Onderzoek GitHub

Ik heb de opdracht gekregen om het thema GitHub te onderzoeken, dit is belangrijk voor een correcte oplevering van de casus.

Er zijn op dit moment drie taken:

1. Het verzamelen van documentatie over GitHub
2. Het doen van een tutorial zodat we later alles in een keer kunnen opleveren
3. Kijken wat we bij de oplevering moeten gebruiken

Hoofdvragen:

1. Wat is GitHub? v/
2. Hoe documenteer je binnen GitHub? v/

Inhoud

[Documentatie 3](#_Toc55306809)

[Hello world! 5](#_Toc55306810)

[Best practices 6](#_Toc55306811)

[1. Push **nooit** direct naar de master 6](#_Toc55306812)

[2. Commit geen code als niet-herkende auteur 6](#_Toc55306813)

[3. Definieer code-eigenaren voor snellere codebeoordelingen 6](#_Toc55306814)

[4. Lek geen geheimen naar bronbeheer 7](#_Toc55306815)

[5. Leg geen dependencies\*/ afhankelijkheden vast in bronbeheer 7](#_Toc55306816)

[6. Commit geen locale config-files\*(configuratie bestanden) naar bronbeheer 7](#_Toc55306817)

[7. Maak een .gitignore bestand 7](#_Toc55306818)

[8. Archiveer “dode” repositories 8](#_Toc55306819)

[9. Vergrendel pakketversie 8](#_Toc55306820)

[10. Specificeer standaardpakketversies 8](#_Toc55306821)

[11. Maak gebruik van de takenlijst 9](#_Toc55306822)

[12. Gebruik een branch naming convention\* 9](#_Toc55306823)

[13. Verwijder oude branches 10](#_Toc55306824)

[a. Een branch vewijderen 10](#_Toc55306825)

[b. Een branch herstellen 10](#_Toc55306826)

[14. Houd branches up-to-date 10](#_Toc55306827)

[15. Verwijder inactieve GitHub leden 10](#_Toc55306828)

[16. Schakel beveiligingswaarschuwingen in 10](#_Toc55306829)

[Bijlage: 11](#_Toc55306830)

[Maak een nieuwe repository: 11](#_Toc55306831)

[Maak een branch 12](#_Toc55306832)

[Maak en voer veranderingen door 13](#_Toc55306833)

[Open een pull request 14](#_Toc55306834)

[Merge de pull request 15](#_Toc55306835)

[CODEOWNERS file 16](#_Toc55306836)

# Documentatie

**Wat is GitHub?**

GitHub bestaat uit twee woorden, namelijk Git + Hub.

**Waar staat Git voor?**

**Git is een open source** **versiebeheersysteem**. De **website GitHub** is gemaakt op het versiebeheersysteem Git, je hebt bij GitHub dan ook alle mogelijkheden van Git + extra features.

**Waar staat Hub voor?**

Met “Hub” wordt eigenlijk het centrale deel bedoeld, waar alle projecten samenkomen die beheerd worden met Git. De website GitHub is de centrale plek waar programmeurs hun projecten beheren met Git.

**Waar word GitHub voor gebruikt?**

In een versiebeheersysteem als GitHub kunnen programmeurs projecten beheren met code. Het is mogelijk om verschillende versies te beheren en eventueel terug te vallen op een oude versie, mocht er iets misgaan.

**Welke andere mogelijkheden heeft GitHub?**

Het is duidelijk dat GitHub wordt gebruikt om het versiebeheer van een project bij te houden, Je kunt ook zelf een project bijhouden op GitHub (hier heb je alleen een account voor nodig!)

**Welk doel heeft het gebruiken van GitHub?**

Ik denk dat je door middel van GitHub een beter overzicht krijgt van de aanpassingen die je doet binnen je code en het geeft een simpelere mogelijkheid om aan meerdere versies te werken. Nu er veel vanuit thuis gewerkt wordt zal dit handig zijn wanneer een groep programmeurs aan dezelfde code moet gaan werken.

Je weet wie, wat en wanneer heeft gedaan. De community is groot (40 miljoen!) dus er zal genoeg hulp zijn bij vragen.

Verder zijn er deze noemenswaardige mogelijkheden:

* Een project maken gebaseerd op een project dat al bestaat
* Discussie starten over een project
* Code reviewen
* Aparte branches maken, waarin je bijvoorbeeld code aanpassingen doet om te testen, die niet gelijk in de “productieversie” komen
* Code van branch samenvoegen met andere branch
* Tags meegeven aan verschillende versies, zoals V1.0 en V2.0
* Kwetsbaarheden in de code makkelijker ontdekken, GitHub stuurt ook een mail als het kwetsbaarheden ontdekt
* Een website hosten

**Wat zijn specifieke termen die in GitHub gebruikt worden?**

|  |  |
| --- | --- |
| **GitHub term** | **Betekenis** |
| Repository | Een repository kan je zien als de hoofdmap van je project. Hierin staan alle bestanden van je project en de historie van wijzigingen die je hebt gedaan. |
| Branch | Aparte plek binnen je repository, waar je bijvoorbeeld nieuwe code kan testen, zonder dat te hoeven doen op de “productversie”. |
| Master | Dit is de hoofdbranch, oftewel de “productversie” van je project. Nieuwe code die is getest en goedgekeurd, wordt samengevoegd met de Master branch. |
| Fork | Een fork is een kopie van een repository. Hierdoor kan je werken aan een project van iemand anders, zonder het origineel aan te passen. |
| Commit | Git commando die veranderingen toevoegt aan je lokale repository. |
| Push | Git commando om aanpassingen naar je (remote) repository te sturen, die staat op GitHub. |
| Pull | Git commando om aanpassingen van je (remote) repository naar je lokale bestanden te sturen. |
| Merge | Git commando om aanpassingen van een branch samen te voegen met een andere branch. Bijvoorbeeld aanpassingen die getest en goedgekeurd zijn in de “Develop” branch samenvoegen met de “Master” branch. |
| Checkout | Deze Git commando wordt vaak gebruikt om te switchen tussen branches. Je checkt als het ware uit bij een branch en gaat aan de slag in een andere branch. |

# Tutorials

## Hello-world

Ik weet dat het bijgeloof is, maar voor de tutorial heb ik gekozen voor de “hello world!” tutorial die beschikbaar is op GitHub.com (<https://guides.github.com/activities/hello-world/>)

In deze tutorial worden de volgende vier onderwerpen uitgelegd:

* Create and use a repository (een hoofdmap aanmaken)
* Start and manage a new branch (belangrijk voor verschillende versies)
* Make changes to a file and push them to GitHub as commits
* Open and merge a pull request

Ik werk ieder onderwerp uit en maak een screenshot van de output, deze zullen terug te vinden zijn in de bijlage van dit document.

## Mark down

Markdown is een lichtgewicht en gemakkelijk te gebruiken syntaxis voor het stylen van alle vormen van schrijven op het GitHub-platform.

# Best practices

Om ervoor te zorgen dat er geen voorkombare foutjes gemaakt worden heb ik de volgende lijst aan best practices\* gevonden: (<https://www.datree.io/resources/github-best-practices>)

**\*Wat is een best practice?**

De beste werkwijze om iets te doen

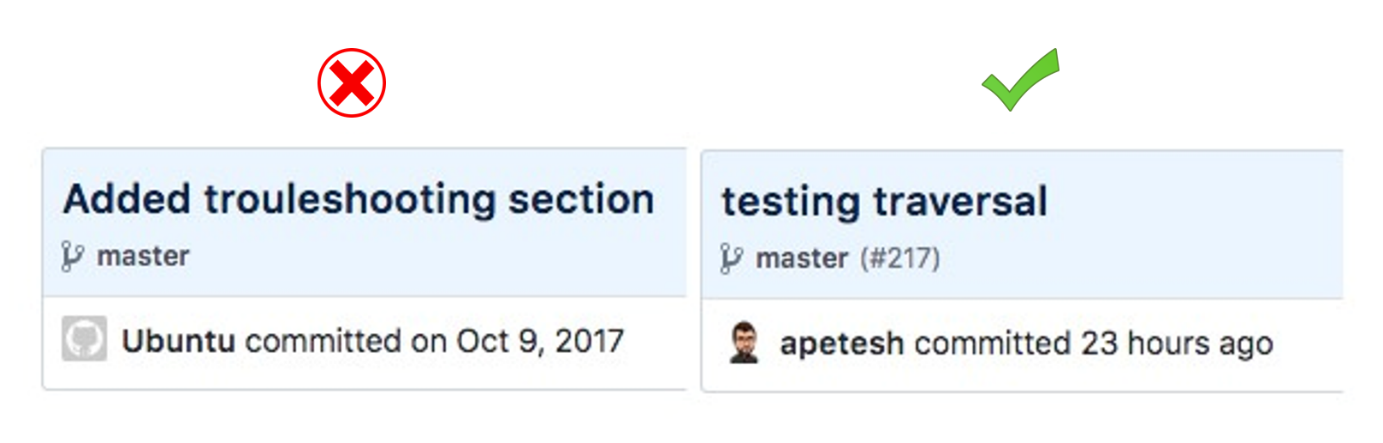
## Push **nooit** direct naar de master

het is altijd een goed idee om git branch protection aan te zetten om directe commits te voorkomen en ervoor te zorgen dat je main branch code te allen tijde inzetbaar is. Alle commits moeten worden gepusht door middel van pull-requests.

## Commit geen code als niet-herkende auteur

Soms commit je code met het verkeerde e-mailadres, en als resultaat laat GitHub zien dat je commit een niet-herkende auteur heeft . Het hebben van commits met niet-herkende auteurs maakt het moeilijker om bij te houden wie welk deel van de code heeft geschreven.

Om te checken of er bij een commit een correcte auther genoteerd staat kun je bij de pull requests kijken of hier een nietherkende autheur staat. Leer om dit na iedere pull-request te doen!



## Definieer code-eigenaren voor snellere codebeoordelingen

Door GitHub wordt het mogelijk om heel veel dingen tegelijk te doen, als er bijvoorbeeld honderden of duizenden repositorys zijn waar meerdere programmeurs aan werken wordt het moeilijker om te zien welke stukken code je bezit en moet reviewen.

