PhoneAsId

Development Environment

Team Ewok
Michal Wozniak
Francis Cote-Tremblay
Ahmed Dorias
Harrison Ianatchkov
Sebastian Rafique Proctor-Shah
Simon Moniere Abes
3rd March, 2017

Table of contents

Overview

Our system is split in 3 application. Backend server is responsible to serving all the requests from all clients : web and mobile.

System Structure

1. **Mobile Application :** Android & iOs .
2. **Web Client application :** AngularJs 1.5.8
3. **Api Server:** Asp.net Core with 4.6 Asp.Net framework and Oracle database

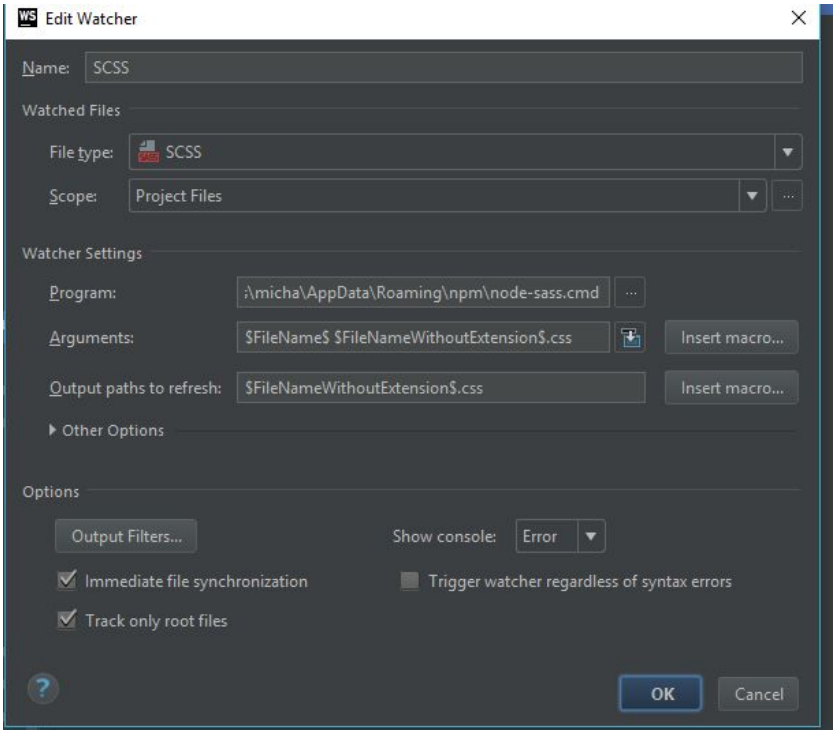
Mobile Application

Development - Android

1. Install node js . You will use npm to install all the dependencies.
2. You can use your prefered IDE but we will show how to build the application using WebStorm
3. Currently the code is on https://github.com/mv740/E-Wok-MyConcordia
4. Requirements forced us to use only 1 repository even though we have 3 application. We have place each application in a different folder and we only split them in different branches
5. Clone "dev" branch
6. Follow the Android platform guide to install required tools for development
    - 6.1. Install Java Development Kit
    - 6.2. Install the Android Stand-alone SDK Tools
    - 6.3. Add SDK Packages
        - 6.3.1. Install Android Platform SDK
        - 6.3.2. Install Android SDK build-tools version 19.1.0 or higher
        - 6.3.3. Install Android Support Repository
    - 6.4. Set environment variables
        - 6.4.1. JAVA_HOME to the location of your jdk installation
        - 6.4.2. ANDROID_HOME to the location of your Android SDK installation
        - 6.4.3. Add the following path to your PATH variable

```
C:\Development\android-sdk\platform-tools
C:\Development\android-sdk\tools
```

7. Open the project in Webstorm (or some other IDE)
8. Use the webstorm terminal or a command prompt in the PhaseId folder
9. `npm install -g cordova ionic`
10. `npm ionic state restore`
11. `bower install`
    11.1. If don't have bower installed please install it using
    `npm install -g bower`
12. Setting up an SCSS compiler (Android)
    12.1. Install node-sass globally using npm
    `npm install -g node-sass`
    12.2. add a new watcher in webstorm by going in Settings> Tools > File Watchers > + (in green on the right) > scss
    12.3. use the following configuration



13. Build application
    13.1. Generate Icons & Splashscreen
        13.1.1. `$ ionic resources`
    13.2. `$ gulp template`
    13.3. `$ ionic run android`

Development - iOS

1. Install node js . You will use npm to install all the dependencies.
2. You can use your prefered IDE but we will show how to build the application using WebStorm
3. Currently the code is on https://github.com/mv740/E-Wok-MyConcordia
4. Requirements forced us to use only 1 repository even though we have 3 application. We have place each application in a different folder and we only split them in different branches
5. Clone "dev" branch
6. Follow the iOS platform guide to install required tools for development
    6.1. Install XCode (Through App Store, or Apple Developer Downloads)
    6.2. Once XCode is Installed:
        6.2.1. Open the PhoneAsId project in Webstorm (or some other IDE)
        6.2.2. Open Webstorm terminal
        6.2.3. Run: `xcode-select --install`
        6.2.4. Run: `npm install -g ios-deploy`
7. Open the PhoneAsId project in Webstorm (or some other IDE)
8. Use the webstorm terminal or a command prompt in the PhaseId folder
9. Run: `npm install -g cordova ionic`
10. Run: `npm ionic state restore`

11. Run: `ionic resources` (Generates icons & Splashscreen)
12. Run: `npm install -g bower`
13. Run: `bower install`
14. Run: `gulp template`
15. Setting up a SCSS compiler (iOS)
    - 15.1. Run: `npm install -g node-sass`
    - 15.2. Add a new watcher in webstorm by going in Webstorm > Preferences > File Watchers > + (bottom left of page) > scss
    - 15.3. Use the following configuration:
        - 15.3.1. Program:/usr/local/lib/node-modules/node-sass/bin/node-sass
        - 15.3.2. Arguments: $FileName$ $FileNameWithoutExtension$.css
        - 15.3.3. Output paths to refresh: $FileNameWithoutExtension$.css.map
          Important: copy paste ALL those settings, even though the default settings looks similar they are not
        - 15.3.4. Save and Apply
16. Build application
    - 16.1. In the Webstorm terminal, Run: `ionic build ios`.
        - 16.1.1. If error occurs, follow these substeps. Otherwise continue.
            - 16.1.1.1. Run: `rm -rf ~/.cordova`
            - 16.1.1.2. Run: `cordova add platform ios`
            - 16.1.1.3. Run: `ionic build ios`
    - 16.2. Under “E-Wok-myConcordia/PhoneAsId/platforms/ios”, click on “PhoneAsId.xcodeproj”
    - 16.3. Plug your iPhone device into your MacBook.
    - 16.4. In Xcode, go to the “Xcode” tab, then “preferences”, then set up your apple developer id by clicking the “+” sign.

16.5. Click on the “Navigate” tab, then “reveal in Project Navigator”.

16.6. Under “Signing”, checkmark “Automatically manage signing”, then set your apple ID under “Team”.

16.7. Hit “cmd + shift + k”, then “cmd + option + shift + k” to clean up build files.

16.8. Click the “Play” button to run the application on your iPhone Device.

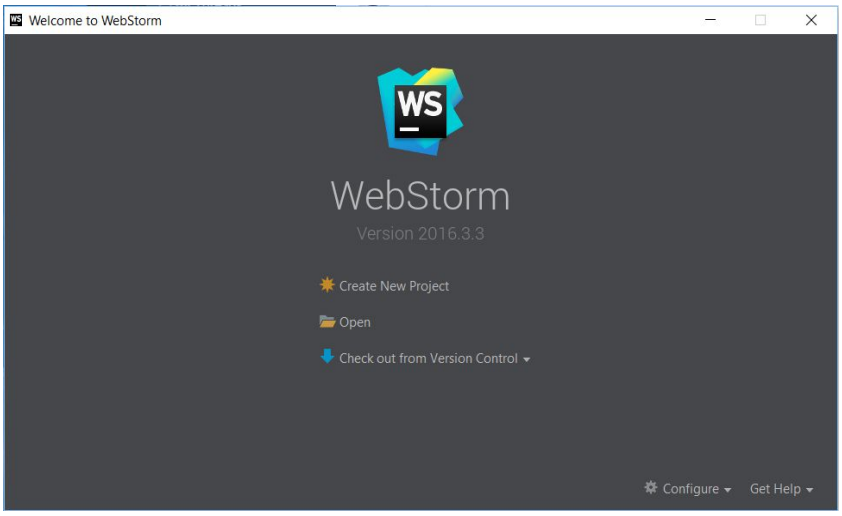
16.9. If the app is not trusted on your device, go to Settings, General, Device management, “Your apple ID”, Trust “Your apple ID”.

