



Professional Bachelor Applied Information Technology



Integration of popular calendars

Sinan Samet

Supervisors:

Mr. Mitch Dries
Mr. Johan Cleuren

Otys
PXL University of Applied Sciences and Arts



Eindwerk Academiejaar 2018 - 2019



**Professional Bachelor Applied
Information Technology**



Integration of popular calendars

Sinan Samet

Supervisors:

Mr. Mitch Dries
Mr. Johan Cleuren

Otys
PXL University of Applied Sciences and Arts

Acknowledgements

Het is een periode met up en downs geweest. Ik heb hier enorm veel bijgeleerd, niet alleen in de vorm van programmeren, maar ook in de vorm van cultuur en zelfontwikkeling. De dingen die mij het meest bij zijn gebleven zijn onder andere het StuddyBuddy programma, en natuurlijk mijn stageperiode. Mijn stage was een periode dat veel sneller is voorbijgegaan dan verwacht. De drie maanden dat ik er was zijn voorbij gevlogen.

Er zijn vele mensen geweest die mij geholpen hebben bij het realiseren van mijn eindwerk te veel om ze allemaal op te noemen. Daaronder wil ik graag de volgende personen nog in het bijzonder bedanken.

Als eerst Marijke Willems en Johan Cleuren die mij de kans aan hebben geboden om stage te mogen lopen in Praag. Ze hebben mij beide immens veel geholpen, niet alleen door feedback te geven, maar ook door elk moment dat het even niet mee zat, snel te reageren en een oplossing te vinden. Dit heeft mijn stressvolle tijden altijd in een klap weer omgezet naar een gemotiveerde sfeer. Het was mij zonder hun niet gelukt om hier te komen.

Mijn dank aan Otys voor de leerrijke ervaring die ik er heb opgedaan. Buiten dat ik er veel heb geleerd is ook altijd iedereen vriendelijk geweest en heb ik er veel leuke momenten met ze kunnen beleven. Het was niet alleen werken met de collega's, maar ook samen feesten en lachen.

Ik wil ook graag mijn ouders bedanken die altijd achter mij hebben gestaan, in mij geloofd hebben en mij ondersteund hebben, meer dan dat ik had kunnen willen.

Mijn dank aan hun en mijn vriendin die mij zijn komen bezoeken in Praag zodat ik ook onvergetelijke tijden met hun hier kon beleven.

Eveneens wil ik mijn vrienden bedanken die mij ook altijd de motivatie en inspiratie hebben gegeven om door te gaan.

Daarbij wil ik nog iedereen die mij direct of indirect geholpen hebben zoals de vele seminaries en andere iTalent invloeden bedanken.

Iedereen, enorm bedankt!

Abstract

In this bachelor project Google Calendar and Outlook Calendar are implemented into the Otys Calendar feature. This helps to keep everything in one place and avoid the complex situation of having to use multiple calendar applications just to see if something is planned.

To integrate these calendars, CalDav is used. CalDav is a service that allows calendars to easily communicate with each other so they can be synchronised in an orderly way. Once one calendar has been made available for integration, it is easier to implement many of the other possible calendars.

Fortunately, Google Calendar is already preconfigured in CalDav synchroniser, but it is also possible to implement a custom calendar.

To integrate the Outlook Calendar, the new Microsoft Graph API is used with its many great features.

As for the research part, it is not necessary to make a large application in AngularJS, but rather in the new Angular instead. Large web applications made in Angular JS have the tendency to be less structured and slow. This causes employees to make progress in a much slower pace as everything is just rather hard to find in the AngularJS code. This is mostly because, AngularJS is not meant to contain such a large project. Instead, Angular 4 is the best option for large scale projects.

The research focuses on the migration of the whole project from AngularJS to Angular. With NgUpgrade it is possible to upgrade the project in a hybrid way. This way it is possible to gradually migrate to the new version component by component. The goal of the research is to describe both the best method to achieve a smooth migration as the pitfalls of this process. This is documented with a proof of concept in which one module is migrated from AngularJS to Angular.

Table of contents

Acknowledgements	4
Abstract	5
Table of contents	6
List of used images	8
List of tables	9
List of abbreviations	10
Introduction	11
1 Traineeship report	12
1.1 About Otys	12
2 Internship assignment	13
2.1 The problem	13
2.2 The goal	13
2.3 Environment	13
2.3.1 Existing version of Google Calendar	13
2.3.2 Database	13
2.3.3 AngularJS	13
2.3.4 JavaScript, HTML and CSS	14
2.3.5 PHP	14
2.3.6 CalDav	14
2.3.7 IDE	14
2.4 The development processes	15
2.4.1 Integrating the existing version	15
2.4.2 Creating the cronjob (Daemon) script	23
2.4.3 Integrating Outlook Calendar	24
2.4.4 The parser	28
2.4.5 The connection	31
2.4.6 OAuth verification	32
2.4.7 The comparator	32
3 Reflection	33
4 Research topic	34
5 Research method	34
5.1 Total conversion	34

5.2	ng-forward	35
5.3	ngUpgrade	35
5.4	Hybrid Router	36
6	Elaborating on the research	37
6.1	What is the AngularJS framework? [2]	37
6.2	What is Angular and how is it better? [4]	39
6.2.1	What is the Angular framework?	39
6.2.2	Why is it better than AngularJS?	39
6.2.2.1	JavaScript vs Typescript	39
6.2.2.2	Speed performance of AngularJS vs Angular	39
6.2.2.3	The learning curve of AngularJS vs Angular [5]	39
6.2.2.4	The ahead of time (AOT) compiler of Angular [6]	40
6.2.2.5	Angular components	40
6.2.2.6	Babel	40
6.2.2.7	Webpack	40
6.3	Preparing the migration from AngularJS to Angular [3]	41
6.4	Conclusion	52
	References	53

List of used images

Figure 1: OAuth table structure.....	15
Figure 2: OAuth table example	15
Figure 3: Google synchronisation setting.....	16
Figure 4: Granting permission	16
Figure 5: Choosing account in Google.....	17
Figure 6: Confirming access.....	17
Figure 7: Setting the calendar ID	17
Figure 8: Granted permission to Google	18
Figure 9: The strategy pattern	18
Figure 10: Folder structure with the implementation of the design pattern	19
Figure 11: iConnection interface.....	19
Figure 12: getComparison being called	20
Figure 13: Implementing iConnection	20
Figure 14: Singleton pattern	20
Figure 15: getComparison method	21
Figure 16: New structure	21
Figure 17: Old structure.....	21
Figure 18: addEventService in Caldav.....	22
Figure 19: addEventService in Outlook	22
Figure 20: Cron script.....	23
Figure 21: iConnection.php	25
Figure 22: iComparison	26
Figure 23: OutlookConnection	26
Figure 24: CalendarSynchroniser	27
Figure 25: Data binding example.....	38
Figure 26: Error because define is used in a wrong way.....	45
Figure 27: Compiled JavaScript code	46
Figure 28: Code to the entry file	46
Figure 29: Example AMD code.....	47
Figure 30: Example ES6 code	48

List of tables

Table 1: List of abbreviations.....10

List of abbreviations

Acronym	Definition
AOT	Ahead of Time
HTML	Hypertext Transfer Mark-up Language
CSS	Cascading Style Sheets
PHP	PHP: Hypertext Pre-processor
JS	JavaScript
IDE	Integrated Development Environment
AMD	Asynchronous Module Definition
ES6	ECMAScript 6
NPM	Node Package Manager
SOLID	Single responsibility, Open closed, Liskov substitution, Interface Segregation, Dependency inversion
Ctag	Content tag
ID	Identification

Table 1: List of abbreviations

Introduction

In this thesis I explain about what I did during my internship at Otys in Prague. It contains both my task for the internship and my research.

The task for my internship I got assigned was to implement different services like Google calendar and Outlook calendar to the Otys calendar. This way it became possible to synchronise all three of these calendars, so, that there is no more need to switch between them. This is a very useful functionality and it was a very interesting project to work on. I learned to tackle problems in a different way than I was used to and most of all it helped me be a better problem solver.

My research in this thesis is about migrating AngularJS to Angular. I chose to research this subject because, I believe it will be a very valuable step for Otys and I think it can make a breaking change to the company. It seemed like a great idea to me to be a part of this.

1 Traineeship report

1.1 About Otys

The life of recruiters can be hard if everything has to be done manually. There is a lot of data to be tracked and it is easy to lose sight over this. It is also of importance that the people they recruit are valuable for the company. But how will they know if this is true, and where can they find such people?

This is where Otys steps in. Otys is a company that works on perfecting their recruiting software in which everything related to recruiting has to be covered. For this reason, the life of a recruiter using this software will be a lot better.

Because the process is then optimized, and a big part of it is automated it will be possible to give the customers and candidates more attention. On top of this they will not only have more but also better matches that suit the needs of the company.

2 Internship assignment

2.1 The problem

Otys has a calendar feature in which people can add appointments, days off, birthdays and so forth. But people want to use their trusted calendar service which they use on their phone or other devices in their daily lives and not keep switching between calendars and see different things. That is how the problem exists of not being able to manage all calendars inside one calendar.

2.2 The goal

So, since the current situation is that the Otys calendar only has its own data, the goal is to integrate both the Google Calendar and the Office 365 Calendar. Fortunately, the Google Calendar integration is already developed but this is explained later. So, it should be possible to synchronise both these calendars so, that all the data can be seen on just one calendar.

2.3 Environment

2.3.1 Existing version of Google Calendar

As mentioned before, the Google Calendar version is already present. It was made a year ago by an intern from PXL. The problem is that he did not integrate it into the application. So, the task is to integrate the Google Calendar into the software and afterwards create and integrate the Office 365 version.

2.3.2 Database

The settings of the synchronisation of this module can be retrieved from MongoDB. This is because the settings in Otys are all handled in MongoDB. The rest of it is done in MySQL using the IDE Navicat.

2.3.3 AngularJS

The frontend is made in AngularJS, but there is not much work to do in there because the calendar already exists. All that has to be done is writing some functions to synchronise it with the services of Google Calendar and Office 365 Calendar. Most of the work is done in the backend so that in the frontend, the backend can be called. The backend will then provide all data to the frontend.

2.3.4 JavaScript, HTML and CSS

As mentioned before, the frontend is made with the framework AngularJS, specifically version 1.3. The languages used in AngularJS are HTML, CSS and JavaScript. Besides this the framework structure itself also has to be known. However, this is not used much for the task of syncing the calendar module.

2.3.5 PHP

PHP is the main language used in the backend in Otys. It is a language which offers to program in OOP or just procedural if desired. Fortunately, since the scale of the project is this large, the PHP used in this company is OOP based. However, unfortunately Otys is not using any framework for PHP and this might make things difficult as the structure of the project is custom made.

The job that PHP will do within this task is provide the layer between the frontend and the calendar services Office 365 and Google Calendar. Also, the calls to the database will again be made in PHP.

2.3.6 CalDav

CalDav is the suggested service by the company to use to handle this task. It is a plugin that helps the synchronisation between Google Calendar, Outlook and many other calendar services. This service makes the two-way synchronisation much easier and deplete the need of setting up basic functions just to get to the main work done.

2.3.7 IDE

The IDE that is mainly used in Otys is PHPStorm. PHPStorm is a text editor to develop programs as it indexes everything and makes it easier to go to where a specific function is defined and where it is called. Of course, there are many more features this software provides to make programming easier. A disadvantage of this IDE is that it starts up slowly.

