**LIVING PROGRESS - BUILD – WWF**

**Deployment Guide**

Revision History

|  |  |  |
| --- | --- | --- |
| **Author** | **Revision Number** | **Date** |
| TCSCODER | 1.0 | 01/19/2017 |
| Lech Migdal | 1.0.0.1 | 01/29/2017 |
| Lech Migdal | 1.0.0.2 | 02/09/2017 |
| Lech Migdal | 1.0.0.3 | 02/27/2017 |
| Lech Migdal | 1.0.0.4 | 03/09/2017 |
| Lech Migdal | 1.0.0.5 | 03/22/2017 |
| Ward Loving | 1.0.0.6 | 04/04/2017 |

[1. Overview of components 2](#_Toc477952036)

[2. Backend Deployment Instructions 4](#_Toc477952037)

[1. Organization of Submission 4](#_Toc477952038)

[2. Application Setup 4](#_Toc477952039)

[3. Database Setup 4](#_Toc477952040)

[2. External Login Providers Configuration 8](#_Toc477952041)

[1. Google 8](#_Toc477952042)

[2. Twitter 13](#_Toc477952043)

[3. Facebook 15](#_Toc477952044)

[3. IIS Deployment Instructions 16](#_Toc477952045)

[4. Visual Studio Deployment Instructions for local development 21](#_Toc477952046)

[5. Visual Studio Publishing Instructions (on server like staging or PROD) 21](#_Toc477952047)

[6. Unit Tests Verification (for development) 21](#_Toc477952048)

[7. Web API Verification 24](#_Toc477952049)

[3. Frontend Deployment Instructions 26](#_Toc477952050)

[1. Prerequisites 26](#_Toc477952051)

[2. Installing Dependencies 26](#_Toc477952052)

[3. Configuration 26](#_Toc477952053)

[4. Test Accounts 27](#_Toc477952054)

[5. Running in dev mode (for development only) 27](#_Toc477952055)

[6. Deploying in production mode 27](#_Toc477952056)

[7. Tests (development only) 28](#_Toc477952057)

[8. Suggested upgrade approach 28](#_Toc477952058)

[4. Proxy installation 30](#_Toc477952059)

[5. Current setup 32](#_Toc477952060)

[1. Staging 32](#_Toc477952061)

# Overview of components

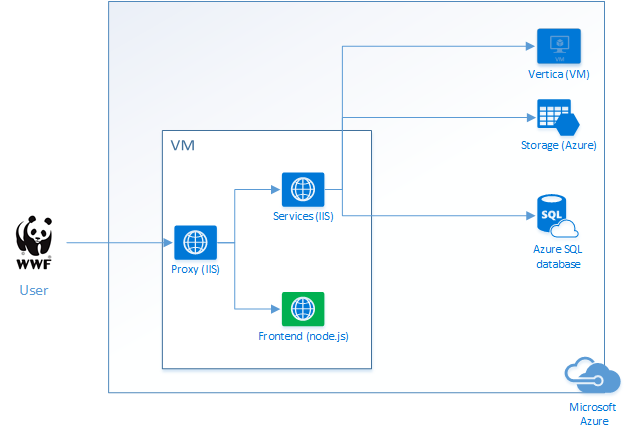


Figure 1 Detect.trade logical architecture (key components deployment wise)

# Backend Deployment Instructions

<https://github.com/topcoderinc/HP-LP-WWF-Detect-IT-Services>

## Organization of folders

* 1. Directory Structure

|  |  |
| --- | --- |
| **Directory / File Name** | **Description** |
| **Database** | Contains all required scripts to setup the database. |
| **Docs** | Contains the deployment guide, and Postman exported collection. |
| **Src** | The source code. |

## Application Setup

Preconditions (web server)

* All custom components can be downloaded with Web Platform Installer - <https://www.microsoft.com/web/downloads/platform.aspx?lang>= (5.0 at the time of writing)
* Microsoft IIS 8.5 (Optional, can also run in IIS Express)
  + Web Deploy if you want to publish application from Visual Studio (go with Recommended Package for web hosting) <https://www.iis.net/learn/install/installing-publishing-technologies/installing-and-configuring-web-deploy-on-iis-80-or-later>
* Windows OS with .NET Framework 4.5+ (Required, should be installed with Web Deploy, but it’s good to double check this)
* Open on firewall all ports that you want to expose your app through (80 and 443 are opened, but everything crazy like 8000+ may require Windows Firewall rule - <https://support.rackspace.com/how-to/managing-the-windows-server-2012-firewall/>

Database server

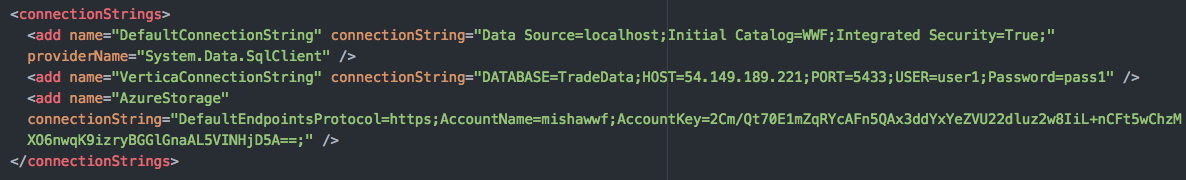
* Microsoft SQL Server 2012+ is installed. (Required)

Client

* Visual Studio 2015 is installed. (Required to build solution and run tests. For deployment in IIS – optional for publishing)

## Database Setup

This application is backed up by two different database servers: MS SQL Server and Vertica. These are configured in the web.config file as shown below:



The 1st connection string is for SQL Server, it stores data like alerts, aggregated reports etc… which are calculated from the raw data from Vertica.

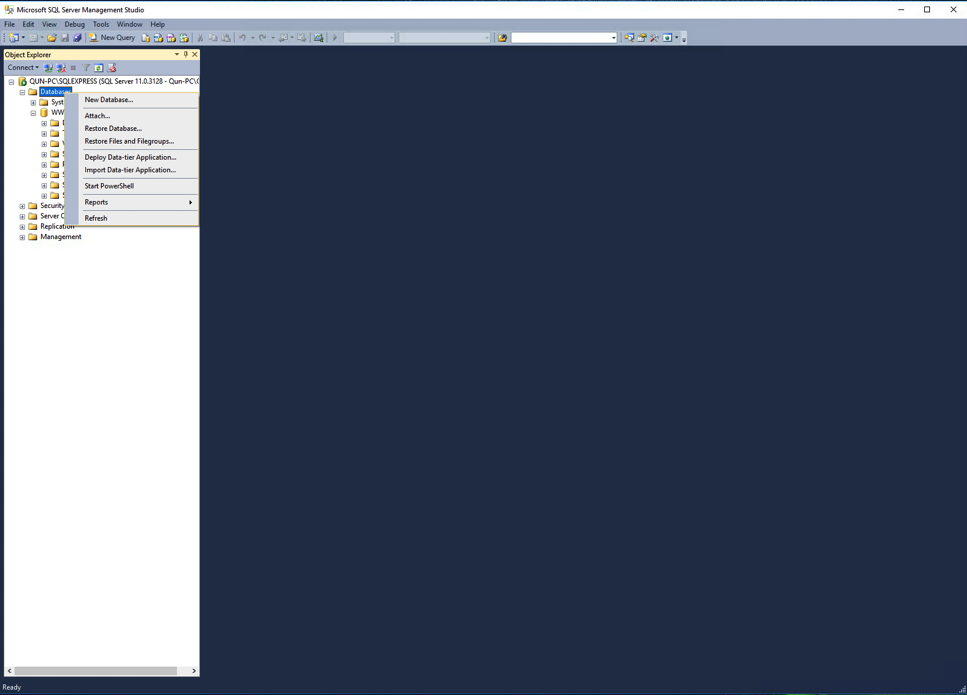
The 2nd connection string is for Vertica, which stores the raw trade data.

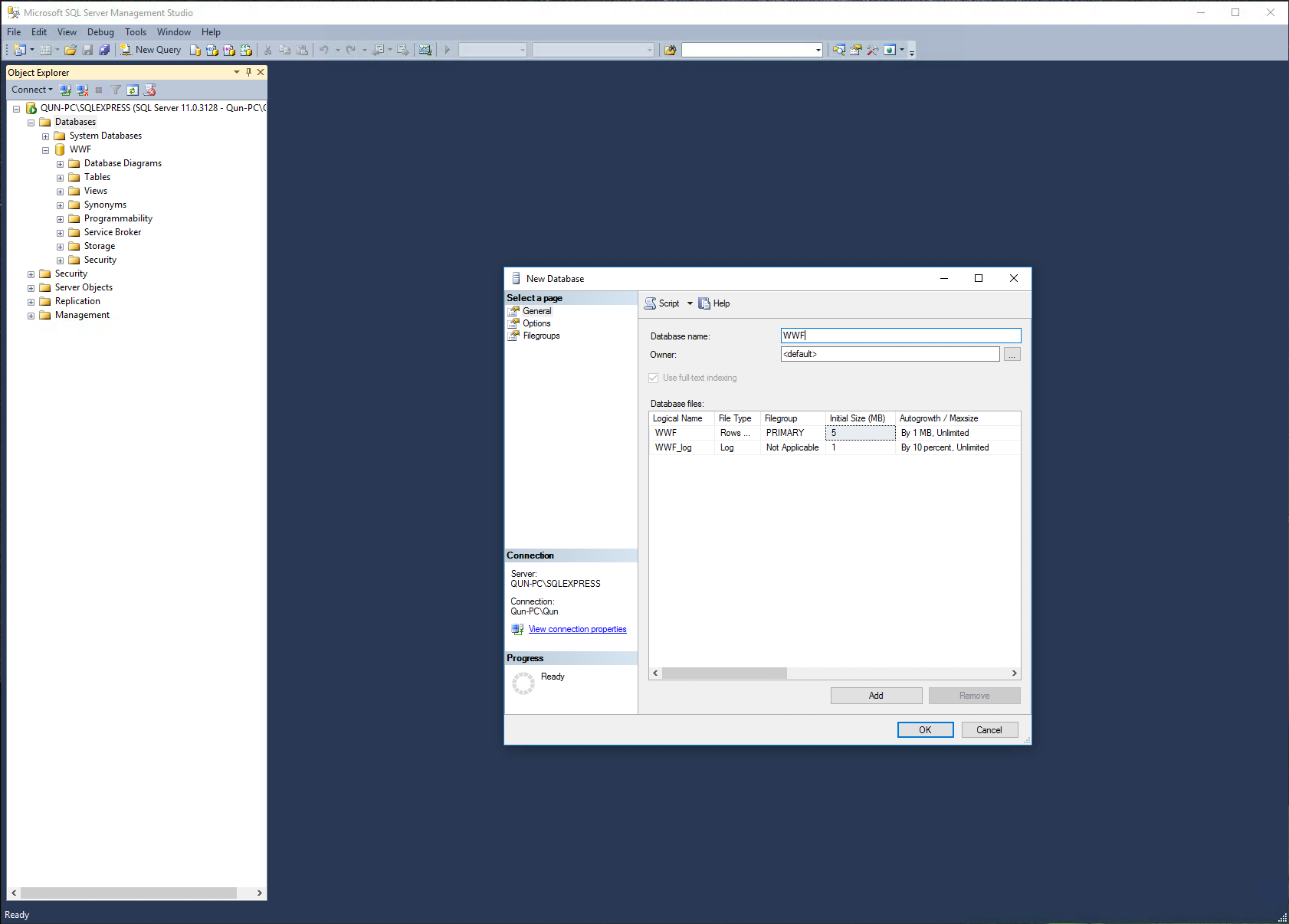
This application reads data from Vertica database and writes calculated data into the SQL Server which the application also reads data from to display on our frontend pages.

* 1. MS SQL Server

Basically all you need to do is to Install the scripts under **Database** directory.

1. Create database using ***SQL Server Management Studio*** (default configuration points to **WWF** database on a localhost)

