

**DVI Systems**  
**Matthew McConnell & James Hambleton**  
**Final Report**

**I. Abstract**

Data Verity is a consulting firm that provides its clients with data analysis from data provided by their clients. As a part of their services, they want to supply their clients with a calendar system within their current software. The problem that they have with current Customer Relationship Management (CRM) software on the market is that the calendars systems do not account for multiple users. In other words, firm managers were unable to see multiple employee calendars at the same time using CRM. Along with this functionality, Data Verity would also like to implement an email event sharer and reminder within the calendar. The problem currently faced with the email implementation is that Data Verity currently does not have an email server, so they need a system setup that can send these reminders to clients without being blacklisted by major email providers. The overall benefit this system will provide is better management for the clients who use Data Verity's software.

The goal of the project is to implement a calendar system and embed it into Data Verity's current software, along with setting up an email server so that calendar notifications and reminders can be sent out.

**II. Introduction**

The purpose of the project is to provide Data Verity, a consulting firm, the ability to have a calendar and email notification system embedded into their current software that they provide their clients. The idea behind this project came from the realization that current CRM software on the market do not provide their users a way of setting up meetings and events.

**III. Requirements**

**A. Functional Requirements**

The first functional requirement of the project was to obtain a calendar system that could be embedded into the DVI's current software. This requirement helped to make sure that we minimized the amount of time that was spent on basic calendar implementation. To do this a calendar system that uses similar coding language, which is Ext-JS, as the Data Verity's current software was preferred. Once the calendar system was obtained, it had to be modified to account for multiple calendars, which current CRM systems are incapable of doing.

In order to provide the functionality the client wanted for the calendar system, the calendar creation wizard had to be overloaded. The original calendar creation wizard did not support functionality to share the calendar amongst multiple users. Along with this, the idea of personal calendars also had to be handled. The personal calendar allows the user to decide whether they want their fellow employees to be able to see their events, just their availability, or nothing at all. Within the shared calendar option of the wizard, the user would be able to invite others to a shared calendar. The calendar creation wizard also supports the feature to grab others' calendars and put them into a single calendar so that a user could view them.

Another one of the requirements of the calendar system is that it must be able to send emails and interface with Microsoft Outlook's calendar system. Both of these require sending data over the internet and in this day and age, email is a common part of day-to-day life. This is easy to do with Simple Mail Transfer Protocol (SMTP). SMTP has grown over the years and is no longer secure enough. This is where email authentication comes in. It provides the receiver with a way of knowing if the email comes from where it says it does. The main problem with sending emails is that each of the main email providers uses a different type of email authentication. Microsoft uses Sender ID, AOL uses SPF, while Yahoo and Gmail use DomainKeys. In order for our client's email server to send emails it must implement each of the above forms of authentications. On top of having these three forms of authentications, they also need to report their DNS so that reverse DNS lookup is possible. With all of these methods implemented, the client's emails will be able to reach their customers without much trouble. This will allow Data Verity's calendar system to send calendar invites to people who are not using their calendar from day to day.

The other side of the email system is being able to take an incoming email and checking to see if it is a calendar invite.

## **B. Non-Functional Requirements**

Non-functional system constraints included confidential client information on Data Verity servers that prevented us, as students, from directly connecting to and using the servers. This provided a constraint in the fact that we had to directly work with our advisor for server related work. The reason this constraint exists is that Data Verity guarantees their clients a certain level of security within the code, and server side scripts that they run.

Another constraint would be when it comes time to install the needed software on the system. We will need direct access to servers, with the advisor overseeing the installation process to ensure that no security protocols were breached.

## **C. Scope**

Requirements that were absolutely required:

- Implementation of calendar system into current software
- Providing functionality for email reminders
- Sending calendar invites
- Able to check incoming email to see if they are calendar invites
- Ensuring that emails adhere to the several main email authentication types
- Extending functionality of the calendar system

#### IV. Design

The calendar and email system that was produced for Data Verity was embedded into their current Ext-JS webtop. To begin a client will login to Data Verity's software from their computer (see Figure 1). Once they can access the virtual website desktop (webtop), the client then will be able to open the calendar program from the webtop (see Figure 2). From there, the program will load the clients' private calendars. Through the additional functionality that was provided to the system, users are able to see other users shared calendars, or the calendars of employees underneath them. From here, the user may add and or edit events on the calendar(s) that he or she has opened.