For that reason, Visual Code was also used to assist PHPStorm in the features that it is missing. It starts up much faster and it capable of searching for specific words with good filter options. The downside of Visual Code is that it does not index the project and because of this, it cannot link from a function to its definition or the other way around.

To keep track of the database, Navicat was used as IDE which is a great application to manage databases and has everything that is needed.

2.4 The development processes

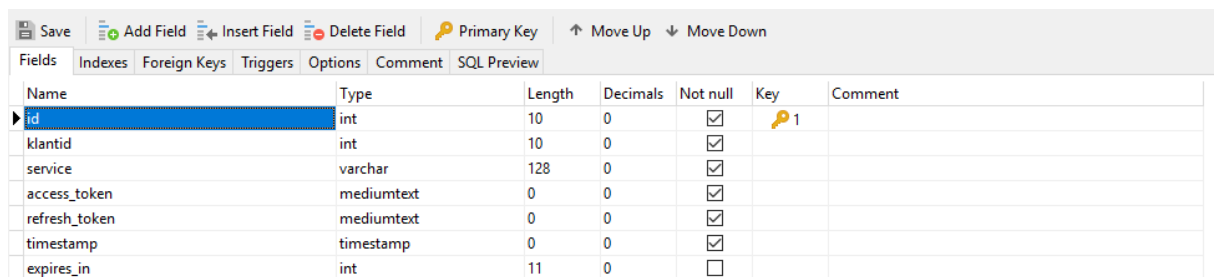
2.4.1 Integrating the existing version

To start integrating the existing version, its function and code is researched. It is a completely stand-alone application which is ready to integrate into the Otys application. Part of the CalDav application is already integrated by a developer of Otys. So, the integration started from the place the developer left off.

There is a need to keep in mind that the Outlook Calendar also needs to be integrated. However, the focus for now is to just integrate the CalDav Calendar and after that to look on how to make it generic so that it will be possible to add more services to it.

First, there is a separate database which needs to be implemented into the Otys database. For the tokens there is already a table with a proper structure. The data for all tokens is inserted in that table.

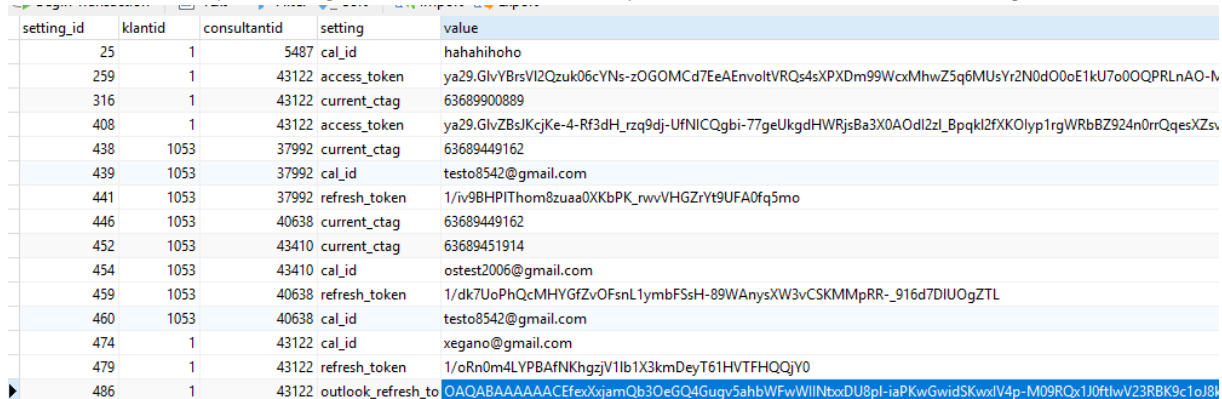
Figure 1 illustrates the structure of the tokens table:



Name	Type	Length	Decimals	Not null	Key	Comment
id	int	10	0	<input checked="" type="checkbox"/>	1	
klantid	int	10	0	<input checked="" type="checkbox"/>		
service	varchar	128	0	<input checked="" type="checkbox"/>		
access_token	mediumtext	0	0	<input checked="" type="checkbox"/>		
refresh_token	mediumtext	0	0	<input checked="" type="checkbox"/>		
timestamp	timestamp	0	0	<input checked="" type="checkbox"/>		
expires_in	int	11	0	<input type="checkbox"/>		

Figure 1: OAuth table structure

And the currently existing table moved to the Otys database can be seen in figure 2:



setting_id	klantid	consultantid	setting	value
25	1	5487	cal_id	hahahihoho
259	1	43122	access_token	ya29.GlvYBr5VI2Qzuk06cYNs-zOGOMCd7EeAEnvoltVRQs4sXPXDm99WcxMhwZ5q6MUsYr2N0d0oE1kU7o0OQPRLnAO-M
316	1	43122	current_ctag	63689900889
408	1	43122	access_token	ya29.GlvZBsKcjKe-4-Rf3dH_rzq9dj-UfNICQgbi-77geUkgdHWRjsBa3X0AOdl2zL_Bpqkl2fXK0lyp1rgWRbBZ924n0rrQqesXZsv
438	1053	37992	current_ctag	63689449162
439	1053	37992	cal_id	testo8542@gmail.com
441	1053	37992	refresh_token	1/iv9BHPIThom8zuua0XKbPK_rvwVHGZrYt9UFA0fq5mo
446	1053	40638	current_ctag	63689449162
452	1053	43410	current_ctag	63689451914
454	1053	43410	cal_id	ostest2006@gmail.com
459	1053	40638	refresh_token	1/dk7UoPhQcMHYGFzVofSnL1ymbFSsH-89WANysXW3vCSKMMpRR-_916d7DIUOgZTL
460	1053	40638	cal_id	testo8542@gmail.com
474	1	43122	cal_id	xegano@gmail.com
479	1	43122	refresh_token	1/oRn0m4LYPBAfNKhgzjV11b1X3kmDeyT61HVTFHQQjY0
486	1	43122	outlook_refresh_to	0AQABAAAAACFexXjamQb3OeGQ4Gugv5ahbWFwWlINbxDU8pl-iaPKwGwid5KwxlV4p-M09RQx1J0ftlwV23RBK9c1oJ8l

Figure 2: OAuth table example

In figure 2, refresh tokens as well as access tokens can be seen. These are of course moved to the OAuth table shown above it. Google calendars requires a calendar ID and a CTAG for the purpose of synchronising efficiently. These values are still kept in the moved table and used from there. The same approach is used for the Outlook Calendar although it does look a little different since they are generalised, the differences are made alike so that it will not cause errors.

After setting up the database correctly, the application is moved as well. The application is made keeping the Otys software in mind, so it is possible to just move the service into the services, the model to the models and the application to the features folder. To make the connection from the backend to the frontend, the service is called from the frontend.

So, at the frontend search for the setting “Google calendar synchronisation” like shown in figure 3.



Figure 3: Google synchronisation setting

When pressed on the setting, the button “Grant Permission” will be visible.

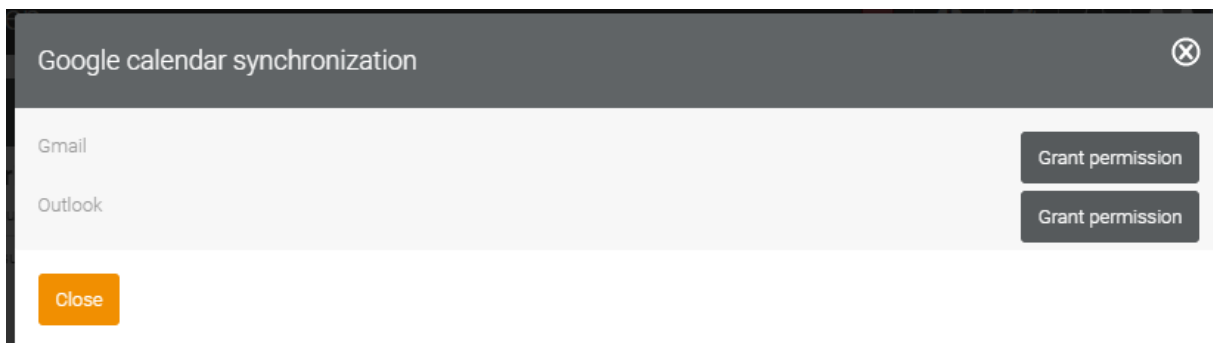


Figure 4: Granting permission

After pressing on that, it will show options of accounts to choose from which can be linked to the Otys calendar.

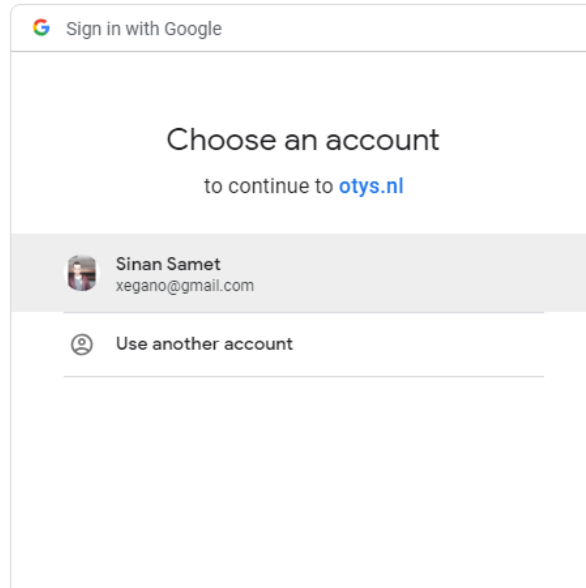


Figure 5: Choosing account in Google

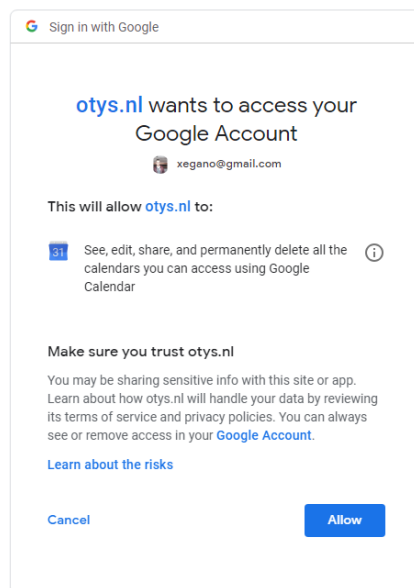


Figure 6: Confirming access

When that is done, it will return to the Otys page and since Google Calendar requires a calendar ID, it will also ask to fill in the calendar ID of the user.

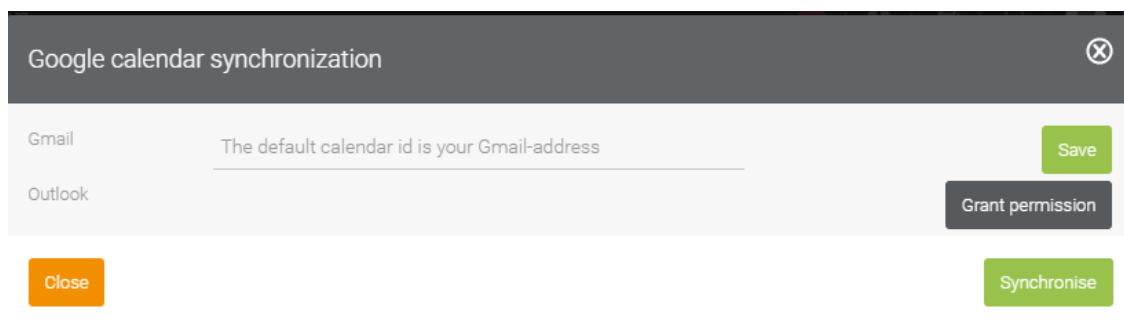


Figure 7: Setting the calendar ID

The user has to fill in the email-address of the account used for the calendar to allow access to the default calendar. After that has been done, the user has to press “Save” which can be seen in figure 7. Once that has been done, the agenda will synchronise. It is then possible to manually synchronise by either clicking the “Synchronise” button or just let the cronjob do its job of synchronising every five minutes. This process is explained in 2.4.2.