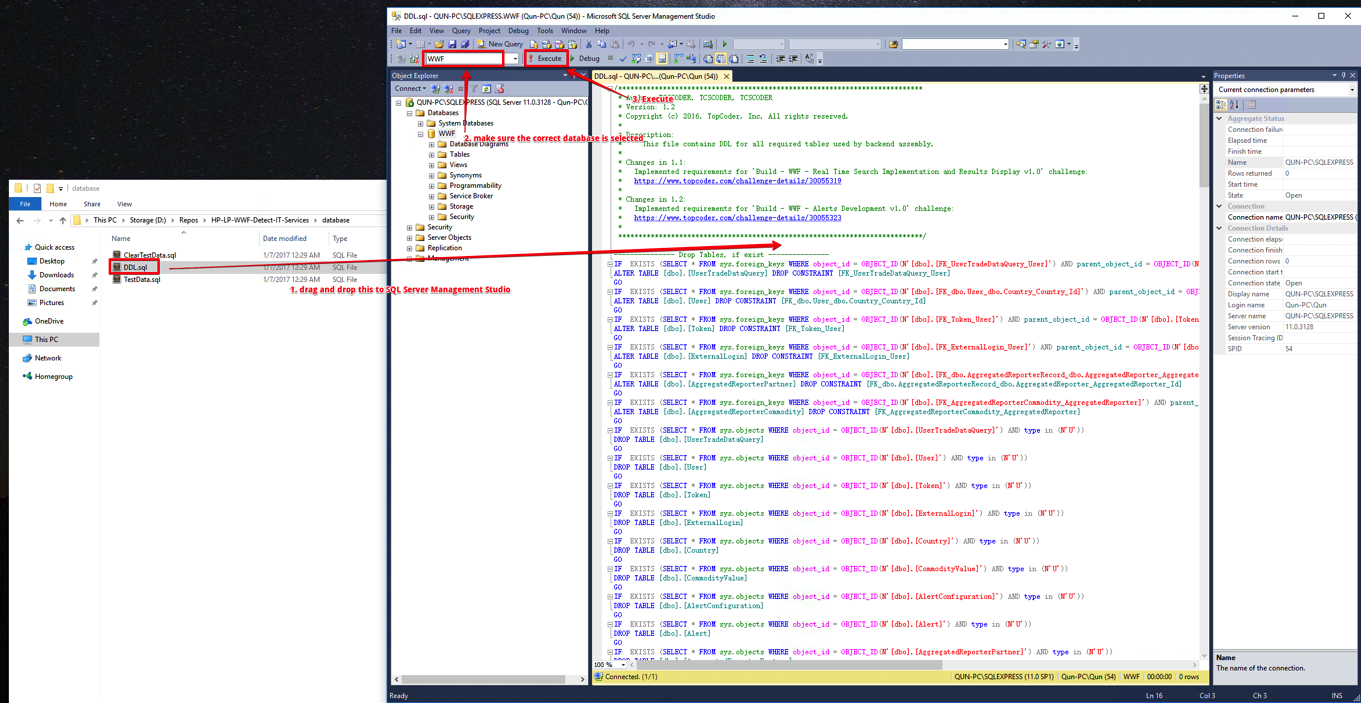




1. Once you have the database created, you can just execute the following scripts in order:

* DDL.sql
* [optionally] TestData.sql – required to verify Web API with Postman.

You can simply drag and drop the script file into SQL Server Management Studio, and then execute it:



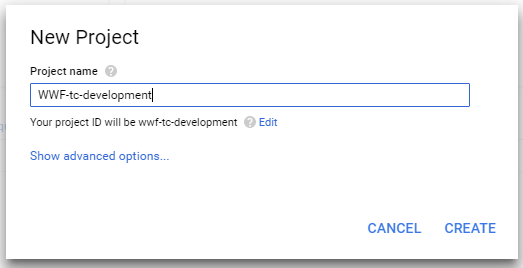
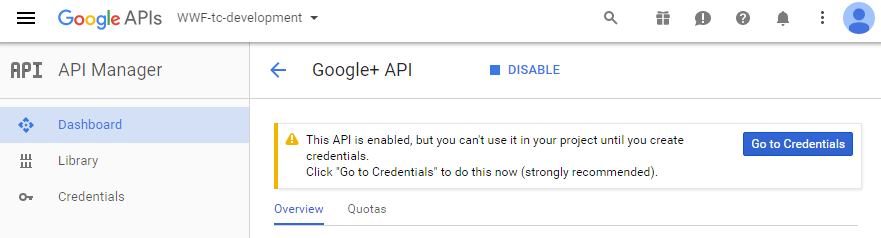
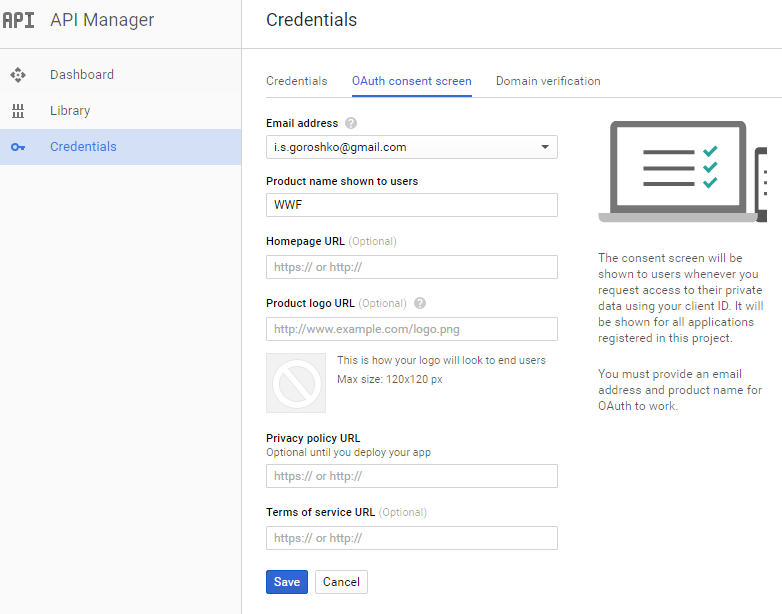
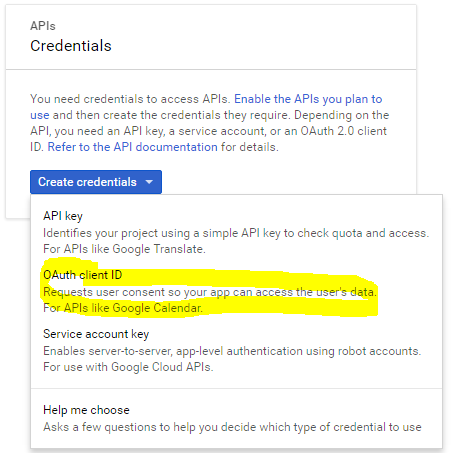
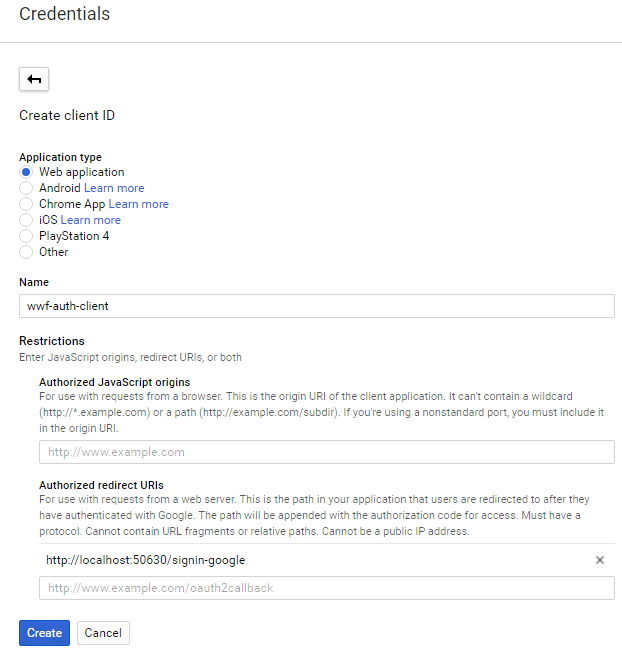
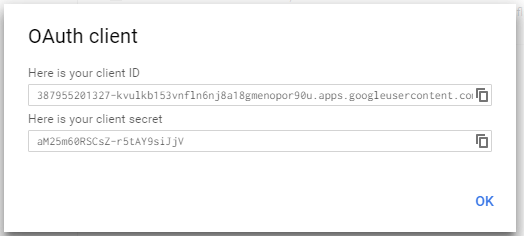
* 1. Vertica

