1IT/CLD

OSLO Word add-in

|  |  |
| --- | --- |
| Author | BROEDERS Léon (1IT/CLD) |
| Date | 28/09/2020 |
| Revision | 2 |
| Sensitivity | Unrestricted |

Contents

[1. Introduction 3](#_Toc52211872)

[2. Schematic overview 4](#_Toc52211873)

[3. Tools needed 5](#_Toc52211874)

[3.1 Microsoft documentation 5](#_Toc52211875)

[3.2 Node 5](#_Toc52211876)

[3.3 Visual Studio Code 5](#_Toc52211877)

[3.4 Microsoft Edge DevTools / Debugger 5](#_Toc52211878)

[4. Code structure 6](#_Toc52211879)

[5. Testing the code 7](#_Toc52211880)

[6. Sideloading 8](#_Toc52211881)

[7. Deploying on a web server 9](#_Toc52211882)

# 

# Introduction

This document provides some developer documentation for the OSLO Word plugin. The goal of this project is to create a Word add-in, which can make use of the [OSLO reference data](https://data.vlaanderen.be/standaarden/) of Informatie Vlaanderen (Vlaamse overheid, the Flemish government).

The first milestone release of the add-in supported look-up of selected text in the OSLO database (an Elasticsearch database, made available on the web). The second milestone extended this with automatic search on selection, a button to search the next matching OSLO phrase in the Word document, and two buttons to add the OSLO description and reference as a footnote or endnote respectively. To make this kind of searching possible, the add-in now loads in the entire dataset when it's initialized (0.5MB or so, relatively small on today's web).

The following screenshot demonstrates how this add-in looks when loaded into Word on the desktop:

A screenshot of a social media post

Description automatically generated

The add-in side-pane is opened by clicking on the "VO OSLO" button in the toolbar. Selecting any text will automatically perform a lookup in the OSLO database. A search box is also available for free-text lookups. When more than one search result is present, the user can select one via a check-box, and insert its info as either a footnote or endnote into the Word document.

# Schematic overview

Office plugins used to be written in native code, which ran inside of the same process as the Office applications (somewhat sandboxed). With the arrival of O365, Microsoft created a new type of add-in, which is basically a web application (HTML, JS, CSS) that runs inside a browser. Office applications on the desktop then run the add-in in an embedded browser, while the online applications already run inside a browser, so the plugin is loaded into an IFRAME.

The advantage of this architecture is that (in theory) the same add-in can be loaded in either the desktop or the online applications, and unchanged on different OSes. The disadvantages are that extra infrastructure is needed to host the add-in (i.e. a web server), and that the extension capabilities of the new add-ins are much more limited than the legacy plugins.

The architecture for a Word add-in looks somewhat like the following, on the left the online case, and on the right the desktop case:

Office.js API

Office.js API

Serves

Add-in host  
Node.js server for development  
Some web server for production

O365 Cloud

Add-in  
(HTML + JS + CSS)

Browser  
(embedded Edge/IE11)

Word document

Word (Desktop)

IFRAME

Add-in  
(HTML + JS + CSS)

Word Online

Browser

The

Elasticsearch API

OSLO  
Database

Serves

Serves

The OSLO database can be accessed either directly by the add-in (by making HTTP calls from JavaScript code), or using the add-in host as a proxy/backend. The former option is easiest for testing and development, but will cause Different Origin security issues in some browsers. Therefore the latter option is better for production.

# Tools needed

This documentation assumes the development environment is MS Windows. Although most of the needed tools are also available for Linux and Mac OS, developing under Windows makes it easier to test the add-in in MS Word.   
To install these tools, administrator access will be required.

## Microsoft documentation

The root of the MS documentation can be found at:  
<https://docs.microsoft.com/en-us/office/dev/add-ins/>

Good starting points for our case are:  
<https://docs.microsoft.com/en-us/office/dev/add-ins/overview/learning-path-beginner>

The basic frame for the OSLO Word plugin was created by following the instructions from:  
<https://docs.microsoft.com/en-us/office/dev/add-ins/tutorials/word-tutorial>

## Node

Node.js is a tool chain written in server-side JavaScript/TypeScript, and is used as the build environment. The latest LTS version is needed, which can be found here (Windows 64-bit):  
<https://nodejs.org/download/release/latest-v14.x/win-x64/>

Just download the \*.exe file and install with default options.