GitHub lost dit op door een feature die “**Code owners**” heet, dit is een bestand waarin je een team definiëren dat verantwoordelijk is voor deze repository (<https://docs.github.com/en/free-pro-team@latest/github/creating-cloning-and-archiving-repositories/about-code-owners>)

Ik heb in de bijlage een voorbeeld gezet van een CODEOWNERS file.

## Lek geen geheimen naar bronbeheer

Geheimen, of geheime sleutels of geheime referenties, omvatten zaken als accountwachtwoorden, API-sleutels, privé-tokens en SSH-sleutels. Deze moet je niet in de broncode laten rondslingeren!

Dit soort geheime informatie kan beter vanuit een veilige externe opslag worden gehaald, in voorbeeld van een veilige externe opslag tool dat is ingericht voor dit soort geheime informatie is Vault (<https://www.vaultproject.io/>).

In deze turtorial wordt er verder in gegaan over het beheren van geheimen: (<https://www.datree.io/resources/secrets-management-aws>)

Omdat we binnen dit project niet te maken hebben met “geheime informatie” hoeft deze tutorial niet te worden uitgevoerd.

## Leg geen dependencies\*/ afhankelijkheden vast in bronbeheer

Door depencies naar de externe origin te puschen, wordt de repository groter, Verwijder alle projectafhankelijkheden die zijn opgenomen in de repositories en laat een pakketbheerder ze downloaden tijdens iedere build. Wanneer er zorgen ontstaan om de availability van dependencies kan er een binaire repository manager- oplossing overwogen worden. De b.r.m van GitHub heet “Git-Sizer” (<https://github.com/github/git-sizer>).

**\*Wat zijn dependencies?**

**/**

## Commit geen locale config-files\*(configuratie bestanden) naar bronbeheer

Privé configuratie bestanden mogen niet gepushed worden naar een remote locatie omdat deze geheimen, persoonlijke voorkeuren, geschiedenis of algemene informatie kunnen bevatten die allen in de lokale omgeving moeten blijven.

**\*Wat zijn config-files?**

**/**

## Maak een .gitignore bestand

Een .gitignore bestand wordt gebruikt om vooraf definieerde bestanden en mappen te negeren, het helpt om het lekken van geheime sleutels te voorkomen. Op toptal staan veel templates van .gitignore bestanden (<https://www.toptal.com/developers/gitignore>)

**Hebben wij in dit project een .gitignore bestand nodig?**

**/**

**Waar moeten we naar kijken bij het kiezen van een template?**

**/**

## Archiveer “dode” repositories

Een niet onderhouden repository (dead repository) kan op verschillende manier ontstaan: Soms maken ontwikkelaars repo's voor een ad-hoc-use-case\*, een POC of een andere reden. Soms erven ze opslagplaatsen met oude en irrelevante code. In ieder geval zijn deze repo’s dus intact gelaten, maar niemand doet er programmeer werk mee, dus het is verstandig om deze repo’s op te ruimen zodat ze niet perongelijk gebruikt worden.

De beste manier om dit te doen is om de repository op “**read-only**” te zetten

**\*Wat betekend adhoc use-case?**

Jane Mairs zegt,“ Ad hoc is a word that originally comes from Latin and means “for this” or "for this situation." In current American English it is used to describe something that has been formed or used for a special and immediate purpose, without previous planning.”(Mairs, 2013)

Een repository voor een adhoc use case is dus een repository voor een use-case, die specifiek voor deze use-case gemaakt is en met als enige doel heeft om deze use-case in uit te werken.

## Vergrendel pakketversie

Het manifestbestand bevat informatie over alle packages\* en dependencies van het project en haar versies.

De beste werkwijze is een om een versie of versiebereik op te geven voor elk pakket en elke afhankelijkheid die in het manifest\* worden vermeld.

Anders weet je niet zeker welke versie wordt geïnstalleerd tijdens de volgende build, en als gevolg kan de code breken.

**Wat is een package?**

**/**

**Wat is een manifest?**

/

## Specificeer standaardpakketversies

Zelfs wanneer iedereen in je team dezelfde pakketten/packages gebruikt, kan hergebruik en het testen van code voor verschillende projecten moeilijk zijn als de pakketten van verschillende versies zijn.

Als je een pakket hebt dat in meerdere projecten wordt gebruikt, probeer dan op zijn minst dezelfde hoofdversie van het pakket te gebruiken.

**Hoe zie je de package informatie van een repository?**

Je kunt alle paketten in een orgsanisatie bekijken maar ook de pakketten die je zelf hebt gepubliceerd, Dit doe je door op de hoofdpagina van de repository te klikken op packages(<https://docs.github.com/en/free-pro-team@latest/packages/publishing-and-managing-packages/viewing-packages>).

## Maak gebruik van de takenlijst

Takenlijsten bieden u een manier om taken direct bij te houden in opmerkingen, problemen en zelfs MarkDownfiles\*

Takenlijsten bieden een uitstekende manier om een ​​overzicht op hoog niveau van een taak of probleem vast te leggen, en om andere op de hoogte te houden van de status ervan.