16.10. Run the application on your iPhone device.

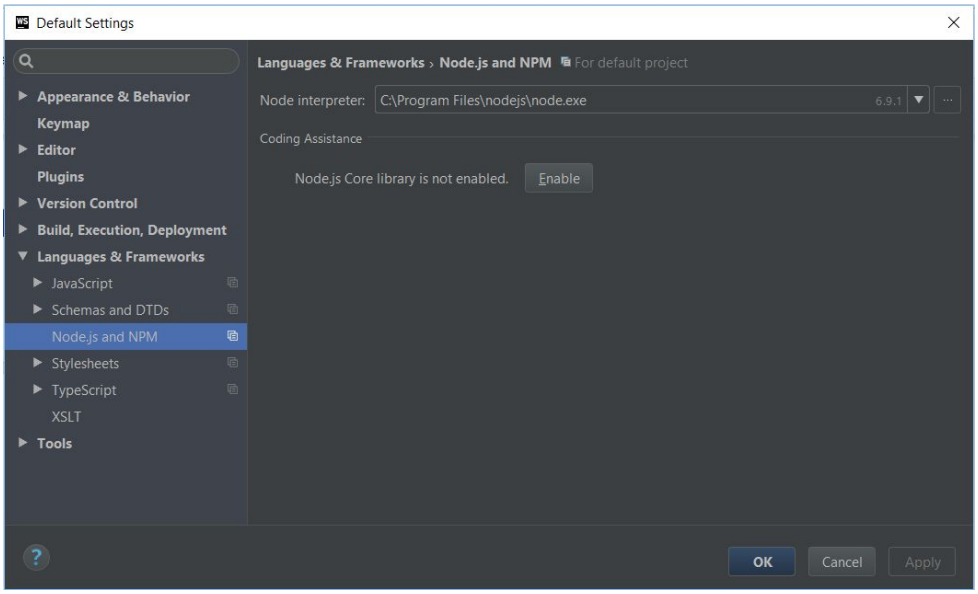
Web Client Application

Development

1. Install WebStorm
   1.1. We regularly update our version of WebStorm. Currently we are working with version 2016.3.3.
2. Install NodeJS
   2.1. Ensure you are installing it in the default location so that WebStorm detects the node executable “node.exe”
   2.2. We have versions 6.9.1 or 6.9.4, so either one works
3. Ensure NPM is linked to WebStorm
   3.1. Open WebStorm
   3.2. The picture below should be what you see. Press “Configure” and select “Settings”



3.3. Go to Languages & Frameworks. Select Node.js and NPM (should be visible if installed correctly). Ensure that the Node interpreter points to the installed node.exe executable.



4. Download or clone the project.
5. Open the project
    5.1. C:\the\path\to\the\project\E-Wok-MyConcordia\WebApp
6. Open Terminal
7. Run `npm install -g`
    7.1. This will install most dependencies required to develop the project
8. Install node-sass by following these instructions in the Android section
    8.1. This will be required in order to write css.
9. Structure Notes
    9.1. The application is separated into partials: login, sideNav, review, admin, and event.
    9.2. The web component is made up of several libraries, including
        9.2.1. fullpage.js for the separation of the content within each partial by sections
        9.2.2. Angular-fusioncharts for the display of the attendee statistics
        9.2.3. angular-translate, angular-translate-loader-static-files, angular-translate-storage-cookie, and angular-cookies for the localization component of the web application
        9.2.4. Bootstrap for some basic user interface structuring
        9.2.5. Bootstrap-datepicker and bootstrap-timepicker for the calendars displayed when selecting a date for either the update period or the event
        9.2.6. Mousetrap for the keybindings
        9.2.7. Angular-materials for some of the user interface components including the md-dialogs when prompting the user to perform actions, or the md-toast when providing the user with feedback based on his or her actions.

9.3. The angular translate library is configured in the app.js folder. The translations are gathered from two json files found in the localization.json folder called “locale-en.json” and “local-fr.json”. These two jsons will be read asynchronously and render translations to the screen based on the indicated path.

9.3.1. When performing the translations, it is recommended to stay away from performing the translations in the controllers as the asynchronous nature of the translations can cause confusions with the callbacks and unexpected behaviour. This was experienced with the statistics fusioncharts, as well as the md-toast feedback popups. It is highly recommended, if possible to have everything in the html partials. If however, this is not feasible, then understand of how angular promises work should be obtained through code and documentation analysis.

