GitHub repo: <https://github.com/alisia142/itvb23ows-starter-code>

Ontwikkelstraten Week 1

Hier ziet u dat ik een fork heb gecreëerd op basis van de gegeven starter code. Ook heb ik het de issues aangemaakt op basis van de gegeven bugs en features. Ik heb er duidelijke namen aan gegeven, en de details gezet in de beschrijving van de desbetreffende bug of feature. Zo kun je per bug of feature zien waar het aan moet voldoen voordat de issue kan worden opgelost.

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

A screenshot of a computer

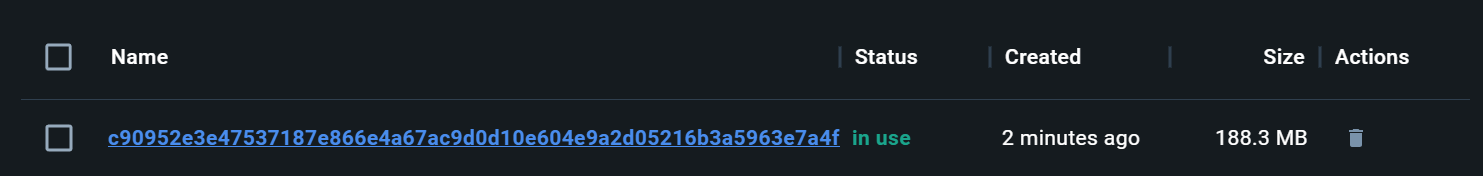
Description automatically generated

Ontwikkelstraten Week 2

Hier ziet u de dockerfiles en de configuratie die nodig is om de bestaande code te kunnen draaien. Als eerste ziet u de containers, daarna de images die nodig zijn voor de build en als laatste de volume die wordt gebruikt.

A screenshot of a computer

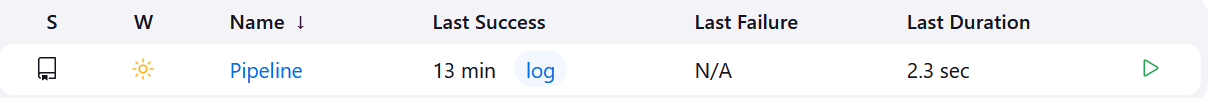
Description automatically generatedA screenshot of a computer

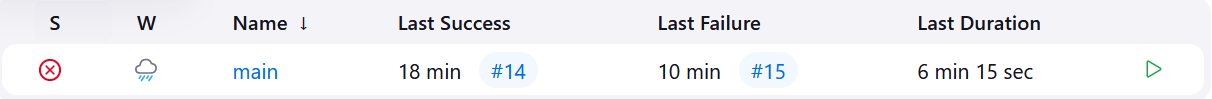
Description automatically generatedA screenshot of a computer

Description automatically generated

Bestanden: dockerfile, hive.sql, docker-compose.yaml, database.php

Ontwikkelstraten Week 3

Had wat problemen met Jenkins, heb daar 3 of 4 dagen aangewerkt. Inmiddels opgelost door alle volumes, containers en images weg te gooien en alles opnieuw aan te maken. 

A white rectangular object with black lines

Description automatically generated

Uitleg en verantwoording gekozen branchingstrategie:

Voor de branchingstrategie is er gekozen voor een Trunk-Based-Development strategie. Hierbij worden alle wijzigingen gepushed in main en zijn er geen langdurige feature branches. De reden hiervoor is dat ik als enige auteur aan dit project werk, en geen merge-conflicten hoef op te lossen met andere auteurs. Omdat het een relatief kleinschalig project is, zal het hebben van verschillende branches minder impact hebben op overzichtelijkheid dan normaal gesproken.   
Stel ik zou dit project verder ontwikkelen met een andere auteur, dan zullen er dan wel branches worden aangemaakt zodat er overzicht blijft in de ‘main’ branch.

Bestand: Jenkinsfile

Ontwikkelstraten Week 4

In week 4 is SonarQube gekoppeld aan de Jenkins pipeline. Bij de issues vindt u een globale beschrijving van de verschillende issues en hoe het opgelost zou kunnen worden.

Bestand: Jenkinsfile

A computer screen with text on it

Description automatically generated

A screenshot of a computer

Description automatically generated

Na een aantal bestanden te hebben aangepast:  
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

In SonarQube zijn de verschillende issues verdeeld onder meerdere kopjes.

Software Quality:

* Maintainability: beschrijft hoe behoudbaar deze code is, en hoe “schoon” je code is.   
  - Rood betekent: High Severity, wat betekent dat deze snel en correct moet worden opgelost voor behoud van je code. Dit heeft dus ook hoge impact op de kwaliteit van je code.  
  - Geel betekent: Medium Severity, wat betekent dat het beter is dat je dit oplost, maar het werkt ook redelijk zonder dat je deze oplost. Dit heeft redelijke impact op de kwaliteit van je code.  
  - Blauw betekent: Low Severity, wat betekent dat het prima is als deze code zo blijft, maar er wordt een suggestie gegeven dat het ook anders kan. Dit heeft lage impact op de kwaliteit van je code.  
  Maintainability zorgt ervoor dat ontwikkelaars prioritering kunnen toepassen om basis van de impact op de codekwaliteit. A screenshot of a computer

  Description automatically generated
* Security: heeft te maken met de beveiliging van je code. Als je bijvoorbeeld een database connectie gebruikt, kun je de onderstaande error krijgen. Dit betekent dat t beter zou zijn als je een wachtwoord meegeeft voor je database connectie zodat er geen manipulatie kan plaatsvinden.

A screenshot of a computer

Description automatically generated

* Reliability: heeft te maken met hoe betrouwbaar je code is, en dus ook hoe kwalitatief betrouwbaar je code is.

Clean Code Attribute:

* Consistency: deze treedt op als je verschil qua haakjes of aanhalingstekens hebt in je code. Bijvoorbeeld: ‘dit is een string’, “dit is ook een string”. SonarQube geeft dan aan dat je beter een van de twee varianten kunt gebruiken door je code heen, zodat het consistent is. Maar hier hoort ook het type casing bij van je code (uppercase of camelcase etc.) en hoe je de taal (php o.i.d) toepast.
* Intentionality: deze treedt op als je code verbetert kan worden op het gebied van logica en een “schone” structuur. Bijvoorbeeld: je code is slecht leesbaar door te veel tekens op 1 lijn, SonarQube geeft dan aan dat je het beter kan opsplitsen in 2 lijnen.
* Adaptability: deze treedt op als je code niet modulair of testbaar is. Bijvoorbeeld als je assertions mist in je testcase.

A screenshot of a computer

Description automatically generated

* Responsibility: deze treedt op in het geval van ethische kwesties. Bijvoorbeeld bij de database connectie, treedt deze ook op. Omdat in het geval van een database zonder password protection, is er wel een zekere verantwoordelijkheid voor de ontwikkelaar om te zorgen dat zijn ontwikkelde project geen privé gegevens schaad.

Ontwikkelstraten week 5 en week 6 vindt u in de repository zelf.

Ontwikkelstraten week 7

In week 7 moest er een Dockerfile worden geschreven voor de mogelijkheid om tegen een AI te spelen. Hieronder ziet u het bewijs van de dockerfile, de toevoeging aan docker-compose.yaml en de draaiende docker container.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated