



hogeschool  
Leiden

12-1-2017

# Ontwerp Ontwikkelstraat

Groep 4

**Auteur:**

Dennis van Bohemen

**Module:**

Ipsenh

**Opleiding:**

Klas: INF3B

Opleiding: Informatica

School: Hogeschool Leiden

Groep: 4

**Versie:**

Versie 2.3 Final

Versie	Datum	Omschrijving	Opgeslagen door:
0.1 Ontwikkel	08-12-2016	Initiële versie	Dennis van Bohemen
0.2 Ontwikkel	01-1-2017	OTAP bijgevoegd	Dennis van Bohemen
0.3 Ontwikkel	05-1-2017	Eerste gedeelte Realisatie stappen bijgevoegd	Dennis van Bohemen
0.4 Ontwikkel	06-1-2017	Tweede gedeelte Realisatie stappen bijgevoegd	Dennis van Bohemen
1.1 Concept	08-1-2017	Tools gebruik en configuratie toegevoegd	Dennis van Bohemen
1.2 Concept	10-1-2017	Sonarqube toegevoegd	Dennis van Bohemen
1.3 Concept	10-1-2017	Issuetracking toegevoegd	Dennis van Bohemen
2.1 Pre-final	12-1-2017	Spelling controle en enkele aanvullingen	Dennis van Bohemen
2.2 Pre-final	13-01-2017	Toevoeging front-end gerelateerde onderwerpen.	Roy Touw
2.3	13-01-2017	Afronding document.	Roy Touw

# INHOUDSOPGAVE

1	Inleiding.....	2
2	Het huidige Proces .....	3
3	OTAP .....	4
3.1	Ontwikkel .....	4
3.1.1	IDE (Integrated Development Environment) .....	4
3.1.2	Versiebeheersysteem .....	5
3.1.3	Het proces.....	5
3.2	Test .....	6
3.2.1	Unittests frameworks .....	6
3.2.2	End-to-end testen .....	7
3.2.3	Code style en format checking .....	7
3.2.4	Continious Integration .....	7
3.2.5	Build automatisering .....	7
3.2.6	Het proces.....	8
3.3	Acceptatie .....	8
3.3.1	Het proces.....	9
3.4	Productie.....	9
3.4.1	Het proces.....	9
4	Een visueel overzicht van het gehele proces.....	10
4.1	Front-end.....	10
4.2	Back-end.....	11
5	Ontwerp beslissingen .....	12
5.1	Stappenplan realisatie .....	12
5.1.1	Stap 1: Opzetten GIT repositories .....	12
5.1.2	Stap 2: Opzetten Jenkins.....	12
5.1.3	Stap 3: Automatic building .....	17
5.1.4	Stap 4: Manual building .....	18
5.1.5	Stap 5: Automatic testing C# .....	18
5.1.6	Stap 6: Automatic testing in Angular 2 .....	19
5.1.7	Stap 7: Code quality checking .....	19
5.1.8	Stap 8: Issue tracking .....	21
6	Tools gebruik en configuratie.....	22
6.1	Jenkins .....	22
6.1.1	Gebruik.....	22

6.1.2	Configuratie .....	30
6.2	NAntContrib .....	36
6.2.1	Gebruik .....	36
6.2.2	Configuratie .....	38
6.3	NUnit .....	39
6.3.1	Gebruik .....	39
6.3.2	Configuratie .....	39
6.4	Git en Github .....	39
6.4.1	Gebruik .....	39
6.4.2	Configuratie .....	39
6.5	Git en de 42Windmills server .....	40
6.5.1	Gebruik .....	40
6.5.2	Configuratie .....	40
6.6	SonarQube .....	41
6.6.1	Gebruik .....	41
6.6.2	Configuratie .....	45
6.7	Issue tracking .....	48
6.7.1	Gebruik .....	48
7	Conclusie .....	48
8	Bronnen .....	49

# 1 INLEIDING

Voor de ontwikkeling van een API voor de 42Windmills is een ontwikkelstraat nodig om voor Continuous Delivery en Continuous Integration te zorgen. De gekozen tools zijn tot stand gekomen door het vooraf uitgevoerde onderzoek naar de toegevoegde waarde van een ontwikkelstraat en de voor ons bruikbare tools. Deze sluiten aan bij de verschillende programmeertalen en technieken die gebruikt worden voor de API. Dit betreft C# voor de back-end van de applicatie en Javascript in combinatie met het Angular 2 framework aan de front-end.

In dit document zal de ontwikkelstraat beschreven worden.

Eerst zal beschreven worden hoe de ontwikkelstraat volgens de OTAP principes is ingedeeld en welke tools hierbij zijn gekozen. Na het benoemen van de tools kunnen we meer inzoomen op de verschillende tools en deze beschrijven. Voor een uitgebreide argumentatie van de verschillende gekozen tools verwijzen we naar het onderzoeksrapport.

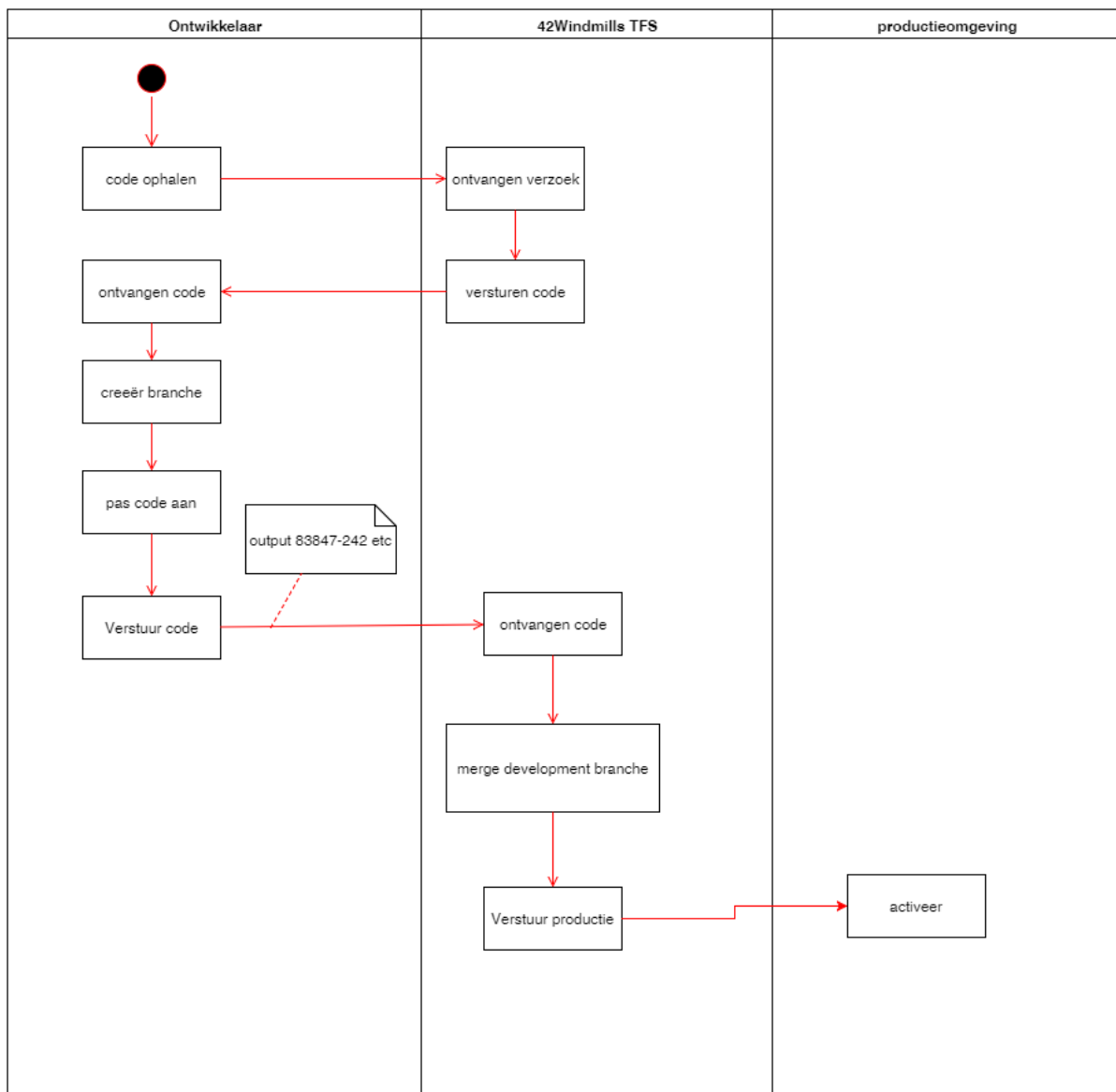
Vervolgens zal beschreven worden hoe deze ontwikkelstraat tot stand is gekomen en hoe de verschillende ontwerp uitdagingen zijn opgelost, hoe deze worden beschreven alsmede met de diverse ontwerp beslissingen.

## 2 HET HUIDIGE PROCES

Zonder het gebruik van een ontwikkelstraat zullen de volgende stappen doorlopen moeten worden om een nieuw versie toe te voegen aan de ontwikkelomgeving en productieomgeving.

Hierbij wordt door de ontwikkelaar gebruik gemaakt van een Integrated Development Environment genaamd Visual Studio. De 42Windmills hebben hiervoor gekozen omdat de programmeertaal C# betreft. Visual Studio heeft de beste ondersteuning voor C#.

Om de code gescheiden te houden en de verschillende versies goed te ordenen gebruikt de 42Windmills een Versioning Systeem genaamd GIT. Deze tool kopieert alle code naar de TFS server van de 42Windmills welke vervolgens de code in een productieomgeving zet. Op de volgende pagina wordt een visueel overzicht gegeven van het huidig proces van het in productie zetten van een nieuwe versie.



Omdat de API gebruik zal maken van de source code van de 42Windmills en we bij de ontwikkeling Continuous Integration en Delivery dienen toe te passen, hebben we de ontwikkelstraat zodanig moeten inrichten dat het proces van 42Windmills niet onderbroken wordt. In de volgende hoofdstukken zal duidelijk worden hoe dit vraagstuk beantwoord is.

## 3 OTAP

OTAP is een veel gebruikte methodiek binnen een ontwikkelstraat. De termen geven de fases aan die door een software product doorlopen worden. OTAP staat voor:

O: Ontwikkel  
T: Test  
A: Acceptatie  
P: Productie

Om onze ontwikkelstraat goed in kaart te brengen maken wij gebruik van deze methodiek om de ontworpen ontwikkelstraat uit te lichten met de daarbij behorende tools. Omdat bij het ontwikkelen van de front-end en back-end verschillen zitten in het proces van de ontwikkelstraat wordt met regelmaat onderscheid gemaakt tussen front-end en back-end in de beschrijving van het proces.

### 3.1 Ontwikkel

Een programma wordt eerst ontwikkeld door een team. Dit gebeurt in de ontwikkelomgeving waarin iedereen werkt aan één de zelfde versie. Deze versie wordt aan het einde van de dag op het versiebeheersysteem gezet op een ontwikkelserver.

Bij de ontwikkelstraat die door ons ontworpen is hebben we diverse tools in gebruik in de ontwikkelomgeving welke zich in de ontwikkelfase bevindt. Later in dit ontwerp zal meer ingezoomd worden op de verschillende tools.

#### 3.1.1 IDE (Integrated Development Environment)

Een Integrated Development Environment, afgekort een IDE genoemd, wordt gebruikt voor de ontwikkeling van de code. Deze IDE dient goed aan te sluiten bij de programmeertaal waarmee ontwikkeld wordt. Omdat de code die door onze ontwikkelstraat gehaald dient te worden C# betreft, hebben wij gekozen voor de IDE Visual Studio 2015. De IDE is als het ware een tekst editor met een hoop extra functionaliteiten aan de tekst editor toegevoegd. Bijvoorbeeld het herkennen van de syntax fouten in een stuk code of de toevoeging van een debugger om fouten gemakkelijk op te sporen.

### 3.1.2 Versiebeheersysteem

Om de verschillende versies gescheiden te houden van elkaar maken we gebruik van het GIT systeem. Hiervoor is gekozen omdat dit goed aansluit bij het huidige proces van 42Windmills. Ze gebruiken bij het uitbrengen van een nieuwe versie van het programma ook het Versioning Systeem GIT.

### 3.1.3 Het proces

Een ontwikkelaar voegt een stukje functionaliteit toe aan het programma en controleert of deze code correct is. Wanneer dit volgens hem het geval is zal hij vervolgen beginnen met het schrijven van testen in de daarvoor bestemde tools. Na het schrijven van de testen worden de nieuwe issues en opgeloste issues vastgelegd in het issue document.

#### 3.1.3.1 Back-end

Om de unittests te schrijven binnen de back-end wordt gebruik gemaakt van het Nunit framework. Deze tests worden geschreven door de verschillende paden door te lopen die de toegevoegde functionaliteit doorloopt. Hierbij wordt alleen voor de toegevoegde code tests toegevoegd en dus niet voor de al bestaande code. De tests worden aan de test folder van het project bijgevoegd.

Wanneer de tests zijn toegevoegd wordt de ontwikkelde code en de tests naar de TFS server van de 42Windmills gepushed door middel van het GIT systeem. Hierbij kan Visual Studio de GIT commands uitvoeren.

#### 3.1.3.2 Front-end

Binnen de front-end wordt er gebruik gemaakt van de tool Jasmine om unittests te schrijven. Ook hier geldt dat de tests de verschillende paden moet doorlopen die de nieuwe functionaliteit doorloopt binnen de zelf ontwikkelde code. Voor toegevoegde functionaliteit die door de gebruiker uitgevoerd wordt, dient ook een end to end test aangemaakt te worden. Wanneer de tests gereed zijn kan het Versioning Systeem gebruikt worden om de nieuwe versie te pushen. In tegenstelling tot de back-end, waarbij de code naar de TFS server van de 42Windmills gepushed wordt, wordt de code van de front-end gepushed naar een publiek toegankelijke GITHUB server. Zo blijft de ontwikkelde front-end gescheiden van de back-end.

Nu de verschillende stukken codes zijn gekopieerd naar de Git repositories, kan de Continious Integration plaatsvinden. Dit zal beschreven worden in het hoofdstuk "Test"



## 3.2 Test

Wanneer de nieuwste versie van de front-end en de back-end op de GIT repository is geplaatst komt de Continuous Integration tool Jenkins in actie. De code wordt om vooraf ingestelde tijden opgehaald vanaf de verschillende GIT repositories door de testserver. Dit geldt voor zowel de front-end als de back-end code. In deze "testomgeving" kan vervolgens geautomatiseerd de verschillende tests uitgevoerd worden.

### 3.2.1 Unittests frameworks

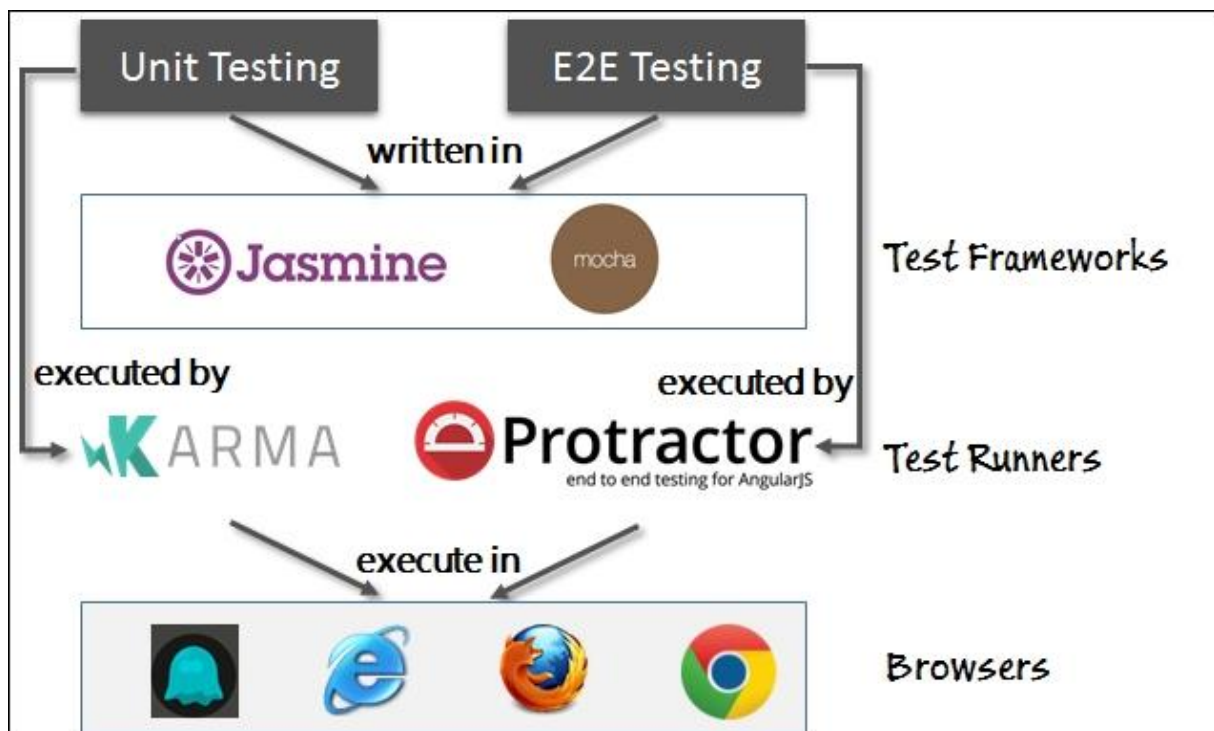
Om de mogelijkheid te hebben om unittests uit te voeren moet het programma de beschikking hebben over een bepaald unittest framework. Deze moest geautomatiseerd uitgevoerd kunnen worden door de Continuous Integration tool.

#### 3.2.1.1 Back-end

De back-end van het programma maakt gebruik van het Nunit framework. Dit framework is ontwikkeld voor het schrijven van unittests voor de programmeertaal C#.

#### 3.2.1.2 Front-end

In het front-end deel van de applicatie worden de tests geschreven in typescript en het Jasmine framework. De tests worden uitgevoerd met de tool Karma, deze kan de tests een of meerdere malen uitvoeren en met gebruik van verschillende reporters de resultaten van de tests opslaan in een xml bestand waar de ontwikkelstraat mee over weg kan. De tests worden uitgevoerd in een webbrowser, in dit geval eentje zonder een grafische interface genaam PhantomJs. In onderstaande afbeelding is de relatie tussen de frameworks, de tools en de browsers goed te zien voor zowel de unit tests als de e2e tests.



### 3.2.2 End-to-end testen

Om een klik op het scherm na te bootsen kunnen end-to-end testen gebruikt worden om te deze handeling te testen. Dit wordt alleen ingesteld aan de front-end omdat hier het begin van de test (de klik) zal plaatsvinden.

#### 3.2.2.1 Front-end

De e2e tests worden evenals de unit tests van het front-end deel van de applicatie geschreven in typescript en met behulp van het Jasmine framework. De tool die de tests vervolgens uitvoert is Protractor, en deze voert de tests uit in de browsers PhantomJs.

### 3.2.3 Code style en format checking

Om de code kwaliteit in het project hoog te houden zijn er diverse tools beschikbaar voor het controleren van deze kwaliteit. We hebben in dit project gekozen voor de tool Sonarqube. Door deze tool te gebruiken wordt er een webpagina gehost die naast de Jenkins pagina draait. Sonarqube voert vervolgens op commando van Jenkins verschillende code checks uit en weergeeft de resultaten in de Sonarqube webpagina. Deze tests kunnen voor zowel de front-end (Javascript) als de Back-end uitgevoerd worden.