9.3.2. The JSON localization files are organized by how substantial that component would be. For example, the TOASTEDFEEDBACK major category contains the translations for the various components and their service calls.

9.4. When working on this application, it is highly recommended that you read at least some of the documentation of each of the libraries used in a particular view to understand how the controllers and services are interacting with them.

Api Server

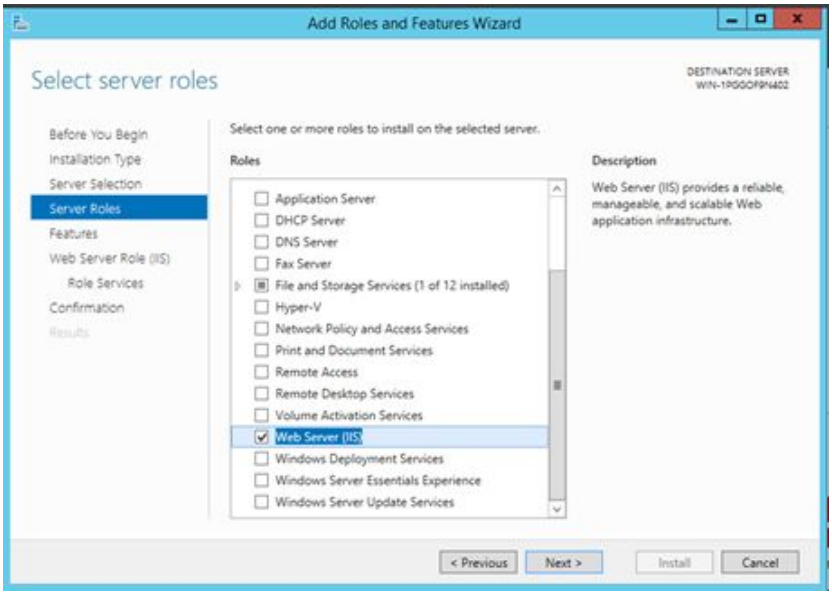
Development

1. Install Visual Studio
2. Make sure you have the Update 3 installed

3. Install the .Net Core Tools preview for Visual Studio
    3.1. Download
4. Install Oracle Developer Tools for Visual Studio 2015
    4.1. Download
    4.2. *WARNING* : After every visual studio updates, you will need to uninstall and reinstall this tool.

Deployment - IIS

1. Have fully updated windows 2012 R2 server
2. Web server (IIS)
    2.1. Install Web Server (IIS)



2.2. On the Role service step, accept the default role services provided.

2.3. Proceed through the confirmation step to install the web service role and services.

3. .Net Core Windows Server Hosting bundle

3.1. Install the Microsoft Visual C++ 2015 Redistributable

3.1.1. If installation fails, the prerequisite path KB2999226 must be installed
"VS2015 Universal C Runtime Prereq KB2999226 64-bit"

3.2. 3.2. Install the .Net Core Window Server Hosting bundle on the server. It will install the .NET Core Runtime, .Net Core Library, and the ASP.NET Core Module.

The module creates the reverse-proxy between IIS and the Kestrel server.

3.3. Restart the server or execute "***net stop was /y***" followed by "***net start w3svc***" from the command prompt to pick up a change to the system PATH

4. Publishing with Visual Studio – Web deploy

If intend to deploy the application with Web Deploy in Visual Studio, please follow

these steps.

4.1. Install latest version of Web Deploy on the server.

4.2. Use the Web Platform Installer or download it directly from Microsoft supported Downloads

5. Configure the Website in IIS

5.1. On the target IIS server, create a folder to contain the application's published folders and files, which are described in Directory Structure.

