



hogeschool  
Leiden

13-1-2017

# Gebruikershandleiding ontwikkelstraat

Groep 4

**Auteurs:**

Dennis van Bohemen  
Roy Touw

**Module:**

IpsenH

**Opleiding:**

Klas: INF3B  
Opleiding: Informatica  
School: Hogeschool Leiden

**Versie:**

Versie 1.1 Final

Versie	Datum	Omschrijving	Opgeslagen door:
0.1 Ontwikkeld	10-1-2017	Initiële versie	Dennis van Bohemen
0.2 Ontwikkeld	10-1-2017	H1, H2, H3, H4 Deels bijgevoegd	Dennis van Bohemen
0.3 Final	13-01-2017	Jenkins stukje uitgebreid.	Roy Touw
1.1 Final	13-1-2017	Laatste wijzigingen	Dennis van Bohemen

# INHOUDSOPGAVE

1	inleiding.....	1
2	Stap 1: Voor ontwikkeling .....	2
3	Stap 2: Tijdens ontwikkeling .....	2
3.1	Back-end.....	2
3.2	Front-end.....	2
4	Stap 3: Na Ontwikkeling .....	2
4.1	Back-end.....	2
4.2	Front-end.....	3
5	Stap 4: Tools na pushen van code.....	3
5.1	Jenkins .....	3
5.2	SonarQube .....	9

# 1 INLEIDING

Dit document zal een leidraad zijn voor de ontwikkelaar voor het gebruik van de ontwikkelstraat. We beschrijven hoe en welke stappen de ontwikkelaar moet doorlopen om een nieuwe ontwikkelde feature of bug fix door de ontwikkelstraat te doorlopen. Dit document is geschikt voor de ontwikkelaar die gebruik zal maken van de vooraf geconfigureerde en ingericht ontwikkelstraat waarbij geen noemenswaardige aanpassingen meer vereist zijn van de ontwikkelstraat.

## 2 STAP 1: VOOR ONTWIKKELING

Voor de ontwikkeling dient de juiste repositories gecloned te worden door het gebruik van GIT. Dit kan gerealiseerd worden door het commando "GIT clone [https://fortytwowindmills.visualstudio.com/DefaultCollection/\\_git/TravelPlanner4](https://fortytwowindmills.visualstudio.com/DefaultCollection/_git/TravelPlanner4)" uit te voeren in de GIT Bash. Dit dient te worden gedaan in de gewenste folder van de ontwikkelaar. Dit proces dient ook herhaald te worden met de URL <https://github.com/Phillywonka/travel-planner.git>. Na deze commando's is het project klaar voor aanpassingen.

## 3 STAP 2: TIJDENS ONTWIKKELING

Tijdens de ontwikkeling maakt de ontwikkelaar gebruik van Visual Studio. Visual moet de beschikking hebben over de Nunit tools die voorgeschreven zijn in het Ontwikkelstraat Ontwerp document. Hiermee kan de ontwikkelaar nieuwe features ontwikkelen en vervolgens testen.

### 3.1 Back-end

Wanneer een ontwikkeling voltooid is of een bug gefixt is zal de ontwikkelaar de ontwikkelde Unit tests moeten realiseren in NUnit. Na de ontwikkeling van de tests worden deze in ieder geval eenmalig uitgevoerd in Visual Studio om zo fouten vroegtijdig op te sporen, te analyseren en op te lossen.

### 3.2 Front-end

Wanneer een ontwikkeling voltooid is of een bug gefixt is zal de ontwikkelaar de ontwikkelde Unit tests moeten realiseren in Jasmine. Na de ontwikkeling van de tests worden deze in ieder geval eenmalig uitgevoerd in de Angular CLI om zo fouten vroegtijdig op te sporen, te analyseren en op te lossen. De tests worden uitgevoerd door het commando ng-test uit te voeren.

## 4 STAP 3: NA ONTWIKKELING

### 4.1 Back-end

Wanneer een nieuwe feature is ontwikkeld door de ontwikkelaar van de back-end zal deze gecommited en gepushed moeten worden naar de TFS repository van de 42Windmills door middel van Git. Dit gebeurt door de onderstaande stappen te volgen.

1. Schrijf een commit message waarbij de opgeloste issues genoemd worden en het verrichte werk op een volgende regel. Voeg als laatste regel het Issuenummer toe.

2. Push de nieuwe feature of bug fix naar een branche met de naam als betreffende werkzaamheden.
3. Wanneer de einde van de dag is genaderd wordt de branche waarin de werkzaamheden zijn verricht gemerged met de develop branche in de TFS van de 42Windmills. Hier wordt tevens het issuenummer toegevoegd
4. Voeg wanneer nodig een issuenummer toe aan de issuelijst samen met een beschrijving van een nieuwe issue
5. Schrijf in de issuelijst de oplossing bij een eventueel opgeloste issue

## 4.2 Front-end

Wanneer een nieuwe feature is ontwikkeld door de ontwikkelaar van de front-end zal deze gecommitt en gepushed moeten worden naar de Github repository door middel van Git. Dit gebeurt door de onderstaande stappen te volgen.

1. Schrijf een commit message waarbij de opgeloste issues genoemd worden en het verrichte werk op een volgende regel. Voeg als laatste regel het Issuenummer toe.
2. Push de nieuwe feature of bug fix naar de develop branche.
3. Wanneer de einde van de dag is genaderd wordt de develop branche gemerged met de master branche in Github. Hier wordt tevens het issuenummer toegevoegd
4. Voeg wanneer nodig een issuenummer toe aan de issuelijst samen met een beschrijving van een nieuwe issue

# 5 STAP 4: TOOLS NA PUSHEN VAN CODE

## 5.1 Jenkins

Jenkins biedt een server voor Continious Integration projecten. Wij maken hier gebruik van door diverse plug-ins bij te voegen. In dit hoofdstuk leggen we uit hoe het gebruik van Jenkins plaats vindt.

### 5.1.1.1 Het dashboard

Dit is het algemene overzicht van de verschillende projecten die gebuild moeten worden. Vanuit hier kunnen de projecten aangeklikt worden voor meer informatie betreft het aangeklikte project. We zien hier in de bolletjes of de projecten succesvol uitgevoerd zijn (blauw) of niet succesvol uitgevoerd zijn (rood). Daarnaast staat er informatie vermeld over de hoeveelheid succesvolle builds onder het kopje W. Dit geeft een snel overzicht hoe de laatste tijd verlopen is. Hierbij is zonnig veel succesvolle builds

en storm is weinig succesvolle builds de afgelopen tijd. Ook kan vanuit hier in het meest rechter rondje met play knop het project apart gebuild worden

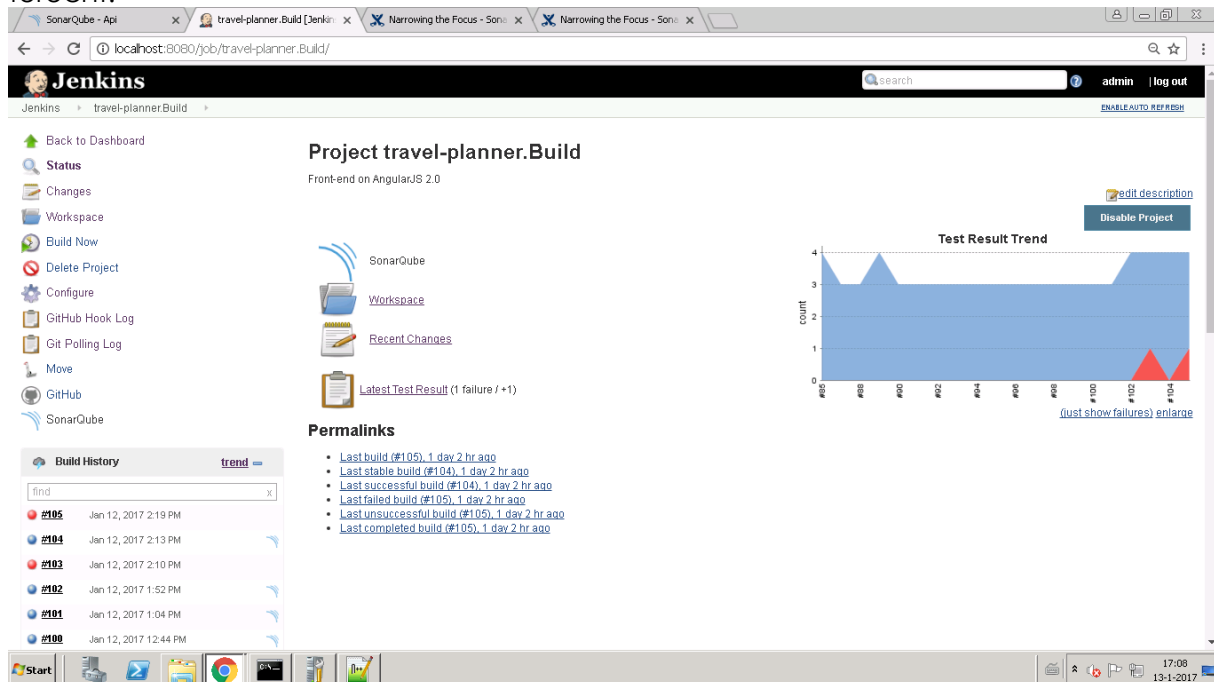
The screenshot shows the Jenkins web interface at localhost:8080. The left sidebar contains navigation links: New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The main content area displays a table of build history for the 'travel-planner' project. The table has columns for Status (S), Weather icon (W), Name, Last Success, Last Failure, and Last Duration. Three builds are listed: 'travel-planner-back-end\_Build', 'travel-planner\_Build', and 'Travelplanner-back-end-V2\_Build'. Below the table, there are links for 'Icon: S M L' and 'Legend' with RSS feeds for all, failures, and latest builds. The bottom of the page shows the generation date and version information.

S	W	Name	Last Success	Last Failure	Last Duration
		<a href="#">travel-planner-back-end_Build</a>	5 days 2 hr - <a href="#">#159</a>	16 days - <a href="#">#140</a>	1 min 11 sec
		<a href="#">travel-planner_Build</a>	4 days 17 hr - <a href="#">#81</a>	6 days 0 hr - <a href="#">#87</a>	47 sec
		<a href="#">Travelplanner-back-end-V2_Build</a>	N/A	1 day 22 hr - <a href="#">#242</a>	1 min 43 sec

Page generated: Jan 8, 2017 3:03:11 PM CET [REST API](#) [Jenkins ver 2.19.4](#)

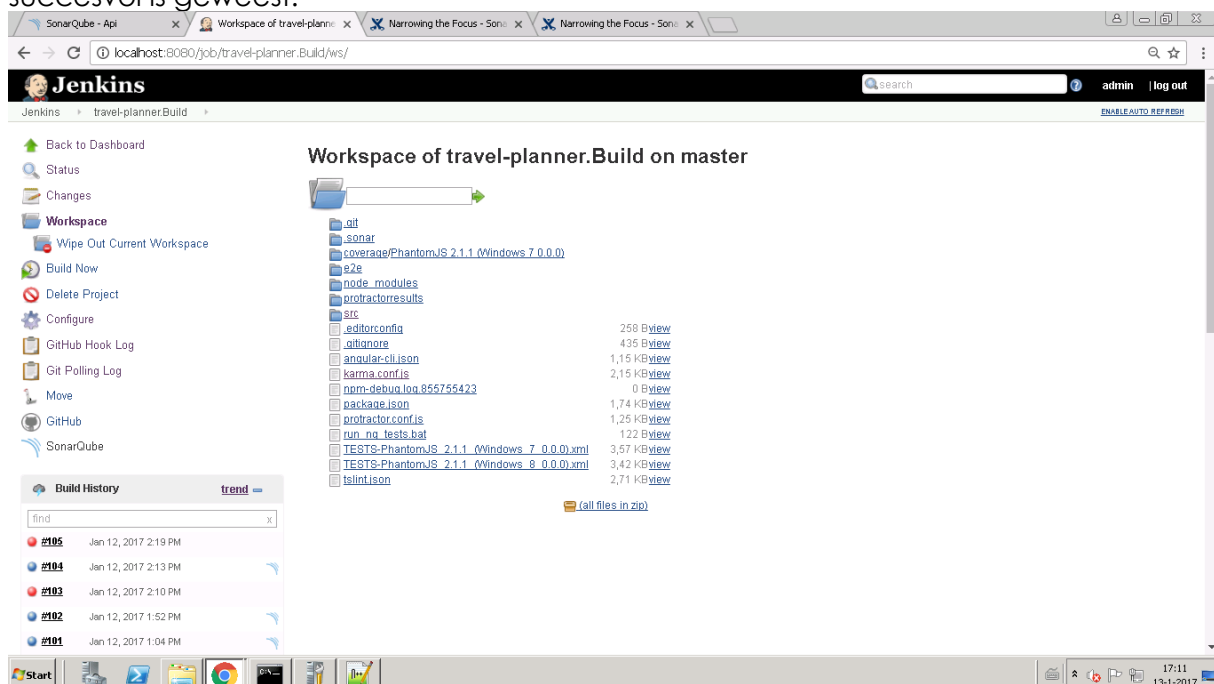
### 5.1.1.2 Project

Als er vervolgens een project aangeklikt wordt kom je op het onderstaande scherm terecht:



Rechts in beeld zie je een grafiek met de hoeveelheid uitgevoerde unit tests in het blauw, en in het rood welke er gefaald zijn. In dit voorbeeld zijn er dus vier unit tests waarvan er een gefaald is in de build met nummer 105.

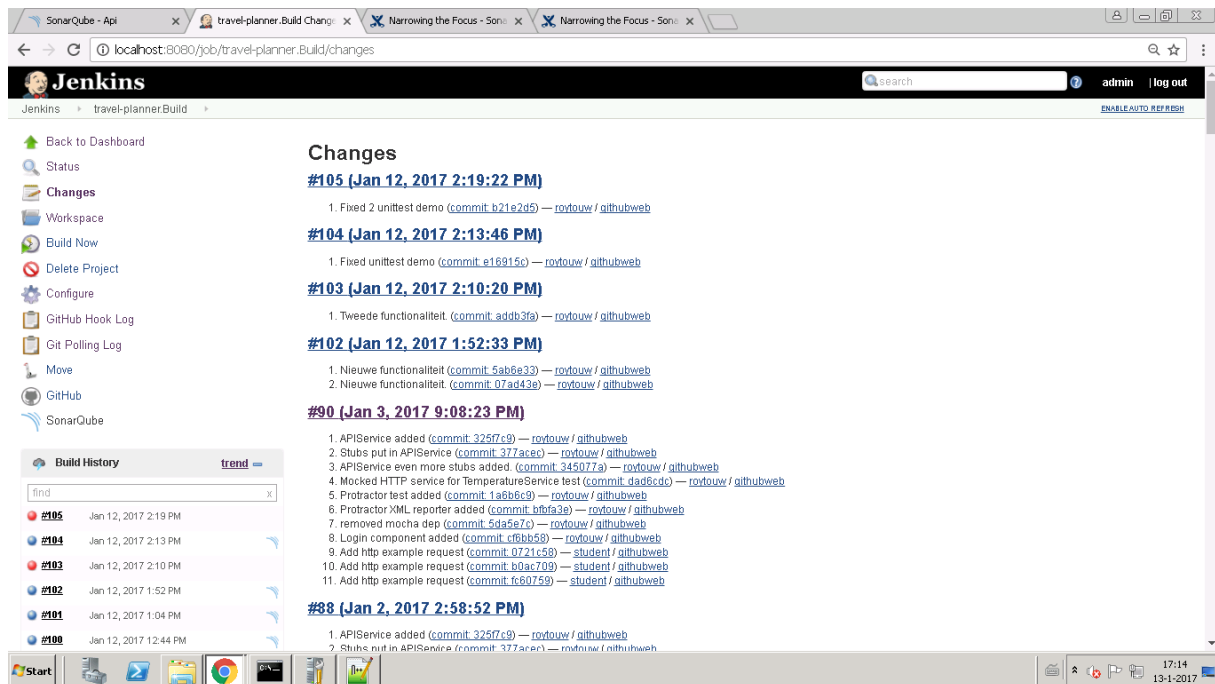
Links zie je vier hyperlinks om naar SonarQube, de Workspace, de laatste wijzigingen en de laatste resultaten van de tests te gaan. Hieronder zie je ook nog de laatste builds staan. Als je op de workspace link klikt krijg je het onderstaande scherm te zien: Helemaal links onder zie je de lijst met builds staan met daarbij een datum en of deze succesvol is geweest.



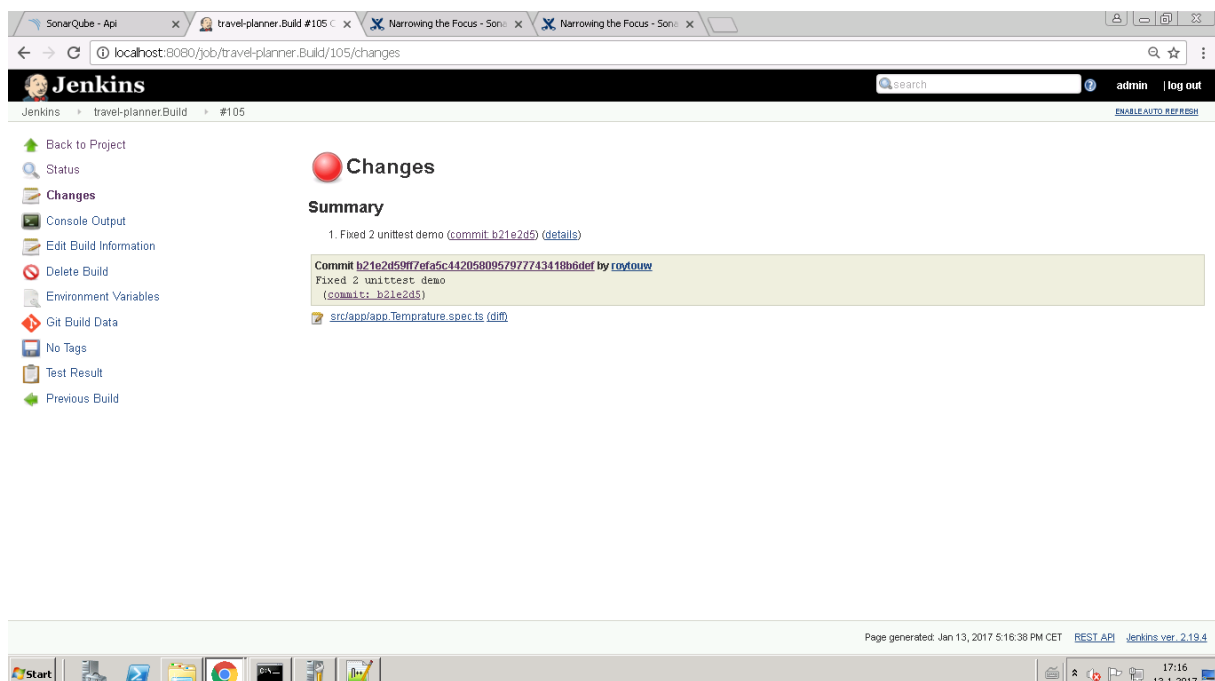


Hierin staan alle files van de desbetreffende build van een geselecteerd project. Als je op een bestand klikt download je dit bestand.

Als we in het voorgaande scherm op de recent changes hyperlink klikt kom je op een scherm terecht met de laatste wijzigingen, dus een lijst met de commits met diens commit message erbij; zoals te zien in onderstaande afbeelding:



Net zoals in Github kan je op deze commits klikken om te zien wat er gewijzigd is en of de build nog succesvol is na de desbetreffende commit, te zien aan de rode bol in onderstaande afbeelding welke blauw is bij een succesvolle build.



Na deze commit is de build dus niet succesvol.

Het laatste onbehandelde scherm is de results, de resultaten na een build, zoals te zien valt in onderstaande afbeelding:

The screenshot shows the Jenkins Test Results page for build #105. The page has a sidebar on the left with links to Back to Project, Status, Changes, Console Output, Edit Build Information, History, Environment Variables, Git Build Data, No Tags, Test Result, and Previous Build. The main content area is titled 'Test Result' and shows '1 failures (+1)'. A progress bar indicates 4 tests (±0) with a duration of 0.78 sec. Below this, there is a section for 'All Failed Tests' with a table showing the test name, duration, and age. The test name is 'PhantomJS 2.1.1 (Windows 7.0.0.0) TemperatureService.TemperatureService Demo purpose unit test' with a duration of 20 ms and an age of 1. Below this, there is a section for 'All Tests' with a table showing the package, duration, fail, skip, pass, total, and dim. The package is 'PhantomJS 2.1.1 (Windows 7.0.0.0)' with a duration of 0.78 sec, 1 fail, 1 skip, 0 pass, 3 total, and 4 dim.

Op dit scherm is te zien dat er een test gefaald is, zijnde de TemperatureService Demo Purpose unit test. Als hier vervolgens op geklikt wordt krijg je de stacktrace te zien.

The screenshot shows the Jenkins Regression page for build #105. The page has a sidebar on the left with links to Back to Project, Status, Changes, Console Output, Edit Build Information, History, Environment Variables, Git Build Data, No Tags, Test Result, and Previous Build. The main content area is titled 'Regression' and shows 'PhantomJS 2.1.1 (Windows 7.0.0.0) TemperatureService.TemperatureService Demo purpose unit test. (from PhantomJS 2.1.1 (Windows 7.0.0.0))'. The test failed with 1 failure and 4 tests. The duration was 0.78 sec. Below this, there is a section for 'Stacktrace' showing the expected false to equal true. The stacktrace includes the following lines:   
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
invoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
onInvoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
invoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
run@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
src/test.ts:9135:35  
invoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
onInvoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
onInvoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
invoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
run@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
src/test.ts:9135:35  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
invokeTask@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
runTask@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
invoke@webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
webpack: /C:/Program Files (x86)/Jenkins/workspace/travel-planner.Build/src/app/Temperature.spec.ts:76:26 <- src/test.ts:48747:30  
Standard Output

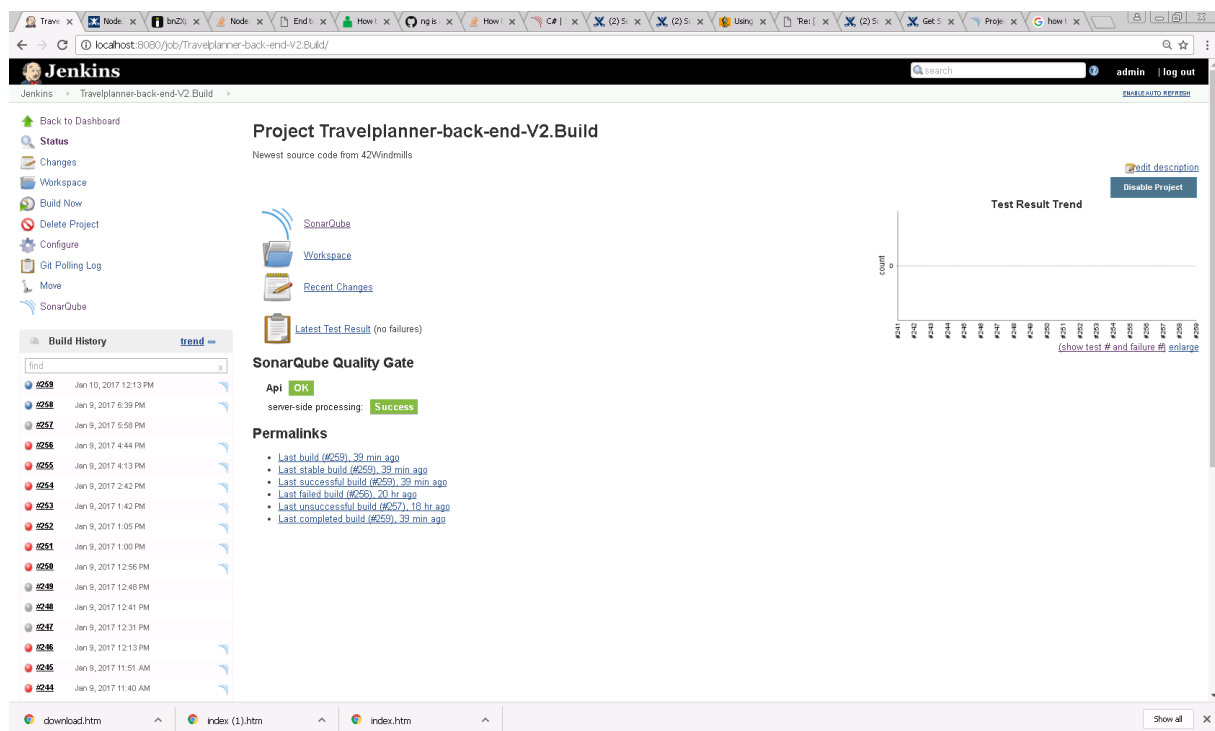
Ook kan je hier een beschrijving toevoegen door op add description te klikken, dit is straight forward dus laat ik dit buiten de screenshots.

## 5.2 SonarQube

SonarQube verzorgt analyses van code om de code kwaliteit hoog te houden. Deze analyses worden volledig geautomatiseerd uitgevoerd wanneer er een Jenkins build heeft plaatsgevonden. In dit hoofdstuk beschreven we de noodzakelijke gebruikshandelingen.

Wanneer een build heeft plaatsgevonden kan via het Jenkins project overzicht genavigeerd worden naar de webpagina van SonarQube. In dit overzicht kan vervolgens de analyse die zojuist uitgevoerd is geraadpleegd worden.

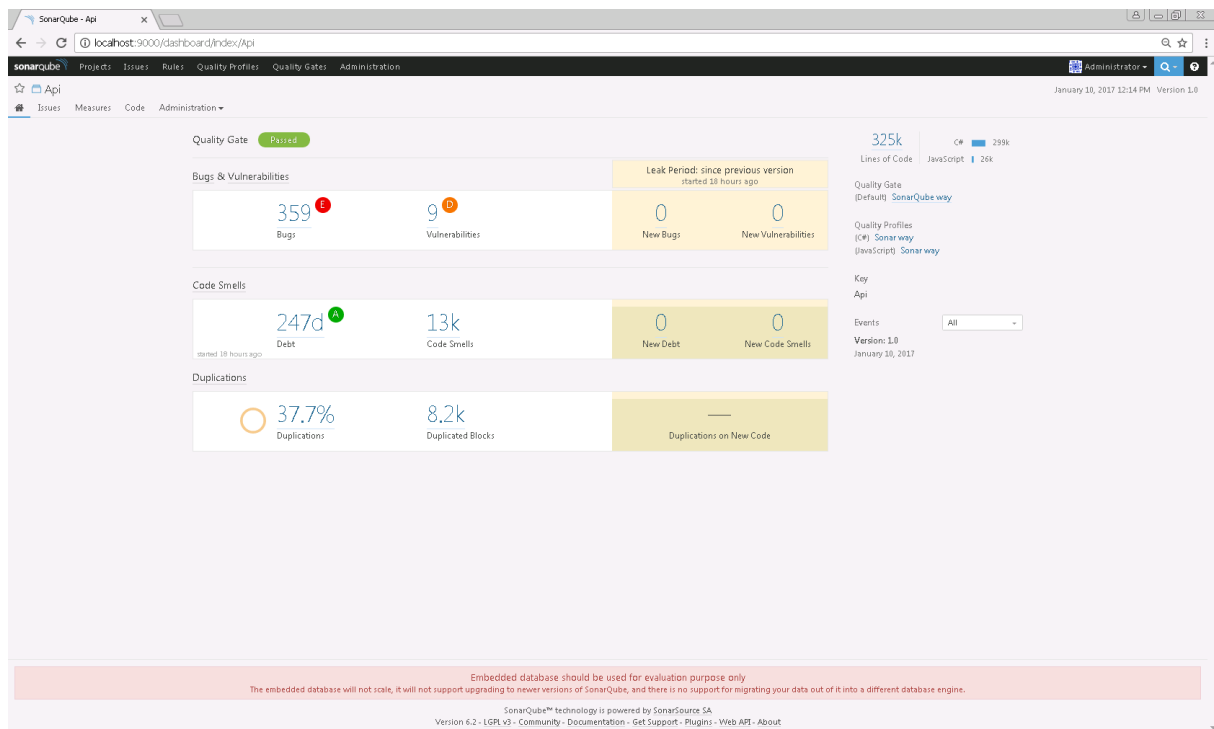
Dit overzicht wordt geopend door op de SonarQube link te klikken, dit opent het dashboard van SonarQube samen met de analyse van het huidige project.



De SonarQube link is aangegeven met de 3 blauwe bogen in het Jenkins overzicht.

Wanneer hier op wordt geklikt wordt het volgende scherm geopend. We zien in het bovenstaande overzicht daarnaast dat het Api project een OK status heeft wat betreft de code kwaliteit en daarmee ook de "quality gate" op "passed" staat.

Het onderstaande dashboard overzicht wordt geopend bij het openen van SonarQube.

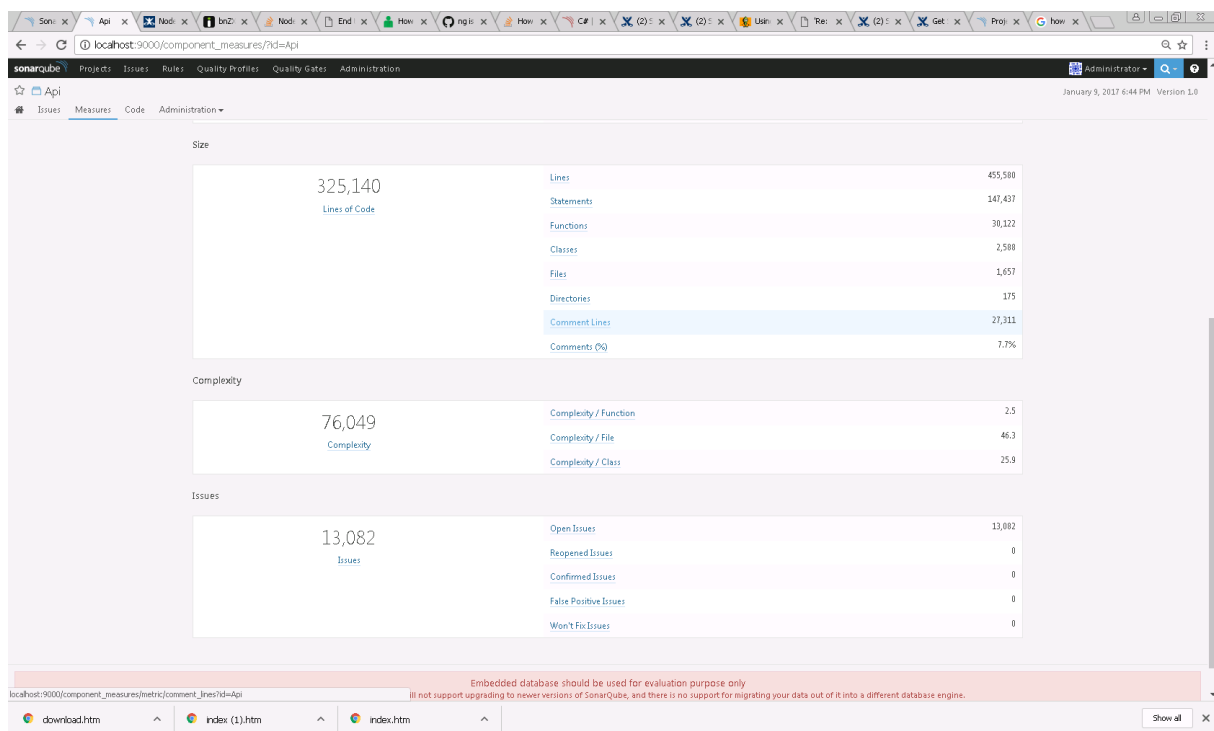
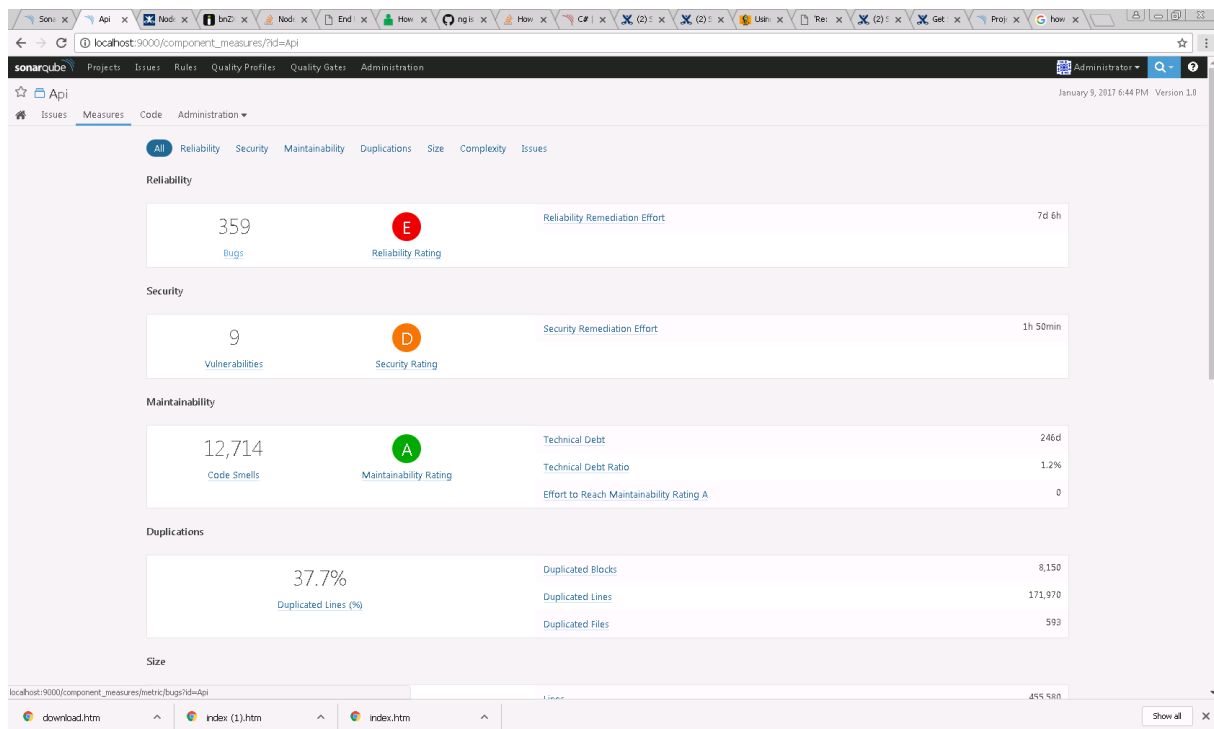


We zien hier een aantal statistieken die de analyses heeft waargenomen. De statistieken worden hier kort behandeld.

### Statistieken

- Bugs & Vulnerabilities: Bugs en mogelijke beveiliging lekken in de code na een statische analyse. De beveiligingslek komt voort uit een stuk verkeerd gebruikte code.
- Code Smells: Style en formaat regels die overtreden worden in de code.
- Duplications: Code die elders in het programma het zelfde formaat heeft waardoor duplicaties van code ontstaan.

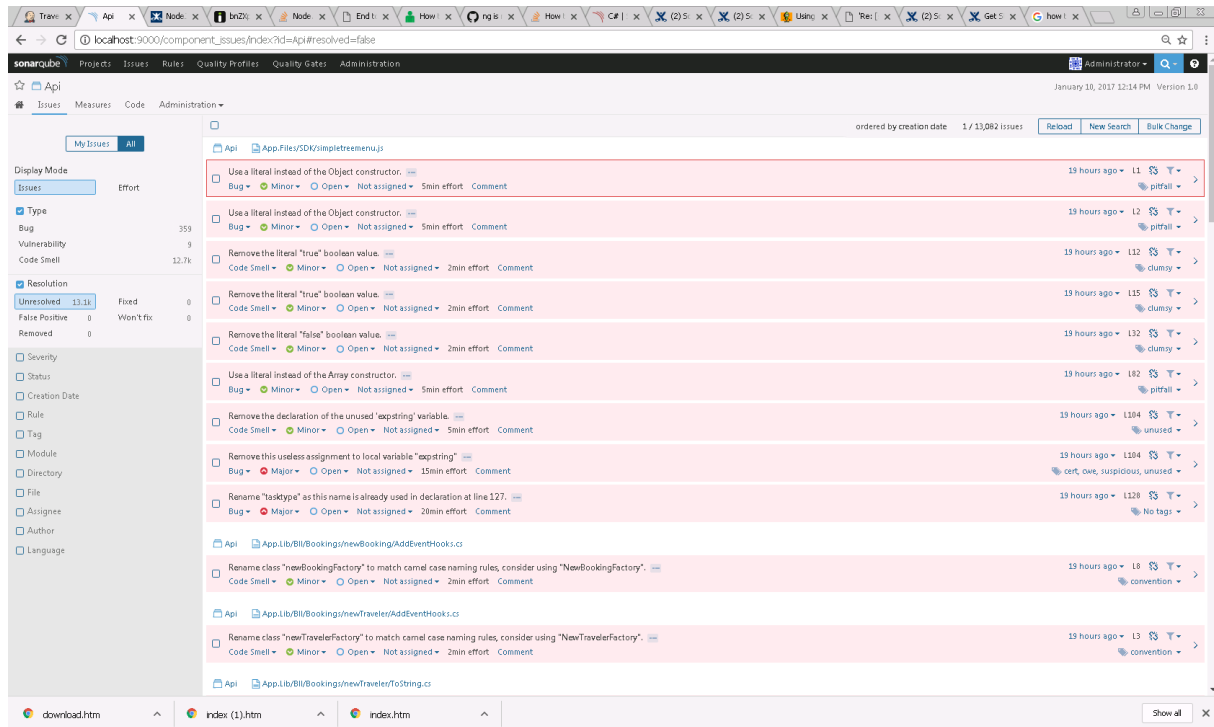
In een crème kleur staan naast de standaard statistieken de nieuw opgetreden statistieken. Deze worden onderschept wanneer er een nieuwe analyse wordt uitgevoerd en de uitkomst afwijkt van de vorige keer. Wanneer er nieuwe dus bijvoorbeeld nieuwe code smells regels overtreden worden kunnen deze direct worden onderschept. Wanneer we vervolgens op measures klikken belanden we in een uitgebreid overzicht van de waargenomen metingen



We kunnen in dit overzicht meer details vinden over de geanalyseerde code en de bijbehorende statistieken. Voor meer informatie over de getoonde statistieken verwijzen

we naar [www.sonarqube.org](http://www.sonarqube.org). Hier staat duidelijk beschreven wat elke statistiek betekend.

Als laatst behandelen we de issues lijst in SonarQube, welke hieronder is weergegeven.



Dit overzicht weergeeft een gedetailleerd beeld van de statistieken die zijn vrijgegeven. We zien hier exact wat de foutieve code heeft veroorzaakt en welke regel hiertoe behoort. De wijzigingen kunnen vervolgens worden doorgevoerd of worden overgeslagen in SonarQube. SonarQube houdt deze gegevens bij in de issueslijst en kan met behulp van dit overzicht de ontwikkelaar precies wijzen op de overtreden style en format regels.