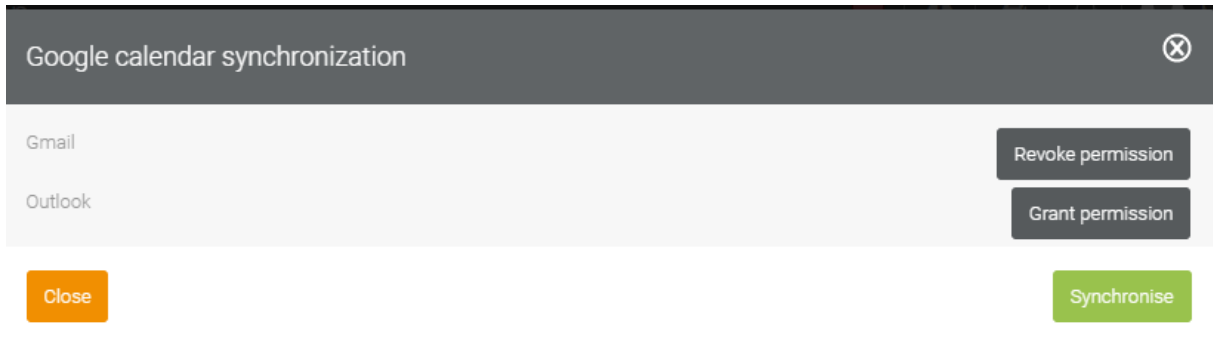


Figure 8: Granted permission to Google

It is also possible to revoke the agenda in which case the Google events will be removed.

From here on out, the integration of Outlook started. To integrate Outlook, the first thing that came to mind was the fact that there will be another service added. Since there is a need for this now, it might mean that there will also be a need for it with a new service in the future. For that reason, it was wise to implement a design pattern. Specifically, the pattern “Strategy”.

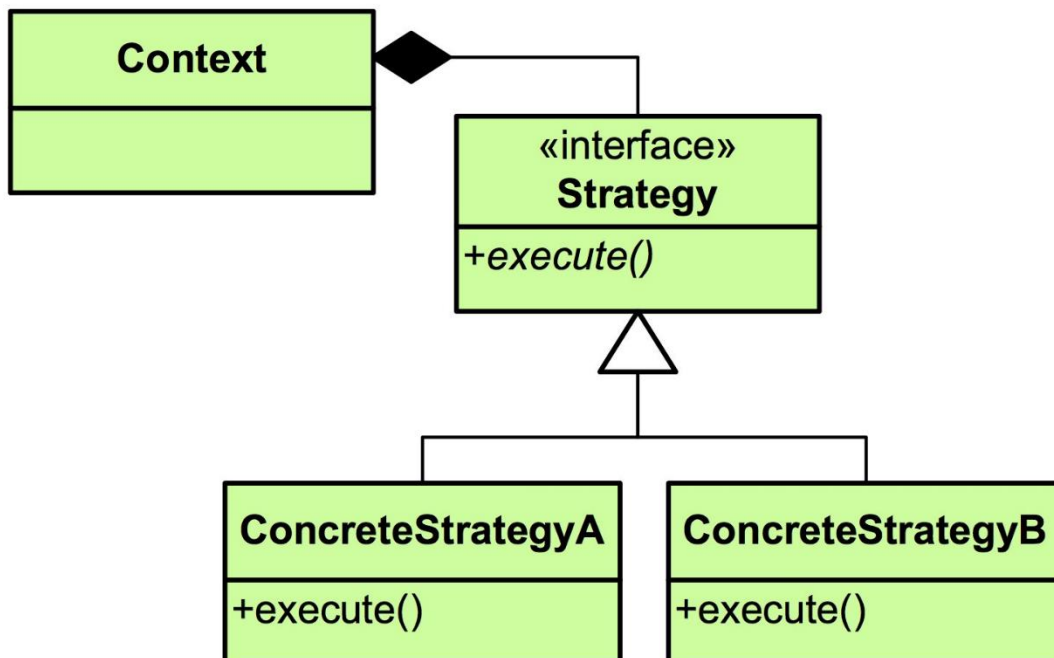


Figure 9: The strategy pattern

If you refer to figure 9, it is possible to see how this would work in the case of the synchronisation feature. “Strategy” stands for the interface. The services like Google and Outlook, implement this interface. The synchronisation application implements the interface, instead of the services. That way there is no more need to use an “if” or a “switch” statement. The structure of the directories containing services look a lot better after this upgrade as you can see in figure 10.

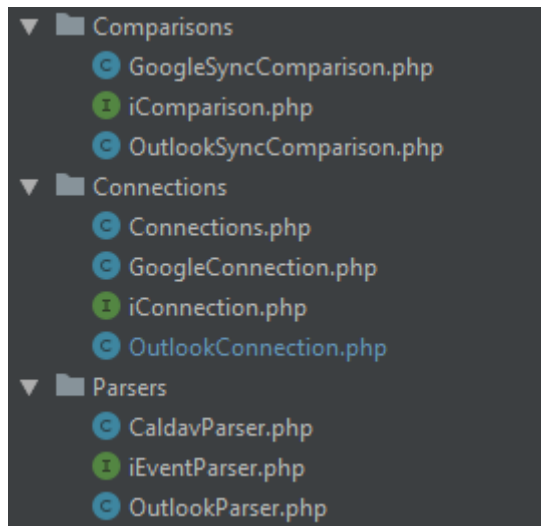


Figure 10: Folder structure with the implementation of the design pattern

Every service that derives from the iConnection class also needs the class of their service from iEventParser, and iComparison. This can now be easily done since the design pattern was implemented. Inside the connection service, the services needed, are instantiated. When the synchroniser needs parser or comparator, it will be called through the connection. The connection has a method to call the parser or comparator like shown in figure 11:

```
interface iConnection
{
    public static function getConnection();
    public function makeDbConnection($dbServerId, $user);
    public function getComparison();
    public function getServiceName();
    public function setCalendarId($calId);
    public function getCalendarId();
    public function setUser($user);
    public function createClient($user);
    public function getCTag();
    public function getAccessToken();
    public function setAccessToken($authCode);
    public function getTokens();
    public function createUri($new = false, $uid = null);
    public function checkIfEventExists($uid);
    public function refreshUserTokens();
    public function refreshToken($refreshToken);
    public function getAllEventETags();
    public function getMultipleEventsByUid($uids, $raw = false);
    public function updateServiceCalendar($vCalendar, $uid);
    public function addEvent($vCalendar);
    public function deleteEventByUid($uid);
    public function getAllEvents();

    public function addEventToQtysDb($serviceEvents);
    public function addEventToService($dbEvents);
}
```

Figure 11: iConnection interface

It is then possible to call for that method within the synchroniser:

```
class CalendarSynchroniser
{
    private $db;
    private $connect;
    private $userManager;
    private $parser;
    private $compareService;

    public function __construct($dbServerId, $user, iConnection $connection)
    {
        $this->db = CalendarConnection::getConnection($dbServerId, $user);
        $this->connect = $connection;
        $this->connect->getConnection();
        $this->connect->createClient($user);
        $this->connect->makeDbConnection($dbServerId, $user);
        $this->userManager = UserManager::getUserManager($dbServerId);
        $this->compareService = $this->connect->getComparison();
        $this->parser = $this->compareService->getEventParser();
        $this->db->setParser($this->parser);
        $this->synchroniseUserCalendars($user);
    }
}
```

Figure 12: getComparison being called

It is also possible to see that the type of the connection is not the name of the server, but the name of the interface. This assures that “\$connection” is always of the type iConnection.

Within each service, iConnection is implemented:

```
class OutlookConnection implements iConnection
```

Figure 13: Implementing iConnection

When instantiating the connection, the connection is made with the Singleton pattern. This assures that there will be one and only one instance of the connection:

```
public function __construct() {
    $this->guzzle = new Client(['base_uri' => 'https://graph.microsoft.com/']);
}

/**
 * Use this to create this object
 * @return static
 */
public static function getConnection()
{
    if (OutlookConnection::$connection == null) {
        OutlookConnection::$connection = new OutlookConnection();
    }
    return OutlookConnection::$connection;
}
```

Figure 14: Singleton pattern

Then within the same service, the class needed by it can be called through its method like shown in the next figure:

```
public function getComparison() {  
    return new OutlookSyncComparison($this->user, $this);  
}
```

Figure 15: getComparison method

When you compare the old structure to the new one, it is easy to see that it is a lot better.

Before

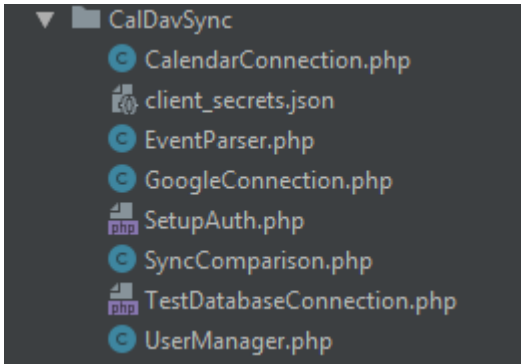


Figure 17: Old structure

After

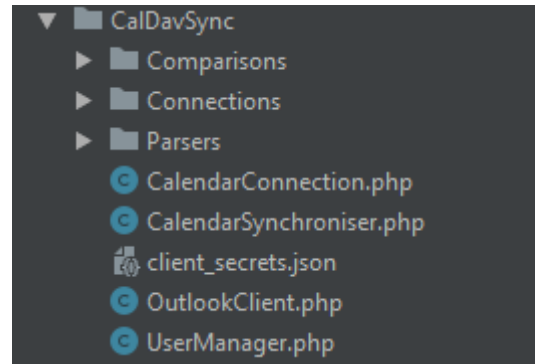


Figure 16: New structure

Now it is a lot easier to add a new service to the synchroniser when needed.

It is also worthy to mention, that Google Caldav should be updated to the new Google Calendar API. Google Caldav seems to be very old, hard to work in, and very complex. Using the new API will make things much easier and keep the application bug free. The Outlook API is a great example of this. You can compare the Outlook API with the new Google Calendar API.