**Wat is een markdownfile?**

**/**

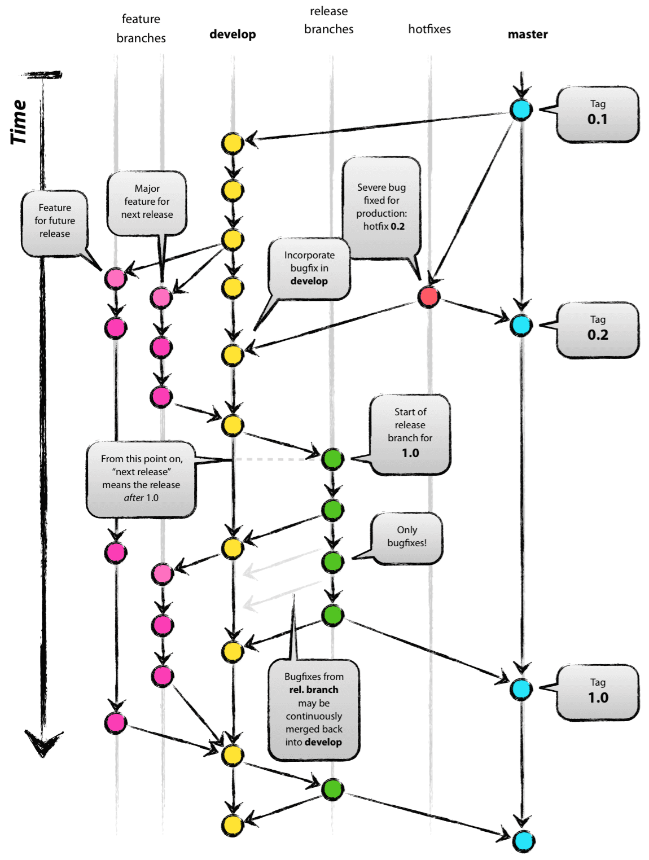
gebruikers moeten schrijftoegang hebben tot de opslagplaats om wijzigingen aan te brengen in MarkDownfiles

**Waar moeten deze taaklijsten genoteerd worden?**

**/**

## Gebruik een branch naming convention\*

Het is belangrijk om consistentie naamgevingsconventie te hebben voor het bedenken van een branch naam. Door middel van een efficiënte naamgevingsconventie worden samenvoegconflicten tot een minimum. Een voorbeeld van een bekende naamgevingsconvetie is git flow (<https://nvie.com/posts/a-successful-git-branching-model/>):



**\*Wat is een naming convention?**

Een naamgevingsconventie is een algemeen overeengekomen schema voor het benoemen van dingen. Het zijn dus afspraken over hoe je bijvoorbeeld een bestand moet noemen. Het is vooral belangrijk dat wanneer deze er is, iedereen in het team deze toepast (dit wordt al lastiger als het om een groot team gaat).

## Verwijder oude branches

Elke keer dat een branch wordt samengevoegd met een andere, wordt de branch die wordt samengevoegd oud(ervan uitgaande dat er geen verder werk in wordt gedaan).

Hoewel het misschien nuttig of zelfs noodzakelijk lijkt om de extra gegevens bij de hand te houden, is de realiteit dat verouderde takken 98% van de tijd worden niet meer gebruikt worden en het project rommelig maken.

### Een branch vewijderen

Na het mergen van een pull request verschijnt er automatisch een pop-up die je vraagt of je de branch wil verwijderen, ook kun je op de repository pagina van deze branch op het tandwieltje klikken waar onderaan de optie om een branch te verwijderen vermeld staat.

### Een branch herstellen

Het is mogelijk om een branch van een afgesloten pullrequest te herstelllen als bijvoorbeeld de code van een verwijdere feature terug moet worden gehaal(<https://docs.github.com/en/free-pro-team@latest/github/administering-a-repository/deleting-and-restoring-branches-in-a-pull-request>)

## Houd branches up-to-date

stel: Laten we zeggen dat je eindelijk wat werk afhebt op een lang openstaande branch en je bent klaar om het in de master branch mergen. Je pulled vanaf remote, klikt op merge en wordt plotseling geconfronteerd met een muur aan samenvoegconflicten.

Wat is er gebeurd?

Het is je niet gelukt om je branch up-to-date te houden met de branch waarin je probeert te mergen. Er gingen veel commits voorbij en sommige waren in strijd met je wijzigingen ... nu moet je tijd en energie besteden aan het oplossen van een onnodig aantal samenvoegconflicten.

De beste praktijk hier is om ervoor te zorgen dat je **consequent** je basisbranch met je huidige branch samenvoegt terwijl je werkt, vooral als het een lang openstaande branch is.

## Verwijder inactieve GitHub leden

Zorg ervoor dat u bijdragers uit uw organisatie verwijdert die niet langer bijdragen aan uw codebase, je weet maar nooit.

## Schakel beveiligingswaarschuwingen in

Wanneer GitHub een zwakte in de repository vind (bijvoorbeeld een dependency probleem) zal hij je alerten en mogelijk een oplossing geven. Dit is handig zodat je weet dat er überhaupt een probleem is en gerichter kan zoeken naar een oplossing.

(<https://docs.github.com/en/free-pro-team@latest/github/managing-security-vulnerabilities/about-alerts-for-vulnerable-dependencies>)

# XToepassing in casus

In de repository van het project staat een template van school, ik zal hier ieder onderdeel beschrijven zodat dit gebruikt kan worden als referentie.

