $comment = trim($comment);
        $comment = $comment === "" ? null : $comment;

        // check for authenticated user
        if($userId === null || $this->ratingRepository->canEditRating
($ratingId, $userId) == 0) {
            $errors |= self::Error_NotAuthenticated;
        }

        if(!$errors) {
            // try to edit rating
            $this->ratingRepository->editRating(
                $ratingId,
                $productId,
                $rating,
                $comment
            );
        }

        return $errors;
    }
}
```

Listing 6. RegisterCommand.php

```
<?php

namespace Application\Commands;

use Application\Interfaces\UserRepository;

class RegisterCommand
{
```

```
    const Error_UsernameAlreadyExists = 0x01; // 1
    const Error_CreateUserFailed = 0x02; // 2
    const Error_InvalidUsername = 0x04; // 4
    const Error_InvalidPassword = 0x08; // 8

    public function __construct(
        private UserRepository $userRepository
    ) { }

    public function execute(string $username, string $password): int
    {
        $errors = 0;
        $username = trim($username);

        if($this->userRepository->getUserForUserName($username) !==
null) {
            $errors |= self::Error_UsernameAlreadyExists;
        } else {

            if(strlen($username) == 0) {
                $errors |= self::Error_InvalidUsername;
            }

            if(strlen($password) < 4) {
                $errors |= self::Error_InvalidPassword;
            }

            if(!$errors) {
                $userId = $this->userRepository->createUser($username,
$password);
                if($userId === null) {
                    $errors |= self::Error_CreateUserFailed;
                }
            }
        }

        return $errors;
    }
}
```

Listing 7. RemoveRatingCommand.php

```php
<?php

namespace Application\Commands;

use Application\Interfaces\RatingRepository;
use Application\Services\AuthenticationService;

class RemoveRatingCommand
{
    const Error_NotAuthenticated = 0x01; // 1

    public function __construct(
        private AuthenticationService $authenticationService,
        private RatingRepository $ratingRepository
    ) { }

    public function execute(int $ratingId): int
    {
        $userId = $this->authenticationService->getUserId();
        $errors = 0;

        // check for authenticated user
        if($userId === null || !$this->ratingRepository->canEditRating
($ratingId, $userId)) {
            $errors |= self::Error_NotAuthenticated;
        }

        if(!$errors) {
            // try to remove rating
            $this->ratingRepository->removeRating($ratingId);
        }

        return $errors;
    }
}
```

Listing 8. SignInCommand.php

```
<?php

namespace Application\Commands;

use Application\Interfaces\UserRepository;
use Application\Services\AuthenticationService;

class SignInCommand
{
    public function __construct(
        private AuthenticationService $authenticationService,
        private UserRepository $userRepository
        ) {
    }

    public function execute(string $username, string $password): bool
    {
        $this->authenticationService->signOut();
        $user = $this->userRepository->getUserForUserNameAndPassword
($username, $password);
        if($user != null) {
            $this->authenticationService->signIn($user->getId());
            return true;
        }

        return false;
    }
}
```

Listing 9. SignOutCommand.php

```php
<?php

namespace Application\Commands;

use Application\Services\AuthenticationService;

class SignOutCommand
{
    public function __construct(
        private AuthenticationService $authenticationService
    ) {
    }

    public function execute(): void
    {
        $this->authenticationService->signOut();
    }
}
```

3.1.2. Entities

Listing 10. Product.php

```php
<?php

namespace Application\Entities;

class Product {

    /**
     * Product constructor.
     * @param int $id
     * @param string $producer
     * @param string $userId
     * @param string $name
     */
    public function __construct(
        private int $id,
        private string $producer,
        private string $userId,
        private string $name)
    { }

    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return string
     */
    public function getProducer(): string
    {
        return $this->producer;
    }

    /**
     * @return string
     */
    public function getUserId(): string
```

```
    {
        return $this->userId;
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }
}
```

Listing 11. Rating.php

```
<?php

namespace Application\Entities;

class Rating {

    /**
     * Rating constructor.
     */
    public function __construct(
        private int $id,
        private int $userId,
        private int $productId,
        private int $rating,
        private ?string $comment,
        private string $createdDate
    ) { }

    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return int
     */
```

```
    public function getUserId(): int
    {
        return $this->userId;
    }

    /**
     * @return int
     */
    public function getProductId(): int
    {
        return $this->productId;
    }

    /**
     * @return int
     */
    public function getRating(): int
    {
        return $this->rating;
    }

    /**
     * @return string
     */
    public function getComment(): ?string
    {
        return $this->comment;
    }

    /**
     * @return string
     */
    public function getCreatedDate(): string
    {
        return $this->createdDate;
    }
}
```

Listing 12. User.php

```php
<?php

namespace Application\Entities;

class User {
    public function __construct(
        private int $id,
        private string $userName
    )
    { }

    public function getId(): int {
        return $this->id;
    }

    public function getUserName(): string {
        return $this->userName;
    }
}
```

3.1.3. Interfaces

Listing 13. ProductRepository.php

```
<?php

namespace Application\Interfaces;

use Application\Entities\Product;

interface ProductRepository
{
    public function getProduct(int $id): ?Product;
    public function getProducts(): array;
    public function getProductsForFilter(string $filter): array;
    public function addProduct(string $producer, int $userId, string
$name): ?int;
    public function canEditProduct(int $id, int $userId): bool;
    public function editProduct(int $id, string $producer, string
$name);
}
```

Listing 14. RatingRepository.php

```
<?php

namespace Application\Interfaces;

interface RatingRepository {

    public function getRatingAverageForProduct(int $productId): float;
    public function getRatingCountForProduct(int $productId): int;
    public function getRatingsFromProduct(int $productId): array;
    public function addRating(int $userId, int $productId, int $rating,
?string $comment): ?int;
    public function editRating(int $id, int $productId, int $rating,
?string $comment): void;
    public function removeRating(int $id);
    public function canEditRating(int $id, int $userId): bool;
}
```

Listing 15. Session.php

```
<?php

namespace Application\Interfaces;

interface Session {
    public function get(string $key): mixed;
    public function put(string $key, mixed $value): void;
    public function delete(string $key): void;
}
```

Listing 16. UserRepository.php

```
<?php

namespace Application\Interfaces;

interface UserRepository {
    public function getUserForUserName(string $userName) : ?
\Application\Entities\User;
    public function getUserForUserNameAndPassword(string $userName,
string $password) : ?\Application\Entities\User;
    public function getUser(int $id): ?\Application\Entities\User;
    public function createUser(string $userName, string $password) :
int;
}
```

3.1.4. Models

Listing 17. ProductData.php

```
<?php

namespace Application\Models;

class ProductData {

    public function __construct(
        private int $id,
        private string $producer,
        private UserData $user,
        private string $name,
        private float $rating,
        private int $ratingCount
    ) { }

    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return string
     */
    public function getProducer(): string
    {
        return $this->producer;
    }

    /**
     * @return string
     */
    public function getUserName(): ?string
    {
        if($this->user) {
            return $this->user->getUserName();
        } else {
            return null;
```

```
        }
    }

    /**
     * @return int
     */
    public function getUserId(): ?int
    {
        if($this->user) {
            return $this->user->getId();
        } else {
            return null;
        }
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @return int
     */
    public function getRating(): float
    {
        return $this->rating;
    }

    /**
     * @return int
     */
    public function getRatingCount(): int
    {
        return $this->ratingCount;
    }


}
```

Listing 18. ProductDetailData.php

```php
<?php

namespace Application\Models;

class ProductDetailData extends ProductData
{

    /**
     * ProductDetailData constructor.
     */
    public function __construct(
        int $id,
        string $producer,
        UserData $user,
        string $name,
        private array $ratings
    ) {

