|  |
| --- |
|  |
| William Wong |
| Bachelor Toegepaste Informatica  Afstudeerrichting? |
|  |
|  |
| Software Testing  Frank Serneels |
|  |
| Academiejaar 2021-2022 |

|  |  |
| --- | --- |
| Opdracht 3 | |
|  |

# Opdracht 3:

Voor de derde opdracht heb ik besloten om een bestaande project, die ik had gemaakt voor een vorige vak, te verbeteren via software testing. Ik zal het project dus aanpassen met behulp van software testing. Ik zal het project ook op github zetten. Ik zal zowel manuele als automated testen uitvoeren voor deze opdracht. Het omvat alles wat ik heb gezien in de Udemy cursus.

# Databank:

Zoals gewoonlijk zal ik eerst een databank aanmaken. Dit doe ik in PHPMyAdmin. Ik zal vijf tabellen aanmaken in mijn databank, nl: customers, products, categories, ShoppingCarts en ShoppingCartItems.

Tabel customers:

Graphical user interface, application, chat or text message

Description automatically generated

Deze tabel heeft als primary key de login. Bovendien mag geen enkele veld leeg zijn.

Tabel categories:



Deze tabel heeft als primary key: category.

Tabel products:

Graphical user interface, text, application, chat or text message

Description automatically generated

Deze tabel heeft als primary key een ID. Hiervoor geldt auto increment. Bovendien mag geen enkele veld leeg zijn.

Tabel shopping cart:

Graphical user interface, chat or text message

Description automatically generated with medium confidence

Deze tabel heeft een ID voor de primary key. Alle andere velden mogen niet leeg zijn.

Tabel shopping cart items:

Graphical user interface, application

Description automatically generated

In deze tabel mag geen enkele veld leeg zijn. De combinatie van product id en shopping cart id vormen een samengestelde primary key.

Databank structuur:

Graphical user interface, application

Description automatically generated

Products heeft een foreign key, nl: de category\_name verwijst naar het veld category in de tabel categories. De tabel shopping cart items heeft ook een foreign key, de shopping cart id verwijst naar de Id van de tabel shopping cart. Deze tabel heeft nog een foreign key, de product id verwijst naar een Id in de tabel categories. De tabel shopping cart heeft dan weer een foreign key, van customer login naar het veld login in de tabel customers.

# Project:

Diagram

Description automatically generated

Dit is het project die ik heb gemaakt voor een andere vak. Het bestaat uit een aantal pagina’s, waar ik een aantal testen voor zal uitvoeren. Ik zal zowel de front als back end testen. De backend zal ik testen via postman. Op basis van die testen zal ik de front aanpassen, zodat de front een aantal problemen zal opvangen voordat iets in een databank wordt gestoken. Ik moet ook nog API’s maken voor dit project, die ik ook eerst zal testen in Postman voordat ik gebruik zal maken van die API’s in mijn project. Ik zal gebruik maken van test scenario’s.

Extra informatie over de website: De gebruiker kan producten toevoegen in de winkelwagen en zijn winkelwagen ook bekijken. Bovendien kan de gebruiker de currency veranderen zodat de prijzen aangepast worden aan de dagelijkse exchange rate (gebruik van API voor dagelijkse wisselkoersen). De gebruiker moet kunnen inloggen en kan ook enkel iets in zijn/haar winkelmand toevoegen als hij/zij ingelogd is. Een persoon moet zich kunnen registreren. De gebruiker moet kunnen filteren op basis van categorie en prijs. De gebruiker kan ook een winkelwagen afrekenen.

# Test scenario’s:

## Usability test scenario’s:

### Login form:

controleren of er een veld voorzien is voor error messages en of deze goed zichtbaar en duidelijk is.

Controleren of de gebruiker kan zien voor welke gegeven een field dient => passwoord en login.

Controleren of er genoeg ruimte is tussen de labels en input, zodat de form overzichtelijk is.

Controleren of de form een button heeft om de form kan submitten.

### Registratie form:

controleren of er een veld voorzien is voor error messages en of deze goed zichtbaar en duidelijk is.

Controleren of de gebruiker kan zien voor welke gegeven een field dient => gebruikersnaam, voornaam, naam etc.

Controleren of er genoeg ruimte is tussen de labels en input, zodat de form overzichtelijk is.

Controleren of de gebruiker een bericht krijgt als de registratie goed verlopen is.

Controleren of de form een button heeft om de form kan submitten.

### Website algemeen:

Controleren of er op elke pagina een link is om naar de home page te gaan.

Controleren of er in elke pagina de mogelijkheid is om te navigeren naar elke andere pagina.