Outlook

```
public function addEventToService($dbEvents) {
    foreach ($dbEvents as $event) {
        $invites = $this->db->getInvitationsByEventId($event['id']);
        if ($invites) {
            $event['invitations'] = $this->parser->extractInvitationsFromOtyaEvent($event);
        }
        $vEvent = $this->parser->parseDatabaseEventToServiceFormat($event);
        $iCalendar = $this->parser->createServiceCalendarObject($vEvent);
        $this->addEvent($iCalendar);
    }
}
```

Figure 19: addEventService in Outlook

Google Caldav

```
public function addEventToService($dbEvents) {
    foreach ($dbEvents as $event) {
        $event['uid'] = "otya-testing-event-". $event['id']. "-0003-". $event['kianid'];
        $vEvent = $this->parser->parseDatabaseEventToServiceFormat($event);
        $iCalendar = $this->parser->createServiceCalendarObject($vEvent);
        $invites = $this->db->getInvitationsByEventId($event['id']);
        if ($invites) {
            $attendees = array();
            foreach ($invites as $invite) {
                if ($calendarId = $this->db->getUserByUserId($invite['consultantid'], $invite['kianid'])) {
                    $calendarId = $calendarId['cal_id'];
                    $invite['status'] = $this->parser->formatAttendeeStatus($invite['status'], true);
                    $attendees[$calendarId] = array(
                        'PARTSTAT' => $invite['status']
                    );
                }
                $iCalendar->VEVENT->add('ATTENDEE', 'mailto:'.$calendarId, [ 'PARTSTAT' => $invite['status']]);
                $iCalendar->VEVENT->XORCHATEE = "mailto:".$invite['organizer_email'];
            }
            //Invited user ".$invite['consultantid']." has no calendar ID coupled
        }
    }

    $rawAddedEventArray = $this->addEvent($iCalendar);
    $rawAddedEvent = $rawAddedEventArray[0];
    $rawAddedEventExceptions = $rawAddedEventArray[1];
    $addedEvent = $this->parser->parseServiceEventToObject($rawAddedEvent);
    $addedEvent = $this->parser->parseEventToDatabaseFormat($addedEvent);
    $this->db->updateEventById($addedEvent, $event['id']);
    if ($rawAddedEvent->STATUS != $rawAddedEvent->STATUS) {
        $childEvents = $this->parser->getRecurringChildEvents($rawAddedEvent);
        foreach ($childEvents as $childEvent) {
            $childRecurrenceId = $childEvent->MSTART->getTime();
            foreach ($rawAddedEventExceptions as $exception) {
                if ($childRecurrenceId == $exception->['recurrence-id']->getTime()) {
                    $childEvent = $exception;
                }
            }
            $this->db->updateRecurringEventByParentId($childEvent, $event['id']);
        }
    }
    if ($invites) {
        $this->db->updateEventInvitesTag($event['id'], $addedEvent['etag']);
    }
}
```

Figure 18: addEventService in Caldav

As it can be seen. The Outlook version is a lot shorter and much easier to understand. That means that it is less likely to be bugs in there and if there are bugs, they will be easier to fix.

2.4.2 Creating the cronjob (Daemon) script

Another task of this project was to use a daemon script which ensures that the script is executed for a certain period of time. This period can be set using a crontab. This is the line of code in the crontab looks:

```
*/30 * * * * otys /usr/bin/php /data/otys-  
daemons/daemons/scripts/synchronise_calendars.php 21
```

The period is defined by the first part “*/1 * * * *”. Each space means that the next value is targeted. The first value is minutes which is from 0 to 59. An asterisk means that it is for every minute. Every minute divided by 30 means every 30 minutes. If it is not divided by 30, it would take every 30th minute. So, in this line of code the script “otys /usr/bin/php /data/otys-daemons/daemons/scripts/synchronise_calendars.php” with parameter “21” is called every 30 minutes, every hour, every day, every month and every year.

```
<?php  
/**  
 *  
 * Daemon which synchronises all calendars to calendar services like Google and Outlook  
 *  
 */  
  
use Otys\Models\CaldavModel;  
use Otys\DatabaseManager;  
use Otys\User;  
  
require_once __DIR__ . '/../../bootstraps/production.php';  
  
$dbServerId = (int)$argv[1];  
  
$db = DatabaseManager::getWhateverDbConnection();  
$query = "SELECT * FROM otys_backend_calendar_sync_configs WHERE setting = 'refresh_token';"  
$tokens = $db->getRows($query);  
  
$caldav = new CaldavModel();  
  
foreach ($tokens as $token){  
    $refreshToken = $token['value'];  
    $user = User::loadById($token['consultantid']);  
    $caldav->synchroniseUserCalendarsFromCron($dbServerId, $user, $refreshToken);  
}
```

Figure 20: Cron script

This is how the script itself looks. This is the script that is called for every certain period of time. With “\$argv[1]” the first parameter is called which was “21” in the example code shown above. Then there is a database connection made and all places where “refresh_token” is available (in other words, the user has granted access) the

“synchroniseUserCalendarsFromCron()” function will be running. To test this piece of code the “php” command can be used in the terminal to see if it works or if it gives errors.

2.4.3 Integrating Outlook Calendar

The largest part of the project was to integrate the Outlook Calendar. This part is so large because almost everything is made from scratch. Fortunately, the structure was already made, so using the example of the Google Calendar service a lot of things are copied and adjusted to work with Outlook. However, since the Google API is different it still took a lot of changes to make it work.

The link to the API given in the project task is: <https://docs.microsoft.com/en-us/previous-versions/office/office-365-api/api/version-2.0/calendar-rest-operations>. But this link was not usable anymore and it took some time to find that out. After making the OAuth connection, when trying to connect to the calendar, it will give an error about that the link is wrong. This is due to the fact that the Outlook OAuth connection expects a connection with the new Graph API of Microsoft.

At first it was an off-putting thought because it is a different API than expected. But it is a better API which offers much more functionality. The new API offers a “delta” feature. When a call is made to request events, the request also gives a delta in the response. If this delta is used in the next request, it calls all events with the exclusion of the events that were called in the last request which gave this delta value. This makes it possible to make synchronisation calls much more efficient.

The problem, however, is that the first call needs to contain a start date and an end date. The events called will be in the range of these dates. The same parameters also apply on the delta. This means that every time the end of the range is reached, the dates need to be refreshed. The solution that is used for this problem is to call all events for the past and coming five years. After five years the request will be renewed, and the new dates will be called according to the past dates. Before doing this of course, it is mandatory to make a last call using the old dates and new delta so, that all events of those dates are definitely called.

But before getting into all of this it was important to know that according to this task it is possible that Otys might want more services to add to the calendar besides Google Calendar and Outlook Calendar. Knowing that, this task is made in a way that more services can be added. For this reason, interfaces have been used. So, that if there is a new service, all there needs to be done is to plug in those interfaces and the new service will be easily added to the calendar.

The first file that had to be compared was the “Google_Client.php” file. The most important thing this file does is to manage the OAuth data. It is installed as a composer package, but the problem is that Outlook does not provide the same package. That is why “OutlookClient.php” also had to be made. To see which functions are needed in this file it was useful to connect to the OutlookClient instead of the GoogleClient. Because of this, it is now possible to see which functions are missing and which need to be changed.

Everything closed off from the service like connections to the Otys database and Otys calendars are kept the same which is why they do not have a separate interface. They are just made to run with all the services without needing to change them anymore.

There are 3 files that now have an interface because they are specifically made for the service. These are: the parser, the connection to the service, and the comparison between the service and the Otys Calendar. The connection is the first important one because that one is needed first and will point to places it is missing.

This is how the iConnection.php interface looks:

```
interface iConnection
{
    public static function getConnection();
    public function makeDbConnection($dbServerId);
    public function getComparison();
    public function setCalendarId($calId);
    public function getCalendarId();
    public function setUser($user);
    public function createClient();
    public function getCTag();
    public function getAccessToken();
    public function setAccessToken($authCode);
    public function getTokens();
    public function createUri($new = false, $uid = null);
    public function checkIfEventExists($uid);
    public function refreshToken($refreshToken);
    public function getAllEventETags();
    public function getMultipleEventsByUid($uids, $raw = false);
    public function updateGoogleCalDav($vCalendar, $uid);
    public function addEvent($vCalendar);
    public function deleteEventByUid($uid);
    public function getAllEvents();
}
```

Figure 21: iConnection.php

An important thing to mention is that if the OutlookConnection class is called which derives from this interface, the comparator of Outlook is also called and there is never a need for another comparator to call. The best solution to do this is to add a “getComparison();” method. Which will return an “OutlookComparison()” class deriving from the “iComparison()” interface. This method can be used to call the right service for the needed interface per service every time.

However, since PHP 5.5 does not yet allow to typecast or return a specific type, PHPStorm will give errors because it does not know that the returned type is the expected one and cannot make a connection to its methods.

The same way in iComparison, the “getEventParser()” method will be called so that its methods can be used.

```
interface iComparison
{
    public function getEventParser();
    public function updateEtags();
    public function isSyncNeeded();
    public function updateCtag();
    public function getEventsToAddToCaldav();
    public function getOwnedEventsToAddToOtys();
    public function getInvitedEventsToAddToOtys();
    public function getEventsToUpdate($user);
    public function getInvitesToUpdate($user);
    public function getUidsToDelete();
}
```

Figure 22: iComparison

In the piece of code in the figure above of “OutlookConnection.php” it can be seen that this is called as a Singleton to make sure that there is only one instance of this class. Same goes for the other classes because it is not needed more than once. It can also be seen that the “getComparison()” method calls a new OutlookSyncComparison() class. This connection will be made at the beginning so that it can be accessed whenever it is needed.

```
class OutlookConnection implements iConnection
{
    private static $connection;
    private $db;
    private $calid;
    private $baseUri = 'https://apidata.googleusercontent.com/caldav/v2/';
    private $user;
    private $guzzle;
    private $events;
    /** @var OutlookClient**/
    protected $client;

    public function __construct() {
        $this->guzzle = new Client(['base_uri' => 'https://graph.microsoft.com/']);
    }

    /**
     * Use this to create this object
     * @return static
     */
    public static function getConnection()
    {
        if (OutlookConnection::$connection == null) {
            OutlookConnection::$connection = new OutlookConnection();
        }
        return OutlookConnection::$connection;
    }

    /**
     * @return OutlookSyncComparison
     */
    public function getComparison() {
        return new OutlookSyncComparison($this->user, $this);
    }

    public function makeDbConnection($dbServerId){
        $this->db = new CalendarConnection($dbServerId);
    }
}
```

Figure 23: OutlookConnection

So next, all files come together in the file “CalendarSynchroniser.php”. This is the actual “application” file that executes everything.

```
namespace Otys\Calendar\CalDavSync;

use Otys\Calendar\CalDavSync\Connections\iConnection;
use Otys\Calendar\CalDavSync\Comparisons\iComparison;

class CalendarSynchroniser
{
    private $db;
    private $connect;
    private $userManager;
    private $parser;
    private $compareService;

    public function __construct($dbServerId, $user, $refreshToken, iConnection $connection)
    {
        $this->db = CalendarConnection::getConnection($dbServerId);
        $this->connect = $connection;
        $this->connect->getConnection();
        $this->connect->createClient();
        $this->connect->makeDbConnection($dbServerId);
        $this->userManager = UserManager::getUserManager($dbServerId);
        $this->compareService = $this->connect->getComparison();
        $this->parser = $this->compareService->getEventParser();
        $this->synchroniseUserCalendars($user, $refreshToken);
    }

    public function synchroniseUserCalendars($user, $refreshToken)
    {
        //region Define user for this iteration
        $this->userManager->setUser($user);
        $this->connect->setUser($user);
        $this->db->user = $this->userManager->getUser();
        $this->connect->setCalendarId($this->userManager->getUserCalendarId());
        $this->connect->refreshToken($refreshToken);
        //endregion
    }
}
```

Figure 24: CalendarSynchroniser

In the figure above in the constructor can be seen that every class first gets initialised. So, that they are ready to use once the application (the synchroniseUserCalendars() function) is executed. The basic user and OAuth connection information is set in the method itself.