## Repository structuur



We maken één repository waarin wij onze “final version” inleveren, de structuur van deze repository wordt in de rechter afbeelding weergegeven 🡪

Legenda:

* **README.md**
  + Casus beschrijving (in een mark-down file).
  + Links naar GitHub Pages
    - documentatie
    - installatie instructie
* **.gitignore**
  + Files die niet worden meegenomen in het versie beheer, zo nodig gegenereerd.
* **.gitattributes**
  + Files die opgeslagen worden door

middel van Git LFS.

* **(Folder) docs**
  + GitHub Pages met de complete documentatie (homepage is extended abstract)
* **(Folder) <project>**
  + Project met daarin code en/of andere deliverables
* (Folder) lfs
  + Binary files, zoals afbeeldingen of PDF's, die wel in de repository moeten staan

Door het scannen van het template dat zuyd gegeven heeft zijn de volgende vragen ontstaan:

**Wat zijn Markdown files?**

## Markdown files

(<https://guides.github.com/features/mastering-markdown/>)

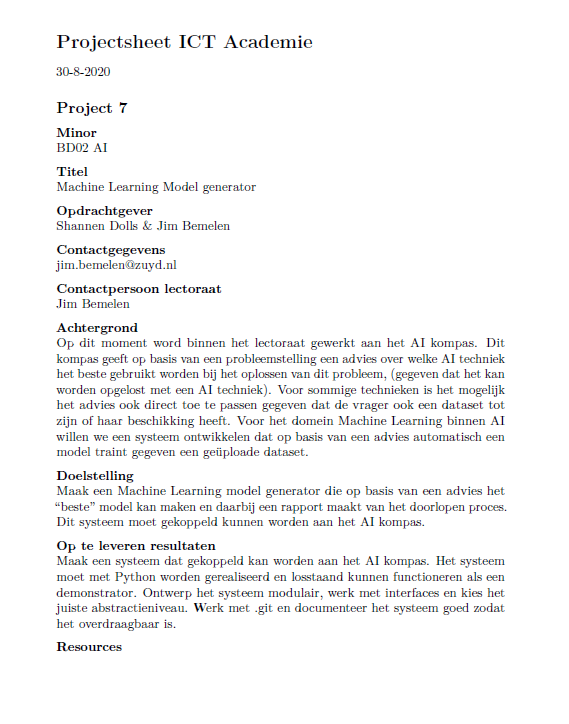
Markdown is een lichtgewicht en gemakkelijk te gebruiken syntaxis voor het stylen van alle vormen van schrijven op het GitHub-platform. Markdown is dus een manier om tekst te stylen, dit moeten wij gebruiken bij de readme.md, dit kun je zien aan het type bestand (.md)

**Waar meoten wij mark-down files voor gebruiken?**

We moeten de casus beschrijving die wij hebben in word omzetten naar de casusbeschrijving in een mark-down file, dit betekent dat de tekst moet worden aangepast en worden vertaald naar een mark-down opmaak.

**Maken wij deze files zelf of gebruiken we een soort van pdf 🡪 .md converter?**

De casusbeschrijving in een pdf document

In de tekst word gebruik gemaakt van koppen, vetgedrukte woorden en andere letter grote.

Om te kijken hoe een markdown file er precies uitziet is het bovenstaande pdf bestand ingevoert bij een pdf 🡪 .md converter. Hieronder staat het resultaat van de conversie(<https://pdf2md.morethan.io/>):

Het valt op dat bij ieder woord dat niet bestaat uit “standaard tekst” een teken staat, ik denk dat we aan de hand van deze tekens kunnen aangeven hoe een woord eruit hoor te zien.

In de bijlage staat een cheat sheet waar de syntax van een mark-down bestand beschreven staat.

|  |
| --- |
| # Projectsheet ICT Academie  ### 30-8-2020  ## Project 7  \*\*Minor\*\*\  BD02 AI  \*\*Titel\*\*\  Machine Learning Model generator  \*\*Opdrachtgever\*\*\  Shannen Dolls & Jim Bemelen  \*\*Contactgegevens\*\*\  jim.bemelen@zuyd.nl  \*\*Contactpersoon lectoraat\*\*\  Jim Bemelen  \*\*Achtergrond\*\*\  Op dit moment word binnen het lectoraat gewerkt aan het AI kompas. Dit  kompas geeft op basis van een probleemstelling een advies over welke AI techniek  het beste gebruikt worden bij het oplossen van dit probleem, (gegeven dat het kan  worden opgelost met een AI techniek). Voor sommige technieken is het mogelijk  het advies ook direct toe te passen gegeven dat de vrager ook een dataset tot  zijn of haar beschikking heeft. Voor het domein Machine Learning binnen AI  willen we een systeem ontwikkelen dat op basis van een advies automatisch een  model traint gegeven een geüploade dataset.  \*\*Doelstelling\*\*\  Maak een Machine Learning model generator die op basis van een advies het  “beste” model kan maken en daarbij een rapport maakt van het doorlopen proces.  Dit systeem moet gekoppeld kunnen worden aan het AI kompas.  \*\*Op te leveren resultaten\*\*\  Maak een systeem dat gekoppeld kan worden aan het AI kompas. Het systeem  moet met Python worden gerealiseerd en losstaand kunnen functioneren als een  demonstrator. Ontwerp het systeem modulair, werk met interfaces en kies het  juiste abstractieniveau. Werk met .git en documenteer het systeem goed zodat  het overdraagbaar is.  \*\*Resources\*\* |

### checklistnaam

- [ ] taak

- [ ] 🡨 nog te volbrengen

- [x]🡨 volbracht

**Hoe uploaden we de mark-up bestanden?**

**Visual studio code?**

Er is een extention in visual studio code waarmee je een pull request kunt maken, dit heb ik echter niet aan de praat gekregen omdat hij steeds een validatie error aangeeft.