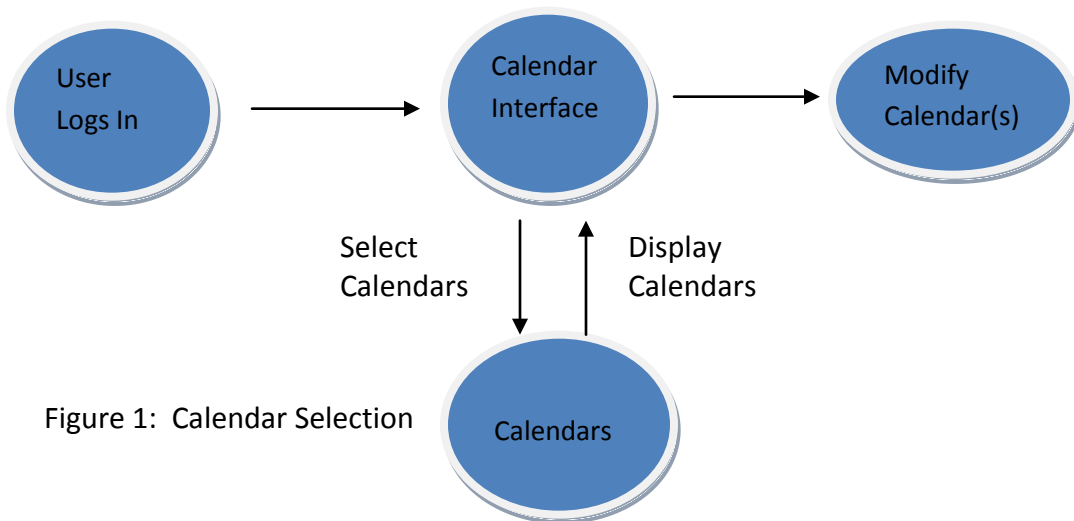


Figure 1: Calendar Selection

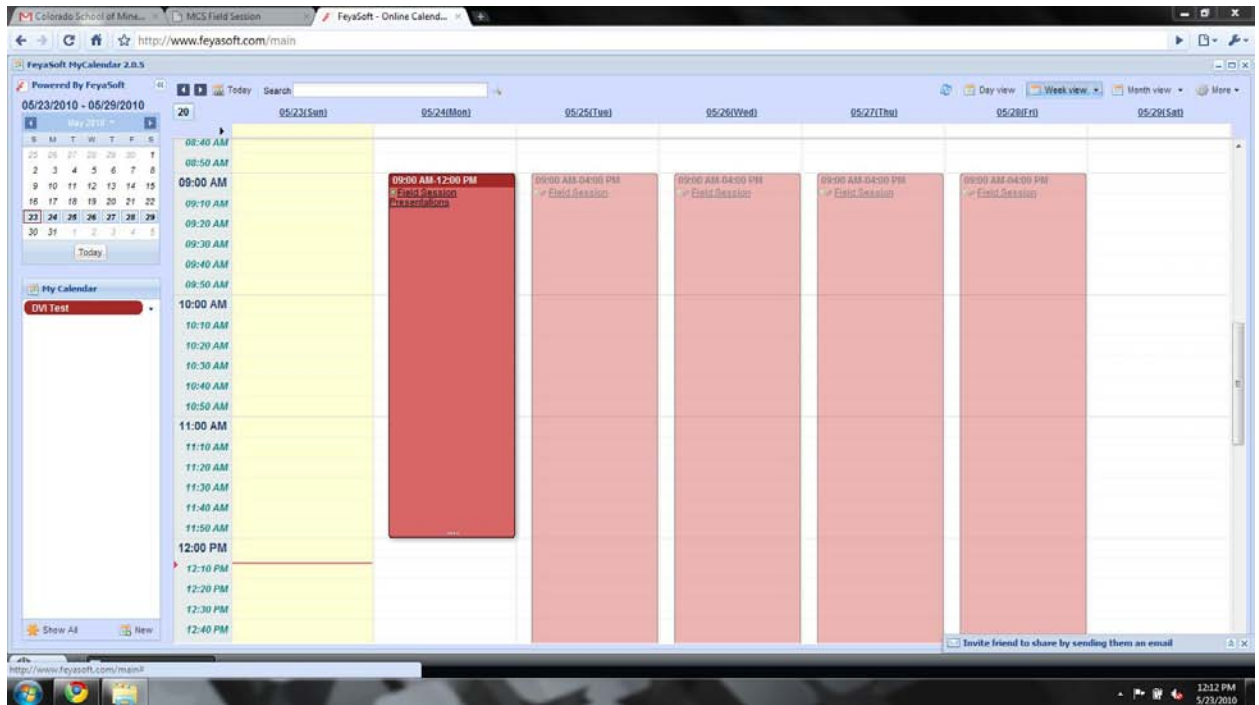


Figure 2: Webtop Calendar

The other portion of the system was to have the calendar, through an email server that we helped Data Verity setup, send out email notifications to the user of calendar events. Figure 3 outlines how the calendar system will notify the user of an event. Upon creation of a shared event, the calendar will send out a request to the email server to send out a notification(s) to the user(s).