### 3.2.4 Continious Integration

Om Continious Integration mogelijk te maken worden diverse tools gebruikt. We kijken hierbij naar een tool die verschillende commando's kan uitvoeren om tests automatisch uit te voeren bij de front-end en back-end en een tool die hierbij ook een inzicht kan geven van de rapportage die gemaakt wordt van de uitgevoerde tests. We hebben als tool hiervoor gekozen voor Jenkins. Jenkins biedt een complete CI beleving door gebruik te maken van verschillende dashboards en verschillende downloadbare plug-ins die gebruikt kunnen worden in combinatie met de testtools.

### 3.2.5 Build automatisering

Om de programmacode om te zetten in een uitvoerbaar programma om hier vervolgens geautomatiseerde tests uit te voeren moet het programma wel uitvoerbaar zijn. Dit wordt gedaan door een Nant script uit te voeren. Dit is een speciaal soort Nant script genaamd NantContrib. Deze tool biedt de mogelijkheid om met een eenvoudig script MSBUILD op te starten en te configureren waardoor de programmacode wordt omgezet in een uitvoerbaar programma. De build automatisering geldt alleen voor de back-end van het programma.

## 3.2.6 Het proces

### 3.2.6.1 Back-end

De CI tool Jenkins zal om een vooraf ingestelde tijd de back-end van de TFS server van de 42Windmills halen. Wanneer deze code is opgehaald wordt het NAnt script uitgevoerd door Jenkins zodat de zojuist opgehaalde code gebuild wordt en de programmacode een programma is geworden welke uitvoerbaar is.

De tests kunnen vervolgens door Jenkins worden aangeroepen. Dit gebeurt door een commando uit te voeren die Nunit aanroept om de Nunit tests uit te voeren. Nunit zal vervolgens een rapport genereren in XML formaat van de uitgevoerde tests.

Nadat de tests zijn uitgevoerd kan er worden gecontroleerd op de code style en de code format. Deze tests worden volledig geautomatiseerd aangeroepen door Jenkins en uitgevoerd door Sonarqube. Nadat de analyse is verricht kunnen de rapportages van de tests worden ingezien door de Sonarqube webpagina te bezoeken. Deze pagina is bereikbaar via het Jenkins platform.

### 3.2.6.2 Front-end

De CI tool Jenkins zal registreren wanneer er een verandering heeft plaatsgevonden in GITHUB. Wanneer er een verandering heeft plaatsgevonden wordt de nieuwste versie door middel van het GIT systeem opgehaald en op de test server geplaatst. Wanneer de code is opgehaald wordt het commando ng-test uitgevoerd. Dit commando wordt aangeroepen in de tool NodeJS welke wordt gebruikt in combinatie met het Angular framework. Nodejs zal vervolgens de tool KARMA aanroepen om alle Jasmine tests uit te voeren en hierna PhantomJS aanroepen om een rapport te generen in XML formaat.

Nu alle rapporten zijn gegenereerd kunnen deze worden uitgelezen. Jenkins zal dit doen door de gegenereerde XML bestanden op te halen en te tonen in een Dashboard van Jenkins. Dit overzicht weergeeft hoeveel testen en welke testen goed en fout zijn gegaan.

Naast de getoonde rapporten toont Jenkins ook een overzicht van de verschillende builds en of de builds uitgevoerd konden worden en er dus geen bugs of compile time zijn gevonden.

Nadat de tests zijn uitgevoerd en de resultaten getoond zijn, kan er worden gecontroleerd op de code style en de code format. Deze tests worden volledig geautomatiseerd aangeroepen door Jenkins en uitgevoerd door Sonarqube. Nadat de analyse is verricht kunnen de rapportages van de tests worden ingezien door de Sonarqube webpagina te bezoeken. Deze pagina is bereikbaar via het Jenkins platform.

## 3.3 Acceptatie

De acceptatie fase geeft de klant de mogelijkheid om te zelfstandig de software door te lopen. Daarnaast kunnen andere belanghebbende het programma doorlopen om te

zien hoe de releaseversie er uit komt te zien zonder dat de dagelijkse productie onderbroken wordt.

### 3.3.1 Het proces

De CI tool Jenkins heeft alle tests uitgevoerd omdat dit automatisch al heeft plaatsgevonden tijdens het vooraf ingestelde tijdstip. Hierdoor heeft ook het build proces plaatsgevonden en kan de belanghebbende het programma functioneel testen. De front-end kan gebruikt worden om doorheen te klikken die op zijn beurt de back-end aanspreekt om de gegevens uit de database te halen. Deze acceptatieomgeving bevindt zich op de zelfde server waar Jenkins zich bevindt en dus ook de Testomgeving zich bevindt. In Jenkins moeten de verschillende versies worden gedocumenteerd door de openstaande en afgesloten issuenummers te benoemen in de desbetreffende build. Daarnaast worden de toegevoegde features en overige werkzaamheden benoemd.

## 3.4 Productie

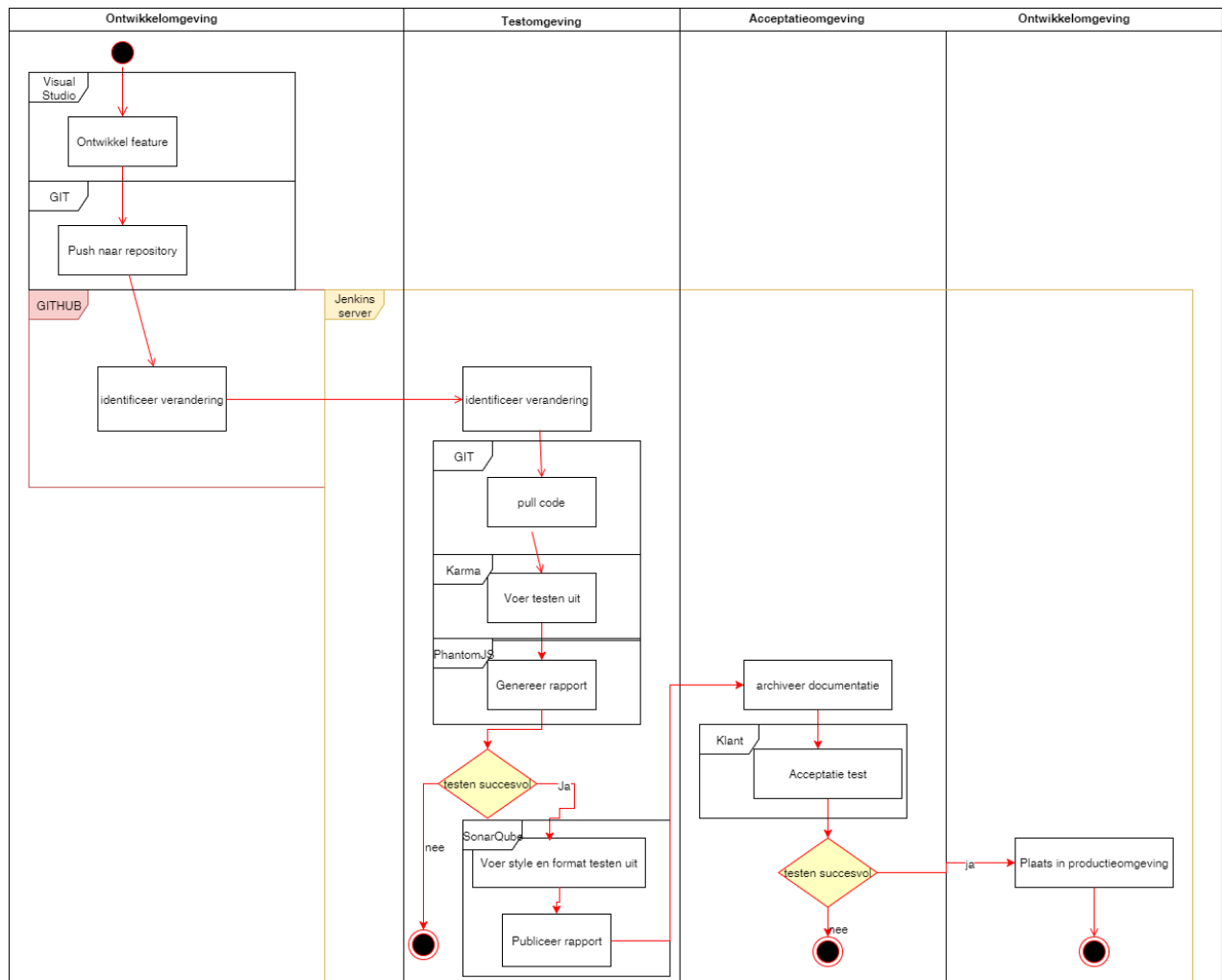
Wanneer de belanghebbende de acceptatie heeft geaccepteerd kan de release versie in productie worden gezet. Dit gebeurt door het programma in productieomgeving te zetten zoals dit gedocumenteerd is.

### 3.4.1 Het proces

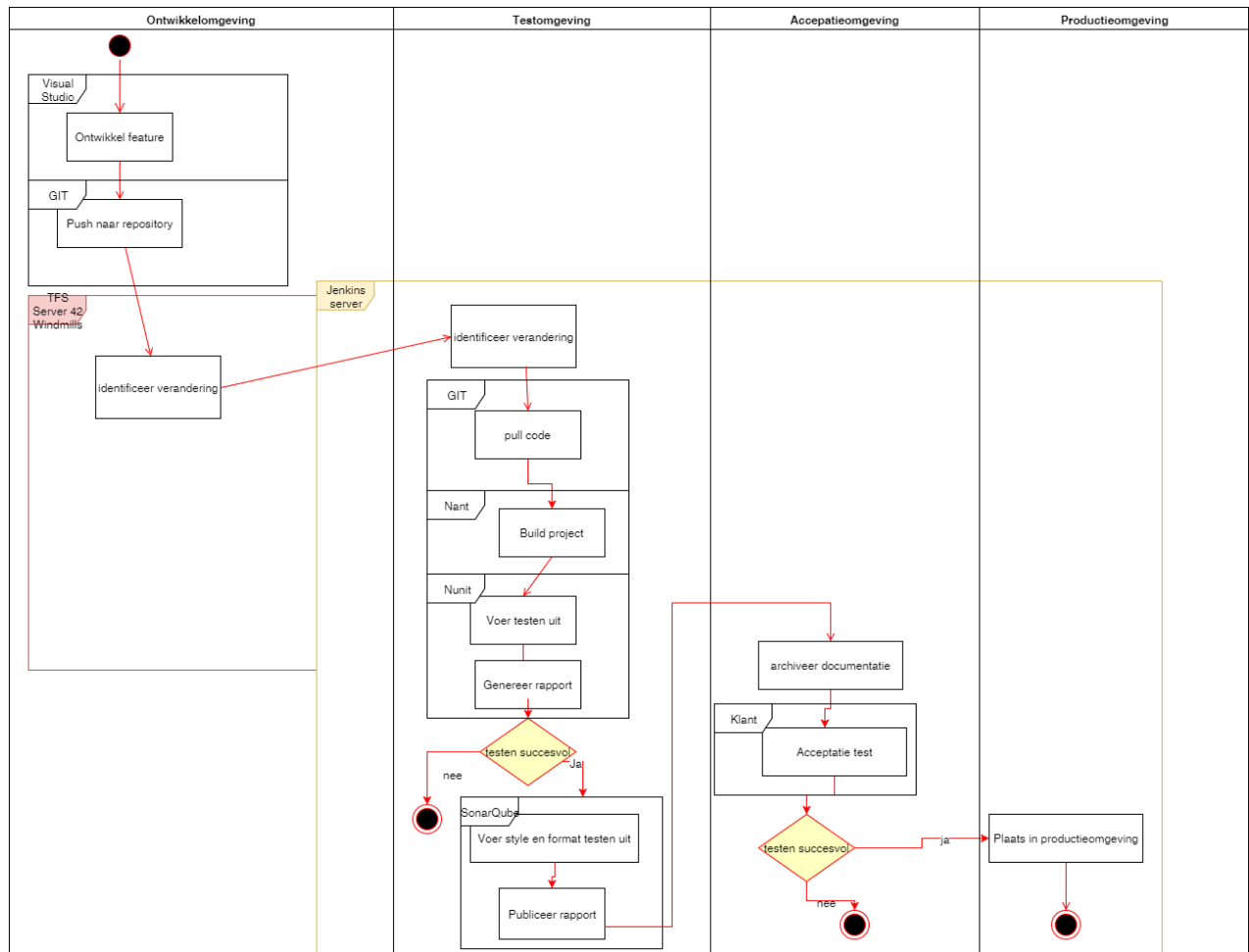
De belanghebbende kan deze release goedkeuren door de acceptatie omgeving doorlopen te hebben. Wanneer dit geaccepteerd is kan handmatig of geautomatiseerd Jenkins een commando uitvoeren om het programma naar de productieomgeving te zetten.

## 4 EEN VISUEEL OVERZICHT VAN HET GEHELE PROCES

### 4.1 Front-end



## 4.2 Back-end



## 5 ONTWERP BESLISSINGEN

### 5.1 Stappenplan realisatie

In dit hoofdstuk zullen we beschrijven hoe dit ontwerp tot stand is gekomen en hoe de realisatie van de ontwikkelstraat heeft plaatsgevonden. Naast de stappen beschrijven we de verschillende ontwerp beslissingen die genomen zijn tijdens de realisatie om zo een duidelijk beeld te geven van de verschillende uitdagingen die we zijn tegen gekomen tijdens het opzetten van de ontwikkelstraat.

#### 5.1.1 Stap 1: Opzetten GIT repositories

##### **Back end**

Zowel de front-end als de back-end moeten gebruik maken van een GIT repository om versioning toe te passen in de code. De back-end was al voorzien van een repository doordat de 42Windmills hier een server voor hadden opengesteld waar we gebruik van konden maken. We hebben hierdoor geen aparte GIT repository gemaakt voor de back-end.

##### **Front end**

De front-end heeft heeft GITHUB server toegewezen gekregen met een publiek account. We hebben voor deze configuratie en server gekozen omdat we graag de front-end en back-end gescheiden wilden houden. Daarnaast heeft GITHUB een goede integratie mogelijkheid met Jenkins, de CI server waarvoor wij gekozen hadden.

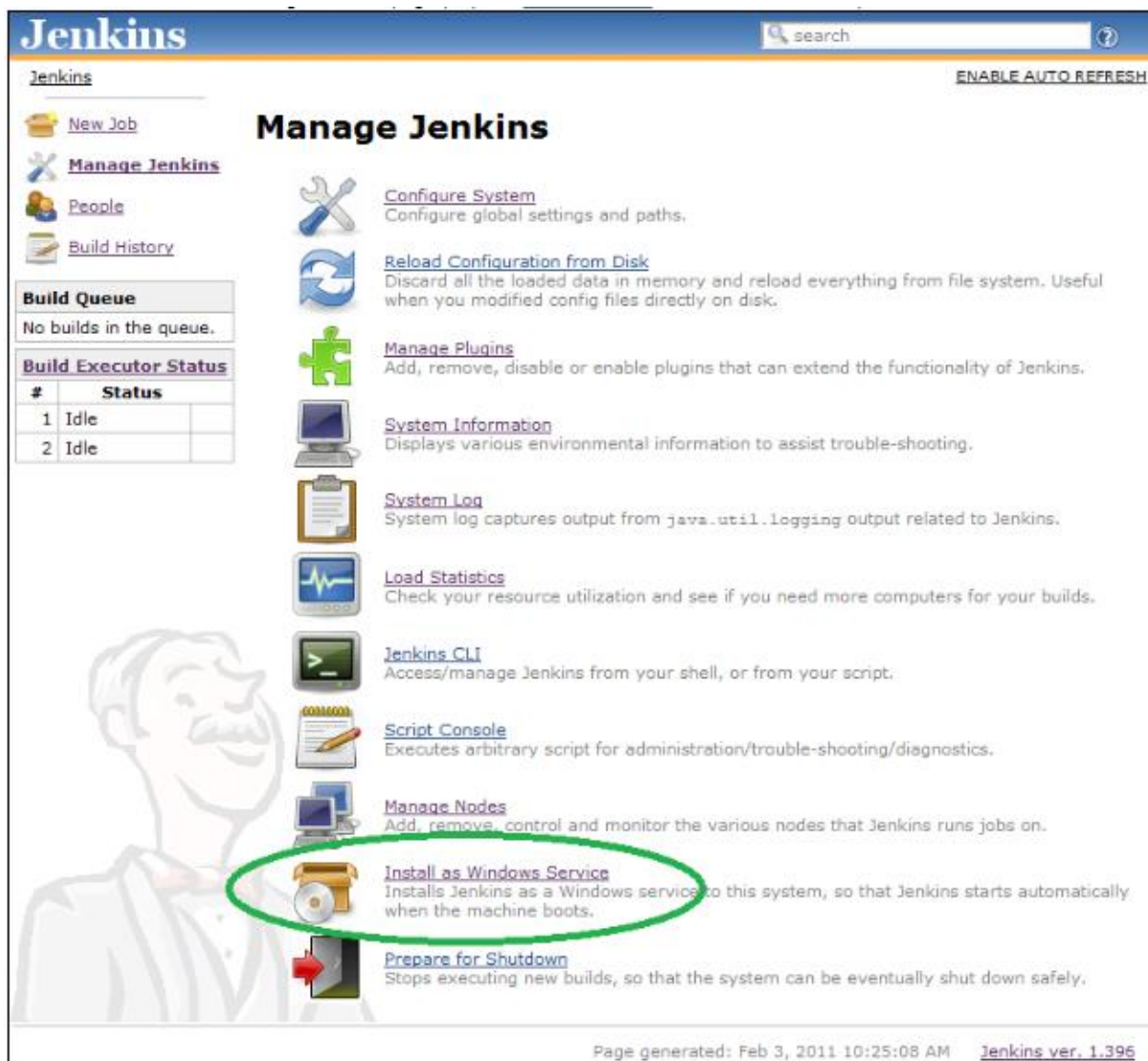
#### 5.1.2 Stap 2: Opzetten Jenkins

Onze CI server had een duidelijke documentatie waardoor het opzetten van Jenkins relatief voor zich zelf spreekt. Jenkins heeft daarnaast veel handige functionaliteit zoals de ingebouwde dashboards en ondersteuning voor diverse externe plug-ins.

##### *5.1.2.1 Jenkins stap 1: Installatie Jenkins*

Als eerst dient de Jenkins installatie gedownload te worden. Nadat de download was voltooid was duidelijk dat tevens de Java SDK gedownload moest worden om Jenkins te installeren omdat de installatie file van Jenkins een .war file bevat welke uitgevoerd moest worden. Na de installatie van Java kon Jenkins worden geïnstalleerd. Om vervolgens Jenkins op te laten starten wanneer de server opstartte hadden we gekozen om Jenkins op te laten starten als Windows Service. Handmatig konden we Jenkins opstarten door via de cmd van Windows Jenkins als command uit te voeren. Dit startte de Jenkins server en hiermee konden we via de localhost verbinding maken met de interface van Jenkins.

Nu de interface bereikbaar was moesten we de Windows Service installatie uitvoeren door in het menu naar "Manage Jenkins" te navigeren en hier install as Windows Service aan te klikken.



Dit installeerde de juiste bestanden om Jenkins als een Windows Service te laten draaien.

#### 5.1.2.2 Jenkins stap 2: Ophalen van code uit externe repositories

Om Jenkins zijn functionaliteit tot recht te laten komen dient de code op de Jenkins server worden gezet. Dit is in te stellen door de GITHUB en GIT plug-in te downloaden in het menu Manage Plugins in Jenkins. We konden hiermee tijdens het aanmaken van een nieuwe item zorgen dat de eerste stap van deze item, het checken van veranderingen is en wanneer er veranderingen plaatsvonden deze te laten pullen van de Git repository. Dit gebruikten we voor beide repositories.