This is the database server that stores the raw trade data we use as input to generate alerts, aggregated reports, charts, etc…

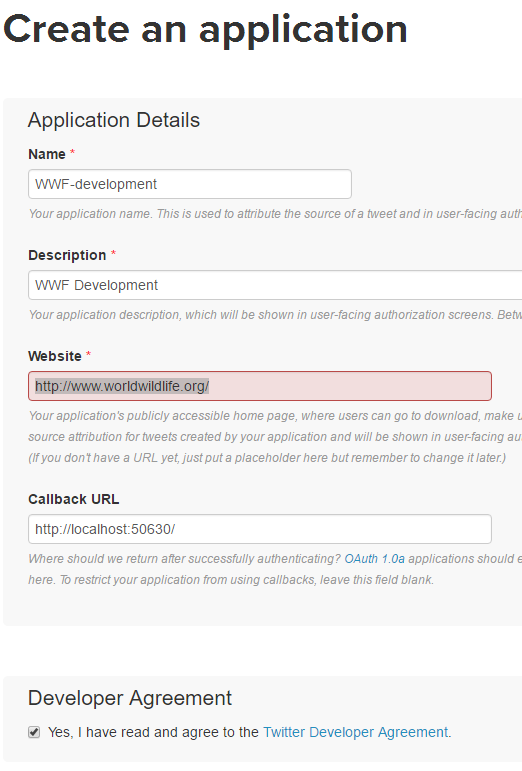
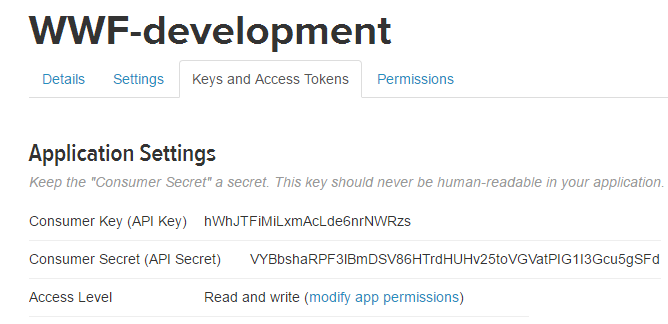
# External Login Providers Configuration

In order to enable authentication using external login providers, you should first register new application and obtain Key and Secret. These values are set in **appSettings** section of the application configuration file.

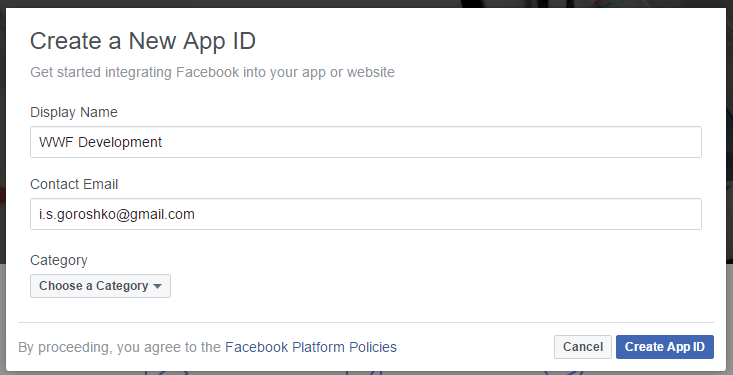
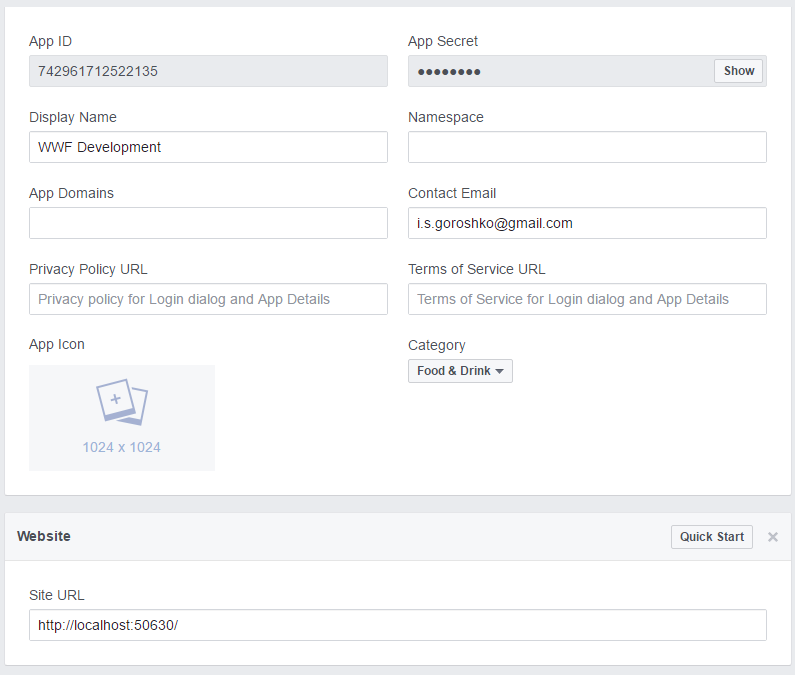
## Google

* Navigate to <https://console.developers.google.com>
* Create new project  
  
* Once project is created, navigate to project Dashboard and click Enable API. In the list of APIs select 'Google+ API', then in an opened page click 'Enable'. You should see following screen:  
  
* Select 'Credentials' tab, and switch to 'OAuth consent screen' sub-menu. Enter product name and click Save.  
  