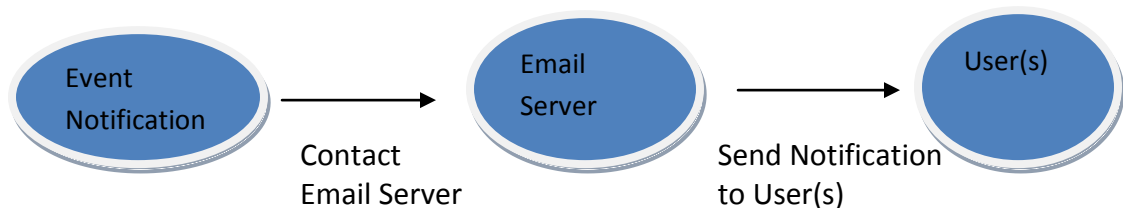


Figure 3: Email Reminders/Notifications

The email that is sent from the calendar, i.e. when someone creates an event, will be compatible with Outlook. This is important since Outlook is the standard when it comes to calendar systems. Due to this, our clients wanted to make sure that their calendar system would talk with Outlook. This was done by recreating the format Outlook uses when it sends out notifications. This required scripts on the server that recreated the same headers that Outlook uses.

Sending an email via PHP was rather easy. The tricky part was making sure that the message was formatted in the right way so that it was sent as a calendar invite. This was done by setting up the email headers in the right format.

```
From: "P. David Flammer" <pflammer@mines.edu>
To: <gordon@dataverity.net>, <mmcconne@mines.edu>, <jhamblet@mines.edu>,
    <jhdvITest@gmail.com>
Subject: Testing parsing function
Date: Tue, 15 Jun 2010 11:38:20 -0600
Message-ID: <46CAD62B6CA1461BA1635B5CE60C1CB7@MammaJamma>
MIME-Version: 1.0
Content-Type: text/calendar; method=REQUEST; charset="utf-8"
Content-Transfer-Encoding: 7bit
X-Mailer: Microsoft Office Outlook 11
Thread-Index: ACSMSYu21dqAYH5NRHanBC1X1a2tJA==
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.5931
```

Figure 4: Calendar Invite Headers

```
Message-ID: <4C164EAC.4000508@mines.edu>
Date: Mon, 14 Jun 2010 09:45:48 -0600
From: Roman Tankelevich <rtankele@mines.edu>
User-Agent: Thunderbird 2.0.0.24 (windows/20100228)
MIME-Version: 1.0
To: <jhamblet@mines.edu>, <mmcconne@mines.edu>
Subject: Last week
Content-Type: text/plain; charset="ISO-8859-1"; format=flowed
Content-Transfer-Encoding: 7bit
```

Figure 5: Plain Text Headers

Figure 4 and 5 both show a part of the email headers from two different emails. Figure 4 is from an email with calendar information and Figure 5 is just a plain text email. The formatting is different but that is because two different email systems were used. Figure 4 is from Outlook while Figure 5 is from Thunderbird. The formatting does not matter too much as long as the right information is in the email. As we can see, both are using MIME, Multipurpose Internet Mail Extensions, which means that they can send more than just plain text. The line that has Content-Type: tells the receiving email server what will be in that in email. In the case of Figure 4, the type is "text/calendar" where as in Figure 5 the type is "text/plain". There are many other types out there as well. For example, you can have "multipart/mixed" meaning you are going to have several different types in the email, "text/html" for when you need to use html in the email, or "application/ics" for sending attachments. Ics is the file extension for vCalendar so "application/ics" would be an attachment that has a vCalendar encoded. This field works two ways for us. It allows us to not only send the calendar information out but we can also use it in parsing emails.

```

BEGIN:VCALENDAR
PRODID:-//Microsoft Corporation//Outlook 11.0 MIMEDIR//EN
VERSION:2.0
METHOD:REQUEST
BEGIN:VEVENT
ATTENDEE;ROLE=REQ-PARTICIPANT;RSVP=TRUE:MAILTO:gordon@dataverity.net
ATTENDEE;ROLE=REQ-PARTICIPANT;RSVP=TRUE:MAILTO:mmcconne@mines.edu
ATTENDEE;ROLE=REQ-PARTICIPANT;RSVP=TRUE:MAILTO:jhamblet@mines.edu
ATTENDEE;ROLE=REQ-PARTICIPANT;RSVP=TRUE:MAILTO:jhdvITest@gmail.com
ORGANIZER:MAILTO:pflammer@mines.edu
DTSTART:20100615T180000Z
DTEND:20100615T193000Z
LOCATION:Meyer Hardware in the basement
TRANSP:OPAQUE
SEQUENCE:0
UID:040000008200E00074C5B7101A82E0080000000090CD17417F0CCB010000000000000000100
00000788AE9B179E8704C957F40B01C3D54E4
DTSTAMP:20100615T173820Z
DESCRIPTION:when: Tuesday\, June 15\, 2010 12:00 PM-1:30 PM (GMT-07:00)
Mountain Time (US & Canada).\nwhere: Meyer Hardware in the
basement\n\n*~*~*~*~*~*~*~*~*~*\n\nSomething great is happening. You
should come.\n
SUMMARY:Testing parsing function
PRIORITY:5
X-MICROSOFT-CDO-IMPORTANCE:1
CLASS:PUBLIC
BEGIN:VALARM
TRIGGER:-PT15M
ACTION:DISPLAY
DESCRIPTION:Reminder
END:VALARM
END:VEVENT
END:VCALENDAR