**GitHub.com?**

Het is me gelukt om de bestanden die er al instaan aan te passen (de readme.md) ik zal wanneer alle correcte informatie en alle links zijn toegevoegd de originele inhoud van de readme.md verwijderen.

Ik zal ook een checklist in mark down erin zetten zodat er snel naar deze gerefereerd kan worden op een handige plaats.

**Wat zijn GitHub Pages?**

Git hub pages zijn een soort van website die gerunt kan worden op github, ik vind dit echter niet echt nodig en heb daarom een alternatief om te overwegen:

Door een pdf bestand in de .ifs map te zetten en hiernaar te linken vanaf de readme.md pagina word er een link gemaakt naar een github page waarin de pdf word weergeven.

Pdf opslaan in .ifs map 🡪 link in README.md zetten 🡪 documentatie aan GitHub toegevoegd

**Wat zijn .gitignore files?**

**Wat zijn onze .gitignore files?**

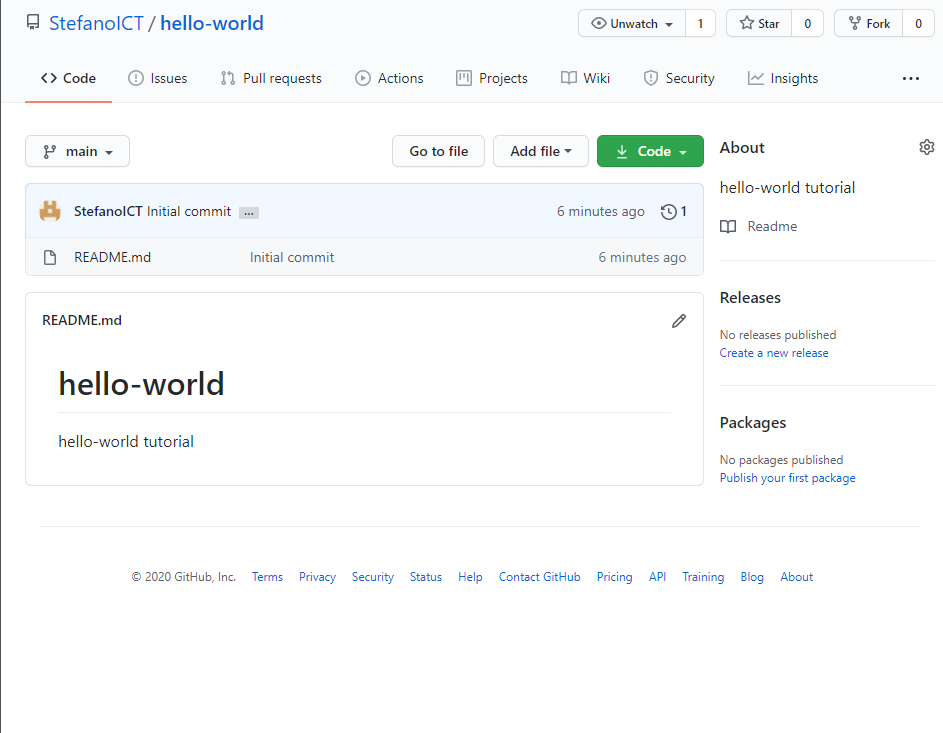
**Wat zijn files die opgeslagen worden door middel van Git LFS?**

**Hoe maken we onze GitHub pages?**

# Bijlage:

## Maak een nieuwe repository:

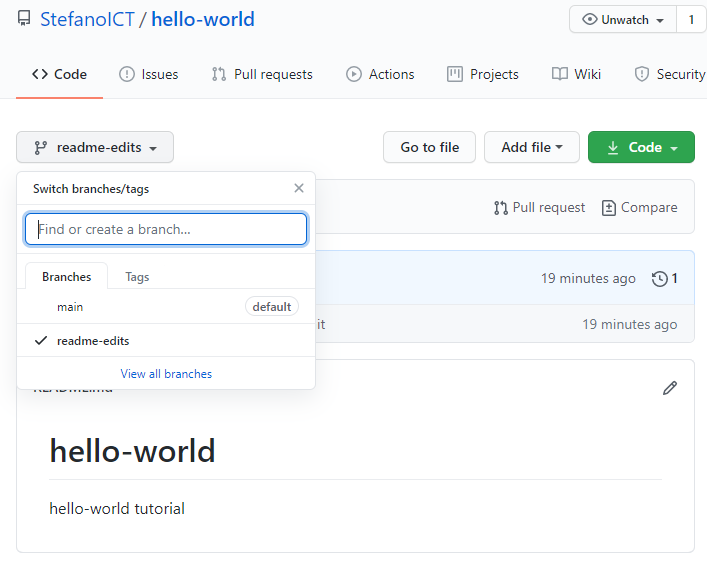
|  |
| --- |
| **To create a new repository**   1. In the upper right corner, next to your avatar or identicon, click  and then select **New repository**. 2. Name your repository hello-world. 3. Write a short description. 4. Select **Initialize this repository with a README**.   new-repo-form  Click **Create repository**. |