* Now, in the Credentials sub-menu click \*\*Create Credentials\*\* and choose \*\*OAuth client ID\*\*  
  
* In the opened page, select \*\*Web application\*\* as Application Type. Enter name and authorized redirect url, it should have following template \*rest\_api\_host/signin-google\*, see example below for localhost deployment:  
  
* Once credentials are created, you should see popup with client ID and client secret:  
  

## Twitter

* Navigate to <https://apps.twitter.com/app/new>  
  
* Click \*\*Create New App\*\*.
* Fill in the details and create the app. The \*\*Callback URL\*\* should match the host URL of REST API.
* Switch to \*\*Keys and Access Tokens\*\* tab, and there you can find API Key and Secret:  
  

## Facebook

* Navigate to https://developers.facebook.com/ (make sure you are logged in)
* Click \*\*My Apps -> Add a New App\*\*.
* Fill in the app details:  
  
* Once new app is created, navigate to \*\*Settings\*\* tab using navigation bar on the left side.
* Click \*\*Add Platform\*\*
* Choose \*\*Website\*\*
* Enter Site URL in an added Website item and click Save. Note that Site URL should match Web API host address.  
  
* You can now copy App Id and App Secret into configuration file.

# IIS Deployment Instructions

1. Install Dynamic Compression Module (from WebDeploy)

<https://technet.microsoft.com/en-us/library/cc753681(v=ws.10).aspx>

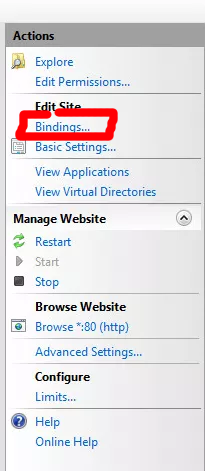
1. Change Error Messages in the Backend App to Custom Error Messages (so that details aren’t rendered to end users)
2. Configure environment variables (<https://lsdbtech.wordpress.com/2015/05/28/windows-server-2012-r2-change-environment-variables/>) described here <https://github.com/topcoderinc/HP-LP-WWF-Detect-IT-Services> to match your specific environment needs. Then you don’t need to manage them inside the web.config file every time you do push.
   1. For recovery purposes after setting all of the parameters, export them to a flat file and store on a github in a repository (different than the source code repo). Run the following from the command line (as administrator)

SET > c:\env-variables-staging.txt

Note that you wont be able to automatically import this file into a new instance of windows. You will need to format this file a bit to in the end get the following structure:

SETX –m $PARAMETER$ $VALUE$

1. Customize the port on which backend is running like described on this page <http://www.vladsitblog.com/change-iis-default-port-in-iis-8-manager/> (Edit Site -> Bindings is the menu you are looking for)



For staging backend service is running on port 8080. Rewrite from 80 and 443 is handled by Proxy service.

1. You can overwrite some configuration in src/WWF/web.config by setting environment variables, if the environment variable is unset, then the default values in web.config will be used, the environment variable will be read once this app starts. The value of the environment variable must set properly according to its type. An invalid value might cause an exception when starting the application.

The following environment variables are currently available for system setup:

|  |  |  |  |
| --- | --- | --- | --- |
| **Environment variable** | **Configuration field to update** | **Value Type** | **Description** |
| WWFB\_GOOGLE\_CLIENT\_ID | appSettings[key=GoogleClientId].value | String | The Google client id |
| WWFB\_GOOGLE\_CLIENT\_SECRET | appSettings[key=GoogleClientSecret].value | String | The Google client secret. |
| WWFB\_FACEBOOK\_APP\_ID | appSettings[key=FacebookAppId].value | String | The Facebook app id |
| WWFB\_FACEBOOK\_APP\_SECRET | appSettings[key=FacebookAppSecret].value | String | The Facebook app secret |
| WWFB\_TWITTER\_CONSUMER\_KEY | appSettings[key=TwitterConsumerKey].value | String | The Twitter consumer id |
| WWFB\_TWITTER\_CONSUMER\_SECRET | appSettings[key=TwitterConsumerSecret].value | String | The Twitter consumer secret |
| WWFB\_AZURE\_BLOB\_NAME | appSettings[key=AzureBlobName].value | String | The Azure Blob's name |
| WWFB\_AZURE\_STORAGE\_NAME | appSettings[key=AzureStorageName].value | String | The Azure Storage's name |
| WWFB\_DEFAULT\_CONNECTION\_STRING | connectionStrings[name=DefaultConnectionString].connectionString | String | Default database connection string |
| WWFB\_VERTICA\_CONNECTION\_STRING | connectionStrings[name=VerticaConnectionString].connectionString | String | Vertica database connection string |
| WWFB\_AZURE\_STORAGE | connectionStrings[name=AzureStorage].connectionString | String | Azure Storage connection string |
| WWFB\_SMTP\_DELIVERY\_METHOD | system.net.mailSettings.smtp.deliveryMethod | Enum, values:  Network | PickupDirectoryFromIis | SpecifiedPickupDirectory | SMTP delivery method |
| WWFB\_SMTP\_FROM | system.net.mailSettings.smtp.from | String | The from field of email |
| WWFB\_SMTP\_HOST | system.net.mailSettings.smtp.network.host | String | SMTP service host |
| WWFB\_SMTP\_PORT | system.net.mailSettings.smtp.network.port | Int32 | SMTP service port |
| WWFB\_SMTP\_ENABLE\_SSL | system.net.mailSettings.smtp.network.enableSsl | Boolean | Whether enable ssl connection of SMTP or not |
| WWFB\_SMTP\_USER\_NAME | system.net.mailSettings.smtp.network.userName | String | UserName of SMTP account |
| WWFB\_SMTP\_PASSWORD | system.net.mailSettings.smtp.network.password | String | Password of SMTP account |

1. For all parameters that are not handled via Environment Variables, please change the **Web.config** in your Visual Studio instance (**\src\WWF\** directory) before pushing the code to the server. Make it fits your environment. Please make sure you configure External Login Provider, most of other properties have default configuration values and should work without any modifications.

|  |  |
| --- | --- |
| **Node name** | **Description** |
| <connectionStrings> | Connection strings to database and Azure storage. |
| <appSettings> | Contains External Login Providers configuration and azure storage/blob names. |
| <unity > | Contains all configuration values that will be injected to services. |
| <log4net> | Log4net configuration. |