Naast de GIT plug-in had de GITHUB plugin nog de functionaliteit om direct naar de GITHUB server te gaan om hier diverse informatie over de repository in te zien.

Omdat we ervoor kozen pas bij aanpassingen van de code de front-end te pullen. Wordt er periodiek een poll gestuurd om te kijken of deze verandering heeft plaatsgevonden. Dit gebeurt geheel automatisch en hoeft niet ingesteld te worden. Bij een verandering wordt de code pas van de server gehaald en op de Jenkins server



gezet. Dit zorgt dat er niet veel requests hoeven plaats te vinden wanneer er niets veranderd is en hierdoor behaalde we een performance slag.

Bij de back-end wordt er door Jenkins om een vast tijdstip een een GIT pull commando uitgevoerd om zo direct met de pull een automatische build uit te voeren. Door dit gelijk te houden blijft het overzichtelijk tijdens dit Jenkins item.

## Issues

Het pullen van de GIT repositories gebeurde al in een snel stadium omdat de configuratie redelijk makkelijk was. Wel hadden we de nodige problemen omdat er een team versie en een normale versie van de GIT repository op de back-end server van de 42Windmills was. Door dit probleem hebben we vrij lang de verkeerde repository opgehaald en door het CI proces laten lopen. Door deze fout werden de tests en builds uitgevoerd op een verouderde versie en liepen we tegen problemen aan wanneer de wijzigingen in de repository waren doorgevoerd. We hebben dit moeten oplossen door een compleet nieuwe ontwikkelstraat op te zetten met alle gebruikte tools opnieuw geconfigureerd en de juiste repository te laten pullen op de Jenkins server.

### 5.1.2.3 Jenkins stap 3: Het automatisch bouwen van de back-end

In Jenkins is het relatief eenvoudig om een build plaats te laten vinden. Wij hebben dit bereikt door een NAnt script aan te roepen welke vervolgens de C# code omzet naar uitvoerbare code. Voor een verdere omschrijving van dit NAnt proces verwijs ik naar het hoofdstuk 5.1.3 Stap 3: Automatic building. We konden dit NAnt script aanroepen door het juiste pad aan te geven in Jenkins en ook hier van een plug-in te installeren. Door apart een script aan te roepen in Jenkins werd het build proces flexibel gehouden omdat dit script tevens apart aangeroepen kan worden, los van Jenkins. Jenkins biedt naast diverse scripts om een build uit te voeren ook de mogelijkheid om d.m.v. een plug-in van MSBuild te downloaden, direct een script te bouwen. Vanwege de minder flexibele werking hebben we ervoor gekozen om dit niet te doen.

### 5.1.2.4 Jenkins stap 4: Uitvoeren van automatische unit tests

Jenkins biedt veel mogelijkheden om verschillende tests uit te voeren bij verschillende programmeertalen door de diverse plug-ins die beschikbaar zijn. Hier kan ook een NUnit plug-in worden toegevoegd om te zorgen dat vanuit Jenkins unit tests automatisch uitgevoerd kunnen worden. In Jenkins kon in de configuratie vervolgens een pre-build step worden bijgevoegd waar een cmd commando wordt in geplaatst die de NUnit Runner aanroept die op zijn beurt de NUnit tests uitvoert en hier een XML rapport over genereert.

### 5.1.2.5 Jenkins stap 5: Uitlezen NUnit test rapport

Wanneer de NUnit plug-in is geïnstalleerd kan tevens het gegenereerde rapport met deze plug-in worden uitgelezen in Jenkins. Om dit te bewerkstelligen zal de locatie van het rapport opgegeven moeten worden in een post-build step. De tests zijn uitgevoerd en weggeschreven in een XML bestand. Nu dit rapport is de uitkomst van de tests in zicht heeft kunnen ze weergegeven worden in Jenkins. Deze genereert een overzicht in een

diagram van de uitgevoerde tests om een overzicht te creëren van de historisch uitgevoerde tests.

#### 5.1.2.6 Jenkins stap 6: Uitvoeren van automatische Angular unit tests

Om de Angular tests uit te voeren, hoeft niet apart een plug-in gedownload te worden voor het uitvoeren. In de configuratie van het project kan standaard al een batch commando worden uitgevoerd die zorgt dat de tests worden uitgevoerd. Het commando "C:\Program Files (x86)\Jenkins\workspace\travel-planner.Build\run\_ng\_tests.bat" roept in een build een batch bestand aan die vervolgens de Angular tests uitvoert. Dit gebeurt door middel van het commando op de volgende pagina.

```
@echo off
```

```
echo "start ng tests"
```

```
cd "C:\Program Files (x86)\Jenkins\workspace\travel-planner.Build"
```

```
ng test --single-run
```

Het commando zorgt na het uitvoeren dat er door PhantomJS een XML rapport wordt gegenereerd die uitgelezen kan worden door Jenkins.

#### 5.1.2.7 Jenkins stap 7: Uitlezen van gegenereerde Angular test rapport

Nadat PhantomJS een XML rapport heeft gegenereerd kan Jenkins dit uitlezen door het gebruik van JUnit reporter. Dit programma zorgt dat het Angular test rapport wordt gegenereerd in een XML formaat die Jenkins kan uitlezen door de JUnit plug-in. Deze plug-in is te downloaden op de gebruikelijke manier van Jenkins.

### Issues

PhantomJS genereert een rapport in een XML formaat die Jenkins standaard niet kan uitlezen. Om dit rapport wel leesbaar te maken voor Jenkins hebben we JUnit reporter moeten toepassen in het Angular framework door deze in de configuratie van NodeJS toe te voegen. Door deze toegevoegde configuratie wordt er een JUnit rapport gegenereerd in plaats van een PhantomJS rapport. Dit XML formaat kan vervolgens in Jenkins worden uitgelezen door de JUnit plug-in.

#### 5.1.2.8 Jenkins stap 8: Uitvoeren SonarQube code analyse

Na de installatie van alle SonarQube onderdelen kan Jenkins de analyse aanroepen door de SonarQube plug-in te installeren in het plug-in center en het commando toe te voegen aan een build-step.

De build-step die toegevoegd moest worden betrof een "Execute SonarQube Scanner" build-step. In deze build step zijn een aantal configuratie parameters meegegeven voor het uitvoeren van de betreffende analyse. Deze staan vermeld in Analysis properties, De betreffende parameters staan op de volgende pagina aangegeven.

```
#Meta data
```

```
sonar.projectKey=Api
```

```
sonar.projectName=Api
```

```
sonar.projectVersion=1.0
```

```
#Source dir
```

```
sonar.sources="."
```

```
sonar.forceAnalysis=true
```

```
#sonar.global.exclusions=**/App.Files/**;**/App.Lib/**;**/App.Lib.Tests/**;
```

```
#**/App.Services.BusinessActivityManager/**; **/FortyTwo.PortalControl/**;
```

```
**/packages/**;
```

```
##*/App.Providers.Membership/**; **/App.Resources/**; **/App.RestService/** ;
```

```
**/ServiceRunner/**
```

```
#encoding
```

```
sonar.sourceEnc
```

Deze paramaters zorgen dat het juiste project wordt geanalyseerd. De project meta data wordt weergeven in de webserver om onderscheidt te maken tussen de verschillende analyses en projecten. De source dir zorgt ervoor dat projecten uitgesloten kunnen worden voor analyse en de decodering is nodig om de bestanden uit te kunnen lezen mits deze in UTF-8 formaat zijn gedecodeerd, wat bij ons het geval is.

## Issues

Het analyseren gaf in eerste instantie een foutmelding wanneer de analyse voltooid was, namelijk een `IllegalArgument exception`. Deze foutmelding is verholpen zoals beschreven in stap 4 van de installatie van SonarQube.

### 5.1.3 Stap 3: Automatic building

Automatic building is nodig om te zorgen dat de code uitgevoerd kon worden. Omdat de back-end code alleen niet direct uitvoerbaar is, is alleen de back-end voorzien van een build stap.

We hebben dit gerealiseerd door een NAnt script aan te roepen welke vervolgens MSBuild aanroept om de build uit te laten voeren. We hebben hiervoor gekozen zodat er de flexibiliteit blijft om met dit script nog andere build stappen uit te voeren. Naast dit voordeel staat in het onderzoeksrapport ook beschreven dat NAnt sneller is. Dit is duidelijk merkbaar. We hebben dit nog getest door MSBuild apart aan te roepen.

#### Issues

Het automatisch bouwen van een C# solution bracht veel uitdagingen met zich mee. Dit zat hem vooral in het feit dat de 42Windmills code diverse dependencies met zich mee bracht. Hieronder zal ik kort beschrijven wat deze issues waren.

1. Diversen project bestanden bevatten een referentie naar het .NET framework 4.5.2. Omdat deze afhankelijkheid bestond waren we verplicht dit framework toe te passen op de Jenkins server. Dit is bracht met zich mee dat NAnt geen ondersteuning biedt voor het .NET 4.5 framework waardoor projecten niet gebuild konden worden door Jenkins. We hebben dit kunnen oplossen door een speciale versie van NAnt te gebruiken, namelijk NAntContrib. Deze versie kan de MSBuild tool direct aanroepen. Door deze oplossing behielden we de flexibiliteit van een apart script waarmee we door gebruik van de Windows cmd nog steeds een aparte build kunnen laten uitvoeren, maar verloren we de snelheid van NAnt omdat hiermee alsnog MSBUILD aangeroepen moest worden.
2. Een verwijzing in de project files naar Visual Studio 2008 gaf een lastig op te lossen probleem. In het Visual Studio Toolspath in de projectfiles van de 42Windmills staat een verwijzing naar de Visual Studio versie die door hen gebruikt wordt. Door deze verwijzing ontstonden diverse moeilijk op te lossen foutmeldingen omdat er werd gesproken over een verkeerde toolspath. Na een onderzoekje konden we achterhalen dat het een verwijzing was naar de Visual studio versie, welke gebruik maakt van een MSBUILD SDK om de code op te bouwen. Deze SDK wordt alleen geleverd met Visual Studio 2008, die niet op de Jenkins server geïnstalleerd stond. Ondanks dat we zelf gebruik maakte van Visual Studio 2015 op de Jenkins server hebben we eerst Visual Studio 2008 moeten installeren en verwijderen om deze SDK te bemachtigen omdat deze niet los te downloaden was. Door de installatie van VS 2008 is de SKD geïnstalleerd maar niet meer verwijderd na verwijdering van VS 2008. Hierdoor was de foutmelding opgelost en konden de files worden gebuild.

3. Vanwege een verwijzing naar VS 2008 in de Solution file van C# konden diverse files niet herkend worden. Dit hebben we kunnen oplossen door deze verwijzing handmatig uit de Solution file te verwijderen. Hierdoor was deze foutmelding verholpen.

#### 5.1.4 Stap 4: Manual building

Vanwege het build proces zoals beschreven in het vorige hoofdstuk is het nodig om de build tool van C# (MSBUILD) te gebruiken. Dit is gedownload en volgens de handleiding geïnstalleerd zonder enige problemen. MSBUILD voorziet de applicatie nu van het build proces samen met NAnt.

#### 5.1.5 Stap 5: Automatic testing C#

Automatisch testen wordt door onze ontwikkelstraat verzocht door het framework NUnit. NUnit voorziet in een framework voor het aanmaken van testen. Naast het aanmaken van testen zijn er nog diverse NUnit tools nodig om de NUnit tests automatisch uit te voeren. Deze stappen worden hier beschreven.

##### 5.1.5.1 Automatic testing step 1: Opzetten NUnit framework

Omdat wij gebruik maken van Visual Studio 2015 kunnen de NUnit tools automatisch worden gedownload en geïnstalleerd. VS 2015 koppelt deze direct aan de C# Solution. Het dit geldt tevens voor het NUnit framework, welke gebruikt wordt voor het aanmaken van NUnit tests en diverse methodes levert om de testen uit te voeren. Wel dient voor elke afzonderlijk project het framework vervolgens toegevoegd te worden.

##### 5.1.5.2 Automatic testing step 2: Opzetten overige tools

Net zoals het NUnit framework kunnen de overige tools die nodig zijn ook via de Nuget package manager van Visual Studio gedownload en geïnstalleerd worden.

Om de NUnit tests automatisch uit te laten voeren worden enkele tools bijgevoegd om de tests te herkennen en vanaf een extern punt uit te voeren. Deze tools worden de NUnit Engine en de NUnit runner genoemd. De Nunit engine zorgt ervoor dat de tests worden herkend buiten Visual Studio en deze houdt dan ook bij waar de tests staan. NUnit Runner maakt het mogelijk om door middel van de herkende tests deze betreffende tests uit te voeren via een Windows cmd command. Deze tools moeten allen wel van de zelfde versie zijn om te zorgen dat ze uitgevoerd kunnen worden.

##### 5.1.5.3 Automatic testing step 3: Genereren NUnit rapport

Door de commands die aangeroepen worden via de NUnit runner kunnen rapporten gegenereerd worden in XML. Hierbij kunnen verschillende soorten formaten geselecteerd worden. Omdat de Jenkins server NUnit plug-in momenteel geen versie 3 rapporten kan uitlezen vanwege een bug was het nodig om deze rapporten in het Nunit 2 formaat te laten gereren.

## Issues

Het NUnit framework genereert standaard versie 3 als XML formaat. Omdat de Jenkins server NUnit plug-in momenteel geen versie 3 rapporten kan uitlezen vanwege een bug was het nodig om deze rapporten in het NUnit 2 formaat te laten genereren. Dit was mogelijk door het commando dat in Jenkins uitgevoerd werd een commando bij te voegen. Dit commando betrof `format=nunit2`.

### 5.1.6 Stap 6: Automatic testing in Angular 2

#### 5.1.6.1 Automatic testing step 1: Opzetten Jasmine framework

Het opzetten van het Jasmine framework is standaard gedaan als je een project maakt met Angular CLI, daarom konden deze gelijk gebruikt worden. Vervolgens moesten de tests in de `.spec` files geschreven worden zodat de Karma runner deze tests uit zou voeren.

#### 5.1.6.2 Automatic testing step 2: Opzetten overige tools

De overige tools zijn Karma en Protractor, deze worden ook automatisch gegenereerd bij het genereren van een Angular2 project gebruikmakend van de Angular CLI. De configuratie files waren zo ingesteld dat ze de `.spec` files uit zouden voeren, dit is onveranderd gebleven omdat dit een Angular best practice is.

#### 5.1.6.3 Automatic testing step 3: Genereren XML rapport

Voor het genereren van de XML rapporten moest JUnitReporter geïnstalleerd worden, dit is gebeurt met het NPM commando `npm install JUnitReporter -g -dev`. Vervolgens is dit klaar voor gebruik en worden de testresultaten in een xml bestand opgeslagen.

### 5.1.7 Stap 7: Code quality checking

Sonarqube wordt in dit project gebruikt voor het checken van de code format en de code style om de code kwaliteit hoog te houden. Om dit process uit te laten voeren zijn er een aantal modules nodig van SonarQube. Deze zijn allen te vinden op de website van Sonarqube.

#### 5.1.7.1 SonarQube Stap 1: Sonarqube installatie

Om de rapportages van de verschillende analyses op te slaan is de installatie van SonarQube nodig. Deze installatie zorgt ervoor dat er een database wordt aangemaakt om de rapportages in op te slaan en een webserver wordt gehost om de rapportages van de analyses worden getoond.

De installatie vond plaats door een gecomprimeerd bestand te downloaden en vervolgens te uit te pakken. Alle configuratie stond standaard al goed waardoor na de installatie de server direct in gebruik kon worden genomen. De installatie vond plaats door het batch commando `InstallNTService` uit te voeren.

Na de installatie kon de server starten door het batch commando `StartSonar.sh` aan te roepen in de commandline. Hierdoor werd webserver beschikbaar en kon data worden opgeslagen.

#### 5.1.7.2 SonarQube Stap 2: Sonar Runner installatie

Om SonarQube volledig automatisch uit te laten voeren dient een zogenaamde Sonar Runner geïnstalleerd te worden. Dit gebeurde door een gecomprimeerd bestand uit te pakken. Dit waren vervolgens alle stappen die uitgevoerd moesten worden om de Sonar Runner te installeren.

#### 5.1.7.3 SonarQube Stap 3: Sonar Scanner installatie

De Sonar Scanner verzorgt de analyses van SonarQube. Het programma analyseert de geconfigureerde code en maakt een rapport van de analyse. Dit rapport kan vervolgens worden uitgelezen door de SonarQube webserver.

De installatie van de Sonar Runner vond plaats door een gecomprimeerd bestand uit te pakken en vervolgens de configureren. De configuratie gebeurde in een sonar-scanner.properties bestand. Het enige wat heur aangepast moest worden om de Analyses uit te kunnen voeren, was de encoding format veranderen naar UTF 8 format. Na deze aanpassing was de scanner gereed.

#### 5.1.7.4 SonarQube Stap 4: Sonar C# Plug-in installatie

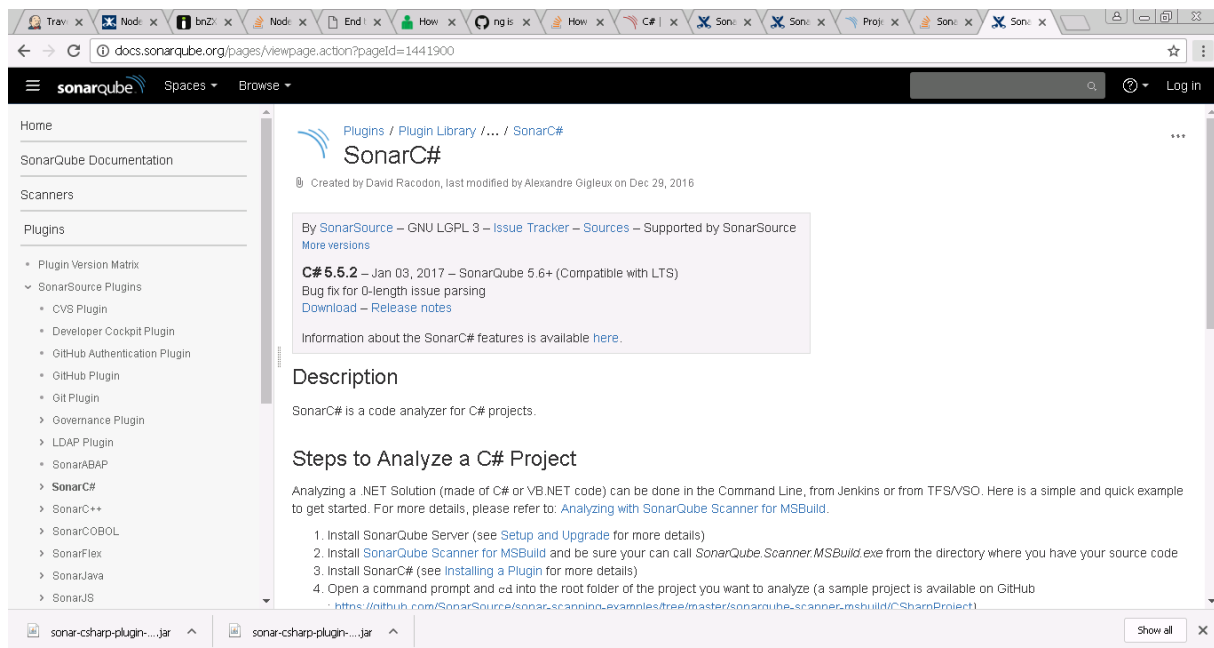
SonarQube leest standaard Java code uit. Omdat dit project beschikt over C# en Javascript code moeten 2 tal plug-ins gedownload worden. Dit kan handmatig door het bestand aan de map downloads toe te voegen of door in de SonarQube webserver de plug-in in het plug-in beheer system de plug-in toe te voegen. We hebben ervoor gekozen om de plug-ins handmatig te downloaden en toe te voegen. Dit heeft later voor problemen gezorgd. Deze zijn verder benoemd onder het kopje "issues".