        $ratingSum = 0;
        $ratingCount = 0;
        if(sizeof($this->ratings) > 0) {
            foreach ($this->ratings as $rating) {
                $ratingCount++;
                $ratingSum += $rating->getRating();
            }
        }

        $ratingAverage = $ratingCount > 0 ? round($ratingSum /
$ratingCount, 2) : 0;

        parent::__construct($id, $producer, $user, $name,
$ratingAverage, $ratingCount);
    }

    /**
     * @return array
     */
    public function getRatings(): array
    {
        return $this->ratings;
    }


}
```

Listing 19. RatingData.php

```php
<?php

namespace Application\Models;

class RatingData {
    public function __construct(
        private int $id,
        private UserData $user,
        private int $rating,
        private ?string $comment,
        private String $createdDate
    ) { }

    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return UserData
     */
    public function getUser(): UserData
    {
        return $this->user;
    }

    /**
     * @return int
     */
    public function getRating(): int
    {
        return $this->rating;
    }

    /**
     * @return string
     */
    public function getComment(): ?string
    {
```

```
        return $this->comment;
    }

    /**
     * @return String
     */
    public function getCreatedDate(): String
    {
        return $this->createdDate;
    }


}
```

Listing 20. UserData.php

```
<?php

namespace Application\Models;

class UserData
{
    public function __construct(
        private int $id,
        private string $userName
    ) {
    }

    public function getId(): int
    {
        return $this->id;
    }

    public function getUserName(): string
    {
        return $this->userName;
    }
}
```

3.1.5. Queries

Listing 21. ProductDetailQuery.php

```
<?php

namespace Application\Queries;

use Application\Interfaces\ProductRepository;
use Application\Interfaces\RatingRepository;
use Application\Interfaces\UserRepository;
use Application\Models\ProductDetailData;
use Application\Models\RatingData;
use Application\Models\UserData;

class ProductDetailQuery {

    public function __construct(
        private ProductRepository $productRepository,
        private RatingRepository $ratingRepository,
        private UserRepository $userRepository,
    )
    {
    }

    public function execute(int $id): ?ProductDetailData {
        $product = $this->productRepository->getProduct($id);

        if($product === null) {
            return null;
        }

        $ratingResult = $this->ratingRepository->getRatingsFromProduct
($id);
        $userResult = $this->userRepository->getUser($product-
>getUserId());

        $ratings = [];

        foreach ($ratingResult as $rating) {

            // Load user if necessary
            $ratingUser = $rating->getUserId() != $userResult->getId()
? $this->userRepository->getUser($rating->getUserId()) : $userResult;
```

```
            $ratings[] = new RatingData(
                $rating->getId(),
                new UserData($ratingUser->getId(), $ratingUser-
>getUserName()),
                $rating->getRating(),
                $rating->getComment(),
                $rating->getCreatedDate()
            );
        }

        return new ProductDetailData(
            $product->getId(),
            $product->getProducer(),
            new UserData($userResult->getId(), $userResult->
getUserName()),
            $product->getName(),
            $ratings
        );
    }
}
```

Listing 22. ProductQuery.php

```
<?php

namespace Application\Queries;

use Application\Interfaces\ProductRepository;
use Application\Interfaces\RatingRepository;
use Application\Interfaces\UserRepository;
use Application\Models\ProductData;
use Application\Models\ProductDetailData;
use Application\Models\RatingData;
use Application\Models\UserData;

class ProductQuery {

    public function __construct(
        private ProductRepository $productRepository,
        private UserRepository $userRepository,
        private RatingRepository $ratingRepository
    )
    {
```

```
    }

    public function execute(int $id): ?ProductData {
        $product = $this->productRepository->getProduct($id);

        if($product === null) {
            return null;
        }

        $userResult = $this->userRepository->getUser($product-
>getUserId());

        return new ProductData(
            $product->getId(),
            $product->getProducer(),
            new UserData($userResult->getId(), $userResult->
getUserName()),
            $product->getName(),
            $this->ratingRepository->getRatingAverageForProduct
($product->getId()),
            $this->ratingRepository->getRatingCountForProduct($product
->getId())
        );
    }
}
```

Listing 23. ProductsQuery.php

```
<?php

namespace Application\Queries;

use Application\Interfaces\ProductRepository;
use Application\Interfaces\RatingRepository;
use Application\Interfaces\UserRepository;
use Application\Models\ProductData;
use Application\Models\UserData;

class ProductsQuery
{
    public function __construct(
        private ProductRepository $productRepository,
        private UserRepository $userRepository,
        private RatingRepository $ratingRepository
```

```
    ) {
    }

    public function execute(?string $filter): array{
        $results = [];

        $products = $filter === null || $filter == "" ? $this-
>productRepository->getProducts() : $this->productRepository-
>getProductsForFilter($filter);

        foreach($products as $product) {

            $user = $this->userRepository->getUser($product->
getUserId());
            $averageRating = $this->ratingRepository-
>getRatingAverageForProduct($product->getId());
            $totalCount = $this->ratingRepository-
>getRatingCountForProduct($product->getId());

            $results[] = new ProductData(
                $product->getId(),
                $product->getProducer(),
                new UserData($user->getId(), $user->getUserName()),
                $product->getName(),
                $averageRating,
                $totalCount
                );
        }

        return $results;
    }
}
```

Listing 24. SignedInUserQuery.php

```php
<?php

namespace Application\Queries;

use Application\Interfaces\UserRepository;
use Application\Services\AuthenticationService;
use Application\Models\UserData;

class SignedInUserQuery
{
    public function __construct(
        private AuthenticationService $authenticationService,
        private UserRepository $userRepository
    ) {
    }

    public function execute(): ?UserData
    {
        $id = $this->authenticationService->getUserId();
        if ($id === null) {
          return null;
        }
        $user = $this->userRepository->getUser($id);
        if ($user === null) {
            return null;
        }
        return new UserData($user->getId(), $user->getUserName());
    }
}
```

3.1.6. Services

Listing 25. AuthenticationService.php

```php
<?php

namespace Application\Services;

class AuthenticationService {

    const SESSION_USER_ID = "userId";

    public function __construct(
        private \Application\Interfaces\Session $session
    ) { }

    public function signIn(int $userId) {
        $this->session->put(self::SESSION_USER_ID, $userId);
    }

    public function signOut() : void {
        $this->session->delete(self::SESSION_USER_ID);
    }

    public function getUserId(): ?int {
        return $this->session->get(self::SESSION_USER_ID);
    }
}
```

3.2. Infrastructure

Listing 26. FakeRepository.php

```
<?php

namespace Infrastructure;

use Application\Entities\Book;
use Application\Entities\Category;
use Application\Entities\Product;
use Application\Entities\Rating;
use Application\Entities\User;
use Application\Interfaces\BookRepository;
use Application\Interfaces\CategoryRepository;
use Application\Interfaces\OrderRepository;
use Application\Interfaces\ProductRepository;
use Application\Interfaces\RatingRepository;
use \Application\Interfaces\UserRepository;

class FakeRepository implements
UserRepository,
    ProductRepository,
    RatingRepository
{
    private $mockUsers;
    private $mockProducts;
    private $mockRatings;

    public function __construct()
    {
        // create mock data
        $this->mockUsers = array(
            array(1, 'scr4', 'scr4')
        );

        $this->mockProducts = array(
            array(1, 'Hilti', 1, 'Schlagbohrmaschine'),
            array(2, 'Bosch', 1, 'Schlagbohrmaschine'),
            array(3, 'Makita', 1, 'Schlagbohrmaschine')
        );

        $this->mockRatings = array(
          array(1, 1, 1, 5, "Hilti quality!", date_format(new \
```