Er wordt dus een hoofdmap aangemaakt, Deze is public (voor iedereen toegankelijk) of **private**(jij kiest wie aanpassingen aan de code mag doen) bij het aanmaken kun je ook kiezen om er een readmy aan toe te voegen, ook een licentie bestand is mogelijk.

Deze stap heeft school al voor ons gedaan toen ze ons via e-mail uitnodigde voor de zuyd (de “BD02-Modelgenerator-groep-4-” repository)

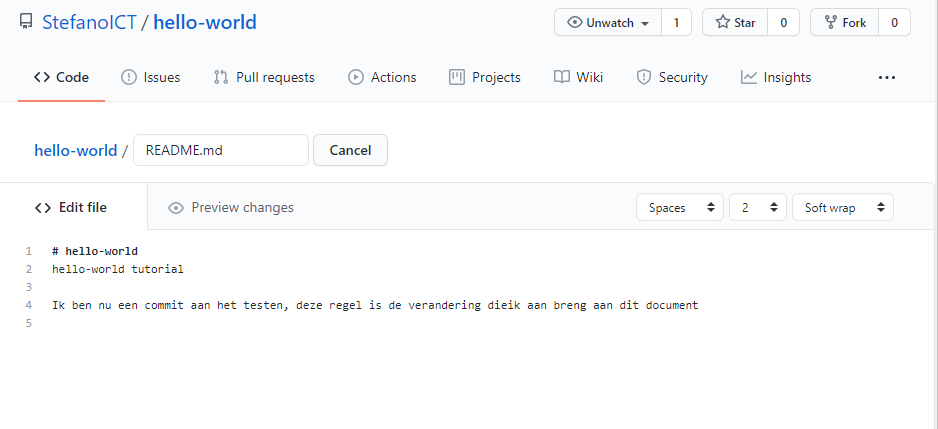
## Maak een branch

|  |
| --- |
| **Branching** is the way to work on different versions of a repository at one time.  By default your repository has one branch named main which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to main.  When you create a branch off the main branch, you’re making a copy, or snapshot, of main as it was at that point in time. If someone else made changes to the main branch while you were working on your branch, you could pull in those updates.  This diagram shows:   * The main branch * A new branch called feature (because we’re doing ‘feature work’ on this branch) * The journey that feature takes before it’s merged into main   a branch  Have you ever saved different versions of a file? Something like:   * story.txt * story-joe-edit.txt * story-joe-edit-reviewed.txt   Branches accomplish similar goals in GitHub repositories.  Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our main (production) branch. When a change is ready, they merge their branch into main.  To create a new branch   1. Go to your new repository hello-world. 2. Click the drop down at the top of the file list that says **branch: main**. 3. Type a branch name, readme-edits, into the new branch text box. 4. Select the blue **Create branch** box or hit “Enter” on your keyboard. |

Er wordt inderdaad een nieuwe branch aangemaakt, deze heeft precies dezelefde inhoud als de main-branch. Het is verder makkelijk te zien welke inhoud er staat opgeslagen op iederen branch omdat je razend snel kan switchten tussen branches.

