

Release Plan Groep 5

Teun Willems, Jules Kreutzer

1 maart 2018

Inhoudsopgave

1	Inleiding	2
1.1	Verklarende Woordenlijst	3
2	Version Control	4
2.1	Branches	4
2.1.1	Master	4
2.1.2	Hotfix	5
2.1.3	Testing	5
2.1.4	Development	5
2.1.5	Features	5
2.1.6	Bugfix/Patches	6
3	Code Style	7
3.1	Release	8
4	Continuous Integration	9
4.1	Pull requests & mergen	9
5	Versioning	10
5.1	Major	10
5.2	Minor	10
5.3	Patch	10
5.4	Overige Informatie	10
6	Continuous Delivery	12

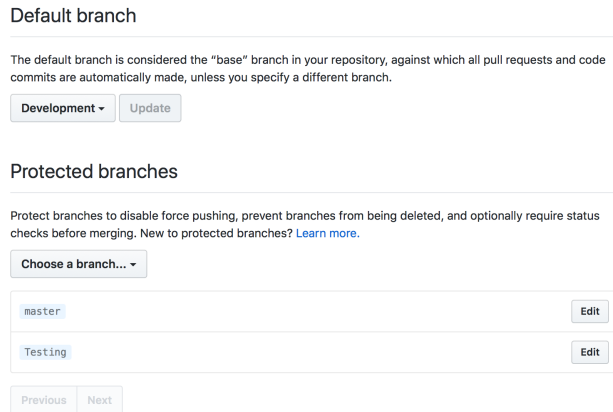
1 Inleiding

Dit release document beschrijft

1.1 Verklarende Woordenlijst

2 Version Control

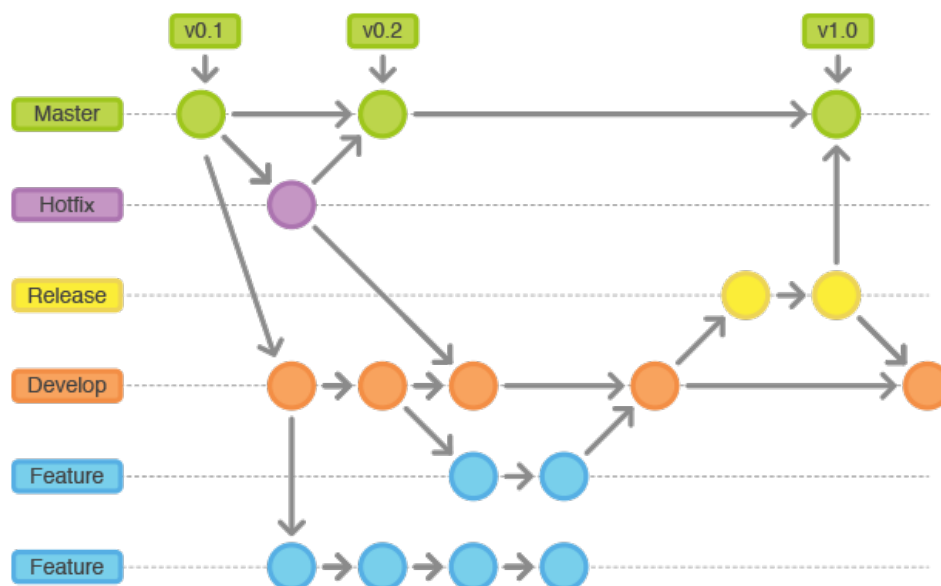
Door middel van version control kunnen meerdere ontwikkelaars aan dezelfde software werken. Tijdens JEA6 zal Git (GitHub) gebruikt worden voor version control. Er is hiervoor gekozen omdat hiermee reeds de meeste ervaring is opgedaan.



Figuur 2.1: Overzicht van default branch en protected branches

2.1 Branches

Voor de repository waar de code van het vak JEA6 wordt bijgehouden, zijn verschillende branches aangemaakt. Wat het nu is van deze verschillende branches, staat in de volgende alinea's uitgelegd.



Figuur 2.2: Schematisch overzicht van de branches

2.1.1 Master

De code die beschikbaar is in de master-branch, is de code van de software versie die in productie gebruikt wordt. Nieuwe functies of bug fixes mogen dan ook niet rechtstreeks naar deze branch gecommited worden. Deze branch is tevens protected. Dit wilt zeggen

dat vanuit een git programma (UI of CLI) niet rechtstreeks naar deze branch gecommitt kan worden. Dit zorgt ervoor dat een ontwikkelaar niet per ongeluk naar deze branch kan committen.

Branch protection for **Testing**

☒ **Protect this branch**
Disables force-pushes to this branch and prevents it from being deleted.