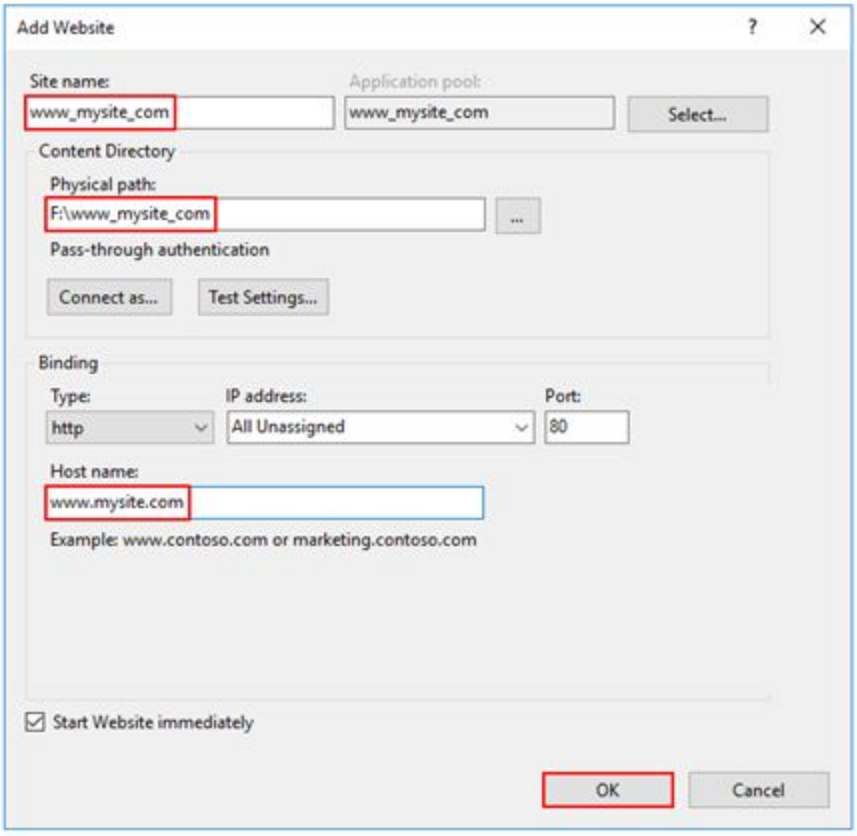
5.2. Within the folder you created, create a *logs* folder to hold application logs (if you plan to enable logging). If you plan to deploy your application with a *logs* folder in the payload, you may skip this step.

5.3. In **IIS Manager**, create a new website. Provide a **Site name** and set the **Physical path** to the application's deployment folder that you created. Provide the **Binding** configuration and create the website.

5.4. Set the application pool to **No Managed Code**. ASP.NET Core runs in a separate process and manages the runtime.

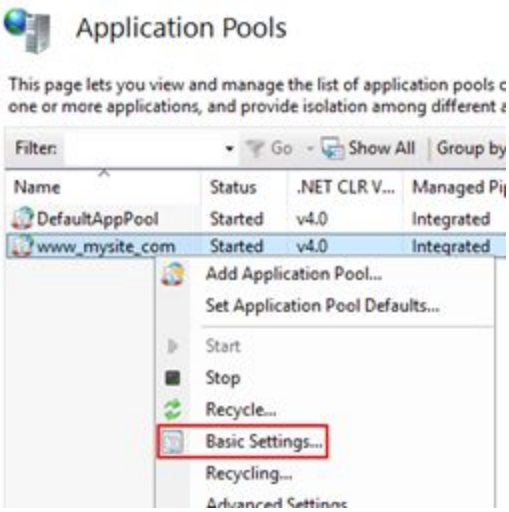
5.5. Open the **Add Website** window.

5.6. Configure the website.



5.7. In the **Application Pools** panel, open the **Edit Application Pool** window by right-clicking on the website's application pool and selecting **Basic Settings...**

from the popup menu.