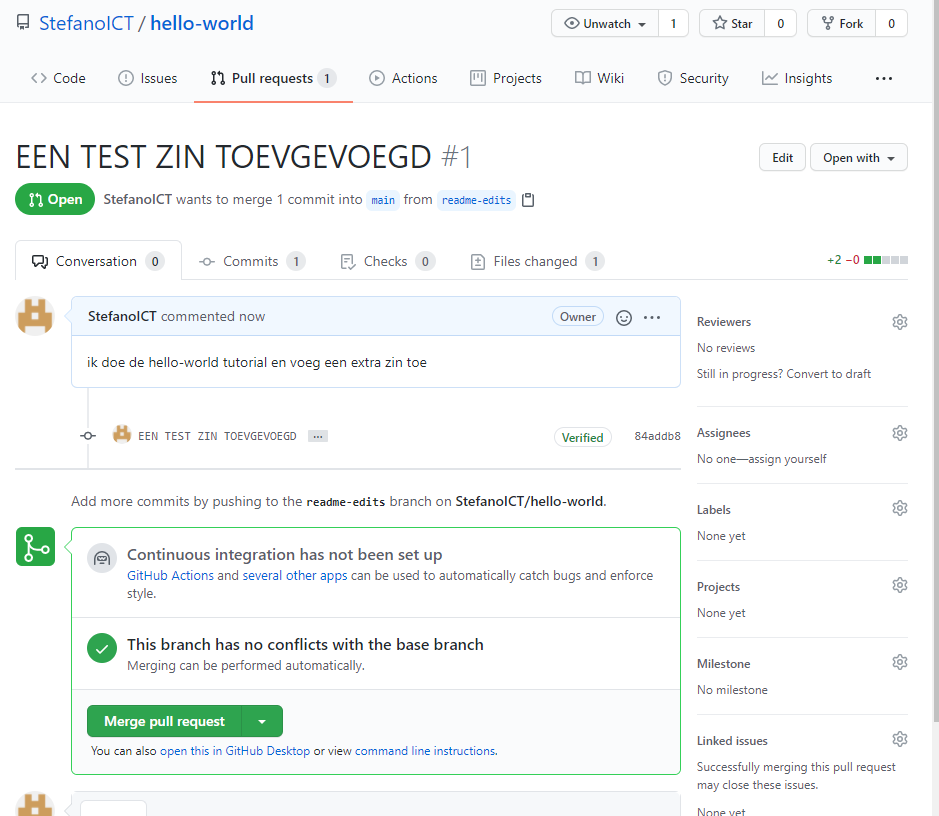
## Maak en voer veranderingen door

|  |
| --- |
| Bravo! Now, you’re on the code view for your readme-edits branch, which is a copy of main. Let’s make some edits.  On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you’ve done and why. Make and commit changes  1. Click the README.md file. 2. Click the  pencil icon in the upper right corner of the file view to edit. 3. In the editor, write a bit about yourself. 4. Write a commit message that describes your changes. 5. Click **Commit changes** button.   commit  These changes will be made to just the README file on your readme-edits branch, so now this branch contains content that’s different from main. |

**Opgeslagen veranderingen** heten “commits”. Na het doen van deze stappen wordt bij deze commit inderdaad alleen deze branch aangepast, maar niet de hoofd branch.

## Open een pull request

|  |
| --- |
| Nice edits! Now that you have changes in a branch off of main, you can open a pull request.  Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.  As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.  By using GitHub’s [@mention system](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.  You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub flow before working on larger projects.  Steps:   1. Click the Pull Request tab, then from the Pull Request page, click the green New pull request button. 2. In the Example Comparisons box, select the branch you made, readme-edits, to compare with main (the original). 3. Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit. 4. When you’re satisfied that these are the changes you want to submit, click the big green Create Pull Request button. 5. Give your pull request a title and write a brief description of your changes. 6. When you’re done with your message, click Create pull request! |



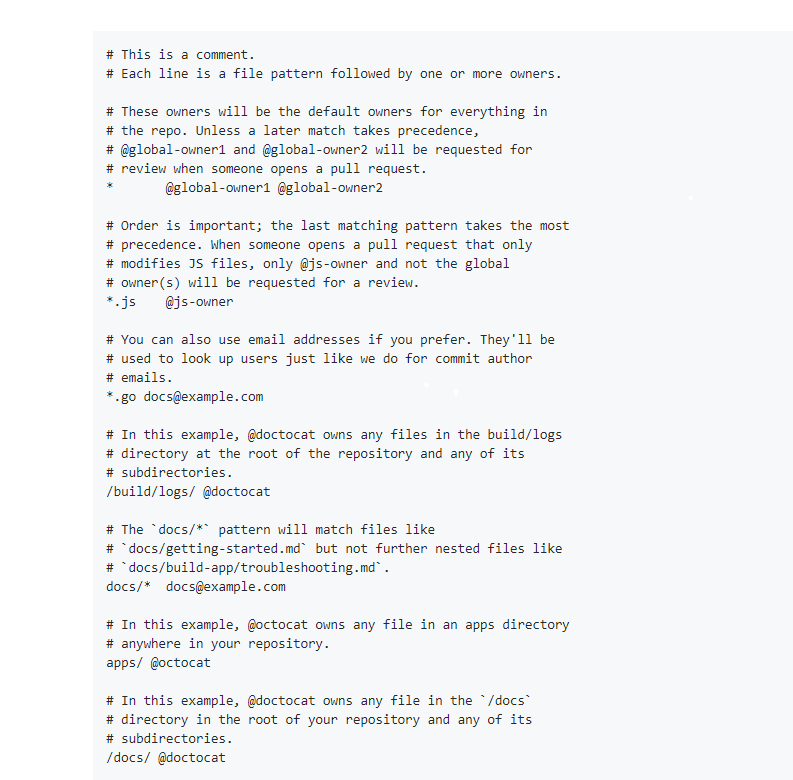
Van de commit uit de vorige stap is een pull request gemaakt. In de conversation tab is er een optie om meteen met de main brench te mergen. Je zegt met een pull request tegen GitHub “hey dit stukje is klaar, er moeten alleen nog dingen besproken worden/ kijken of het samenvoegen goed gaat)

## Merge de pull request

|  |
| --- |
| In this final step, it’s time to bring your changes together – merging your readme-edits branch into the main branch.   1. Click the green **Merge pull request** button to merge the changes into main. 2. Click **Confirm merge**. 3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.   merge delete |

Als je een pullrequest wi samenvoegen met de hoofdbranch vraagt hij je keuze te confomeren, het mooie is dat je de branch waar je de commit op gemaakt hebt meteen hierna kunt verwijderen (omdat deze feature nu in de mainbranch staat is deze readme-edits branch overbodig)

## CODEOWNERS file



Markdown (<https://guides.github.com/features/mastering-markdown/>)