```
DateTime(),"Y-m-d H:i:s"))
        );
    }

    public function getUser(int $id): ?User {
        foreach($this->mockUsers as $u) {
            if($u[0] === $id) {
                return new User($u[0], $u[1]);
            }
        }
        return null;
    }

    public function getUserForUserNameAndPassword(string $userName,
string $password): ?User
    {
        foreach($this->mockUsers as $u) {
            if($u[1] === $userName && $u[2] === $password) {
                return new User($u[0], $u[1]);
            }
        }
        return null;
    }

    public function getUserForUserName(string $userName): ?
\Application\Entities\User
    {
        foreach($this->mockUsers as $u) {
            if($u[1] === $userName) {
                return new User($u[0], $u[1]);
            }
        }
        return null;
    }

    public function createUser(string $userName, string $password) :
int
    {
        $count = sizeof($this->mockUsers);
        $newId = 0;
        if($count > 0) {
            $user = $this->mockUsers[$count - 1];
            $newId = $user[0];
            $newId++;
```

```
        }

        array_push($this->mockUsers,
            array($newId, $userName, $password)
        );

        return $newId;
    }

    public function getProducts(): array
    {
        $result = [];
        foreach($this->mockProducts as $product) {
            $result[] = new Product($product[0], $product[1], $product
[2], $product[3]);
        }

        return $result;
    }

    public function getRatingAverageForProduct(int $productId): float
    {
        $ratingSum = 0;
        $ratingCount = 0;
        foreach ($this->mockRatings as $rating) {
            if($rating[3] === $productId) {
                $ratingCount++;
                $ratingSum += $rating[2];
            }
        }
        return $ratingSum > 0 ? $ratingCount / $ratingSum : 0;
    }

    public function getRatingCountForProduct(int $productId): int
    {
        $ratingCount = 0;
        foreach ($this->mockRatings as $rating) {
            if($rating[2] === $productId) {
                $ratingCount++;
            }
        }
        return $ratingCount;
    }
```

```
    public function getProduct(int $id) : ?Product
    {
        foreach ($this->mockProducts as $product) {
            if($product[0] === $id) {
                return new Product($product[0], $product[1], $product[
2], $product[3]);
            }
        }
        return null;
    }

    public function getRatingsFromProduct(int $productId): array
    {
        $result = [];
        foreach($this->mockRatings as $rating) {
            if($rating[2] === $productId) {
                $result[] = new Rating($rating[0], $rating[1], $rating
[2], $rating[3], $rating[4], $rating[5]);
            }
        }

        return $result;
    }

    public function addRating(int $userId, int $productId, int $rating,
?string $comment): ?int
    {
        $count = sizeof($this->mockUsers);
        $newId = 0;
        if($count > 0) {
            $newId = $this->mockUsers[$count - 1][0];
            $newId++;
        }

        $this->mockRatings[] = new Rating(
            $newId,
            $userId,
            $productId,
            $rating,
            $comment,
            date_format(new \DateTime(),"Y-m-d H:i:s")
        );

        return $newId;
```

```
    }

    public function editRating(int $id, int $productId, int $rating,
?string $comment): void
    {
        for ($i = 0; $i < sizeof($this->mockRatings); $i++) {
            if ($this->mockRatings[$i][0] === $id) {
                $this->mockRatings[$i] = new Rating($id, $this-
>mockRatings[$i][1],$productId, $comment, $rating, $this->mockRatings
[$i][5]);
            }
        }
    }

    public function canEditRating(int $id, int $userId): bool
    {
        foreach($this->mockRatings as $rating) {
            if($rating[0] === $id && $rating[1] == $userId) {
                return true;
            }
        }
        return false;
    }

    public function removeRating(int $id): void
    {
        for ($i = 0; $i < sizeof($this->mockRatings); $i++) {
            if($this->mockRatings[$i][0] === $id) {
                unset($this->mockRatings[$i]);
            }
        }
    }

    public function addProduct(string $producer, int $userId, string
$name): ?int
    {
        $count = sizeof($this->mockUsers);
        $newId = 0;
        if($count > 0) {
            $newId = $this->mockUsers[$count - 1][0];
            $newId++;
        }

        $this->mockProducts[] = new Product(
```

```
            $newId,
            $producer,
            $userId,
            $name
        );

        return $newId;
    }

    public function canEditProduct(int $id, int $userId): bool
    {
        foreach($this->mockProducts as $product) {
            if($product[0] === $id && $product[2] == $userId) {
                return true;
            }
        }
        return false;
    }

    public function editProduct(int $id, string $producer, string
$name)
    {
        for ($i = 0; $i < sizeof($this->mockProducts); $i++) {
            if ($this->mockProducts[$i][0] === $id) {
                $this->mockProducts[$i] = new Product($id, $producer,
$this->mockProducts[$i][2], $name);
            }
        }
    }

    public function getProductsForFilter(string $filter): array
    {
        $products = [];
        foreach($this->mockProducts as $product) {
            if($filter == '' || stripos($product[1], $filter) !== false
|| stripos($product[3], $filter) !== false) {
                $products[] = new Product($product[0], $product[2],
$product[3], $product[4]);
            }
        }
        return $products;
    }
}
```

Listing 27. Repository.php

```
<?php

namespace Infrastructure;

use Application\Entities\Product;
use Application\Entities\Rating;

class Repository
    implements
    \Application\Interfaces\UserRepository,
    \Application\Interfaces\ProductRepository,
    \Application\Interfaces\RatingRepository
{

    private $server;
    private $userName;
    private $password;
    private $database;

    public function __construct(string $server, string $userName,
string $password, string $database)
    {
        $this->server = $server;
        $this->userName = $userName;
        $this->password = $password;
        $this->database = $database;
    }

    // === private helper methods ===

    private function getConnection()
    {
        $con = new \mysqli($this->server, $this->userName, $this-
>password, $this->database);
        if (!$con) {
            die('Unable to connect to database. Error: ' .
mysqli_connect_error());
        }
        return $con;
    }

    private function executeQuery($connection, $query)
```

```
    {
        $result = $connection->query($query);
        if (!$result) {
            die("Error in query '$query': " . $connection->error);
        }
        return $result;
    }

    private function executeStatement($connection, $query, $bindFunc)
    {
        $statement = $connection->prepare($query);
        if (!$statement) {
            die("Error in prepared statement '$query': " . $connection
->error);
        }
        $bindFunc($statement);
        if (!$statement->execute()) {
            die("Error executing prepared statement '$query': " .
$statement->error);
        }
        return $statement;
    }

    // === public methods ===

    public function getUserForUserName(string $userName): ?
\Application\Entities\User
    {
        $user = null;
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT id, passwordHash FROM users WHERE userName = ?',
            function ($s) use ($userName) {
                $s->bind_param('s', $userName);
            }
        );
        $stat->bind_result($id, $passwordHash);
        if ($stat->fetch()) {
            $user = new \Application\Entities\User($id, $userName,
$passwordHash);
        }
        $stat->close();
        $con->close();
```

```
        return $user;
    }

    public function getUserForUserNameAndPassword(string $userName,
string $password): ?\Application\Entities\User
    {
        $user = null;
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT id, passwordHash FROM users WHERE userName = ?',
            function ($s) use ($userName) {
                $s->bind_param('s', $userName);
            }
        );
        $stat->bind_result($id, $passwordHash);
        if ($stat->fetch() && password_verify($password,
$passwordHash)) {
            $user = new \Application\Entities\User($id, $userName);
        }
        $stat->close();
        $con->close();
        return $user;
    }

    public function getUser(int $id): ?\Application\Entities\User
    {
        $user = null;
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT id, userName FROM users WHERE id = ?',
            function ($s) use ($id) {
                $s->bind_param('i', $id);
            }
        );
        $stat->bind_result($id, $userName);
        if ($stat->fetch()) {
            $user = new \Application\Entities\User($id, $userName);
        }
        $stat->close();
        $con->close();
        return $user;
    }
```