Controleren of elke link naar een pagina werkt.

Controleren of elke foto juist wordt weergegeven.

Controleren of de webpagina’s responsive zijn.

Controleren of de gebruiker een bericht krijgt wanneer hij/zij een actie afrondt.

Controleren of de webpagina’s navigeerbaar zijn via het toetsenbord.

Controleren of de webpagina’s een duidelijke titel hebben.

Controleren of er bij elke product een optie is om deze toe te voegen aan het winkelmand.

Controleren of de gebruiker kan navigeren naar zijn/haar winkelmand.

## Functional test scenario’s:

### Forms:

Testen of dat alle verplichte velden worden gevalideerd.

Testen of er geen error message wordt getoond voor optionele velden.

Testen of numerieke velden geen letters bevat.

Testen of elke veld voldoet aan de maximale lengte (opgegeven in de databank).

Testen of elke veld de juiste characters bevat => bv bij email moet een @ teken bevatten.

Testen of elke button de juiste operatie uitvoert.

Testen of de gebruiker naar de error pagina begeleid wordt, wanneer een functionaliteit faalt.

Controleren of de gebruiker naar de juiste pagina gaat, wanneer hij/zij juiste credentials meegeeft bij login.

Controleren of de gebruiker de juiste boodschap krijgt, wanneer hij/zij foute credentials meegeeft bij login.

### Algemene webpagina’s:

Controleren of de prijzen niet onder nul kan gaan.

Controleren of de producten juist worden gefilterd => bv bij prijzen mogen producten die duurder zijn niet getoond worden.

Controleren of de exchange rate API werkt.

Testen of de juiste currency format wordt weergegeven in de gekozen currency.

## Compatibility test scenario’s:

Testen of de website het gewenste resultaat heeft in verschillende browsers.

Testen of de afbeeldingen van de website werken in verschillende browsers.

Testen of de java script code van de website werkt in verschillende browsers.

## Database test scenario’s:

Controleren welke kolommen nullable mogen zijn en welke niet.

Controleren of de primary key en foreign key juist zijn.

De data types testen.

Controleren of de juiste gegevens uit de databank gehaald worden.

Controleren of de data juist wordt opgeslagen.

De data controleren na insert operaties.

Controleren of de lengte van gegevens overeen komen met de ingevoerde gegevens in de front end.

Controleren of de gegevens in de front end hetzelfde zijn als de gegevens van de databank.

Controleren of non valid data niet wordt opgeslagen.

## Security test scenario’s:

Controleren of de gebruiker niet kan inloggen met oude wachtwoorden.

Controleren of er geen gevoelige informatie wordt weergegeven.

Controleren of de gebruiker geen functionaliteiten kan uitvoeren wanneer hij/zij uitgelogd is.

Controleren of de gebruiker kan inloggen met juiste gegevens.

Controleren of de gebruiker niet kan inloggen met foute credentials.

Bovenstaande test scenario’s zal ik testen aan de hand van test cases, waarbij ik voor een aantal zaken met postman zal werken maar ook in de front end zal ik testen. Bovendien zal ik door back end testen met postman kunnen bepalen wat er anders moet in de front end of back end.

# API’s:

Ik zal eerst bepalen welke API’s ik zelf zal moeten maken en testen. Ik heb namelijk verschillende API’s nodig:

Een API om de gebruiker te vinden op basis van zijn/haar credentials. => zowel manueel als automated testen.

Een API om de gegevens van een gebruiker op te vragen. => zowel manueel als automated testen.

Een API om een gebruiker toe te voegen. => zowel manueel als automated testen.

Een API om de producten op te vragen.

Een API om de producten te filteren op basis van categorie.

Een API om de producten te filteren op basis van de prijs.

Een externe API om de exchange rates op te vragen en om te zetten naar de gevraagde currency => fixer.io API.

Een API om een winkelmand toe te voegen. => zowel manueel als automated testen.

# Besluit:

Dit is mijn analyse over hoe ik te werk zal gaan voor het derde opdracht. Volgende week zal ik de lijst afgaan en de testen uitvoeren, zowel manuele als automated testen. Ik zal voor de backend testen gebruik maken van postman. Ik zal gebruik maken van test cases die ik nog zal schrijven.

# Back end:

Ik zal eerst beginnen met een back end te maken en deze te testen. Ik heb de database al aangemaakt, dus ik kan onmiddellijk beginnen met de API’s. Ik heb de databank ook al geplaatst op een server en zodra de API’s klaar zijn zal ik deze op de server zetten. De API’s moeten ook getest worden en aangepast worden als dat nodig is.