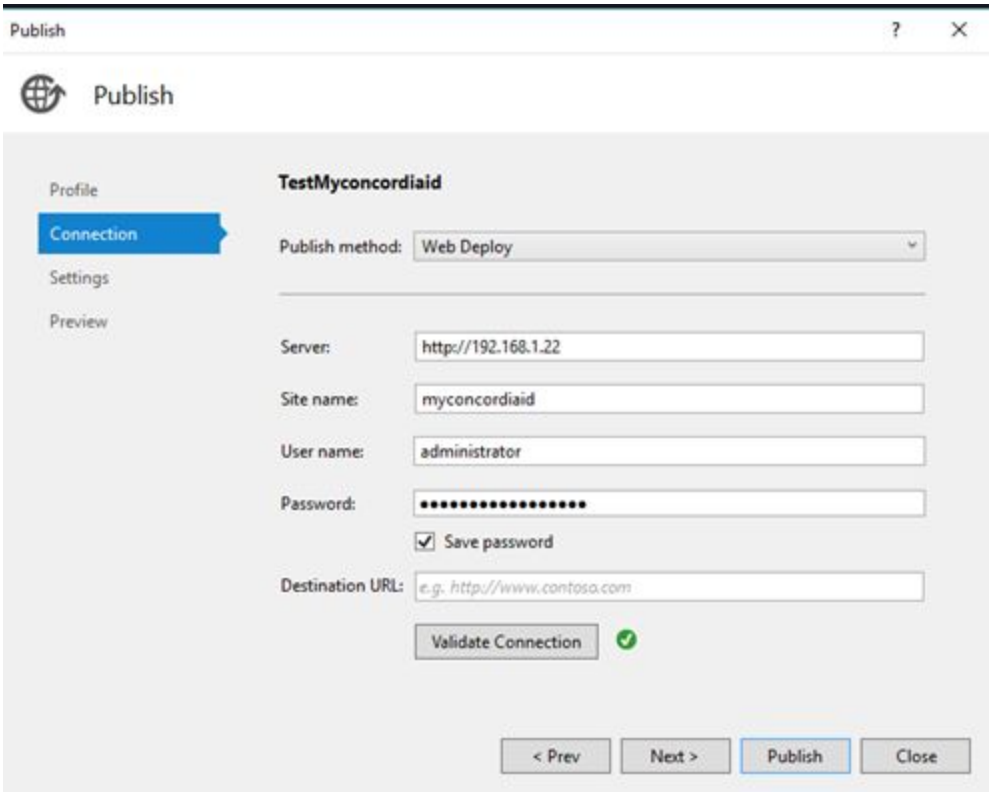
*If you change the default identity of the application pool from **ApplicationPoolIdentity**, verify the new identity has the required permissions to access the application's folder and database.*

5.8. Set the **.NET CLR version** to **No Managed Code**.

6. Publish the Application (web deploy)

6.1. Use Visual Studio Publish feature

6.2. Create a profile with your settings



Server : IIS server IP address or domain name

If you have correctly followed the guide, you will get green icon after clicking validate connections

6.3. Enable PUT and Delete request

6.3.1. WebDav is disallowing these action names by defaults. You will need to override this setting in order to get them to work.

6.3.1.1. Open your web.config file

6.3.1.2. Inside you web.config, add the following 3 lines inside the already existing system.webServer node

```
 <modules>
    <remove name="WebDAVModule" />
</modules>
```

And <remove name="WebDAV" /> inside handler

6.4. Errors

6.4.1. Error Code : ERROR_USER_NOT_ADMIN

6.4.1.1. Add `<AuthType>NTLM</AuthType>` to your publish profile xml

Security Configuration

Api Backend setting

src/MyConcordiaID/Startup.cs Lines 228-244

```
context.Applications.Add(new Application
{
    ApplicationID = "oidcWebClient",
    DisplayName = "My client application",
    RedirectUri = "https://www.myconcordiaid.me/callback.html",
    LogoutRedirectUri = "https://www.myconcordiaid.me/oidc"
    // Secret = "secret_secret_secret"
});

context.Applications.Add(new Application
{
    ApplicationID = "oidcdemomobile",
    DisplayName = "My client application",
    RedirectUri = "https://localhost/oidc",
    LogoutRedirectUri = "https://localhost/oidc",
    // Secret = "secret_secret_secret"
});
```

- ApplicationID : name of our client
- DisplayName : description
- RedirectUri : url of the page initiating the oauth protocol
- LogoutRedirectUrl : currently not used*

**For example, if you log in using your google account, and you logout of our application, this doesn't mean you will be logout of google. If you wish to also logout from your oauth provider then your will need to implement the logout redirect flow.*

Web Client setting

WebApp/app/app.js Lines 41-48

```
41          var link = "https://api.myconcordiaid.me/";
42
43          ngOidcClientProvider.setSettings({
44              authority: link,
45              client_id: "oidcWebClient",
46              redirect_uri: "https://www.myconcordiaid.me/callback.html",
47              post_logout_redirect_uri: "https://www.myconcordiaid.me/oidc",
48              silent_redirect_uri: "https://www.myconcordiaid.me/oidc",
```

- Authority : url to your backend
- Client_id : exactly the one you set in your backend
- All the uris : same as the one set in your backend

Web Client setting

PhoneAsId/www/js/app.js lines 62-67

```
8   angular.module('starter', ['ionic', 'ionic.contrib.d
9     .constant('Settings', {
10      'api' : 'https://api.myconcordiaid.me/api/',
11      'baseUrl' : 'https://api.myconcordiaid.me/'
12    })
```

- Set the urls to your backend

```
62      ngOidcClientProvider.setSettings({
63        authority: Settings.baseUrl,
64        client_id: "oidcdemomobile",
65        redirect_uri: "https://localhost/oidc",
66        post_logout_redirect_uri: "https://localhost/oidc",
67        silent_redirect_uri: "https://localhost/oidc",
68
```

Web app and mobile are using the same code for oauth.