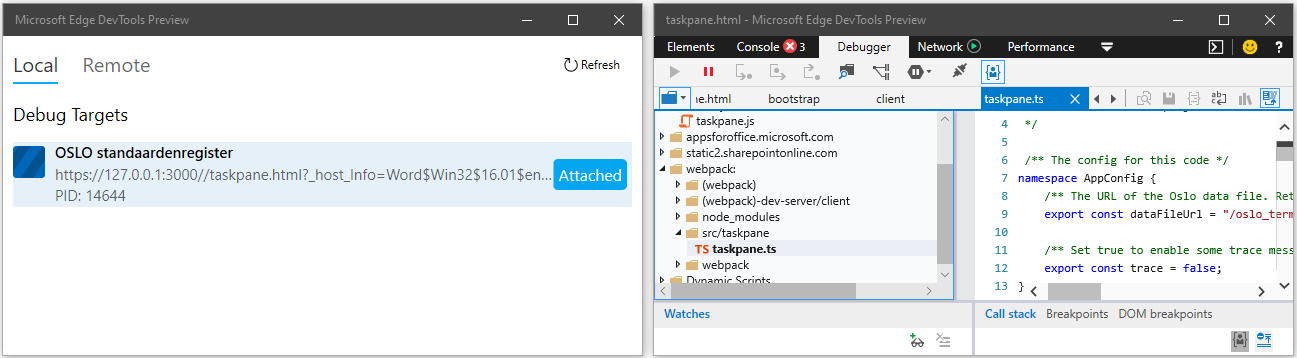
## Visual Studio Code

Any editor or IDE can be used to edit the sources. We followed one of the recommendations from the MS documentation, and used Visual Studio Code, a free-to-use IDE from MS. The installer can be found here:  
<https://code.visualstudio.com/download>

## Microsoft Edge DevTools / Debugger

The add-in is basically a web application (HTML, JS, CSS) that runs inside a browser.  
When running the add-in from within Word Online (Office 365), the browser that was used to open O365 can also be used to debug the add-in. In Edge for example, the Development Tools are opened by pressing the F12 key. Other browsers have similar functionality (CTRL+SHIFT+I in FireFox and Chrome).

When using Word on the desktop, the add-in is running in an embedded Edge browser (or IE11 in some older versions). For this case, MS created a special version of the Development Tools, that can be attached to running processes. For this, just start the tool, and it will display a list of running Edge instances. The process that one should attach to is usually named "OSLO standaardenregister", running from 127.0.0.1:3000 . It's also possible to connect to the O365 instance this way, but it's easier to use the browser's built-in tools for debugging.



This tool is free to use and can be downloaded from the MS site:  
<https://www.microsoft.com/nl-be/p/microsoft-edge-devtools-preview/9mzbfrmz0mnj?rtc=2&activetab=pivot:overviewtab>

When debugging the code with breakpoints, Word on the desktop will restart the add-in when stopped in a breakpoint, because it thinks the add-in is stuck. To disable this behavior, add/modify the following registry key, and restart your computer:

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Office\16.0\Wef]

"AlertInterval"=dword:00000000

# Code structure

The code is initially provided as a ZIP file. Access to a gitlab.com repository can also be provided, if requested.

After unzipping the code has the following structure:

|  |  |
| --- | --- |
| Path | Description |
| assets\vo\_logo.png | The image for the sidebar header |
| assets\vo\_oslo\_logo.png | The image for the toolbar button |
| dist\ | Empty. After a "npm run build" command, the output files come here. These can then be copied to a web server. |
| docs\OSLO Word add-in.docx | This document |
| node\_modules\ | After a "npm install" command, node.js will download its libraries here. |
| src\php-wrapper\searchOslo.php | An small example PHP proxy that wraps the call to the OSLO Elasticsearch (used to demo M1 of the add-in). |
| src\taskpane\taskpane.html | The HTML layout of the add-in. |
| src\taskpane\taskpane.ts | The Typescript code for the add-in. |
| src\taskpane\taskpane.css | The style sheet for the HTML layout. |
| src\taskpane\oslo\_terminology.json | A copy of the OSLO Elasticsearch API response, which is loaded as a web resource instead of calling the API. Needed for demo-purposes after the online DB was repeatedly cleared. |
| manifest.template.xml | A template file to create a valid manifest.xml from |
| manifest.xml | Generated from the template after running "npm run build". This XML descriptor is an integral part of the add-in. It configures the toolbar button, right-click menu, and defines from where the add-in is supposed to be loaded. This manifest is needed for local testing. |
| /dist/manifest.xml | Also generated from the template. This manifest is hosted via the web server, and uses a hosted URL and app domain instead of a local one. It's needed when hosting the add-in on the web. |
| LICENCE | The code license. Currently MIT. Can be changed to whatever by the license holder. |
| README.md | Generated by the YoOffice add-in generator. Modified to add some basic build instructions. |
| package.json package-lock.json | Defines which packages are installed for this node.js project. The lock file contains the exact versions needed, allowing the node\_modules folder to remain empty for distribution. |
| ts-config.json | Configures some typescript compiler options (default). |
| webpack.config.js | This is the node.js build configuration. Some parameters are defined at the top of this file, such as the local web server port (3000), and the URL and hostname for the hosted add-in. |

# Testing the code

Getting up and running is simple, provided everything works as expected. First make sure node.js is installed, and available on the command path (executing "npm" should work).

Then unzip the oslo-word-addin somewhere. To be safe, best use a folder without spaces, e.g.