De API’s opgesomd:

Een API om de gebruiker te vinden op basis van zijn/haar credentials. => zowel manueel als automated testen.

Een API om de gegevens van een gebruiker op te vragen.

Een API om een gebruiker toe te voegen. => zowel manueel als automated testen.

Een API om de producten op te vragen.

Een API om de producten te filteren op basis van categorie.

Een API om de producten te filteren op basis van de prijs.

Een externe API om de exchange rates op te vragen en om te zetten naar de gevraagde currency => fixer.io API.

Een API om een winkelmand toe te voegen. => zowel manueel als automated testen.

# API’s + tests:

## Customer login API:

Deze API dient ervoor om na te gaan of de login credentials van een gebruiker kloppen. Deze API dient er dus voor om een gebruiker terug te geven op basis van login en passwoord.

<?php

header('Access-Control-Allow-Origin: \*');

header('Access-Control-Allow-Methods: POST');

header('Access-Control-Max-Age: 1000');

header('Access-Control-Allow-Headers: Content-Type, Authorization, X-Requested-With');

$conn = new mysqli('ID328986\_webShop.db.webhosting.be', 'ID328986\_webShop', 'azerty123', 'ID328986\_webShop');

$stmt = $conn->prepare("select \* FROM Customers WHERE Login like ? and Password like ?");

if(!$stmt->bind\_param("ss", $postvars['Login'], $postvars['Password'])){

    $response['code'] = 7;

    $response['status'] = 200;

    $response['data'] = $conn->error;

    deliver\_response($response);

}

if (!$stmt->execute()) {

    $response['code'] = 7;

    $response['status'] = $api\_response\_code[$response['code']]['HTTP Response'];

    $response['data'] = $conn->error;

    deliver\_response($response);

}

$result = $stmt->get\_result();

if (!$result) {

    $response['code'] = 7;

    $response['status'] = $api\_response\_code[$response['code']]['HTTP Response'];

    $response['data'] = $conn->error;

    deliver\_response($response);

}

$response['data'] = getJsonObjFromResult($result);

$result->free();

$conn->close();

deliver\_JSONresponse($response);

?>

Nadat ik deze API op de server heb geplaatst (link: williamwong.be/webshop/CustomerGetAPI.php), zal ik deze nu testen via postman.

Graphical user interface, text, application, email

Description automatically generated

Ik krijg echter een error met status code 500.

In progress: ik ben de fout nog aan het zoeken. Van zodra de API werkt, zal ik meer testen uitvoeren

# Get customers API:

<?php

header('Content-type: application/json');

$conn = new mysqli('ID328986\_webShop.db.webhosting.be', 'ID328986\_webShop', 'azerty123', 'ID328986\_webShop');

$data = array();

$sql = "SELECT \* FROM Customers";

$result = $conn

->query($sql);

while($row = $result

->fetch\_assoc()) {

    $data[] = $row;

}

echo json\_encode($data);

exit(0);

?>

Deze API vraagt de lijst op van alle klanten. Ik heb ook een aantal testen geschreven om een aantal zaken na te gaan: Ik zal testen of de status code 200 is en dus of de request gelukt is. Ik zal ook testen of de response time onder de 500 ms is. Ik zal ook de headers checken van de API en ook of alle records worden weergegeven. Bovendien zal ik ook nagaan of alle properties worden meegegeven.

pm.**test**("Status code is 200", () **=>** {

  pm.expect(pm.response.code).to.eql(200);

});

pm.**test**("Response time is less than 500ms", () **=>** {

  pm.expect(pm.response.responseTime).to.be.below(500);

});

pm.**test**("Content-Type header is application/json", () **=>** {

  pm.expect(pm.response.headers.**get**('Content-Type')).to.eql('application/json');

});

const jsonData **=** pm.response.json();

pm.**test**("Data length is correct", () **=>** {

pm.expect(jsonData.length).to.eql(5);

});

const responseJson **=** pm.response.json();

**for**(let row **of** responseJson) {

    pm.**test**("The response has all properties",() **=>** {

    pm.expect(row.Login).to.be.a('string');

    pm.expect(row.FirstName).to.be.a('string');

    pm.expect(row.LastName).to.be.a('string');

    pm.expect(row.Email).to.be.a('string');

    pm.expect(row.Password).to.be.a('string');

    pm.expect(row.Address).to.be.a('string');

    pm.expect(row.Telephone).to.be.a('string');

    });

}

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

In de post API’s zal ik ook gebruik maken van negatieve tests.

# Add customer API:

# Get products API:

# Filter products API by category:

# Filter products API by price:

# Add shopping cart API: