Google Apps for Work Calendar Integration

Vladimir Janous

September 2015

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Computing for Business

Computing Science and Mathematics
University of Stirling

Abstract

Youmanage is a Stirling based company offering a human resources management web application of the same name. One part of the application is an Exchange calendar service allowing its users to receive appointment calendar notifications. Such service inserts relevant system appointments into user calendars based on their company email address.

However, not all of Youmanage clients use Microsoft Exchange server to administer their emails and therefore clients showed a great demand for a service which would support other email providers than Exchange in the same way. In addition as calendar service provides only a basic functionality of inserting calendar events, Youmanage wanted to explore the possibility of extending its current ability.

Further communication with clients revealed that Google Apps for Work calendar service support would create additional business value and served as the main objective of this project. Youmanage application already included a design part for extending the calendar service and therefore new service had to fit within such restrictions. As a secondary objective, Youmanage wanted to explore the possibility of implementing a two way communication between their application and Google calendar resource servers. Furthermore an extension of the new calendar service functionality in a form of new methods responding to the change of system appointments represented another interest of Youmanage.

At first an in-depth analysis of the Exchange calendar service functionality was conducted and an appropriate software development approach was adopted. As the analysis realised unfeasibility of the additional objectives, adopted development approach was flexible enough to accommodate another set of requirements requested by Youmanage throughout the development process. A large part of the design period was dedicated to a research regarding Google API and web-application interactions and can serve as a universal guide for interested readers. In addition a possibility of publishing a Google Marketplace application to ease service configuration was explored.

The output of this project is a new calendar service implemented within the restrictions of Youmanage application offering the same functionality as Exchange calendar service. Moreover configuration of such service can be performed either manually or automatically using a published Marketplace application. In addition a synchronization tool for clients not using a calendar service at the moment was implemented allowing them to insert previous appointments into relevant user calendars. Finally, Youmanage devoted their time testing all implemented features to their satisfaction and have intentions releasing them to its clients.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following:

- Figure 1 and 2 in *Chapter 2* are screenshots of the Youmanage application test instance prior to the development process.
- Figure 4 in *Chapter 2* was generated using Visual Studio IDE and reflects a state of the calendar service prior to its development.
- Figures 5, 7, 8 in *Chapter 2* and Figure 15 in Section 5.5 were generated using Microsoft SQL Management Studio and reflect the state of the data layer.
- Figure 9 in *Chapter 3* was taken from [3].
- Code described in Section 6.3.3.4 was largely taken from [12].

In addition, as classes GoogleServiceProvider and GoogleAppointment implement same interfaces as their Exchange counterparts, they were largely inspired by them.

Signature **Date**

Acknowledgements

I would like to thank my supervisor Dr. Mario Kolberg whose help kept the project on track and formulated special requirements adopted by the client when some of the objectives became unfeasible. I would also like to thank Dr. David Cairns for his observations regarding the project which could be accommodated in a future development.
Moreover I would like to dedicate a special thanks to the team of Youmanage HR Ltd., especially to Jason, Maurice, Gillian and Nick, as without their help the project development would have been uncertain.
Finally, I would like to thank my family and my inspirational girlfriend Lea for all their support throughout the project.

Table of Contents

List of Figures

1 Introduction

1.1 Background and Context

Youmanage is a software company based in Stirling and their product is a human resource management web application of the same name.

The application consists of various modules for managing employees. Such modules often work with appointments created within the system and referring to absences, performance reviews and other facts of employee cycles.

A popular feature of the application is an Exchange calendar service which allows inserting relevant system appointments into employee calendars based on their company emails.

As some of Youmanage clients don´t use Microsoft Exchange there was a great opportunity for extending such service and supporting other email providers. The positive response received from Youmanage clients served as a base for this dissertation project.

1.2 Scope and Objectives

The project was carried out with an agreement among a student, University of Stirling and the client (Youmanage HR Ltd). University acted as an intermediate offering the student a valuable industry experience and the client a possibility to explore new business areas without the need to use their resources. The student undertook the project working as a full-time employee in the clients company with a promise to deliver a final product to the client.

The main agreed objectives consisted of developing a new calendar service supporting other popular email providers (Google and Apple) and offering the identical behaviour as Exchange service.

If time permits additional objectives were formulated as extending the new calendar service either by implementing a “two-way communication” or by implementing additional service methods manipulating calendar events based on the state of system appointments.

The concept of two way communication explored the possibility of creating system appointments based on relevant calendar events in user calendars and vice versa.

Additional service methods supposed to reflect the state of system appointments in user calendars and in particular be able to respond to delete and update actions within the system.

1.3 Achievements

A new calendar service interacting with Google Apps for Work business tools was delivered to the client as well as some additional features formulated by the client during the development. Such features include a Google Marketplace application offering an automated service configuration and a synchronization tool providing an ability to insert system appointments created before any calendar service was configured.

Due to the facts realised during the project development an Apple calendar service couldn´t be accommodated not even if time permitted. Moreover in consultation with the client and time frame left for the project additional objectives extending a new calendar service were not implemented in order to ensure the consistency of offered calendar services. However exploring the possibility of two way communication remained as a research topic and formulated some basic recommendations.

The delivered functionality was fully tested by the client to its satisfaction and is intended to be used in the future applications update.

1.4 Overview of Dissertation

This paper uses a chaptered structure where *Chapter 1* sets the course of the project and summarise the corresponding results. The initial state of the project prior to the development period is described in *Chapter 2* providing essential information related to the front end as well as the back end of the application. Consequently, *Chapter 3* defines and explains a chosen software development approach related to the project. *Chapter 4* provides more information of the project objectives and additional requirements formulated by the client and also reveals encountered issues. *Chapter 5* explores the design of the final solution and provides an essential information about Google technologies used as well as appropriate approach. *Chapter 6* describes the implementation of the Google calendar service alongside with other additional features from the front end and back end perspective. *Chapter 7* then provides the testing options and its results as well as discussing its limitations. Finally, *Chapter 8* summarises the project and compares the achievements with project objectives providing a background and guidelines for a future development.

2 State-of-The-Art

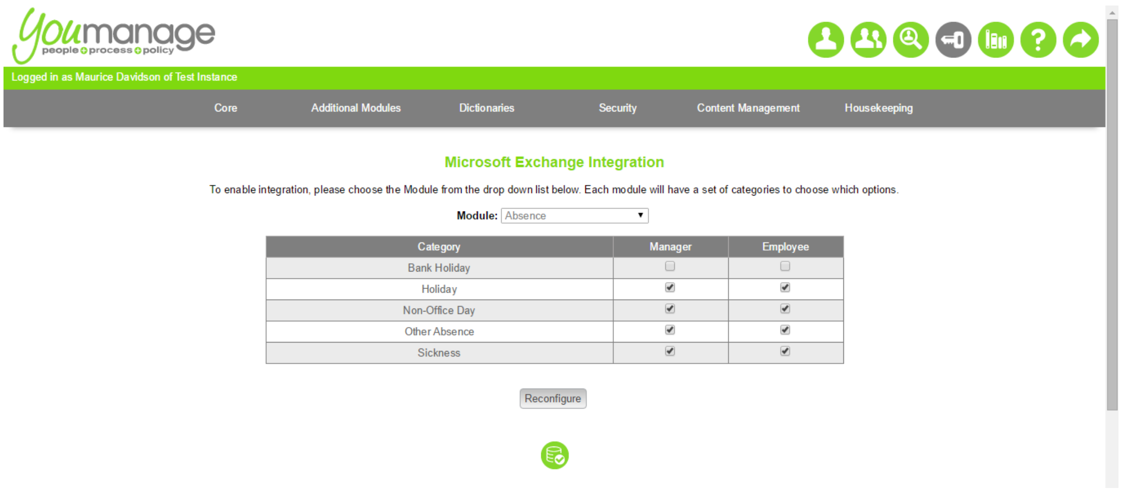
2.1 Existing system

2.1.1 Overview

Youmanage is a web-based HRM application designed to ease and automate various HR processes within an organisation. The application consists of numerous integrated modules that each represents an administrative tasks and activities of an employee. Modules are accessible based on user access levels and therefore the system defines four possible roles of a user as: Managers, HR users, self-serve employees and administrative users [1]. One of the tasks within the system is a calendar service which offers inserting various appointments from the system to employee/manager calendars associated with their Outlook company emails. Calendar service is a popular functionality and as such clients of Youmanage showed a great demand for a calendar service supporting other than Microsoft technologies. The most essential sections of the system regarding this service are summarised below.

2.1.2 Admin section

This section is accessible by users with administrator rights and provides a configuration of the entire system. It allows administrators to configure/reconfigure the Exchange calendar service and set up a permission scope in order to insert event. Such scope defines what kind of events the service is allowed to work with and also what kind of employees can receive a calendar notification.



Logged in as Maurice Davidson of Test Instance

Microsoft Exchange Integration

To enable integration, please choose the Module from the drop down list below. Each module will have a set of categories to choose which options.

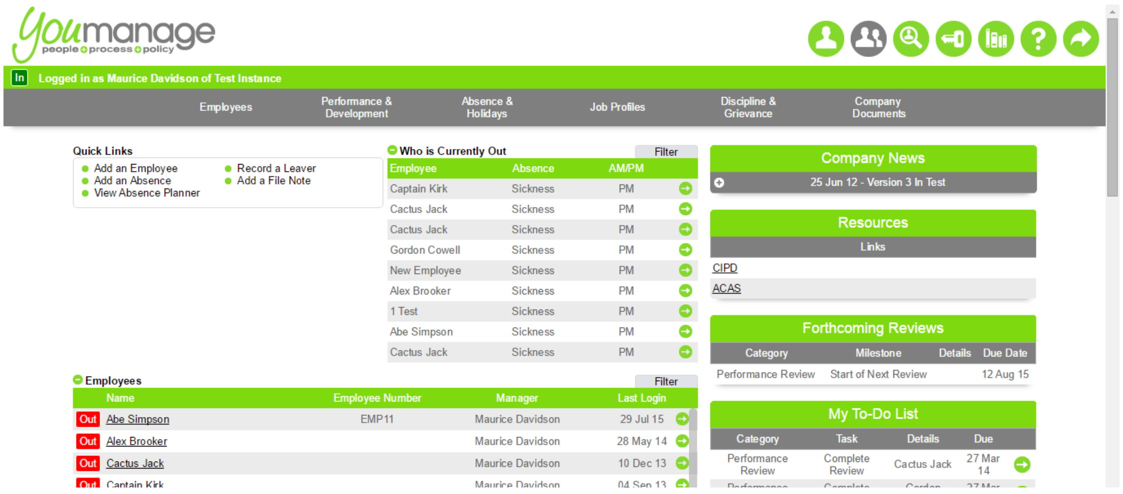
Module: Absence

Category	Manager	Employee
Bank Holiday	☐	☐
Holiday	✓	✓
Non-Office Day	✓	✓
Other Absence	✓	✓
Sickness	✓	✓

Figure 1. Admin Exchange Service Settings

2.1.3 Manager section

This part of the system provides various tools for managers which are essential to manage employees. Users of the system with an access to this section can create performance reviews, disciplinary records, absences and other appointments. Creating or approving an appointment based on the permission scope set by administrators may trigger the calendar service.



Logged in as Maurice Davidson of Test Instance

Employees | Performance & Development | Absence & Holidays | Job Profiles | Discipline & Grievance | Company Documents

Quick Links
- Add an Employee
- Add an Absence
- View Absence Planner
- Record a Leaver
- Add a File Note

Who is Currently Out

Employee	Absence	AM/PM
Captain Kirk	Sickness	PM
Cactus Jack	Sickness	PM
Cactus Jack	Sickness	PM
Gordon Cowell	Sickness	PM
New Employee	Sickness	PM
Alex Brooker	Sickness	PM
1 Test	Sickness	PM
Abe Simpson	Sickness	PM
Cactus Jack	Sickness	PM

Employees

	Name	Employee Number	Manager	Last Login
Out	Abe Simpson	EMP11	Maurice Davidson	29 Jul 15
Out	Alex Brooker		Maurice Davidson	28 May 14
Out	Cactus Jack		Maurice Davidson	10 Dec 13
Out	Captain Kirk		Maurice Davidson	04 Sep 13

Company News

25 Jun 12 - Version 3 In Test

Resources

Links
CIPD
ACAS

Forthcoming Reviews

Category	Milestone	Details	Due Date
Performance Review	Start of Next Review		12 Aug 15

My To-Do List

Category	Task	Details	Due
Performance Review	Complete Review	Cactus Jack	27 Mar 14

Figure 2. Manager Module

2.1.4 System architecture

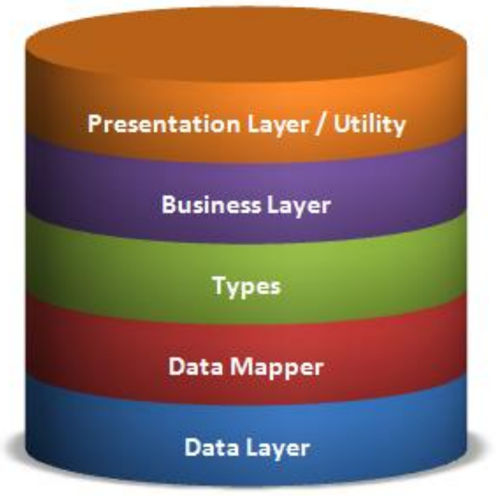


Figure 3. Youmanage Application Architecture

Youmanage web application is running on ISS server and therefore is implemented using Microsoft technologies. Most of the application is still written in .NET Visual Basic programming language, however larger parts are being currently re-written in C#. Due to its ongoing development over significant period of time and many different developer teams engaged the system architecture is very inconsistent. The application is equipped with two different object-relational mappers where the older one is a custom made ORM and the newer one is entity framework (EF). Entity framework is a standard part of .NET technologies offered by Microsoft. In addition vast parts of the system use "code-behind model" in order to practice a separation of presentation and content where the current emphasis is on "model-view-controller" architecture instead.

As we can see (Figure 3) the application consists of five layers, each with a different responsibility. In order to separate their responsibilities each layer can only communicate with layers surrounding it.

The bottom layer consists of Data Layer which is represented by Microsoft SQL Server database and its main purpose is to store and retrieve persistent data. Every client of Youmanage runs their own instance of the application and therefore has an own database to manipulate data. The next layer expresses an interface between a database management system and the application itself where both object-relational mappers are engaged in order to transform data from a database to a corresponding class representation and vice versa. As a result Types layer is a set of classes determining individual tables of the database manipulated by object-relational mappers and Business layer. The latter contains the main logic and methods of the system and is the core of the application. Finally Presentation layer represents a graphic user interface, reflects its state and process user inputs. On contrary Utility layer has no graphical interpretation and contains services which interacts with systems out of the application and without any user engagement. Such service is Exchange calendar service.

2.2 Design of Exchange Calendar Service

2.2.1 Overview

If configured, Exchange calendar service provides user's notifications of various appointments stored in the Youmanage application. Such notifications are in a form of calendar events, which are inserted in their relevant Microsoft Exchange company accounts.

The configuration of calendar service takes place in the Admin module where administrators of the system need to provide configuration details of their Microsoft Exchange servers. Among these details is a location (URL) of the Exchange server, server version, service account with configured impersonation and its credentials. Impersonation allows the service to act on behalf of users from the Microsoft Exchange domain without their explicit consent and therefore such technique needs to be configured within the domain beforehand. If URL of the server is not provided, service offers administrators an "auto discover" function which will try to get the URL based on an email address provided. Successful connection to the Exchange server is accompanied by a successful message, unsuccessful one by an error message. After the successful service setup administrators have either a possibility to reconfigure the service if desired or adjust permissions the service accepts. In order to alter permissions the service depends on, administrators select a module from a dropdown list of all available modules which filters all available notifications offered by the selected module. Administrators can then decide what kind of employees should receive notifications by changing values of check boxes provided. Notifications can either be enabled for both – employee and manager, one or the other, or none of them. Even though the initial dropdown list contains all available modules, at the time of writing this paper only few of the modules are supported and also not all notifications are implemented.

From the user's point of view the actual calendar events are inserted into relevant calendars in a real time as an approved appointment is created. The calendar service inserts appointments on the background and therefore user is uncertain if the task was accomplished successfully or not as no success/fail messages are provided. Finally, the provided service will not create any notifications if an appointment already exists within the system.

Perhaps it might be worth to mention that Exchange calendar service has never been properly tested as Youmanage doesn´t have an Exchange server at their disposal and in addition many of its methods although implemented are unused.

2.2.2 Logical Part

While engineering the Exchange calendar service, programmers involved already bearded in mind a possible extension to a different calendar service providers and therefore a sufficient abstraction was provided during the design period. As a result various interfaces have to be implemented by any new calendar service (Figure 4).

IAppointment interface was designated in order to convert different system appointments into a calendar event used by the particular calendar service. The concrete implementation of this interface is represented by an ExchangeAppointment class. This class holds a private property of type Appointment provided by Microsoft Exchange and implements the interface methods which populate such property with data from the concrete system appointment.

IAppointmentService interface then contains methods for manipulating appointments and every class implementing such interface has to provide methods to create, save, update, delete, cancel and get an appointment.

Last general interface provided is an IExternalService which contains methods representing the service connection details. These methods take care of the connection to relevant servers, network credentials set up, error codes and messages handling as well as service disconnection.

The Exchange calendar service itself then implements two other Exchange specific interfaces. IExchangeAppointmentService interface extends IAppointment service and adds a method to check for appointment conflicts on the Exchange server side. This interface is then extended by IExchangeService interface which also extends IExternalService and provides methods for impersonation and Exchange server auto discovery.

As a result the concrete Exchange calendar service has to implement only IExchangeService interface which contains all other interfaces.

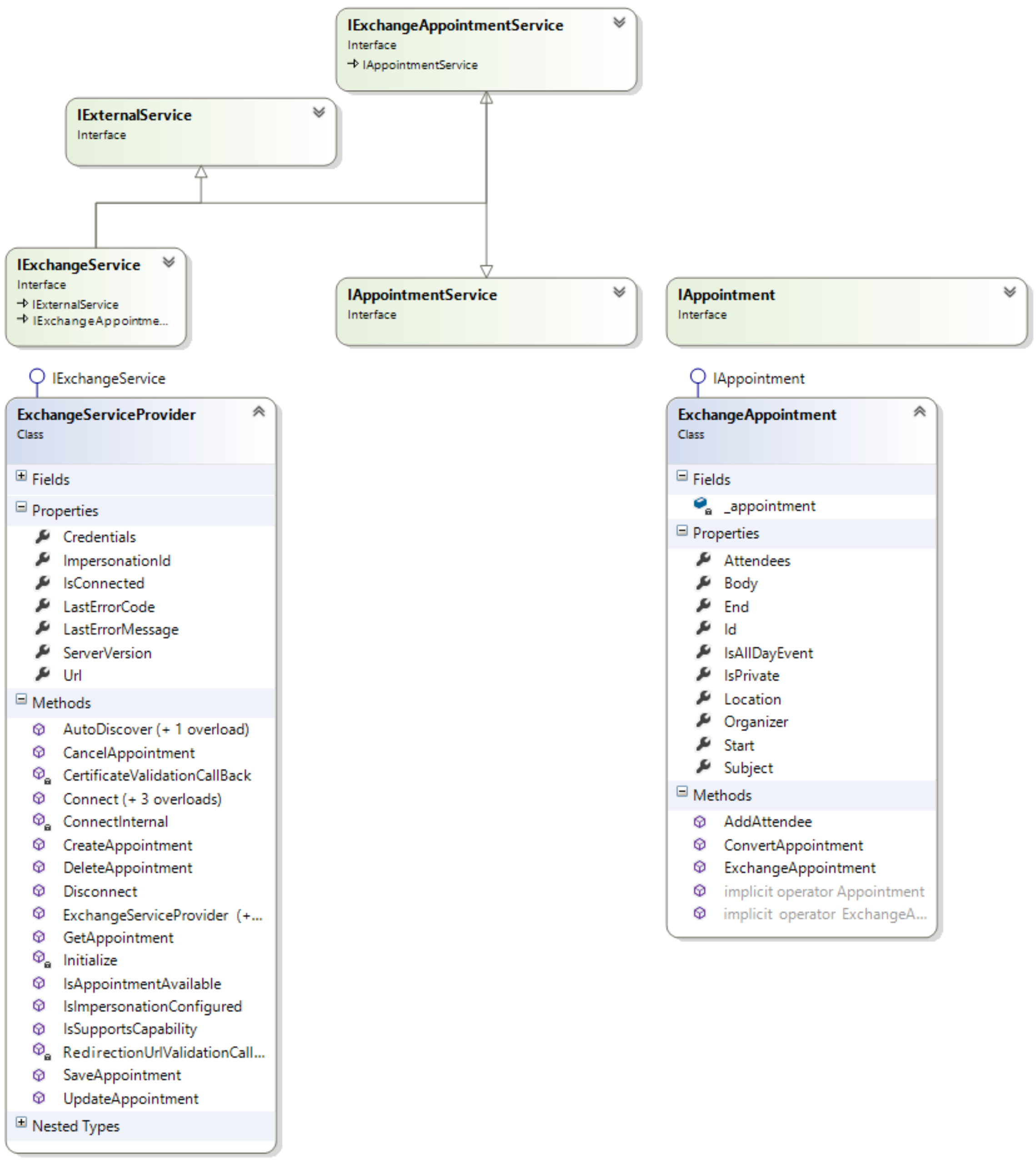
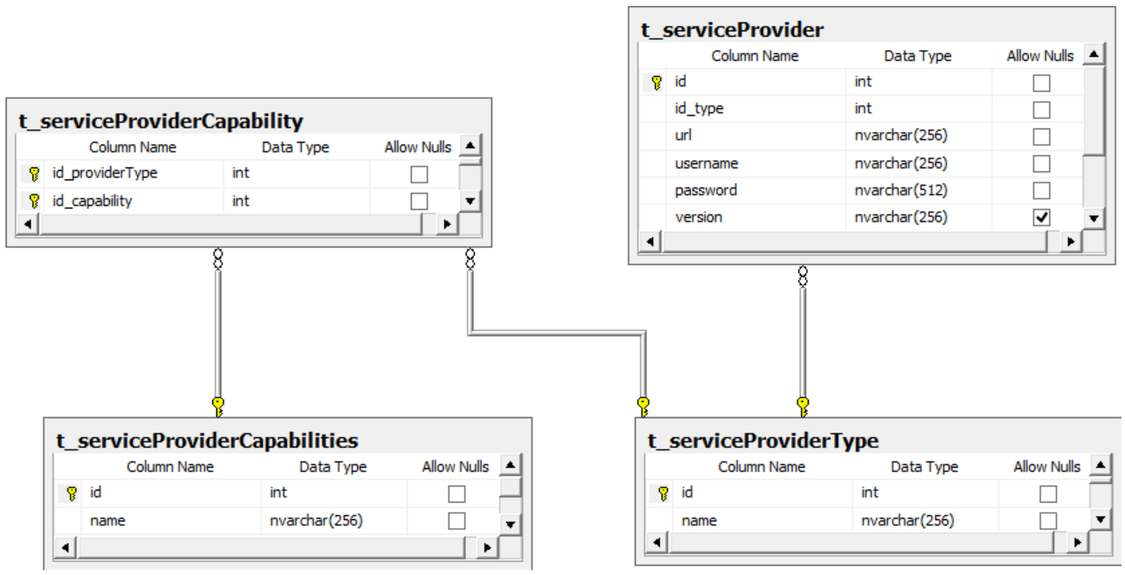


Figure 4. Exchange Service UML

The final ability of an implemented service is to convert any given single system appointment into an Exchange event and insert it into a single employee calendar based on a provided company email of the user.

2.2.3 Data Part

As stated in the earlier parts of this paper, before the calendar service can be used, administrators must set up the service by providing required information regarding their Exchange server. Such information has to be stored permanently as its essential for running the service. Youmanage application uses Microsoft SQL Server to store all persistent data and its relations in database tables. Each name of a table within the system follows a conventional naming t_tablename (Figure 5) which corresponds to Data Types and Data Accessor classes provided by object relational mapper. Such classes have names without the database prefix "t_" [1].



t_serviceProviderCapability

	Column Name	Data Type	Allow Nulls
🔑	id_providerType	int	☐
🔑	id_capability	int	☐

t_serviceProvider

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	id_type	int	☐
	url	nvarchar(256)	☐
	username	nvarchar(256)	☐
	password	nvarchar(512)	☐
	version	nvarchar(256)	✓

t_serviceProviderCapabilities

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	name	nvarchar(256)	☐

t_serviceProviderType

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	name	nvarchar(256)	☐

Figure 5. Calendar Service ER diagram

2.2.3.1 t_serviceProviderCapabilities

This table contains all service capabilities within the Youmanage application as it's expected that a single service will be able to offer more than just calendar notifications. It contains a unique identifier of a service capability in the form of an integer and a Unicode description hence an nvarchar type.

2.2.3.2 t_serviceProviderType

The architecture of this table is identical to the previous one, however the purpose is to store concrete services provided by Youmanage.

2.2.3.3 t_serviceProviderCapabilities

Since a single service can provide more than one capability and a single capability can be provided by more than one service the previous tables have a many to many relationship. In consequence this table serves as a decomposition table as such relationship can be decomposed into two one to many relationships where this table contains only unique identifiers of previous tables.

2.2.3.4 t_serviceProvider

This last table stores the required settings of concrete services obtained from the administrator front end of the application. Except the already mentioned attributes it contains a unique identifier of type int which auto increments after every insertion and a unique identifier based on the service provider type.

2.3 Implementation of Exchange Calendar Service

Aim of this section would be to describe the implementation of the Exchange calendar service, however as Exchange service is not the main part of this project, more detail regarding its implementation will be provided in the upcoming chapters focused solely on Google calendar service which is nevertheless largely inspired by the Exchange one. In addition the Exchange service wasn't properly tested and several of its features still remain unused.

2.3.1 Front End Setup

As was explain in the earlier chapters, in order to take advantage on the notification service administrators of the system must first provide essential configuration details in the relevant Youmanage application front end.

This part of the presentation layer is represented by an OutlookIntegration.aspx and ExchangeServiceSetup.aspx pages containing HTML tags and ASP elements. The separation between presentation and the underlying logic is done using a "code-behind" Visual Basic class files. Each class is compiled into an object allowing an access to all its content. The .aspx page has to inherit from the relevant base code-behind class in order to use it [2].

As every ASP element in the .aspx page is accessible by the code-behind class, user interface is constructed using ASP placeholder elements. Such elements can be comparable to HTML tag <div> allowing grouping multiple ASP and/or HTML elements together.

Code-behind class implementation then keeps track of user interactions on the .aspx page and displays relevant placeholders. Such technique makes sure that user is presented only information relevant to the stage he is at and doesn't require useless redirections.

After the administrators of the system reach the initial OutlookIntegration page, a SQL query is performed during the page load in order to determine whether a service was already configured or not. The SQL query itself exploits the advantage of the architecture chosen by Youmanage. Since each client has their own instance of Youmanage application and an own database, only a single record of a calendar service can be present in the t_serviceProvider table. As a result the SQL query selects the first occurrence of a service with a calendar capability from such table. Consequently, it either shows a relevant placeholder with service settings – permission scope, reconfiguration – if service was configured, or redirects to ExchangeServiceSetup.aspx page.

The redirected page contains few stages administrators have to go through in order to configure the service. These stages alongside with their transitions and the page life cycle can be seen on Figure 6.

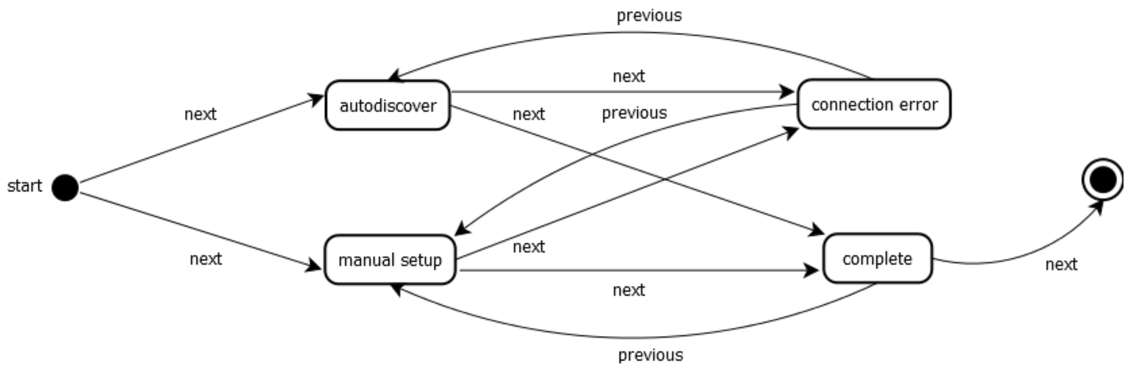


Figure 6. State diagram of ExchangeServiceSetup.aspx page

Initial state of the page is start, at which point users are presented with a placeholder containing two radio buttons. Transition to the next state is then determined by a desired radio button option followed by a button click.

After filling necessary information about the Exchange server for a chosen option and consequent button click, a new instance of ExchangeServiceProvider class is created, user data input passed and relevant service methods are called.

Results of the methods then determine the next state and placeholders presented. In case of successful connection, provided data are stored in the database and placeholder notifying of success is displayed, in second case a connection error placeholder with a received error message is displayed.

At any given state of the page users are presented with a cancel button which as well as the next button in the complete state finishes the life cycle of the page by redirecting it back to the OutlookIntegration.aspx page.

2.3.2 Business Layer Notification

Since the purpose of the service is to convert and insert a single appointment into a single employee calendar the business layer provides a logic which restricts the circumstances under which an instance of a service is created and used. All logic regarding notifications resides in a Visual Basic class file BLNotification. Methods stored inside of this class are then accessible by any code-behind class file which inherits from a page holding an instance of a business layer accessor. Such technique ensures uniform and effective sharing of the application resources.

All functionality offered by BLNotification class regarding the calendar service can be sorted into two distinct categories based on the implementation of methods:

2.3.2.1 Methods interacting with data layer

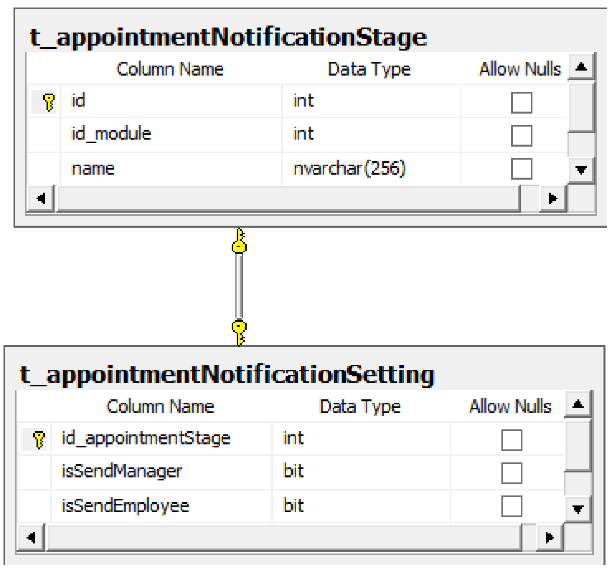
These methods contain on their lowest level various SQL queries manipulating tables described in Figure 5 and are essential while configuring and using a calendar service. In order as implemented in the source file such methods are:

- **IsServiceConfigured**
  - This method checks if a particular type of service provider has been configured and returns either true or false
- **GetServiceProviderByCapability**
  - As the name suggests such method selects a service provider from t_serviceProvider and returns it as an instance of a class Service Provider or an empty instance
- **GetServiceProviderById**
  - A similar method selecting by a different criteria, however having an identical output
- **Insert**
  - Inserts a new service provider into the database based on an instance of a Service Provider object supplied
- **Update**
  - Updates a service provider stored in the database based on an instance of a Service Provider object supplied
- **Delete**

    - Deletes a service provider from the database based on an instance of a Service Provider object supplied

- **GetServiceProviderTypeById**
    - Returns either an instance of existing service provider by id provided or an empty instance
- **GetServiceProviderTypes**
    - Selects and returns a list of all Service Provider Types objects in a database

In addition there are two more methods with a focus towards a different parts of the data layer. As was mentioned in Section 2.2.1, administrators have the possibility to configure a permission scope regarding which employees can receive calendar notifications after the service is configured. Before including these methods perhaps it's worth to explain their affiliation to the system from the view of a data layer (Figure 7).


t_appointmentNotificationStage

Column Name	Data Type	Allow Nulls
id	int	☐
id_module	int	☐
name	nvarchar(256)	☐

t_appointmentNotificationSetting

Column Name	Data Type	Allow Nulls
id_appointmentStage	int	☐
isSendManager	bit	☐
isSendEmployee	bit	☐

Figure 7. Permission Scope ER Diagram

The top table consists of an appointment type identifier, identifier of a corresponding module and a name of the appointment type, where the bottom table contains appointment type identifiers and pairs of Boolean values determining who receives notification of which appointment. Records in both of these tables correspond to the values presented in the table at Figure 1. As a result the last methods accessing data layer are:

- **IsAppointmentEnabledForEmployee**

  - By accepting an appointment identifier as an argument this method returns true or false, based on the value stored in the table t_appointmentNotificationSetting and column isSendEmployee
- **IsAppointmentEnabledForManager**
  - Similarly, this method returns the value in the column isSendManager

2.3.2.2 Methods interacting with types layer

Even though these methods don't communicate with a data layer, due to the fact that calendar service is called after an appointment is approved and hence after stored in the data layer its worth to examine the data layer regarding the system appointments in order to see its relation to employee company emails used by the calendar service (Figure 8).

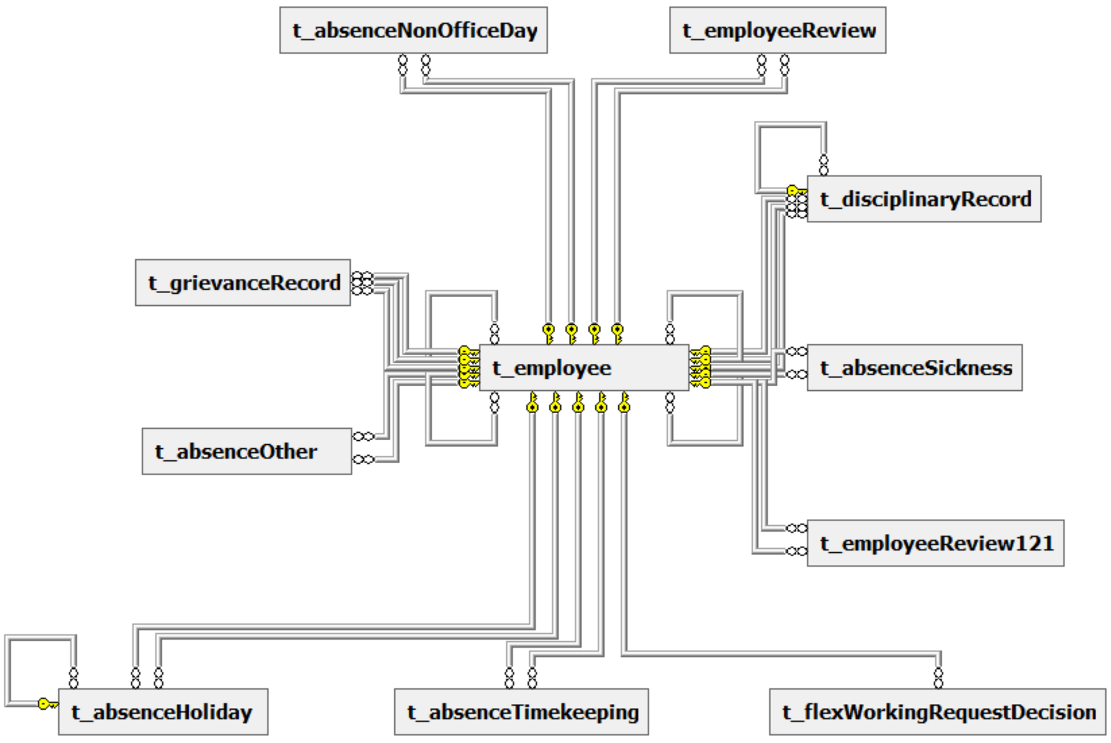


Figure 8. Simplified ER Diagram of Appointments

As we can deduct, every system appointment has a foreign key in a form an employee identifier, which determines the relation among these entities. Moreover as one employee can be managed by a manager which is also an employee and one manager can manage multiple employees, entity employee has a unary relationship. This design helps to determine the relevant manager email in a relation to an employee managed.

Unfortunately, as every appointment type contains unique and distinct data it's not possible to treat them uniformly. As a consequence each appointment type has to implement its own method interacting with a calendar service. Nevertheless each method follows a similar pattern and differs almost exclusively in processing of the system appointment and therefore in this case will be described only once in a pseudo uniform way.

- **CreateAppointment**
    - This method accepts an instance of an object corresponding to a system appointment table record and provides no output value in return. At first it instantiate an IAppointment variable by calling GetAppointmentService method and storing the relevant instance of a calendar service. If the variable is not null hence service was found it then creates an employee variable and stores the instance of an employee based on an employee identifier from the system appointment. Consequently it check if employee is allowed to receive a calendar notification by calling a method IsAppointmentEnabledForEmployee with a system appointment as a parameter. If appointment is enabled and if employee email is not empty it then creates a new variable of IAppointment type. Value of the IAppointment type is then determined by a result of CreateAppointment service method which takes the employee email as a parameter and returns an instance of IAppointment object. Finally, it populates received service appointment with relevant data stored in the system appointment instance and passes it as a parameter to the service method SaveAppointment which will try to insert it in a relevant employee calendar based on the email provided earlier.

      In the end it checks if manager is allowed to receive a calendar notification calling the relevant IsAppointmentEnabledForManager and if true it repeats the rest of the described sequence providing a manager email as a parameter.

      In addition for absence non-office day appointments, such method offers recurrent events to be inserted in user calendars by simply calling service methods create and save appointment in a loop.

Perhaps the most important is the previously mentioned GetAppointmentService method which was designed to return the actual instance of the calendar service provider class by using other methods interacting with Data Layer. In order to accommodate a new calendar service provider only this method needs to be extended.

2.3.3 Exchange Calendar Service Core

The logic of the Exchange notification service resides in the Utility layer of the system and is implemented by ExchangeServiceProvider and ExchangeAppointment classes both written in C# (see sections 2.1.4 and 2.2.2).

2.3.3.1 ExchangeAppointment

This class holds a property of Appointment type representing an Exchange event and implementing IAppointment interface methods. Such methods are simply properties (setters and getters) and their purpose is to populate the attributes of an event from any system appointment. In addition this class offers a method converting an IAppointment instance provided as a parameter into an Appointment object.

2.3.3.2 ExchangeServiceProvider

Implementation of this class handles the entire interaction between Youmanage and Microsoft Exchange servers and its detailed content description is available in the following bullet list.

- **Enumerator ExchangeServerVersion**
    - This enumerator list declares a set of distinct constants in order to determine which Exchange Server version the service is handling.
- **Constants**
    - Inside of the ExchangeServiceProvider class are defined various error codes and corresponding error messages Exchange service can generate. These constants are implemented as integers and strings and serve such purpose.
- **ExchangeService _service**
    - This class variable is an actual representation of the web service provided by Microsoft Exchange API, and therefore credentials from the business layer are passed into _service variable.
- **Boolean _isConnected**
    - While connecting to Exchange servers this class variable keeps track of the current status of the connection and is accessed by a corresponding property.
- **String _lastErrorMessage & int _lastErrorCode**

    - If at any point Exchange service is not able to successfully complete a request, corresponding values of these variables are set to corresponding constants. These variables can be accessed by their properties.

- **Setters and getters manipulating _service class variable**
    - These methods change values of the ExchangeService class variable instead of ExchangeServiceProvider one and allows setting the connection credentials, setting and getting the URL of the Exchange server, setting and getting the email of a user to impersonate, and getting the version of an Exchange Server. Their actual implementation is in the form of class properties.
- **Constructors**
    - ExchangeServiceProvider class defines three custom constructor methods accepting as parameters: URL of the Exchange server, server version and both URL and server version.
- **Initialize method**
    - This method is called within constructors and initializes the ExchangeServiceProvider by setting up a certificate validator and creating a relevant _service instance based on Exchange server version, if no server version is provided Exchange Server 2010 SP1 is an implicit value.
- **CreateAppointment method**
    - At first this method takes one argument representing a user's email address and checks if the service is connected based on the value stored in _isConnected variable. If service is not connected, error code and message are set to a not connected values and no value (null) is returned. In case a service is connected it consequently checks if the provided email is valid followed by a similar processing. In other case a valid email is stored by a setter method responsible for impersonation. Finally, new non-populated instance of ExchangeAppointment is created and returned as an IAppointment type.
- **SaveAppointment method**
    - Initial part of this method checks again if the service is connected and if not sets error codes and error message and returns _lastErrorCode. In other case it converts an instance of IAppointment received as a method

argument into Appointment object and inserts it in to user calendar. If an exception is caught during insertion error codes and messages are set and _lastErrorCode returned. Successful insertion is acknowledged by returning a zero value.

- **AutoDiscover method**
    - This overloaded method either takes one argument in a form of an email address or two in form of an email address and network credentials and attempts to discover setting of Exchange server. Successful discovery is acknowledged by returning a zero value.
- **IsImpersonationConfigured method**
    - Based on the email provided as a parameter this method tries to insert a test appointment into a user's calendar and consequently delete it. Successful operation is acknowledged by returning a zero value, unsuccessful by returning a corresponding error code.
- **Disconnect method**
    - Sets the variable _isConnected to false and returns a zero value to acknowledge successful operation.
- **Connect method**
    - Overloaded method accepting URL of the Exchange server, network credentials or both URL and credentials. This method helps determining the status of a service. Returned value is either error code or a returned value received from calling ConnectInternal method.
- **ConnectInternal**
    - This method determines status of the service connection by attempting to access Inbox folder. Successful connection is acknowledged by returning zero, unsuccessful by returning an error code.

ExchangeServiceProvider contains few additional methods, either offering an experimental functionality, or unused within the system and therefore irrelevant to this paper.

2.4 Time representation of system appointments and Exchange events

This last subsection of this chapter aims to describe a very ambiguous behaviour of Youmanage system regarding the time representation of appointments stored and displayed within the system and their corresponding events in Exchange calendars.

2.4.1 Absences

Youmanage application offers an Absence & Holiday planner feature which displays all employee absences (e.g. sickness, holiday, non-office day) in a simple calendar form, however this simplified calendar works only with dates and morning (AM) or afternoon (PM) absences. All absences stored in the database however possess DateTime attributes including hours and minutes. As Exchange calendar also use DateTime representation for its events its essential to understand the time stored in the database and its effects on Exchange events.

Any morning absence displayed in the absence planner spans in fact from 00:01:00 until 13:01:00 and consequently any afternoon absence spans from 13:01:00 until 00:01:00. Since such time representation doesn't reflect any universal working hours, the notification business layer method CreateAppointment tries and converts the time into a more sensible one. Nevertheless its implementation is ambiguous and unclear perhaps due to the fact that Exchange calendar service hasn't been sufficiently tested.

In case of holiday absence for employee if the appointment spans for more than one day, the end time of the calendar event has six hours less than the end time of the absence in the data layer. Even more confusing is the fact that the same absence for manager translates the time in a different way.

In case of a morning absence, Exchange event starts from 09:01:00 and in case of afternoon absence starts from 01:01:00 the next day. Also in case of an appointment spanning over more than one day the end time of an event has one minute less compared to the database stored record. One day event end time has on the other hand thirty minutes more.

Moreover methods creating appointments from all other absences don't implement any time conversion and as a result time of the calendar events inserted reflects the very same time in the database for both employee and manager.

2.4.2 Employee Reviews and Decision Meeting

The value of start date and time of these appointments is actually provided in the front end by managers of the Youmanage application with five minutes accuracy and therefore there is no need to convert its time. However as such appointments omit the end date attribute, the Exchange calendar event end time is implicitly set one hour away from the start of the appointment.

2.4.3 Decision Trials

These appointments contain only a single date and time attribute which is implicitly set to midnight (00:00:00) and therefore the associated CreateAppointment method adjusts the start time of an Exchange event to 9AM and end time to 10AM.

2.4.4 Grievance Records and Disciplinary Records

Records stored in such tables contain only a start date and time attribute and as a result business layer responsible for creating appointments set such Exchange event as an all-day event implicitly.

3 Incremental and iterative software development

The software crisis in the late sixtieths of the last century has shown that it’s crucial to adopt some sort of methodical approaches while working on larger scale software projects. As the volume of a project increases so does its complexity and not adopting recommended approaches can lead into poor products delivered, leave developers with frustration and customers unsatisfied.

The most traditional development approach and a base for the modern approaches established is a “Waterfall Model” which divides the project and cascades down a series of five stages the development should follow. These stages must be completed from the top to bottom and are as follows: Requirements and Analysis, Design, Implementation, Testing, Maintenance. Only once a stage is completed next stage can begin [1].

Perhaps the biggest disadvantage of this approach is that issues which emerge in one stage affect all previous stages and therefore the entire development has to move back to the very beginning and accommodate changes. In conclusion this approach would be difficult to use in projects where the nature of requirements tends to change rapidly.

Among few other alternative modern approaches are incremental and iterative developments which are in addition often engaged together.

3.1 Incremental development

This approach divides the project into smaller pieces and develops each piece separately. The word increment fundamentally means “add onto” and therefore once a piece of the project is done, another piece then wraps the previous one and extends its functionality [3].

3.2 Iterative development

Focus of an iterative development is more towards the validity of the piece of software and its complying with user requirements. Throughout the stages of the development we examine the functionality of a piece of software and determine whether it works the correct way. The word iterative fundamentally means “re-do” and therefore as development goes from more abstract stages into more concrete ones a certain rework is expected [3].

Engaging both developments together helps to realize a step in a wrong direction followed by a fast consequent adjustment (Figure 9).

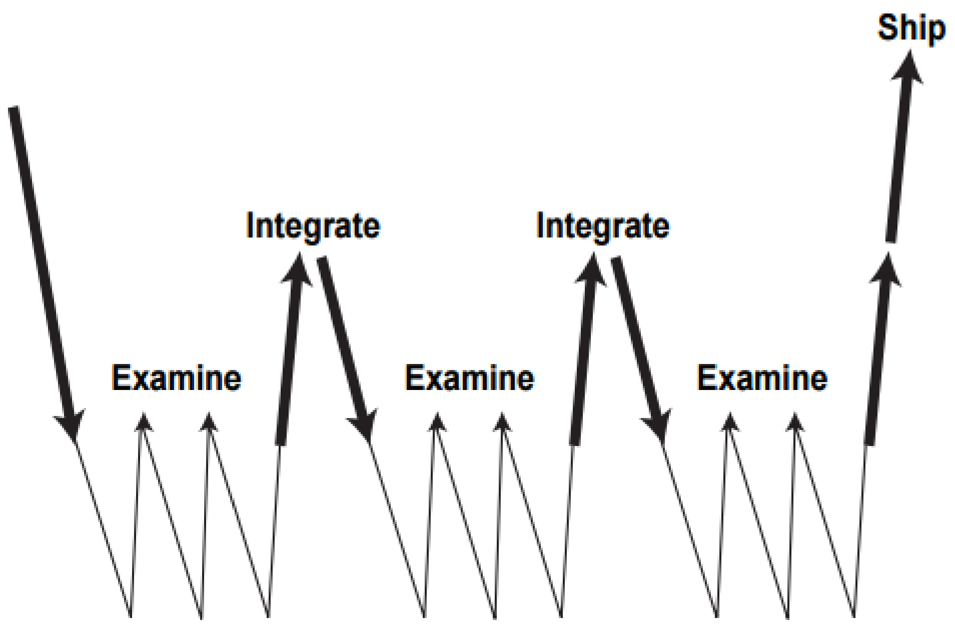


Figure 9. Putting Iterative and Incremental Development Together

As the initial requirements received from Youmanage were vague and possible changes and modifications were expected, both – incremental and iterative – developments were used as being the most reasonable ones.