2.4.4 The parser

Every service needs to have a parser since none of the calendar services will have the exact same structure as the Otys calendar. What the parser essentially does is convert the event from the service format to the Otys format. And from the Otys format to the service format.

To illustrate this an Outlook event is added to the Otys calendar. The JSON representation of the Outlook event would look like this:

```
{
  "attendees": [{"@odata.type": "microsoft.graph.attendee"}],
  "body": {"@odata.type": "microsoft.graph.itemBody"},
  "bodyPreview": "string",
  "categories": ["string"],
  "changeKey": "string",
  "createdDateTime": "String (timestamp)",
  "end": {"@odata.type": "microsoft.graph.dateTimeTimeZone"},
  "hasAttachments": true,
  "iCalUID": "string",
  "id": "string (identifier)",
  "importance": "String",
  "isAllDay": true,
  "isCancelled": true,
  "isOrganizer": true,
  "isReminderOn": true,
  "lastModifiedDateTime": "String (timestamp)",
  "location": {"@odata.type": "microsoft.graph.location"},
  "locations": [{"@odata.type": "microsoft.graph.location"}],
  "onlineMeetingUrl": "string",
  "organizer": {"@odata.type": "microsoft.graph.recipient"},
  "originalEndTimeZone": "string",
  "originalStart": "String (timestamp)",
  "originalStartTimeZone": "string",
  "recurrence": {"@odata.type": "microsoft.graph.patternedRecurrence"},
  "reminderMinutesBeforeStart": 1024,
  "responseRequested": true,
  "responseStatus": {"@odata.type": "microsoft.graph.responseStatus"},
  "sensitivity": "String",
  "seriesMasterId": "string",
  "showAs": "String",
  "start": {"@odata.type": "microsoft.graph.dateTimeTimeZone"},
  "subject": "string",
  "type": "String",
  "webLink": "string",

  "attachments": [ { "@odata.type": "microsoft.graph.attachment" } ],
  "calendar": { "@odata.type": "microsoft.graph.calendar" },
  "extensions": [ { "@odata.type": "microsoft.graph.extension" } ],
  "instances": [ { "@odata.type": "microsoft.graph.event" } ],
  "multiValueExtendedProperties": [ { "@odata.type": "microsoft.graph.multiValueLegacyExtendedProperty"
}],
  "singleValueExtendedProperties": [ { "@odata.type":
"microsoft.graph.singleValueLegacyExtendedProperty" } ]
}
```

This does not match with the fields of an Otys event.

The fields of an Otys event look like this:

```
'id'=>'number',
'parent_id'=>'string',
'consultantid'=>'number',
'klantid'=>'number',
'begintijd'=>'number',
'eindtijd'=>'number',
'titel'=>'string',
'locatie'=>'string',
'omschrijving'=>'string',
'datumtoevoegen'=>'number',
'usertoevoegen'=>'number',
'datumaanpassen'=>'number',
'useraanpassen'=>'number',
'agendaitem_type'=>'string',
'prive'=>'boolean',
'whole_day'=>'boolean',
'uid'=>'string',
'etag'=>'string',
'repeat_count'=>'number',
'recurrence_type'=>'number',
'sequence'=>'number',
'raw_recurrence'=>'string',
```

Because they need to match each other in order to synchronise them with each other, the events are just converted to the other side. That is where the parser comes in and does the job of converting the event.

The hardest parts are not the parts of converting the columns themselves but converting the parts that need special input. For example, recurrences. A recurrence within an event means that the event is repeated over a specified period like weekly, monthly or yearly. And also, for how many times it repeats. Like 2 weeks, 3 weeks, or maybe 10 weeks. The problem is that every calendar handles this logic in a different way. In the piece of code above, it can be seen that Otys just uses the fields 'repeat_count' and 'recurrence_type'. Using these two columns, Otys can make a calculation and execute the logic.

However, the way Outlook handles this is totally different. Instead of two columns, Outlook expects a lot more details in order to be more dynamic. Here is an example of the JSON representation of it:

```
"recurrence": {
  "pattern": {
    "type": "weekly",
    "interval": 1,
    "daysOfWeek": [ "Monday" ]
  },
  "range": {
    "type": "endDate",
    "startDate": "2017-09-04",
    "endDate": "2017-12-31"
  }
}
```

Notice that there is a lot more logic to be handled in this.

It is harder to convert those parts between calendars than just plain text. The method created to convert the recurrence of an Outlook event to the Otys event is like this:

```
private function serviceRecurrenceToOtys($recurrence) {
    $otysRecurrence = [];
    switch ($recurrence['pattern']['type']) {
        case "daily":
            $otysRecurrence['recurrence_type'] = 2; //Each day
            break;
        case "weekly":
            $otysRecurrence['recurrence_type'] = 3; //Each week
            break;
        case "absoluteMonthly":
            $otysRecurrence['recurrence_type'] = 8; //Each month
            break;
        case "absoluteYearly":
            $otysRecurrence['recurrence_type'] = 6; //Each year
            break;
        default: //Relative monthly and relative yearly doesn't exist in Otys
            $otysRecurrence['recurrence_type'] = 3;
            break;
    }
    if($recurrence['range']['type'] == "numbered"){
        $otysRecurrence['repeat_count'] = $recurrence['range']['numberOfOccurrences'];
    }
    else{
        $startDate = new \DateTime($recurrence['range']['startDate']);
        $endDate = new \DateTime($recurrence['range']['endDate']);
        switch ($otysRecurrence['recurrence_type']){
            case 2: //Each day
                $otysRecurrence['repeat_count'] = $startDate->diff($endDate)->d;
                break;
            case 3: //Each week
                $otysRecurrence['repeat_count'] = floor(($startDate->diff($endDate)->d)/7);
                break;
            case 8: //Each month
                $otysRecurrence['repeat_count'] = $startDate->diff($endDate)->m;
                break;
            case 6: //Each year
                $otysRecurrence['repeat_count'] = $startDate->diff($endDate)->y;
                break;
        }
    }
    return $otysRecurrence;
}
```

Here it can be seen that the many values that Outlook has, are being converted to just two values in Otys. Of course, this means that some values need to be sacrificed and have a default setting instead.

2.4.5 The connection

The connection classes are another important part of the communication between the calendars. The logic of every traffic coming from the service is handled in here. The logic of everything going to the Otys database is handled in the CalendarConnection.php file. That is how those two are being separated to keep things more organised. CalendarConnection.php does not need to be separated per service but rather is being called by the services instead, because the Otys events always need to be added, updated or deleted the same way.

The connection classes derive from the interface iConnection.php and all have the purpose to make the authentication and communicate with the API of the belonging service.

An example of this would be to add an event to the service (in this case to Outlook):

```
public function addEvent($vCalendar)
{
    $headers = array(
        'Content-Type: application/json; charset=utf-8',
        'Authorization: Bearer ' . $this->getAccessToken()
    );
    $uri = 'https://graph.microsoft.com/v1.0/me/calendar/events';

    $doc = new \DOMDocument('1.0', 'utf-8');
    $doc->formatOutput = true;

    $event['id'] = $vCalendar['otys_id'];
    unset($vCalendar['otys_id']);
    unset($vCalendar['repeat_count']);
    unset($vCalendar['recurrence_type']);

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $uri);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($vCalendar));
    curl_setopt($ch, CURLOPT_VERBOSE, 1);
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

    $response = curl_exec($ch);
    $response = json_decode($response);
    $this->db->updateEventUid($event, $response->id);
    $this->db->updateEventEtag($event, $response->{'@odata.etag'});
    curl_close($ch);

    return $response;
}
```

2.4.6 OAuth verification

The OAuth verification is used to authenticate to see which calendar is going to be used. It asks the user to log in and accept the permissions that Otys needs in order to complete the synchronisation tasks. This procedure is very straightforward once it has been done, the information is saved to the database so that the connection can persist to exist. Every call to the API needs the token given by OAuth in order to accept the requests.

2.4.7 The comparator

The comparator seems to be like the parser but is not. The parser converts the events from one service to another. But the comparator checks if both the service and the Otys calendars are the same. If they are, nothing has to be changed. If they are not, then actions need to be taken. If there is an event deleted in the Otys calendar but still existing in the calendar of the service, then steps need to be taken to remove the event in the service as well. All that kind of logic is being handled by the comparator.

An example of that can be seen here:

```
private function getUidsToUpdate($fromInvites = false) {
    $uidsToUpdate = array();
    if (!$fromInvites) {
        foreach ($this->otysOwnedEtags as $id => $values) {
            $etag = $values['etag'];
            $uid = $values['uid'];
            if (array_key_exists($uid, $this->caldavEtags) && $etag != $this->caldavEtags[$uid] && $etag != 400) {
                $uidsToUpdate[] = array(
                    'uid' => $uid,
                    'id' => $id
                );
            }
        }
    } else {
        foreach ($this->otysInvitedEtags as $id => $values) {
            $etag = $values['etag'];
            $uid = $values['uid'];
            if (array_key_exists($uid, $this->caldavEtags) && $etag != $this->caldavEtags[$uid] && $etag != 400) {
                $uidsToUpdate[] = array(
                    'uid' => $uid,
                    'id' => $id
                );
            }
        }
    }
    return $uidsToUpdate;
}
```

All ids of all events that need to be updated are being put in an array and returned so that the program knows which events need to be updated. From here on, further steps can be taken to complete the process.

3 Reflection

The intern task to add services to the Otys calendar was interesting and fun to do although it did have its fair share of moments of stress. Implementing the already made service was hard because I was not familiar with the Otys code when I came here. After getting used to it, it got easier but still, it was one of the least fun things to do because I had to work the code of someone else.

The part of implementing Outlook was more fun. As someone who already had knowledge of programming, the usage of interfaces to make the application more professional, universal and cleaner was a different and challenging thing to me. It is something I want to be a lot better in terms of design patterns and SOLID programming. After being able to put this the best way I knew possible it made me proud, and the next time I will be facing this challenger I want to do it even in a better way so that every program I make will reach its maximum performance.

Another thing I learned which was very interesting was the way calendars work. To update an event according to the newest data without colliding to another calendar changing it at the same time will definitely cause problems. If two calendars want to update the same data at the same time, they need to know of each other which one is the newest. To solve this problem, calendars use group ids which is a unique number of the update happening to the group of data. This unique number will tell if it is the most recent version of the group of data updated. This way it is possible to know if the calendar needs to be updated on each or one of the sides of the service, or not. There is also a unique number for each event. Because the service also needs to know if each event needs to be updated or not to save time. And a third unique ID to determine if the event is colliding with another event that is at the same time being updated in order to detect collision and perhaps ask the client which version he or she wants to keep. This is a common problem in the world of a programmer, and I am glad to have faced it early on. After all these kinds of problems are what makes a programmer grow.

4 Research topic

During the first days in Otys the environment had to be set up and the code and structure of the software had to be learned. The project is made in AngularJS, but at a much larger scale and much older version of AngularJS (1.3) than expected. At such a large scale of a project it is hard to keep things organized without a proper structure.