```
    public function createUser(string $userName, string $password): int
    {
        $password = password_hash($password, PASSWORD_DEFAULT);
        $con = $this->getConnection();
        $con->autocommit(false);
        $stat = $this->executeStatement(
            $con,
            'INSERT INTO users (userName, passwordHash) VALUES (?, ?)',
            function ($s) use ($userName, $password) {
                $s->bind_param('ss', $userName, $password);
            }
        );
        $userId = $stat->insert_id;
        $stat->close();
        $con->commit();
        $con->close();
        return $userId;
    }

    public function getProducts(): array
    {
        $products = [];
        $con = $this->getConnection();
        $result = $this->executeQuery(
            $con,
            'SELECT id, producer, userId, name FROM products'
        );
        while ($product = $result->fetch_object()) {
            $products[] = new \Application\Entities\Product($product-
>id, $product->producer, $product->userId, $product->name);
        }
        $result->close();
        $con->close();
        return $products;
    }

    public function getRatingAverageForProduct(int $productId): float
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT AVG(rating) as averageRating FROM `ratings` WHERE
productId = ?',
```

```
            function($s) use ($productId) {
                $s->bind_param('i', $productId);
            }
        );

        $stat->bind_result($averageRating);
        $stat->fetch();

        $stat->close();
        $con->close();

        return round($averageRating, 2);
    }

    public function getRatingCountForProduct(int $productId): int
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT COUNT(ID) as count FROM `ratings` WHERE productId =
?',
            function($s) use ($productId) {
                $s->bind_param('i', $productId);
            }
        );

        $stat->bind_result($count);
        $stat->fetch();

        $stat->close();
        $con->close();

        return $count;
    }

    public function getProduct(int $id): ?Product
    {
        $product = null;
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT id, producer, userId, name FROM `products` WHERE id
= ?',
            function($s) use ($id) {
```

```
                $s->bind_param('i', $id);
            }
        );

        $stat->bind_result($id, $producer, $userId, $name);
        if ($stat->fetch()) {
            $product = new \Application\Entities\Product($id,
$producer, $userId, $name);
        }

        $stat->close();
        $con->close();

        return $product;
    }

    public function getRatingsFromProduct(int $productId): array
    {
        $ratings = [];
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT id, userId, productId, rating, comment, createdDate
FROM `ratings` WHERE productId = ? ORDER BY createdDate DESC',
            function($s) use ($productId) {
                $s->bind_param('i', $productId);
            }
        );

        $stat->bind_result($id, $userId, $productId, $rating, $comment,
$createdDate);
        while ($stat->fetch()) {
            $ratings[] = new Rating($id, $userId, $productId, $rating,
$comment, $createdDate);
        }

        $stat->close();
        $con->close();

        return $ratings;
    }

    public function addRating(int $userId, int $productId, int $rating,
?string $comment): ?int
```

```
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'INSERT INTO `ratings` (`userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES (?, ?, ?, ?, NOW())',
            function($s) use ($userId, $productId, $rating, $comment) {
                $s->bind_param('iiis', $userId, $productId, $rating,
$comment);
            }
        );

        $newRatingId = $stat->insert_id;
        $stat->close();
        $con->close();

        return $newRatingId;
    }

    public function editRating(int $id, int $productId, int $rating,
?string $comment): void
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'UPDATE ratings SET productId = ?, rating = ?, comment = ?
WHERE id = ?',
            function($s) use ($id, $productId, $rating, $comment) {
                $s->bind_param('iisi', $productId, $rating, $comment,
$id);
            }
        );
        $stat->close();
        $con->close();
    }

    public function canEditRating(int $id, int $userId): bool
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT COUNT(id) as count FROM `ratings` WHERE id = ? &&
userId = ?',
            function($s) use ($id, $userId) {
```

```
                $s->bind_param('ii', $id, $userId);
            }
        );

        $stat->bind_result($count);
        $stat->fetch();

        $stat->close();
        $con->close();

        return $count == 1;
    }

    public function removeRating(int $id)
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'DELETE FROM ratings WHERE id = ?',
            function($s) use ($id) {
                $s->bind_param('i', $id);
            }
        );
        $stat->close();
        $con->close();
    }

    public function addProduct(string $producer, int $userId, string
$name): ?int
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'INSERT INTO products (producer, userId, name) VALUES (?,
?, ?);',
            function($s) use ($producer, $userId, $name) {
                $s->bind_param('sis', $producer, $userId, $name);
            }
        );

        $newProductId = $stat->insert_id;
        $stat->close();
        $con->close();
```

```
        return $newProductId;
    }

    public function canEditProduct(int $id, int $userId): bool
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'SELECT COUNT(id) as count FROM products WHERE id = ? &&
userId = ?',
            function($s) use ($id, $userId) {
                $s->bind_param('ii', $id, $userId);
            }
        );

        $stat->bind_result($count);
        $stat->fetch();

        $stat->close();
        $con->close();

        return $count == 1;
    }

    public function editProduct(int $id, string $producer, string
$name)
    {
        $con = $this->getConnection();
        $stat = $this->executeStatement(
            $con,
            'UPDATE products SET producer = ?, name = ? WHERE id = ?',
            function($s) use ($producer, $name, $id) {
                $s->bind_param('ssi', $producer, $name, $id);
            }
        );
        $stat->close();
        $con->close();
    }

    public function getProductsForFilter(string $filter): array
    {
        $filter = "%$filter%";
        $products = [];
        $con = $this->getConnection();
```

```
        $stat = $this->executeStatement(
            $con,
            'SELECT id, producer, userId, name FROM products WHERE
(name LIKE ? || producer LIKE ?)',
            function ($s) use ($filter) {
                $s->bind_param('ss', $filter, $filter);
            }
        );

        $stat->bind_result($id, $producer, $userId, $name);
        while ($stat->fetch()) {
            $products[] = new \Application\Entities\Product($id,
$producer, $userId, $name);
        }
        $stat->close();
        $con->close();

        return $products;
    }
}
```

Listing 28. Session.php

```php
<?php

namespace Infrastructure;

class Session implements \Application\Interfaces\Session {

    public function __construct() {
        session_start();
    }

    public function get(string $key): mixed {
        return $_SESSION[$key] ?? null;
    }

    public function put(string $key, mixed $value): void {
        $_SESSION[$key] = $value;
    }

    public function delete(string $key): void {
        unset($_SESSION[$key]);
    }
}
```

3.3. Presentation

3.3.1. Controllers

Listing 29. Error404.php

```
<?php

namespace Presentation\Controllers;

use Application\Queries\SignedInUserQuery;
use Presentation\MVC\Controller;
use Presentation\MVC\ViewResult;

class Error404 extends Controller {

    public function __construct(
        private SignedInUserQuery $signedInUserQuery
    ) { }

    public function GET_Index(): ViewResult {
        return $this->view("error404"
         , [
             "user" => $this->signedInUserQuery->execute()
         ]
    );
    }
}
```

Listing 30. Home.php

```
<?php

namespace Presentation\Controllers;

use Application\Queries\SignedInUserQuery;
use Presentation\MVC\Controller;
use Presentation\MVC\ViewResult;

class Home extends Controller {

    public function __construct(
        private SignedInUserQuery $signedInUserQuery
    ) { }

    public function GET_Index(): ViewResult {
        return $this->view("home"
         , [
             "user" => $this->signedInUserQuery->execute()
         ]
    );
    }
}
```

Listing 31. Products.php

```
<?php

namespace Presentation\Controllers;

use Presentation\MVC\ActionResult;
use Presentation\MVC\Controller;
use Presentation\MVC\ViewResult;

class Products extends Controller {

    public function __construct(
        private \Application\Queries\ProductsQuery $productsQuery,
        private \Application\Queries\ProductQuery $productQuery,
        private \Application\Queries\ProductDetailQuery
$productDetailQuery,
        private \Application\Commands\AddProductCommand
$addProductCommand,
```