4 Requirements Analysis

Constructing a set of requirements involves interactions with stakeholders and being the initial phase of the development cycle usually sets the nature of the whole project. Such interactions are commonly referred as eliciting requirements and all requirements demanded must be realistic, measurable and testable and must provide sufficient detail regarding the project. It is quite common that stakeholders lack the ability to express what they want and therefore is necessary to guide them.

Proposed requirements then need to be evaluated in order to determine their clarity, completeness and possible conflicts. In existing projects such as Youmanage application requirements shouldn't affect the overall integrity of the system.

Output of requirement analysis is usually recorded as a document in a natural-language or as a set of use cases for both developers and stakeholders to understand. In addition this phase may be base for a legally binding contract.

During the initial interview with Youmanage it was discussed that clients of their application wish to extend the offered Exchange notification service by supporting other various calendars. Google and Apple calendars were originally brought to the table. Basic requirements demanded the exact same functionality as implemented service and additional requirements were exploring the options of implementing a two way communication between user calendars and Youmanage application, or implementing additional features of the calendar service. Such features should allow a calendar service to delete or update an inserted event if a state of a corresponding system appointment would change accordingly.

Following negotiations led into an agreement that all calendar services should provide the same functionality in order to offer consistency, moreover implementing additional requirements for all services including Exchange service would violate the condition of testable requirements. As a result additional requirements vanished, however a possibility of two way communication remained as an object of a research.

Unfortunately clients of Youmanage were not involved during the initial interviews and as a consequence the basic requirements were misunderstood and wrongfully formulated. This fact was realised during the early iterations of the development and therefore didn't cause a larger rework. It was discovered that clients demanded a support of another email business tools similar to Microsoft Exchange and that was in the contradiction with Gmail and Apple as personal emails.

Further research found Google Apps for Work business tool as a possible candidate for new requirements which consequent communication between Youmanage and its clients confirmed as originally demanded. Nevertheless Apple doesn't provide any business solutions at the time of writing this paper.

In addition extra requirements were formulated throughout the development to suit Youmanage and its client's needs. Among those was a synchronisation feature offering clients of Youmanage to insert appointments stored in the system before a calendar service was configured.

Finally, the biggest emphasis was put on the integrity of the system restricting the development into designing and integrating solution without affecting Exchange calendar service and by using abstraction already provided in the system.

4.1 Use case modelling

This user-oriented software engineering technique is used to translate requirements into a more in-depth graphical representation where the proposed functionality is represented by various actors and their interactions with the system. Primary actors represent actual users and their role within the system whereas secondary actors can represent an external entity such a web service.

While analysing requirements proposed by Youmanage, three actors were determined and following use cases were designed in order to illustrate possible interactions with the system (Figure 10).

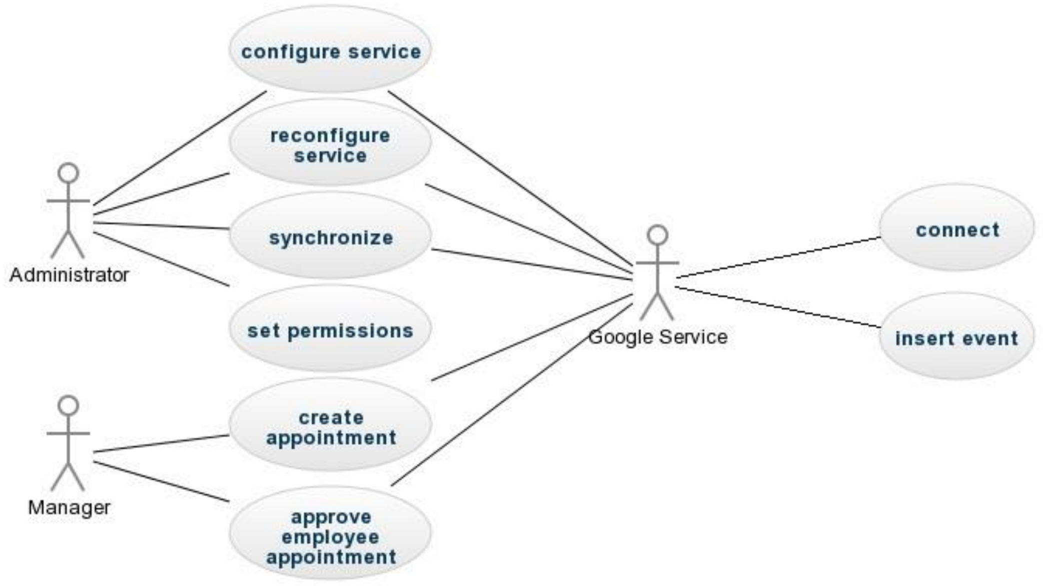


Figure 10. Requirements Use Case

Reflecting the functionality of Exchange calendar service administrators should be able to configure and reconfigure the service using its connection status and in order to offer such functionality a large piece of administrator front end would have to be re-implemented. It has been agreed that biggest emphasis will be put on the user friendly interface in order to ease the configuration process.

Use case synchronize was an additional feature requested by Youmanage offering its clients a possibility to inserts appointments stored in the system before the calendar service has been configured. Such feature should be a onetime opportunity restricted to administrators in order to avoid possible conflicts in user calendars and should allow to select a specific date to synchronize from and use the permissions provided.

Use case set permissions is another functionality already implemented by Exchange calendar service and in case of new calendar service requires its integration in the new administrator front end. This functionality should have no direct relation to the calendar service.

Manager actor contains two functionalities regarding manipulating system appointments where is expected that in both of these cases the calendar service should trigger its inserting feature. It was stated that these functionalities are already implemented in a uniform way and should stay unchanged by Google calendar service taking advantage over them. However for sickness absences this feature was removed and therefore it was agreed that this part will be re-implemented. Moreover implementation of some of these methods at the business layer completely omitted a permission scope set up by administrators and therefore was requested by Youmanage to adjust such parts.

Actor Google service should than be an interface between Youmanage application and Google Apps for Work servers offering the same functionality described in 2.3.3.

In addition as not all employees in the client systems including administrators might have a company email address, all features should be able to process such fact.

4.2 Prerequisites

Before the project started various prerequisites were put forward from both of the sides to ensure mutual understanding and smooth developing process.

Youmanage limited the options of possible technologies to reflect their system and hence Google calendar service had to be implemented either using C# or VB .NET programming languages. Furthermore there were no additional restrictions regarding data layer allowing it to be modified if needed.

For the sake of development and especially testing the Google calendar service two working Google Apps for Work domains had to be provided. As Google Apps for Work is a commercial tool such prerequisite would require additional resource from Youmanage, however this tool is available for a free trial period of thirty days and therefore only requirements regarding registering the tool had to be met. Such requirements included a proof of a registered domain in order to set up a company email. Youmanage is in a possession of several registered domains regarding their business and as a consequence all requirements were met and accommodated.

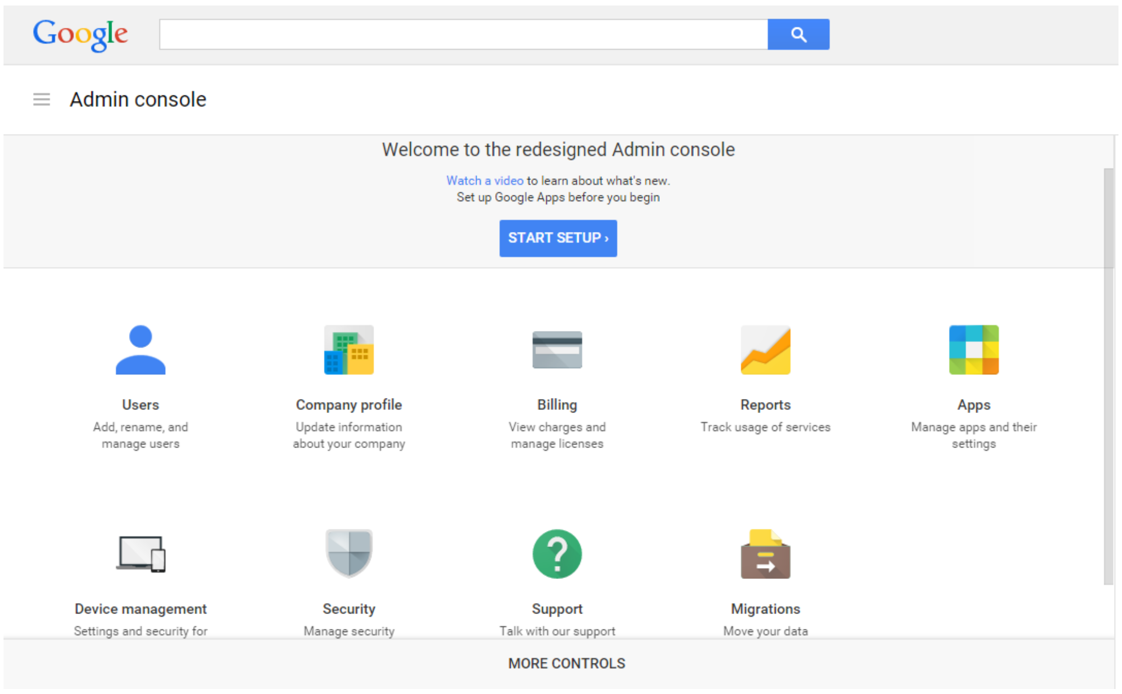
Lastly, to offer a more user friendly experience an initial fee of five dollars had to be paid in order to publish a simple application at Chrome Web Store. This fee presents a onetime fee associated with first publishing from a Google Chrome Developer Dashboard account.

5 Design

Aim of this chapter is to provide an overall design of the final solution, however in order to achieve that an in-depth analysis of Google Apps for Work and possible solutions had to be concluded beforehand.

5.1 Google Apps for Work

This commercial business tool offers companies a way to manage their emails, calendars, Google drives and other services offered by Google. Accounts created and email addresses associated are tied onto a domain owned and provided by the company. Google Apps for Work company emails are treated as any other google accounts and therefore can be accessed from the same URL (e.g. http://www.gmail.com). Google Apps for Work domain includes an administrator interface which can be accessed by a user with administrator privileges. Such interface includes various business tools where the most related offer managing users, changing security settings of the Google Apps domain or installing Google Marketplace applications (Figure 11).


Welcome to the redesigned Admin console

Watch a video to learn about what's new.
Set up Google Apps before you begin

Figure 11. Google Apps for Work Admin Console

Managing users offers adding new users into the domain and creating their email addresses, deleting or suspending users, transferring them into another organization or setting their privileges and grouping them.

Within the security settings the most notable option allows administrators to manually control access of third party applications using OAuth 2.0 protocol.

Marketplace applications then allow users of Google Apps for Work domain to install extensions and therefore setting third party applications access automatically.

5.2 OAuth 2.0 and Google Resources

An OAuth 2.0 is a token-based authorization protocol and its main purpose is to provide a uniform and secure access to resources from third-party applications authorized by owners of the resources. Authorization is performed with user consent however without sharing user credentials. OAuth 2.0 is a standard for accessing Google APIs [4].

For developing applications accessing Google resource servers, developers must use Google Developers Console in order to obtain their application specific OAuth 2.0 credentials. Such credentials are then presented to Google Authorization server alongside with a scope defining API(s) the application wants to access. In a response message from Google Authorization server an access token with a limited lifetime is send and once extracted can be used to access the relevant API. The lifetime of an access token is usually around one hour and therefore if an application need to access the resources beyond such lifetime it can obtain a refresh token which allows the application to ask for another access token once it expires [4].

In addition Google OAuth 2.0 documentation provides various scenarios with a different use of this protocol.

5.2.1 Web server applications

In this scenario before the application can access resources of a certain user a redirection to a Google authorization URL with a parameter specifying the type of access requested is performed. At such URL Google handles the user authentication, session selection and user consent. The result is a redirection back from OAuth 2.0 server to a client application where the redirection URL has to be specified in Google Developers Console in order to receive server response message. Such message contains an authorization code which is consequently exchanged for an access token and refresh token if necessary (Figure 12) [4]. Such authorization is sometimes referred to as a three-legged authorization.

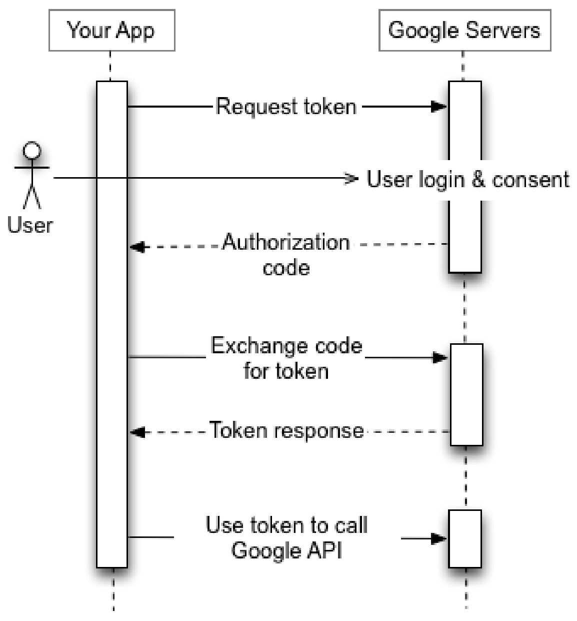


Figure 12. OAuth Web Server Application Flow

Considering this scenario for a Google calendar service the system would need to extend the data layer in order to store refresh tokens of all the users. Moreover each user would have to give an explicit consent before hand. Finally, the biggest issue would present the redirection with the authorization code back to a client application. The credentials required by Google Authorization server are application specific, however each client of Youmanage runs their own instance on a specific URL. How would then Google realize which instance of Youmanage originated the request and consequently where to redirect with the authorization code? This would mean that each instance of Youmanage would need to have their own project or their own Google Developers Console account in order to have a unique set of credentials determining the relevant instance of Youmanage application.

5.2.2 Service accounts

Service accounts are used either while accessing own resources or when the access is granted by a higher authority and therefore user consent is not necessary. Such authority may be an administrator of a Google Apps for work domain. In this scenario the third party application accesses the Google API on behalf of the service account. Service account credentials are obtained from a Google Developers console and contain a unique email address, client id and a cryptographic key pair. The application use one part of the key and Google Authorization server the other. As such third party application uses the service account email address and a private key to create a signed JWT and constructs an access token request which similar to the web server applications scenario contains scope defining what resources the service account wants to access. This request is send to a Google Authorization server which decrypts the JWT and returns an access token which is used by the application to access Google API (Figure 13). When the token expires, application has to repeat the process [4]. This type of authorization is often referred to as two-legged.

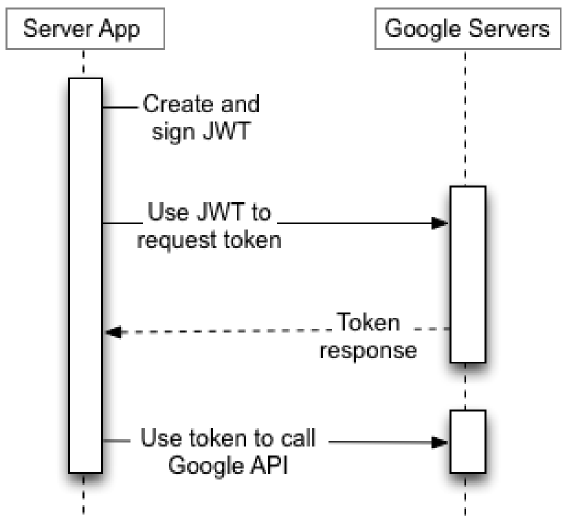


Figure 13. OAuth Service Account Flow

In a relation to a Google calendar service administrators of a Google Apps for Work can authorize a service account for accessing available APIs without user consent. This is referred as "delegating domain-wide authority". However service account itself is not a member of a Google Apps for Work and therefore in order to access user data it must act as a user of the domain with sufficient privileges [4]. This corresponds to the impersonation behaviour of a Microsoft Exchange calendar service.