De de installaties en het oplossen van de issues was de SonarQube webserver gereed voor gebruik.

### Issues

Wanneer de C# plug-in is gedownload ontstonden er fouten bij het uitvoeren van een analyse in Jenkins. De analyse zelf werd keurig uitgevoerd, maar naarmate de analyse ten einde kwam en het rapport uitgelezen moest worden, kwam een probleem naar voren.

Ten tijde van de installatie van de C# plug-in werd er niks waargenomen over enige bugs in het uitlezen van C# code. Echter werd duidelijk dat het rapport niet uigelezen kon worden door de C# plug-in omdat hierin een bug zat. Wanneer duidelijk werd dat er een nieuwe versie beschikbaar was, kwam ook de bug naar voren. Onderstaand is een screenshot bijgevoegd van de betreffende bug fix in het programma in een nieuwere versie van de plug-in.



Dit probleem was makkelijk te verhelpen door de nieuwe versie handmatig te updaten. De nieuwe versie werd gedownload en in de downloads map geplaatst. Nu bleef de foutmelding aanhouden waardoor we dachten dat er nog iets anders fout was aan het programma of dat het niet opgelost was. Maar na een onderzoek te hebben verricht bleek het nodig om de plug-in alsnog in de webserver automatisch te laten installeren omdat de nieuwe versie niet herkend was.

### 5.1.8 Stap 8: Issue tracking

Om de openstaande issues en afgesloten issues bij te houden wordt geen specifieke tool gebruikt. Hier is voor gekozen omdat de ontwikkeling van het project te kleinschalig is voor een apart issue tracking systeem en volstaat het gebruik van een Excel document als issue tracking tool.



## 6 TOOLS GEBRUIK EN CONFIGURATIE

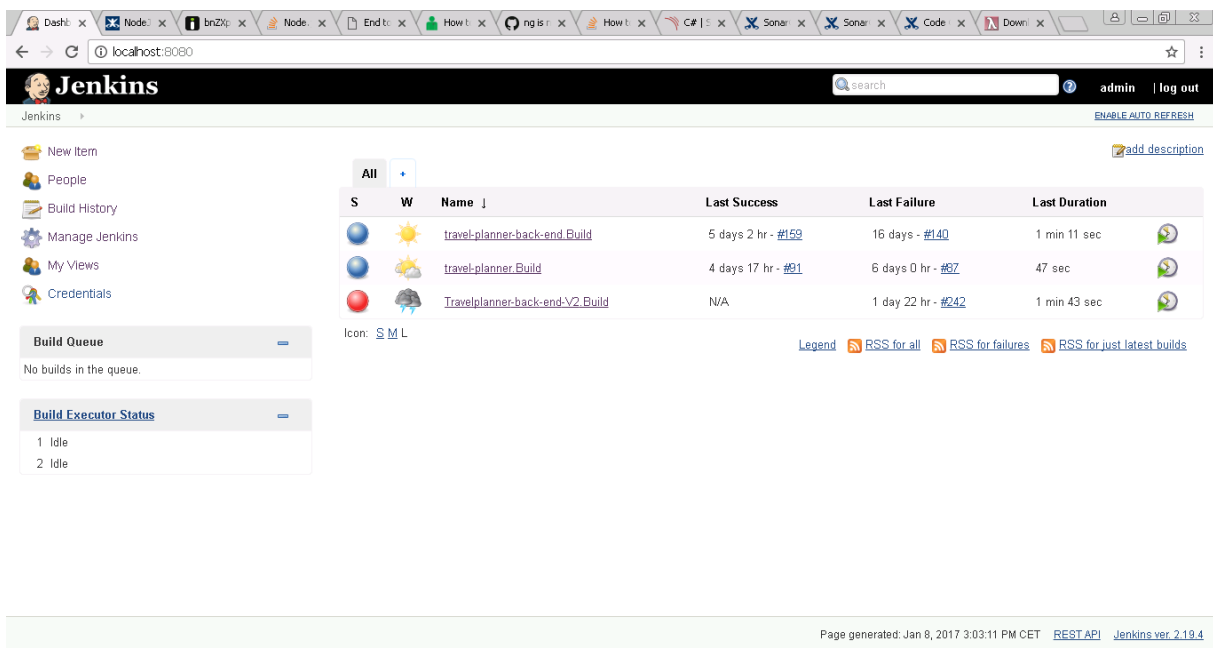
### 6.1 Jenkins

#### 6.1.1 Gebruik







Jenkins biedt een server voor Continuous 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.

##### 6.1.1.1 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



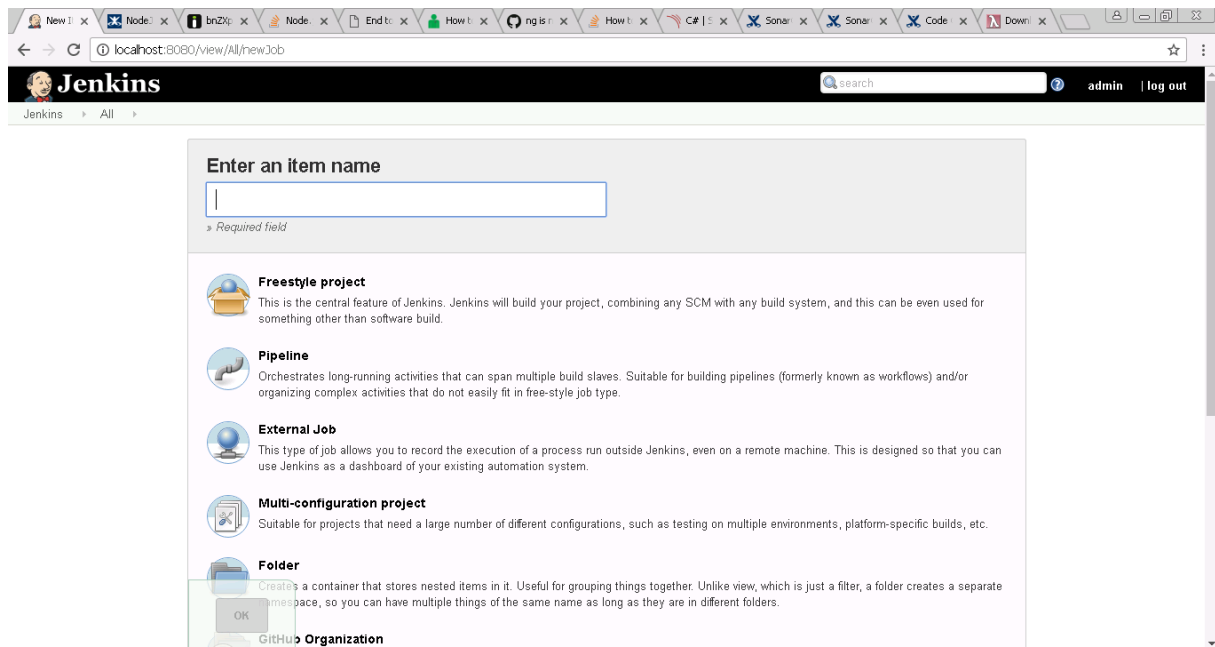
The screenshot shows the Jenkins dashboard interface. At the top, there's a navigation bar with the Jenkins logo, a search bar, and links for 'admin' and 'log out'. Below the navigation bar, there's a sidebar on the left with links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Credentials'. The main content area displays a table of builds. The table has columns for 'S' (Status), 'W' (Weather icon), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. There are three builds listed: 'travel-planner-back-end\_Build', 'travel-planner\_Build', and 'Travelplanner-back-end-V2\_Build'. The first two builds show success status (blue circle) and weather icons (sun and cloud), while the third shows a failure status (red circle) and a storm icon. Below the table, there's a 'Build Queue' section showing 'No builds in the queue.' and a 'Build Executor Status' section showing '1 Idle' and '2 Idle'. At the bottom, there's a footer with the text 'Page generated: Jan 8, 2017 3:03:11 PM CET' and links for 'REST API' and 'Jenkins ver 2.19.4'.

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="#">#91</a>	6 days 0 hr - <a href="#">#97</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

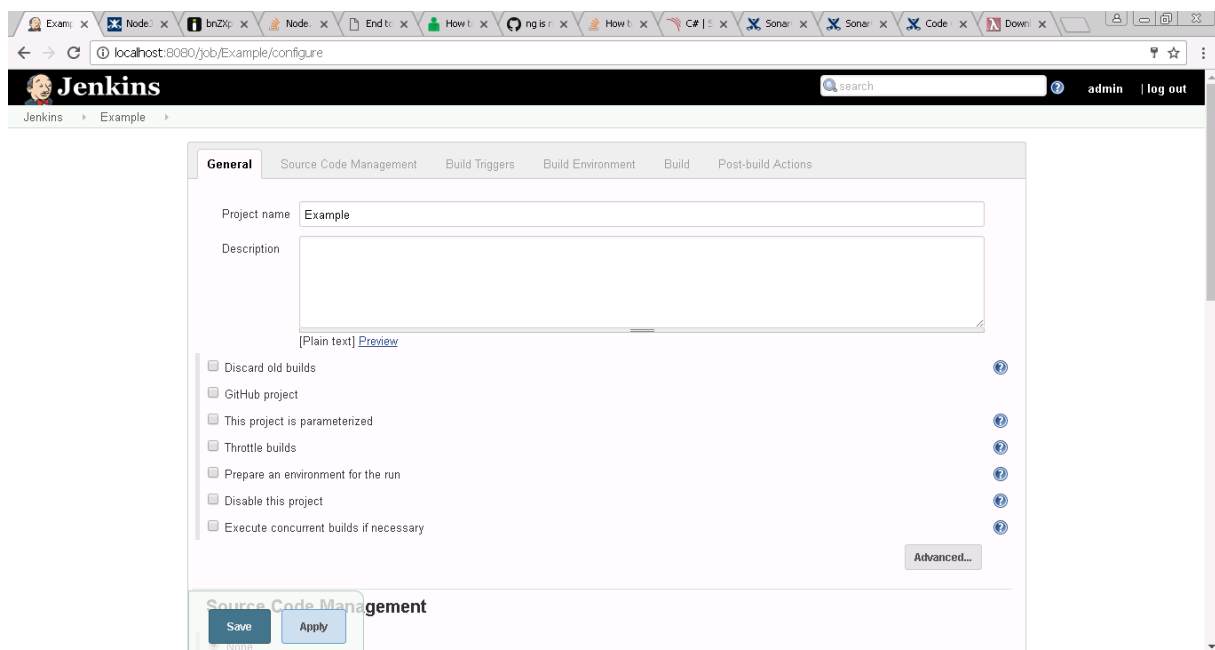
### 6.1.1.2 Nieuw item

Onder het kopje New item kan een nieuw project worden toegevoegd. Dit is dus ook gedaan om onze builds/items op te bouwen.

Wanneer we op het knopje hebben geklikt ontstaat het volgende scherm.



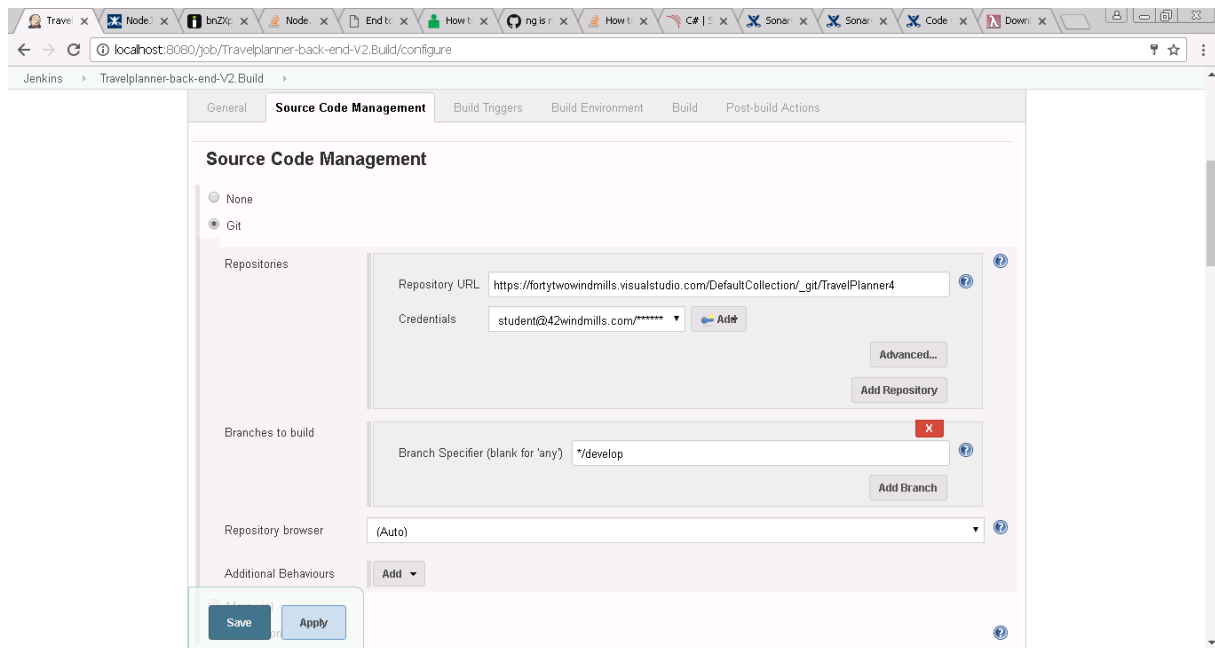
Hiermee kunnen verschillende soorten projecten gekozen worden. Wij hebben hiervoor Freestyle project gekozen omdat wij zoveel mogelijk instellingen willen kunnen aanpassen op de server. Daarboven geven we de naam. Wanneer we op OK klikken belanden we in het volgende scherm.



In dit scherm kunnen we een beschrijving meegeven en met het gebruik van plug-ins opties bijvoegen voor integratie met externe servers.

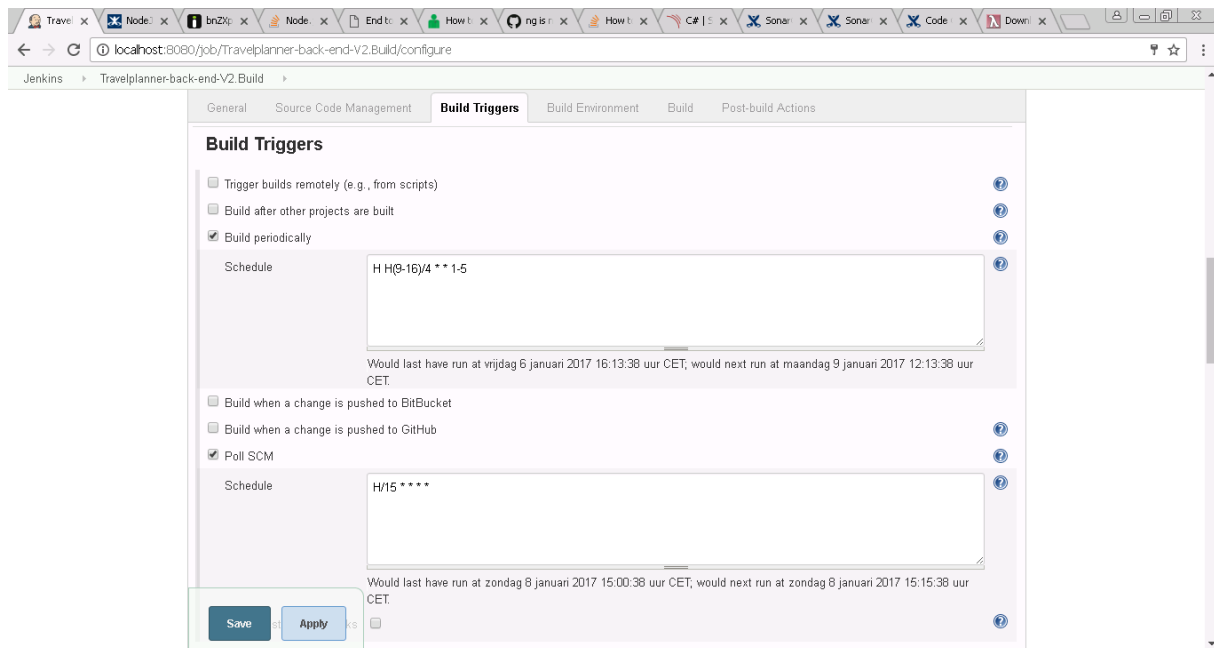
### 6.1.1.3 Source Code Management

Onder Source code management kunnen we een server bijvoegen waar de Jenkins server de code vandaan haalt. In dit voorbeeld wordt de repository van 42Windmills gebruikt met bijbehorende account en user name. Daarnaast wordt aangegeven welke branche wordt opgehaald.



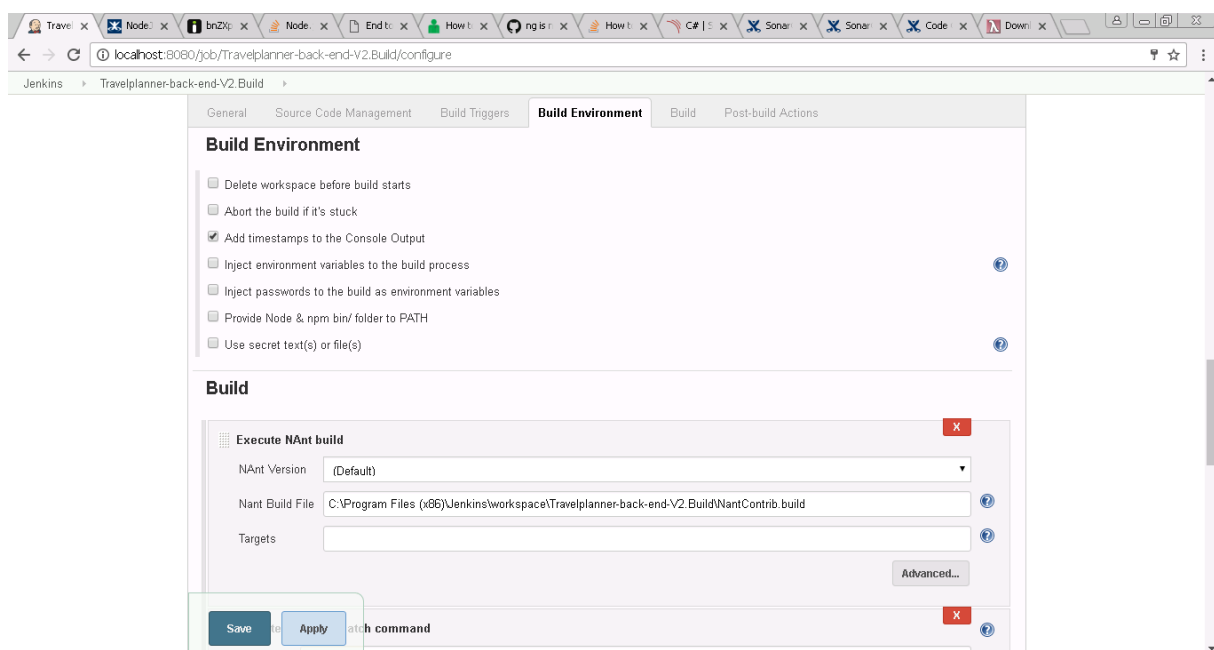
#### 6.1.1.4 Build triggers

In dit kopje in de configuratie van build triggers kan aangegeven worden wanneer een project gebuild moet worden. In dit het onderstaande voorbeeld gebeurt het periodiek. Daarnaast wordt om het kwartier een poll gestuurd om te checken of er veranderingen hebben plaatsgevonden in de Source Code Management server.



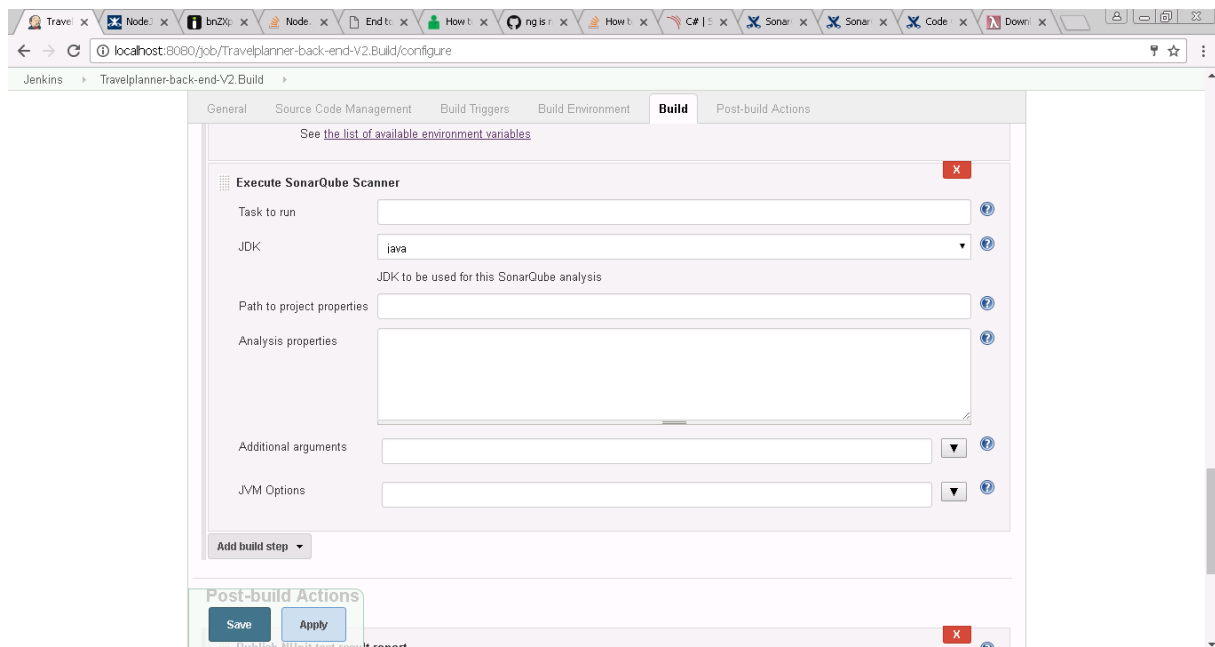
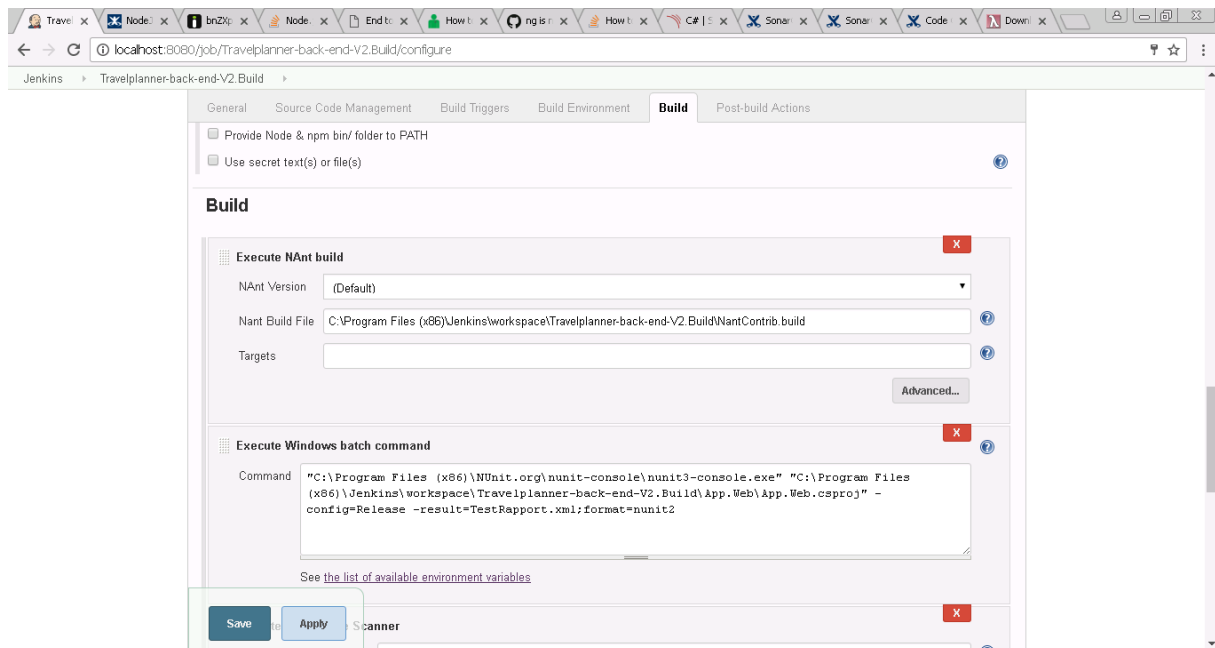
#### 6.1.1.5 Build Environment

Onder dit kopje kunnen verschillende opties bijgevoegd worden aan de build omgeving. In dit voorbeeld wordt een timestamp toegevoegd aan de console output.



### 6.1.1.6 Build

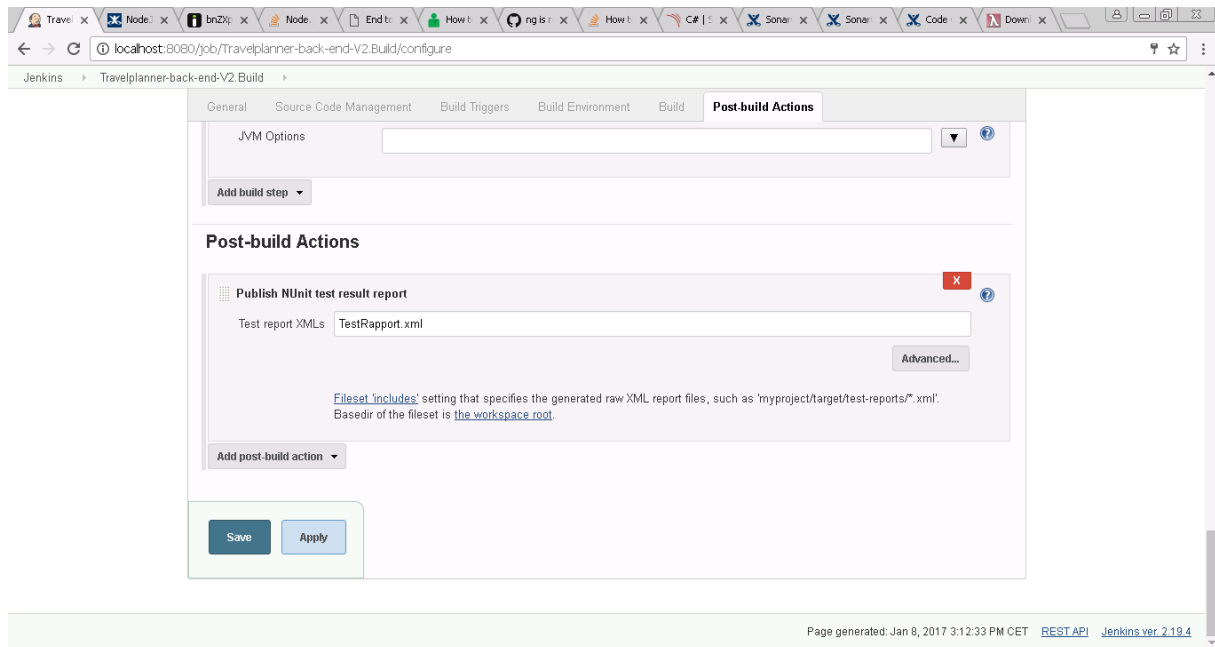
In dit onderdeel van de configuratie kan de build van het project plaatsvinden. We specificeren hier eerst een build methode door deze als build step toe te voegen. Omdat we de plug-ins hebben geconfigureerd kunnen we het uitvoeren van een NAnt build selecteren en hierbij de juiste file selecteren met de betreffende versie.



Bij het toevoegen van een build step kan ook gekozen worden om een SonarQube Scan uit te voeren.

### 6.1.1.7 Post Build Action

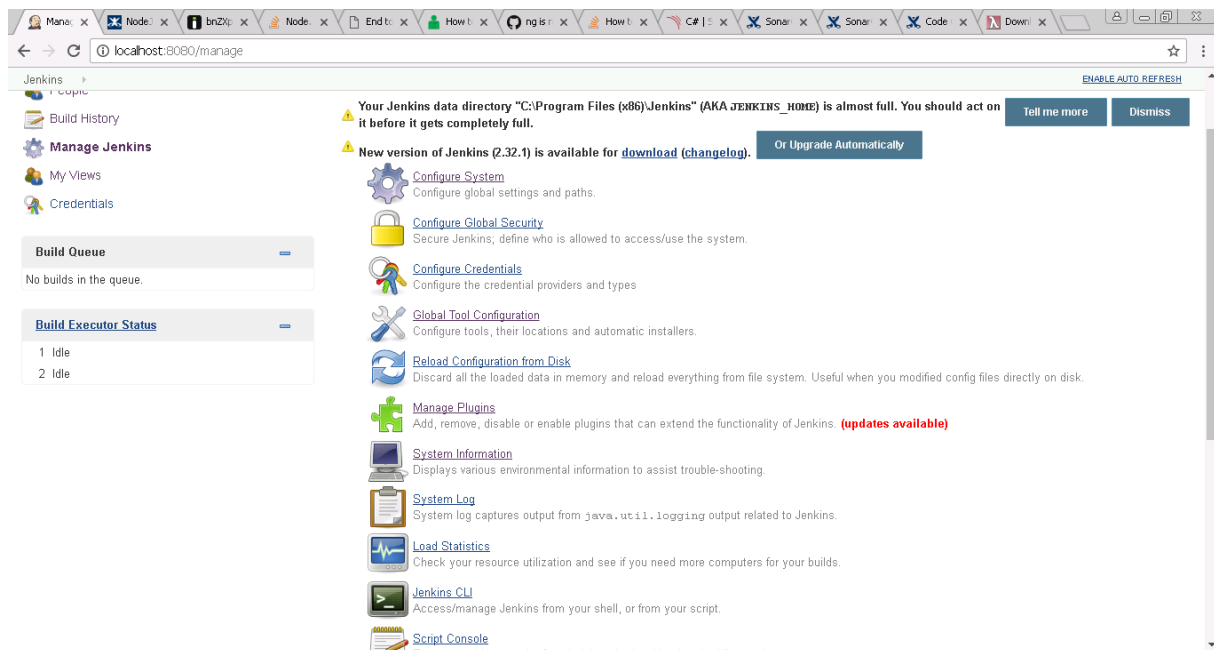
Wanneer een build heeft plaatsgevonden kunnen acties worden toegevoegd aan de hand van geïnstalleerde plug-ins. Door de installatie van Nunit plug-in kan het rapport van Nunit worden uitgelezen.



Hier is de naam aangegeven van het test rapport van NUnit. Door dit toe te voegen leest Jenkins het uit. Wanneer vervolgens op save wordt gedrukt zijn de instellingen opgeslagen.

### 6.1.1.8 Manage Jenkins

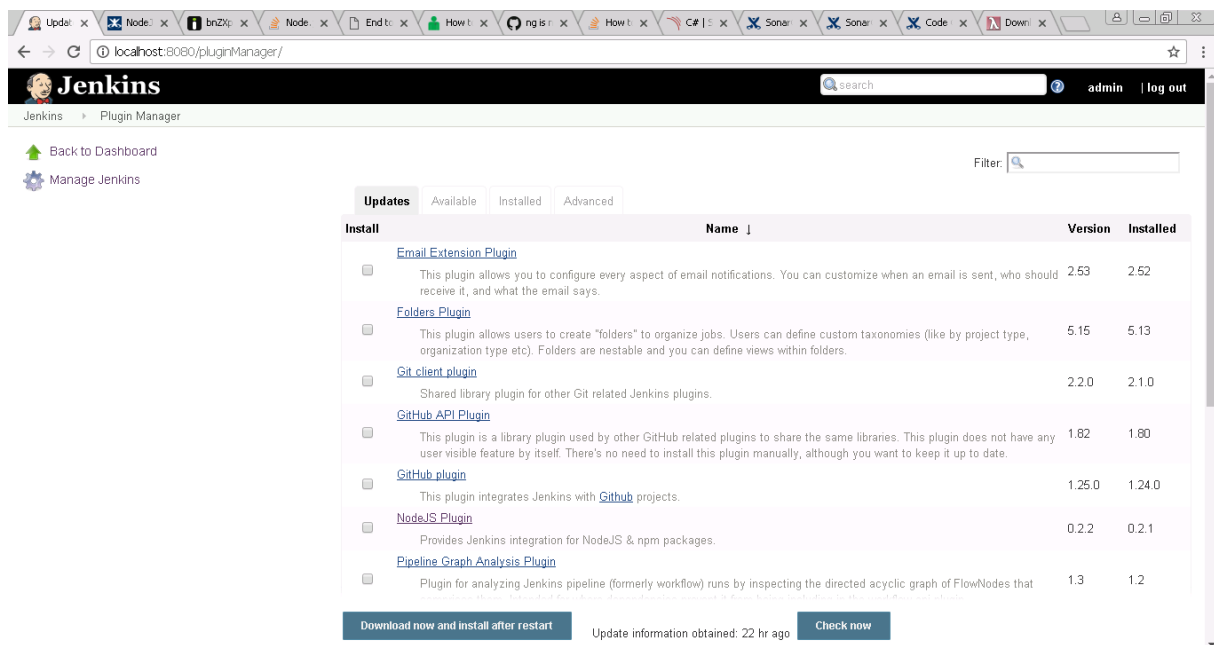
Om Jenkins aan te passen door bijvoorbeeld de gebruikte Plug-ins aan te passen of omgevingsvariabele gebruikt door Jenkins toe te voegen, kan dit scherm geraadpleegd worden. In dit scherm worden de verschillende Jenkins instellingen beheert.



Hier is tevens informatie over de versie van Jenkins te vinden en de console van Jenkins te gebruiken.

#### 6.1.1.9 Manage Plugins

In de configuratie van Jenkins worden de verschillende plug-ins beheerd. Door deze plug-ins toe te voegen wordt functionaliteit toegevoegd om externe applicaties aan te sturen zoals bijvoorbeeld het uitlezen van NUnit rapport.

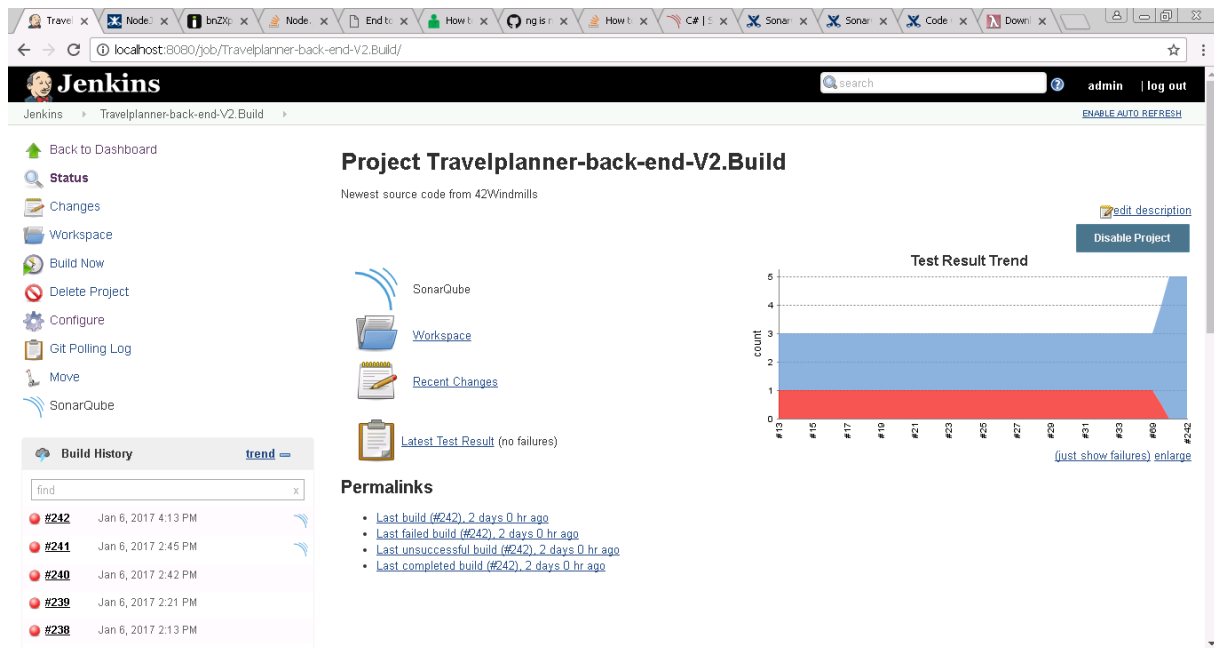


Het overzicht weergeeft in het bovenstaande voorbeeld de plug-ins waarvoor updates beschikbaar zijn. Daarnaast zijn verschillende overzichten beschikbaar om plug-ins te downloaden en bekijken. Om een plug-in toe te voegen kan naar available genavigeerd worden en door te zoeken de juiste plug-in te selecteren en vervolgens op

de knop “Download new and install after restart” te klikken. Hierdoor wordt hij gedownload en na het restarten van Jenkins actief gezet en geïnstalleerd. Het installeren van een plug-in zorgt voor meer opties bij het uitvoeren van een item in Jenkins.

#### 6.1.1.10 Project overzicht

Wanneer in het dashboard overzicht op een project wordt geklikt wordt het volgende scherm getoond.



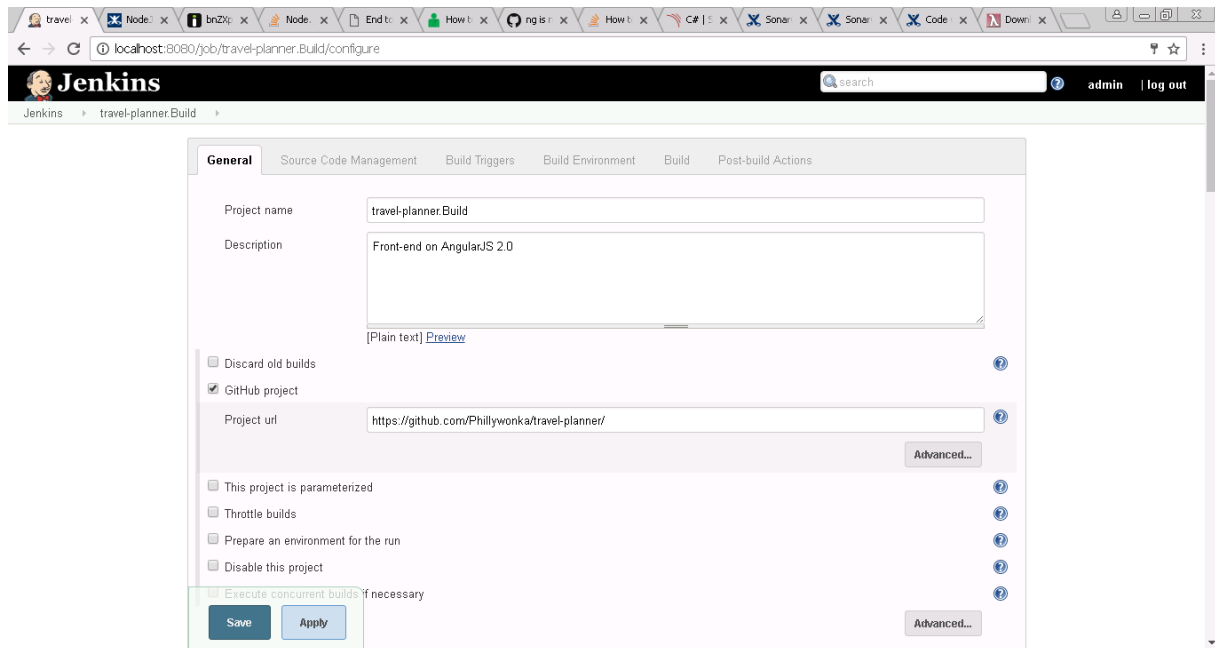
Dit overzicht weergeeft de test resultaten en de laatste build historie. Vanuit dit overzicht kunnen naar verschillende schermen worden genavigeerd om dit project in te stellen (configure) of om de log van GIT te bekijken. Changes weergeven de aanpassingen die gedaan zijn door middel van git en dit zorgt dan ook voor een log in dit project van de verschillende aanpassingen. Wanneer snel naar de source code genavigeerd wil worden die Jenkins gebruikt kan het Workspace mapje gebruikt worden.



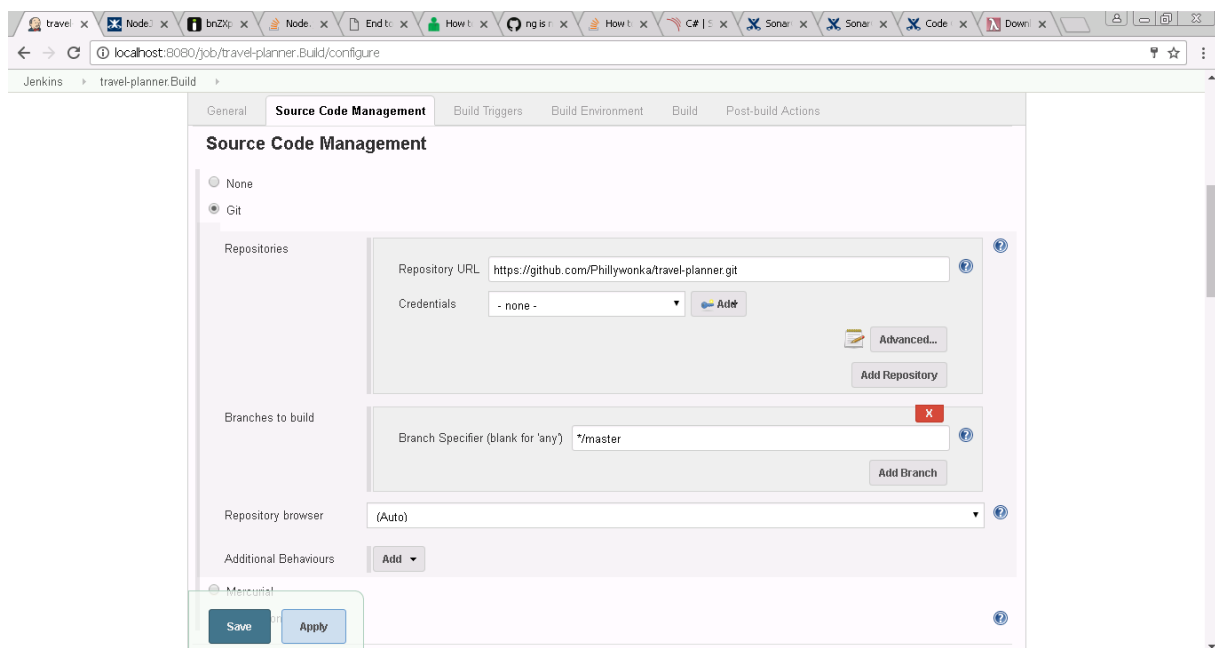
## 6.1.2 Configuratie

### 6.1.2.1 Front-end project

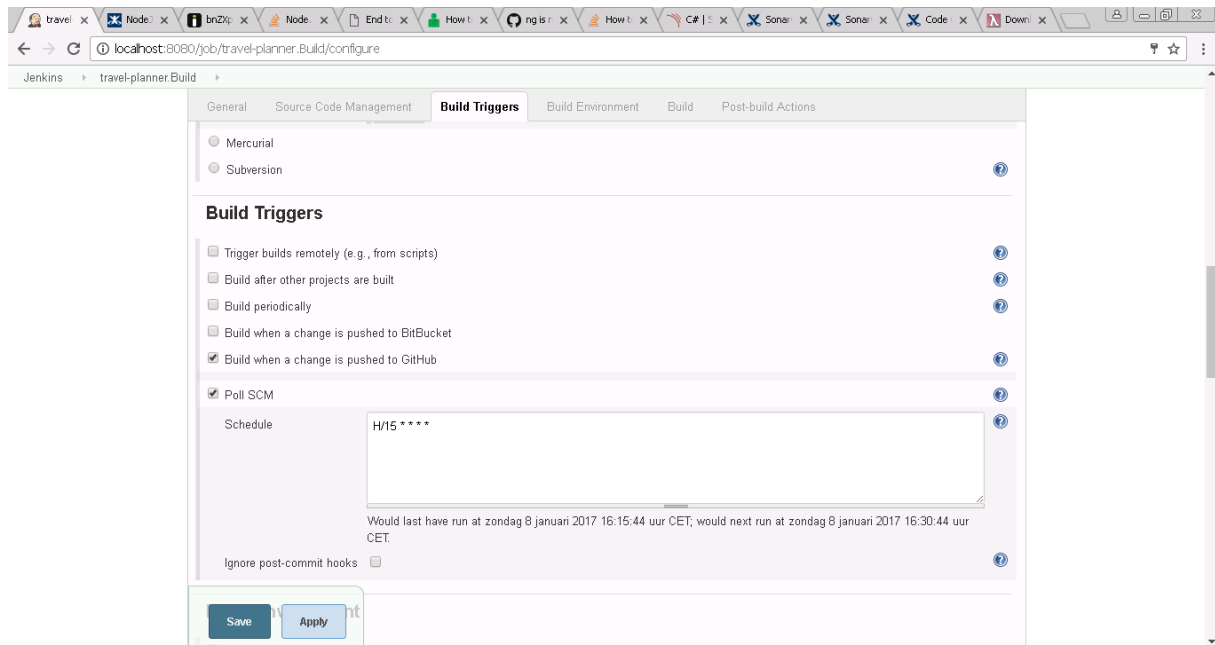
Het onderstaande overzicht toont de configuratie van het front-end project in Jenkins. Dit project heet travel-planner.build. en beschikt alleen over de code van het Angular project.



De beschrijving en naam onderscheidt dit project van de back-end. Daarnaast kunnen we zien dat het project van GITHUB wordt gehaald en hier een link van is toegevoegd zodat Jenkins is geïntegreerd met GITHUB.D



De Source Code Management is ingeregeld zodat de repository van Github gebruikt wordt. Omdat dit een publiek project is, hebbe we geen credentials opgegeven. We gebruiken de Master branche om de code op te halen.



De build triggers worden alleen gebruikt om de code te pullen van Github door middel van Git. We hebben dit ingesteld zodat omdat de 15 minuten wordt gechecked of hier een verandering heeft plaatsgevonden. Wanneer de verandering heeft plaatsgevonden wordt de code gedownload van Github. Daarnaast zijn er geen build triggers nodig omdat dit project niet daadwerkelijk gebuild wordt omdat de code al uitvoerbaar is. Om deze rede slaan we de Build Environment overzicht over.

Het volgende scherm weergeeft het aanroepen van een batch commando bestand om de angular tests uit te voeren. In dit Jenkins commando wordt dan ook niets anders gedaan dan het uitvoeren van dit .bat command:

```
@echo off
```

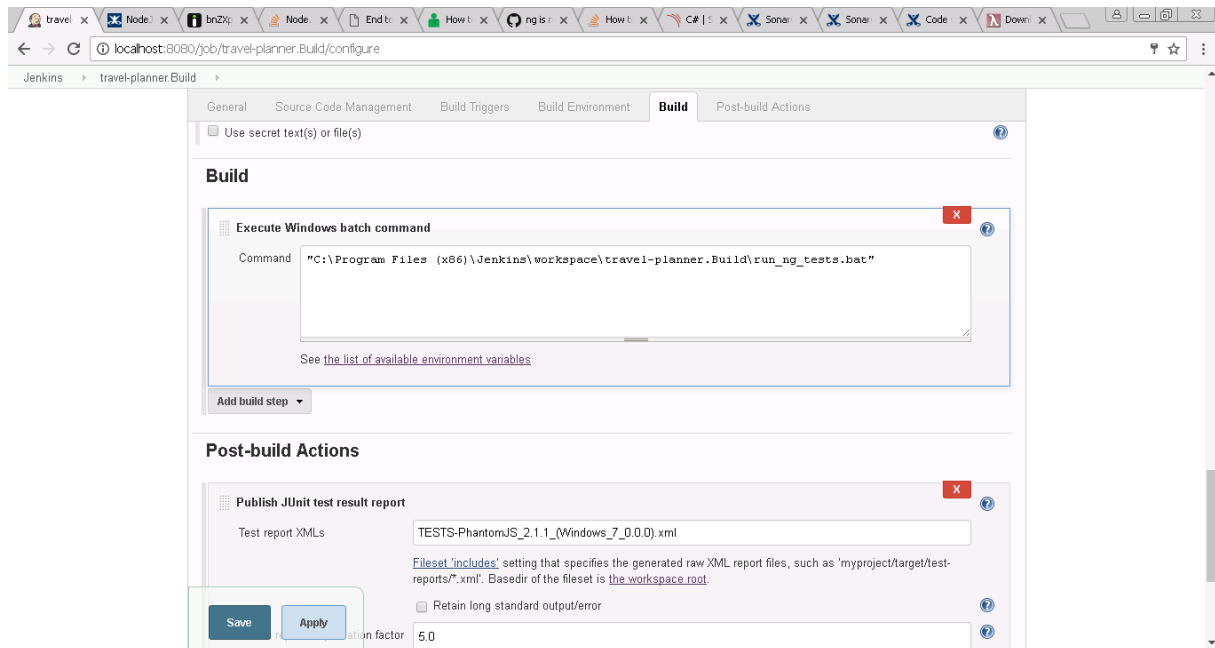
```
echo "start ng tests"
```

```
cd "C:\Program Files (x86)\Jenkins\workspace\travel-planner.Build"
```

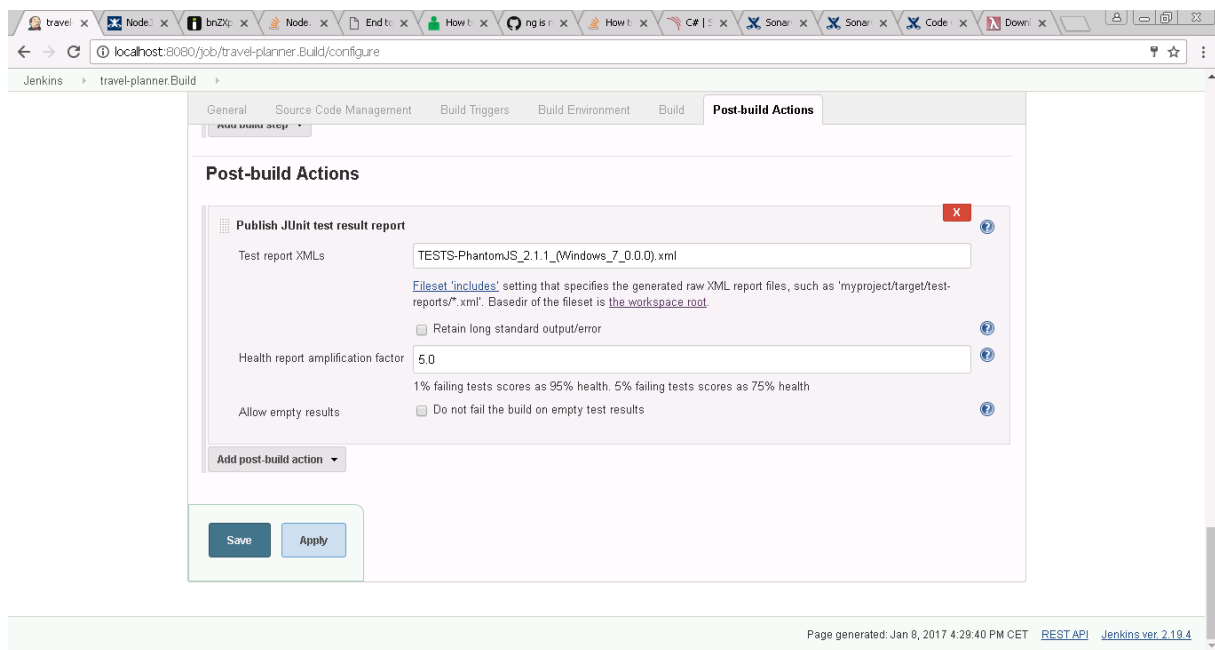
```
ng test --single-run
```

Hierbij wordt aangegeven dat de tests starten en genavigeerd naar de travel-planner.Build workspace. Vervolgens wordt aangegeven dat te tests eenmalig gedraaid moeten worden.

Onderstaand het aanroepen van het bestand in Jenkins.



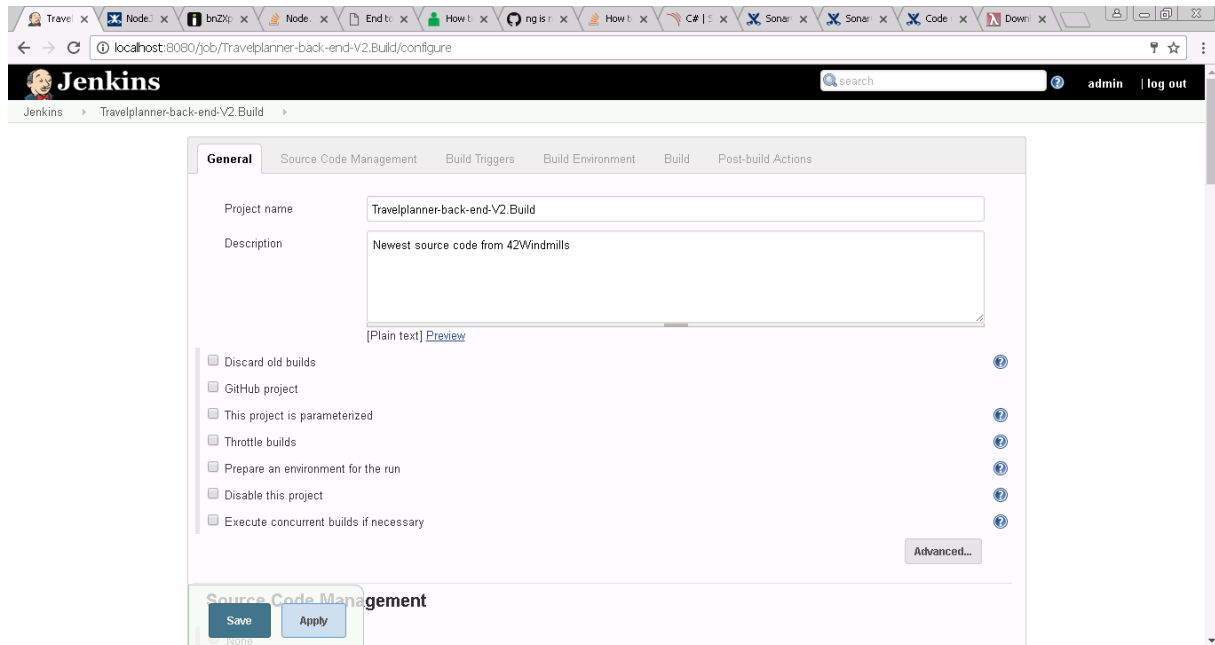
Nu hierbij geautomatiseerd de tests worden uitgevoerd en het rapport wordt gegenereerd wordt door de JUnit plug-in de optie toegevoegd om een rapport uit te lezen. De rapportnaam wordt gedefinieerd in de Post-build Actions.



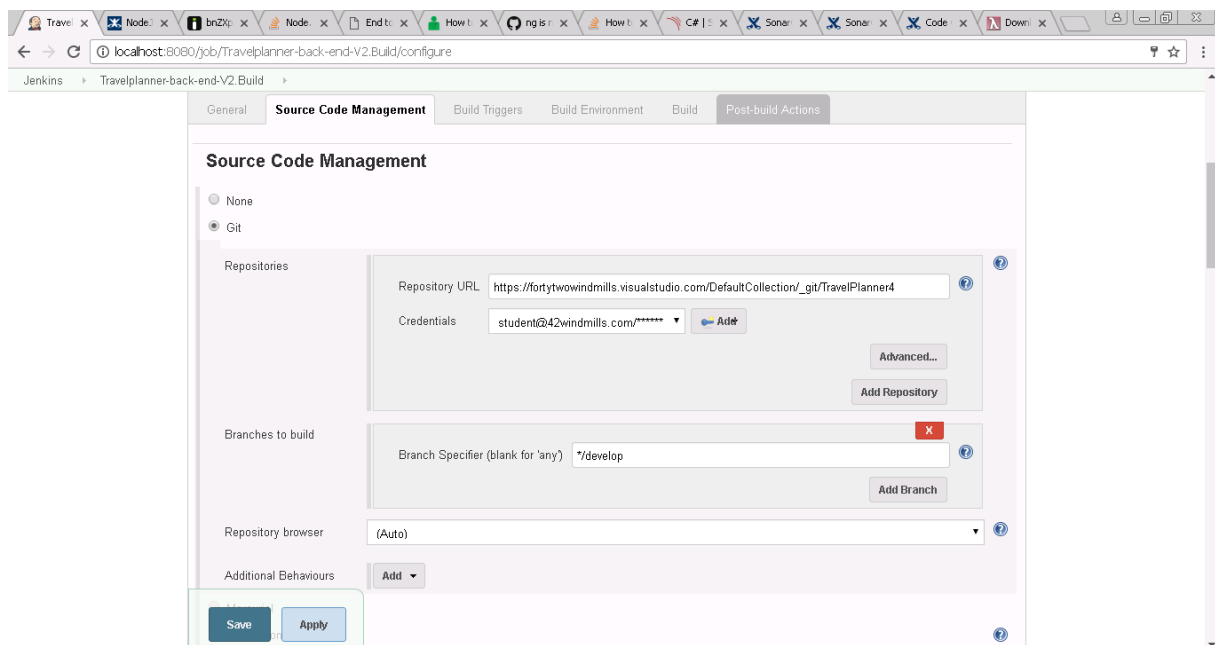
We geven de naam en het pad van het test rapport op en de Health factor. Dit betreft een factor hoe vaak een fout mag voorkomen. Wij hebben gekozen dat 5 procent van de testen fout mogen gaan om succesvol te bouwen.

### 6.1.2.2 Back-end project

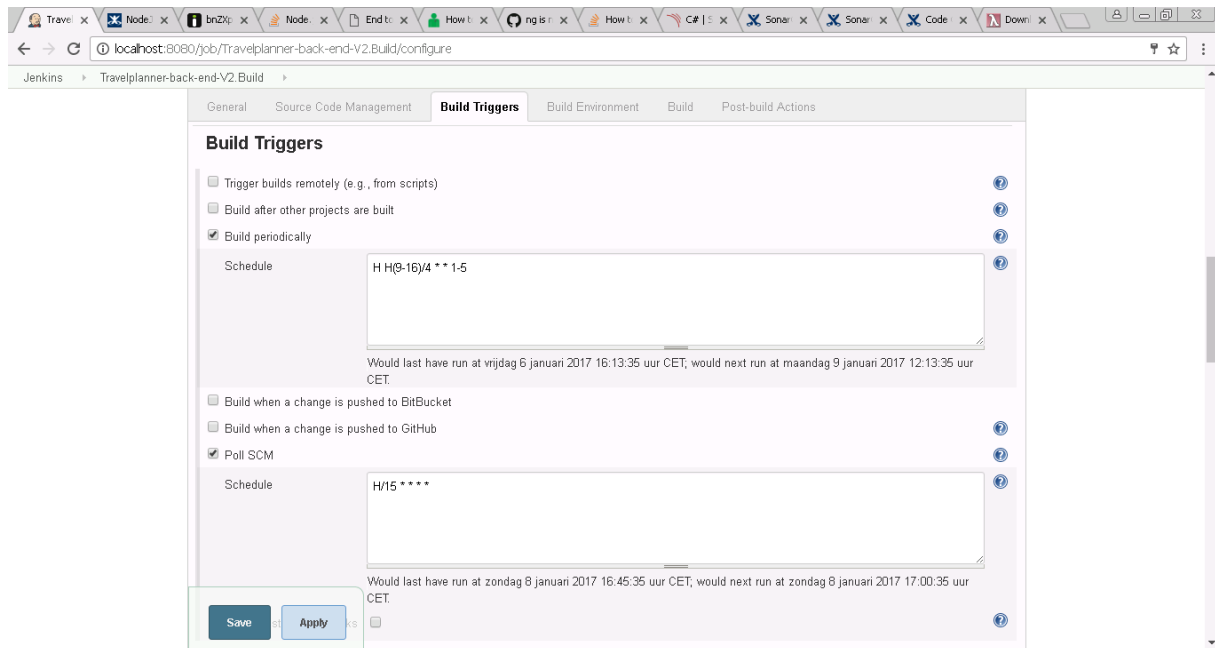
De back-end beschrijving en de naam is in de general configuratie tab te vinden. Dit staat hier onder uitbeeld.



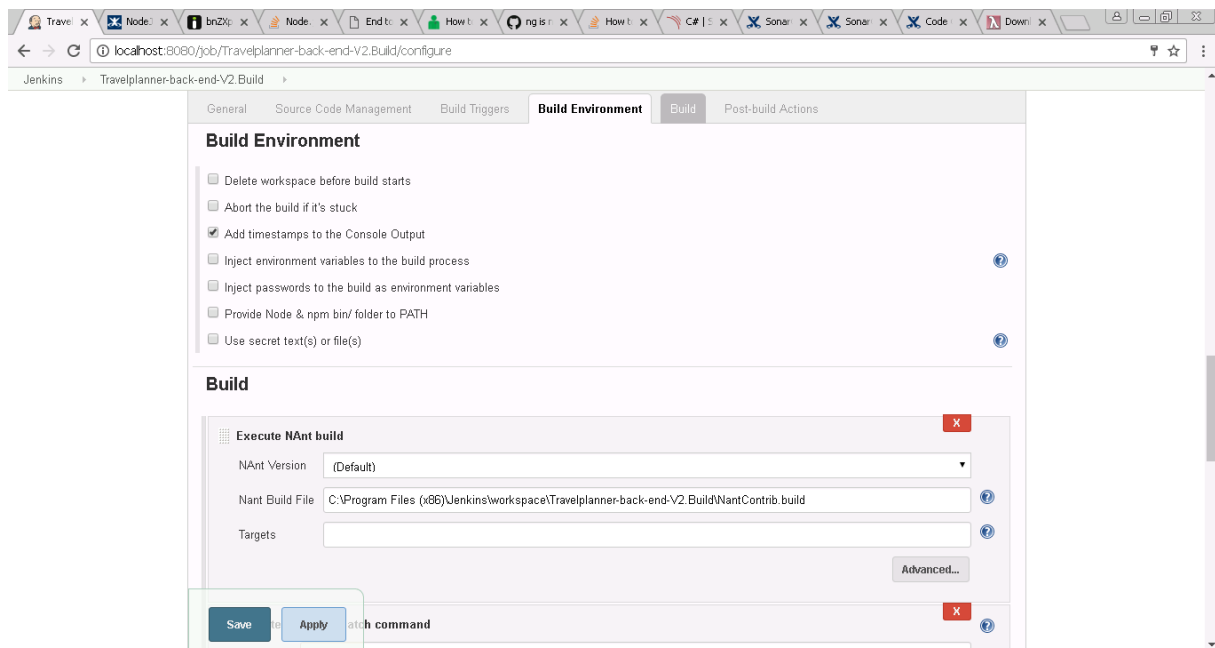
Om de Source code op te halen is er gebruik gemaakt van de server van 42Windmills. Hier staat de GIT repository url aangegeven en het username en password. We gebruiken de Develop branche om vanaf te pullen.



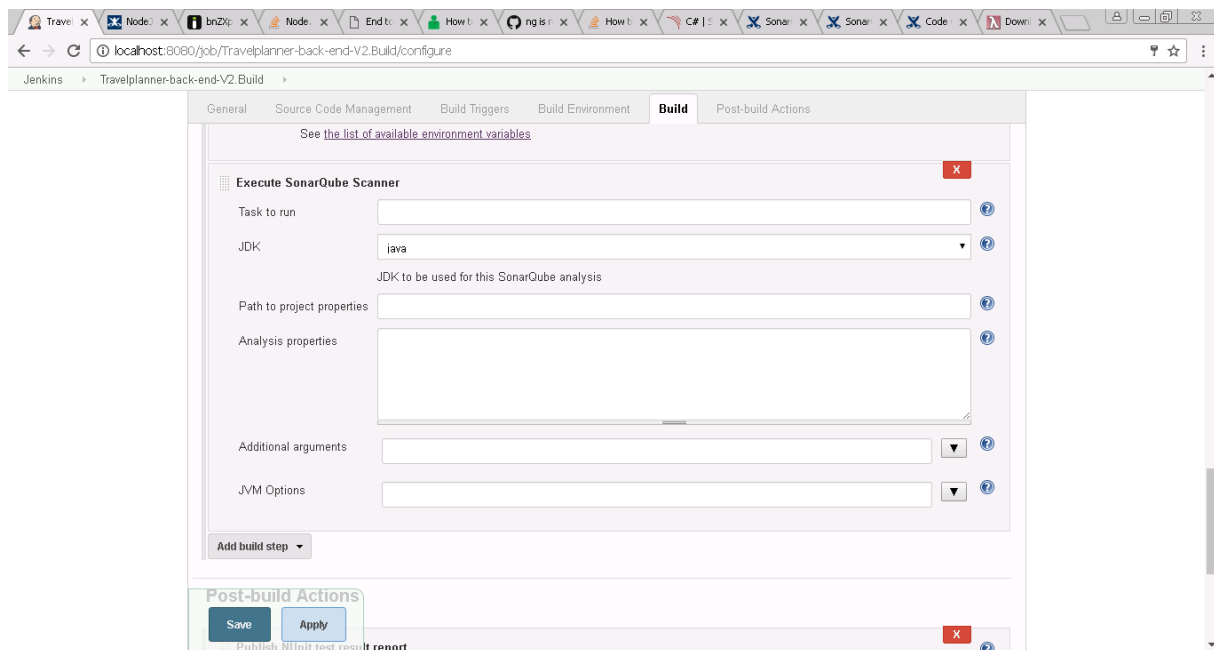
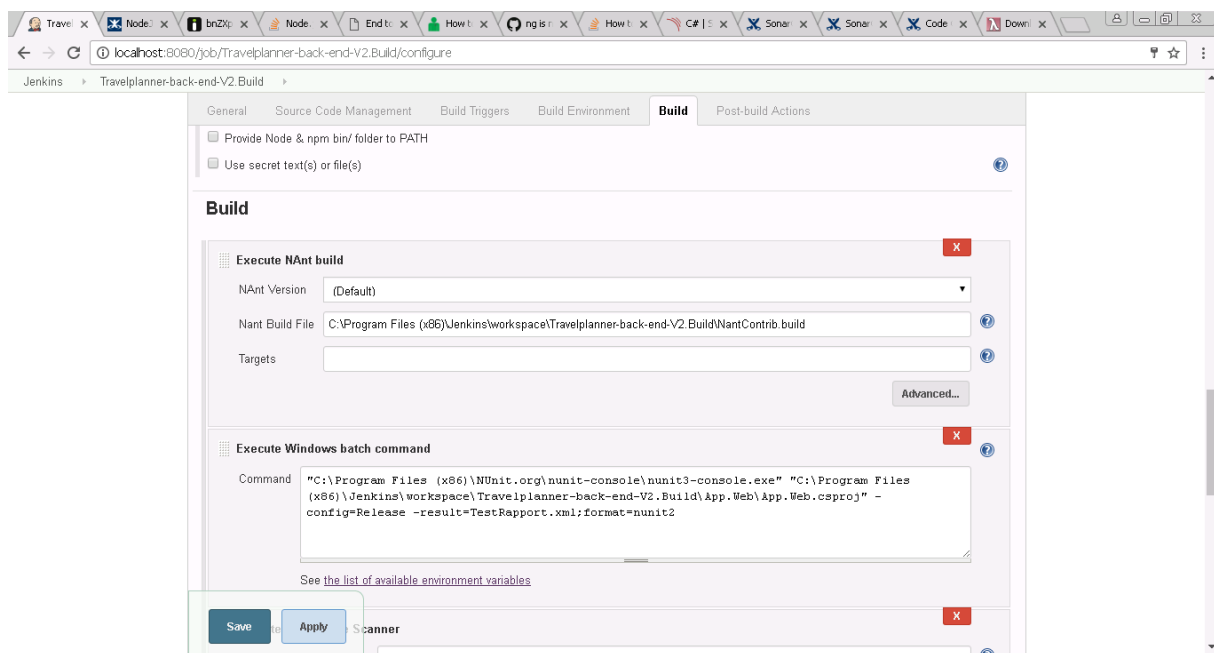
De build triggers geven aan wanneer de build moet plaatsvinden. We hebben in de back-end code gekozen om deze periodiek te laten bouwen. De syntax is te vinden op de Jenkins website wanneer hier een aanpassing nodig is. Om te controleren of er een verandering heeft plaatsgevonden wordt er door Jenkins om de 15 minuten een poll gestuurd naar de 42Windmills server.



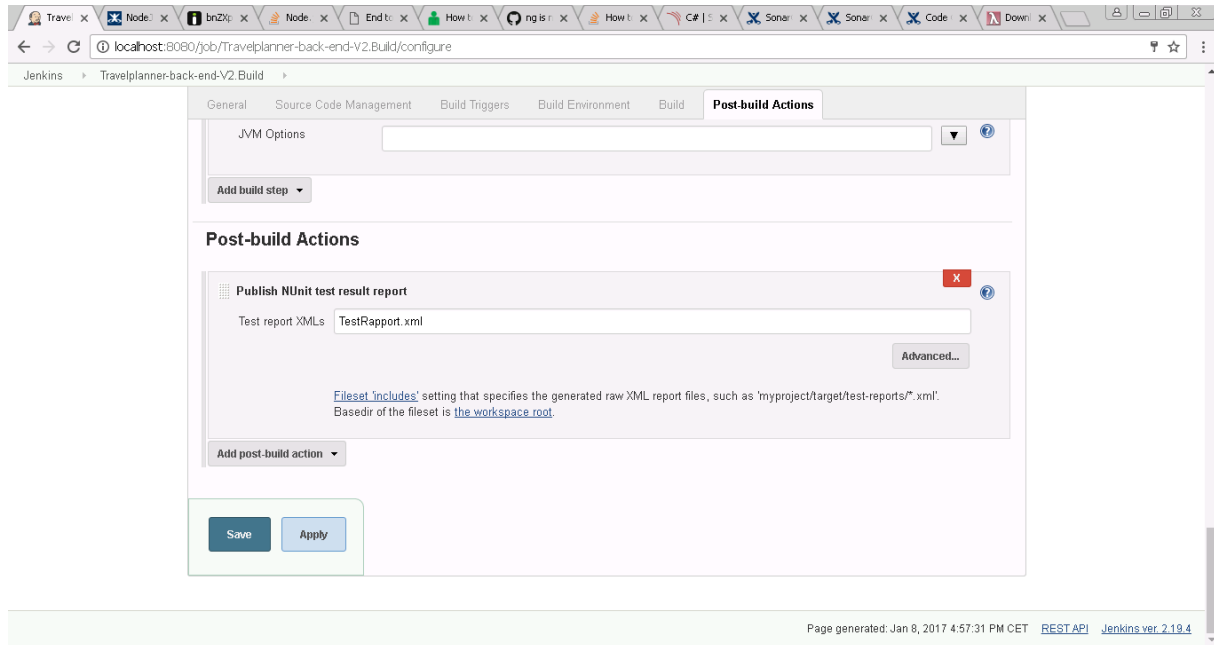
In de build omgeving wordt een time stamp toegevoegd om zo gedetailleerdere logs te bekijken en zo sneller fouten op te sporen.



De build wordt gedaan door een NAntContrib.build bestand. Onder het kopje Execute NAnt build kan deze ingesteld worden door het pad aan te geven waar het bestand zich bevindt. Daarnaast wordt er een Windows batch commando uitgevoerd om de NUnit tests uit te voeren. In dit commando staat beschreven waar Nunit-console staat, waar de tests staan en welke build soort getest moet worden. Vervolgens is aangegeven in welk formaat het rapport wordt gegenereerd.



Als laatst wordt het NUnit testrapport uitgelezen door het NUnit plug-in systeem. Hier geef je in Jenkins aan hoe het rapport heet en wat het pad is. Dit bestand staat in het project dus is het niet vereist om het pad op te geven.



## 6.2 NAntContrib

### 6.2.1 Gebruik

NAntContrib is een project voor taken en tools die het niet in de NAnt distributie hebben gehaald voor wat voor rede dan ook. NAnt kan gebruikt worden om C# projecten te bouwen. NAntContrib heeft hier enkele toegevoegde functies om MSBUILD direct aan te roepen, waarbij een C# Solution opgebouwd kan worden. Hieronder staat een uitleg over het gebruik van NAntContrib.

1. Gebruik de <loadtasks> taak in de build files om aan te geven welke assembly voor gescand moet worden.

#### Bijvoorbeeld:

```
<project name="NAntContrib" default="test">
  <target name="test">
    <loadtasks assembly="c:/nantcontrib-
0.85/bin/NAnt.Contrib.Tasks.dll" />
    ...
  </target>
</project>
```

2. Om te voorkomen dat alle build bestanden worden geüpdatet wanneer je de ene NAnt.Contrib.Tasks van plaats veranderd in een andere directory, kan je de

directory registreren welke de NAnt.Contrib.Tasks assembly bevat in een omgeving variabele en vervolgens gebruik maken van die omgevingsvariabele in de build files.

### Bijvoorbeeld:

```
<project name="NAntContrib" default="test">
  <target name="test">
    <loadtasks assembly="${path::combine(environment::get-
variable('NANTCONTRIB_DIR'), 'bin/NAnt.Contrib.Tasks.dll')}" />
    ...
  </target>
</project>
```

In het bovenstaande voorbeeld wordt de omgevingsvariabele "NANTCONTRIB\_DIR" gebruikt om het pad op te halen van de build bestanden.

- Als laatst kunnen er meerdere frameworks worden geconfigureerd. In het project wat wij gebruiken was dit nodig om niet het default framework te gebruiken maar om 4.5.2 te gebruiken. Hier onder is een voorbeeld waarbij het .net 1.1 framework gebruikt wordt.

### voorbeeld

```
<framework
  name="net-1.1"
  family="net"
  version="1.1"
  description="Microsoft .NET Framework 1.1"
  ....
>
  <task-assemblies>
    ...
    <include name="c:/nantcontrib-0.85/bin/NAnt.Contrib.Tasks.dll"
  />
  ...
  </task-assemblies>
  ....
</framework>
```

Na het instellen van de configuratie file dient alleen het commando "nant" uitgevoerd te worden in de cmd van Windows. Dit zorgt dat NAnt opzoek gaat naar .build bestanden en wanneer deze gevonden wordt, begint NAnt met het opbouwen van de solution of projecten.



## 6.2.2 Configuratie

Dit hoofdstuk geeft een configuratie van het project. Wat gedaan dient te worden om het project op te bouwen door middel van het gebruik van NAnt. Het enige wat in principe is bijgevoegd aan het project is het volgende bestand met de onderstaande configuratie:

```
<project name="TravelPlanner" default="build" basedir=".">

  <target name="build" description="Compiles the .Net solution">

    <description>The back end of the Travelplanner NANTContrib build with MSBuild
    call to build solution.</description>

    <property name="build-version" value="{1}" />

    <!-- build the solution -->

    <echo message="Building ${project::get-name()} v${1}" />

    <!-- need nant.contrib.tasks.dll for msbuild -->

    <loadtasks assembly="C:\Program Files (x86)\nantcontrib-
0.92\bin\NAnt.Contrib.Tasks.dll" />

    <echo message="-----Building TravelPlanner.sln-----" />

    <msbuild project="TravelPlanner.sln">

      <arg value="/p:Configuration=release" />

      <arg value="/p:Platform=Any CPU" />

      <arg value="/t:Rebuild" />

    </msbuild>

  </target>

</project>
```

Omdat de basedir een "." Bevat gaat NAnt zoeken in de huidige directory waar het bestand zich bevindt. Hier kan NAnt het msbuild project TravelPlanner.sln vinden. Door het gebruik van de solution file kan NAnt alle projecten opbouwen die zich in de solution file bevinden en hoeven de projecten niet apart opgesomd te worden in de configuratie file.

De target name geeft de naam aan wat uitgevoerd moet worden en de build versie geeft de huidige versie aan. In de msbuild tasks worden vervolgens de release versie aangegeven om te bepalen welke configuratie gebuild moet worden en welke cpu's gebruikt moeten worden.

## 6.3 NUnit

### 6.3.1 Gebruik

NUnit zorgt ervoor dat de tests geautomatiseerd uitgevoerd kunnen worden. Om dit werkend te maken zal in kan deze automatisch voor de betreffende projecten worden. Om dit goed te laten voorlopen moet het NUnit framework gebruikt worden bij het aanmaken van tests. Dit wordt gedaan door de test klas "[TestFixture]" mee te geven en de betreffende test methoden moet "[Test]" boven komen te staan. Dit zorgt voor herkenning van de NUnit Engine zodat de tests automatisch uitgevoerd kunnen worden door de NUnit Runner.

### 6.3.2 Configuratie

Bij het aanroepen van de tests worden modules NUnit Engine en NUnit runner aangeroepen. Visual Studio zal de installatie moeten uitvoeren door het gebruik van de Nuget package manager om het NUnit framework, en de NUnit Engine en NUnit Console Runner te downloaden en installeren op het betreffende test project. Om te zorgen dat deze modules werken dienen de modules over de zelfde versie te beschikken, namelijk versie 3. Voor de rest hoeft er geen specifieke configuratie plaats te vinden.

## 6.4 Git en Github

### 6.4.1 Gebruik

Wanneer een nieuwe feature is ontwikkeld door de ontwikkelaar van de front-end zal deze gecommited 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. Schrijf in de issuelijst de oplossing bij een eventueel opgeloste issue

### 6.4.2 Configuratie

Tijdens het ontwikkelen is het belangrijk dat in de juiste branche wordt gewerkt, namelijk de develop branche. Dit zodat aan het einde van de dag de develop branche gemerged kan worden met de master branche. In de git configuratie moet dan naar de repository URL <https://github.com/Phillywonka/travel-planner> gepushed worden.

## 6.5 Git en de 42Windmills server

### 6.5.1 Gebruik

Wanneer een nieuwe feature is ontwikkeld door de ontwikkelaar van de back-end zal deze gecommit 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

### 6.5.2 Configuratie

Voor elke dag wordt er een nieuwe branche aangemaakt waarin de functies worden toegevoegd. De betreffende URL om naar te pushen betreft

[https://fortytwowindmills.visualstudio.com/\\_git/TravelPlanner4](https://fortytwowindmills.visualstudio.com/_git/TravelPlanner4)

## 6.6 SonarQube

### 6.6.1 Gebruik

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.

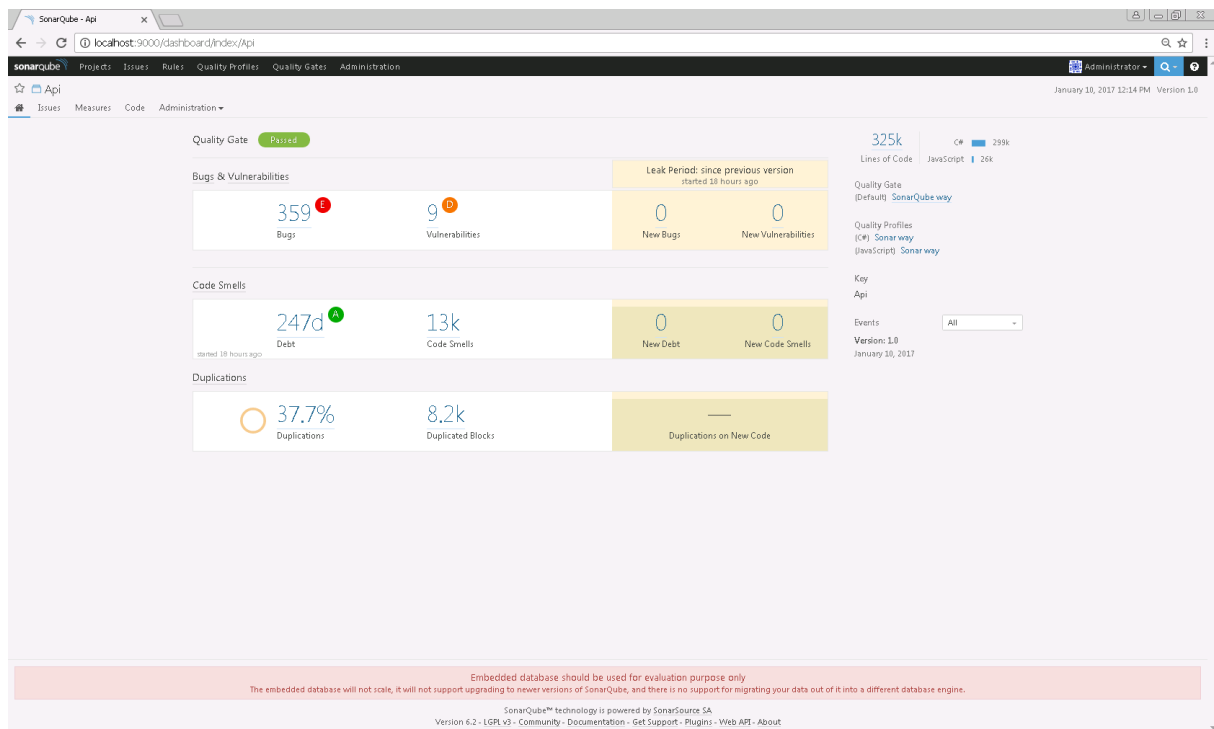
Dit overzicht wordt geopend door op de SonarQube link te klikken, dit opent het dashboard van SonarQube samen met de analyse van het huidige project.

The screenshot shows the Jenkins web interface for the project 'Travelplanner-back-end-V2.Build'. The interface is divided into several sections:

- Left Sidebar:** Contains navigation links such as 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', 'Git Polling Log', 'Move', and 'SonarQube'.
- Main Content Area:**
  - Project Travelplanner-back-end-V2.Build:** Displays the project name and a link to 'edit description'.
  - SonarQube:** A link to the SonarQube dashboard, highlighted with a blue arrow.
  - Workspace:** A link to the workspace, highlighted with a blue arrow.
  - Recent Changes:** A link to recent changes, highlighted with a blue arrow.
  - Latest Test Result:** A link to the latest test result, showing 'no failures'.
  - SonarQube Quality Gate:** Shows the status of the quality gate, with 'Api OK' and 'server-side processing: Success'.
  - Permalinks:** A list of links to various build states, including 'Last build', 'Last stable build', 'Last successful build', 'Last failed build', 'Last unsuccessful build', and 'Last completed build'.
- Test Result Trend:** A graph showing the trend of test results over time, with a 'count' on the y-axis and build numbers on the x-axis.

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.

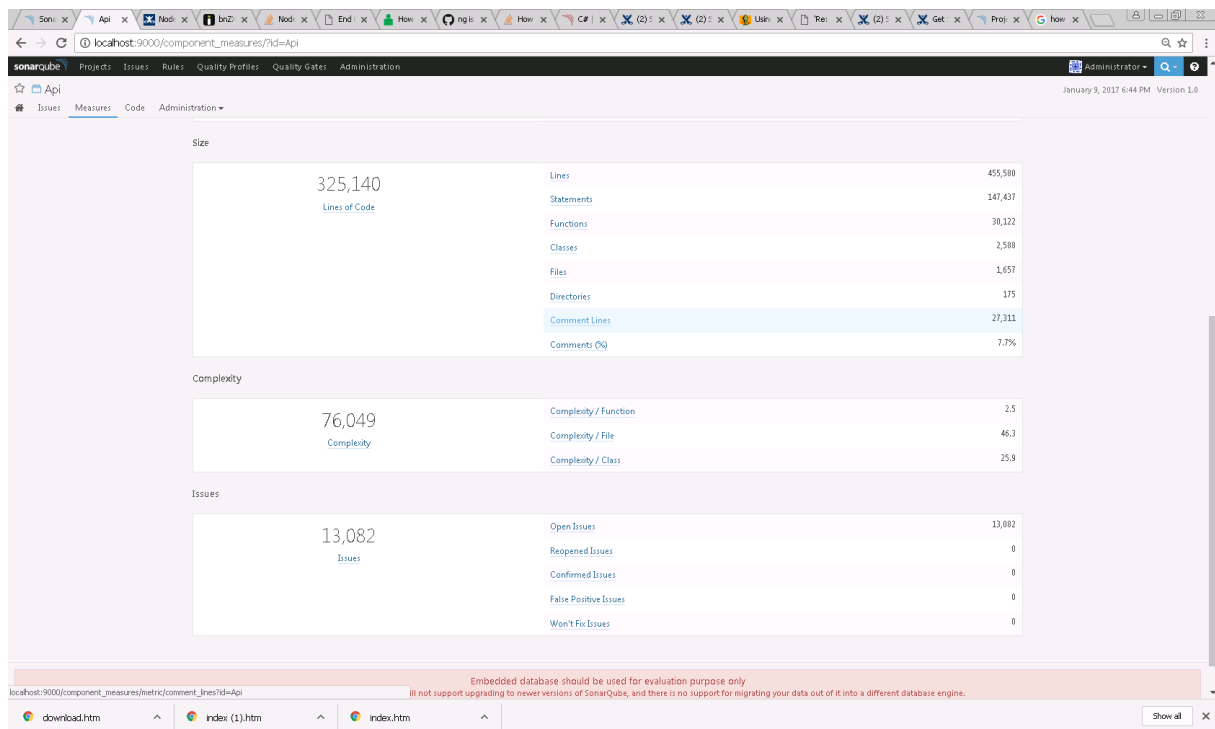
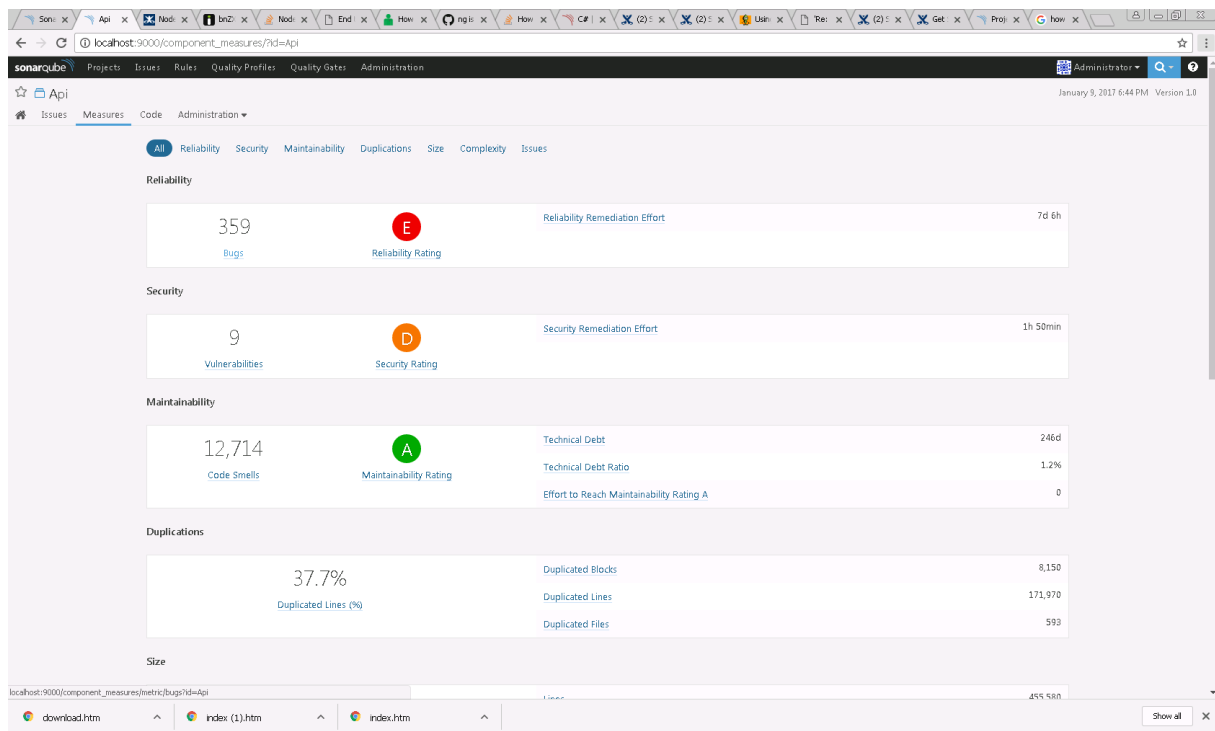


We zien hier een aantal statistieken die de analyses heeft waargenomen. De statistieken worden hier kort behandeld.

### 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.

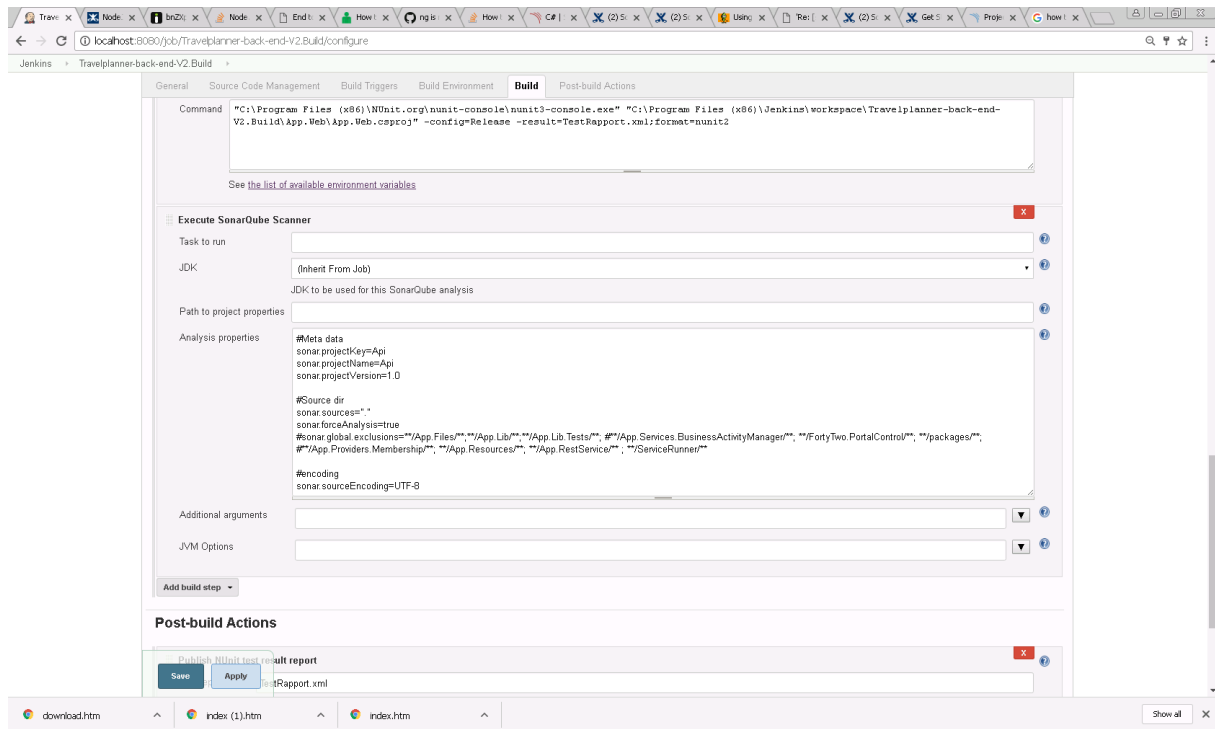
Als laatst behandelen we de issues lijst in SonarQube, welke hieronder is weergegeven.

The screenshot shows the SonarQube web interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. The main content area is titled 'Issues' and shows a list of issues for the project 'Api'. The issues are sorted by creation date. The table lists various issues, including 'Use a literal instead of the Object constructor', 'Remove the literal 'true' boolean value', and 'Rename class 'newBookingFactory' to match camel case naming rules'. Each issue entry includes a checkbox, a description, a severity level (e.g., Minor, Major), and an effort estimate (e.g., 5min, 15min, 28min). The sidebar on the left provides filters for Display Mode (Issues, Measures, Code, Administration), Type (Bug, Vulnerability, Code Smell), and Resolution (Unresolved, Fixed, False Positive, Won't fix). The bottom of the interface shows a list of open tabs: download.htm, index (1).htm, and index.htm.

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.

## 6.6.2 Configuratie

Alvorens een analyse wordt uitgevoerd dient SonarQube worden geconfigureerd. De belangrijkste configuratie bevindt zich in de Jenkins configuratie van het betreffende project. Hier onderen zijn de configuratie mogelijkheden weergegeven achter “Analysis properties”.



### Configuraties paramaters

- ProjectKey: De sleutel van het project (moet uniek zijn)
- projectName: De naam van het project (moet uniek zijn). Aan de hand van deze naam zal ook het project in de SonarQube dashboard weergegeven worden.
- ProjectVersion: De versie van het project. Dient gelijk te zijn aan de versie van de build.
- Sources: Pad waar het project staat dat geanalyseerd moet worden. Een “.” Betekend de huidige directory.
- forceAnalyse: forceert een analyse.
- Global.Exclusions: Bestanden die uitgesloten moeten worden van analyse. In dit voorbeeld staan er geen projecten uitgesloten omdat het weg gecommmentarieerd is.
- Encoding: In welk formaat de code gedecodeerd moet worden. In dit geval UTF-8 formaat.

Naast deze configuratie is het belangrijk de volgende configuraties op de volgende manier geconfigureerd te hebben.

### Sonar.properties (bevindt zich in de config directory van SonarQube)



#Configure here general information about the environment, such as SonarQube DB details for example

#No information about specific project should appear here

#----- Default SonarQube server

#sonar.host.url=http://localhost:9000

#----- PostgreSQL

#sonar.jdbc.url=jdbc:postgresql://localhost/sonar

#----- MySQL

#sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8

#----- Oracle

#sonar.jdbc.url=jdbc:oracle:thin:@localhost/XE

#----- Microsoft SQLServer

#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMethod=Cursor

#----- Global database settings

#sonar.jdbc.username=sonar

#sonar.jdbc.password=sonar

#----- Default source code encoding

sonar.sourceEncoding=UTF-8

#----- Security (when 'sonar.forceAuthentication' is set to 'true')

#sonar.login=admin

#sonar.password=admin

**Scanner.properties (bevindt zich in de config directory van Sonar Scanner)**

#Configure here general information about the environment, such as SonarQube DB details for example

#No information about specific project should appear here

#----- Default SonarQube server

#sonar.host.url=http://localhost:9000

#----- Default source code encoding

sonar.sourceEncoding=UTF-8

#----- Global database settings (not used for SonarQube 5.2+)

#sonar.jdbc.username=sonar

#sonar.jdbc.password=sonar

#----- PostgreSQL

#sonar.jdbc.url=jdbc:postgresql://localhost/sonar

#----- MySQL

#sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8

#----- Oracle

#sonar.jdbc.url=jdbc:oracle:thin:@localhost/XE

#----- Microsoft SQLServer

#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMethod=Cursor

Deze configuratie zorgt dat alle code die geanalyseerd wordt gedecodeerd wordt naar UTF-8 om fouten te voorkomen. De rest van de configuratie volstaat wanneer de standaard waarde gebruikt wordt. Meer informatie betreft de configuratie van SonarQube is te vinden op [www.sonarqube.org/Documentation](http://www.sonarqube.org/Documentation)

## 6.7 Issue tracking

### 6.7.1 Gebruik

Om issues bij te houden wordt een Excel document opgezet genaamd Issues.xls. Dit document zal bestaan uit de volgende gegevens:

- Jenkins build nummer
- Issue nummer
- Commit commentaar
- Omschrijving
- Opgelost
- Een omschrijving hoe het issue opgelost is

Deze dienen aangevuld te worden op elk moment wanneer een commit wordt uitgevoerd. Vervolgens wordt in Git tevens het Issuenummer opgegeven.

Naast het document worden de issues ook bijgehouden in Jenkins door de opgeslagen commit messages. Jenkins archiveert automatisch deze messages en kunnen te allen tijde worden ingezien.

## 7 CONCLUSIE

Er is een ontwikkelstraat opgezet die niet alle beoogde features die in het vooronderzoek zijn onderzocht in werking heeft gezet.

Bij NAnt is een omweg nodig geweest om de functionaliteit van NAnt te kunnen gebruiken en daarmee ook de voordelen die NAnt met zich meebrengt.

Issuetracking heeft in het ontwerp tevens niet het beoogde doel behaald. We hebben geen aparte tool in gebruik genomen omdat het project in onze ogen te kleinschalig is om hier een aparte tool voor in gebruik te nemen.

De database is niet voorzien in de acceptatieomgeving vanwege een tekort aan schijfruimte op de server. Dit zorgt voor een acceptatieomgeving die geen gebruik kan maken van de database van de applicatie. De applicatie zal opstarten en vervolgens geen connectie met de database kunnen opzetten. Echter is dit probleem opgelost wanneer de schijfruimte van de omgeving wordt vergroot.

Als laatst willen we graag vermelden dat enkele configuratie mogelijkheden niet optimaal zijn doorgevoerd. Dit heeft als oorzaak dat de volledig functionele oplevering van de server pas 2 weken na de start van het project heeft plaatsgevonden. Hierdoor was er minder tijd beschikbaar om de configuratie optimaal te configureren.

## 8 BRONNEN

- [www.jenkins.io](http://www.jenkins.io)
- [www.github.com](http://www.github.com)
- [www.sonarqube.org](http://www.sonarqube.org)
- [www.nunit.org](http://www.nunit.org)
- [www.karma.io](http://www.karma.io)
- [www.angular.org](http://www.angular.org)