

CiviCore Final Report

Cody Gonzales, Gary Scheid

Abstract

The visual representation of data in both a meaningful and aesthetically pleasing manner can go a long way in the modern business world. We have created a solution to take simple data and turn it into appealing reports. Our solution takes plainly formatted XML input and transforms it to a well formatted and styled HTML report based on the user's needs. The application gives users options to customize their reports from an intuitive, non-technical interface. It also allows for the insertion of summary data, company logos, headers and footer, and allows for the customization of the report's style.

I. Introduction

Our client for the field session project was CiviCore. CiviCore is a Denver-based company founded in 2000 that provides non-profit clients with information solutions and consulting. CiviCore had been providing their clients with reports on a per request basis. These reports vary greatly from client to client, but are intended to help CiviCore's clients get meaningful information about their companies. CiviCore needed to involve developers each time one of their clients needed a new report. This was not cost efficient and CiviCore wanted to enable their clients to build reports themselves in an intuitive, non-technical environment. Providing CiviCore and their clients with the ability to create reports for themselves would both reduce CiviCore's costs and potentially increase their customers' satisfaction.

Due to the complexity of the project, CiviCore had originally requested that we research and implement existing business intelligence (BI) solutions, rather than create one from scratch. However, after reviewing the possible solutions, CiviCore decided instead to have us design a web-based report designer to be integrated with their current software used for creating reports. A summary of our BI solution research can be found in Appendix A.

The following document details the design requirements and details for the reporting solution that we developed for CiviCore.

II. Requirements

Our client has given us several functional and non-functional requirements that the report designer we implement must meet.

A. Functional Requirements

- Sort report data
- Group report data

- Sum or count items within report groups
- Customize the report appearance through:
 - Customizable report colors and fonts
 - User specified image or text
- Allow users to create and use configuration files to skip the report designer and immediately generate a report.

B. Non-Functional Requirements

- Run on a Java or PHP Linux server.
- The report designer should be a web-based application.
- Method of report output must be HTML

C. Scope

All of the above requirements were to be met in our final solution. If we had extra time, the client would have liked additional output formats to be supported, specifically PDF.

D. Risks

1. Technology Risks

- The only foreseeable technological risk in the scope of this project was the interfacing of our application with the client's software. Based on our design, the input from the client's software must be very specifically formatted to perform correctly.

2. Skills Risks

- Team members needed to learn how to work in a server environment and with tomcat.
- Ultimately the team needed to learn PHP as well.

III. System Design

CiviCore currently uses a query builder to interface with their database and retrieve data to be included in a report. Data from the query is stored in an XML file, formatted as shown in Figure 1. Our software must take this data and provide an interface for a user to create a report from it. Our software must then take the data, parse it, and format it into a report. The formatting will be based on the input by the user to our report designer.

```
<?xml version="1.0"?>
<query>
  <row>
    <col10><![CDATA[Name]]></col10>
    <col2><![CDATA[Gender]]></col2>
    <col3><![CDATA[Ethnicity]]></col3>
    <col6><![CDATA[City]]></col6>
    <col7><![CDATA[State]]></col7>
    <col8><![CDATA[Dollars Donated]]></col8>
  </row>
  <row>
    <col10><![CDATA[Kornelius]]></col10>
    <col2><![CDATA[Male]]></col2>
    <col3><![CDATA[African American]]></col3>
    <col6><![CDATA[Pinson]]></col6>
    <col7><![CDATA[Alabama]]></col7>
    <col8><![CDATA[5.00]]></col8>
  </row>
  <row>
    <col10><![CDATA[Fleta]]></col10>
  </row>
</query>
```

Figure 1: XML input

The overall work flow of our system can be summarized in three components: data parsing, report preference collection and report generation. As shown in Figure 2, the user first uses the query builder to write the desired fields from the database into XML. Our application then parses the XML and provides a series of screens asking the user for specific details regarding the report's format. The report generators then uses the data from the XML file to create the report in the specified format and display it as an HTML page.

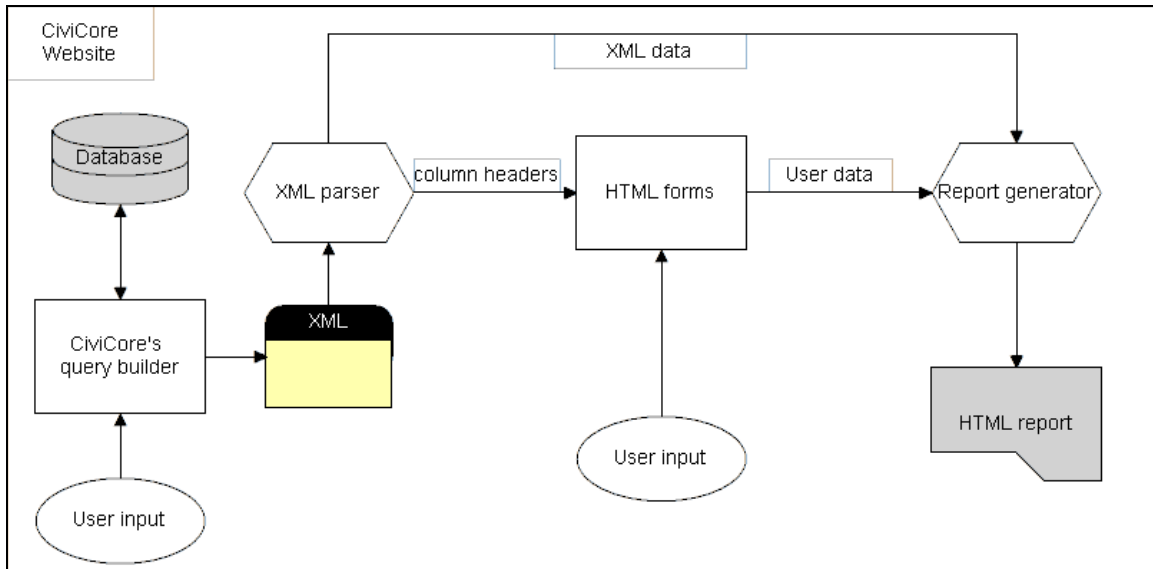


Figure 2: Architecture

A. Data Parsing

The data parsing component accepts the XML input from the query builder and parses it for use in both the report designer and report generator modules. In particular the parser reads the first row of XML input and uses this data for column headings; the columns headings are stored in as part of the user's PHP session data. In addition to retrieving the column headings, the parser strips the input of its XML tags and stores the raw data as a multidimensional array in the PHP session array which is eventually passed to the report generator. The XML parsing is accomplished through the use of PHP's SimpleXML library. In the data collection and report generation components, the number of columns is used to determine column names and formatting.

B. Report Specification

The component consists of five separate PHP-backed HTML pages:

1. The first form gives the user two options; create a new report, or use a saved configuration file to generate a report based on previous report specification. If the user chooses to use a configuration file, all of the report's styles will be as they were in the configuration file; however the data the styles are applied to will come from the new XML input. This will result in a final report page showing after the 'use this configuration file' button is selected. If the user chooses to create a new report, the second form will appear.

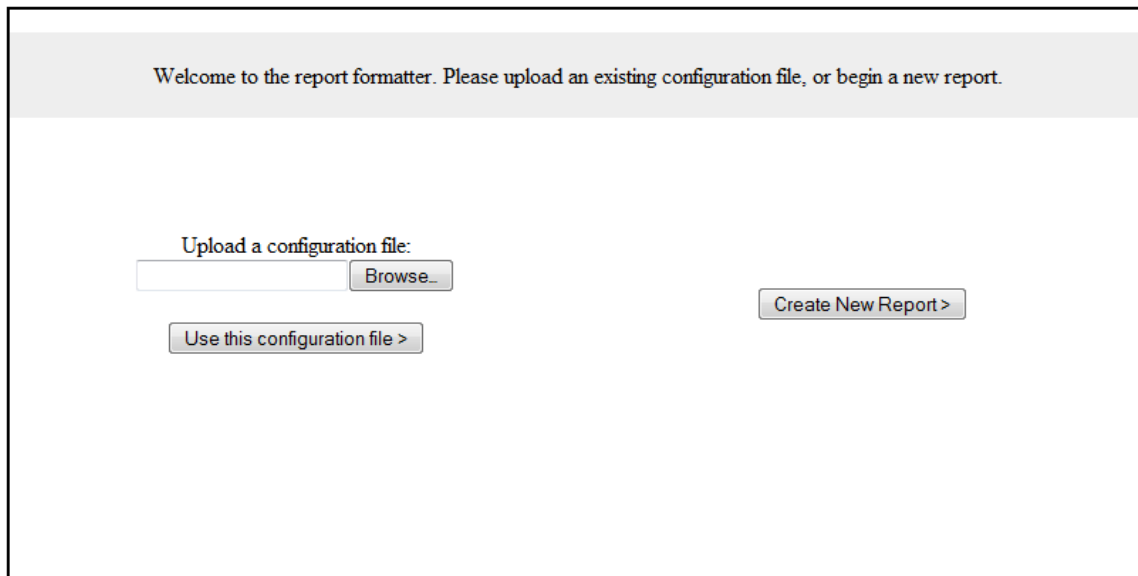


Figure 3: Welcome page

2. The second form requests four pieces of data from the user, none of which are mandatory to proceed (see Figure 4). The options here allow the user to customize the look of the report to be generated. This page contains the data parsing component which is responsible for collecting and storing an XML input passed to the application via the client application.

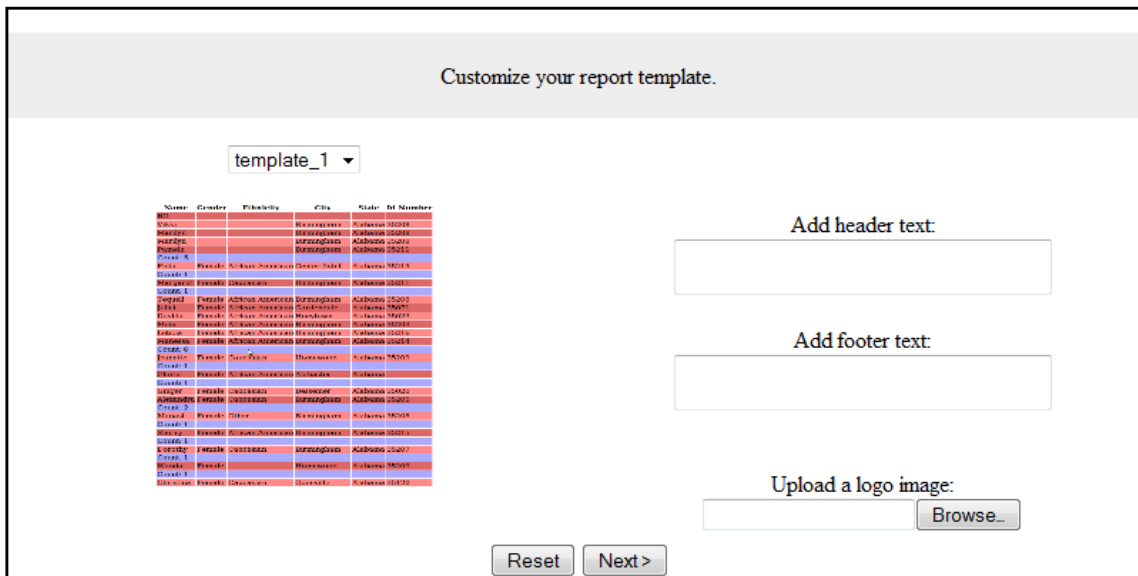


Figure 4: Report style page

The form first requests the user to select a template to style their report with. These templates are responsible for creating difference in the final report's color,

font type, component positioning and other visual aspects. We only created two templates to test with, as the template design is not in the scope of the project. The CSS formatting of the template will be referenced by the report generation script during the report creation component of the application. When selecting a template, the user is shown a preview picture to give the user an idea of the style of the template.

The form also asks the user for both header and footer text, which is added to the top and bottom of the final report. The last piece of information which the form asks for is an image to be uploaded. This image is uploaded via an HTML browse button, allowing the selection of an image file from user's machine. Once the user finishes entering their desired customizations, they can select the 'next' button. At this point all of the entered data will be stored in the PHP session array. This includes the image, which is stored temporarily stored as binary data.

3. The third form asks the user which fields from the input XML they would like to group by.

The form contains two lists of field headers and arrows controlling the movement of fields between boxes. The left most box will be populated with the column heading or the top most element of every column. In order to group by any of the fields the user must highlight one or more of the fields on the left and press the right arrow. Each of the highlighted items is copied into the right list and deleted from the left. Similarly, items on the right can be selected and moved to the left with the left array. We created this dynamic manipulation using Javascript. The grouping is prioritized in top to bottom order from the fields in the right list. Once the user presses the 'next' button the data in the right list is stored in the PHP session array along with the data from the previous form. Figure 5 shows the page for choosing which fields to group by.

Select which fields you would like to group by. Data will be group in the order you add fields by.

Data fields	Group by
Name Gender Ethnicity City State Id_Number	

Next >

< Back

Figure 5: Group by page

4. The fourth form is responsible for collecting summary statistic preferences from the user.

The form contains a populated list of every field header from the XML input; each followed by a drop down box containing possible summary options. The default option selected for each of the fields is 'None'. The user has the option of selecting either 'Sum' or 'Count'. The sum option simply sums the column based upon each of the group by selections and the count counts the total number of the items in each of the group by selections. There is no error handling for summations of non-numeric columns at this point in the process, however in the report generation component non-numeric columns summed will result in an 'error' message where the summation would be.

Once the user presses the 'finish' button, all of the data from the form will be stored in the PHP session array along with the data from the previous forms. If the user chooses to save the configuration file by selecting the check box, the next page will be form five, otherwise the user will be automatically redirected past form five and to their report. Figure 6 shows the page for selecting which summary data to add in the report.

Select the summary statistics you would like to perform on your data

Perform	none	on	Name	based on	Name
Perform	none	on	Name	based on	Name
Perform	none	on	Name	based on	Name
Perform	none	on	Name	based on	Name
Perform	none	on	Name	based on	Name
Perform	none	on	Name	based on	Name

Save your inputs as a configuration file.

Figure 6: Summary page

5. The fifth form is very simple, but much of the behind the scenes work of the report generator happens on this page. The user data is used to instantiate a report generator. This generator then produces XML to recreate its configuration and writes it as a file on the server. In addition, if the user uploaded an image it is written to the server as well. The user has the option of right clicking the link to the configuration file and saving it. This can be used later to generate a report by all of this sessions choices. After the user selects the 'next' button, they will be redirected to their report.

[Right click save as to download your configuration file.](#)

Figure 7: Save configuration page

C. Report generator

The report generation component will manipulate the parsed XML input data based upon the user's specifications and create a formatted HTML page of the results for the user to view. The report generation is divided into three smaller subroutines. The best way to think of the data for the purposes of the following sections is as a simple block of rows and columns.

1. Sort

The first subroutine is responsible for grouping the data according to the user specifications. In order to accomplish multiple levels of grouping the subroutine will use a stable sorting algorithm. A stable sorting algorithm is one which maintains the relative order of data with the same value. The subroutine will stable sort the rows of the report by the columns selected for grouping in reverse order to get the desired grouping affect. Once the first subroutine has finished, the data will be ordered according to the grouping specifications.

2. Summarize

The second subroutine is responsible for applying summary information to the data. Two options are available: summation and count. In both cases, data will be calculated on a single column, the summary column, based on another column, the basis column. These columns can be the same. The basic implementation behind this subroutine is to sum or count the summary column so long as the basis column remains the same.

For example, a user might have a report containing cities listed by county and want to count the cities in a country. The routine then loops through the data and counts the cities so long as the country remains the same. Once the country changes, a row containing the count would then be inserted and the process starts over for the next country.

3. Output


The last subroutine is responsible for writing the HTML output. This subroutine takes the sorted and summarized data and put it into HTML tables, applying styles given by the selected template and adding the header, footer and image if provided by the user. The styling takes the form of CSS files which are dynamically included and correspond to the template selected by the user. The template data will take the form of CSS style sheets and custom HTML code into which the table of summarized data will be input.

At this point, the HTML page has been generated and the user is able to view it.

D. Example Scenarios

Generate a report detailing the total number of donations in each city. Figure 8 shows what this use case might produce.

1. Generate XML via CiviCore's SQL builder.
2. Choose a style template for the report
3. Add header and footer text
4. Add an image to be included in the report heading (e.g., a logo).
5. Select next button.
6. Highlight the column name corresponding to city and press right arrow button.
7. Select next button.
8. Choose the count options, choose donation as the 'on' field, choose the city as the 'based on' field.
9. Select finish button.



Count on donations per city

Name	Gender	Ethnicity	City	State	Dollars Donated


BO					12.50
					Count: 1
			Alabaster		
Eddie	Male	African American	Alabaster	Alabama	8.00
Gloria	Female	African American	Alabaster	Alabama	11.00
					Count: 2
			BIRMINGHAM		
Otis	Male	African American	BIRMINGHAM	Alabama	15.50
					Count: 1
			Bessemer		
Ginger	Female	Caucasian	Bessemer	Alabama	100.00
Thomas	Male	Caucasian	Bessemer	Alabama	15.00
					Count: 2
			Birmingham		
Margaret	Female	Caucasian	Birmingham	Alabama	50.00

Figure 8: Sample Report

Use a saved configuration file to generate a report.

1. Generate XML via CiviCore's SQL builder.
2. Upload desired configuration file.
3. Select use this file button.

Figure 9 shows what the result would be from this use case if the saved configuration file performed a group by on dollars donated along with a sum on dollars donated.



Sum on dollars donated by denominations

Name	Gender	Ethnicity	City	State	Dollars Donated
					0.50
Owen	Male	African American	Birmingham	Alabama	0.50
Sherry	Female	African American	Birmingham	Alabama	0.50
					Total: 1
					1.00
Vikki			Birmingham	Alabama	1.00
					Total: 1
					1.25
James	Male	African American	McCalla	Alabama	1.25
					Total: 1.25
					10.00
Alexandra	Female	Caucasian	Birmingham	Alabama	10.00
Pamela			Birmingham	Alabama	10.00
Christina	Female	Caucasian	Odenville	Alabama	10.00
					Total: 30

Figure 9: Sample Report

IV. Implementation Details and Results

A. Implementation Details

1. Rather than implementing XML reading from scratch, SimpleXML was used to do the base parsing and writing for both the report data and for the configuration file. Since our use of XML was limited and simple, it made sense to use a library rather than to write our own code to handle XML.
2. We used PHP because it was recommended by CiviCore who are in the process of moving much of their infrastructure to PHP. Additionally, it was much easier to learn and use rather than having to learn how to use tomcat for java.
3. User files are not immediately placed on the server, but are instead stored temporarily as binary. This allows for consistent handling of images from both the configuration file and from user upload.
4. User files created for the report are stored in temporary folders on the server. Unfortunately, they are needed until the last page of our current implementation, so they can not be deleted automatically without interfering with the report display. To facilitate the deletion of these files, a simple bash script is included.

B. Results

1. Our product met all of the functional and nonfunctional requirements given to us by CiviCore. However, we did not implement any additional output formats.
2. The following unit and acceptance tests were used to ensure that we met all functional requirements:
 - a. XML parser - Unit Tests
 - Verified that the XML is correctly parsed into a two-dimensional array of values.
 - b. HTML data collection - Acceptance Tests
 - Can the user navigate the site?
 - Do all forms, buttons and drop-downs perform as expected?
 - Verified that user data is stored correctly when moving back and forward through pages.
 - c. Report generator - Unit Tests
 - Is the data sorted in the correct order based on the groupings?
 - Do the sums and counts come to the correct totals?
 - d. Report generator - Acceptance Tests
 - Verified that the correct subset of data used in the summary statistics.
 - Properly formatted HTML page is generated no matter the inputs.
 - e. Integration Testing
 - All three parts of the system must work correctly together. In particular we:

- Verified that forms correctly populate with XML data.
- Verified user information and XML data was correctly passed to the report generator.