In addition administrators of the Google Apps for Work domain can grant access to the service account either manually by copying appropriate information or in case of installing a Google Apps Marketplace application such permissions are granted automatically [5].

Compared to the previous scenario, all instances of Youmanage application can use the same service account as relevant Google Apps for Work domain is determined by providing an email and impersonating concrete user of the relevant domain using it.

In conclusion, as all Google services are RESTful, corresponding OAuth authorization flow is always carried via HTTP requests and responses and therefore can be constructed either manually or by using client library provided by Google. However as constructing manual requests is error-prone, it was decided to use C# client library in order to secure and automate such tasks.

5.3 Google Calendar API

The actual access to user calendars of a given Google Apps for Work domain is done by a Google Calendar API. This API allows applications to perform various operations such as search for calendar events or create, edit, delete both calendars and events within a google account provided [6]. For this project, Google Calendar .NET API was used in order to implement the calendar functionality.

Google Calendar is built on several basic concepts where a short description of those relevant to this paper is summarized below.

5.3.1 Event and IAppointment

A calendar event contains information such as title, start time, end time or a list of attendees [6]. From a design point of view it is important to understand how event attributes translates into the corresponding IAppointment interface (see 2.2.2) and its properties.

5.3.1.1 Attendees

This property represents a list of users joining an event and corresponds to an event attribute with the same name consisting of several attributes itself. A valid email address of a user is required while adding an attendee. Optional attendee's attributes can consequently specify e.g. number of additional guests, response comments, attendee names, Google+ profile IDs etc. [7].

5.3.1.2 Body

Purpose of this IAppointment property is to describe an event and hence corresponds to an event attribute description. This attribute is optional and is represented by a single string [7].

5.3.1.3 Start and End

These IAppointment properties signify start/end date and time of an event expressed as a non-null able DateTime type. The corresponding event attributes are then consequently represented by Start and End attributes and their class variables dateTime. However such variables are of type null able DateTime and hence no value (null) can be present [7].

5.3.1.4 Id

Such property serves as a unique identifier of an event defined as string. Event contains an attributed of the same name and type. Nevertheless to avoid possible conflicts while these resources are globally distributed a certain restrictions were put upon this attribute by Google. A valid identifier characters must use base32hex encoding, hence using letters a-v and numbers 0-9, the length of the identifier must be at least five characters and not exceed 1024 and of course each identifier must be unique within a calendar. Finally, due to these restrictions it is advised to use an established UUID algorithm to generate such identifier [7].

5.3.1.5 IsAllDayEvent

Unlike the Exchange appointment where all-day event is determined by a Boolean value, Google calendar Event uses attributes Start and End and their class variables Date to reflect such state. Such variables are of type Date and due to its ability to span over multiple days, Google all-day event can represent more than just a single day [7].

5.3.1.6 IsPrivate

A private Google event is determined by its attribute Visibility. Such attribute accepts a string value and can reflect one of four states: default, public, private, confidential. Both private and confidential values signify a private event, which explained by Google is for compatibility reasons [7].

5.3.1.7 Location

In order to set where an appointment takes place a simple string value is assigned to a Location attribute of Google event [7].

5.3.1.8 Organizer

This property represents the email of a person hosting an event. Similarly to the attendee attribute, Google event attribute organizer contains class variables representing email, name and Google+ profile ID [7].

5.3.1.9 Subject

In case of an Exchange appointment a name of the event is represented by a string value of attribute Subject, however in Google Event class such fact is represented by an attribute Summary also accepting a string value [7].

5.3.2 Calendar

A calendar contains a set of Events. Every user with an associated Google account has at least a primary calendar and can create multiple secondary ones. It is possible to delete secondary calendars, however it is not possible to delete a primary one. Nevertheless, all events can be deleted from any calendar.

5.4 Logical Part

Following the Exchange calendar service (See 2.2.2) an extended UML diagram was designed in order to accommodate the Google calendar service functionality (Figure 14).

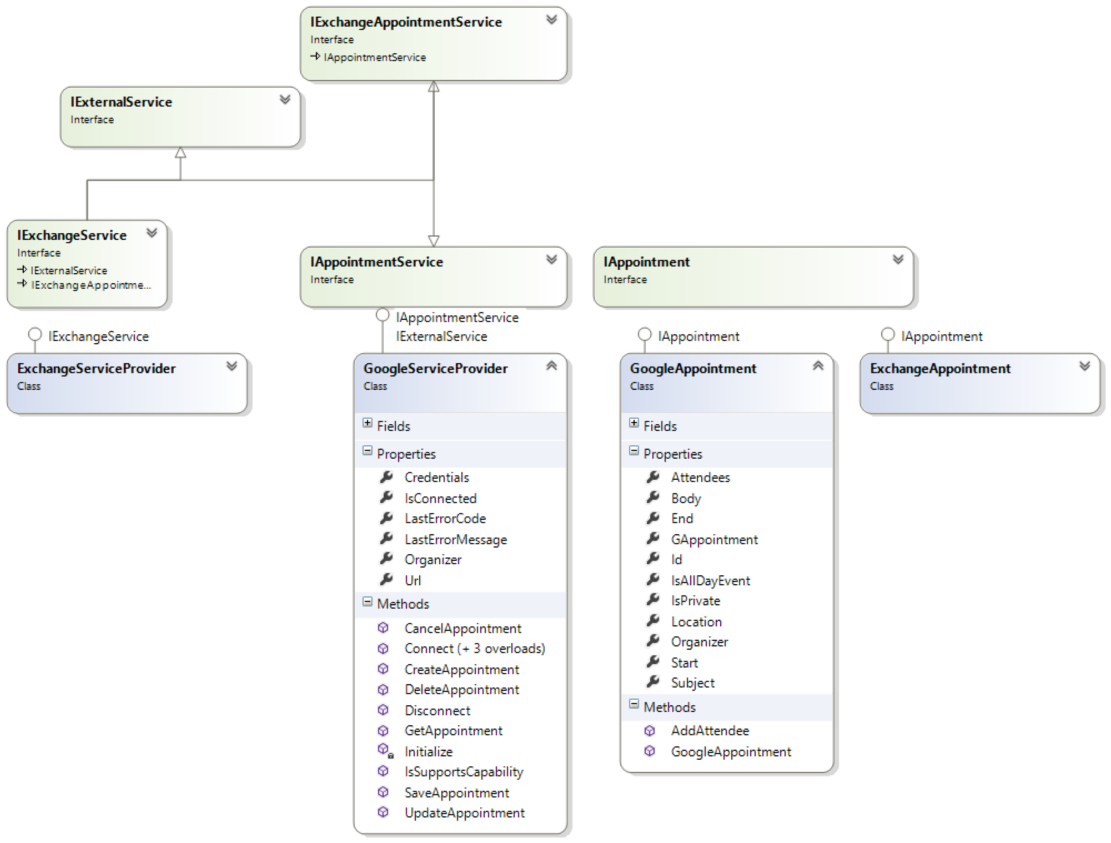


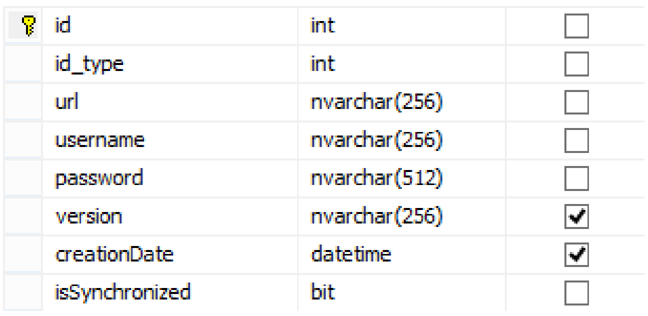
Figure 14. Extended Calendar Service UML

Earlier parts of this paper have already concluded that in order to develop a new calendar service, two interfaces must be implemented. First, which contains the calendar service itself and a set of methods for service configuration and calendar events manipulation, and second which represents the actual calendar event being inserted. In addition it has been realized that earlier calendar service offers an extended, however unused functionality, and therefore even though not all of the functionality has to be present in the new service, the obligation sprung from the interface implementation still remains.

5.5 Data Part

Even though the Exchange and Google credentials essential for a service configuration are different, no extra modifications of the Data Layer were required in order to accommodate the new calendar service.

However, in order to offer an extended functionality in a form of a synchronization tool, a minor Data Layer adjustments had to be performed (Figure 15).



🔑	id	int	☐
	id_type	int	☐
	url	nvarchar(256)	☐
	username	nvarchar(256)	☐
	password	nvarchar(512)	☐
	version	nvarchar(256)	☑
	creationDate	datetime	☑
	isSynchronized	bit	☐

Figure 15. Table t_serviceProvider

Two new attributes creationDate and isSynchronized were created in order to accommodate the synchronization functionality. The first attribute was designed in order to store a timestamp when the service configuration occurred, allowing to determine until which point appointments were not yet inserted. Whereas the latter one is a Boolean value expressing if administrators already used the new functionality and therefore should be able to access such functionality at the administrators front end or not. All adjustments made at the Data Layer had to be reflected in the corresponding Service Provider class in Types and also the Data Mapper layer.

5.6 Chrome Web Store

Creating a Google Apps Marketplace application (See 5.2.2) installable from to the administrator frond end of the Youmanage application would provide much better user experience than a basic manual configuration which is error-prone.

Since the first version of Google Apps Marketplace is deprecated and the new version wasn´t released yet, Google Apps applications are currently published at Chrome Web Store [8].

In order to publish an application at Chrome Web Store, a Chrome developer account (which is different than a Google developer account) has to be registered and a fee of 5 dollars for first-time publishers has to be paid [9].

5.6.1 Google Developer Console

Marketplace applications use a Google Apps Marketplace SDK API which is a toolkit for integrating web applications with Google Apps for Work. Such API must be enabled and configured in the relevant project in a Google Developers Console. Besides a generated project number determining individual application, configuration of a Marketplace API includes a name of the application, a short description, application icons, support URLs, and a scope the application wants to access.

In addition Marketplace applications interacting with other than Gmail or Drive Google services have to provide a universal navigation extension URL which signifies where will be administrators redirected after the installation process.

The implicit scope of a Marketplace application contains a read access to user email and profile and in case of this paper should be extended by a full access to user calendars.

5.6.2 Chrome Developer Dashboard

Once a Marketplace application is configured (See 5.6.1) it can be published to a Chrome Store from a Chrome Developer Dashboard account. Application is submitted as a zip file containing a JSON manifest file with a compulsory *manifest.json* name and a set of icons specified in such file. Manifest file contains a collection of name/value pairs representing the settings of the application listing at Google Chrome Store. In order to publish an application as a Marketplace application, manifest file must include a DOMAIN_INSTALLABLE value [9]. After the application is submitted, additional settings are required in order to publish it.

Such settings consist of detailed description, screenshots and propagation images, website URL offering this application (a proof of ownership is required), category where the application will be listed at, geographic availability and most notable a visibility of the application. Applications offered at Chrome Store can be either public – visible to everybody, unlisted – accessible only by people with an application URL or private – only trusted users from developer dashboard can see it.

Finally, after all the required steps are completed, the application can be published and review by Google.

6 Implementation

From the design period it has been concluded that the implementation of the new calendar service will not include interacting only with Youmanage application, but also with various tools offered by Google (See 5.2, 5.3) As a consequence a prior configuration of an integrated development environment used was required.

Such configuration included downloading a Google Calendar API client library. This library includes methods and classes for manipulating calendars as well as for the authorization process and is distributed in a form of a NuGet package.

NuGet is a Microsoft development platform package manager integrated in a Visual Studio 2013 with the ability to produce and consume packages [10].

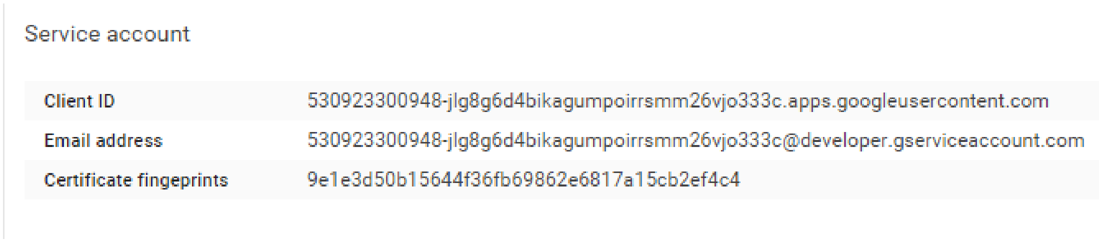
After downloading, such package is added into the project; however in order to access its content an API reference had to be set in the Utility layer.

6.1 Setting up a Google Developers Console

Google Developers Console provides a way to configure authorization process for accessing Google resource servers from a third-party applications. Anybody with a google email account can access such console and create or configure projects they own.

As a consequence it was necessary to create a new project and set the requested API Youmanage application needed to access to Calendar API (See 5.3). In addition accessing Google resources is a subject of free quotas which related to the Calendar API can't exceed 1.000.000 requests per day and five requests per second per user. If the expected requests would exceed such numbers it is however possible to apply for a higher quotas for some additional fees. It was concluded that free quotas should be sufficient to accommodate the need of Youmanage at the time of the development.

Once requested API(s) are set, new credentials must be created in order to access selected resources from the Youmanage application and therefore following design period (See 5.2.2) new service account credentials were created (Figure 16).



Service account

Client ID	530923300948-jlg8g6d4bikagumpoirrsmm26vjo333c.apps.googleusercontent.com
Email address	530923300948-jlg8g6d4bikagumpoirrsmm26vjo333c@developer.gserviceaccount.com
Certificate fingerprints	9e1e3d50b15644f36fb69862e6817a15cb2ef4c4

Figure 16. Service Account Credentials

Service account credentials contain a unique generated identifier of the project, an email address acting as a service account and a generated set of private/public keys in a form of a standard P12 file and JSON. The P12 key needs to be downloaded to a secure location accessible by the application in order to make authorized API calls. Google stores only copies of the public keys and thus downloaded keys serve as the only copy [5].

The generated name of the P12 key is the key's thumbprint and in order to use it within an application a password "notasecret" has to be provided. Even though the password is identical for all Google generated keys, each key is cryptographically unique. It is also possible to generate multiple keys for a single service account within the Google Developers Console in order to update application credentials without downtime. Finally, it is not possible to delete a key pair if there is only one assigned to a service account [11].

6.2 Creating GoogleAppointment Class

As the purpose of this class is to translate a system appointment into a Google calendar event by implementing an IAppointment interface, it does not require any underlying communication between Youmanage application and Google servers and therefore implementation of this class is possible once a project reference to a Google Calendar API library is set in the integrated development environment.

Similarly to its ExchangeAppointment counterpart, GoogleAppointment class contains a private class variable of type Event reflecting the Calendar API (See 5.3.1) and is accessible by implemented properties and hence their setter and getter methods. This variable is initialized in a default constructor and is populated by methods implementing an IAppointment interface.

6.2.1 Attendees

The setter method at fist checks whether a list of attendees in an Event object is empty and if not clears the list. Consequently it reinserts attendees in a loop one by one from a list provided as a method argument by calling an add attendee method.

The getter method firstly creates a new IList instance of type string and if there are attendees in the Event, inserts each attendee email in the list and returns the populated list or in other case an empty list is returned.

6.2.2 Add Attendee Method

This method takes one argument of type string representing an email address. At first it checks whether the attendees list of Event instance is empty and in that case a new instance of EventAttendee type list is created.