***BasePersistenceService*** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| Logger | The log4net ILog instance. | Yes |
| ConnectionString | The persistence connection string. | Yes |
| VerticaConnectionString | The connection string to Vertica DB. | No |

**LookupService** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| *all properties from the BasePersistenceService class* | | |

**UserService** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| *all properties from the BasePersistenceService class* | | |
| PasswordResetEmailSubject | The subject for password reset notification email. | Yes |
| PasswordResetEmailBodyTemplate | The body template for password reset notification email. | Yes |
| TokenExpiration | the token expiration time in minutes. | Yes |

**EmailService** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| Logger | The log4net ILog instance. | Yes |
| SmtpClient | The SmtpClient used for sending emails. | Yes |
| FromAddress | The 'From' email address used for sending email notifications. | Yes |

**LookupController** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| LookupService | The Lookup service. | Yes |

**UserController** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| UserService | The User service to manage users and security operations. | Yes |

**TradeDataService** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| *all properties from the BasePersistenceService class* | | |

**DashboardService** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| *all properties from the BasePersistenceService class* | | |

**ContentfulService** configuration properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | | **Required** |
| *all properties from the BasePersistenceService class* | | | |
| ApiUrl | The contentful API URL. | Yes | |
| OAuthToken | The contentful OAuth token. | Yes | |
| SpaceId | The contentful space id. | Yes | |

**TradeDataController** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| TradeDataService | The Trade Data service. | Yes |

**DashboardController** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| DashboardService | The Dashboard service. | Yes |

**ContentfulController** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| ContentfulService | The Contentful service. | Yes |

**AuthorizationFilter** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| Logger | The log4net ILog instance. | Yes |
| UserService | The User service. | Yes |

**ExceptionFilter** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| Logger | The log4net ILog instance. | Yes |

**LoggingActionFilter** configuration properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Required** |
| Logger | The log4net ILog instance. | Yes |

Sample **external login providers** configuration. In case you want to use these values, make sure you host your backend WEB API on **http://localhost:50630/**:

<appSettings>

<add key="GoogleClientId" value="307492493635-03007v505jia4rkbtchi90in0j6668ap.apps.googleusercontent.com" />

<add key="GoogleClientSecret" value="huirCOj2OVcRcErZyX5LNDoF" />

<add key="FacebookAppId" value="742961712522135" />

<add key="FacebookAppSecret" value="195dca355731b31a03287ecc01a9812b" />

<add key="TwitterConsumerKey" value="hWhJTFiMiLxmAcLde6nrNWRzs" />

<add key="TwitterConsumerSecret" value="VYBbshaRPF3lBmDSV86HTrdHUHv25toVGVatPIG1I3Gcu5gSFd" />

</appSettings>

Sample **log4net** configuration:

<log4net>

<appender name="FileAppender" type="log4net.Appender.FileAppender">

<file value="weblog.txt" />

<appendToFile value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger - %message%newline" />

</layout>

</appender>

<logger name="default">

<level value="DEBUG" />

<appender-ref ref="FileAppender" />

</logger>

</log4net>

Sample SMTP configuration:

<system.net>

<mailSettings>

<smtp deliveryMethod="SpecifiedPickupDirectory">

<specifiedPickupDirectory pickupDirectoryLocation="d:\tc2016\temp\emails\"/>

</smtp>

</mailSettings>

</system.net>

Sample Azure storage configuration:

<appSettings>

<add key="AzureBlobName" value="wwf-blob" />

<add key="AzureStorageName" value="mishawwf" />

</appSettings>

<connectionStrings>

<add name="AzureStorage" connectionString="DefaultEndpointsProtocol=https;AccountName=mishawwf;AccountKey=2Cm/Qt70E1mZqRYcAFn5QAx3ddYxYeZVU22dluz2w8IiL+nCFt5wChzMXO6nwqK9izryBGGlGnaAL5VINHjD5A==;"/>

</connectionStrings>

Sample Unity configuration:

<unity>

*{type aliases removed for clarity}*

<containers>

<container>

<register type="IEmailService" mapTo="EmailService">

<property name="Logger" />

<property name="SmtpClient" />

<property name="FromAddress" value="your\_address@gmail.com" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="ILookupService" mapTo="LookupService">

<property name="ConnectionString" value="DefaultConnectionString" />

<property name="Logger" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="IUserService" mapTo="UserService">

<property name="PasswordResetEmailSubject" value="Password reset link." />

<property name="PasswordResetEmailBodyTemplate" value="Hi %FirstName%, here is a link to reset your password." />

<property name="TokenExpiration" value="60" />

<property name="ConnectionString" value="DefaultConnectionString" />

<property name="Logger" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="IContentfulService" mapTo="ContentfulService">

<property name="ApiUrl" value="https://api.contentful.com" />

<property name="SpaceId" value="9i1m79rt44ru" />

<property name="OAuthToken" value="aae29fde36d7a2139c759de98bb19ce22f5ef92143f4abaa740ac8c1200785d5" />

<property name="Logger" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="AuthorizationFilter">

<property name="Logger" />

<property name="UserService" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="ExceptionFilter">

<property name="Logger" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="LoggingActionFilter">

<property name="Logger" />

<method name="CheckConfiguration" />

<lifetime type="singleton" />

</register>

<register type="LookupController">

<property name="LookupService" />

<method name="CheckConfiguration" />

</register>

<register type="UserController">

<property name="UserService" />

<method name="CheckConfiguration" />

</register>

<register type="ContentfulController">

<property name="ContentfulService" />

<method name="CheckConfiguration" />

</register>

</container>

</containers>

# Visual Studio Deployment Instructions for local development

* 1. Extract the package.
  2. Setup the database. Refer to section #3.
  3. Open the file \**src\WWF.Backend.sln** in Visual Studio.
  4. Configure the **Web.config** (refer section 4.4) in project **WWF** Make it fits your environment.
  5. Set **WWF** as startup project.
  6. Click menu **Debug** -> **Start Without Debugging** (or Press Ctrl + F5) to host Web API.