Google setting

Startup.cs lines 179-187

```
branch.UseGoogleAuthentication(new GoogleOptions
{

    ClientId = Configuration["Google:ClientId"],
    ClientSecret = Configuration["Google:ClientSecret"],
    Scope = { "email", "profile" },


});
```

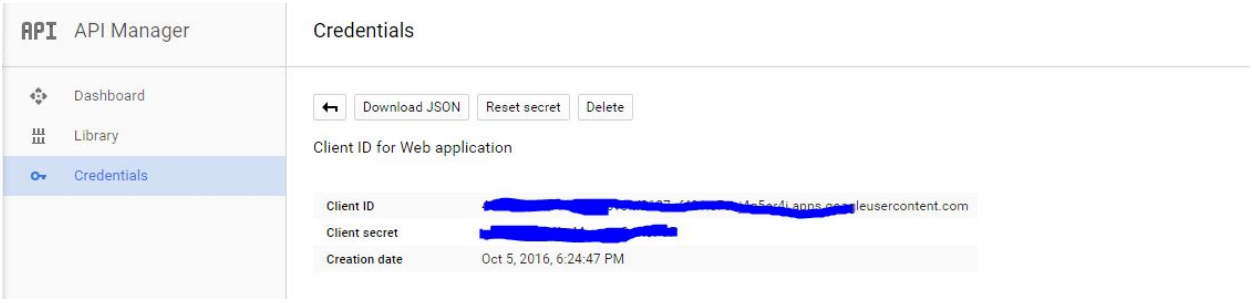
Configuration file : appsetting.json

- Set CliendId
- Set ClientSecret

Getting these two keys

1. https://console.developers.google.com
2. Click credentials
3. Click Create credentials
4. Select OAuth client Id
5. Name your client
6. Authorized Javascript origins eg : https://api.myconcordiaid.me

7. Authorized redirect URIs eg : https://api.myconcordiaid.me/signin-google
8. NEED TO USE HTTPS
9. Click create
10. Copy the client Id and your client secret
11. In Production, you should pass these two by creating 2 environment variables



Path configuration

Every path are set in the app.js. The $routeProvider has a method called “when” which will load each page with their template.

Eg

```
}).when('/event', {
    templateUrl: 'partials/event/event.html',
    css: 'sass/views/event.css',
    authenticate: true
})
```

- TemplateUrl : your html template

- Css : location of our css file
- Other properties

In production, the publish directory is WebApp/app/. Everything inside app folder must be available online. The file "index.html" must be directly accessible.

- https://www.myconcordiaid.me/login

If you don't specify which folder is the first, then you will need to set the base href in your index.html to "`<base href="/WebApp/app/">`".

- https://concordiaidclient.netlify.com/WebApp/app/login

Inherited Issues/Bugs

It's important to note the different inherited issues and bugs across the the backend, web and mobile applications. They are as follows:

Backend Issues

Web Application Issues

- One of the issues that we have been encountering for the web application is illustrating the dates in a localized manner. The current workaround is to display the dates numerically, but it takes away from the aesthetics of the page. Ideally, the dates should be displayed alphanumerically.

- For the localization of the web application, there is a bug related to the “md-select” label in the Open/Closed option of the event creation page. The option doesn’t translate until you’ve selected an option. The bug has to do with the rendering process of AngularJS. Once this issue is resolved from their end, this bug will persist.

- When searching for a student in the review page of the web application, you can only search for the student using an english date of birth. French localization has not been implemented yet in this regard.

- We have encountered issues with the fullpage.js and are therefore using a customized version of it. The regular version of fullpage.js navigates up, down, left and right on a page by modifying the url. This is not convenient for our purposes since we are developing a single page AngularJS application. Therefore, we have extracted commands executed by the full page library as a fix for our issue. An example is when slider are used in the event/attendee/statistics page. There are left/right arrows that appear due to the regular fullpage.js. In order to remove them from the page, we have to manually target them in the css page and set their display to “none”.

- We started off using angular-modals for our version of the “popup” prompts for when user action is required or additional information is to be displayed. However, since angular-materials provided more customizability, we started switching over to angular-material’s md-dialog. This transition is not complete; we are still using angular-modals in the review page, as well as in the events page when pressing the View Details button. The switch from angular-modal to angular-material md-dialogs is not complicated, nor is it imperative, but to ensure consistency throughout the application, as well as increased flexibility in interface design, this switch should be considered.