For this project the structure is hard to maintain, mainly because AngularJS is not made for this job. Since it is not even component based yet, it is hard to keep track of everything and everyone does things differently which is why the same things are solved differently by different people. For this reason, the project code is complex and fixing this will enhance the development speed many times.

At this moment it became very clear that by upgrading the project from AngularJS to Angular would speed up the performance of this company by a lot.

5 Research method

There are at least three ways to tackle this problem [1]. These ways are:

- Total conversion
- ng-forward
- ngUpgrade
- Ui-router Angular Hybrid

5.1 Total conversion

As explained earlier, this way of upgrading is not a realistic strategy at all for a project this size. To do this the old project has to be converted to the new version.

So, it is not known during the conversion if everything will work with each other. The pieces of code cannot be tested or validated. This will be depressing because things will keep getting in the way.

However, if this was a small project, this would still be an option.

5.2 ng-forward

Ng-forward is a third-party library which can be written in AngularJS in Angular 2 style. This way it can be gradually upgraded towards Angular.

However, this one also has its flaws. For example, the templates are not compatible with Angular. So, everything needs to be adjusted to work with Angular instead. This way the same work will be repeated twice.

Also, because it is a third-party library it is not supported and maintained by Google itself so there is no guarantee on how long and good it will work. That is why this is not an ideal option either.

5.3 ngUpgrade

Another method is the ngUpgrade method which is made by Google itself and fully satisfies the needs of this project to migrate in a seamless way to Angular.

With ngUpgrade what has to be done is run both AngularJS and Angular alongside each other. This immediately shows a downside of this solution which is that two projects are executed at the same time instead of one.

So, the benefits of using ngUpgrade is that it can be built incrementally. That means that the project can be kept the way it is. And upgrade (or better yet migrate) to Angular each component one by one. So, it does not matter if it has to be completed in a week, or a year. The project itself will not be disturbed by this process.

Another thing is that it is created by Google. Which means it will be supported by the creators of Angular itself, so it is safe to assume that it is dependable.

Since this is a hybrid solution which fits the company needs to migrate to the new Angular, this will be the most dependable method to do further research on.

Also, within the ngModule migration method, there are two different commonly used methods to approach:

- Vertical Slicing
- Horizontal Slicing

Vertical slicing is a case in which the migration starts at the top level of each module and work it down to the lower levels. This method is recommended for larger projects, but it has a downside. While working down the code there will most likely be duplicate code to keep the application from breaking.

In horizontal slicing, however, it is meant to start at the lowest level of each module and work up from there. This way there will not be any duplicate code and it will be easy to start with.

The downside of this method is that it is to understand the code because both versions exist at the same time in the same module. For that reason, this method is recommended for smaller projects, because it is harder to work this way in larger teams.

5.4 Hybrid Router

Another way to migrate as advised by Mitch Dries here at Otys is to use the ui-router angular hybrid module. For this method it is not possible to skip the conversion to ES6 because the module itself is written in ES6.

The way this module works is that there are two lists of routes. One for AngularJS and one for Angular. Once a component is done being migrated. The route can be set from the AngularJS component to the Angular component route. This will allow for an easy way to migrate.

The conclusion to this is that for this project, it is better to use either the vertical slicing method, or the Hybrid Router method, because the project is at a very large scale, and the teams are as well. This way the performance will increase at a faster rate, it will be easier to debug and there will be better motivation since it is not as depressing to work in it anymore.

6 Elaborating on the research

6.1 What is the AngularJS framework? [2]

As it was said by its official documentation “Angular is a structural framework for dynamic web apps” [3]. Which means that the HTML content (templates) can be dynamically changed using JavaScript as its model.

The most important concepts of AngularJS are:

- Views
- Scopes
- Controllers

As explained before the HTML part is the view which can be changed dynamically using the “backend”. The backend in this case would be the scope and the controller.

The scope makes the connection between the controller and the view. Inside a controller scopes can be defined, which will be bound to the view. An example of this is:

```
var newApp = angular.module('angularApp',[]);
newApp.controller('NewController', ['$scope', function($scope) {
  $scope.message = 'This is the welcome message!';
}]);
```

In the code above, the controller “NewController” is made in “newApp”. Inside this controller there is a scope defined called “message” with the value “This is a welcome message!”.

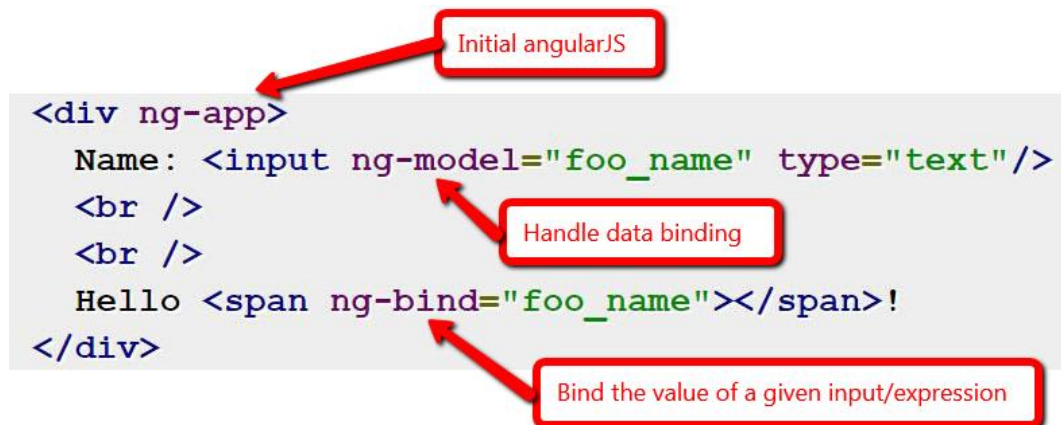
So, to output this inside the view (HTML) the following can be done:

```
<div ng-controller="NewController">
  {{ message}}
</div>
```

What is done here is that the controller is called which was named “NewController”. Inside the div the scope named “message” is called. So, this will display “This is a welcome message!” inside the div. Double curly brackets are used to let AngularJS know that the scope data is going to be accessed.

A real powerful tool, for example, is the data binding. Using data binding the data in the JavaScript part as well as the html part can be changed at the same time.

```
<div ng-app>  
  Name: <input ng-model="foo_name" type="text"/>  
  <br />  
  <br />  
  Hello <span ng-bind="foo_name"></span>!  
</div>
```



The diagram shows three red callout boxes with arrows pointing to specific parts of the code: 'Initial angularJS' points to the opening <div ng-app> tag; 'Handle data binding' points to the ng-model attribute in the input tag; and 'Bind the value of a given input/expression' points to the ng-bind attribute in the span tag.

Figure 25: Data binding example

In the figure above it can be seen that in the first place (ng-app) the AngularJS application is defined. Because of this AngularJS knows that it has to initiate itself here. The second thing that can be seen is the “ng-model” attribute. This defines the variable “foo_name” to the content of the input field. The last thing seen in the example is the “ng-bind” attribute. This tells AngularJS that it requires the data inside “foo_name”. So, as soon as any change happens inside the input field, the text between the “span” tags will change to that as well.

A feature like this makes the lives of frontend developers a lot easier. This is also how a framework differs from a library. Because when a library is used, the code can be overseen, and the library can be used. While when a framework is used, the framework is in charge and it uses the code when it needs it.

So, the user calls the specific functions of a library in the code while a framework is wrapped around the code.

AngularJS takes charge of the structure of the code and this way it keeps things more organised. Larger projects are easier to manage because of this.

But that is not everything that AngularJS can provide for a developer. Like other frameworks it is a higher level of abstraction which is at the cost of flexibility. So, for most web applications such a framework will be ideal, because it has done a lot of the work that needs to be done in any of those projects already in advance. This saves a lot of time.

However, this cost of flexibility, of course, means that it is not suited for everything. Things that need to manipulate a large amount of times like games are not suited to be made in such a framework. For that, a framework like Unity can be used.

6.2 What is Angular and how is it better? [4]

6.2.1 What is the Angular framework?

The Angular framework is a lot like AngularJS in terms of being a frontend framework. However, Angular is structured better and has a lot more features. AngularJS is what they call the versions of Angular until 2.0. From version 2.0 and up it is called Angular. The reason the name was changed is, because it the code transformed when it changed to Angular. The most obvious reason is that AngularJS uses JavaScript (hence JS) and Angular uses Typescript. But besides this, there have been many changes in structure and feature-wise.

6.2.2 Why is it better than AngularJS?

6.2.2.1 JavaScript vs Typescript

As mentioned before, Angular uses Typescript as opposed to AngularJS which uses JavaScript. However, this does not mean that JavaScript cannot be used in Angular instead. It is also possible to just uses JavaScript inside Angular. A benefit of this is that it makes the migration from AngularJS to Angular so much easier since it can be done simultaneously because of this. But how is it possible that both can be used? Typescript exists within JavaScript and so it is a superset of ES6 (JavaScript version 6).

6.2.2.2 Speed performance of AngularJS vs Angular

One reason to migrate towards Angular would be the speed. Once the migration is complete, the project will be five times faster than AngularJS. This is mostly because it contains AOT, which will be explained later.

6.2.2.3 The learning curve of AngularJS vs Angular [5]

As one of the advantages that AngularJS has is that it is much easier to learn. It is not as complicated as Angular. AngularJS is straightforward and everything is easier to understand. But Angular does not aim to be easy to learn. Instead it aims to be more professional, faster, and structured. Thus, it will be more maintainable regarding large projects as opposed to AngularJS which is a better choice for small projects.

This means that for the company Otys, it is highly recommended to start migrating to Angular as it is a too large of a project for AngularJS.

6.2.2.4 The ahead of time (AOT) compiler of Angular [6]

One of the biggest advantages of Angular is that it has an ahead of time compiler. The AOT compiler converts the HTML code and Typescript code into efficient JavaScript code in the server. So, the client will receive the code as efficient JavaScript code which is easier and faster to render. Also, because the HTML and CSS are combined, they do not need to be called for separately which helps to make fewer asynchronous requests.

AOT also provides more security because the code is compiled in the server. This helps to prevent a lot of injection attacks.

There are two different options as to compile the application with the AOT compiler. One of which is the JIT compiler which compiles during runtime. And the other is the AOT compiler which like explainer compiles during runtime. JIT is the default option that Angular provides this can be changed to use AOT by enabling this option when serving the application. But it is built in production mode, it will always use AOT.

6.2.2.5 Angular components

Like explained before in the AngularJS part, AngularJS uses scopes to send data between a view and a controller. It also has a rootScope in which variables can be defined which have to be used on every page. But Angular has a much better approach towards this. It uses a hierarchy of components. A component is a directive in AngularJS with a template. This helps the project to be a lot more organised and well structured.

6.2.2.6 Babel

This is a very important step which can cause a lot of trouble if not done right or save a lot of time if done right. Babel allows ES6 files to be converted to ES5. This means that all new JavaScript code is converted to old JavaScript code and so they are compatible with older browsers and older frameworks. Since Otys is using an old version of AngularJS, this is a step that cannot be forgotten. To use new modules written in ES6, Babel has to be configured. Once that is done, it will be converted to the old JavaScript and it will not be causing errors anymore.

This definition, however, is very specific to the configuration that is going to be used for Otys. In general, it is possible to convert any version of JavaScript to be compatible with each other as a pre-set. And Babel is many more possibilities than just this.

6.2.2.7 Webpack

Webpack is one of the few known module loaders. Using a module loader, NPM modules can be loaded in a much easier way. They are available from everywhere instead of having to give up the whole directory for it. It handles everything that needs to be handled within the module.