```

Figure 6: vCalendar code block

Figure 6 is a block of vCalendar code. This is what is interpreted, along with the headers, to make the calendar invite we see in our emails. The email sending function can generate this code block so that it sends a calendar invite that has the accept field like in Figure 7.

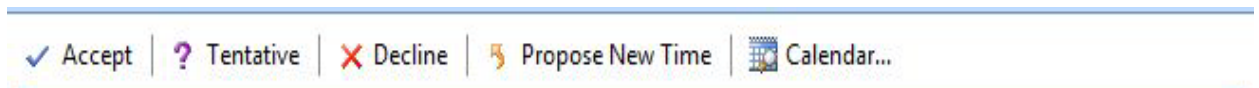


Figure 7: Calendar accept field (taken from Microsoft Outlook 2007)

In order to make this bar, the method fields of the Content-Type line and in the vCalendar must be set to request. If they are not the same, the bar will not show up. It is a mismatched type problem. The email server is not sure how to handle it, so it just ignores it. Generating the vCalendar code is simple. You need to pass in who is to be invited, the person who organized the event, when it will start, when it will end, a description and a summary of the event. From Figure 4 and 6, the TO and the ATTENDEE lines use the same emails and the FROM and ORGANIZER line do as well. The DTSTART, DTEND, and DTSTAMP are the date fields. DTSTART is when the event will start, DTEND is when it will end and DTSTAMP is the time stamp for when the email was sent. DTSTAMP



is a generated field long with the UID or the unique identifier. All of the other fields are hard coded and are standard in all vCalendar blocks. The only field we do not worry about is the VALARM. A VALARM is just the reminder that is sent a few minutes before the event. This type of reminder is more of an active reminder. It will send the email and/or make a pop up, saying your email is open.

```
To: jhamblet@mines.edu, rexregium@gmail.com, jhDVITest@gmail.com
Subject: Demo for final
From: jhamblet@subverted.mines.edu
MIME-version: 1.0
Content-class: urn:content-classes:calendarmessage
Content-type: text/calendar; method=REQUEST; charset=UTF-8
Content-Transfer-Encoding: 7bit
Message-Id: <20100618212938.5CA6263890D@subverted>
Date: Fri, 18 Jun 2010 15:29:38 -0600 (MDT)
```

```
BEGIN:VCALENDAR
PRODID:-//Data Verity Inc//Data Verity Calendar 1.0 MIMEDIR//EN
VERSION:2.0
METHOD:REQUEST
BEGIN:VEVENT
ATTENDEE;ROLE=REQUIRE-PARTICIPANT;RSVP=TRUE:MAILTO:jhamblet@mines.edu
ATTENDEE;ROLE=REQUIRE-PARTICIPANT;RSVP=TRUE:MAILTO:rexregium@gmail.com
ATTENDEE;ROLE=REQUIRE-PARTICIPANT;RSVP=TRUE:MAILTO:jhDVITest@gmail.com
ORGANIZER:MAILTO:jhamblet@subverted.mines.edu
DTSTART:20100625T161000Z
DTEND:20100625T163000Z
LOCATION:little room
TRANSP:OPAQUE
SEQUENCE:0
DTSTAMP:20100618T212938Z
UID:119e54d2ecla2eb57ee0ae9f9a06c884
DESCRIPTION:Demoing the email function
SUMMARY:Demo
PRIORITY:5
CLASS:PUBLIC
END:VEVENT
END:VCALENDAR
```

Figure 8: vCalendar code block made by the email sending function

Figure 8 shows the output from the email sending function. It has all the needed headers and parts for the vCalendar code.