☒ **Require pull request reviews before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with at least one approved review and no changes requested before it can be merged into **Testing**.

☒ **Dismiss stale pull request approvals when new commits are pushed**
New reviewable commits pushed to a branch will dismiss pull request review approvals.

☐ **Require review from Code Owners**
Require an approved review in pull requests including files with a designated code owner.

☐ **Require status checks to pass before merging**
Choose which **status checks** must pass before branches can be merged into **Testing**. When enabled, commits must first be pushed to another branch, then merged or pushed directly to **Testing** after status checks have passed.

☒ **Include administrators**
Enforce all configured restrictions for administrators.

Save changes

Figuur 2.3: Overzicht van protectie wat ingesteld is op branch

2.1.2 Hotfix

2.1.3 Testing

De Testing-branch zal gebruikt worden om nieuwe functionaliteit die ontwikkeld zijn in de development branch te testen. Wanneer de functionaliteit in de testing branch geaccepteerd wordt, zal deze worden doorgezet naar de master branch zodat de verbeteringen en/of nieuwe wijzigingen beschikbaar zijn voor alle gebruikers op de productie omgeving. De Testing-branch heeft dezelfde setup als de Master-branch. Dit wilt zeggen dat het op deze branch ook niet mogelijk is om rechtstreeks code te committen.

De testing omgeving die gebruikt wordt om de code van de testing branch beschikbaar te stellen, kan tevens gebruikt worden door testers om nieuwe functionaliteit of verbeteringen te testen. Daarnaast kan de release manager akkoord geven op de versie die in deze omgeving staat, waardoor deze verplaatst zal worden naar de productie omgeving

2.1.4 Development

De Development-branch bevat nieuwe functionaliteit of verbeteringen wat ontwikkeld zijn. Deze branch is vrij toegankelijk voor de ontwikkelaar om eventueel kleine(re) verbeteringen door te voeren. Het is als nog de bedoeling dat nieuwe functionaliteit niet rechtstreeks op deze branch wordt gecommitt. Mocht een ontwikkelaar nieuwe functionaliteit of grotere verbeteringen willen ontwikkelen, kan hiervoor een zogenaamde Feature-branch worden aangemaakt. Meer informatie hierover is in de volgende alinea te lezen

2.1.5 Features

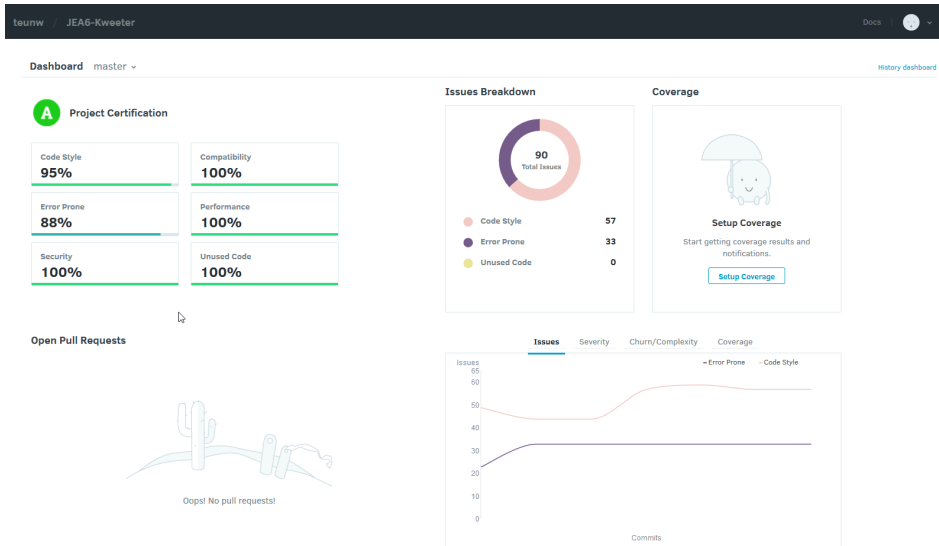
De Features-branch is geen branch die direct aangemaakt is op Github. Dit komt doordat "Features" een overkoepelende term is. Wanneer een ontwikkelaar nieuwe functionaliteit of verbeteringen wilt ontwikkelen, dan zal hiervoor een nieuwe branch moeten worden aangemaakt, bijvoorbeeld "Dev-exampleFeature". Aan de hand van de prefix "Dev-" weet iedereen dat dit een branch betreft waar nieuwe functionaliteit of verbeteringen op worden ontwikkeld. Het gedeelte achter de prefix moet dan ook beschrijven aan welke functionaliteit of verbeteringen worden gewerkt.