- In addition to these issues, developers can look into any remaining bugs that we have left in the github under the label “bug”.

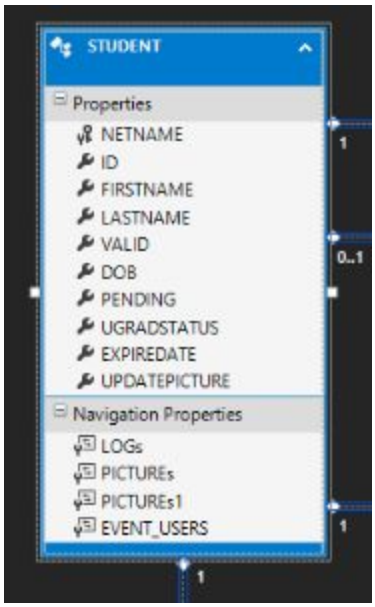
Mobile Application Issues

- Something to keep in mind when developing for the mobile application is that the android version doesn’t always translate perfectly to the iOS. There have been times where the UI looks fine on android, but doesn’t scale properly on iOS. Other times where certain features are implemented properly on android, but requires a workaround for iOS, due to functions working fine on one operating system, but being deprecated on another.

To Remove/Update

Student Table - database

During development, we create some table in our database to be able to mock our users.



We use the “Student” table to mock student accounts. We had to randomly generate these informations to be able to test all the functionalities of our system. During each successful login, the backend system is checking if that google user already exist in our database, if not, then it will generate a new account. This is done in AutorizationController.cs at line 172 to 182.

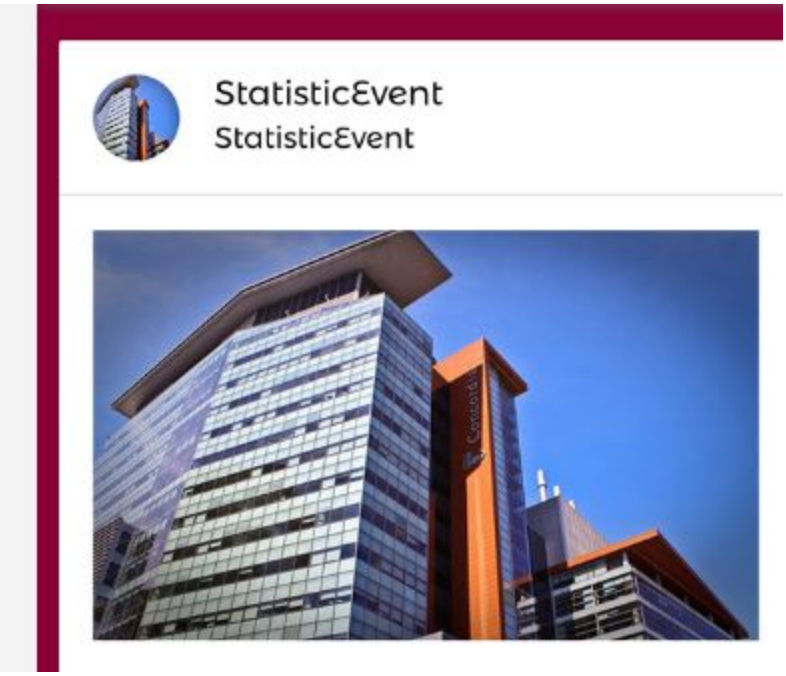
```
172             //add the user to the database if he doesn't exist yet
173             var firstName = GetGivenName(ticket);
174             var lastname = GetSurname(ticket);
175
176             if(!_studentsRepo.DoesStudentExist(firstName, lastname))
177             {
178                 var newStudent = StudentHelper.CreateStudent(firstName, lastname);
179
180                 _studentsRepo.Add(newStudent);
181
182             }
```

Removing Student table

1. Remove student table
2. Update StudentRepository, PictureRepository, AdminRepository and EventRepository because most the backend queries are calling the student table. They will need to be updated to get access to student information.

Event Description Picture

The image for the event is currently hardcoded in the mobile application and we didn’t handle uploading and storing during creation on the web client. We were also considering a fixed image based on your account. For eg, an ECA board member would have a specific account and his events will have the ECA logo by default.



Current default picture

Marshalling Card

Currently the api “api/graduation” is returning static information for demo purposes. It should be connected to another web service which hold the real information.

GraduationRepository.cs