Consequently, if the email supplied as an argument has a value or is not empty, it creates a new instance of the class EventAttendee in the attendees list and sets the email address of such instance.

6.2.3 Body

The setter method simply assigns the value received as an argument to the Description attribute of Event instance, whereas getter method retrieves the value stored in such attribute. No additional checks are present.

6.2.4 Start and End

Setter methods of these properties take a non-null able DateTime object as an argument and create a new instance of a null able DateTime object using the method argument as a constructor parameter. Afterwards it initializes the relevant Event attribute and assigns the null able value into a DateTime class variable of such attribute (See 5.3.1.3)

On contrary getter methods defines a not-null able DateTime variable and in case the DateTime class variable of a relevant attribute has a value, this value is assigned to the not-null able variable, in other case current date time value is assigned to avoid null pointer exception. Finally, the defined variable and its value are returned.

6.2.5 Id

Due to the insignificance of this property in the business layer and constraints regarding its representation in Event class (See 5.3.1.4), implicit setter and getter methods were used.

Implementing this property can be therefore a subject of future development perhaps regarding a two-way communication between Youmanage application and user calendars in Google Apps for Work domain.

6.2.6 IsAllDayEvent

All day event property in IAppointment interface is determined by a Boolean value, however Event class use a different mechanisms to determine such state (See 5.3.1.5). Moreover ExchangeAppointment expects all day event to be related to a single day, although Google allows it to span over multiple days.

As a result the setter method was constructed in a way to comply with both representations of all-day event. If the value passed as a parameter is true it first determines whether the proposed Event is a one day all-day event or multiple days in a row as it is expected that a one date event has no value assigned to an End attribute.

In case of the first scenario if a DateTime class variable of the attribute Start has a value this value is converted into a Date type and assigned to the class variable Date of attributes Start and End.

In case of the latter one, if a DateTime variable of attribute Start has a value it converts both of the values in Start and End attributes from DateTime type to a Date and stores them in the relevant class variables.

Corresponding getter method just need to check if class variable Date of attributes Start and End have a value an in that case return true, otherwise false.

6.2.7 IsPrivate

If the value received as a parameter in setter method is true, then attribute Visibility is set to value “private”, otherwise its set to “default”.

Similarly, getter method checks if the Visibility attributes equals to “private” and return either true or false.

6.2.8 Location

The setter method simply assigns the value received as an argument to the Location attribute of the Event instance, whereas getter method retrieves the value stored in such attribute. No additional checks are present.

6.2.9 Organizer

The setter method checks beforehand if the Organizer attribute is empty and if so it initialize it, consequently it assigns the method argument into an Email class variable. The getter method simply returns value of such property.

6.2.10 Subject

The setter method simply assigns the value received as an argument to the Summary attribute of Event instance, whereas getter method retrieves the value stored in such attribute. No additional checks are present.

6.3 Creating GoogleServiceProvider Class

The GoogleServiceProvider class implements IAppointmentService and IExternalService interfaces (See 5.4) in order to provide a uniform handling of all calendar services offered by Youmanage application. This class consists of various constants, class variables, inherited interface methods and methods essential to the class itself and an in-depth explanation of each of them is provided in the following subsections.

6.3.1 Constants

Constants within GoogleServiceProvider class can be divided into two distinct groups each serving a different purpose.

6.3.1.1 Credentials

In order to access Google Calendar API using a service account, corresponding credentials generated by Google Developers Console must be provided in the authorization flow. Such set of credentials consist of a P12 cryptographic key file and an email address of the actual service account (See 5.6.1).

Both credentials can be either stored in a Data Layer, or in other case P12 file could become a part of the project solution accessible by its path and corresponding service account email address could be hardcoded into the source code. Latter solution was determine as the more secure one and hence was implemented.

Nevertheless, due to the architecture of Youmanage application where one instance is distributed over multiple client domains, an absolute path of the P12 file couldn't be provided. As a consequence the P12 file was moved into a root subdirectory App Data and its relative path corresponding to root directory was stored in a string constant. This path can be then combined with an address of a hosting environment creating a correct absolute path.

Consequently, a string value of the service account email was assigned into another constant.

6.3.1.2 Error Codes and Error Messages

Similarly to the ExchangeServiceProvider class (See 2.3.3.2), during various testing phases, the GoogleServiceProvider class has formulated a set of distinct exceptions it can generate. Such exceptions were consequently translated into error codes and corresponding error messages and stored in pairs of integer and string constants (Figure 17).

Except the X509 certificate message which can be determined during the loading of the cryptographic file, other error messages occur at the same block of code regarding the authorization process.

Error Code	Error Message
-2	Permission not authorized. Check API scopes in the Admin console.
-3	Client not authorized. Check client access in the Admin console.
-4	Email account of user you are trying to impersonate is not valid.
-5	Provided certificate is invalid.
-6	User account you are trying to impersonate is probably suspended in GAPPS.
-7	Unknown error occurred.
-8	Cannot find the X509 certificate.

Figure 17. Error Codes and Messages

As can be seen Error Codes start from a value -2, which is due to the fact that value -1 was reserved and provided as a default return value of unused or experimental methods implemented solely by interface obligation.

6.3.2 Class variables

6.3.2.1 CalendarService _service

This class variable holds the actual instance of a Google calendar service able to manipulate user calendars and is accessible by its corresponding property defined within the class.

6.3.2.2 String _organizer

This variable serves in order to store the email address of an event organizer which is provided by a CreateAppointment method and used by the service Initialize method.

6.3.2.3 Boolean _isConnected

Purpose of this variable is to keep track of the connection status of the service throughout the service interactions with user calendars and the value stored is accessed by a corresponding class property. Moreover there is a minor difference between the understanding of a connection status in case of ExchangeServiceProvider and Google calendar service.

In case of ExchangeServiceProvider class, connection status is determined whether a calendar service is able to connect to an Exchange server, however the current implementation of Google calendar service analyses whether it can access Calendar API resources instead of Google Apps for Work domain.

6.3.2.4 String _lastErrorMessage & int _lastErrorCode

Similarly to ExchangeServiceProvider (See 2.3.3.2) these variables are used to store a relevant code and message constants values determined by exception caught during the service interactions. Such values are retrieved by using corresponding properties defined within the class.

6.3.3 Methods

6.3.3.1 CreateAppointment

This method is implemented based on the header provided by IAppointmentService interface (See 2.2.2) and its logic is similar to the ExchangeServiceProvider class (See 2.3.3.2) in order to provide a uniform handling of system appointments.

It accepts an email address of a user account in a form of a string and returns an instance of IAppointment interface initialized by a GoogleAppointment class.

The body of this method firstly checks if the email address supplied is a valid email address. In this context valid email address is any address with a value or a not empty string. No regular expressions or any form of validators were used as arguments supplied are expected to come from a Data Layer.

In case of an invalid email address, corresponding error code and error message class variables are set and no value (null) is returned. In other case class variable _organizer is assigned to the email address argument.

Consequently, service Initialization method is called and if service connection status represented by the class variable _isConnected is false no value is returned.

In case the service was successfully connected to Google Calendar API, a new instance of GoogleAppointment class is created, its organizer property is set to the email address supplied and the created instance is returned.

6.3.3.2 SaveAppointment

Analogously to the ExchangeServiceProvider class this method receives an argument in a form of a populated IAppointment instance and returns an integer code signifying a result of the event insertion.

At first the method determines whether the service is successfully connected to the Calendar API and if not returns the value of last error code variable. In other case a precondition of the method expects that the supplied argument is actually initialized by an instance of GoogleAppointment class and hence can be casted. As a result a local variable of type

GoogleAppointment is used to store a converted argument. Consequently, this variable is a part of a Calendar API call formulating a RESTful request inserting a new event into user's calendar. Such call includes an instance of the Event being inserted and a relevant calendar as parameters. Due to restrictions put on by Google (See. 5.3.2) it has been decided to insert all events into primary calendars for all users as those calendars can't be permanently deleted.

Formulated request is then executed in a try and catch block in order to determine a final result of the operation. If an exception is caught, relevant last error code and message are set based on the nature of the exception, and last error code is returned as a result. Otherwise, zero is returned on order to reflect a successful result.

6.3.3.3 Connect

Implementation of this interface method firstly calls a service initialize method and based on the state of class variable _isConnected then either returns zero to acknowledge successful connection to Google Calendar API, or last error code value if unsuccessful.

6.3.3.4 Initialize

This method is the heart of the GoogleServiceProvider class as its main purpose is to authenticate service account provided by Google Developers Console towards a relevant Google Apps for Work domain configured and consequently impersonate the concrete user and create a new instance of CalendarService class.

Implementation of the method was largely inspired by a Google Plus API example provided in the Google OAuth 2.0 documentation [12].

The method body starts with defining a string array representing Google API requested scopes. As GoogleServiceProvider need only a full access to Calendar API only a single item representing such fact is defined.

Consequently, a local variable of a certificate class is defined and instantiated in a try catch block by an absolute path to the stored P12 certificate alongside with the password requested by Google authorization server (See 6.1). If the certificate file can't be loaded into the instance of the class, class variable _isConnected is set to false, corresponding error code an error message are set and method call ends.

In other case a series of Google Calendar API library calls are constructed. Firstly, a new instance of ServiceAccountCredentials class is defined and instantiated, and unlike in the case of example Google Plus API code, an additional parameter in a form of organizer email is passed into its initializer method alongside with the array of scopes, service account email and certificate instance.

Purpose of the email parameter is to specify the corresponding Google Apps for Work domain as well as the user account to impersonate.

At this point, following the design and implementation of the ExchangeServiceProvider, a calendar service has to be able to determine whether it is connected or not before its methods CreateAppointment and SaveAppointment are called. However, during the initial testing phase it was determined that the actual token exchange between Youmanage application and Google Calendar API is initiated at the point of inserting an event into a calendar. As a result in order to determine the connection status of the calendar service beforehand, a Calendar API call requesting a token is made and wrapped in a try and catch block.

If an exception is caught during the call, the cause of the exception is determined, connection status class variable is set to false and corresponding error messages and codes are set. In other case the token response was successful, an access token was issued, a new instance of CalendarService class with given credentials was assigned to the class variable _service and the corresponding connection status was set.

6.3.3.5 HandleException

During the testing period an ambiguous behaviour has been encountered while handling exception. Perhaps due to the project or server settings, all exceptions related to Calendar API were wrapped by a System.AggregateException type [13]. However, on other servers with other settings it might be presented without the wrapper exception and therefore in order to accommodate both possibilities a handle helper method was created.

As a consequence this method takes an exception as a parameter and therefore in one case analyses it and in other case if the exception contains an inner exception, inner exception is supplied as a method argument. Based on the analysis appropriate error codes and messages are set within the method.

6.3.3.6 Other methods

There are few other methods which must be implemented by interface obligation, such methods are however unused and hence not important for the current calendar service functionality. As a consequence these methods bodies contain only either empty or default -1 return statement.

6.4 Front End Setup

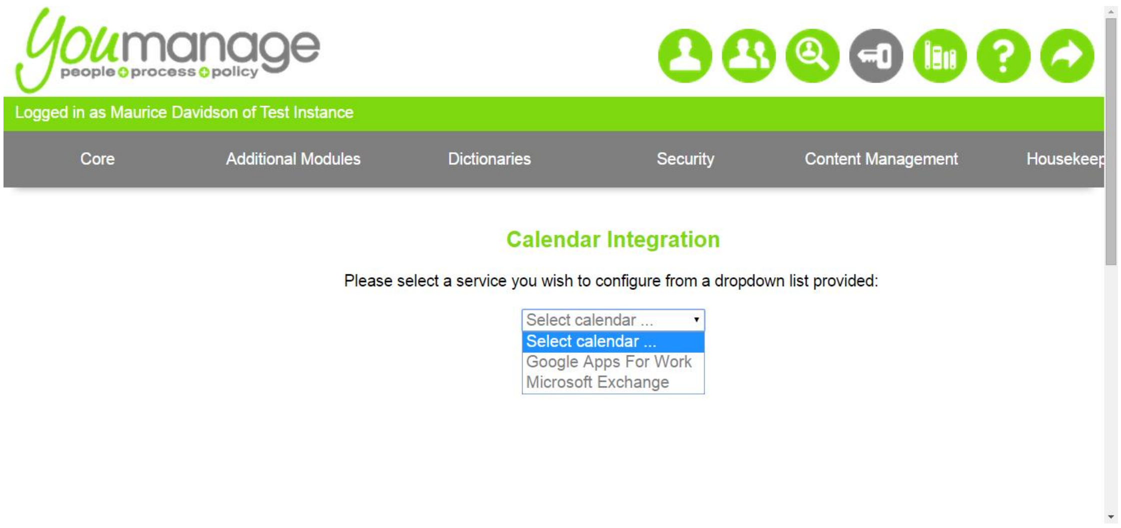
Once the Google calendar service was created, an administrator front end modification was required in order to offer the new functionality. The process of adjusting the presentation layer alongside with its corresponding code-behind files can be divided into two steps.

6.4.1 OutlookIntegration

Purpose of this page is to offer all available calendar services and based on the selected service to redirect into a corresponding setup page. Consequently, after the setup part successfully finishes this page to ensure the correct service settings are displayed.

As a result correspondingly to the earlier version (See 2.3.1), page load determines whether any calendar service was yet configured. In case no service was configured a placeholder containing a dynamical drop down list is loaded. The drop down list simply lists all t_serviceProviderType (See 2.2.3) records saved in the Data Layer, by displaying attribute “name” and by using attribute “ID” as a selected value. The drop drown list action is fired once a selected value of an item has changed and as a consequence the first record in the dropdown list serves as a dummy item in order to resolve a first item issue. A drop down list always pre-selects a first value stored when loaded and therefore without the dummy item selecting a first option would not change the selected value and hence no action would be carried out (Figure 18).

Based on the value selected from the drop down list a redirection to a service setup page is triggered. In case of Microsoft Exchange, the redirection page is ExchangeServiceSetup.aspx (See 2.3.1) and in case of Google, GoogleServiceSetup.aspx page was created and implemented.



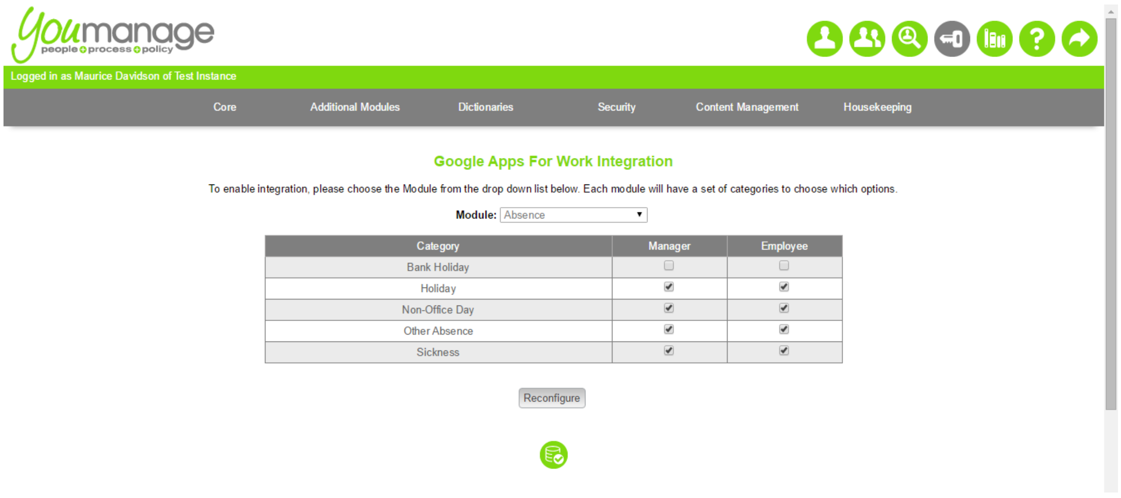
Calendar Integration

Please select a service you wish to configure from a dropdown list provided:

Select calendar ...
Select calendar ...
Google Apps For Work
Microsoft Exchange

Figure 18. OutlookIntegration.aspx Service Selection

In case a service was already configured and a record exists in the Data Layer, page load shows a corresponding placeholder with service settings determined by a service provider type id. Such placeholder contains a dynamical heading showing a name of the service provider and a uniform set of settings – permission scope and reconfigure button. Updating the permission scope updates values in t_appointmentNotificationSetting table which is not related to any concrete calendar service but serves more as a global configuration. Finally, based on the service provider type, a reconfigure button redirects to the corresponding calendar service setup page allowing it to go through the configuration process again and afterwards simply updates the selected record of t_serviceProvider (Figure 19).