When it comes to parsing the email, we look for the same fields since those hold the important information about the event. While the fields themselves are standard, the format of the fields is very relaxed. Figure 9 shows the vCalendar code from Google calendar.

```

BEGIN:VCALENDAR
PRODID:-//Google Inc//Google Calendar 70.9054//EN
VERSION:2.0
CALSCALE:GREGORIAN
METHOD:REQUEST
BEGIN:VEVENT
DTSTART:20100615T010000Z
DTEND:20100615T023000Z
DTSTAMP:20100614T145432Z
ORGANIZER;CN=jhamblet@mymail.mines.edu:mailto:jhamblet@mymail.mines.edu
UID:g87sdtrbrfthoclhr7f5r6ukk@google.com
ATTENDEE;CUTYPE=INDIVIDUAL;ROLE=REQ-PARTICIPANT;PARTSTAT=ACCEPTED;RSVP=TRUE
;CN=jhamblet@mymail.mines.edu;X-NUM-GUESTS=0:mailto:jhamblet@mymail.mines.e
du
ATTENDEE;CUTYPE=INDIVIDUAL;ROLE=REQ-PARTICIPANT;PARTSTAT=NEEDS-ACTION;RSVP=
TRUE;CN=rexregium@gmail.com;X-NUM-GUESTS=0:mailto:rexregium@gmail.com
CREATED:20100614T145008Z
DESCRIPTION:lalalalala\nView your event at http://www.google.com/calendar/e
vent?action=VIEW&eid=Zzg3c2R0cmJyZnRob2NsaHJ1N2Y1cjZ1a2sgcmV4cmVnaXVtQG0&to
k=MjUjamhhbWJsZXRAbXltYWlsLm1pbmVzLmVkdTYyY2JiYzMzYjA3MTNkZmQ3NmRiYjNjZWU2Z
mY2MGMzMTMxMGJmZmQ&ctz=America%2FDenver&hl=en.
LAST-MODIFIED:20100614T145431Z
LOCATION:da box
SEQUENCE:0
STATUS:CONFIRMED
SUMMARY:uber test
TRANSP:OPAQUE
END:VEVENT
END:VCALENDAR

```

Figure 9: vCalendar Code Block made by Google

From Figure 8 and 9 we can see that the ATTENDEE lines are vastly different. One reason for this difference is that Google adds the organizer to the attendee list and sets the PARTSTAT field to ACCEPTED among other information that is added. PARTSTAT stands for participate status. This does make sense though since we can assume the organizer will be going to his/hers own event. What this does though is make parsing incoming emails more difficult. Since there are more fields, we need to search for not only that field but what the delimiter is as well. The parsing function uses a helper function for this. It is passed the field that we are looking for in the ATTENDEE line, i.e. PARTSTAT, and it will return the value, the part after the '=' and before the ';' or after the ':' in the case of the MAILTO field. Below is the output from the parsing function:



```

Array
(
  [start] => 2010-06-09 08:00:00
  [end] => 2010-06-09 08:30:00
  [date] => 2010-06-04 15:54:19
  [description] => When: Wednesday\, June 09\, 2010 8:00 AM-8:30 AM (GMT-07:00) Mountain Time (US & Canada).\nWhere: CTLM 132\n\n*****\n\n\n
  [summary] => New invitation
  [location] => CTLM 132
  [attendee] => Array
    (
      [org] => pflammer@mines.edu
      [attendee1] => jhamblet@mines.edu
    )
  [status] => Array
    (
      [org] => none
      [attendee1] => none
    )
  [role] => Array
    (
      [org] => none
      [attendee1] => REQ-PARTICIPANT
    )
  [type] => Array
    (
      [org] => none
      [attendee1] => none
    )
)

```

Figure 10: Output from Email Parsing Function

The following Use Cases describe the process of how the calendar system and email system works:

#### Primary Flow #1:

1. User Opens Calendar(s)
2. Program Displays Users Calendar(s)
3. User Adds Event to Calendar(s)
4. Program Displays New Event to User

After Data Verity's clients log into the Data Verity software they can proceed to open the calendar program. After the initial loading, the calendar program will show the user his or her calendars. From there the user can create and/or modify events on his or her calendars. For creating an event, the user merely needs to double click a time slot, or click and drag the mouse across a period of time on the calendar. From there the user inputs the event subject, description, and which calendar the event belongs too. The user may also decide to input further detail such as repeating the event if he or she goes

to the advanced event editor. Once the user is done, they can click the save button and the calendar program will render with the new event.

**Primary Flow #2:**

1. User Shares Event with Other(s)
2. Program Asks User for Event Sharers Information
3. User Provides Event Sharers Information
4. Invite(s) are Sent Out