V. Conclusion

A. Future Work

Due to time constraints and the scope of creating a fully featured reporting tool not every possible feature has been implemented. Work to extend the application could likely be focused on one of the following areas:

1. Extending Summary Options

Currently summary options are limited to sum and count. Add the ability to sum or count only unique items, or adding the ability to sum or count only if a field meets some condition.

2. Extending Grouping Options

Currently groups are sorted in ascending order. Add the options of sorting in descending order.

3. Output formats

Currently the final report is displayed as an HTML page. Add the option of exporting the report to a PDF format.

4. Extending and Creating Templates

There are two templates provided with the application. Create additional templates and extending the style options beyond their current state.

B. Lessons Learned

Through the work on this application we have learned several valuable lessons, both regarding technical aspects and project management.

1. simpleXML: PHP has a supported set of functions for the manipulation of XML formatted data. simpleXML was very useful for changing the input data into arrays, as well as for storing or data for the configuration functionality.
2. Storing binary data in a file: Base64_encode proved to be very useful in the storage of user data within a document. The function writes the image as binary data, which can later be translated back to an image.
3. Project Management: CiviCore indicated to us that this is how many “real world” projects might go. An initial research stage to identify existing solutions, their viability, and their patterns is an important part of software design might be a part of many development cycles.

4. **Paired Programming:** Paired programming helped tremendously during the course of the project. It helped us provide better formatted code, more robust solutions and it created a more equal workload on the team.
5. **Version Control:** Git helped us work on the project more efficiently. We spend little time informing one another of small changes, and overall had a more uniform, directed project.

VIII. Glossary

Business Intelligence (BI) - refers to computer-based techniques used in spotting, digging-out, and analyzing business data, such as sales revenue by products and/or departments or associated costs and incomes.

CSS (Cascading Style Sheets) - a style sheet language used to describe the presentation semantics (that is, the look and formatting) of a document written in a markup language.

Git - a distributed revision control system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development.

HTML (HyperText Markup Language) - the predominant markup language for web pages.

PDF (Portable Document Format) - is a file format created by Adobe Systems in 1993 for document exchange.

PHP: Hypertext Preprocessor - a widely used, general-purpose scripting language that was originally designed for web development to produce dynamic web pages.

Stable Sorting Algorithm - sorting algorithm which preserves the input order of equal elements in the sorted output.

SQL (Structured Query Language)- a database computer language designed for managing data in relational database management systems (RDBMS).

Tomcat - an open source servlet container developed by the Apache Software Foundation (ASF).

XML (Extensible Markup Language) - a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards..

Definitions from Wikipedia.

Appendix A

I. Introduction to Business Intelligence (BI)

There are a couple of different architecture types that can be implemented with business intelligence software. One possibility is for the platform to run locally and produce reports that can then be distributed. For the purposes of this appendix this will be referred to as desktop-based BI. This architecture is shown in Figure 10. The BI software can actively run on a server and be accessed from a web browser. This architecture is shown in Figure 11. In this architecture, reports are designed, either with a desktop-based BI tool or through some sort of integrated web based designer, and placed online for users to view and possibly edit to their specifications. For the purposes of this appendix, this type of BI will be referred to as server-based BI.

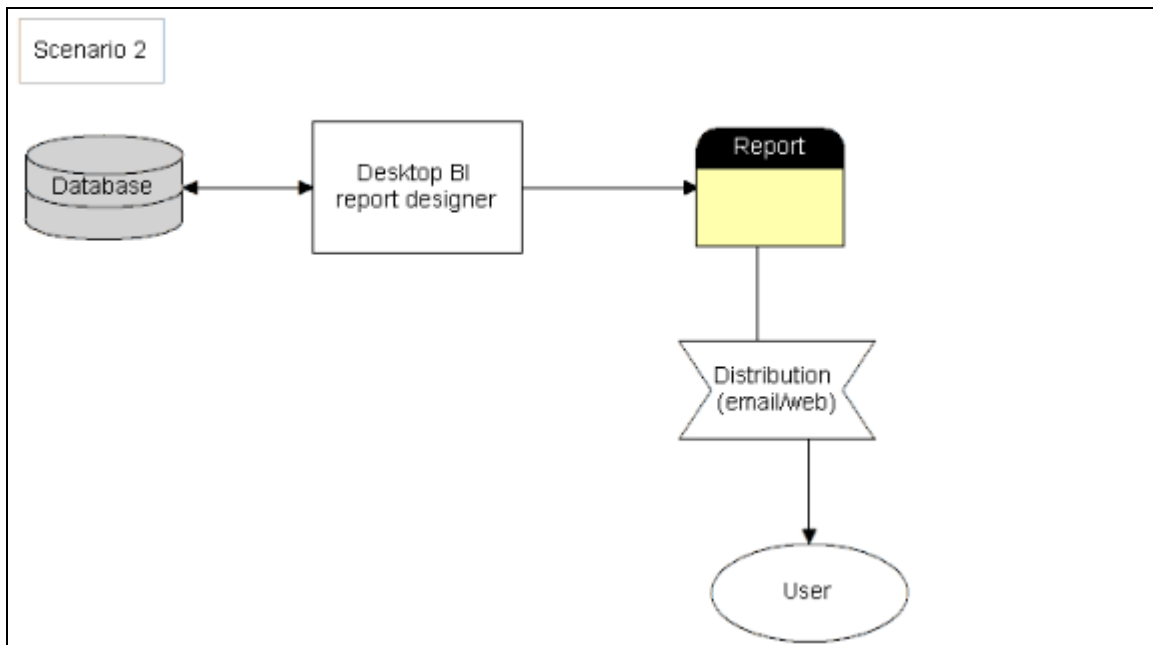


Figure 1: Desktop based BI architecture

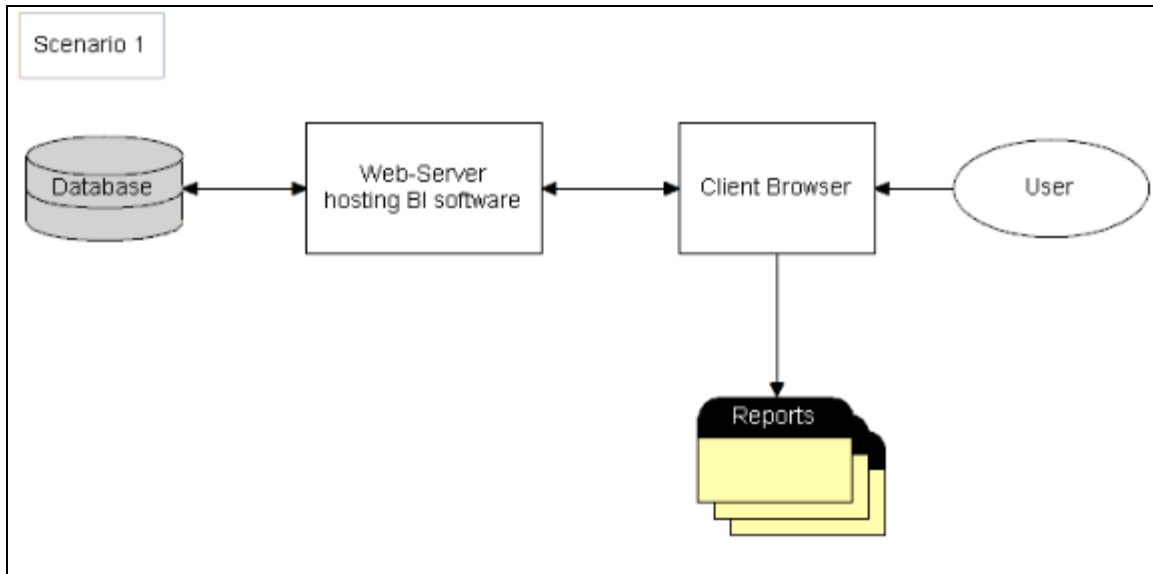


Figure 2: Server based BI architecture

II. Business Intelligence Platform Summary

Based upon the functional and non-functional requirements given to us by CiviCore, we evaluated eight BI platforms. We evaluated these platforms based on cost, report retrieval methods, report designing, security and output formats. Additionally, all of the platforms met the basic requirements given to us by CiviCore. Therefore, all of these platforms are either java or PHP-based, interface with MySQL, and allow for the selection of specific fields, groupings and summarize data.

Our evaluations of these products is as follows:

A. BIRT

BIRT stands for Business Intelligence and Reporting tools. BIRT is free and open source. It is developed by the Eclipse Foundation. A report designer plugin is available for the eclipse IDE providing a desktop-based BI solution, however this tool is not very intuitive for a novice, and would likely necessitate some knowledge of SQL to use properly. BIRT can also be used embedded on a server. Supported output formats include HTML, PDF, WORD, XLS, and PostScript.

B. Jaspersoft

Jaspersoft offers a suite of open source BI solutions written in Java. It has both a free community edition and a \$250.00 enterprise edition. Both editions offer a desktop report designer, iReport, which requires some knowledge SQL. Both editions also offer the ability to view reports online via Jasper Server. Access privileges can be controlled per user. Jaspersoft's enterprise edition offers a variety of output formats for the report including; PDF, HTML, XLS, CSV, RTF, TXT, XML and Flash.

C. Pentaho

Pentaho offers a suite of open source BI solutions written in Java. It has both a free community edition and a for cost enterprise editions. Both editions offer a desktop report designer, Pentaho Reporting, which would require SQL knowledge, like most of the BI software we reviewed. Additionally, Pentaho offers a BI Platform which provides server-based functions. This would allow users to access reports online. Access privileges can be controlled per user.

D. MyDBR

MyDBR is a server based BI solution written in PHP. It is not open source, but rather it is distributed as encoded PHP. This requires the installation of the ionCube decoder in order to use this software. The software is offered as a free community edition, requiring monthly license renewal and having some limitations, or as a premium edition, with a yearly cost of 129 EUR.

The report designer was not very intuitive, and would definitely require SQL knowledge to use it efficiently. Additionally, the functions provided by the software for operations such as summing and counting columns were text based and would need to be learned to effectively use the software. Report retrieval was online and reports could either be viewed as an HTML document or output to an excel file. The system allowed for users and groups to be created and reports to be assigned to be accessible based on user or group privileges.

E. Crystal Reports

Crystal Reports is a desktop-based BI solution. It is sold for cost as a \$400.00 enterprise edition. Crystal Reports offers a desktop report designer which is relatively intuitive compared to the other tested solutions, although it still requires some knowledge of SQL. Access privileges are controlled at the database level. Crystal Reports report formats include; PDF, HTML, XLS, CSV, RTF, TXT, XML and DOC.

F. SpagoBI

SpagoBI is a full BI suite which takes advantage of a variety of open source BI engines, including BIRT and Jasper. SpagoBI is free and open source providing a server-based BI platform. It is entirely web based, providing both report design and retrieval functionality. Security is user based and reports can be output to HTML, PDF, XLS, XML, TXT, CSV, and RTF.

G. LogiReports

LogiReports provides a free web-based report platform. Security is user based. Reports are created and retrieved online. It is provided for both Java and a .NET. Reports can be output to Excel, Word, CSV, HTML and PDF.

H. Elixir

Elixir is a professional BI solution. It provides both desktop and server based BI features. However, it is not open source and comes with a \$10,000 price tag. Reports can be

designed and retrieved both on the web or desktop. Security is user based. Reports can be output to PDF, HTML, PS, XLS, CSV, Glint, IML, image, RTF, TXT, or XML.