Logged in as Maurice Davidson of Test Instance

Google Apps For Work Integration

To enable integration, please choose the Module from the drop down list below. Each module will have a set of categories to choose which options.

Module: Absence

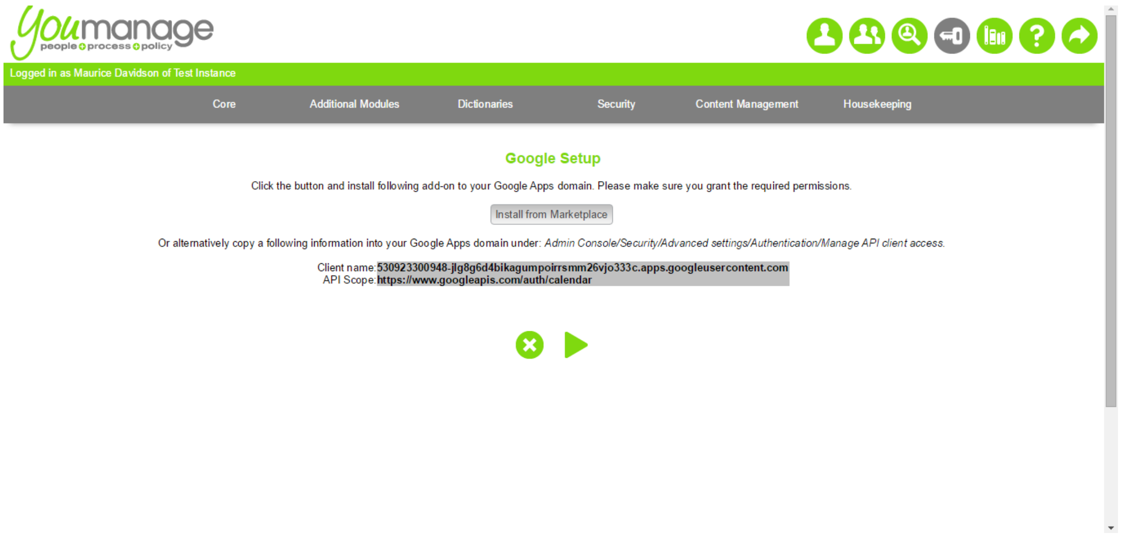
Category	Manager	Employee
Bank Holiday	☐	☐
Holiday	✓	✓
Non-Office Day	✓	✓
Other Absence	✓	✓
Sickness	✓	✓

Figure 19. OutlookIntegration.aspx Service Settings

6.4.2 GoogleServiceSetup

Similarly to ExchangeServiceSetup (See 2.3.1), the initial placeholder of this page shows the service account credentials (See 5.2.2) administrators have to adjust in their Google Apps for Work domain and an alternative way to ease this configuration step in a form of a Google Apps Marketplace button (See 5.6). Both ways have their pros and cons and therefore even though the alternative way can automate such task both of them are provided (Figure 20).

If administrators choose to copy the credentials displayed manually, they would need to allow Youmanage application access only user calendars, however if they decide to install the Marketplace application provided they would have to allow access to two additional scopes (See 5.6.1) which might not be desired. Moreover, Marketplace applications installed in the Google Apps for Work domain can´t be uninstalled, only disabled.



Google Setup

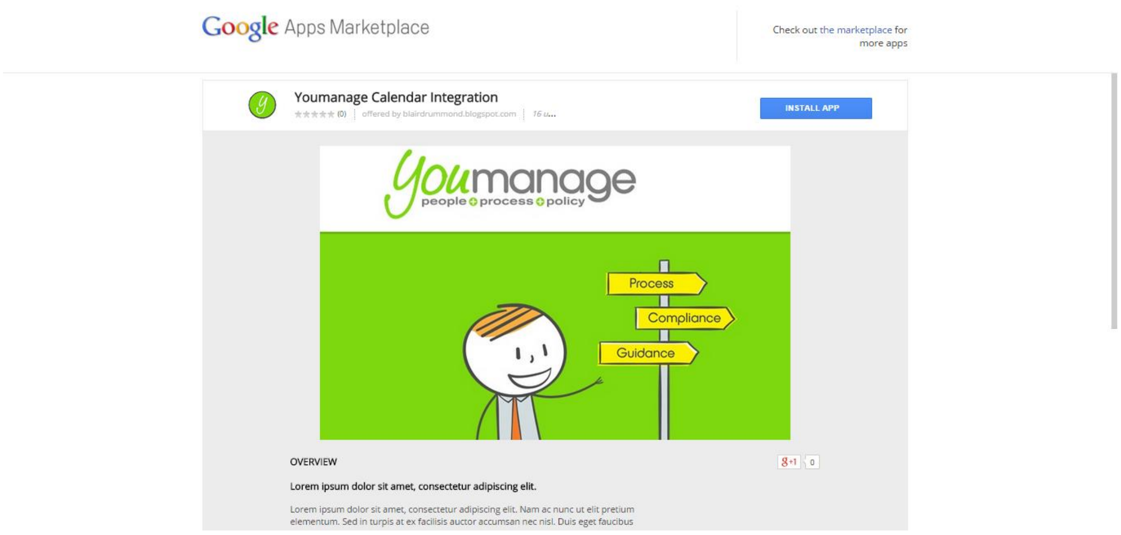
Click the button and install following add-on to your Google Apps domain. Please make sure you grant the required permissions.

Or alternatively copy a following information into your Google Apps domain under: *Admin Console/Security/Advanced settings/Authentication/Manage API client access.*

Client name: 530923300948-jlg8g6d4bikagumpoirrsmm26vjo333c.apps.googleusercontent.com
API Scope: https://www.googleapis.com/auth/calendar

Figure 20. GoogleServiceSetup Initial Placeholder

The process of implementing the Marketplace application itself simply followed the design period (See 5.6.2) and due to an amount of graphical elements presented serves as a temporary solution for Youmanage to adjust to their needs (Figure 21).


Google Apps Marketplace
Check out the marketplace for more apps
Youmanage Calendar Integration
(0) offered by blairdrummond.blogspot.com 16 u...
INSTALL APP
Process
Compliance
Guidance
OVERVIEW

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam ac nunc ut elit pretium elementum. Sed in turpis at ex facilisis auctor accumsan nec nisl. Duis eget faucibus

Figure 21. Youmanage Marketplace Application

Due to the architecture of Youmanage application, as Google cannot have the knowledge of the application instance originating the request, the redirect URL of the Marketplace application represents an official site of Youmanage Ltd. As a consequence providing a Google Marketplace button, which redirects to Google Marketplace and after application installation to the redirect URL, would lose the instance of Youmanage opened in the browser. Finally, to resolve this issue a JavaScript button was provided. Upon a button click a new browser tab is opened with the URL of the Marketplace application and thus after installation user is redirected to the official site of Youmanage still having the Youmanage instance opened in the previous tab. However as some users might block pop-up windows in their browsers not even this solution is ideal.

After administrators configure its Google Apps for Work domain, next placeholder asks them to enter a valid and persistent email of a user from their domain. As administrators might not have their Google emails listed in the Youmanage application Data Layer, entered email is used to determine the correct domain and the user to impersonate during Google calendar service connection. In addition the text box used to gather such email address is equipped with validators notifying a user if no input or not an email in its format was supplied.

Consequently, the next button click action creates a new instance of a GoogleServiceProvider class in the code-behind, passes the entered email into its class variable _organizer using the property of the same name and invokes the service Connect method (See 6.3). The next placeholder is then determined by the output of such method.

In case the connection was successful, new instance of Service Provider class is created, where during the constructor call creationDate attribute is set to current time and date and isSynchronized attribute is set to false (See 5.5). This instance is consequently populated by the email address supplied and the service provider type identifier corresponding to Google calendar service. Finally, the instance is stored in the Data Layer by the Data Mapper and a success placeholder is shown and once acknowledged redirection back to the OutlookIntegration page is performed.

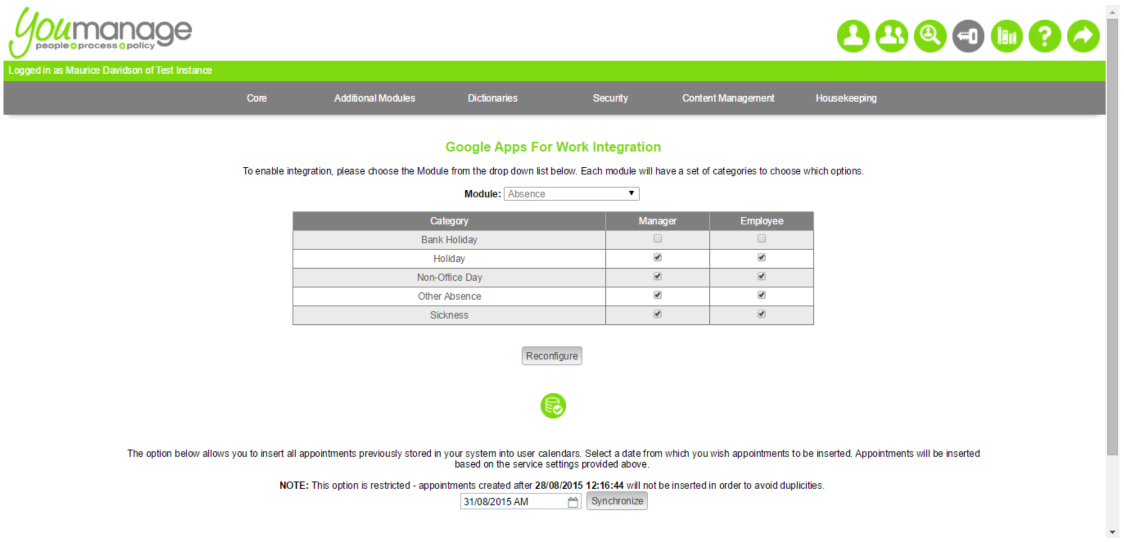
If service connection was unsuccessful, error placeholder with an appropriate error message is shown instead.

6.5 Synchronization

This functionality was implemented in the OutlookIntegration code-behind and uses the corresponding presentation expressed by an additional (synchronization) placeholder. Such placeholder is shown based on the value stored in t_serviceProvider table and its isSynchronized attribute (See 5.5).

OutlookIntegration page load determines whether service was yet configured and if stores the corresponding t_serviceProvider record in an instance of the Service Provider class. If the value of isSynchronized attribute is false, such operation wasn't yet performed and placeholder containing short description, date picker and a button is displayed.

The date picker allows administrators to select a date from which system appointments should be inserted and consequently the button click fires the synchronization method (Figure 22).



Logged in as Maurice Davidson of Test Instance

Google Apps For Work Integration

To enable integration, please choose the Module from the drop down list below. Each module will have a set of categories to choose which options.

Module: Absence

Category	Manager	Employee
Bank Holiday	☐	☐
Holiday	✓	✓
Non-Office Day	✓	✓
Other Absence	✓	✓
Sickness	✓	✓

The option below allows you to insert all appointments previously stored in your system into user calendars. Select a date from which you wish appointments to be inserted. Appointments will be inserted based on the service settings provided above.

NOTE: This option is restricted - appointments created after **28/08/2015 12:16:44** will not be inserted in order to avoid duplicities.

31/08/2015 AM

Figure 22. Modified OutlookIntegration.aspx Service Settings

This method simply queries the Data Layer and stores lists of all system appointments created from the given day until the date and time stored in t_serviceProvider table attribute creationDate which serves as a date and time stamp of when service was configured (See 6.4.2). Such restrictions make sure that no duplicated appointments can be inserted into user calendars if calendar service will be engaged without prior synchronization. For each system appointment in each list BLNotifications method is called in order to insert it into a user calendar.

Due to the chosen implementation, time performance of synchronization method can extend into tens of seconds or even greater based on the number of system appointments selected. As a consequence it is important to notify administrators about such latency and therefore once the button carrying this operation is clicked, it is consequently disabled and its label changed on the client side in order to notify user of its ongoing action. Once synchronization is completed, isSynchronized value of the Service Provider instance is set to true, such change is updated on the Data Layer and new page load omits the synchronization placeholder.

6.6 Business Layer Notification

In order to accommodate the new calendar service, some minor modifications were required in the business level and particularly in BLNotifications GetAppointmentService method (See 2.3.2.2).

This method takes care of instantiating IAppointmentService variable with the appropriate service provider configured. Firstly, calling a method GetServiceProviderByCapability returns an instance of Service Provider class containing service configuration settings and consequently based on its service provider type the rest of the GetAppointmentService method instantiate an interface variable either with ExchangeServiceProvider or GoogleServiceProvider class and its credentials. Lastly, if calendar service connection status is successful, such variable is returned.

Another difference compared to the earlier version of BLNotification is the time representation of the calendar events and its ambiguous behaviour which was uncovered while testing a Google calendar service (See 2.4). Youmanage provided a list of correct times for AM and PM appointments representation and requested a re-implementation of affected parts which was accommodated.

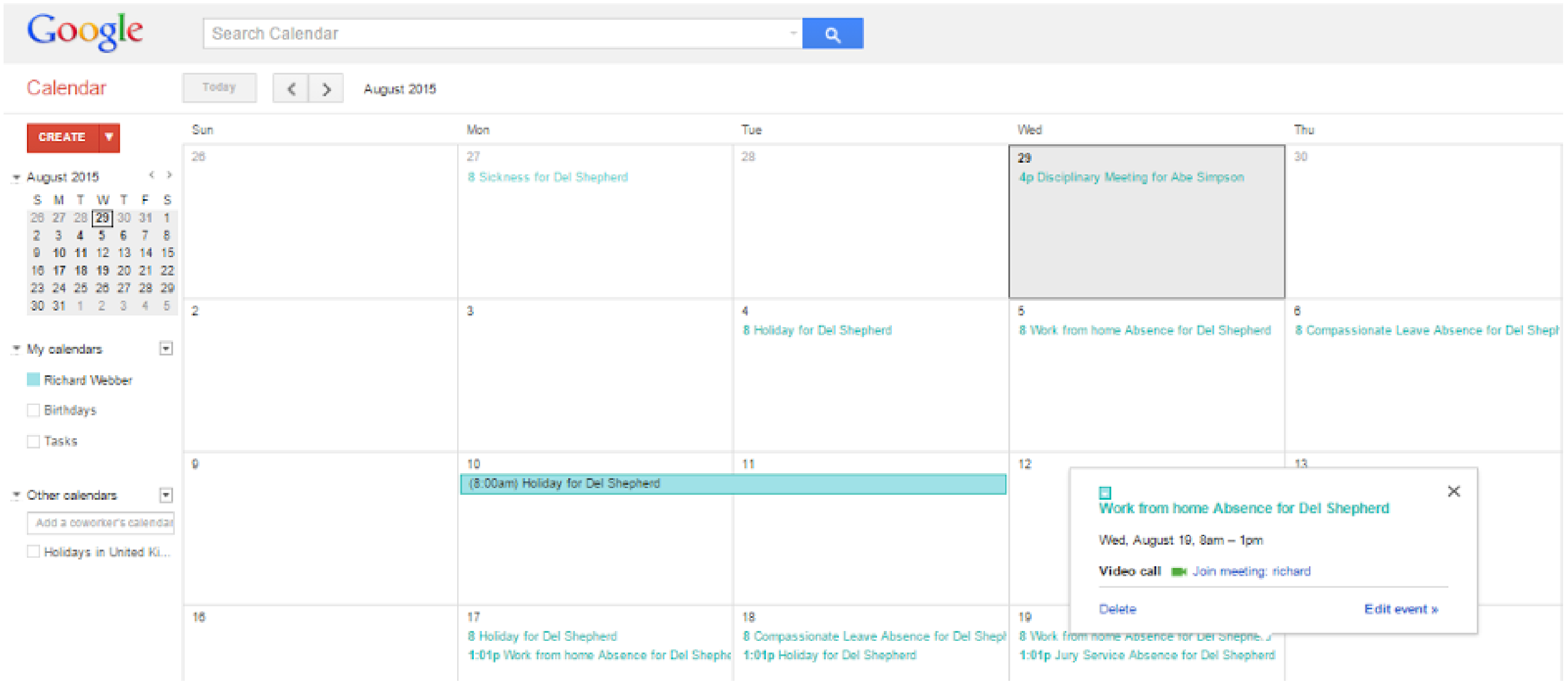
Youmanage requested any full day appointments to be represented as an event spanning from 8:00 AM till 6:00 PM and therefore to represent AM appointment as 8:00 AM – 1:00 PM and consequently PM appointment as 1:01 PM – 6:00 PM. Moreover appointments without an end time should be exactly one hour long.