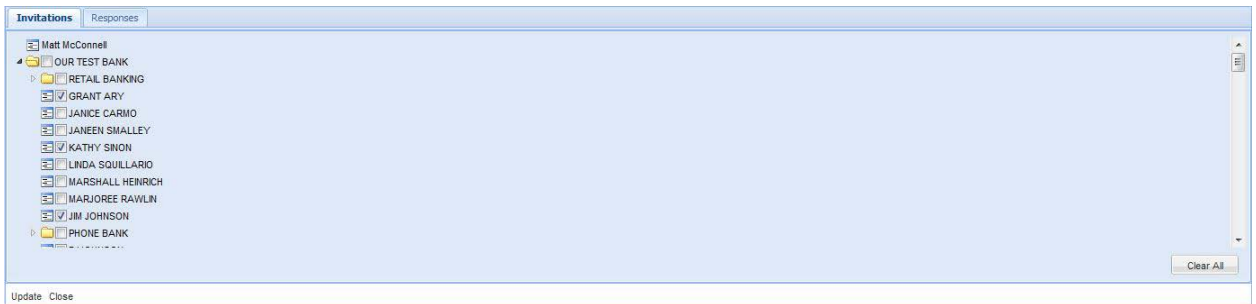


Figure 11: Event Sharer – Invitation Tree

If the user wishes to share an event with other users of their firm, they need merely right-click the event and click 'Share Event' (see Figure 11). The first tab of the screen that pops up will display an Ext-JS Tree hierarchy of the users firm, where upon the user can select the employees he or she wishes to share the event with. From there the user can click 'Update' to send out the invites to the other employees. In the second tab, (see Figure 12) the user can view the status of everyone's invitation in an Ext-JS Grid and whether they have accepted, declined, or need a different time. A short message can be added to their status if they so desire.

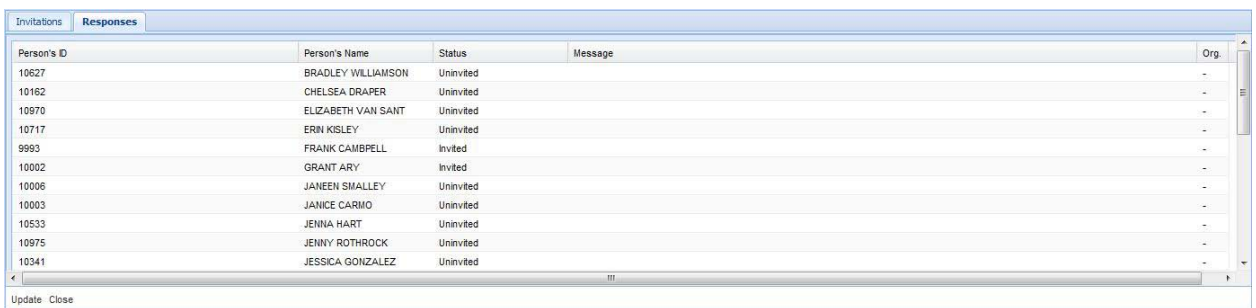


Figure 12: Event Sharer – Response Tree

**Primary Flow #3:**

1. User Creates a New Calendar
2. Program Asks User for Basic Calendar Information
3. User Provides Basic Calendar Information

4. Program Asks User for Calendar Type
5. User chooses between Personal, Shared, or View Other

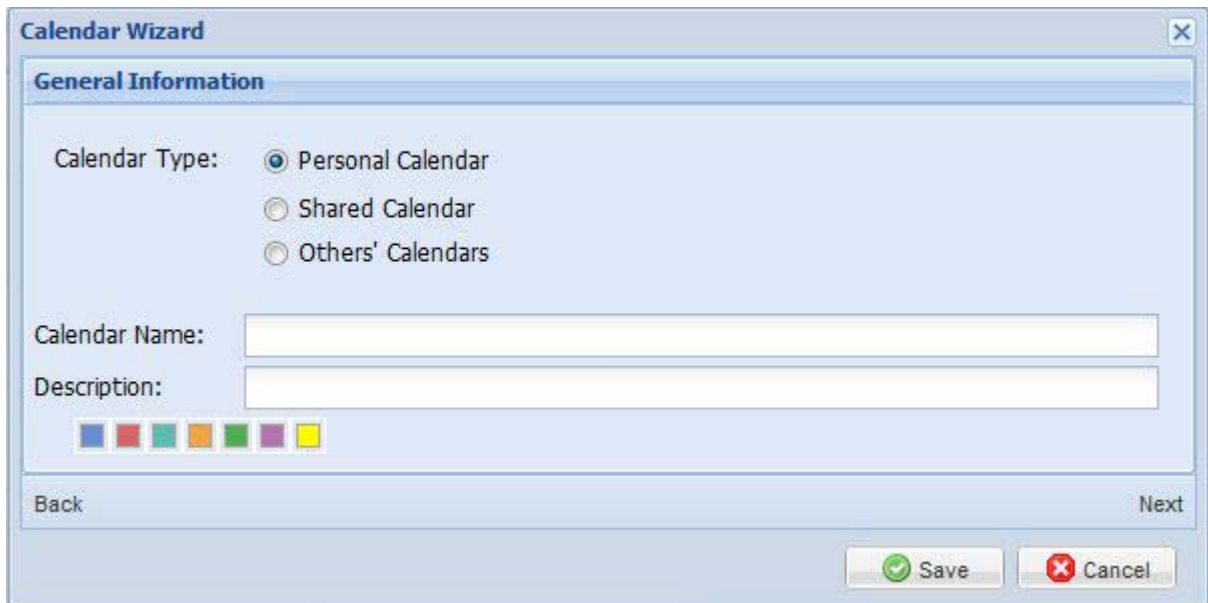


Figure 13: Calendar Creation Wizard

The Calendar Creation Wizard (see Figure 13) has been updated and redone from the original Calendar Creation Wizard that came with the calendar program. The User will now click on the “create new calendar button”, whereupon the program will pop up the first part of the wizard. The Program will ask for the same basic information that it did from the original calendar wizard. The user will input the name, description, and select a color for the calendar to be displayed in. The program will then ask the user for the type of calendar. The user will select between Personal Calendar, Shared Calendar, or View Other Calendar.

