Kwaliteitsplan web shop

Inhoud

[Inleiding 3](#_Toc184200073)

[Kwaliteitsdoelstelling 3](#_Toc184200074)

[Rollen en verantwoordelijkheden 3](#_Toc184200075)

[Kwaliteitsnormen 4](#_Toc184200076)

[Kwaliteitscontroleprocessen 6](#_Toc184200077)

[Teststrategie 6](#_Toc184200078)

[Risicobeheer 6](#_Toc184200079)

[Kwaliteitsborging 6](#_Toc184200080)

[Conclusie 6](#_Toc184200081)

# Inleiding

In dit project is het mijn doel om een web shop te maken die op klant en manager levels werkt. Hierdoor zal ik een API een web applicatie en een desktop applicatie moeten maken.

Omdat er drie projecten gemaakt worden moet ik ervoor zorgen dat ze allemaal werken en handig te gebruiken zijn voor de gebruiker. Om zeker tew eten dat dit het geval zal zijn stel ik dit kwaliteitsplan op.

# Kwaliteitsdoelstelling

* Het systeem moet niet onverwacht crashen.
* Gebruikers moeten een fijne ervaring hebben bij interactie met de webshop.
* Alle code moet voldoen aan OO en SOLID principles.
* Er moet een testdekking zijn van minimal 80% op kritieke functies.

# Rollen en verantwoordelijkheden

* **Martijn Verhappen**: Als ontwikkelaar en projectmanager ben ik verantwoordelijk voor het plannen, uitvoeren en evalueren van de kwaliteitscontrole.
* **Docent/begeleider**: Controleert de eindkwaliteit en geeft feedback.
* **Stakeholders** “klanten” en “managers”: Testen of de webshop of manager applicatie aan hun eisen voldoet.

# Kwaliteitsnormen

* Ontwikkelstandaarden:
* **C# Codestijlconventies**

**Naamgevingsconventies:**

* Klassen, methoden en variabelen moeten PascalCase gebruiken (bijv. MyClass, DoSomething(), myVariable).
* Privévelden moeten camelCase gebruiken (bijv. \_myPrivateField).
* Constanten moeten UPPER\_SNAKE\_CASE gebruiken (bijv. MAX\_VALUE).
* Interfaces moeten beginnen met het voorvoegsel "I" (bijv. IRepository).

**Haakjes en inspringing:**

* Gebruik altijd accolades {} voor alle controleconstructies, zelfs bij enkelvoudige regels.
* Spring code in met 4 spaties per niveau.

**Commentaar en documentatie:**

* Gebruik XML-opmerkingen voor publieke types en leden.
* Voorzie duidelijke en beknopte opmerkingen om het doel en de functionaliteit van de code uit te leggen.

**Foutafhandeling:**

* Gebruik try-catch-blokken om uitzonderingen af te handelen.
* Gooi specifieke uitzonderingen in plaats van de generieke Exception-klasse te gebruiken.

**Opmaak:**

* Gebruik consistente spaties en regelafbrekingen voor leesbaarheid.
* Lijn methodparameters en argumenten verticaal uit waar mogelijk.
* **React Codestijlconventies**

**Naamgevingsconventies:**

* Componenten moeten PascalCase gebruiken (bijv. MyComponent).
* Props moeten camelCase gebruiken (bijv. myProp).
* State-variabelen moeten camelCase gebruiken (bijv. myState).

**Bestandsstructuur:**

* Plaats elke component in een eigen bestand, genoemd naar de component.
* Groepeer gerelateerde componenten in dezelfde map.

**Functionele componenten en hooks:**

* Geef de voorkeur aan functionele componenten boven klassecomponenten.
* Gebruik hooks (bijv. useState, useEffect) om state en side-effects te beheren.

**JSX-opmaak:**

* Gebruik consistente inspringing en regelafbrekingen voor leesbaarheid.
* Plaats openings- en sluitingstags op nieuwe regels voor elementen met meerdere regels.
* Gebruik zelfsluitende tags voor lege elementen (bijv. <img />, <br />).

**Prop spreading:**

* Vermijd het direct doorgeven van props aan het componentelement.
* Gebruik objectdestructurering om alleen de noodzakelijke props door te geven.

**Foutafhandeling:**

* Gebruik try-catch-blokken of ErrorBoundary-componenten om runtimefouten af te handelen.
* Voorzie duidelijke en informatieve foutmeldingen voor de gebruiker.

**Styling:**

* Gebruik CSS-in-JS-oplossingen (bijv. styled-components, Emotion) of modulaire CSS voor stijlen op componentniveau.
* Vermijd globale CSS-stijlen om encapsulatie te behouden.
* **Databasebeheer**: Er zal consistente data-integriteit in MySQL plaatsvinden.
* **Gebruiksvriendelijkheid**: De Web Content Accessibility Guidelines (WCAG) zal gevolgd worden.
* **Beveiliging**: Er zal gebruik gemaakt worden van encrypptie (TLS) en multi-factor authenticatie.

# Kwaliteitscontroleprocessen

* **Code-review**: Er zal gebruik gemaakt worden van code analysis en feedback van de docent om de kwaliteit van de code te garanderen.
* **Geautomatiseerde tests**: Unit- en integratietests worden gebruikt voor de API en front-ends.
* **Handmatige tests**: Door middle van usability tests zal de GUI op fouten en gebruiksvriendelijkheid getest worden.
* **Bug-tracking**: Er zal gebruik gemaakt worden van github om bugs te beheren.

# Teststrategie

* **Unit-tests**: Controleert de individuele functies van mijn project.
* **Integratietests**: Zorgt dat de communicatie tussen de front-end en de API vlekkeloos verloopt.
* **End-to-end tests**: Test een volledige gebruikersflow, zal van het opzoeken van een product tot het toevoegen aan een winkelmand testen.

# Risicobeheer

**Risico**: Gebruikerservaring kan lijden onder bugs.

* **Maatregel**: Regelmatige handmatige tests en bugfixes.

**Risico**:

# Kwaliteitsborging

* Regelmatig evalueren van de voortgang met de projectbegeleider.
* Rapportage van testresultaten en kwaliteitscontrole in documentatie.

# Conclusie

Als alles dat in dit document staat gevolgd wordt zal het resultaat van mijn project een functionele en gebruikersvriendelijke webshop en manager applicatie zijn.