# Visual Studio Publishing Instructions (on server like staging or PROD)

* 1. Extract the package or git clone it from the github
  2. Open the file \**src\WWF.Backend.sln** in Visual Studio.
  3. Configure the **Web.config** (refer section 4.4) in project **WWF** Make it fits your environment.
  4. Run Publish with parameters as retrieved from your IIS Web Deploy setup (note that user must be prefixed with domain)

For more details and information about parameters to use during publish, check the following page - <http://www.developerhandbook.com/visual-studio/publish-website-to-iis-using-web-deploy/> (“Create a Web Deploy Publish Profile” section talks about publish profile – just download it from the server and import into Visual Studio)

# Unit Tests Verification (for development)

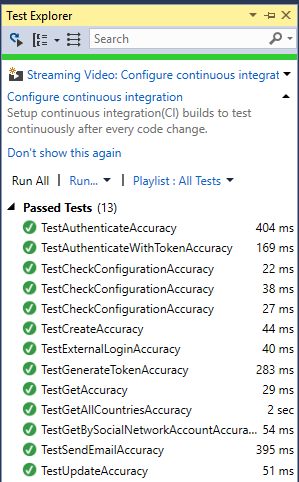
* 1. Follow section #5 to set up deployment.
* Configure the **App.config** in project **WWF.Tests.** Make it fits your environment. Refer section 4.4 for more details. Please note that **App.config** doesn’t require configuration for Controllers and Action Filters.

You also need to update path to email folders App.config file:

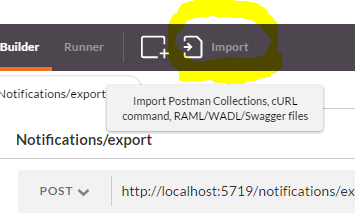
<appSettings>

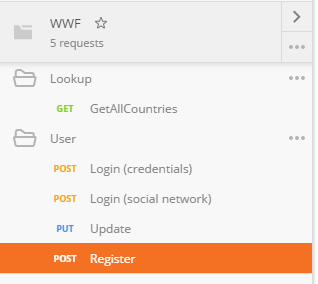
<add key="EmailFolder" value="d:\tc2016\temp\emails\"/>

</appSettings>

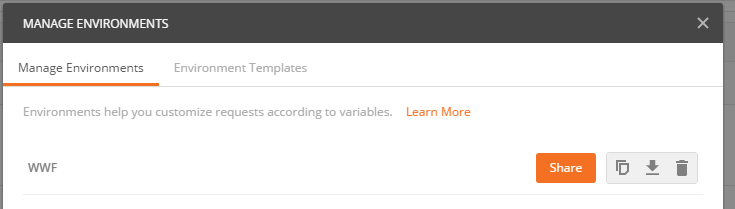
* 1. *Please note that default unit tests connection string points to the same database* ***WWF****. This means that test data will be cleared in this DB. If you want to keep your test data, change the connection string to point to other test DB.*  
     Click menu **Test -> Run -> All Tests**As a result you should see all tests passed:  
     

# Web API Verification

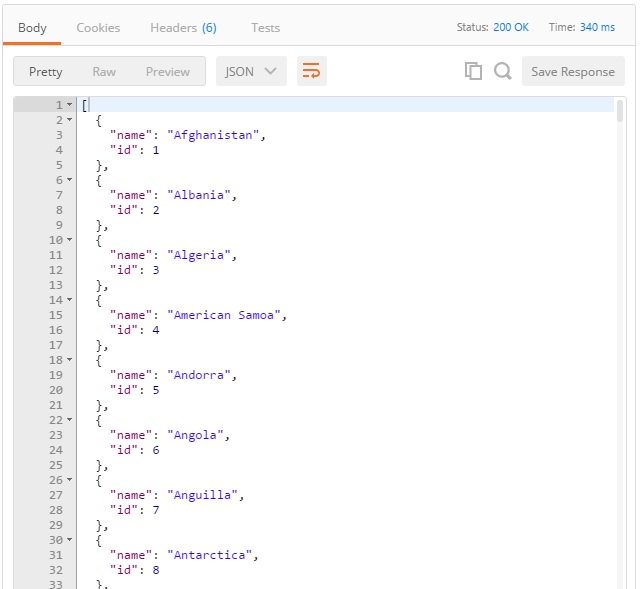
1. Import the test data: **\Database\TestData.sql** to database.
2. Host application in either IIS or by using VS 2015 as described in previous sections.
3. Open [PostMan](https://www.getpostman.com/) Chrome App
4. Import collection from **\docs\WWF.json.postman\_collection** by pressing highlighted button:   
   

All predefined queries should be loaded:  


1. Import environment variables from **\docs\WWF.postman\_environment**



6) Invoke the GetAllCountries API and if you are seeing the following it means backend deployment is successful:



*Note: In some cases you might need to re-import test data, otherwise request may not work.*

# Frontend Deployment Instructions

(<https://github.com/topcoderinc/HP-LP-WWF-Detect-IT>)

## Prerequisites

* Make sure backend services are properly deployed as described above.
* Install git <https://git-scm.com/download/win>
* Install Node.js: <http://nodejs.org>
  + On OSX use homebrew (<http://brew.sh)>: brew install node
  + On Windows use chocolatey (<https://chocolatey.org/)>: choco install nodejs
* libsass is needed for the gulp scss plugin to run as it uses the native libsass package as the bundled strategy gives weird bugs
  + npm install -g bower gulp nodemon
  + Refer to [instructions on how to not require sudo](https://github.com/sindresorhus/guides/blob/master/npm-global-without-sudo.md)

## Installing Dependencies

* npm install
* bower install

## Configuration

Some frontend configuration parameters are handled by environment variables see “Environment variables” section at <https://github.com/topcoderinc/HP-LP-WWF-Detect-IT> (Warning for 2017-Mar-22 - (LM) I wasn’t able to make the WWFF\_REST\_API\_BASE\_URL variable work through env variables, so I am still updating the config.js).

For everything else that you want to configure, you need to manually update *config.js* file, that is located under *src\client\app\core*. Following parameters can be configured there:

* REST\_API\_BASE\_URL: default value is <http://localhost:50630>
  + Note that you MUST update it to reflect the proxy (IIS) IP/DNS and port, even if those two run on the same server “localhost” will not work fine (unless you are doing local development). For STAGE/PROD – use https, NOT http, otherwise users will get error in their browsers, since https-secured frontend will make http-insecure api calls.
* PERIOD\_YEARS: Defines the range of years in the search data page. Default value is [2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015]
* SEARCH\_DATA\_MAX\_IMPORTERS\_COUNT: Defines maximum number of importers in trade data search query. Default value is 5
* SEARCH\_DATA\_MAX\_EXPORTERS\_COUNT: Defines maximum number of exporters in trade data search query. Default value is 5
* SEARCH\_DATA\_MAX\_COMMODITIES\_COUNT: Defines maximum number of commodities in trade data search query. Default value is 5
* DASHBOARD\_CASE\_STUDY\_COUNT: Defines count of latest case studies to display on the right-side panel. Default value is 3
* CSV\_FILE\_NAMES: Defines CSV file names for exported files in Dashboard, Alert Details, and Data Comparison pages.
* contentfulSpace: The Contentful space.
* contentfulAccessToken: The Contentful access token.
* COUNTRY\_CODE\_MAPPING: This is a mapping from Vertica country name to datamaps.js country codes. It is used to colorize countries on a map.