```
        private \Application\Commands\EditProductCommand
$editProductCommand,
        private \Application\Queries\SignedInUserQuery
$signedInUserQuery
    ) {
    }

    public function GET_Index(): ViewResult
    {
        $filter = $this->tryGetParam('f', $filter) ? trim($filter) :
null;

        return $this->view("productList",
            [
                "user" => $this->signedInUserQuery->execute(),
                "products" => $this->productsQuery->execute($filter),
                "filter" => $filter,
            ]);
    }

    public function GET_Detail(): ActionResult
    {
        $errors = [];
        $product = null;
        $idParam = "";
        if(!$this->tryGetParam("pid", $idParam)) {
            $errors[] = "Product does not exist.";
        } else {

            $id = intval($idParam);
            if(intval($idParam) != 0) {
                $product = $this->productDetailQuery->execute($id);
            }

            if($product === null) {
                $errors[] = "Product does not exist.";
            }
        }

        if(sizeof($errors) > 0) {
            return $this->view("productDetail", [
                    "user" => $this->signedInUserQuery->execute(),
                    "product" => null,
                    "errors" => $errors
```

```
            ]);
        } else {
            return $this->view("productDetail",
                [
                    "user" => $this->signedInUserQuery->execute(),
                    "product" => $product
                ]);
        }
    }

    public function GET_Create(): ViewResult {
        return $this->view("newProduct", [
            "user" => $this->signedInUserQuery->execute()
        ]);
    }

    public function Post_Create(): ActionResult {

        $name = $this->getParam("nm");
        $producer = $this->getParam("pd");

        $result = $this->addProductCommand->execute($producer, $name);

        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\AddProductCommand
::Error_NotAuthenticated) {
                $errors[] = "Product can only be added when user is
signed in.";
            }

            if($result & \Application\Commands\AddProductCommand
::Error_InvalidName) {
                $errors[] = "Product name is required.";
            }

            if($result & \Application\Commands\AddProductCommand
::Error_InvalidProducer) {
                $errors[] = "Brand name is required.";
            }

            if($result & \Application\Commands\AddProductCommand
::Error_CreateProductFailed) {
                $errors[] = "Creating of product failed.";
```

```
            }

            return $this->view(
                "newProduct", [
                "user" => $this->signedInUserQuery->execute(),
                "errors" => $errors
            ]);
        }

        return $this->redirect("Products", "Index");
    }

    public function GET_Edit(): ActionResult {

        // check for valid id

        $errors = [];
        $product = null;
        if(!$this->tryGetParam("pid", $idParam)) {
            $errors[] = "Product could not be found!";
        } else {

            $id = intval($idParam);
            if(intval($idParam) != 0) {
                $product = $this->productQuery->execute($id);
            }

            if($product === null) {
                $errors[] = "Product could not be found!";
            }
        }

        if(sizeof($errors) > 0) {
            return $this->view(
                "editProduct", [
                "user" => $this->signedInUserQuery->execute(),
                "product" => null,
                "errors" => $errors
            ]);
        } else {
            return $this->view("editProduct",
                [
                    "user" => $this->signedInUserQuery->execute(),
                    "product" => $product
```

```
                ]);
        }
    }

    public function Post_Edit(): ActionResult {

        $productId = $this->getParam("pid");
        $name = $this->getParam("nm");
        $producer = $this->getParam("pd");

        $result = $this->editProductCommand->execute($productId,
$producer, $name);

        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\EditProductCommand
::Error_NotAuthenticated) {
                $errors[] = "Product can only be updated when user is
signed in.";
            }

            if($result & \Application\Commands\EditProductCommand
::Error_InvalidName) {
                $errors[] = "Product name is required.";
            }

            if($result & \Application\Commands\EditProductCommand
::Error_InvalidProducer) {
                $errors[] = "Brand name is required.";
            }

            return $this->view(
                "editProduct", [
                "user" => $this->signedInUserQuery->execute(),
                "product" => $this->productQuery->execute($productId),
                "errors" => $errors
            ]);
        }

        return $this->redirect("Products", "Index");
    }
}
```

Listing 32. Ratings.php

```
<?php

namespace Presentation\Controllers;

use Presentation\MVC\ActionResult;
use Presentation\MVC\Controller;

class Ratings extends Controller {

    public function __construct(
        private \Application\Commands\AddRatingCommand
$addRatingCommand,
        private \Application\Commands\EditRatingCommand
$editRatingCommand,
        private \Application\Commands\RemoveRatingCommand
$removeRatingCommand,
        private \Application\Queries\ProductDetailQuery
$productDetailQuery,
        private \Application\Queries\SignedInUserQuery
$signedInUserQuery,
    ) {
    }

    public function POST_Create(): ActionResult {
        $rating = $this->getParam("rt");
        $comment = $this->getParam("ct");
        $pid = $this->getParam("pid");

        $result = $this->addRatingCommand->execute($pid, $rating,
$comment);

        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\AddRatingCommand
::Error_NotAuthenticated) {
                $errors[] = "Rating can only be added when user is
signed in.";
            }

            if($result & \Application\Commands\AddRatingCommand
::Error_CreateRatingFailed) {
                $errors[] = "Creating of rating failed.";
            }
```

```
            return $this->view(
                "productDetail", [
                "user" => $this->signedInUserQuery->execute(),
                "product" => $this->productDetailQuery->execute($pid),
                "errors" => $errors
            ]);
        }

        return $this->redirect("Products", "Detail", ["pid" => $pid]);
    }

    public function POST_Edit(): ActionResult {
        $rating = $this->getParam("rt");
        $comment = $this->getParam("ct");
        $rid = $this->getParam("rid");
        $pid = $this->getParam("pid");

        $result = $this->editRatingCommand->execute($rid, $pid,
$rating, $comment);
        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\EditRatingCommand
::Error_NotAuthenticated) {
                $errors[] = "Rating can only be update when user is
signed in.";
            }

            return $this->view(
                "productDetail", [
                     "user" => $this->signedInUserQuery->execute(),
                     "product" => $this->productDetailQuery->execute
($pid),
                     "errors" => $errors
                ]);
        }

        return $this->redirect("Products", "Detail", ["pid" => $pid]);
    }

    public function POST_Remove(): ActionResult {
        $rid = $this->getParam("rid");
        $pid = $this->getParam("pid");
```

```
        $result = $this->removeRatingCommand->execute($rid);
        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\EditRatingCommand
::Error_NotAuthenticated) {
                $errors[] = "Rating can only be removed when user is
signed in.";
            }

            return $this->view(
                "productDetail", [
                "user" => $this->signedInUserQuery->execute(),
                "product" => $this->productDetailQuery->execute($pid),
                "errors" => $errors
            ]);
        }

        return $this->redirect("Products", "Detail", ["pid" => $pid]);
    }
}
```

Listing 33. User.php

```
<?php

namespace Presentation\Controllers;

use Presentation\MVC\ViewResult;
use Presentation\MVC\ActionResult;
use Presentation\MVC\Controller;

class User extends Controller {
    public function __construct(
        private \Application\Commands\SignInCommand $signInCommand,
        private \Application\Commands\SignOutCommand $signOutCommand,
        private \Application\Queries\SignedInUserQuery
$signedInUserQuery,
        private \Application\Commands\RegisterCommand $registerCommand
    ) {

    }

    public function GET_LogIn() : ActionResult {
        // only show login view when there is no authenticated user
```