Wanneer de ontwikkelaar het implementeren heeft voltooid, zal dit eerst getest moeten worden op zijn eigen, lokale omgeving. Wanneer die nieuwe functionaliteit werkt met de rest van de ontwikkelde software, kan de ontwikkelaar zijn branch, bijvoorbeeld "Dev-exampleFeature" gaan samenvoegen met de Development-branch. Hiervoor maakt de ontwikkelaar een merge request aan wat ervoor zorgt dat de nieuwe code wordt samengevoegd op de Development-branch.

2.1.6 Bugfix/Patches

3 Code Style

Voor de code style worden de standaard code style settings van IntelliJ IDEA gebruikt. De code style wordt ook gecontroleerd door Codacy, een code analyse tool. Codacy controleert je Java (en Javascript) codebase op verschillende errors.



Figuur 3.1: Voorbeeld van code analysis door middel van Codacy

Ook kan er gecontroleerd worden op beveiligingsproblemen in je applicatie, stel dat je bijvoorbeeld SQL-query parameters in je queries hebt staan, dan wordt dat netjes gerapporteerd. Andere dingen waartegen je beschermt wordt zijn:

- Authentication
- CSRF aanvallen
- File access
- SQL Injection
- XSS







Natuurlijk biedt Codacy ook een gedetailleerd overzicht van de verschillende fouten in je code.



Figuur 3.2: Verschillende fouten in een Typescript bestand

3.1 Release

Codacy heeft de mogelijkheid om branches individueel te controleren op code. Hierdoor kan je per branch inzien wat voor cijfer deze krijgen. Branches worden aan een hoge standaard gehouden, de code moet minimaal een A krijgen om te mogen mergen met de development.

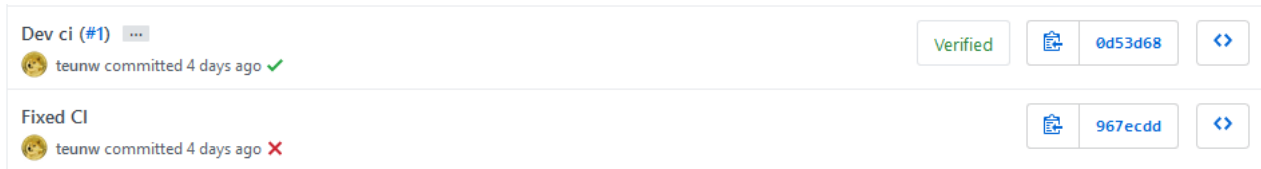
BRANCH	ANALYSIS	GRADE
add-license-1		
dev-readme		
master <small>Main branch</small>		

Figuur 3.3: Branches en hun cijfer

Er is geen infrastructuur om te zorgen dat code met een laag cijfer niet gemerged kan worden, dit is een limitatie van de infrastructuur (dus van Github en Codacy). Op hotfix branches hoeft niet meteen een controle uitgevoerd te worden, aangezien een hotfix zo snel mogelijk live moet zijn is dit onwenselijk. Wel zou deze controle uitgevoerd moeten worden nadat de hotfix toegepast is, omdat slechte code in deze branch het mergen van andere zou kunnen hinderen. De ontwikkelaar is verantwoordelijk voor zijn/haar eigen branch, het is de taak van de ontwikkelaar om een slecht cijfer te repareren.

4 Continuous Integration

Continuous Integration is geregeld door TravisCI, voor open source projecten is TravisCI gratis te gebruiken. Omdat het project met Gradle is gebouwd is het makkelijk om CI ondersteuning toe te voegen aan de backend. Het commando "gradle test" draait alle tests in het project, als er iets fout gaat is dit ook meteen in Github zichtbaar.



Figuur 4.1: Overzicht TravisCI

Travis CI kan geconfigureerd worden met een bestand dat `.travis.yml` heet, dit bestand is zoals te zien in `.yml` formaat. Dit bestand moet in de root van je project geplaatst worden.

```
script:
  - cd "./Kweeter-Backend"
  - gradle test
notifications:
  email: false
```

Figuur 4.2: TravisCI Yaml

Het bovenstaande script zorgt ervoor dat gradle tests automatisch worden uitgevoerd. De uitslag van deze test is zichtbaar in het commit overzicht, zoals in fig. 4.1 te zien is. Het TravisCI script wordt uitgelezen en uitgevoerd op een Ubuntu server van Travis, ontwikkelaars hebben hier zelf geen servers voor nodig. De output van dit script kan ook uitgelezen worden op de website, dit zorgt ervoor dat je kan kijken hoe je tests draaien, en eventueel de oorzaak van errors kan vinden.