Default application port – 8001. If you want to configure it, set the PORT environment variable.

There is also second environment variable – NODE\_ENV that can be set to build to include static files from the ./build/ folder. Otherwise it will be DEV mode which includes static files from ./src/client/, ./ and ./tmp.

For staging PORT environment variable is set to 8090. Rewrite from 80 and 443 is handled by Proxy service.

## Test Accounts

If you executed `testdata.sql` when deploying the backend, you can use the following users for testing:

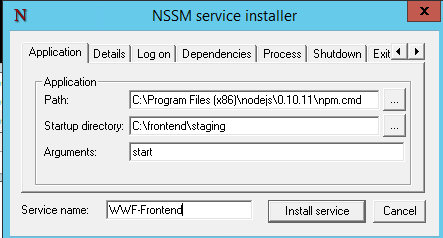
* admin / 123
* user / 123

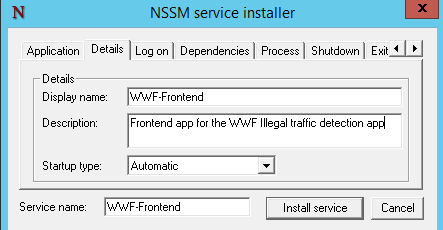
## Running in dev mode (for development only)

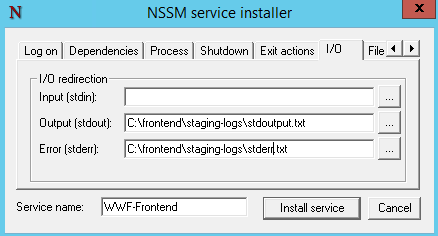
* Run the project with *gulp serve-dev*
* This opens it in a browser and updates the browser with any file changes.

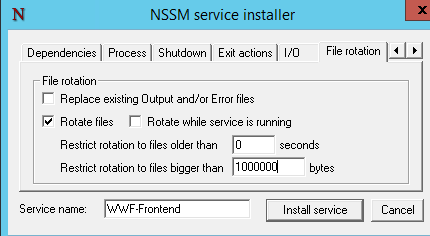
## Deploying in production mode

* Run *gulp inject* to compile css
* Run *npm start* to start the server and confirm that it works. Stop it if it does (Ctrl-C)
* Get the latest version of NSSM - <https://nssm.cc/download>
* Use it to install new service, by running *nssm install WWF-Frontend*, providing the following data:









Now press the [Install service] button. You should get confirmation that the service was installed successfully.

* Run *net start WWF-Frontend* to start the service

## Tests (development only)

* Run the unit tests using *gulp test* (via karma, mocha, sinon).

­

## Suggested upgrade approach

* 1. Fetch new code from the GitHub into a folder with a new name (e.g. c:/frontend/code20170309, *git clone https://github.com/topcoderinc/HP-LP-WWF-Detect-IT.git*)
  2. Run the following commands in this folder:
     1. *npm install*
     2. *bower install*
     3. *gulp inject*
  3. Make all required changes to the configuration file (if any)
  4. Stop the WWF-Frontend service (*net stop WWF-Frontend*)
  5. Rename the c:/frontend/staging folder to c:/frontend/old\_code20170309
  6. Close the console or move out of the c:/frontend/code20170309 folder
  7. Rename the c:/frontend/code20170309 folder to c:/frontend/staging
  8. Start the WWF-Frontend service (*net start WWF-Frontend*)

# Proxy installation

Purpose of the proxy is to route the user traffic between front-end and backend applications.

1. Deployment
2. Create a new application in IIS Management Console, called e.g. Proxy
   1. Use e.g. c:\\frontend-proxy as the root
3. Put the proxy service web.config into the root
4. If web.config is still not into any SCM, here is the source code:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<system.webServer>

<rewrite>

<rules>

<rule name="force SSL" stopProcessing="true">

<match url="(.\*)" />

<conditions>

<add input="{HTTPS}" pattern="Off" />

</conditions>

<action type="Redirect" url="https://{HTTP\_HOST}/{R:1}" redirectType="Permanent" />

</rule>

<rule name="rewrite api to backend calls" stopProcessing="true">

<match url="^api/(.+)$" />

<action type="Rewrite" url="http://10.151.8.4:8080/{R:1}" />

</rule>

<rule name="rewrite everything else to frontend" stopProcessing="true">

<match url="(.\*)" />

<action type="Rewrite" url="http://10.151.8.4:8090/{R:1}" />

</rule>

</rules>

</rewrite>

<defaultDocument>

<files>

<remove value="index.php" />

<remove value="default.aspx" />

<remove value="index.htm" />

<remove value="iisstart.htm" />

<remove value="index.html" />

<remove value="Default.asp" />

<remove value="Default.htm" />

</files>

</defaultDocument>

<httpProtocol>

<customHeaders>

<add name="Access-Control-Allow-Methods" value="GET,POST,DELETE,HEAD,PUT,OPTIONS" />

<add name="Access-Control-Allow-Headers" value="Origin, X-Olaround-Debug-Mode, Authorization, Accept" />

<add name="Access-Control-Expose-Headers" value="X-Olaround-Debug-Mode, X-Olaround-Request-Start-Timestamp, X-Olaround-Request-End-Timestamp, X-Olaround-Request-Time, X-Olaround-Request-Method, X-Olaround-Request-Result, X-Olaround-Request-Endpoint" />

<add name="X-Content-Type-Options" value="nosniff" />

<add name="Strict-Transport-Security" value="max-age=31536000; includeSubdomains" />

<!--<add name="X-Frame-Options" value="SAMEORIGIN" />-->

<add name="X-Xss-Protection" value="1; mode=block" />

</customHeaders>

</httpProtocol>

<httpErrors errorMode="Custom" />

<directoryBrowse enabled="false" />

</system.webServer>

</configuration>

Note that the highlighted areas must be updated to reflect the IP of the server and the ports on which frontend and backend applications where deployed.

# Current setup

## Staging

Backend REST API

* Deployed in the Default App
* Port: 8080
* Deployment method: Visual Studio publish

Frontend node.js

* Deployed in the c:/frontend/staging/ folder
* Port 8090 (default)
* Deployment method: manual (according to the deployment guide)
* Startup: autostarted service