A Generic Web-Based Toolkit for Data Collection

A Manuscript
Submitted to
the Department of Computer Science
and the Faculty of the
University of Wisconsin-La Crosse
La Crosse, Wisconsin

by

Mariah Borgschatz
in Partial Fulfillment of the
Requirements for the Degree of
Master of Software Engineering
December, 2013
A Generic Web-Based Toolkit for Data Collection

By Mariah Borgschatz

We recommend acceptance of this manuscript in partial fulfillment of this candidate’s requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kasi Periyasamy
Examination Committee Chairperson

Date

Dr. David Riley
Examination Committee Member

Date

Dr. Joshua Hursey
Examination Committee Member

Date
Abstract


This manuscript describes the design and development of a generic web-based toolkit that generates a survey tool for data collection. The toolkit, a web application, provides the ability to create a customized web-based data-entry application which can then be deployed and run as a fully-functional survey tool. This survey tool can then be used for data collection where the collected data is stored in a customized database. The toolkit provides the ability to include questions of several different types including multiple choice questions and questions that require users to type in answers in text fields. For each question type, appropriate validation criteria can be specified. The specified criteria are then enforced by the survey tool. The toolkit automatically saves while the user is creating their custom data-entry application.
Acknowledgements

I would like to express my sincerest thanks to my project advisor Dr. Kasi Periyasamy for his valuable guidance. I would also like to thank the project sponsor, Mark Nigogosyan, who was ultimately the motivation for this project. Finally, I would like to thank my fiancé and my parents for their patience and encouragement that kept me motivated to finish what I started.
Table of Contents

Abstract .............................................................................................................................. iii
Acknowledgements ........................................................................................................ iv
Table of Contents ............................................................................................................ v
List of Figures ................................................................................................................... vi
List of Tables .................................................................................................................... vii
Glossary ............................................................................................................................ viii
1. Introduction ...................................................................................................................... 1
  1.1 Background .............................................................................................................. 1
  1.2 Motivation ............................................................................................................... 2
2. Requirements and Assumptions .................................................................................... 4
  2.1 Requirements ......................................................................................................... 4
  2.2 Assumptions .......................................................................................................... 5
3. Design .............................................................................................................................. 6
  3.1 Approach ................................................................................................................ 6
  3.2 Database Design .................................................................................................... 6
  3.3 Interface Design .................................................................................................... 9
  3.4 Architecture .......................................................................................................... 16
  3.5 Question Types ...................................................................................................... 19
4. Implementation .............................................................................................................. 23
  4.1 Technology ............................................................................................................ 23
  4.2 Difficulties ............................................................................................................. 24
  4.3 Validation .............................................................................................................. 24
5. Testing ........................................................................................................................... 31
6. Deployment .................................................................................................................... 33
7. Future Work ................................................................................................................... 35
8. Bibliography .................................................................................................................. 37
Appendix .......................................................................................................................... 38
List of Figures

Figure 1: Iterative Development Model ................................................................. 6
Figure 2: ER Diagram for the Toolkit ................................................................. 8
Figure 3: Toolkit Interface ..................................................................................... 10
Figure 4: Select Project (Optional) ................................................................. 12
Figure 5: Rename Project .................................................................................. 13
Figure 6: Page Preview ..................................................................................... 14
Figure 7: Navigation Tree ............................................................................... 15
Figure 8: Generated Survey Tool Interface ................................................. 16
Figure 9: Class Diagram of the Toolkit ......................................................... 18
Figure 10: Question Type – Text ................................................................. 20
Figure 11: Question Type - Decimal Number ............................................. 21
Figure 12: Question Type - Multiple Choice ..................................................... 22
Figure 13: Project name Validation ................................................................. 26
Figure 14: Navigation Tree Errors ....................................................................... 28
Figure 15: Question Setup Errors ....................................................................... 28
Figure 16: Page Preview Errors ....................................................................... 29
Figure 17: Generate Survey Errors ................................................................. 29
Figure 18: Generated Survey Tool Errors ....................................................... 30
Figure 19: Physician Station - Male Patient .................................................... 38
Figure 20: Physician Station - Female Patient ................................................ 39
Figure 21: Question Class Diagram ............................................................... 40
Figure 22: Page Class Diagram ....................................................................... 41
Figure 23: DatabaseAccess Class Diagram .................................................. 41
Figure 24: CodeGenerator Class Diagram ...................................................... 42
Figure 25: Servlet Class Diagram ..................................................................... 42
List of Tables

Table 1: Test Cases ........................................................................................................ 32
Glossary

AJAX
An acronym for Asynchronous JavaScript and XML that is a group of interrelated web development techniques used on the client-side to create asynchronous web applications.

JavaServer Faces (JSF)
A specification for constructing component-based user interfaces for Java web applications.

JavaServer Pages (JSP)
A technology that uses Java to assist in the creation of dynamic web pages.

JDBC
An acronym for Java database connectivity which is a Java-based API for accessing a database.

jQuery
A cross-browser JavaScript library that aims to simplify client-side scripting for HTML.

Modal Window
A child window (often called a popup or dialog box) that requires user interaction before focus can be returned to the parent window.

Model-View-Controller (MVC)
A software architecture pattern where application domain information is separated from the user interface.
Regular Expression (regex)
A sequence of characters comprising a pattern against which a string can be matched.

Relational Database Management System (RDBMS)
An organized collection of data based on a relational model.

Structured Query Language (SQL)
A programming language designed for managing data in an RDBMS.
1. Introduction

1.1 Background

Global Partners is a non-profit organization associated with Gundersen Health System, a comprehensive health care network whose headquarters are located in La Crosse, Wisconsin [1]. Created in 2008, Global Partners’ goal is to develop long-term, sustainable relationships and community-to-community partnerships that expand beyond Gundersen Health System’s typical borders [2]. One such partner community is the Matagalpa district of Nicaragua. Nicaragua is the second poorest country in the Western Hemisphere. Its mountainous terrain and rural disposition make it difficult for its citizens to access necessary services and opportunities. Global Partners-Nicaragua aims to improve the overall health of the people and their communities by helping communities secure adequate nutrition, a clean water supply, and safe housing.

In 2011, Global Partners-Nicaragua collected data on the health needs and strengths of a community named Santa Celia [2]. Information pertaining to current living conditions was gathered and a baseline community health assessment was performed to quantify health-related issues. This information was used to prioritize volunteer efforts and will be compared with future assessments as a way to measure the effectiveness of Global Partners’ efforts in Nicaragua.

The health assessment was done with paper and pencil due to the lack of internet availability in rural Nicaragua. The answers to the assessment questions were then manually input into a database. With over 200 villagers assessed, data input took a substantial amount of time. Manually entering data into a database also introduces a risk of human error. Moreover, the quality of data was not as good as it could have been for several reasons. One such reason was that several volunteers filled out various forms regarding different aspects. Accordingly, there were chances for inconsistencies in data entry due to many human volunteers and multiple forms. Another example of an
inconsistency that was encountered was from dates being entered in various forms. Since Global Partners’ volunteers were mostly from the United States, they expected dates to be entered in the American format: MM/DD/YYYY. However, some volunteers from countries other than the United States entered the date in the format DD/MM/YYYY. The person manually entering the hand-written data into the database had to make an educated guess as to which format was used for each form. Furthermore, some questions had unreadable or incomplete answers while other questions were not answered at all.

1.2 Motivation

In 2012, Dr. Mark Nigogosyan, a member of Global Partners, reached out to the Computer Science department at the University of Wisconsin-La Crosse for assistance in developing an iPad application for the data-collection portion of the health assessment. Dr. Kasi Periyasamy, the director of the Master of Software Engineering program, initiated an independent study with the goal of developing an application that would meet Global Partners’ needs.

As a result, a web-based data-entry application was developed that was run locally on a MacBook Air. The laptop was set up as a server that other laptops, iPhones, iPods, and iPads were able to connect to. Volunteers then used a connected device’s web browser to access the application. Submitted data was stored in a database on the MacBook Air that was hosting the application. The application had validation that ensured all questions were answered with data that met relevant criteria for those questions as specified by Dr. Nigogosyan. In addition, dependencies among the questions were also incorporated; for example, some questions were shown and some were hidden according to previously answered questions. These question dependencies saved the assessor time in not asking questions that do not make sense (e.g. asking a man, a child, or a 90 year old woman if they are currently pregnant). The appendix contains several screenshots of the application that are relevant to this document.
The survey tool worked wonderfully on Global Partners’ next trip to Nicaragua. Not only did they save a lot of time, but their data quality was also improved. However, several changes to the questions were requested which required a substantial amount of work after the independent study had been completed. Realizing that similar requests would be likely after each visit to Nicaragua, the developer was motivated to create a generic web-based toolkit for data collection. With such a generic toolkit, Global Partners, as well as other organizations, would be able to create their own data-entry survey tool.
2. Requirements and Assumptions

2.1 Requirements

Almost all the requirements for the generic toolkit were derived from the initial prototype that Global Partners used. However, the developer paid considerable attention to ensure that the toolkit can be used by any other group as well. As a result, the following requirements were established for the project:

- The toolkit must have predefined questions types including plain text, multiple choice, and numeric type questions.
- Each question type must have appropriate validation criteria that can be defined via the toolkit while creating the survey tool.
- The toolkit must generate a fully functional data collection tool (referred to as a survey tool) that can be deployed as a web-based application.
- The toolkit must generate a new database, with a table to store the collected data, which will be utilized by the generated survey tool.
- The generated survey tool must validate the data entered against validation constraints defined by the toolkit user.
- Generation of the survey tool is considered to be an independent project within the toolkit. Accordingly, the toolkit must be able to save a user’s projects and reload them later so that the user can continue working where they left off. For the ease of reading, this document uses the word “project” that refers to the development of one instance of a survey tool.
- Both the toolkit and the generated survey tool must be easy for anyone with basic computer skills to use.
2.2 Assumptions

The toolkit development was designed with the following assumptions:

- The toolkit and generated survey tool will rely heavily on JavaScript. Therefore, it is assumed that the user will have JavaScript enabled in their web browsers.
- A project may include several pages of questions. Every project must have at least one page and each page must have at least one question.
- Validation will ensure that the survey tool is only generated when all questions have been fully set up.
- Validation will ensure that the collected data will not be saved to the database until all questions meet the specified answer criteria.
- Projects will be saved automatically, at any time, regardless of validation errors.
3. Design

3.1 Approach

There are numerous development models in existence, each with its own advantages and disadvantages. Due to the complex nature of this project, the iterative development model was chosen. The iterative (incremental) development model involves designing, developing, and testing repeatedly, in a cyclical manner, until the software is fully functional and meets all requirements. Each iteration begins as soon as the previous one is completed. The diagram in Figure 1 illustrates the iterative development model used for this project.

![Figure 1: Iterative Development Model](image)

3.2 Database Design

MySQL was chosen for the relational database management system (RDBMS) for both the toolkit and the generated survey tool mainly because it is a free and open source
The entity relationship (ER) diagram for the toolkit is shown in Figure 2. The database design for the toolkit consists of three tables: survey_project, page, and question. There is a one-to-many relationship between the survey_project and page tables and between the page and question tables. Each table’s primary key is an auto-incrementing integer. As shown in Figure 2, the primary key of survey_project, iProjectID, is a foreign key in page. Similarly, iPageID, the primary key of page, is a foreign key in question. These constraints give the tables meaningful relationships that satisfy the requirements and assumptions specified previously.
The database design for the generated survey tool is very simple; there is a sole table which is named the same as the project title. The columns of the table correspond to the questions in the survey tool. When the user is designing their application with the toolkit, one of the required fields for each question is a unique question ID. This question ID is then used as the name of the column in the resulting table. Requiring the toolkit user to name the columns of the generated table was a design decision made to improve the readability of the generated table. For instance, if column names were generic, such as ‘question1’, ‘question2’, ‘question3’, and so on, it would be difficult for someone to look at the data and interpret it. However, if the user specified meaningful names, such as ‘age’, ‘weight’, ‘bmi’, etc., then the table is much easier to interpret and analyze.

When a user submits their answers with the generated survey tool, their answers are stored as one record in the generated table. The record is not inserted until the user answers every question in a manner that satisfies the validation criteria specified with the
toolkit. An auto-incrementing primary key of type integer, called iEntryID, is present to simplify the querying of records for whoever wishes to analyze it.

### 3.3 Interface Design

The interface for the toolkit was designed to be powerful, yet easy to use. The graphical user interface (GUI) is contained on a single web page. This seemed to be the most natural approach for the user to design their custom survey tool. It is comprised of a neutral color scheme and an easy-to-read font. As shown in

![Test Project 1](image)

Figure 3, there are three distinct sections of the page, each with a specific set of actions for assisting the user with different functions.
The header at the top of the page displays the user-defined project title on the left. The button, titled “Generate Survey”, is used to initiate the generation of the custom survey tool as specified by the user’s selections on the rest of the page. To its left is a button dropdown menu that contains three actions: Switch Project, Rename Project, and Copy Project.
Selecting “Switch Project” launches the modal window displayed in Figure 4. It allows the user to switch to an existing project or create a new project (and switch to it). When the user first launches the application, they will be required to select an existing project or provide a name for a new project (the “x” button in the upper right hand corner and the “Cancel” button are disabled when the user does not have a project loaded in session). Similarly, if there are no existing projects, the “Existing Project” radio button and its drop down list will be disabled.
Selecting “Rename Project” or “Copy Project” displays the modal window shown in Figure 4. The current project name will be pre-populated in the textbox for the user’s convenience. Renaming the project simply updates the vchTitle field in the survey_project table. On the other hand, copying the project creates a complete copy.
(functionality similar to that of “Save As” in other standard software applications) of the project but does not switch the user’s working project to the newly created project. If the user wishes to work on the new copy, they will have to click “Switch Project” and select the name of the new copy (as they specified in the modal window).

![Figure 5: Rename Project](image)

The panel on the left-hand side of the page contains a navigation tree. Clicking on a question node of the tree will display the configuration options for that question in the work area of the screen (the bigger panel to the right). Any options set previously will be loaded into the form. When a page node is selected from the navigation tree, a functional preview of that page of the survey, complete with validation and error messages, will be displayed, as seen in Figure 6.
Figure 6: Page Preview

The currently selected node is indicated by white text in a gray rectangle (see Figure 6 or Figure 7) around the page/question name. As shown in Figure 7, a page can be collapsed and expanded by clicking on the plus/minus icon to its left. This would be beneficial for a user that is designing a large project with many pages and/or questions.
The left side panel also contains a dropdown menu that contains three actions: “Add Page”, “Add Question”, and “Delete Selected”. Each button performs the action implied. When the menu item “Add Page” is selected, a new page is added to the project. Since every page is required to have at least one question, a question is automatically added to the new page. Similarly, when the menu item “Add Question” is selected, a new question is added to the selected page. If a question is currently selected, the new question will be added to the page in which the selected question is on. “Delete Selected” will delete the selected page or question. If a page is selected, all questions on the page will be deleted in addition to the page. If a question is the sole question on a page, the entire page (including the question) will be deleted. However, when there is only one page remaining, it cannot be deleted.

At the request of the project sponsors, the interface of the generated survey tool is designed to be very simple. An example can be seen in Figure 8. The project title is displayed on top, the questions are displayed in the middle (main) section, and a “Next” or “Submit” button is at the bottom of the page. Clicking “Next” takes the user to the next page of the survey. The current implementation does not contain a back button for the user to change the answers they have provided for their survey. While this is a bit of an
inconvenience, it is not unusual for a survey and is something that could be done in the future.

When the user is on the final page, the button will say “Submit” instead of “Next”. When the user clicks submit, their answers will be inserted into the database for the application and they will be redirected to the first page of the survey so that it can be taken again, if desired. A popup message will be displayed notifying the user that their answers were saved successfully so that the redirection does not cause any confusion.

3.4 Architecture

This section describes the architecture of the toolkit. A simple object-oriented model-view-controller (MVC) architecture was used for both the toolkit and the generated
survey tool. The class diagram for the toolkit is shown in

Figure 9; its model consists of four classes: Page, Question, DatabaseAccess, and CodeGenerator. A detailed view of each class can be seen in the appendix. The view is a single web page, Index.jsp, and the controller consists of a sole class, Servlet. The Page and Question classes are structural representations of their database table counterparts, page and question. Meanwhile, DatabaseAccess and CodeGenerator are helper classes for the servlet.
Figure 9: Class Diagram of the Toolkit

The purpose of the DatabaseAccess class is to separate the database layer from other layers of the application. The separation of code into layers eases later adjustments and maintenance. All database interactions are encapsulated in the DatabaseAccess class. Numerous stored procedures were created to facilitate database interactions. While stored procedures offer many benefits, not everything could be done via stored procedures. For instance, writing a stored procedure to create a table like the one described for the generated survey tool was simply not feasible because of how dynamic the table must be. The table name, the column names, and the column types are all dynamic. Trying to write a stored procedure for creating such a table would be exponentially more difficult than to simply generate and call a regular create statement.

The CodeGenerator class is used to generate (build) the code used in the survey tool that the toolkit creates. Additionally, for format considerations, the class maintains a record of how many spaces should prefix a line in order to generate readable code. If every line of code was smashed together with no line breaks and no indentation, it would
be incredibly challenging for a human to read. By properly formatting the code, it is easier to understand and troubleshoot.

The architecture for the generated survey tool is somewhat similar to that of the toolkit. The main difference is that there are no models in the MVC architecture. The number of web pages that make up the view is dynamic; there are as many as there are pages that were designed while generating the survey tool. The controller is just a single class named Servlet. There is no separate database layer (like DatabaseAccess in the toolkit) because there is a sole call to the database – inserting a single record when the user submits the survey.

### 3.5 Question Types

As shown in Figure 3, the toolkit user can indicate whether or not each question requires an answer from the user of the survey tool. Additionally, the current implementation of the toolkit includes four question types: text, decimal number, whole number, and multiple choice. Each type contains validation criteria that can be customized to meet the needs of the toolkit user.

For a question of type text, the user must specify a minimum and maximum length of the answer. The maximum will be used to set the length of the associated column in the generated database for the survey tool. If the user indicates that the question is required, they cannot set a minimum length of zero or they will receive a validation error. If the user specifies a maximum length greater than 100, the field will be rendered as a text area in the generated survey tool. Otherwise, it will be a regular textbox. Additionally, the user can opt to validate the text by indicating which characters should be allowed. This can be seen in Figure 10. A few character types, uppercase letters, lowercase letters, and digits, are predefined. Furthermore, the special characters type allows the user to specify any character they wish to allow (including characters from the predefined character types). For instance, if the user only wanted to allow hexadecimal characters, they could
disallow uppercase letters, disallow lowercase letters, allow digits, and allow the following in the special characters box: ABCDEF.

Figure 10: Question Type – Text

Validation criteria for whole number and decimal number type questions are very similar. The criteria for decimal number type questions are shown in Figure 11. The criteria for whole number type questions are the same except that there is no “Number of Decimal Places” criterion. For the “Validation” setting, the user can opt to have no validation, specify a minimum, specify a maximum, or specify both a minimum and a maximum.
The other question type is multiple choice; the associated criteria can be seen in Figure 12. If the toolkit user indicates that the user may select one answer, then they can choose either radio buttons or a drop down list for the display type. On the other hand, if they choose 2 or more answers, the display type can only be checkboxes. Then, the user can add the answer choices they want presented to the survey tool user. Using the arrows, they can alter the order the choices will be shown. Alternatively, they can remove a choice by selecting the red x button. If the “Add “Other” choice (with textbox)” option is selected, the user can specify a choice that, when selected, will allow the survey tool user to type an alternative answer. The same validation criteria for the text question type are available to the user for the “Other” choice textbox.
Figure 12: Question Type - Multiple Choice
4. Implementation

4.1 Technology

For both the toolkit and the generated survey tool, Java was chosen as the primary programming language due to the developer’s familiarity with it. The JavaServer Faces (JSF) 2 framework was selected based on a recommendation. While version 1 of JSF used JavaServer Pages (JSP) as its template system, JSF 2 uses Facelets. This proved to be an insurmountable obstacle for the developer. After several months of struggling with JSF, the developer started the project over with JSP. These struggles are described in a later section titled ‘Difficulties’.

AJAX was used heavily throughout the toolkit. Because a single web page is used for the entire application, every post back to the server is done with AJAX via jQuery, which, along with JavaScript, is also used profusely. The overall look and feel comes from Twitter Bootstrap (commonly referred to as simply Bootstrap) JavaScript and CSS. Bootstrap is a front-end toolkit that was created to enable rapid web development that has a consistent and easy-to-use appearance.

The generated survey tool uses a static JavaScript/jQuery file that contains validation functions. Another JavaScript/jQuery file – that is dynamic – is generated by the toolkit and contains the AJAX necessary for communicating with the server. These server requests are used for navigating to the next page of the survey and for inserting the record into the database when the survey is submitted.

MySQL Workbench was used to facilitate database implementation. It provided an easy-to-use interface for designing the database for the toolkit. It also facilitated viewing table data and creating/updating stored procedures. The NetBeans IDE was chosen for development because of the developer’s previous experience with it. The JDBC API was used to integrate the Java code with MySQL.
4.2 Difficulties

Initially, the plan was to use the JSF framework to develop the toolkit even though the developer’s only previous web development experience was with JSP. While JSF 1 and JSP are fairly similar, JSF 2, which was selected for this project based on a recommendation, is very different than JSP. Other difficulties encountered over the course of this project are explained in the Deployment section of this manuscript.

JSF offers some very convenient features. One such feature is component libraries like PrimeFaces which offers the user customizable components that come with pleasant looking CSS styles. The navigation tree in the panel on the left side of the application could be easily created with PrimeFaces. Without such a component library, the navigation tree had to be coded by hand using jQuery and CSS to make it functional. Accordingly, PrimeFaces has the potential to save the developer a significant amount of time.

The JSF framework might be a plausible choice for a project such as this one if the developer has mastered it, but a complex project where there is a single web page that constitutes an entire web application does not seem to be what the framework was designed for. Consequently, the developer was unable to understand the JSF framework quickly enough to continue using it for this project. Instead of continuing to struggle for every tidbit of progress, the decision was made to scrap the entire project and start over with JSP. Within a few short weeks of switching to JSP, the developer had already surpassed the amount of progress made over the course of several months of working with JSF. Thus, the risk of switching frameworks after a few months of work paid off.

4.3 Validation

A good interface does not allow the user to do nonsensical things. For example, using a drop down list of US states prevents the user from entering something invalid, such as
“ZZ”. Both the toolkit and the generated survey tool try to adhere to this guideline. For instance, the ‘maxlength’ attribute of text fields prevents the user from entering more characters than specified. All text fields, on both the toolkit and the generated survey tool, have the ‘maxlength’ property set. In both applications, any validation errors encountered are displayed as red text to noticeably indicate the error. It is clearly visible and on the screen and descriptive enough so that the user will be able to identify the problem and take corrective actions to fix the error.

Another example of preventing users from entering nonsensical data is that any text field that expects only digits suppresses non-digit key presses (note: other vital key presses such as tab, shift, control, enter, and the arrow keys are not suppressed) so that the user cannot enter meaningless data. (If the field is for a decimal number, periods are allowed as well.) Because the user is still allowed to paste text into the field (by pressing Ctrl+V or right clicking and selecting paste), the contents of the field must still be validated to ensure the actual value meets the criteria specified.

When the user is creating a new project, renaming a project, or copying a project, the modal window validates that a new, unique, project name has been specified. If it has not, an error message will be shown like in Figure 13. It is important to have meaningful error messages. If the error was the result of the name not being unique, having a message that says “Invalid project name” does not explain to the user that they cannot choose that name because it already exists.
One of the developer’s favorite features of the toolkit is the fact that it automatically saves every time the user switches from one page/question to another. This will prevent the user from accidentally losing their work. When the project is saved, it is also validated. The navigation tree in the panel on the left side will display a red warning symbol next to any question that has a validation error. If a page has questions with errors, the page will also have the symbol next to it. This can be seen in Figure 14. If the
selected question has one or more errors, they will be displayed at the top (see Figure 15). If a page with errors (on one or more of its questions) is selected, the errors for all of the pages will be displayed (see Figure 16).
Figure 14: Navigation Tree Errors

Figure 15: Question Setup Errors
If the user clicks “Generate Survey”, their design will first be validated (and automatically saved). If there are any errors, they will see a message on a popup window explaining that all errors must be resolved before the project can be generated. This can be seen in Figure 17.

Each question on the generated survey tool has its own validation criteria based on the specification designed via the toolkit. When the user clicks the “Next” or “Submit” button, all of the questions on the current page are validated. Any errors result in a message being shown next to the question with the error as shown in Figure 18.
Figure 18: Generated Survey Tool Errors
5. Testing

Because of the iterative development approach selected for this project, testing was done for each chunk of code that was written, as it was developed. When everything was implemented, the developer then tested the entire application as a whole. There were two areas of testing that were done: testing the interface of the toolkit and testing the interface of the generated survey tool. However, the same ideology was used to test both portions.

To test the interface of the toolkit, basic test cases were developed for each iteration in order to ensure all possible scenarios of the newly implemented functionality worked. For example, when the “Delete Selected” option was implemented, the test cases in Table 1 were evaluated. Additionally, some basic regression testing was performed to ensure that the addition did not break something that was implemented in a previous iteration.

After everything had been implemented and tested iteratively, the full functionality was tested. Several volunteers, as well as the developer, did full run-throughs with the toolkit and the subsequently generated survey tool. Having multiple people test an application is always a good idea since everyone thinks uniquely. Because the developer knows how the code is written, they may not think to test a specific scenario. Meanwhile, the other testers will have no preconceptions and will simply try to use the application. This testing is also a good indication of the usability of the software; if the testers are confused or become frustrated with the interface, then the application could use improvement.
<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Navigation Tree</th>
<th>Selected Node</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Question 3&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>Page 1 - Question 1</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
</tr>
<tr>
<td>2</td>
<td>Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>Page 2</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
</tr>
<tr>
<td>3</td>
<td>Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>Page 2 - Question 1</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
<td>(prompt to confirm)&lt;br&gt;Page 1&lt;br&gt;Question 1&lt;br&gt;Question 2&lt;br&gt;Page 2&lt;br&gt;Question 1&lt;br&gt;Page 3&lt;br&gt;Question 1</td>
</tr>
<tr>
<td>4</td>
<td>Page 1&lt;br&gt;Question 1</td>
<td>Page 1 - Question 1</td>
<td>Error message</td>
<td>Error message</td>
</tr>
<tr>
<td>5</td>
<td>Page 1&lt;br&gt;Question 1</td>
<td>Page 1</td>
<td>Error message</td>
<td>Error message</td>
</tr>
</tbody>
</table>

Table 1: Test Cases
6. Deployment

The deployment aspect of this project was perhaps the most complex. Subsequently, the issue was never fully resolved. In order for someone, besides the developer, to use the toolkit, it must be deployed somewhere other than the machine where it was developed and tested. Regardless of whether it’s deployed on a web server or installed on a computer, a bit of work must be done to get everything set up and functioning. The toolkit requires an existing database with tables and stored procedures already present. To become a useable application, some kind of installation package would need to be developed that would automatically, and properly, set up everything required for the toolkit to function.

It is fairly straightforward to output working code for the generated survey tool. The challenge, however, lies in compiling that code and producing a war file. When the toolkit generates the code, a pre-existing template (or skeleton) NetBeans project is first copied and then named by the application being generated. This template project contains CSS files, jQuery and Bootstrap libraries, and a static JavaScript/jQuery file full of functions that will be called by the generated code. The servlet is also static and already present in the template project. A custom JavaScript file is created by the toolkit. This file contains the code that validates that the survey fulfills all specified criteria when the user selects “Next” or “Submit”. For every page the user designed with the toolkit, a JSP page is created with the HTML elements necessary for the new application. Lastly, a new database and table are created. Their names match the name of the project that was generated.

If the project has already been generated, the user will be asked if they would like to overwrite the existing (generated) project. If they opt against that, then nothing happens (they could rename the project to resolve the conflict without overwriting if they so desire). If they grant permission to overwrite, then both the generated database and generated project will be deleted and regenerated according to the current configuration specified with the toolkit.
After the files are created, the code must be compiled. This is accomplished via the javax.tools API. Unfortunately, the developer was unable to find a reasonable way to create a war file. However, if the user has NetBeans, there is an incredibly easy way to obtain the war file from the generated project. The user can simply select “Clean and Build” on the generated project and NetBeans will produce the war file. While this is not an ideal way to obtain the war file since it requires the user to perform extra actions, it does get the job done easily.
7. Future Work

The survey generator toolkit could be more versatile and useful to more people with some additional features. One of the limitations is the inability to choose which RDBMS the user prefers. Currently, both the toolkit and the generated survey tool use MySQL. A person or organization wishing to utilize this toolkit may use a different database server and being required to use MySQL could be very frustrating. Allowing the user to specify the database technology the toolkit and generated survey tool use would significantly increase the usability.

Another significant limitation is the types of questions that are currently supported. Most notably, there is no way to specify a format that an answer must abide by. Allowing the user to specify a regular expression for an answer in a textbox to be validated against would be a substantial improvement to the system. However, the toolkit is designed to be easy to use and many people do not understand how to write a regular expression. Accordingly, frequently used formats such as phone numbers and dates should be available as question types. Having multiple phone number and date formats available could prove to be very beneficial to the user. Furthermore, a user should be able to specify restrictions for the date (i.e. requiring the date to be in the future, be in the past, or be in some range).

A third limitation of the current work is the inability to specify question dependencies. For instance, some questions may be irrelevant based on answers to previous questions. To illustrate this, imagine that the survey taker indicates that they are male, a child, or a 90 year old woman. They should not be required to answer the question “Are you currently pregnant?” as it is impossible. Thus, the ability to dictate question dependencies in the toolkit would be a tremendous addition that could ultimately save the survey taker time as well as frustration from having to answer irrelevant questions.

As users submit their answers to the questions via the generated survey tool, the answers are stored in a database. The toolkit user that created the questions will
undoubtedly wish to view the answers they have received. Because not everyone will be comfortable with using the database table, a helpful addition would be for the generated survey tool to allow the user to view submitted surveys. For multiple choice and numerical type questions, statistics on the answers could be displayed for the user to analyze. Some users may even wish to be able to edit or change the answers that were previously entered for a survey.

Another functionality that would be useful to many users of the toolkit is the ability to reorder the pages and/or questions they have designed, move a question to a different page, or even move or clone (copy) a question or page from one project to another. Adding this functionality could potentially save the user a significant amount of time.
8. Bibliography


Appendix

Figure 19: Physician Station - Male Patient
Figure 20: Physician Station - Female Patient
<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name : String</td>
</tr>
<tr>
<td>- ID : int</td>
</tr>
<tr>
<td>- questionIndex : int</td>
</tr>
<tr>
<td>- questionID : String</td>
</tr>
<tr>
<td>- questionText : String</td>
</tr>
<tr>
<td>- isRequired: boolean</td>
</tr>
<tr>
<td>- questionType : String</td>
</tr>
<tr>
<td>- min : String</td>
</tr>
<tr>
<td>- max : String</td>
</tr>
<tr>
<td>- validateText : boolean</td>
</tr>
<tr>
<td>- allowTypes : String</td>
</tr>
<tr>
<td>- validSpecialCharacters : String</td>
</tr>
<tr>
<td>- decimalPlaces : String</td>
</tr>
<tr>
<td>- validationType : String</td>
</tr>
<tr>
<td>- answerChoices : String</td>
</tr>
<tr>
<td>- otherChoice : String</td>
</tr>
<tr>
<td>- displayType : String</td>
</tr>
<tr>
<td>- numberOfAnswers : String</td>
</tr>
</tbody>
</table>

```java
+ Question(id : int, name : String, questionIndex : int)
+ clearAll()
+ longestAnswerChoiceLength() : int
+ accessor methods for all private methods
```

Figure 21: Question Class Diagram
### Page

- name : String
- questions : ArrayList<Question>
- pageIndex : int
- ID : int

+ Page(id : int, name : String, pageIndex : int)
+ addQuestion(id : int, name : String, questionIndex : int)
+ getQuestionDBByID(index : int)
+ getQuestion(name : String) : Question
+ accessor methods for all private attributes

Figure 22: Page Class Diagram

### DatabaseAccess

- _con : Connection
- _st : Statement
- _cStmt : CallableStatement
- _rs : ResultSet

- _Initialize()
+ _InsertSurveyApplication(title : String) : int
+ _InsertPage(iApplicationID : int, name : String, iIndex : int) : int
+ _InsertQuestion(iPageID : int, name : String, iIndex : int) : int
+ _CreateTable(sName : String, columns : HashMap<String, String>)
+ _ListSurveyApplications() : ResultSet
+ _ListPages(iApplicationID : int) : ResultSet
+ _ListQuestions(iPageID : int) : ResultSet
+ _UpdateQuestion(q : Question)
+ _renameProject(iProjectID : int, newName : String)
+ _copyProject(iProjectID : int, newName : String) : int
+ _DeletePage(iPageID : int)
+ _DeleteQuestion(iQuestionID : int)

Figure 23: DatabaseAccess Class Diagram
### CodeGenerator Class Diagram

```java
public class CodeGenerator {
    + _spaces : String
    + spaceCount : int
    + code : String
    + DIR : enum(F, S, B, FB)

    + getPageHeader(pageCount : int, surveyTitle : String)
    + getMultipleChoiceCode(answers : ArrayList<String>, q : Question)
    + getRadioButtonListCode(answers : ArrayList<String>, q : Question)
    + getDropDownListCode(answers : ArrayList<String>, q : Question)
    + getCheckBoxListCode(answers : ArrayList<String>, q : Question)
    + getTextBoxCode(id : String, maxlength : int, display : boolean)
    + getWholeNumberCode(id : String)
    + getDecimalNumberCode(id : String, decimalPlaces : int)
    + getMeetsLengthRequirementsCode(id : String, min : String, max : String)
    + getTextValidationCode(q : Question)
    + getDecimalValidationCode(q : Question)
    + getWholeNumberValidationCode(q : Question)
    + getMultipleChoiceValidationCode(q : Question)
    + getNextButtonCode()
    + getSubmitButtonCode()
    + getAJAX(json : CodeGen, last : boolean)
    + getSaveColumnsCode(q : Question)
    + getSQLColumnDeclaration(q : Question, dbColumns : HashMap<String, String>)
    + addLine(direction : DIR, newHTML : String)
}
```

**Figure 24: CodeGenerator Class Diagram**

### Servlet Class Diagram

```java
public class Servlet {
    # processRequest(request : HttpServletRequest, response : HttpServletResponse)
    # doPost(request : HttpServletRequest, response : HttpServletResponse)
    # doGet(request : HttpServletRequest, response : HttpServletResponse)
    + getServletInfo() : String
    - reindex(pages : ArrayList<Page>)
    - getNullSafeString(s : String) : String
    - validateQuestion(q : question) : String
}
```

**Figure 25: Servlet Class Diagram**