```
        $user = $this->signedInUserQuery->execute();
        if($user != null) {
            return $this->redirect("Home", "Index");
        }
        return $this->view("login", [
            "userName" => "",
            "user" => $user
        ]);
    }

    public function POST_LogIn(): ActionResult {
        $ok = $this->signInCommand->execute(
            $this->getParam("un"),
            $this->getParam("pwd"));

        if(!$ok) {
            return $this->view("login", [
                "user" => $this->signedInUserQuery->execute(),
                "userName" => $this->getParam("un"),
                "errors" => ["Invalid user name or password."]
            ]);
        }

        return $this->redirect("Home", "Index");
    }

    public function POST_LogOut(): ActionResult {
        $this->signOutCommand->execute();
        return $this->redirect("Home", "Index");
    }

    public function GET_Register() : ActionResult {
        // only show register view when there is no authenticated user
        $user = $this->signedInUserQuery->execute();
        if($user != null) {
            return $this->redirect("Home", "Index");
        }
        return $this->view("register", [
            "userName" => "",
            "password" => "",
            "user" => null
        ]);
    }
```

```
    public function POST_Register(): ActionResult {
        $userName = $this->getParam("un");
        $password = $this->getParam("pwd");
        $result = $this->registerCommand->execute(
            $userName,
            $password
        );

        // Check for errors
        if($result != 0) {
            $errors = [];
            if($result & \Application\Commands\RegisterCommand
::Error_UsernameAlreadyExists) {
                $errors[] = "User with username already exists.";
            }
            if($result & \Application\Commands\RegisterCommand
::Error_CreateUserFailed) {
                $errors[] = "User creation failed.";
            }
            if($result & \Application\Commands\RegisterCommand
::Error_InvalidUsername) {
                $errors[] = "Username is required.";
            }
            if($result & \Application\Commands\RegisterCommand
::Error_InvalidPassword) {
                $errors[] = "Password is required and needs a minimum
length of 4.";
            }

            if(sizeof($errors) == 0) {
                $errors[] = "Something went wrong.";
            }

            return $this->view("register", [
                "userName" => $userName,
                "password" => $password,
                "user" => null,
                "errors" => $errors
            ]);
        }

        //sign in if successful
        $ok = $this->signInCommand->execute($userName, $password);
```

```
        if(!$ok) {
            return $this->view("register", [
                "userName" => $userName,
                "password" => $password,
                "user" => null,
                "errors" => ["Sign in after registration was
unsuccessful!"]
            ]);
        }

//        return $this->view("register");
        return $this->redirect("Home", "Index");
    }
}
```

3.4. Views

3.4.1. Pages

Listing 34. error404.inc

```
<?php $render("partial/header", $data); ?>
<div class="center">
    <h1 class="text-center">404<br/>
        <small>Page could not be found!</small>
    </h1>
</div>
<?php $render("partial/footer", $data); ?>
```

Listing 35. editProduct.inc

```
<?php $render('partial/header', $data); ?>

    <?php $product = $data['product'] ?>

    <?php if ($product !== null) {?>
        <h1 class="mb-3">Edit product</h1>
        <?php $beginForm('Products', 'Edit', ["pid" => $product-
>getId()], method: 'post'); ?>
            <div class="mb-3">
                <label for="name" class="form-label">Name</label>
                <input class="form-control" id="name" name="nm"
value="<?php $htmlOut($product->getName()) ?>" autocomplete="off">
            </div>
            <div class="mb-3">
                <label for="producer" class="form-label">Brand</label>
                <input class="form-control" id="producer" name="pd"
value="<?php $htmlOut($product->getProducer()) ?>" autocomplete="off">
            </div>
            <button class="btn btn-outline-secondary">Update</button>
        <?php $endForm(); ?>
    <?php } ?>

<?php $render('partial/footer', $data); ?>
```

Listing 36. home.inc

```
<?php $render("partial/header", $data); ?>
    <h1 class="mb-3">Welcome</h1>
    <p>Welcome to the product rating app!</p>
<?php $render("partial/footer", $data); ?>
```

Listing 37. login.inc

```
<?php $render('partial/header', $data); ?>

    <h1 class="mb-3">Login</h1>

    <?php $beginForm('User', 'LogIn', method: 'post'); ?>
        <div class="mb-3">
            <label for="userName" class="form-label">User name</label>
            <input class="form-control" id="userName" name="un"
value="<?php $htmlOut($data['userName']); ?>" autocomplete="off">
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input type="password" class="form-control" id="password"
name="pwd" autocomplete="off">
        </div>
        <button class="btn btn-outline-primary">Log in</button>
    <?php $endForm(); ?>

<?php $render('partial/footer', $data); ?>
```

Listing 38. newProduct.inc

```
<?php $render('partial/header', $data); ?>

    <h1 class="mb-3">New product</h1>

    <?php $beginForm('Products', 'Create', method: 'post'); ?>
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input class="form-control" id="name" name="nm"
autocomplete="off">
        </div>
        <div class="mb-3">
            <label for="producer" class="form-label">Brand</label>
            <input class="form-control" id="producer" name="pd"
autocomplete="off">
        </div>
        <button class="btn btn-outline-secondary">Create</button>
    <?php $endForm(); ?>

<?php $render('partial/footer', $data); ?>
```

Listing 39. productDetail.inc

```
<?php $render("partial/header", $data); ?>

    <?php if($data['product'] !== null) { ?>
        <h1>Product</h1>

        <?php $render("partial/productInfo", $data); ?>
        <?php $render("partial/productRatings", $data); ?>

    <?php } ?>

<?php $render("partial/footer", $data); ?>
```

Listing 40. productList.inc

```
<?php $render("partial/header", $data); ?>

    <h1 class="mb-4">Products</h1>

    <?php $beginForm("Products", "Index"); ?>
    <div class="row justify-content-between">
        <div class="row col-auto mb-3">
            <div class="col-auto">
                <input autocomplete="off" class="form-control" name="f"
value="<?php $htmlOut($data["filter"]); ?>">
            </div>
            <div class="col-auto">
                <button class="btn btn-outline-success">Search</button>
            </div>
        </div>
        <?php if($data['user'] !== null) { ?>
        <div class="col-auto mb-3">
            <?php $link("Create new product", 'Products', 'Create',
cssClass: 'btn btn-outline-secondary'); ?>
        </div>
        <?php } ?>
        </div>
    <?php $endForm(); ?>

    <?php $render("partial/productTable", $data); ?>

<?php $render("partial/footer", $data); ?>
```

Listing 41. register.inc

```
<?php $render('partial/header', $data); ?>

    <h1 class="mb-3">Register</h1>

    <?php $beginForm('User', 'Register', method: 'post'); ?>
        <div class="mb-3">
            <label for="userName" class="form-label">Username</label>
            <input class="form-control" id="userName" name="un"
value="<?php $htmlOut($data['userName']); ?>" autocomplete="off">
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input class="form-control" id="password" name="pwd"
type="password" value="<?php $htmlOut($data['password']); ?>"
autocomplete="off">
            <div class="form-text">A password with a minimum length of
4 characters is required.</div>
        </div>
        <button class="btn btn-outline-primary">Register</button>
    <?php $endForm(); ?>

<?php $render('partial/footer', $data); ?>
```

3.4.2. Partial views

Listing 42. errors.inc

```
<div class="alert alert-danger">
  <p>
    <strong>Please correct the following and try again:</strong>
  </p>
  <ul>
    <?php foreach ($data as $errMsg): ?>
      <li><?php $htmlOut($errMsg); ?></li>
    <?php endforeach; ?>
  </ul>
</div>
```

Listing 43. footer.inc

```
</div>
    <footer class="bg-light fixed-bottom mt-5">
      <div class="container">
        <div class="text-muted py-2 d-flex justify-content-between">
            <span>
                Author:
                <a href="https://github.com/awenzelhuemer" class="link-
secondary">Andreas Wenzelhuemer</a></span>
            <?php $htmlOut(strftime('%c')); ?>
        </div>
      </div>
    </footer>
    <script src="js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Listing 44. header.inc