C:\test

Then open a command prompt, and go to this folder

C:  
cd \test\oslo-word-addin

Next, the node.js modules need to be downloaded. For this, node needs a direct connection to the internet.

npm install

npm update

In this version, the manifest.xml is automatically generated based on the configuration of the webpack.config.js. To generate the manifest.xml, run the following command:

npm run build

If a proxy is used, it can be configured first via (for example):

npm config set proxy http://proxy.company.com:8080

npm config set https-proxy http://proxy.company.com:8080

The update might complain about vulnerabilities (none at the time of this writing). One can execute the following command to fix (most of) them, but note that these are vulnerabilities in the *build* tool, and the add-in sources don't use these libraries (for the most part) and therefore don't contain the vulnerabilities.

npm audit fix

To start testing, now simply execute:

npm start

This should start up a second window, compile the sources, pack them, start up a local webserver on 127.0.0.1:3000, then start up Word, and sideload the add-in into it. The add-in then still needs to be opened by clicking the "VO OSLO" button on the toolbar.

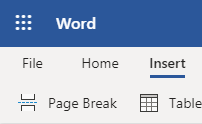
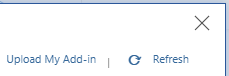
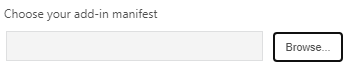
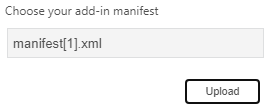
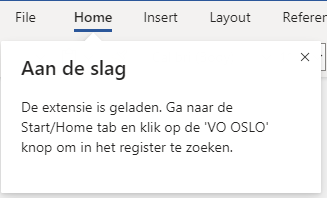
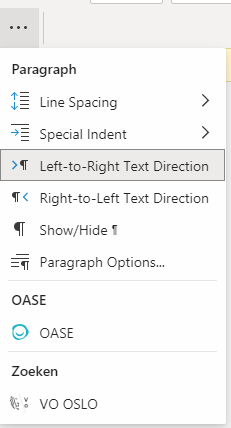
Once this build is started, it keeps rebuilding automatically whenever the source code changes. This will also reload the add-in automatically, so debug sessions will be terminated, and some UI state may be lost.

To stop auto-building and close Word, just issue the command:

npm stop

# Sideloading

To test the plugin in Office 365, the add-in needs to be manually sideloaded. The instructions for this are:

1. Open a Word document in o365 (new or existing)
2. Open the "Insert" tab  
   
3. Click the last button "Add Ins"  
   
4. Click the "Upload My Add-in" link at the top right  
   
5. Click the Browse button  
   
6. In the file dialog, select or enter the following:  
   C:\test\oslo-word-addin\manifest.xml (substitute your working folder)  
   Alternatively, you can enter a URL to the hosted manifext.xml :  
   <https://127.0.0.1:3000/manifest.xml>  
   Just make sure that you adjust the build parameters first in the latter case, because by default the manifest will contain a dummy hostname, and the add-in will fail to load.
7. Click the Open button to select the file and close the selection dialog
8. Click the Upload button to upload the file and close the upload dialog  
   
9. You should see a message that the add-in is loaded:  
   
10. Click on the VO OSLO button to open the side panel. You may have to click on the … button if your screen isn't wide enough  
    

This opens the side panel which initially shows instructions on how to use it

# Deploying on a web server / publishing

The manifest.xml file needs to contain the correct URLs, so Word will know from where to load the add-in. To this end, a template was created (manifest.template.xml) which contains two placeholder variables:

{{plugin\_host\_url}}

{{plugin\_host\_domain}}

The actual values for these placeholders can be found at the top of webpack.config.js:

/\*\* The port node.js starts a web server on, always on https://127.0.0.1 . \*/

const dev\_webserver\_port = process.env.npm\_package\_config\_dev\_server\_port || 3000;

/\*\* The host URL and domain used to generate the /dist/manifest.xml \*/

const plugin\_host\_url = process.env.npm\_package\_config\_plugin\_host\_url || "https://oslo.mywebserver.dev/";

const plugin\_host\_domain = process.env.npm\_package\_config\_plugin\_host\_domain || "oslo.mywebserver.dev";

Replace https://oslo.mywebserver.dev/ with your hosting URL, and oslo.mywebserver.dev with your hosting domain.

After this run the following command in the project root to generate files that can be deployed on a web server:

npm run build

This will also generate 2 manifest.xml files: one in the project root, needed with the node.js web server on 127.0.0.1, using the dev\_webserver\_port value and the other one in dist/, needed when deploying on a standalone web server, using the plugin\_host\_url and plugin\_host\_domain values.

The files created in the dist folder are static web resources that can be deployed on any simple web server like Apache or MS IIS. While using the node.js dev webserver to host these resources on the web is technically possible, it is not recommended for security reasons.

More information about the different add-in publishing options can be found here:  
 <https://docs.microsoft.com/en-us/office/dev/add-ins/publish/publish>

Note that to load an add-on into Office.js, the connection needs to be secured with TLS, or office won't initialize the plugin. This also means that a valid TLS certificate is needed, and by extension, a domain (instead of a raw IP address). One exception to this is the node.js dev webserver, which generates its own temporary TLS certificate, and installs it locally, which allows local testing.

More info on how to create/manage TLS certificates in MS Azure App Service can be found here:  
<https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-certificate>