In the case of Otys this is not the only positive thing about Webpack. Another thing is that Otys can finally get rid of the lib and bower_components directories because they will be obsolete.

6.3 Preparing the migration from AngularJS to Angular [3]

It is hard to find a decent example on how to start on doing this. However, a great example can be found on the Angular docs itself.

A good thing to start out with after doing the preparations is to integrate Typescript since Angular works with that.

To do this, typescript is installed with:

```
npm i typescript --save-dev
```

Next thing is to install all packages that belong to Angular with this command:

```
npm install @types/jasmine @types/angular @types/angular-animate  
@types/angular-cookies @types/angular-mocks @types/angular-resource  
@types/angular-route @types/angular-sanitize --save-dev
```

To configure the Typescript compiler which generates the JavaScript files a tsconfig.json file is needed. The following configurations will be used for this project:

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "target": "es5",  
    "moduleResolution": "node",  
    "sourceMap": true,  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "lib": [ "es2015", "dom" ],  
    "noImplicitAny": true,  
    "suppressImplicitAnyIndexErrors": true  
  },  
  "exclude": [  
    "node_modules"  
  ]  
}
```

Now to run the compiler, this must be added to the package.json file

```
"scripts": {  
  "tsc": "tsc",  
  "tsc:w": "tsc -w",  
}
```

And run it in the command-line with:

```
npm run tsc:w
```

As soon as this is done, the compiler will start watching and recompiling the changes that were made. If there are no TypeScript files it will give an error about it, but since these will be made there is no need to worry about it.

So, citing from Angular “*Since TypeScript is a super-set of ECMAScript 2015, which in turn is a super-set of ECMAScript 5, you can simply switch the file extensions from .js to .ts and everything will work just like it did before.*” This sounds to be too good to be true. And it is. The first problem faced during this process is the module loader. Otys is using RequireJS as their module loader. The problem with this is that it is based on AMD and not ES6.

AMD is a module definer which uses the old JavaScript however, TypeScript uses ES6 which is the newer JavaScript. To convert every file to TypeScript, all AMD files need to be converted to ES6.

The best way to tackle this problem is the opposite of what most would think. Using a JavaScript compiler, all JavaScript files need to be converted to the old JavaScript. Compared to upgrading all to the new JavaScript, this gives the advantage of all browsers being able to support the JavaScript files.

Babel is the JavaScript compiler that can do this for us. If set up right, it will compile all code to the old JavaScript before running it. That is how the newer and older versions of JavaScript will be able to work alongside each other.

There is also another problem that will make things a lot easier if solved. Otys is still using Bower. Bower is an outdated package manager and for that reason it is smart to upgrade to NPM instead.

An amazing feature which enhances NPM is Webpack. This is a great reason to upgrade to NPM instead of Bower. Using Webpack packages can be easily imported without being dependent of the path of the packages. Webpack will handle everything very conveniently.

So, to upgrade to NPM and use Babel the best approach would be to first configure Webpack. To do this a couple packages need to be installed in node, or in the case of Otys, using Yarn because Otys already changed to Yarn which is the same but a lot of times it causes fewer problems. The following packages are needed:

- Webpack
- Webpack-dev-server
- Webpack-cli

To install these using yarn the following command can be used:

```
yarn add webpack webpack-dev-server webpack-cli
```

Now yarn will add all three packages that needs to be configured in Webpack. After doing that, a file named “webpack.config.js” has to be made. This configuration file will be executed whenever Webpack ran. The file needs to be created at the root of the project. So, for Otys this would be in the folder “go”. To define the configurations, JSON needs to be used. So, everything will go in a JSON array. The JSON array needs to be given to “module.exports”. So that would mean “module.exports = (JSON Configuration array)”.

The first thing that needs to be done in the configuration file is to point the entry file. The entry file is the file that contains the JavaScript going to the source code. In the case of Otys, this file is located in “go/app/js/app/main.js”. To point to that entry file, the property “entry” will be given the path like so:

```
module.exports = {
  entry: './app/js/app/main.js';
}
```

Now the entry file is known. It is possible to convert this file, and all the files that are being used within to the output file that it is configured to be. To do this, the output file has to be set as well. However, since paths are not always the same on every system. It is wise to use the “path” module. This module will make paths safer. To implement this path, add it using “yarn add path”. Then to put the output file and implement the “path” module the following has to be added to the configuration:

```
const path = require("path");

module.exports = {
  entry: './app/js/app/main.js';
  output: {
    path: path.resolve(__dirname, 'app/build'),
    filename: '[name].js'
  }
}
```

This means that the output file will be written to the location “go/app/build” and the name of the file will be equal to the name of the entry file using “[name]”.

The next step is to configure the rules. Within these rules, babel has to be loaded and used. But also, the “node_modules” folder needs to be excluded. The reason for excluding this folder is because Webpack should not compile that folder, because it does not need it. That folder is already being used by Webpack on its own way without converting it. It is, however, possible to convert it. But it will drastically lower the performance.

So, the final configuration file for Otys will be:

```
const path = require("path");

module.exports = {
  entry: './app/js/app/main.js';
  output: {
    path: path.resolve(__dirname, 'app/build'),
    filename: '[name].js'
  },
  module: {
    rules: [{
      exclude: [/node_modules/],
      loader: 'babel-loader',
      query: {
        presets: ['@babel/env']
      }
    }]
  }
};
```

So, in the loader, Webpack will now load Babel and know how to use it. In the query attribute the preset needs to be defined. The preset is defined as “@babel/env”. This tells Webpack that Babel needs to compile the JavaScript files to a version that is compatible for all versions.

Before being able to proceed from this step, the Babel packages need to be installed. The following packages are needed for that.

- @babel/core
- @babel/preset-env
- babel-loader

We do the same to do this with the yarn command:

```
yarn add @babel/core @babel/preset-env babel-loader
```

After this is done, a change in package.json needs to be made. In the “scripts” attribute, a script needs to be added that will run webpack and compile the file that is needed. All that needs to be added is the following line.

```
build-webpack: "webpack"
```

Now in the command line the following has to be executed:

```
yarn run build-webpack
```

The webpack command will be executed, and the file will be compiled into the path specified.

Next step is to direct the project to the compiled output file rather than the entry file. This is mainly done in the main HTML file. However, for Otys this step is different. The main html file is actually multiple files which is index.html and modular.html. But index.html will not be used anymore so, it was advised to implement this on modular.html instead. Modular.html loads RequireJS as multiple scripts, but not the main entry file. The main entry file is configured within the configuration file of RequireJS which is located in “go/app/js/require/config.js”. Within this file there is a packages attribute. The first package to be loaded is the main.js file. It has to be directed to the output file, instead of the entry file. So, this has to be changed:

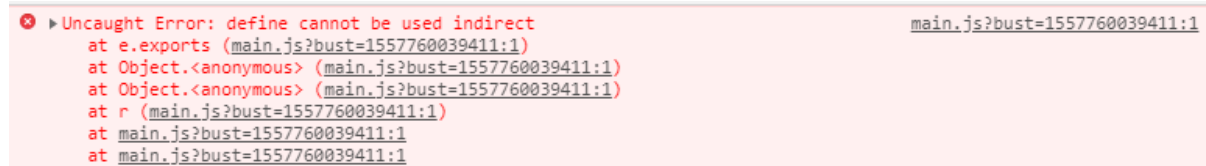
```
packages: [{
  name: 'app',
  location: 'js/app'
}]
```

To this:

```
packages: [{
  name: 'app',
  location: 'build'
}]
```

It will automatically select the “main.js” file unless specified differently.

Now at this point a problem exists. In the website console the following error will occur:



```
Uncaught Error: define cannot be used indirect
at e.exports (main.js?bust=1557760039411:1)
at Object.<anonymous> (main.js?bust=1557760039411:1)
at Object.<anonymous> (main.js?bust=1557760039411:1)
at r (main.js?bust=1557760039411:1)
at main.js?bust=1557760039411:1
at main.js?bust=1557760039411:1
```

Figure 26: Error because define is used in a wrong way

After doing a lot of research and asking around on Stackoverflow [7], it was possible to conclude that this error is caused by a module that is not compatible to be compiled. To be able to find out by which error the problem is caused, the “noParse” attribute has to be used in the Webpack configuration file.

```
module: {
  noParse: /node_modules/
}
```

The code above shows how to let Webpack not parse node modules. Instead of the whole node modules directory, each module has to be selected separately to be able to find out which module is causing this problem. Since this will take a lot of time, it will not be possible to fix this problem at the moment.

The compiled file contains this piece of code:

```
function (e) {
  var n = {};

  function r(t) {
    if (n[t]) return n[t].exports;
    var o = n[t] = {id: t, parentId: 0, exports: {}};
    return o.call(o.exports, o, o.exports, r), o.l = 10, o.exports;
  }

  r.m = e, r.c = n, r.d = function (e, n, t) {
    r.o(e, n) || Object.defineProperty(e, n, {enumerable: !0, get: t});
  }, r.r = function (e) {
    "undefined" != typeof Symbol && Symbol.toStringTag && Object.defineProperty(e, Symbol.toStringTag, {value: "Module"});
  }, r.t = function (e, n) {
    return e[n];
  }, r.u = function (e, n) {
    if (!1 && n && (e = r(e)) && 0 && n) return e;
    if (!4 && n && "Object" == typeof e && e && e.__esModule) return e;
    var t = Object.create(null);
    if (r.r(t), Object.defineProperty(t, "default", {enumerable: !0, value: e})), 2 && n && "string" != typeof e) for (var o in e) r.d(t, o, function (n) {
      return e[n];
    });
    return t;
  }, r.n = function (e) {
    var n = e && e.__esModule ? function () {
      return e.default;
    } : function () {
      return e;
    };
    return r.d(n, "default", n);
  }, r.o = function (e, n) {
    return Object.prototype.hasOwnProperty.call(e, n);
  }, r.p = ""; r(r.s = 0);
}({function (e, n, r) {
  var t;
  void 0 === (t = function () {
    "use strict";
    r(1)('app', ['lib/angular', 'lib/has'], function (e, n) {
      return e.module('app', ['ngAnimate', 'ngSanitize', 'angular-require']);
    }), function (e) {
      console.error(e);
    }
  )
  .apply(n, []) || (e.exports = t);
}, function (e, n) {
  e.exports = function () {
    throw new Error("define cannot be used indirectly");
  }
});
```

Figure 27: Compiled JavaScript code

When the code is compared to the entry file code which is written in AMD:

```
define([module: 'app', deps: [
  'lib/angular',
  'lib/has',
  'lib/angular-sanitize',
  'lib/angular-animate',
  'lib/angular-component',
  'lib/angular-strap',
  'lib/bootstrap-select',
  'ng',
  'app/environment', //Build with grunt
  'ng!app/ui-router',
  'ng!app/locale', //Build with grunt
  'ng!app/lazyLoader',
  'ng!app/user/modules',
  'ng!app/ows',
  'ng!app/files',
  'ng!app/notifications',
  'ng!app/chat',
  'ng!app/messaging',
  'ng!app/utills',
  'ng!app/topBar',
  'ng!app/REELIa',
  'ng!app/entities',
  'ng!app/modules',
  'ng!app/components/phone',
  'ng!app/components/window', // for sidebar
  'css!lib/bootstrap-select',
  'css!lib/fullcalendar.css', // I put it separate as I need it to be loaded before app
  'css!lib/bootstrap.css',
  'css!lib/bootstrap-responsive.css',
  'css!css/app.css',
  'async!css!fonts/font.css',
  'async!css!lib/fontawesome-pro/svg-with-js/css/fa-svg-with-js.css',
  'lib/fontawesome-pro/svg-with-js/js/fontawesome.min.js',
  'font/fontawesome/optimized/regular.js', //Build with gulp
  'font/fontawesome/optimized/solid.js', //Build with gulp
  'font/fontawesome/optimized/light.js', //Build with gulp
  'font/fontawesome/optimized/brands.js', //Build with gulp
  'css!css/print.css!print'
], payload: function (angular, has) {
  "use strict";

  return angular.module('app', ['ngAnimate', 'ngSanitize', 'angular-require'])
```

Figure 28: Code to the entry file

It can be seen that it is written in a much different way.

There is a reason as to why they are so different from each other. The first reason is that after compiling, the code is minimized to make the code as efficient as possible with the least amount of code and procedures. This is a good thing and that is why the understandable variable names are changed to a, b, c and so forth.

After solving this problem, it is possible to get rid of Bower components and the whole project will already be at a higher level, because all packages will be easily accessible. That will save a lot of code and make it possible to convert all files from AMD to ES6.

The conversion can be done with tools like the one mentioned before at:

<https://www.npmjs.com/package/amd-to-es6> [8]. But there are files that this script is not compatible with. That is why some files need to be configured manually which is fairly easy to do.