As absences are the only appointments at the moment which can span over multiple days and must be able to distinguish working time representation of AM and PM a new method ConvertTime was implemented. Such method takes an instance of IAppointment class created by the calendar service alongside with start and end times of concrete absence as method parameters. Consequently, if the absence start time is 00:01 hence an AM absence, start time of the IAppointment instance Start property is set to eight hour more and one minute less making it 8 AM. The other case signifies a PM start time which is already represented correctly as 01:01 PM. And finally, if the end time of absence is 00:01 thus signifying a PM absence, end time of the IAppointment property End is set to six hours and one minute behind to the 6 PM. In other case to represent an end time of an AM absence one minute is deducted.

As there are currently four different absences supported by calendar service at the moment, in each of its CreateAppointment method a ConvertTime method is called.

In order to represent one hour appointments in each of the relevant CreateAppointment methods, an End property of IAppointment instance is simply set to one hour ahead compared to the Start property just before its passed as a parameter to calendar service SaveAppointment method (See 2.3.2.2).

The output of the Google calendar service with appropriate times can be seen in Figure 23.



Sun	Mon	Tue	Wed	Thu
26	27 8 Sickness for Del Shepherd	28	29 4p Disciplinary Meeting for Abe Simpson	30
2	3	4 8 Holiday for Del Shepherd	5 8 Work from home Absence for Del Shepherd	6 8 Compassionate Leave Absence for Del Sheph
9	10 (8:00am) Holiday for Del Shepherd	11	12	13
16	17 8 Holiday for Del Shepherd 1:01p Work from home Absence for Del Shephe	18 8 Compassionate Leave Absence for Del Shepl 1:01p Holiday for Del Shepherd	19 8 Work from home Absence for Del Shephe 1:01p Jury Service Absence for Del Shepherd	

Work from home Absence for Del Shepherd
Wed, August 19, 8am – 1pm
Video call Join meeting: richard
Delete Edit event »

Figure 23. Google Calendar Service Output

7 Testing

Testing period is crucial to determine a correct and expected behaviour of an implemented functionality. It also serves as a presentation for the client in to determine whether all requirements were met [1].

7.1 Functional Testing

Functional testing focuses on a single functionality by providing it with various inputs and analysing the resulting outputs and as such was ideal for the chosen software development approach (See 3). Each of the iteration included testing in order to fix various bugs and decrease the change of a faulty behaviour. Moreover it hasn't focused solely on the possible scenarios generated within Youmanage application, but also scenarios which could arise within a Google Apps for Work domain.

7.2 System Testing

System testing tests an application as a whole in order to determine how various parts interact with each other and is the last step before releasing a piece of software to public. Within Youmanage environment, system testing required to make a build of the current application including the new implemented functionality and its consequent release onto a testing server.

Testing was performed by an employee of Youmanage with such responsibility and initially determined some undesired behaviour and provided a set of actions to be taken in order to satisfy client's requirements (See 6.6).

Moreover, performed system testing was concerned about Youmanage functionality and its correct output in user calendars omitting the possible scenarios coming from a Google Apps for Work domain (e.g. deactivating or deleting a user).

However, final system testing confirmed the expected functionality and satisfied the requirements put on the project by Youmanage (See Appendix 1 – System Testing).

Finally, even though Google calendar service passed this part of software development lifecycle its correct behaviour is arguable as it depends on parts which couldn't be tested. As Google Developers Console provides quotas (See 6.1) for accessing Calendar API and same quotas can also be adjusted by administrators within a Google Apps for Work domain, implemented Google calendar service can still generate some undesired output.

8 Conclusion

8.1 Summary

Youmanage is a Stirling HR software company with a web-based product of the same name. One part of their product offers its clients using Microsoft Exchange server to insert system appointments into calendars corresponding to the company emails of relevant users. Such service is very popular and therefore clients demanded a support of some other available business tools in the similar way.

During analysis of the implemented Exchange solution it has been revealed that possible extension of calendar service was already expected and sufficient abstraction was provided in order to develop a new service.

While formulating requirements for the new calendar service it has been concluded that clients of Youmanage would appreciate a Google Apps for Work support offering the same functionality as the Exchange one. Moreover Youmanage wanted to offer an additional tool allowing its clients to insert appointments created in the system previously.

As the scope of the project could be divided into single functionalities and moreover as additional requirements were expected, an incremental and iterative software development approach was adopted.

During the research and design period it has been understood that many Google tools will be engaged in the project development. Some additional tools provided by Google were used in order to provide a better experience for users involved in the calendar service configuration details.

Consequently, implementation period consisted of a part interacting with Google servers and a part responsible for providing the actual functionality within Youmanage application not only by implementing the underlying logic but also redesigning responsible user front end.

Due to the nature of adopted development approach functional testing represented multiple phases of the development. Finally, system testing determined the overall success of the project.

8.2 Evaluation

Even though the feedback from Youmanage was positive and the company have an intention to offer the implemented solution to its clients, the reliability of the service can still be affected by many variables Youmanage has no control over (See 7.2).

The main objective of the project (See 1.2) was to provide an exact same functionality offered by Youmanage web application which would however interact with a different business tool. Moreover constraints required to build the functionality seamlessly into the existing system and without affecting the functionality already provided.

Additional objectives were labelled as “if time permits” and consisted of exploring the possibility of two way communication or implementing additional calendar service methods manipulating calendar events.

A two way communication supposed to not only offer the already implemented functionality but also analyse Google user calendar events and insert relevant ones as Youmanage system appointments. It has been concluded that the time allocated to this project would not be sufficient to achieve this goal; however the objective remained as a research topic and will be discussed in the next subsection. Moreover, providing such functionality would provide system inconsistency as Exchange calendar service doesn´t support it.

As the main objective provides only method for inserting events into the user calendars, additional methods intended to also update and delete event if such fact happened to the relevant appointment within the Youmanage application. Even though such functionality is implemented in Exchange calendar service it is not used within the application and in addition these methods were not even subjects of testing and hence their accurate functionality is unknown. As a consequence in order to accommodate such methods a new detailed system analysis would be required and its consequent implementation would affect multiple parts of the existing system. Moreover, corresponding testing period would have to test every affected part of the system which using the time allocated to this project wouldn’t be feasible.

Finally, provided time was sufficient to implement the main objective, and therefore another set of requirements (See 4) were formulated during the project development.

Lastly, the main objective and other additional requirements formulated afterwards were accommodated to the full satisfaction of Youmanage.

In conclusion, as the project was carried out in a real software company it provided a genuine industry experience for the developer. Such experience included tackling new technologies not encountered in the related academic sphere as well as gaining an inspiration from a source code written by other developers.

8.3 Future Work

Even though Exchange calendar service was designed providing a sufficient abstraction for a future extension in a uniform way, new authentication technologies emerged since that time leaving the Google service an obligation to implement interface methods it doesn't require only in order to be treated uniformly.

Possible future development could therefore focus on abstracting away relevant parts of a relevant service. Moreover as initialization part of new Google calendar service is almost unchanged for initializing any Google resource server, it could be also abstracted away as a template for initializing other future Google services (e.g. Google Drive).

During the development it has been also revealed that while engaging a calendar service, user is notified of its connection status at the time of service configuration, however while creating system appointments user has no knowledge of the connection status or event insertion result. While analysing this behaviour large part of the system would need a rework in order to offer error messages corresponding to not only an employee, but also a manager.

In addition, in case of business layer CreateAppointment method for absence non-office day, recurrence is solved using loops and inserting events, just by modifying dates and one by one, however it has been found that both (Exchange and Google) service events support recurrence which seems as a more efficient way to handle such events in the future.

Finally, for the future development of a two way communication it was expected that Youmanage application would access user calendars perhaps one time at the morning and one time in the afternoon each day to determine possible events entitled as system appointments. As each system appointment calendar event is determined by its prefix, name of the employee, organizer attribute, attendee attribute, start date and end date it should be possible to determine if a particular event belongs to the Youmanage application as well as to which employee and manager. However this would require comparing all appointments stored in the Youmanage application with all events in a user calendar based on such attributes which seems as an inefficient solution. By modifying Data Layer tables (and corresponding Types and Accessor classes) for each appointment type and adding an additional attribute determining an identifier of an event (using UUID algorithm) it would be much easier to distinguish what kind of events were generated by Youmanage application and which were not, thus providing a filtered set of events to be analysed. Moreover providing an UUID identifier as an event id while engaging calendar service from within a Youmanage application would easy a possible future implementation of delete and update methods of such service.

References

[1] Graham, J., *Redesign & Reconstruction of a HR Absence and Holiday Planner*, School of Natural Sciences, University of Stirling, 2013.

[2] Microsoft. How to work with code-behind class files in an ASP.NET application by using Visual Basic .NET, https://support.microsoft.com/en-us/kb/312311, August 2015

[3] Cockburn, A. Using Both Incremental and Iterative Development, *Software Engineering Technology*, 27-30, May 2008

[4] Google. Using OAuth 2.0 to Access Google APIs, https://developers.google.com/identity/protocols/OAuth2?hl=en, August 2015

[5] Google. Using OAuth 2.0 for Server to Server Applications, https://developers.google.com/identity/protocols/OAuth2ServiceAccount, August 2015

[6] Google. Google Apps Calendar API Concepts and Use Cases, https://developers.google.com/google-apps/calendar/concepts, August 2015

[7] Google. Events, https://developers.google.com/google-apps/calendar/v3/reference/events, August 2015

[8] Stack Overflow. Google Apps Marketplace vs Chrome Web Store, http://stackoverflow.com/questions/29248168/google-apps-marketplace-vs-chrome-web-store, August 2015

[9] Google. Publishing Your App, https://developers.google.com/apps-marketplace/listing, August 2015

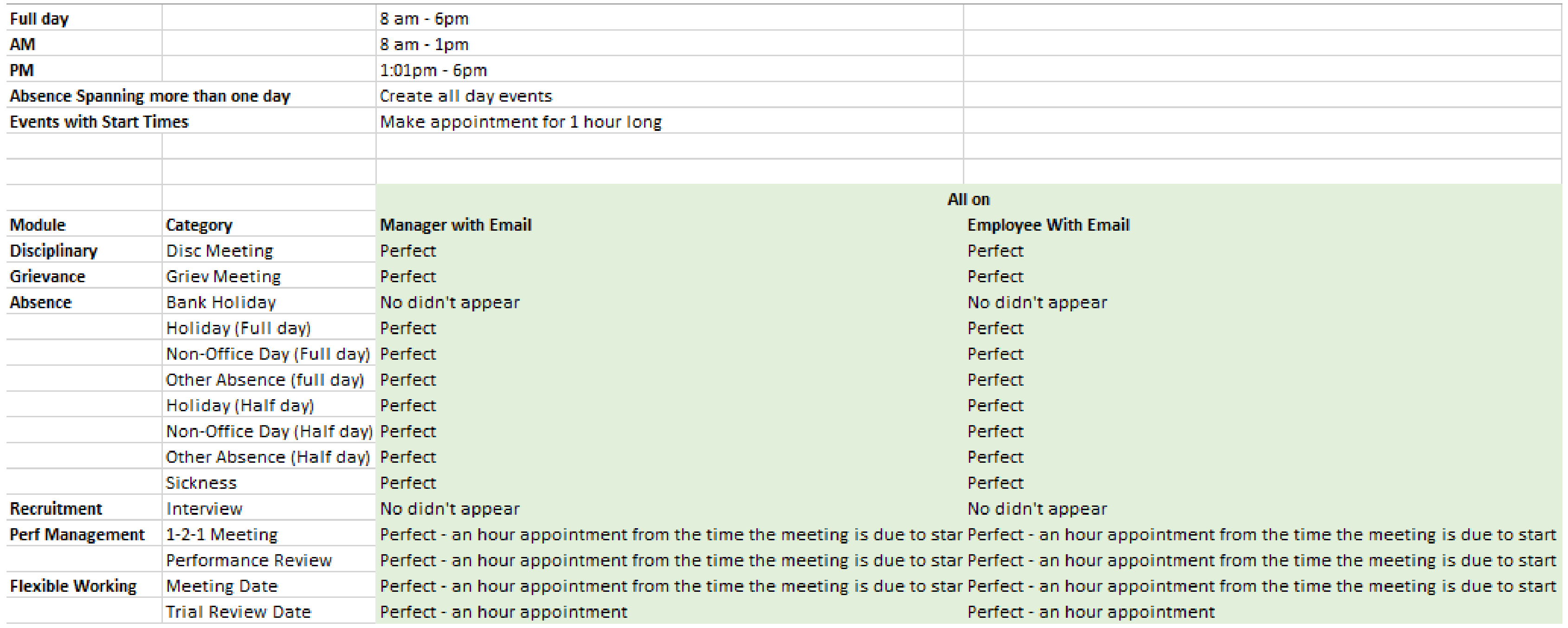
[10] Nuget. Home, https://www.nuget.org/, August 2015

[11] Google. Developers Console Help, https://developers.google.com/console/help/new/#serviceaccounts, August 2015

[12] Google. OAuth 2.0, https://developers.google.com/api-client-library/dotnet/guide/aaa_oauth, August 2015

[13] Stack Overflow. Using GoogleApisClient ServiceAccountCredential calling RequestAccessTokenAsync throws Exception, http://stackoverflow.com/questions/21483599/using-googleapisclient-serviceaccountcredential-calling-requestaccesstokenasync, August 2015

Appendix 1 – System Testing



Full day		8 am - 6pm	
AM		8 am - 1pm	
PM		1:01pm - 6pm	
Absence Spanning more than one day		Create all day events	
Events with Start Times		Make appointment for 1 hour long	
		All on	
Module	Category	**Manager with Email**	**Employee With Email**
Disciplinary	Disc Meeting	Perfect	Perfect
Grievance	Griev Meeting	Perfect	Perfect
Absence	Bank Holiday	No didn't appear	No didn't appear
	Holiday (Full day)	Perfect	Perfect
	Non-Office Day (Full day)	Perfect	Perfect
	Other Absence (full day)	Perfect	Perfect
	Holiday (Half day)	Perfect	Perfect
	Non-Office Day (Half day)	Perfect	Perfect
	Other Absence (Half day)	Perfect	Perfect
	Sickness	Perfect	Perfect
Recruitment	Interview	No didn't appear	No didn't appear
Perf Management	1-2-1 Meeting	Perfect - an hour appointment from the time the meeting is due to star	Perfect - an hour appointment from the time the meeting is due to start
	Performance Review	Perfect - an hour appointment from the time the meeting is due to star	Perfect - an hour appointment from the time the meeting is due to start
Flexible Working	Meeting Date	Perfect - an hour appointment from the time the meeting is due to star	Perfect - an hour appointment from the time the meeting is due to start
	Trial Review Date	Perfect - an hour appointment	Perfect - an hour appointment