Primary Flow #3A:

6. User Chooses Personal Calendar
7. Program Asks How User Wishes to Display this Calendar to Others
8. User Chooses between Hide, Show Availability, or Show All
9. Program Asks if User would like to make Exceptions
10. User Chooses Individual Users who have Exceptions
11. User Saves Calendar
12. Program Allows Users through View Other to See Users Calendar Based Off the Settings the User has Selected

If the user chooses to create a Personal Calendar, the Calendar Creation Wizard will then go on and ask the user for additional information. Since this is the Users Personal Calendar, the user will have to choose how he or she wishes others to see this calendar.

The user can choose to hide it from others, show only availability (the events will show as just busy), or show all. The Program will then ask the user if there are any exceptions to the rule. The user then can manually go employee by employee and choose different options for individual employees or leave the exceptions on 'none' to have the original option be the rule.

#### Primary Flow #3B:

6. User Chooses Shared Calendar
7. Program Asks User who the Viewers are for the Shared Calendar
8. User Selects Users who can View Calendar
9. Program Asks User who the Modifiers are for the Shared Calendar
10. User Selects Users who can Modify the Calendar
11. User Saves Calendar
12. Calendar Invites are Sent Out
13. User can Check Status on whether Users have Accepted, Declined, Etc.

If the user chooses Shared Calendar, the program will proceed to give the user a tabbed panel where the user can select between the viewers and modifiers of the calendar. Each individual tab will look the same as the Event Sharer, with the user being able to select users through the tree hierarchy on the first sub-tab and being able to see the status of the invitations on the second sub-tab. Once the User Saves the Shared Calendar, the Calendar Invites will be sent out to the employees selected.

#### Primary #3C:

6. User Chooses View Other
7. Program Asks User Tree w/ Firm Employees
8. User Selects User(s) whose Calendar(s) he/she wants to View
9. User Saves Calendar

If the user chooses to View Others' Calendars, the program will display a tree hierarchy similar to that seen in the Event Sharer, where the user can individually select which user's calendars they wish to view. Upon saving the calendar, the program will determine whether through the personal calendar, the user can see everything, nothing, or just at what times the owner of the calendar is busy.

#### Alternate Flow #1:

10. User Opens Calendar(s)
11. B) Program is Unable to Open Required Calendar(s)
12. B) User chooses new Calendar(s) --> continue at step 2

#### Alternate Flow #2:

6. C) User Chooses to Skip Reminder
7. C) Program Does Not Remind User

#### Primary Flow #4:

1. User Event is Coming Up
2. Program Sends Reminder to Email Server
3. Email Server Sends Notification(s) to User(s) about Event

#### Primary Flow #5

1. User creates/updates a calendar
2. The information is sent to the database
3. The database calls the email sending function, passing the need information to it
4. The function creates the need headers and body for the email
5. The function sends the email

Primary Flow #4, above, shows how emails are created after a user makes and/or updates a calendar. The email sending function does the bulk of the work as it generates and sends the calendar invite.

#### Primary Flow #6

1. Email is sent to the server
2. Call email parsing function

After the email is parsed, one of two things can happen. If the email has calendar informant in it, that information will be collected and sent to the database where it can be added to each person's calendar. If it does not contain calendar information, the email will be handled as normal. Below are the two alternate flows.

#### Alternate Flow #1, email has calendar information

1. Parsing function collects all the need information
2. Parsing function returns the collected information
3. Information is sent to the database the calendar system uses
4. The information is used to generate new calendars and populate them
5. User receives a notification that they have a new calendar invite

#### Alternate Flow #2, email has no calendar information

1. Parsing function returns false, meaning there is no calendar information in that email
2. The email is handled as a normal email, i.e. it is sent to the intended recipient



## V. Current System

Data Verity currently uses Ext-JS for their webtop. Within the webtop, Data Verity's clients are able to provide their data to Data Verity's database. From there Data Verity provides the clients with reports on the data they provided. The calendar system for this project was integrated into the Ext-JS webtop, so on top of being able to send data and access reports, clients are now able to use the calendar to setup meetings and events. See Figure 14 for an Architecture Design View of Data Verity's System with the integrated calendar and email systems.