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.5.0/font/bootstrap-icons.css">
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <link href="css/styles.css" rel="stylesheet">
    <title>Product Rating App</title>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark sticky-
top">
        <div class="container">
            <?php $link("Product Rating App", "Home", "Index",
cssClass: "navbar-brand"); ?>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbar">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbar">
                <nav class="navbar-nav me-auto">
                    <?php $link("Home", "Home", "Index", cssClass:
"nav-link"); ?>
                    <?php $link("Products", "Products", "Index",
cssClass: "nav-link"); ?>
                </nav>
                <?php $render("partial/user", $data["user"])  ?>
            </div>
        </div>
    </nav>
    <div class="container my-5">
    <?php if(isset($data["errors"])) {
        $render("partial/errors", $data["errors"]);
    } ?>
```

Listing 45. productInfo.inc

```
<ul class="list-group">
    <?php $product = $data['product'] ?>
    <li class="list-group-item"><strong>Product name:</strong> <?php
$htmlOut($product->getName()) ?></li>
    <li class="list-group-item"><strong>Created from:</strong> <?php
$htmlOut($product->getUsername())  ?></li>
    <li class="list-group-item"><strong>Brand:</strong> <?php
$htmlOut($product->getProducer()) ?></li>
    <li class="list-group-item"
        data-bs-placement="top" title="<?php $htmlOut($product-
>getRating())?> out of 5">
        <strong>Average Rating:</strong>
        <?php for ($i = 0; $i < 5; $i++) {
            if($i < $product->getRating()) {?>
                <i class="bi bi-star-fill text-warning"></i>
            <?php } else {?>
                <i class="bi-star"></i>
            <?php } ?>
        <?php }?>
        / <small class="text-secondary"><?php $htmlOut($product-
>getRatingCount())?> ratings</small>
    </li>
</ul>
```

Listing 46. productRatings.inc

```
    <h2 class="mt-4 mb-3">Ratings</h2>
    <?php $product = $data['product'] ?>
    <div class="accordion" id="accordionExample">
        <?php if($data["user"] === null && sizeof($product-
>getRatings()) == 0) { ?>
            <div class="alert alert-info" role="alert">
                No ratings.
            </div>
        <?php } else if($data["user"] !== null) { ?>
        <div class="accordion-item">
        <h2 class="accordion-header">
            <button class="accordion-button" type="button" data-bs-
toggle="collapse" data-bs-target="#newRating" aria-expanded="true">
                New rating
            </button>
        </h2>
```

```
            <div id="newRating" class="accordion-collapse collapse
show" data-bs-parent="#newRating">
                <div class="accordion-body">
                    <?php $beginForm('Ratings', 'Create', ["pid" =>
$product->getId()], method: 'post'); ?>
                    <span class="me-4">Rating</span>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio"
name="rt" id="rating1" value="1">
                        <label class="form-check-label" for="rating1">
                        <i class="bi bi-star"></i>
                        </label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio"
name="rt" id="rating2" value="2">
                        <label class="form-check-label" for="rating2">
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                        </label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio"
name="rt" id="rating3" value="3">
                        <label class="form-check-label" for="rating3">
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                        </label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio"
name="rt" id="rating4" value="4">
                        <label class="form-check-label" for="rating4">
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                        </label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input class="form-check-input" type="radio"
name="rt" id="rating5" value="5" checked>
                        <label class="form-check-label" for="rating5">
```

```
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                            <i class="bi bi-star"></i>
                        </label>
                    </div>
                    <div class="my-3">
                        <label for="ct" class="form-label">
Comment</label>
                        <textarea class="form-control" id="ct"
name="ct" rows="3"></textarea>
                    </div>
                    <button class="btn btn-outline-primary">Add new
rating</button>
                    <?php $endForm(); ?>
                </div>
            </div>
        </div>
        <?php } ?>
        <?php foreach($product->getRatings() as $rating) { ?>
        <div class="accordion-item">
            <h2 class="accordion-header">
                <button class="accordion-button" type="button" data-bs-
toggle="collapse" data-bs-target="#rating<?php $htmlOut($rating-
>getId()) ?>" >
                    <div class="d-flex justify-content-between w-100">
                        <span>
                            <?php for ($i = 0; $i < 5; $i++) {
                                if($i < $rating->getRating()) {?>
                                    <i class="bi bi-star-fill text-
warning"></i>
                                <?php } else {?>
                                    <i class="bi-star"></i>
                                <?php } ?>
                            <?php }?>
                            <span class="ms-2">by <?php
$htmlOut($rating->getUser()->getUserName()) ?></span>
                        </span>
                        <span class="me-2">from <?php $htmlOut($rating-
>getCreatedDate()) ?></span>
                    </div>
                </button>
            </h2>
```

```
            <div id="rating<?php $htmlOut($rating->getId()) ?>"
class="accordion-collapse collapse <?php ($rating->getComment() !==
null) ? $htmlOut('show') : $htmlOut('')?>">
                <div class="accordion-body">
                    <?php if($data["user"] !== null && $data["user"]-
>getId() === $rating->getUser()->getId()) {?>
                        <?php $beginForm('Ratings', 'Remove', ["pid" =>
$product->getId(), "rid" => $rating->getId()], method: 'post'); ?>
                        <button class="btn btn-sm btn-outline-danger
float-end bi bi-trash-fill"
                                data-bs-toggle="tooltip" data-bs-
placement="top" title="Remove rating" ></button>
                        <?php $endForm(); ?>
                        <?php $beginForm('Ratings', 'Edit', ["pid" =>
$product->getId(), "rid" => $rating->getId()], method: 'post'); ?>
                            <span class="me-4">Rating</span>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input"
type="radio" name="rt" id="rating1Update" value="1" <?php ($rating-
>getRating()) === 1 ? $htmlOut('checked') : $htmlOut('') ?>>
                                <label class="form-check-label"
for="rating1Update">
                                    <i class="bi bi-star"></i>
                                </label>
                            </div>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input"
type="radio" name="rt" id="rating2Update" value="2" <?php ($rating-
>getRating()) === 2 ? $htmlOut('checked') : $htmlOut('') ?>>
                                <label class="form-check-label"
for="rating2Update">
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
                                </label>
                            </div>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input"
type="radio" name="rt" id="rating3Update" value="3" <?php ($rating-
>getRating()) === 3 ? $htmlOut('checked') : $htmlOut('') ?>>
                                <label class="form-check-label"
for="rating3Update">
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
```

```
                                </label>
                            </div>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input"
type="radio" name="rt" id="rating4Update" value="4" <?php ($rating-
>getRating()) === 4 ? $htmlOut('checked') : $htmlOut('') ?>>
                                <label class="form-check-label"
for="rating4Update">
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
                                    <i class="bi bi-star"></i>
                                </label>
                            </div>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input"
type="radio" name="rt" id="rating5Update" value="5" <?php ($rating-
>getRating()) === 5 ? $htmlOut('checked') : $htmlOut('') ?>>
                                <label class="form-check-label"
for="rating5Update">
                                    <i class="bi bi-star text-
secondary"></i>
                                    <i class="bi bi-star text-
secondary"></i>
                                    <i class="bi bi-star text-
secondary"></i>
                                    <i class="bi bi-star text-
secondary"></i>
                                    <i class="bi bi-star text-
secondary"></i>
                                </label>
                            </div>
                            <div class="my-3">
                                <label for="ct" class="form-label"
>Comment</label>
                                <textarea class="form-control" id="ct"
name="ct" rows="3"><?php $htmlOut($rating->getComment()) ?></textarea>
                            </div>
                            <button class="btn btn-outline-secondary"
>Update existing rating</button>
                        <?php $endForm(); ?>
                    <?php } else { ?>

                    <?php if($rating->getComment() !== null) {
```

```
                        $htmlOut($rating->getComment());
                    } else { ?>
                        <div class="alert alert-secondary" role=
"alert">
                            This rating includes no comment!
                        </div>
                    <?php } ?>
                <?php } ?>
                </div>
            </div>
        </div>
        <?php } ?>
    </div>
```

Listing 47. productTable.inc