```
define( moduleName: [
  'lib/require',
  'lib/angular',
  'lib/lodash',
  'lib/moment',
  'lib/angular-bootstrap',
  'ng!app/components/timerpicker'
], deps: function (require, angular, _, moment) {
  "use strict";

  return angular.module('app/activities/directives/info', [])
    .directive('activityInfo', ['ows', function (ows) {
      return {
        restrict: 'A',
        templateUrl: require.toUrl('../partials/activityInfo.html')
        require: '^activityDetail',
        scope: true,
        link: function (scope, element, attributes, controllers) {
          scope.$watch('activityUId', function (activityUId) {
            controllers.getDetail(activityUId)
              .then(function (activity) {
                scope.$evalAsync(function (scope) {
                  scope.activity = activity;
                })
              });
          });

          angular.extend(scope, {
            edit: edit,
            cancel: cancel,
            save: save
          });
        }
      };
    })
  ];
});
```

Figure 29: Example AMD code

The piece of code above is an example of AMD. It does not matter how small or how big the code is. All that has to be changed is the define function. The packages that are imported in the define function, are now being imported with the “import” function instead.

Here is an example of the code converted to ES6.

```
import angular from 'angular';
import require from 'lib/require!';
import 'lib/angular';
import 'ng!app/c... Module is not installed more... (Ctrl+F1)
import ' from 'lib/lodash';
import moment from 'lib/moment';

export default angular.module( name: 'app/activities/directives/info', requires: []
  .directive( name: 'activityInfo', directiveFactory: ['ows', function (ows) {
    return {
      restrict: 'A',
      templateUrl: require.toUrl('../partials/activityInfo.html'),
      require: '^activityDetail',
      scope: true,
      link: function (scope, element, attributes, controllers) {
        scope.$watch('activityUId', function (activityUId) {
          controllers.getDetail(activityUId)
            .then(function (activity) {
              scope.$evalAsync(function(scope) {
                scope.activity = activity;
              });
            });
        });
      };
      angular.extend(scope, sources: {
        edit: edit,
        cancel: cancel,
        save: save
      });
    }
  });
});
```

Figure 30: Example ES6 code

It can be seen in the figure above that the code looks much cleaner and shorter in ES6. But that is not the only advantage. It is also possible to see that the editor gives a warning when a package is not available. It will also give the opportunity to solve it by itself by installing the module. Another great benefit of it is that the packages that are not being used, will be grayed out. This way it is easier to know which packages are just a waste of processing time. It is also a good idea to convert all bower components to node modules. The package “require” is being extracted from the “lib” directory. By adding requirejs to NPM it is possible to just say “import require from ‘requirejs’” instead. This way the package will be loaded from NPM and is much safer. Otys currently relies on the location of the packages. And since the root of Otys is not available to access on the website, it cannot use packages from the root folder. But since Webpack will compile all of that already if it is configured right, there is no need to worry about it.

A last thing that should be mentioned is the export default part. This has to be done in order to export the code and be recognized by the project.

And that is it. That is all it takes to convert a file from AMD to ES6. However, since Otys is such a large application and have hundreds of files, it will not be easy to do this for all files manually. That is why using a script to do the most automatically is recommended.

After having everything converted to ES6. It is now possible to just change the extension of all files to “.ts” from “.js”. Since this alone brings also a lot of benefits with it along with the changes done before, it will have a big impact on the application in terms of performance, safety, and readability. Which means fewer errors, and faster productivity.

After having done the conversion to TypeScript, it is now possible to use the “Angular Hybrid” module. To do this, all that has to be done is [9]:

```
yarn add @uirouter/angular-hybrid @angular/common @angular/compiler
@angular/core @angular/platform-browser @angular/platform-browser-dynamic
@angular/upgrade
```

Now along with the uirouter module, Angular is also installed. After this step, the “<ng-app>” attribute needs to be removed from everywhere. This is because it will be done in JavaScript from now on. Then, in “go/app/js/app/ui-router.js”, “ui.router.upgrade” has to be added like so:

```
define(['lib/angular', 'lib/lodash', 'lib/angular-ui-router'], function
(angular, _) {
  var modalUrlMatcherList = [];
  return angular.module('app/ui-router', ['ui.router',
  'ui.router.upgrade'])
    .factory('stateChangePromise', ['$rootScope', '$q',
function($rootScope, $q) {
  return function() {
    return $q(function(resolve, reject) {
      var _offSuccess = $rootScope.$on('$stateChangeSuccess',
function(e) {
        _offSuccess();
        _offCancel();
        _offError();
        resolve(e);
```

In the example above it can be seen that the ui.router.upgrade package is added to angular.module.

After that. The file NgModule has to be made in the “app” folder. This is the first file that belongs to Angular and not AngularJS. Inside this file AngularJS needs to be connected to Angular.

Then, the modules BrowserModule, UpgradeModule and UIRouterUpgradeModule.forRoot() need to be imported within the file.

All providers can be created within the “providers” attribute.

The ngDoBootstrap function is the part that combines AngularJS with Angular.

```
@NgModule({
  imports: [
    BrowserModule,
    // Provide angular upgrade capabilities
    UpgradeModule,
    // Provides the @uirouter/angular directives and registers
    // the future state for the lazy loaded contacts module
    UIRouterUpgradeModule.forRoot({ states: [contactsFutureState] }),
  ],
  providers: [
  ]
})
export class SampleAppModuleAngular {
  constructor(private upgrade: UpgradeModule) { }

  ngDoBootstrap() {
    this.upgrade.bootstrap(document.body,
    [sampleAppModuleAngularJS.name], { strictDi: true });
  }
}
```

Within “go/app/js/app/main.js”, UI-Router needs to be told that it should wait until all bootstrapping is done before synchronizing the URLs. To do that, this line has to be added:

```
ngmodule.config(['$urlServiceProvider', ($urlService: UrlService) =>
$urlService.deferIntercept()]);
```

After that the application has to be told how it should be bootstrapped by adding:

```
platformBrowserDynamic()
  .bootstrapModule(AppModule)
  .then(platformRef => {
    // Intialize the Angular Module

    // get() the UIRouter instance from DI to initialize the router
    const urlService: UrlService =
platformRef.injector.get(UIRouter).urlService;

    // Instruct UIRouter to listen to URL changes
    function startUIRouter() {
      urlService.listen();
      urlService.sync();
    }

    platformRef.injector.get < NgZone > NgZone.run(startUIRouter);
  });
```

It is now possible to route the AngularJS routes like it was already done for each component in Otys. But as it can be seen below, Angular components are registered with the “component” attribute instead of “templateUrl” and “controller”.

```
var foo = {
  name: 'foo',
  url: '/foo',
  component: 'fooComponent'
};
stateProvider.state(foo);

var bar = {
  name: 'foo.bar',
  url: '/bar',
  templateUrl: '/bar.html',
  controller: 'BarController'
};
stateProvider.state(bar);
```

From this point on, it is just a matter of making the Angular side of each component, where after the route has to be redirected to the Angular route like shown above.

6.4 Conclusion

Even though the research did not reach the last part, it is definitely possible to upgrade to Angular. However, it will be a very long process. First of all, once the migration process finished, every component from then on has to be made in the new Angular. Since there are many developers, it will not be too easy to keep control of this. Another hardship will be the components that need to be migrated to the new Angular. It seems the best way would be to put a developer to migrate component by component. While this component is being migrated, other developers should not work on the same component while the migration is in process. Once it is done, the bug fixes and new features need to be made in the Angular part and not in the AngularJS part. This will not be a problem, because once the component has been migrated, the AngularJS version will no more respond. The process of migration will take a lot of time, but it will not slow down the development process of the overall application.

The easy and also very beneficial part of this project to migrate to Angular is the first steps. The steps of converting everything to Typescript. Unfortunately, because there is an error that needs to be fixed, it was not yet possible to make use of these advantages. But if it is fixed, the project will already have a lot of benefits like being able to import packages in a very easy and secure way. The way that packages are currently handled are time consuming and missing out on the benefits of the new JavaScript ES6. Webpack and Babel are already configured but it seems Babel is not configured the right way yet. After the error has been solved, every file can be converted to ES6 and to the extension “.ts” making it TypeScript.

It is not a hard process to fix the bug with the broken node module. The problem is that there is currently not enough time. Spending some time on this problem will allow it to be easily fixed.

References

- [1] Rangle.io, "Tutorial: How to Start Using Angular 2 with Your Angular 1.X Code Base," [Online]. Available: <https://www.youtube.com/watch?v=ucUy0CoN57Q&t=1466s>.
- [2] Angular, "What Is AngularJS?," Google, [Online]. Available: <https://docs.angularjs.org/guide/introduction>.
- [3] Angular, "PhoneCat Upgrade Tutorial," Google, [Online]. Available: <https://angular.io/guide/upgrade#phonecat-upgrade-tutorial>.
- [4] B. Hartmann, "Taking the leap to go from AngularJS to Angular," UruIT, [Online]. Available: <https://www.uruit.com/blog/angular-1-vs-2-migrate/>.
- [5] J. Bandi, "Angular 2 vs. AngularJS: A teacher's perspective," Medium, [Online]. Available: <https://medium.jonasbandi.net/angular-2-vs-angularjs-a-teachers-perspective-d7e10ba29ede>.
- [6] Angular, "The Ahead-of-Time (AOT) compiler," Google, [Online]. Available: <https://angular.io/guide/aot-compiler>.
- [7] S. Samet, "Stackoverflow," [Online]. Available: <https://stackoverflow.com/questions/56060050/getting-define-cannot-be-used-indirect-error-when-bundling-with-webpack>.
- [8] jonbretman, "AMD to ES6 module transpiler," NPMJS, [Online]. Available: <https://www.npmjs.com/package/amd-to-es6>.
- [9] UI-Router, "UI-Router angular-hybrid," [Online]. Available: <https://github.com/ui-router/angular-hybrid>.