4.1 Pull requests & merging

TravisCI draait je scripts automatisch bij elke commit die naar Github wordt gemaakt, en het resultaat is binnen een minuut zichtbaar. Daarnaast kan een pull request worden tegengehouden zodra TravisCI een fout detecteerd in je build. Als dit gebeurt moet de code eerst gerepareerd worden zodat de build weer werkt, daarna kan de pull request gemerged worden.

5 Versioning

Door gebruik te maken van versioning kan het voor een ontwikkelaar duidelijk zijn in welke versie van de software een eventueel probleem is aangetroffen. Een manier om versioning toe te passen is door gebruik te maken van Semantic Versioning (versie 2.0.0). Meer informatie en de volledige specificatie van Semantic Versioning is beschikbaar op hun website: <https://semver.org>

Bij Semantic Versioning is een versienummer opgebouwd uit drie verschillende delen. Namelijk:

- Major
- Minor
- Patch

5.1 Major

Het eerste getal van het versie nummer geeft de versie van de software aan. Dit getal begint meestal bij 1, maar in sommige gevallen kan ook versie 0 voorkomen. In deze gevallen geeft de 0 aan dat de software nog in ontwikkeling is, en dat de software nog niet als "stable" gezien kan worden. Wanneer het major nummer van de versie veranderd, geeft dit meestal aan dat er wijzigingen in de API zijn gedaan die niet meer compatible zijn met een vorige versie. Zo kan de data die naar een endpoint gestuurd wordt bijvoorbeeld op een andere manier worden terug gegeven.

5.2 Minor

Het tweede getal in het versienummer geeft aan welke wijziging zijn toegevoegd aan het huidige software pakket. Wijzigingen die ervoor zorgen dat het minor getal toeneemt, is meestal nieuwe functionaliteit wat beschikbaar wordt gesteld. Een vereiste van deze functionaliteit is dat deze compatible is met de huidige implementatie van de software. Wanneer een of meerdere functie's niet meer ondersteund worden, moet dit volgens de Semantic Versioning specificatie ook worden weergegeven in het minor nummer van de versie.

5.3 Patch

Wanneer in de huidige versie van de software een fout wordt ontdek en dit verholpen moet worden, een zogenaamde "bugfix". Deze bugfix kan ook worden weergegeven in het versienummer van de software. Dit gebeurt aan de hand van het derde nummer: het patch nummer. Het is vanzelfsprekend dat de huidige functionaliteit niet veranderd mag worden. Mocht dit wel het geval zijn, dan is het ook vereist dat het major of minor nummer van de versie aangepast wordt.

5.4 Overige Informatie

Bij de opbouw van het versie nummer is het niet toegestaan om een nummer te laten beginnen met 0 (bv. 1.01.0 is niet toegestaan). Een uitzondering hierop is de opbouw van het versie nummer wanneer de software nog in ontwikkeling is zoals beschreven in *5.1 Major*.

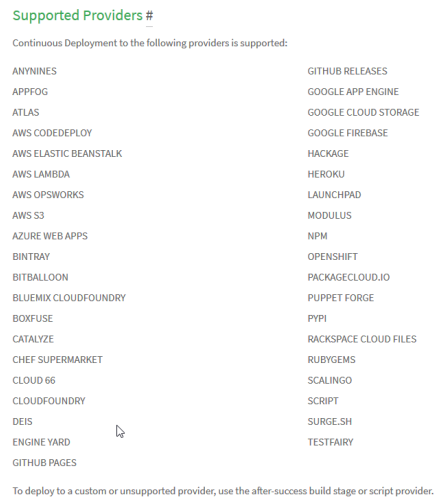
Wanneer ervoor gekozen wordt om een pre-release versie van de software beschikbaar te stellen, kan dit tevens worden weergegeven in het versie nummer. Dit gebeurt door de

versie van de pre-release door middel van een koppelteken te plaatsen achter het daadwerkelijke versie nummer van de software. Een aantal voorbeelden hiervan zijn: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7

Eventuele build metadata mag tevens worden weergegeven in het versie nummer. Dit wordt gebruikelijk direct na het patch nummer of de pre-release versie weergegeven met behulp van het koppelteken +.

6 Continuous Delivery

Met TravisCI kun je ook continuous delivery regelen, op de Travis worden verschillende providers genoemd.



Figuur 6.1: Providers met support in TravisCI

Het deployen van applicaties gebeurt ook via de .travis.yml. Omdat de .travis.yml file publiekelijk toegankelijk is, is er ook de mogelijkheid om de wachtwoorden van verschillende services encrypted in de file te zetten. Deze encryptie moet ervoor zorgen dat anderen niet zomaar op jouw account acties kunnen uitvoeren.