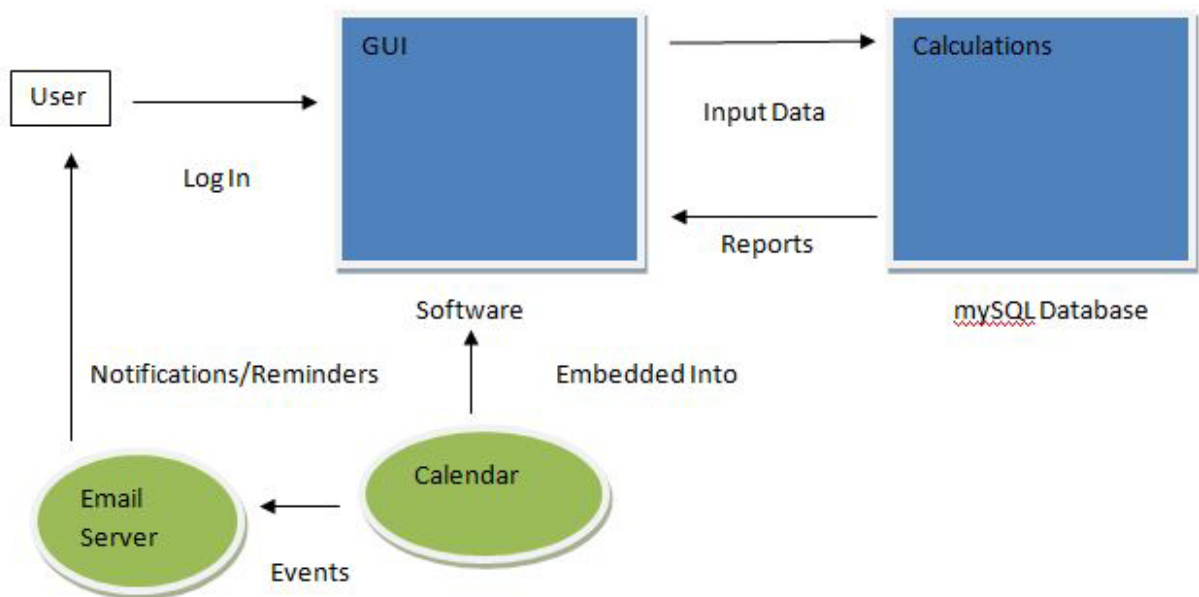


Figure 14: Architecture Design

## VI. System Testing

The calendar system was tested through the creation of a test server provided by Data Verity on Subverted. After code was added or edited, we were able to refresh the website page that currently embeds the calendar into it. After embedding the calendar into Data Verity's current Ext-JS webtop, we were able to continue to test it on a test server containing Data Verity's software. In testing the email server, we sent out test reminders through the calendar to email addresses of the different major providers ensure they were not blacklisted and were being received. In addition, we used Outlook to make sure the calendar system syncs with it and can send calendar information to Outlook.

## VII. Glossary

Customer Relationship Management (CRM) – Is a widely used strategy that involves managing the way firms interact with clients. The technology used in this process provides organizational and synchronization in business practices between sales, the clients, and the firm.

DomainKeys – A cryptographic, signature-based authentication. Email senders' computers generate a public/private key pair, and the public key is published in their DNS records.

Domain Name System (DNS) – A hierarchical naming system for computers, services, or any resource connected to the internet or a private network.

Ext-JS – Is a JavaScript library used to build interactive web applications.

Grid – An Ext-JS class, which is the primary interface of representing data in a tabular format of rows and columns. Similar to Excel.

Reverse DNS Lookup – The determination of a domain name that is associated with a given IP address using the DNS of the internet.

Sender ID – Very similar to SPF but Sender ID verifies either the “Mail From” or the non-visible “From” line.

Sender Policy Framework (SPF) – IP-based protocol that verifies the sender IP address by checking the domain in the email listed in the visible “Mail From” line against a published record the sender has in the DNS.

Simple Mail Transfer Protocol (SMTP) – An internet standard for email transmission across internet protocol (IP) networks.

Tree – An Ext-JS class, which provides a tree-structured UI representation of data. Similar to an operating systems folder hierarchy system.

Web Desktop (Webtop) – A desktop environment embedded into a website or a virtual desktop running in a web browser.

DKIM (DomainKeys Identified Mail) - is a method for email authentication that allows an organization to take responsibility for a message it has sent in a way that can be validated by a recipient. The technique is based on public-key cryptography: Responsibility is claimed by the *signer* by adding a digital signature to a message's header, the *DKIM-Signature* header field. The *verifier* recovers the signer's public key using the DNS, and then verifies that the signed parts have not been altered.

DomainKeys - is an e-mail authentication system designed to verify the DNS domain of an e-mail sender and the message integrity.

MIME (Multipurpose Internet Mail Extensions) – An internet standard that expands the format of email to be able TO support more than just plain text