```
<?php if($data['products'] !== null) { ?>
    <?php if(sizeof($data['products']) > 0) { ?>
        <div class="table-responsive">
            <table class="table table-sm table-striped table-hover">
                <thead>
                <tr>
                    <th>Name</th>
                    <th>Brand</th>
                    <th>Rating</th>
                    <th>Created user</th>
                    <th></th>
                </tr>
                </thead>
                <tbody>
                <?php foreach($data['products'] as $product) { ?>
                    <tr>
                        <td>
                            <?php $link($product->getName(),
'Products', 'Detail', ['pid' => $product->getId()], cssClass: 'link-
primary'); ?>
                        </td>
                        <td><?php $htmlOut($product->getProducer());
?></td>
                        <td data-bs-placement="top" title="<?php
$htmlOut($product->getRating())?> out of 5">
                            <?php for ($i = 0; $i < 5; $i++) {
                                if($i < $product->getRating()) {?>
                                    <i class="bi bi-star-fill text-
```

```
warning"></i>
                                <?php } else {?>
                                    <i class="bi-star"></i>
                                <?php }?>

                            <?php }?>/
                            <small class="text-secondary"><?php
$htmlOut($product->getRatingCount())?> ratings</small></td>
                        <td><?php $htmlOut($product->getUserName());
?></td>
                        <td>
                            <?php if($data['user'] !== null &&
$product->getUserId() === $data['user']->getId()) {?>
                                <?php $link("", 'Products', 'Edit',
['pid' => $product->getId()], cssClass: 'btn btn-sm btn-outline-
secondary bi bi-pencil-fill'); ?>
                            <?php } ?>
                        </td>
                    </tr>
                <?php } ?>
                </tbody>
            </table>
        </div>
    <?php } else { ?>
        <div class="alert alert-info" role="alert">
            No products founds.
        </div>
    <?php } ?>
<?php } ?>
```

3.5. Styles

Listing 48. styles.css

```
.center {
    margin: 0;
    position: absolute;
    top: 50%;
    left: 50%;
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
}
```

4. Datenbank

4.1. Datenbankmodell

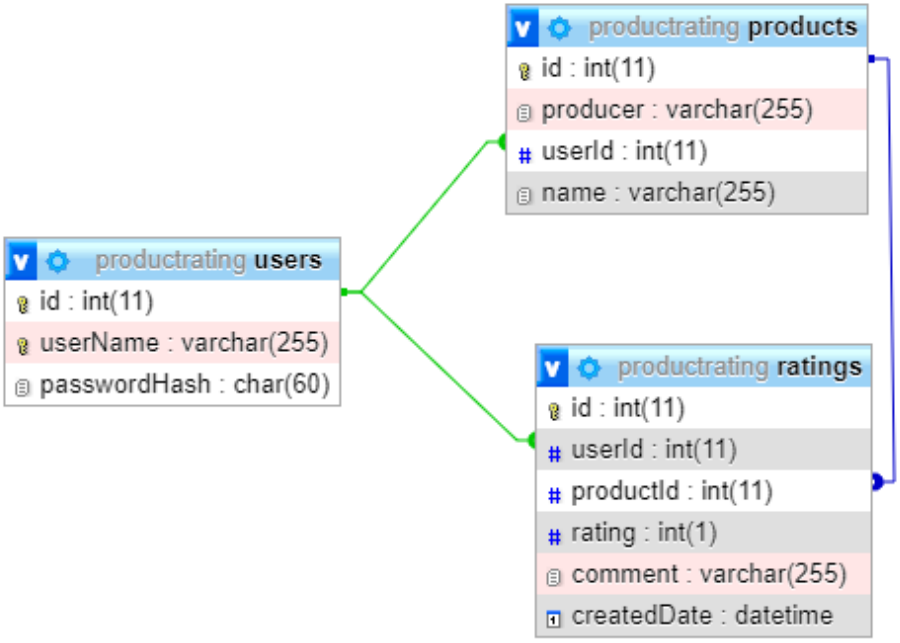


Figure 1. Modell

4.2. Scripts

Listing 49. CREATE_DB.sql

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

DROP DATABASE IF EXISTS productRating;

CREATE DATABASE IF NOT EXISTS `productRating` DEFAULT CHARACTER SET
latin1 COLLATE latin1_general_ci;
USE `productRating`;

-- table creation
CREATE TABLE `users` (
                        `id` int(11) NOT NULL,
                        `userName` varchar(255) NOT NULL,
                        `passwordHash` char(60) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `products` (
                           `id` int(11) NOT NULL,
                           `producer` varchar(255) NOT NULL,
                           `userId` int(11) NOT NULL,
                           `name` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `ratings` (
                          `id` int(11) NOT NULL,
                          `userId` int(11) NOT NULL,
                          `productId` int(11) NOT NULL,
                          `rating` int(1) NOT NULL,
                          `comment` varchar(255) NULL,
                          `createdDate` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- primary keys
ALTER TABLE `users`
    ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `userName` (`userName`);

ALTER TABLE `products`
```

```
    ADD PRIMARY KEY (`id`),
  ADD KEY `userId` (`userId`);

ALTER TABLE `ratings`
    ADD PRIMARY KEY (`id`),
  ADD KEY `userId` (`userId`),
  ADD KEY `productId` (`productId`);;

-- auto incrementing ids
ALTER TABLE `users`
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
ALTER TABLE `products`
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;
ALTER TABLE `ratings`
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;

ALTER TABLE `products`
    ADD CONSTRAINT `products_ibfk_1` FOREIGN KEY (`userId`) REFERENCES
`users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE `ratings`
    ADD CONSTRAINT `ratings_ibfk_1` FOREIGN KEY (`userId`) REFERENCES
`users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
    ADD CONSTRAINT `ratings_ibfk_2` FOREIGN KEY (`productId`)
REFERENCES `products` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;
```

<< .DB_SEEDING.sql

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

-- seeding
INSERT INTO `users` (`id`, `userName`, `passwordHash`) VALUES
(1, 'scr4',
'$2y$10$0dhe3ngxlmzgZrX6MpSHkeoDQ.dOaceVTomUq/nQXV0vSkFojq.VG');

INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(1, 'Bosch', 1, 'Schlagbohrmaschine GSB 13 RE');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(2, 'Bosch', 1, 'Schlagbohrmaschine EasyImpact 550');
```

```
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(3, 'Tilswall', 1, 'Schlagbohrmaschine 3000');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(4, 'Bosch', 1, 'Bosch Schlagbohrmaschine UniversalImpact');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(5, 'Einhell', 1, 'Schlagbohrmaschine TC-ID 650E');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(6, 'Makita', 1, 'Schlagbohrmaschine HP1631');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(7, 'Makita', 1, 'Kombihammer SDS-Plus');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(8, 'Metabo', 1, 'Schlagbohrmaschine SBEV 1000-2');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(9, 'Hi-Spec', 1, 'Akkubohrmaschinenset');
INSERT INTO `products` (`id`, `producer`, `userId`, `name`) VALUES
(10, 'Bosch', 1, 'Schlagbohrmaschine GSP 19-2 RE');

INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(1, 1, 1, 5, 'Works perfectly!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(2, 1, 2, 4, 'Works good!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(3, 1, 3, 3, 'Works perfectly, some parts are missing!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(4, 1, 4, 4, 'Works fine!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(5, 1, 5, 5, 'Works perfectly!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(6, 1, 6, 5, 'Works perfectly!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(7, 1, 7, 2, 'Not working!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(8, 1, 7, 2, 'Not good at all!', NOW());
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(9, 1, 7, 1, NULL, NOW());
```

```
INSERT INTO `ratings` (`id`, `userId`, `productId`, `rating`,
`comment`, `createdDate`) VALUES
(10, 1, 8, 1, 'Total crap!', NOW());
COMMIT;
```